Xaldjigitov A.A., Zaynalov N.R.,
Qarshiyev A.B., Yakubdjanova D.K.

DASTURLASHDAN MISOL VA MASALALAR TO'PLAMI

O'quv qo'llanma

O‘ZBEKISTON RESPUBLIKASI
OLIY VA O‘RTA MAXSUS TA‘LIM VAZIRLIGI

MUHAMMAD AL-XORAZMIY NOMIDAGI
TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI
SAMARQAND FILIALI

Xaldjigitov A.A., Zaynalov N.R.,
Qarshiyev A.B., Yakubdjanova D.K.

DASTURLASHDAN MISOL VA MASALALAR TO‘PLAMI

O‘quv qo‘llanma

O‘zbekiston Respublikasi oliy va o‘rta maxsus ta‘lim vazirligi tomonidan o‘quv qo‘llanma sifatida tavsiya etilgan

5330300 – Axborot xavfsizligi
5330500 – Kompyuter injiniringi
5330600 – Dasturiy injiniring
5350100 – Telekommunikasiya texnologiyalari
5350200 – Televizion texnologiyalar
5350300 – Axborot-kommunikasiya texnologiyalari sohasida iqtisodiyot va menejment
5350400 – Axborot-kommunikasiya texnologiyalari sohasida kasb ta’limi

«Mahalla va oila nashriyoti»
Toshkent – 2021

UDK: 004.42:51(075.8)
D 24

Dasturlashdan misol va masalalar to'plami [Matn] : o'quv qo'llanma / A.A. Xaldjigitov, N.R. Zaynalov, A.B. Qarshiyev, D.K. Yakubdjanova .-Toshkent: Mahalla va Oila,2021.-522 b.

UO'K 004.42:51(075.8)

Mazkur o'quv qo'llanma "Dasturlash I" o'quv fani dasturi asosida yozilgan, unda dasturlashga doir har xil murakkablikdagi masalalar yoritib berilgan. Asosiy maqsad qilib har qanday masalani yechimini uning algoritmini yaratishga va uni batafsil tahlil qilishga qaratilgan. Keltirilgan masalalarning **PascalABC** va **C++** dasturlash tillaridagi yechimi ham keltirilgan. Shu bois talabalar ushbu tilni oddiy imkoniyatlarini bilgan holda kelgusida murakkab dasturlar yaratishlari mumkin bo'ladi. 5330300–Axborot xavfsizligi, 5330500–Kompyuter injiniringi, 5330600 –Dasturiy injiniring, 5350100–Telekommunikasiya texnologiyalari, 5350200–Televizion texnologiyalar, 5350300–Axborot-kommunikasiya texnologiyalari sohasida iqtisodiyot va menejment, 5350400–"Axborot-kommunikasiya texnologiyalari sohasida kasb ta'limi" ta'lim yo'nalishlari bo'yicha tahsil olayotgan talabalarga mo'ljallangan.

O'quv qo'llanmadan maxsus ixtisoslashtirilgan maktab va akademik litsey o'quvchilari va o'qituvchilari, oliy o'quv yurtlari o'qituvchilari, informatika fanini, hamda dasturlash texnologiyasini o'rganayotgan talabalar ham foydalanishlari mumkin.

Taqrizchilar:

Samarqand davlat universiteti "Axborotlashtirish texnologiyalari" kafedrasi mudiri, texnika fanlari doktori professor I.I.Jumanov;

TATU Samarqand filiali "Kompyuter tizimlari" kafedrasi mudiri, texnika fanlari nomzodi dotsent Q.A.Bekmurodov.

ISBN 978-9943-7776-3-7

MUNDARIJA

Ushbu kitob kompyuterni o'yinchoq qilib olganlar uchun emas, balkim kompyuterni o'ziga asbob deb bilganlarga mo'ljallangan. Kitob, kompyuterni nimaga qodir ekanligini o'rganish maqsadida, o'z mahoratini oshirish yo'lida zamonaviy dasturlar yaratishga ishtiyoqi bo'lgan iste'dodli yoshlarga qaratilgan. Xuddi shu maqsadga erishish uchun kitobda algoritmlar va ular asosida tuzilgan dasturlar tahlili keltirilgan. Umid qilamizkim, ushbu kitobni qo'liga ushlaganlar kelgusida, nafaqat dasturlash sohasida, balkim o'zi tanlagan istalgan sohada yetuk mutaxassis bo'lib yetishadi.

Mualliflardan

SO'Z BOSHI

Axborot texnologiyalarining keskin rivojlanishi ushbu yo'nalishdagi bizning munosabatimizni doimiy o'zgarishiga sabab bo'lmoqda. Bundan 20 yillar ilgari kompyuter savodxonligi asosan dasturlash bilan bog'liq bo'lgan bo'lsa, keyinchalik shaxsiy kompyuterlarning keng tarqalishi natijasida biz faqat qaysidir tugmalarnigina bosishni o'rganishga o'tib ketdik. Buning natijasida hisoblash markazlari qoshida tashkil topgan dasturlovchilar maktablarining batamom tarqalib ketishiga olib keldi. Vaholanki, hozirgi kunda keng tarqalgan dasturlar, masalan, **Microsoft** kompaniyasi mahsulotlari bevosita dasturlovchilar tomonidan ishlab chiqilganligini yoddan chiqarmasligimiz lozim.

Natijada nafaqat zamonaviy dasturlash texnologiyalari va hatto ularga oid tushunchalar ham chop etilgan milliy o'quv qo'llanmalarda to'liq yoritilganicha yo'q.

Albatta, ommaviy foydalanilayotgan dasturiy mahsulotlar faqatgina bir kishi tomonidan emas, balki dasturlovchilar guruhlari tomonidan yaratilgan. Shunday ekan, biz birinchi navbatda dasturlovchilarni, keyinchalik esa ma'lum loyiha asosida ishlaydigan dasturlovchilar guruhlarini tashkil qilishimiz lozim. Bu esa eng muhim vazifalardan biri bo'lib hisoblanadi.

Hozirgi kunda respublikamizda chop etilayotgan kitoblar asosan kompyuter savodxonligi doirasida bo'lib, ularda dasturlash tillari juda oz miqdorda aks ettirilgan. Bu hol esa, o'z navbatida, talabalarimizni zamonaviy dasturlash texnologiyalaridan ancha uzoqlashtirmoqda.

Ushbu o'quv qo'llanma mavjud bo'shliqni to'ldirishga yo'naltirilgan bo'lib, bo'lajak mashhur dasturlovchilarning paydo bo'lishida turtki bo'lib xizmat qiladi, deb umid qilamiz.

Dasturlashni o'rganish uchun faqatgina dasturlash lozim, uning boshqa yengil yo'li mavjud emas. Lekin bizgacha ishlab chiqilgan algoritmlarni chuqur o'rganib chiqishimiz kerak bo'ladi, shundan so'ng qo'yilgan murakkab masalalarni yechishda ularni qo'llash maqsadga muvofiq bo'ladi. Shu tariqa biz dasturlovchilarning mahoratini oshirimiz mumkin bo'ladi.

Hozirgi kunda axborot texnologiyalari deganimizda, birinchi navbatda, kompyuter bilan ishlash qobiliyati to'g'risidagi tasavvur paydo bo'ladi. Lekin, axborot texnologiyalari bozorini tahlil qiladigan bo'lsak, bu yerda ushbu segmentning asosiy qismini dasturiy ta'minotni ishlab chiqaruvchi kompaniyalar egallab turganini ko'ramiz.

Afsuski, oxirgi yillarda o'quv yurtlarimizda zamonaviy dasturlash tillariga kerakli darajada e'tibor berilmay qolindi, buning asosiy sabablari bo'lib malakali mutaxassislarning yetishmasligi va mazkur yo'nalishda o'quv qo'llanmalarning kamligi hisoblanadi.

Muallif lar bir necha yillar davomida ushbu yo'nalishda to'plagan bilimlarini boshqalarga yetkazishni o'z oldiga maqsad qilib qo'ydilar. Ushbu qo'llanma materiallari mualliflarning bir necha yillik tajribalarini qamrab olgan.

Qo'llanmada keltirilgan dasturlar **PascalABC** va **C++** tillari asosida yaratilgan bo'lib, zaruriyat tug'ilganda foydalanuvchilar ularni boshqa algoritmik tillarga o'girishlari mumkin bo'ladi. Dasturda keltirilgan algoritmlarni ishlash jarayonini chuqur o'rganish uchun ularni talabalar bevosita kompyuterda bajarishlari lozim bo'ladi. Faqatgina tayyor algoritmlarni va ular asosida tuzilgan dasturlarni o'rganish orqali dasturlashni to'liq o'rganib bo'lmaydi. Buning uchun mustaqil dasturlar yaratish kerak, shu bois har bir bobning oxirida topshiriqlar keltirilgan. Keyinchalik Internetda mavjud maxsus veb-sahifalardagi misollarni bajarib, o'sha yerni o'zida tekshirishlarni amalga oshirish orqali malakangizni oshirishni qat'iyan tavsiya etamiz.

Ushbu qo'llanmada barcha algoritmlarni to'liq keltirish qiyin masala albatta, shu bois, mazkur qo'llanma talabalarni keyinchalik mustaqil bilim olishlariga asos bo'ladi, deb umid qilamiz.

O'quv qo'llanmaning qo'l yozmasini kompyuterga kiritishda, undagi mashqlarni tahlil qilishda o'zlarining qimmatli vaqtlarini ayamagan va o'z maslahatlari bilan uni boyitgan kasbdoshlarimizga va

shogirdlarimizga, chunonchi Suvankulov Aminjon, Bekchanov Huseynbay, Abdullaev Sardor, Zaynalov Mirzo, Maxmadiyorov Faxriddin va Hamraqulov Farruxlarga o'z minnatdorchiligimizni bildiramiz. Bulardan tashqari, ushbu qo'llanmani yaratilishiga asos solgan sobiq shogirdlarimizni eslab o'tish o'rinli deb hisoblaymiz, chunonchi Krasnoyartsev Sergey, Aliyev Ernest, Jafarov Konstantin.

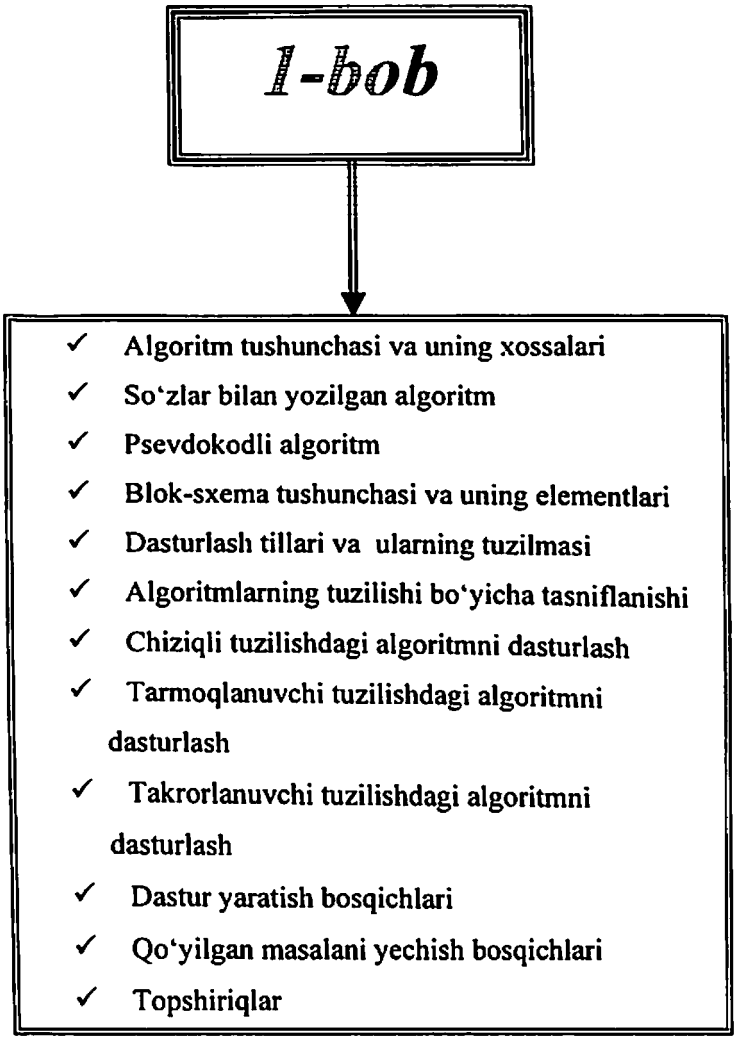
Qo'llanmada keltirilgan algoritmlar va dasturlar balkim masalaning yechimini qidirishdagi optimal yechimi bo'lmasligi ham mumkin, lekin ular kelgusida yaratilishi mumkin bo'lgan algoritmlarga asos bo'lishi mumkin. Bundan tashqari dasturlarda kiritilayotgan qiymatlarni dasturlash tilida mavjud chegaralar doirasida bo'lishi tekshirilmaydi, agar masalaning shartida bu narsa aniq ko'rsatilmagan bo'lsa.

Mualliflar mazkur qo'llanma haqidagi fikrlaringizni, yangi algoritmlaringizni katta e'tibor bilan quyidagi elektron manzil bo'yicha qabul qiladi: algoritmlar@inbox.uz.

1-bob. DASTURLASHNING ALGORITMIK ASOSLARI

Informatika fanining, xususan, dasturlashning ustunlaridan biri bu, albatta, algoritm tushunchasidir. Ushbu tushunchaning kelib chiqishi bevosita matematika bilan bog'liq bo'lib, hozirgi kunda eng keng qo'llanilayotgan tushunchalardan biri hisoblanadi.

Mazkur bobda algoritm tushunchasi, uning bosqichlari va xossalari, turlari va tuzilishi, algoritmning samaradorlik darajasi kabi ma'lumotlar berilgan bo'lib, unda quyidagi bo'limlar yoritilgan:



1.1. Algoritm tushunchasi va uning xossalari

Algoritm tushunchasi hozirgi zamon matematikasining eng keng qo'llanilayotgan tushunchalaridan biri hisoblanadi. Algoritm so'zi o'rta asrda paydo bo'lgan bo'lib, Al-Xorazmiy nomi bilan bog'liqdir. Hozirgi paytda o'nlik sanoq sistemasida arifmetik amallarni bajarish usullarini hisoblash algoritmi soddagina bo'lgan misol sifatida keltirsa bo'ladi.

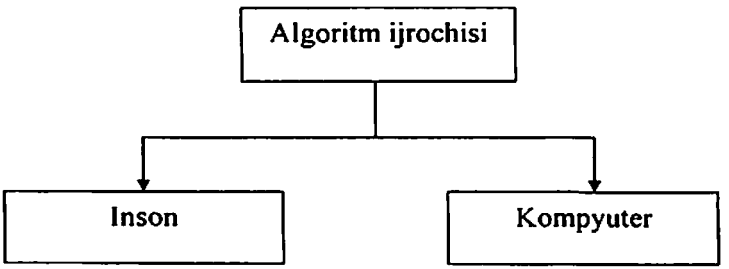
Ma'lumki, inson kundalik turmushida turli - tuman ishlarni bajaradi. Har bir ishni bajarishda esa bir qancha elementlarni ketma - ket amalga oshirishga to'g'ri keladi. Mana shu ketma-ketlik yozilsa, u bajariladigan ishning algoritmi bo'ladi.

Algoritm ma'lum bir buyruqlar to'plami bo'lib, bajaruvchi uchun aniq ko'rsatmalarni o'z ichiga mujassamlashtiradi. Ushbu buyruqlar bajaruvchiga ko'rsatilgan maqsadga erishish uchun asos bo'lishi kerak.

Demak, qo'yilgan masalani bajarishga ma'lum ketma-ketlikda elementlarni ijro etish orqali erishiladi. Bunda algoritmni bajaruvchi algoritm ijrochisi hisoblanadi. Umuman, algoritmlarni 2 guruhga (1.1-rasm) ajratish mumkin:

1-guruh algoritmlarining ijrochisi faqat inson bo'lishi mumkin.

2-guruh algoritmlarining ijrochisi ham inson, ham kompyuter bo'lishi mumkin.



1.1-rasm. Algoritm ijrochisi bo'yicha tasnifi

Bu guruh algoritmlari ijrochisini kompyuter zimmasiga yuklash mumkin. Buning uchun algoritmni kompyuter tushunadigan biror dasturlash tilida yozib, keyin uning xotirasiga kiritish kifoya.

Umuman olganda ijrochi algoritmda mavjud maqsadni bilmaydi. U bevosita keltirilgan buyruqlarni bajaradi. Informatikada algoritmlarni ijrochisi kompyuter deb hisoblanadi.

Shunday qilib, biz, **algoritm** deganda, berilgan masalani yechish uchun ma'lum tartib bilan bajarilishi lozim bo'lgan chekli sondagi ko'rsatmalar ketma - ketligini tushunamiz.

Algoritmlarni yozish uchun qo'llaniladigan tillar **algoritmik tillar** deb ataladi.

Algoritmik tilni kompyuter ham tushunsa, u holda bu til **dasturlash tili** deb ataladi. Demak, algoritmik yoki dasturlash tili ham berilgan masalani yechish algoritmini yozish vositalaridan biri hisoblanar ekan. Algoritm biron dastulash tilida yozilsa dastur hosil bo'ladi.

Algoritmlarni o'rganish davomida biz quyidagi asosiy tushunchalar bilan tanishamiz: algoritm, blok-sxema, chiziqli algoritmlar, tarmoqlanuvchi algoritmlar, takrorlash algoritmlari, iteratsion algoritmlar.

Biror masalani kompyuterda yechishda eng muhim va ma'suliyatli ishlardan biri bu masalani yechish algoritmini yaratish bo'lib, bu jarayonda bajarilishi lozim bo'lgan barcha bo'lajak buyruqlar ketma-ketligi aniqlanadi. Algoritmda yo'l qo'yilgan xatoliklar hisoblash jarayonining noto'g'ri bajarilishiga olib keladi, bu esa, o'z navbatida, xato natijaga olib keladi.

Umuman, istalgan masalaning yechimi deyilganda, masalani yechish algoritmi mavjudligi va ushbu algoritm natijaga erishishni ta'minlashi zarurligi tushuniladi.

Masalalarni, ularning qo'yilishi bo'yicha, 4 ta sinfga ajratish mumkin:

1. Aniq ta'riflangan va yechimga erishadigan algoritmlari mavjud masalalar. Ushbu sinfga doir masalalar bevosita to'g'ridan - to'g'ri kompyuterda bajariluvchi dasturlarga aylantirilishi mumkin.
2. Masala qo'yilishida yoki uning yechimida noaniqliklar mavjud bo'lib, algoritmda ushbu noaniqliklarni e'tiborga olish zaruriyati paydo bo'ladi. Bu yerda asosan tashqi muhitning o'zgarishi bilan algoritmning shunga moslanishi nazarda tutiladi.
3. Bilimlarni qayta ishlash doirasida berilgan idrokiy masalalar. Bu yerda mantiqiy tushunchalar bilan ishlay oladigan algoritmik tillarning mavjudligi va ular asosida kompyuter uchun dasturlar yaratish mumkinligi nazarda tutiladi.
4. Inson faoliyatiga bog'liq masalalarni modellashtirishga qaratilgan bo'lib, aniq yechish algoritmlari mavjud bo'lmagan masalalar. Ya'ni inson faoliyatini ma'lum bir model doirasida dasturlash mumkinligi

nazarda tutilgan bo'lib, algoritm ushbu faoliyatni to'liq qamrab ololmaydi, masalan, shaxmat o'yini.

Berilgan masala algoritmini yozishning turli usullari mavjud bo'lib, ular qatoriga so'z bilan, blok-sxema shaklida, formulalar, operatorlar yordamida, algoritmik yoki dasturlash tillarida yozish kabilarni kiritish mumkin.

Endi biror usulda tuzilgan algoritmning ayrim xossalarini va ularga qo'yilgan talablarni ko'rib chiqaylik.

1) Algoritm har doim to'liq bir qiymatlidir, ya'ni uni bir xil boshlang'ich qiymatlar bilan ko'p marotaba qo'llash har doim bir xil natija beradi.

2) Algoritm birgina masalani yechish qoidasi bo'lib qolmay, balki turli-tuman boshlang'ich shartlar asosida ma'lum turdagi masalalar to'plamini yechish yo'lidir.

3) Algoritmni qo'llash natijasida chekli qadamdan keyin natijaga erishamiz yoki natijaga erishish mumkin emasligi haqidagi ma'lumotga ega bo'lamiz.

Yuqorida keltirilgan xossalarni umumlashtirib, algoritmlarning quyidagi asosiy xossalarini ta'kidlash zarur.

1. Algoritm **boshlang'ich qiymatli** argumentlarga ega bo'ladi.

Algoritmni bajarishdan maqsad bu natija olishdir, ya'ni algoritm boshlang'ich qiymatlarni biror usul orqali natijaga aylantiruvchi avtomatdir.

Demak, algoritmni qo'yilgan masalaning har xil boshlang'ich qiymatlari uchun qo'llash mumkin. Lekin algoritmning ijrochisi inson bo'lsa, boshlang'ich qiymat ko'rsatilmagan bo'lishi ham mumkin, masalan, "qog'ozda kvadrat chizing"

Har xil boshlang'ich qiymatlar ishlatilishi mumkin bo'lganligi sababli algoritmning **ommaviylik** xossasi yuzaga keladi. Lekin ba'zi - bir hollarda faqatgina individual boshlang'ich qiymatga mo'ljallangan algoritm ham mavjud bo'ladi.

2. Algoritm ijrochiga **tushunarli** bo'lishi lozim, ya'ni ijrochining bazasida mavjud bo'lgan buyruqlar qo'llanilishi zarur. Shu bois, algoritmni yaratish jarayonida ijrochining imkoniyatlari va nozik tomonlari e'tiborga olinishi darkor.

3. Algoritm cheklangan qadamlar ketma-ketligidan iborat bo'ladi va har bir qadam to'liq bajarilgandan so'ng keyingi qadamga o'tiladi. Ushbu xossa algoritmning **diskretli** xossasi deb ataladi.

4. Algoritm chekli qadamlar bajarilgandan so‘ng tugatiladi. Ya’ni chekli qadamlardan so‘ng algoritm natijaga erishishi kerak yoki natija olish mumkin emasligi aniqlanishi lozim. Ushbu xossa algoritmning **natijaviylik** xossasi deb ataladi.

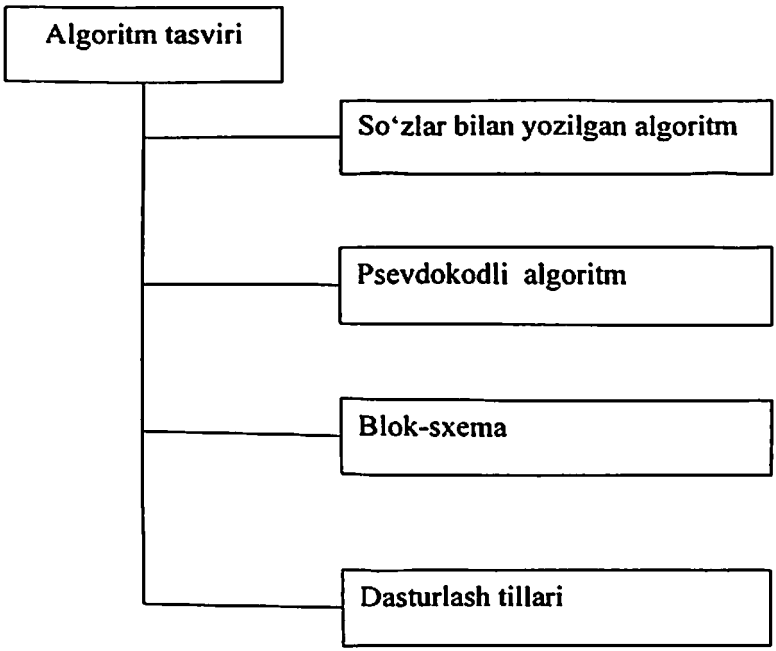
5. Algoritmning har bir qadami aniq ta’riflangan bo‘lib, ko‘p ma’noli bo‘lishi mumkin emas. Ya’ni har bir qadamdan so‘ng keyingi qadam aniqlangan bo‘lishi yoki algoritm tugatilishi kerak.

Algoritmning ushbu xossasi **aniqlik** xossasi deb yuritiladi va algoritmni bevosita kompyuterda bajarish uchun imkon beradi.

Ushbu xossaga binoan, bir xil boshlang‘ich qiymatlar bilan bajariladigan algoritm doimo bir xil natijaviy qiymatlar bilan tugallanadi.

6. Algoritmning har bir qadami aniq va chekli vaqt davomida bajarilishi lozim. Ushbu xossa algoritmning **samaradorligi** deb yuritiladi.

Algoritmning har bir qadamida bajariladigan ko‘rsatma **buyruq** deb aytiladi. Ushbu buyruqlar nafaqat chekli vaqt davomida, balki qisqa vaqt mobaynida tugatilishi lozim. Bunday shartlar barcha olimpiada masalalarida doimo keltiriladi.



1.2-rasm. Algoritm tasviri bo‘yicha tasnifi

1.2. So'zlar bilan yozilgan algoritm

So'zlar bilan yozilgan algoritmlar asosan insonlar uchun tuziladi. Har bir qadam raqamlanadi va yozuvlarda deyarli hech qanday qonun - qoidalarga amal qilinmaydi. Misol sifatida Yevklid algoritmini keltiramiz, ushbu algoritm ikki natural sonning eng katta umumiy bo'luvchisini aniqlashda qo'llaniladi:

1. Agar sonlar teng bo'lsa, ulardan biri chop etilsin, aks holda 2-bandga o'tilsin.
2. Ikkala sondan kattasi aniqlansin.
3. Katta sondan kichik son ayirilsin va natija katta songa berilsin.
4. 1-bandga o'tilsin.

Ushbu yozuvdan ko'rinib turibdiki, bevosita yozuvlarda ixtiyoriy qoidalarga asoslanib algoritm tuzish mumkin, masalan, bevosita formulalarni ham qo'llash mumkin.

1.3. Psevdokodli algoritm

Bu maxsus tanlangan so'zlar yordamida ma'lum bir qonun va qoidalarga asoslanib yoziladigan algoritmdir. Ushbu tartib algoritmlarni yagona ko'rinishda tasvirlashda juda qo'l keladi.

Bu yerda tanlangan maxsus so'zlar yagona ma'noga ega bo'lib, algoritmda ularni qalin qilib yoki ostiga chizish yo'li bilan ajratishadi. Ushbu uslub ham bevosita insonlarga mo'ljallangan bo'lib, algoritmni mashinaday bajarishga imkon beradi.

Misol sifatida yuqorida keltirilgan Yevklid algoritmini keltiramiz.

Yevklid algoritmi;
boshlash
kiritish m,n
L: agar m=n bo'lsa T nuqtaga o't;
agar m>n bo'lsa m:=m-n aksincha n:=n-m;
L nuqtaga o't;
T: tamom.

1.4. Blok-sxema tushunchasi va uning elementlari

Algoritmlarni yozish usullaridan biri blok - sxema bo'lib, u algoritmning ma'lum geometrik shakllar yordamida yaqqol yozilishidir.

Har bir geometrik shakl (blok) ma'lum ma'noni anglatadi. Bloklar o'zaro yo'naltirilgan chiziqlar orqali bog'lanadi va chiziqning yo'nalishi algoritmning bajarilish ketma - ketligini belgilaydi.

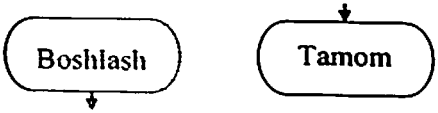
Blokdagi shakllar ichida oddiy tilda tushunarli matematik simvollarni qo'llash mumkin.

Blok - sxema algoritm tuzuvchi uchun dastur tuzilishini yaqqol tasavvur qilishning yaxshi usulidir. Berilgan masala algoritmining blok - sxemasini dasturlash tillarining qoidalariga moslab tuzish shart emas. Chunki blok - sxema shaklida yozilgan algoritmni dasturlash tillaridan xabarsiz har bir kishi ham o'qiy olishi lozim.

Agar masalaning blok - sxemasi berilgan bo'lsa, undan foydalanib, dastur tuzish mumkin. Buning uchun har bir blokni shu til qoidalari asosida ko'chirib yozish yetarli.

Blok - sxema quyidagi asosiy elementlardan iborat :

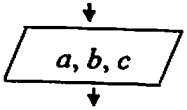
1) Algoritmning **boshlanishi** yoki **oxirini** bildiruvchi blok :



1.3-rasm. Algoritmning boshlash va tamomlash bloklari

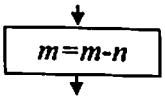
Ular shakl jihatidan bir xil bo'lgani uchun algoritm boshlanishini bildirgan blok ichiga "Boshlash", oxirini bildiruvchi blokning ichida esa "Tamom" so'zlari yozib qo'yiladi. Algoritmning boshlanishi (oxiri) ni bildiruvchi blokdan faqat chiqish (kirish) mumkin.

2) Blok - sxemalarda **ma'lumotlarni standart kiritish** va **chiqarish** blokining umumiy ko'rinishi parallelogramm shaklida bo'lishi mumkin. Qaysi o'zgaruvchilarning qiymatini kiritish kerak bo'lsa, shu o'zgaruvchilarning nomlari blokning ichiga yozib qo'yiladi. Bu blokka kirish va undan chiqish mumkin, ya'ni blok bitta kirish va bitta chiqish ko'rsatkichiga ega.



1.4-rasm. Algoritmda ma'lumotlatni kiritish va chiqarish bloki

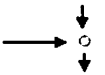
3) **Amallar bajaruvchi blok**ning ko'rinishi to'g'ri to'rtburchak shaklida bo'ladi.



1.5-rasm. Algoritmda amal bajarish bloki

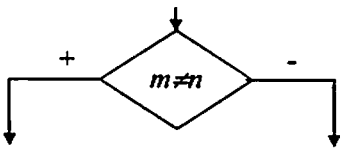
Bu ko'rinishdagi blok **chiziqli blok** ham deyiladi. Bajarilishi lozim bo'lgan ish blokning ichiga yozib qo'yiladi. Blokning ichida bir qancha formulalar ham bo'lishi mumkin. Blokdan chiqish uchun blok ichidagi hamma buyruqlarni bajarish lozim. Blokni tartibli raqamlash ham mumkin. Bu bloklar ham bitta kirish va bitta chiqish ko'rsatkichiga ega. Bu turdagi blok ichiga yozilgan tenglik belgisini matematik ma'noda tenglik deb tushunmaslik kerak, balki bu belgining o'ng tomonida turgan ifodaning qiymati chap tomonda turgan o'zgaruvchiga jo'natilyapti yoki o'zgaruvchiga qiymat berilyapti yoki qiymat ta'minlanayapti deb tushunish zarur.

Bloklar bir - birlari bilan yo'naltirilgan chiziqlar orqali tutashadi. Mantiq yuzasidan bir necha yo'nalishlar bir joyda tutashadigan bo'lishsa bu holda quyidagi "o" tugma belgisini qo'llash zarur bo'ladi:
masalan,



1.6-rasm. Algoritm yo'nalishlarini tutash belgisi

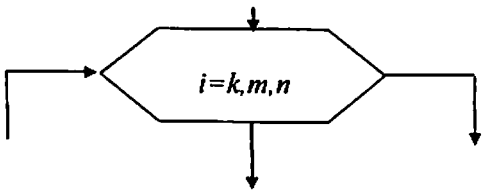
Shartlarni tekshirish uchun romb belgisi qo'llaniladi, masalan:



1.7-rasm. Algoritmda shartni tekshirish bloki

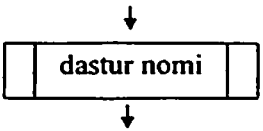
Shartli shaklda ikki vaziyatdan biri bajariladi, agar undagi ifoda bajarilsa yo'naltirilgan chiziq ustida "+" (yoki «ha» so'zi) , aks holda "-" ishorasi (yoki «yo'q» so'zi) qo'yiladi.

Takrorlanadigan buyruqlar jarayonini bajarish uchun quyidagi maxsus belgi qo'llaniladi:



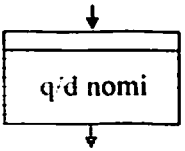
1.8-rasm. Algoritmda takrorlanish bloki

Dastur doirasidan chetda yaratilgan maxsus dasturlardan foydalangan holatda quyidagi belgi qo'llaniladi



1.9-rasm. Algoritmda maxsus tashqi dasturni kiritish bloki

Dastur doirasida yaratilgan qism dasturlardan foydalangan holatda esa quyidagi belgi qo'llaniladi:



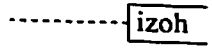
1.10-rasm. Algoritmda qism dasturni kiritish bloki

Blok-sxema bir betga sig'may qolgan paytda betlararo bog'lovchi shaklini qo'llash mumkin

1.11-rasm. Algoritm qaysi betda davom etishini ko'rsatuvchi belgi

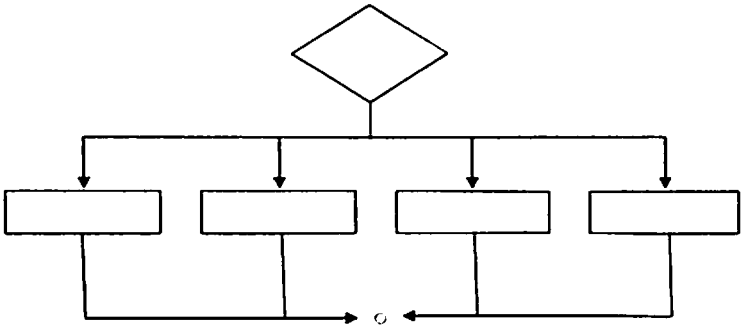
Bu yerda ikkinchi betning 9-blokiga o'tilishi ko'rsatilgan. O'z navbatida, 2-betda ham ushbu shakl bo'lishi kerak va unda qaysi betdan buyruq kelishi ko'rsatilgan bo'lishi lozim, masalan, 1.6, ya'ni birinchi betning oltinchi elementidan.

Bevosita sxemada izohlarni tasvirlashda quyidagi shakl qo'llaniladi:



1.12-rasm. Algoritmda izoh bloki

Hozirgi dasturlash tillarida mavjud bo'lgan **CASE** operatorini sxematik ravishda tasvirlashda quyidagi konstruksiya taklif qilinadi:



1.13-rasm. Algoritmda case operatorini tasvirlash bloki

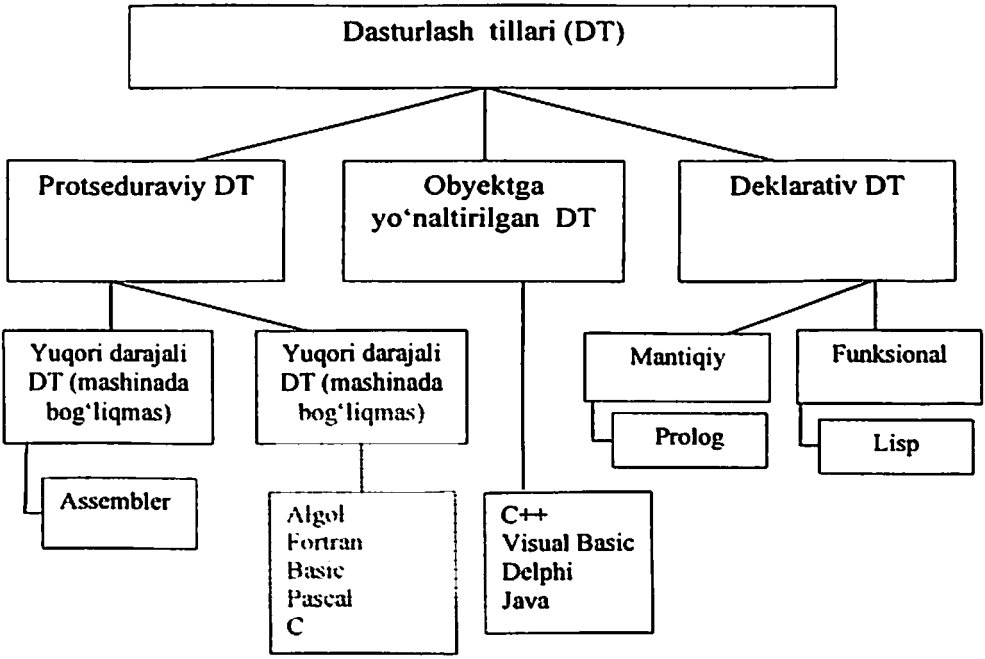
1.5. Dasturlash tillari va ularning tuzilmasi

Yaratilgan algoritmlarni bajarish uchun qo'llaniladigan maxsus qurilma bu elektron hisoblash mashinasi hisoblanadi va u hozirgi paytda kompyuter deb nomlanadi. Har qanday algoritm kompyuter tushuna oladigan tilda yozilishi kerak. Ushbu tillarni dasturlash tillari deb aytishadi, ushbu tilda yozilgan algoritmni esa dastur deb atashadi.

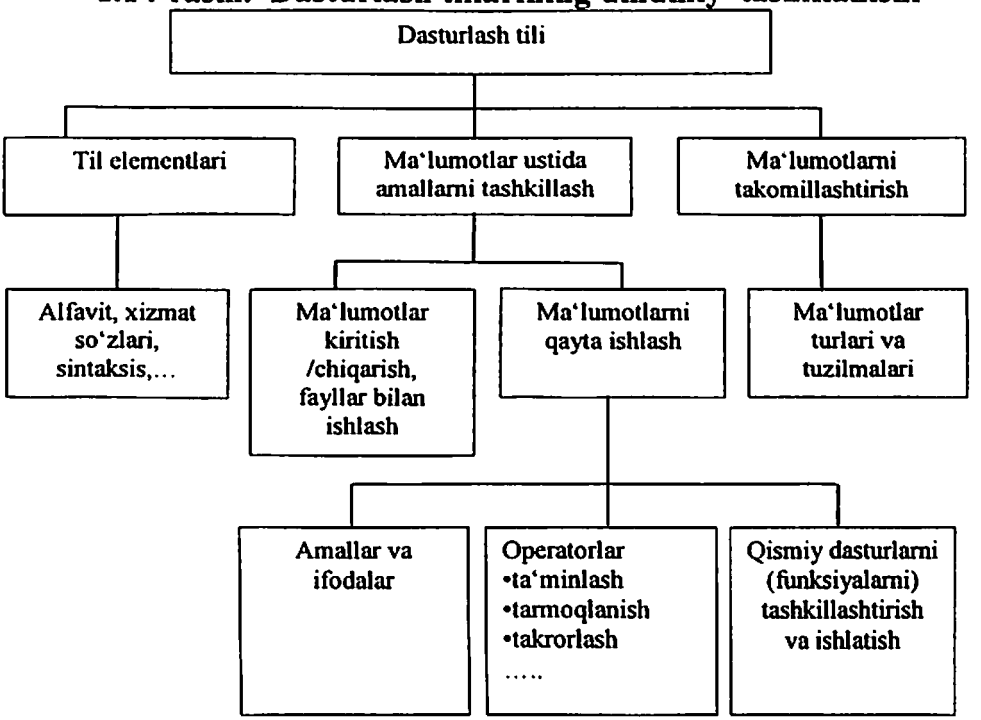
Shunday qilib, dasturlash tili – bu komputerlar uchun dasturlar yoziladigan, uni u yoki bu harakatlarni bajarishiga majbur qiladigan rasmiy til bo'lib, unda yozilgan ko'rsatmalar dastlabki kod deb ataladi.

Har qanday dasturlash tili ma'lumotlarni tashkil etish va ular ustida amallar bajarilishini amalga oshirish usullari va vositalari jamlanmasidan iborat bo'ladi.

Hozirgi vaqtda dasturlash tillari juda ko'p. Ularning umumiy tasniflanishi 1.14-rasmda tasvirlangan. Dasturlash tillari juda ko'p bo'lishaga qaramasdan, ularni o'rganish tarzi (sxemasi) deyarli bir xil. Chunki turli dasturlash tillarining tuzilmalarida umumiy elementlari va xusuyatlari juda ko'p (1.15-rasm).



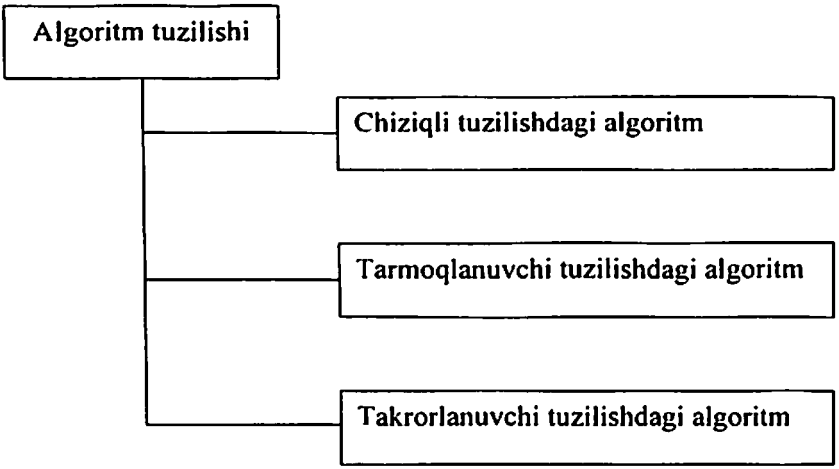
1.14-rasm. Dasturlash tillarining umumiy tasniflanishi



1.15-rasm. Dasturlash tilining umumiy tuzilmasi

1.6. Algoritmlarning tuzilishi bo'yicha tasniflanishi

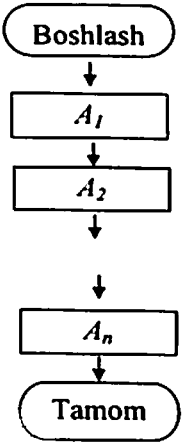
Har qanday algoritmni ma'lum bir mantiqiy elementlar to'plami shaklida yaratish mumkin. Algoritmlarni o'rganib chiqishdan qilingan xulosalar bo'yicha uni 3 ta mantiqiy qismlarga ajratish mumkin. Ya'ni har qanday algoritmda ushbu mantiqiy qismlar mavjud bo'ladi. Demak algoritmlarni o'rganish ushbu tarkibdagi mantiqiy qismlarni o'rganishdan iborat bo'ladi. Shundan kelib chiqqan holda algoritmlar asosan quyidagi sxemada keltirilgan tartib bo'yicha tasniflanadi:



Ushbu turdagi algoritmlar tayanch bo'lib, ularda doimo bitta kirish va bitta chiqish nuqtasi mavjud bo'ladi. Bularni birin-ketin ko'rib chiqamiz.

1.7. Chiziqli tuzilishdagi algoritmni dasturlash

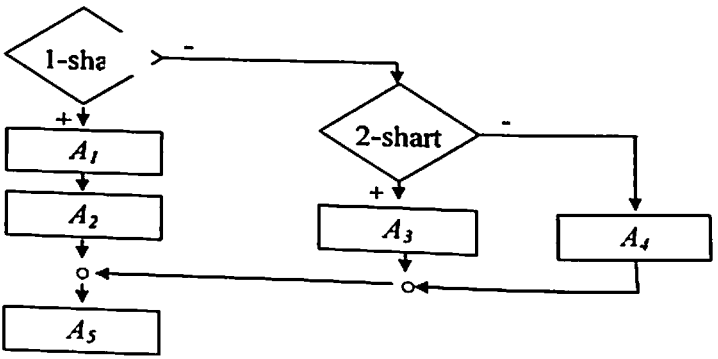
Chiziqli tuzilishdagi algoritmlar umuman mantiqiy shartlarni o'z ichiga olmagan bo'lib, har bir qadam bir marotaba va ketma-ket bajariladi:



Bu yerda A_1, A_2, .., A_n lar ketma-ket bajariladi va algoritm o'z ishini tugatadi.

1.8. Tarmoqlanuvchi tuzilishdagi algoritmni dasturlash

Tarmoqlanuvchi tuzilishdagi algoritmlarda kamida bitta mantiqiy shart bo'lishi kerak. Ushbu shartning qiymati bajarilishi lozim bo'lgan ketma - ketlikni o'zgartirishi mumkin bo'ladi. Masalan,



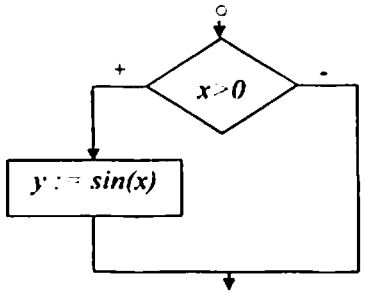
Ushbu algoritmda, agar 1 - shart bajarilsa, u holda A_1, A_2, A_5 qadamlar bajariladi.

Agar 1 - shart bajarilmasa va 2 - shart bajarilsa, bu holda A_3 va A_5 qadamlar bajariladi. Agar 1 - va 2 - shartlar bajarilmasa, bu holda A_4 va A_5 qadamlar bajariladi.

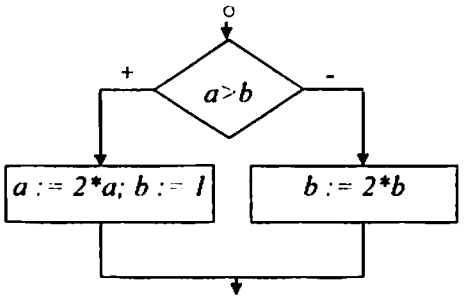
Tarmoqlanuvchi algoritmlarda barcha yo'nalishlar oxiri - oqibat bir nuqtada tutashishi shart.

Quyida har xil ko‘rinishdagi tarmoqlanuvchi algoritmlar keltirilgan:

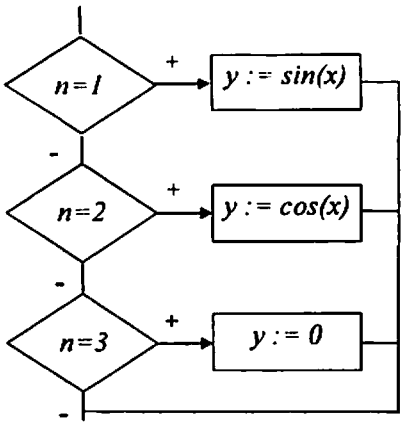
Agar x>0 bo‘lsa,
u holda y:=sin(x)



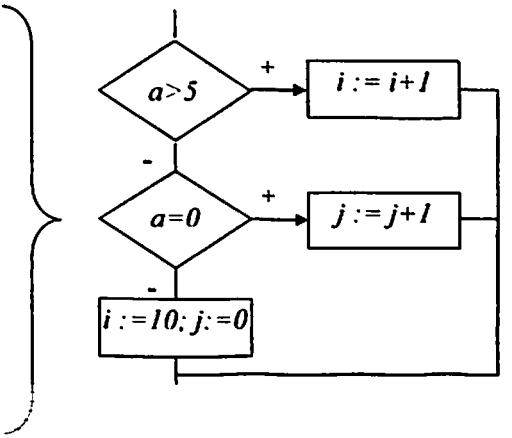
Agar a>b bo‘lsa,
u holda a:=2*a; b:=1
aksincha b:=2*b



Tanlash
n=1 bo‘lsa,u holda y:=sin(x)
n=2 bo‘lsa,u holda y:=cos(x)
n=3 bo‘lsa,u holda y:=0



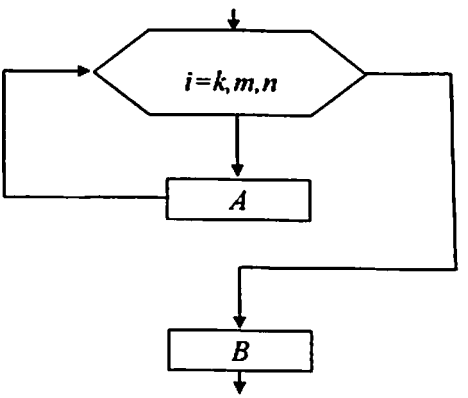
Tanlash
a>5 bo'lsa,u holda i:=i+1
a = 0 bo'lsa,u holda j:=j+1
aksincha i:= 10; j:=0



1.9. Takrorlanuvchi tuzilishdagi algoritmni dasturlash

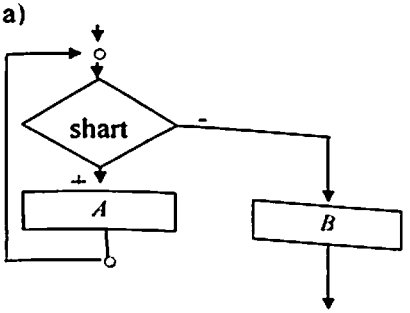
Ba'zi - bir algoritmlarda bajarilishi lozim bo'lgan qadamlar takrorlanishi talab qilinadi. Ushbu jarayon sikl deb ataladi. Takrorlanish qadamlariga ko'ra sikllar 2 ga bo'linadi: - qadamlar soni aniq sikl; - iteratsion sikl.

Qadamlar soni aniq ko'rsatilgan sikllarda chegaraviy qiymat aniq ko'rsatilgan bo'ladi. Bu yerda takrorlanuvchi tarkibni maxsus sikl yasovchilar yordamida tuzish va undagi qadamlar sonini aniqlash mumkin bo'ladi.

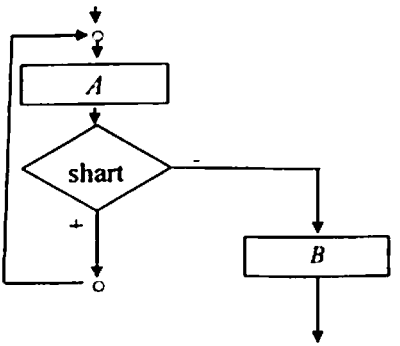


Bu yerda keltirilgan "*A*" bandi siklning sarlavhasi deb atalmish parametrlarga bog'liq bo'ladi, ya'ni *i=k,m,n* larga. Masalan *i*=1,10,1 bo'lsa "*A*" qadami 10 marta bajarilgan bo'ladi. Lekin ba'zida sikl bevosita ma'lum - bir shartni bajarilishi bilan bog'lanadi. Ya'ni agar shart bajarilsa hisoblash jarayoni takrorlanadi, aks holda sikl tugatiladi. Masalan,

a)



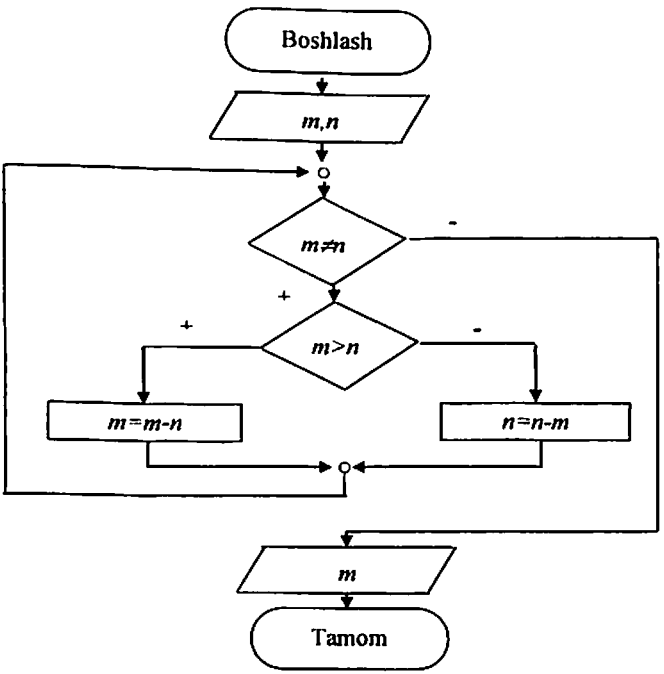
б)



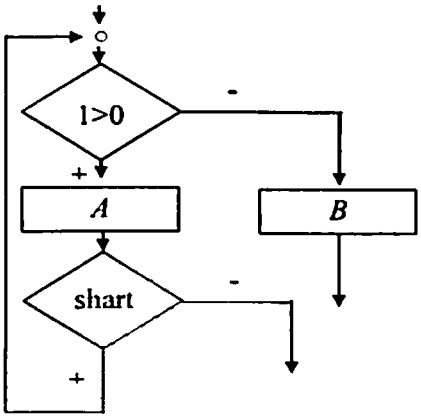
Keltirilgan a) va b) variantlarda sikllar bir - biridan tubdan farq qiladi. Chunki a) variantida shart bajarilgan holatda *A* bandi bajariladi, aks holda ushbu band umuman bajarilmaydi. Ikkinchi, ya'ni b) variantdagi siklda esa, shartning qiymati qanday bo'lishidan qat'iy nazar, kamida bir marotaba *A* bandi bajariladi.

Ushbu ko'rinishdagi takrorlanuvchi algoritmlar iteratsion algoritmlar deb ataladi. Dasturlarda ushbu jarayonli algoritmlarni qo'llashda ehtiyotkor bo'lish zarur, agarda sikldan chiqish ko'rsatilmagan bo'lsa, unda ushbu jarayon cheksiz bajarilishi mumkin.

Misol sifatida qadamlar soni aniq bo'lmagan Yevklid algoritmini to'liq keltiramiz:

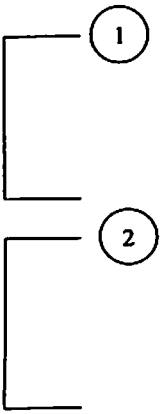


Shunga qaramay ba'zida cheksiz ko'rinishdagi algoritmlardan foydalanish ehtiyoji paydo bo'ladi. Ushbu ko'rinishlagi algoritmlarni, masalan, quyidagicha yaratish mumkin:

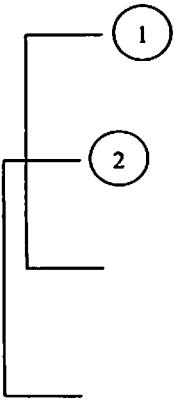


Bu yerda keltirilgan 1-shart, ya'ni "1>0" doimo bajariladi, shundan so'ng "*A*" qadami bajarilib, 2-shartga o'tiladi. Ushbu 2-shartda jarayondan chiqib ketish albatta bo'lishi kerak, aks holda jarayon cheksiz bajariladi va bu algoritmning noto'g'ri tuzilganidan dalolat beradi. Bundan tashqari bu yerda "*B*" band umuman bajarilmaydi.

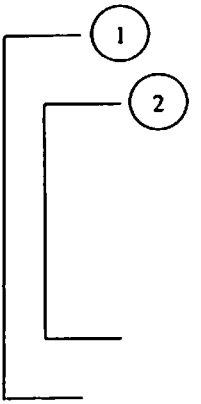
Takrorlanuvchi algoritmlarning yana bir xususiyati mavjud, bu ham bo'lsa ularning bir - biriga nisbatan joylashuvi. Masalan, sxematik ravishda quyidagicha tasvirlangan 2 ta takrorlanuvchi algoritmlar berilgan bo'lsin:



1-rasm



2-rasm



3-rasm

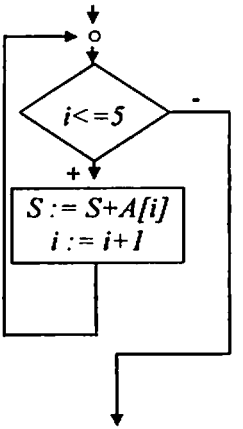
Bu yerda keltirilgan variantlardan 1- va 3-rasmlarda keltirilgan algoritmlarga o'rin bor. 2-rasmda keltirilgan algoritm, ya'ni ikki sikl bir-biri bilan kesishganda, ushbu holat xatoga olib keladi.

Quyida aniq misollar keltirilgan.

Sikl
agar i<=5 bo'lsa,u holda

S:=S+A[i]
i:=i+1

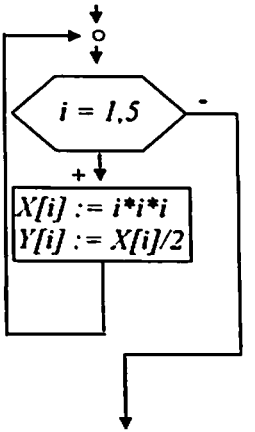
Sikl tugadi



Sikl
i uchun 1 dan to 5 gacha

X[i]:=i*i*i
Y[i]:=X[i]/2

Sikl tugadi

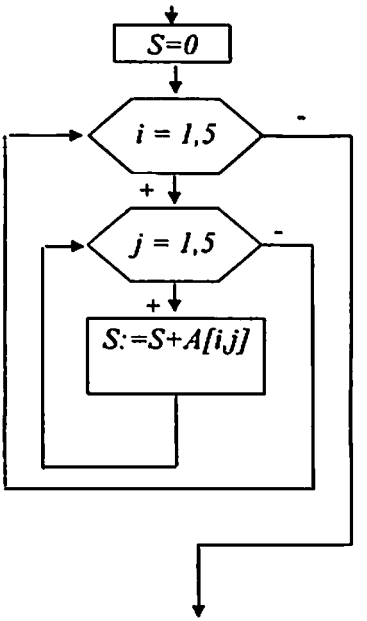


Ketma-ket keladigan sikllarga quyidagi misolni keltirsak bo'ladi:

S:=0;

Sikl
i uchun 1 dan to 5 gacha
Sikl
j uchun 1 dan to 3 gacha
S:=S+A[i,j]
Sikl tugadi

Sikl tugadi



Yuqorida keltirilgan misollardan shunday xulosa qilish mumkin. Har qanday takrorlanuvchi, ya'ni siklik algoritmlar bitta kirish va bitta chiqish nuqtasiga ega bo'ladi.

Umuman, har qanday algoritmni yuqorida keltirilgan 3 ta tipli, ya'ni chiziqli, tarmoqlanuvchi va takrorlanuvchi algoritmlardan foydalanib yasash mumkin. Bu esa o'z navbatida algoritmlarni yig'ish imkonini yaratadi, ya'ni masala kichik masalalarga bo'linadi va ular

uchun alohida algoritmli bloklar tuziladi va keyinchalik ular ketma - ket bir - biri bilan bog'lanadi. Ba'zida ular bir - birining tarkibiga kirishi ham mumkin.

Asosan katta loyihalarda ushbu texnologiya keng qo'llaniladi. Ya'ni, masalaning umumiy blok - sxemasi yaratiladi va ulardagi qadamlar birin - ketin tuzilib, dasturga qo'shilib boriladi. Ushbu texnologiyani qismlash deb atasak bo'ladi, chunki ular uchun yaratiladigan dasturni qism - dastur deb atashadi. Bu yerda oldindan yaratilgan algoritmlarni qo'llash imkoni tug'iladi va bu dasturlovchining qimmatli vaqtini tejashga olib keladi.

1.10. Algoritmlarning asosiy ko'rsatkichlari

Berilgan masalaning yechimini bir necha algoritmlar orqali aniqlash mumkin. Shu bois algoritmlarning sifat ko'rsatkichlarini aniqlash talab qilinadi. Bu yerda quyidagi ko'rsatkichlar kiritilgan:

Davomiylik ko'rsatkichi. Bu ko'rsatkich algoritmni bajarish uchun talab etilgan vaqtni belgilaydi. O'lchov birligi sifatida soniya (1 daqiqa = 60 soniya) qabul qilingan. Lekin bu o'lchov birligi kompyuterning texnik ko'rsatkichlariga bog'liq bo'lib qolganligi tufayli, asosan algoritmda bajariladigan operatsiyalar soni keltiriladi va u kompyuterning ishlash tezligiga bog'liq emas.

Ushbu ko'rsatkich bevosita olinadigan natija bilan kiritiladigan boshlang'ich qiymatlar hajmiga bog'liq bo'ladi. Shu bois har bir algoritm uchun uning davomiyligi $f(n)$ funksiyasi bilan belgilanadi. $f(n)$ – masalada keltirilgan n qiymatining oshishi bilan bajariladigan operatsiyalarning qay darajada o'sishini ko'rsatadi. Agar $f(n)$ funksiyasi n bilan chiziqli bog'langan bo'lsa , ushbu algoritm "yaxshi" algoritm , agar eksponentsial bo'lsa "yomon" algoritm deyiladi.

Hajmiy ko'rsatkichi. Bu ko'rsatkich algoritmning axborotiy murakkabligini anglatadi, ya'ni algoritmda zarur bo'lgan boshlang'ich, oraliq va natijaviy qiymatlarni saqlashda, qayta ishlashdagi cheklovlar ta'sirida paydo bo'ladi.

Bu yerda algoritmni tuzishda ishlatiladigan operatorlarning soni ham e'tiborga olinishi mumkin. Shundan kelib chiqqan holda algoritmning tarkibiy tuzilishi ham chetda qolishi kerak emas. Ya'ni algoritmda natija olinishida har xil yo'llar ko'rib chiqilishi va ularning har biri murakkab bo'ladigan bo'lsa, albatta ushbu algoritmni anglab olish qiyin kechadi.

Davomiylik ko'rsatkichi muhim bo'lganligi sababli unga batafsilroq to'xtalib o'tamiz. Demak, qo'yilgan masalaning yechish algoritmi tuzilgan bo'lsa, u cheklangan qadamlardan so'ng natija beradi. Albatta, har bir qadamda kompyuter tomonidan ma'lum – bir operatsiyalar bajariladi. Masalan, *a* va *b* o'zgaruvchilarning qiymatlarini o'zaro almashtirib qo'yish kerak bo'lsa, uni 3 qadamda amalga oshirish mumkin:

1. temp := a
2. a := b
3. b := temp

Lekin, butun va haqiqiy sonlar bilan bajarilganda sarflangan vaqt bir xil bo'lmaydi. Shunga qaramay, davomiylik ko'rsatkichida bunday vaziyatlarga e'tibor berilmaydi. Keyingi misolga nazar tashlasak, bu yerda 1 dan *n* gacha bo'lgan natural sonlar yig'indisini hisoblash talab etiladi.

```
S:=0;
Sikl
  i uchun 1 dan to n gacha
  S:=S+i
Sikl tugadi
```

Bunday ko'rinishdagi algoritmda bizga *n* ta qiymatlarni hisoblash uchun *n* ta qadamda operatsiyalarni bajarishga to'g'ri keladi. Shu bois davomiylik ko'rsatkichi $f(n)=O(n)$ deb qabul qilsak bo'ladi. Bu yerda ishlatilgan $O(n)$ belgisi "katta O" deb o'qiladi. Umumiy holda, agarda $f(n)=O(g(n))$ bo'lsa, u holda $f(n)$ funksiyasi katta *n* uchun $g(n)$ tartibli bo'ladi deb aytiladi. Kelgusida biz algoritmning samaradorlik darajasi $f(n)=O(g(n))$ tartibli bo'ladi deb so'z yuritamiz.

Yana bir necha misol keltiramiz.

Sinfda *n* ta talaba bo'lsa, birinchi partaga 2 ta talabani nechta usul bilan o'tqazish mumkin. Kombinatorikadan bilamiz, bu yerda jami bo'lib $n(n-1)$ ta variant mavjud bo'ladi. Bizda kiritilgan belgilashlar bo'yicha bu $O(n^2)$ tartibli bo'ladi.

Agarda birinchi partada 3 ta talabani o'tqazish kerak bo'lsa, u holda $n(n-1)(n-2)/6$ ta variantni ko'rib chiqishni taqozo etadi, ya'ni algoritmimiz $O(n^3)$ tartibli bo'ladi. Ko'p hollarda $O(n^k)$ tartibli ko'rsatkich yetarli bo'lmaydi.

Masalan, agarda kompyuterimizning tezligi 1 GGs bo'lsa, unda bu taxminan 300 Mflopsni tashkil qiladi. Biz ko'rayotgan algoritm $O(n^3)$ tartibli bo'lsa va n=10000 bo'lganda, bu taxminan 10^{12} ta hisoblash ishlarini bajarish bo'lsa, yuqoridagi kompyuter bilan bizga 10^{12} : $300*10^6$ soniya talab etiladi, bu esa taxminan bir soat vaqtni talab qiladi. Albatta bunday algoritmlar ko'plarni qoniqtirmaydi. Lekin bundan ham ko'p vaqt talab etadigan algoritmlar mavjud. Ular uchun $O(2^n)$ yoki $O(n!)$ variantlarni ko'rib chiqish talab etilishi mumkin. Bu tipdagi masalaga yuqorida ko'rib o'tilgan misolni keltirish mumkin. Ya'ni, sinfdagi n ta talabani n ta o'ringa talabalarni nechta usul bilan o'tqazish mumkin. Bu yerda bizga $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1 = n!$ ta variantni ko'rib chiqish kerak bo'ladi. Agar n = 10 ga teng bo'lsa bu 3 milliondan ko'p variantni tashkil qiladi.

O'z-o'zidan ko'rinib turibdiki, bunday algoritmlarni juda ham ehtiyotkorlik bilan qo'llash kerak bo'ladi. Lekin shunga o'xshash hollarga duch kelganda masalaning shartidan kelib chiqqan holda variantlarni qisqartirish kerak bo'ladi. Yuqoridagi misolda buni amalga oshirish mumkin emas. Chunki keraksiz variantlar uchun hech qanday mezon berilmagan. Masalan, bo'ylari past talabalar birinchi partadan o'tirish shartini kiritsak, unda ushbu shartni qoniqtirmagan variantlarni ko'rib o'tirish shart bo'lmaydi. Ya'ni, faraz qilamiz, 5 ta talabani oldingi o'rinlardan o'tqazdik, agarda shart bajarilmasa, unda qolgan 5 tasini joylashtirish shart emas. Shu yo'l orqali ko'rib chiqilayotgan variantlarni qisqartirish mumkin.

Shu bois, ko'p hollarda quyidagi samarali ko'rsatkichlarga erishish kerak, masalan, $O(\log^2 n)$, $O(n)$, $O(n\log^2 n)$, $O(n^2)$.

1.11. Dastur yaratish bosqichlari

Kompyuter bilan bevosita ishlashdan oldin qanday bosqichlarni bajarish kerakligini ko'rib chiqamiz. Istalgan hayotiy yoki matematik, fizik, (tijorat, buxgalteriya) va hokazo masala shartlarini ifoda qilish dastlabki ma'lumotlar va fikrlarni tasvirlashdan boshlanadi. So'ngra esa yechishning maqsadi, yani masalani yechish natijasida ayni nimani yoki nimalarni aniqlash zarurligi ko'rsatiladi.

Masala shartining aniq ifodasi masalaning matematik qo'yilishi deb ham ataladi va istalgan masalani yechish eng avval uning qo'yilishidan boshlanadi.

Masala qo'yilishida boshlang'ich ma'lumotlar yoki argumentlarning qiymatlari aniqlanishi kerak bo'ladi.

Masalani qo'yish uni yechishning **birinchi bosqichi** bo'ladi.

Amaliy masalalarni hal etishda ob'ektlar - tabiat hodisalari, fizik yoki ishlab chiqarish jarayonlari, mahsulot ishlab chiqarish jarayonlari, ishlab chiqarish rejalari va shu kabilar bilan ish ko'rishga to'g'ri keladi.

Ana shunday masalalarni qo'yish uchun, birinchi navbatda, tekshirilayotgan ob'ektni matematik atamalarda tavsiflash, ya'ni uning matematik modelini qurish kerak, bu ifoda esa haqiqiy ob'ektni tekshirishni matematik masalani yechishga keltirish imkonini beradi. Modelni haqiqiy ob'ektga moslik darajasi amaliyotda tajriba orqali tekshiriladi. Bu bosqich masala yechishning **ikkinchi bosqichini** tashkil qiladi (Masalan, $S=a \cdot b$ formula orqali to'g'ri to'rtburchakning yuzini hisoblash, bu yerda a,b - boshlang'ich ma'lumotlar, S esa natija). Masalaning matematik modeli yaratilgandan so'ng, yechish usuli izlana boshlanadi. Ayrim holda, masalaning qo'yilishidan keyin to'g'ridan - to'g'ri masalani yechish usuliga ham o'tish kerak bo'ladi.

Bunday masala oshkor ko'rinishdagi matematik model bilan ifodalanmasligi ham mumkin. Bu bosqich masalani kompyuterda yechishning **uchinchi bosqichini** tashkil qiladi.

Navbatdagi **to'rtinchi bosqichda** masalani kompyuterdan foydalanib yechish uchun uning algoritmi tuziladi.

Algoritm turli xil ko'rinishda yozilishi mumkin. Algoritm kompyuterda bajarilishi uchun bu algoritm dasturlash tilida yozilgan bo'lishi lozim. Masalani yechishning bu bosqichi **beshinchi bosqich** bo'lib, unda biror - bir usulda yozilgan algoritm ma'lum bir dasturlash tiliga ko'chiriladi. Masalan, agar algoritm blok - sxema ko'rinishida tasvirlangan bo'lsa, uni dasturlash tiliga ko'chirish uchun har bir blokni tilning mos buyruqlari bilan almashtirish yetarli.

Oltinchi bosqich - dastur ko'rinishida yozilgan algoritmni kompyuter yordamida bajarish. Bu bosqich natija olish bilan tugallanadi. Bu bosqich dastur tuzuvchi uchun eng qiyin hisoblanadi. Chunki dasturni mashina xotirasiga kiritish paytida ayrim xatoliklarga yo'l qo'yish mumkin. Nihoyat, masala yechishning **yettinchi bosqichi** olingan natijalarni tahlil qilishdir. Bu bosqich olingan natija qanchalik haqiqatga yaqinligini aniqlash maqsadida bajariladi. Natijalarni tahlil qilish zarur bo'lgan holda algoritm yechish usulini va modelni aniqlashtirishga yordam beradi.

Ushbu bosqichlardan kelib chiqqan holda, dastur tuzishda quyidagi qoidalarga amal qilish maqsadga muvofiq bo'ladi:

1. Qo'yilgan masalani to'liq o'rganib olish zarur va shundan kelib chiqqan holda uning matematik modelini to'liqligini ta'minlash mumkin bo'ladi. Aks holda olinadigan natijalar kutiladigan natijalarga mos kelmaydi.
2. Kompyuterda bajariladigan hisoblash jarayonlarini e'tiborga olish zarur bo'ladi. Masalan, haqiqiy sonlar ustida bajariladigan arifmetik amallar natijalari taqribiy hisoblanishini e'tiborga olish zarur.
3. Algoritmlarni yaratishda uning oddiyligiga va tushunarli bo'lishiga harakat qilish zarur. Bu yerda nofaqat mazmunan, balkim shaklan algoritmni dasturlash tilida yozish nazarda tutilgan. Ya'ni dasturni kichik bo'laklarga ajratish, ularda qo'llaniladigan algoritmlarni oddiy dasturiy operatorlar orqali belgilash zarur.

Topshiriqlar

1. Agar $a=3$, $b=5$ va $c=2$ bo'lsa, ularning qiymatlarini quyidagi operatorlar ketma-ketligidan so'ng aniqlang:

a) *a:=a+1; b:=a+b; c:=a+b; a:=sqrt(a)*
b) *s:=a*b+2; b:=b+1; a:=c-b*2; b:=b*a;*
v) *b:=b+a; c:=c+b; b:=8/b*c;*
g) *p:=c; c:=b; b:=a; a:=p; c:=a*b*c*p;*
d) *c:=a*(b-3); b:=b-3; a:=(c+1)/2*b; c:=(a+b)*a;*
e) *x:=a; a:=b; b:=c; c:=x; a:=sqrt(a+b+c+x-4);*
j) *b:=(a+c)*2; a:=lg(b*2); c:=c*a*b.*

2. Quyidagi formula bo'yicha funksiyaning qiymatini hisoblashni psevdokod operatorlari yordamida algoritmini tuzing:

a) $y = \frac{a^2+1}{\sqrt{a^2+1}}$ b) $x = \sqrt{\frac{2a+sin|3a|}{3,56}}$ c)

$$x = 3,56(a+b)^3 - 5,8b^2 + 3,8a - 1,5$$

3. Dehqon daryoning chap qirg'og'ida bo'ri, echki va karam bilan turibdi. U bularni hammasini o'ng qirg'oqqa o'tkazishi kerak. Uning qayig'i juda kichik bo'lgani uchun faqat bitta yo'lovchini olishi mumkin - yoki bo'rini, yoki echkini, yoki karamni. Yana, agar bo'ri va echki bir qirg'oqda qoldirilsa, bo'ri echkini yeb qo'yadi, agar echki va karam bir

qirg'oqda qoldirilsa, echki karamni yeb qo'yadi. Hayvonlar faqat dehqon borligidagina tinch turishadi. Dehqon qanday algoritm asosida ish tutishi kerakligini aniqlang?

4. Daryo qirg'og'iga 2 ta askar kelishdi. Ular daryoning narigi qirg'og'iga o'tishlari kerak. Qirg'oq yaqinida ikkita bola qayiqda suzib yuribdi. Qayiq shunchalik kichkinaki yoki faqat ikkita bolani, yoki faqat bitta askarni ko'tarishi mumkin. Qanday qilib askarlar daryodan o'tib olishadi va qayiqni bolalarga qaytarib berishadi?

5. Quyidagi psevdokod operatorlaridan so'ng S ning qiymatini aniqlang:

a)
```
S:=1; a:=1
Sikl
  i uchun 1 dan to 3 gacha
  S:=S+(i+1)*a
  a:=a+2
Sikl tugadi
```

b)
```
Sikl
  i uchun 1 dan to 3 gacha
  S:=0
  Sikl
  j uchun 2 dan to 3 gacha
    S:=S+i+j
  Sikl tugadi
Sikl tugadi
```

c)
```
S:=0; i:=1;
Sikl
 i>1 bo'lganga qadar
  S:=S+1/i
  i:=i-1
Sikl tugadi
```

d)
```
a:=1; b:=1; S:=0;
    Sikl
 a<=5 bo'lganga qadar
  a:=a+b; b:=b+a;
  S:=S+a+b
Sikl tugadi
```

6. Tarmoqlanuvchi algoritmlar yordamida masalani yechimini aniqlang:

a) Berilgan a, b va c lardan nechtasi musbat son.

b) Berilgan a, b va c larni o'sish tartibi bilan chiqaring.

7. Bir kishi aytdi «Men yolg'onchiman yoki qora sochliman». U kishi kimligini aniqlang.

8. *A* kishi aytdi: «Hech bo'lmaganda birimiz qoramiz», *B* kishi aytdi: «Hech bo'lmaganda birimiz yolg'onchimiz». Ular kimlar?

9. Quyidagi jadvalda 1-qator sonlari bilan 2-qator sonlari orasidagi qonuniyatni aniqlang va oxirgi katakdagi sonni toping:

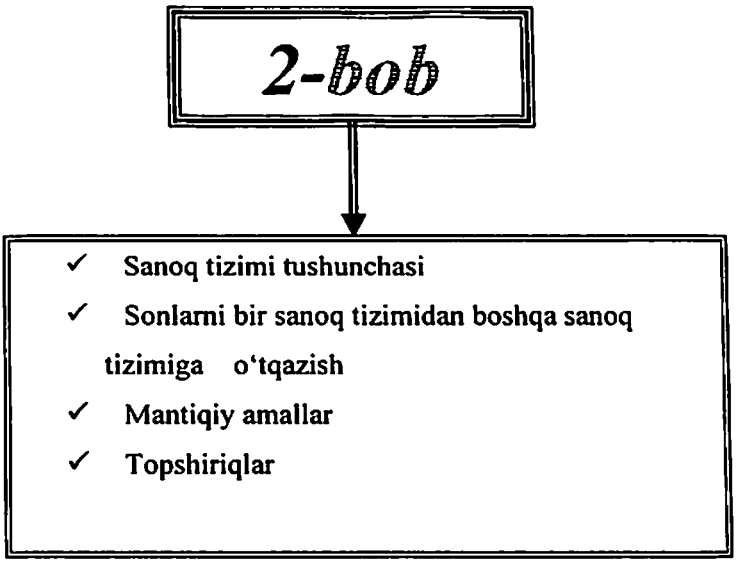
4	5	6	7	8	9
61	52	63	94	46	??

10. Yuqoridagi 5-topshiriqda keltirilgan algoritmlarning blok-sxemani tuzing.

2-bob. TURLI SANOQ TIZIMLARIDAGI SONLAR USTIDA AMALLAR BAJARISHGA DOIR DASTURLAR TUZISH

Tarixga nazar tashlaydigan bo'lsak, barcha xalqlarda hisob-kitob ishlarini bajarish uchun raqamlar va ular bilan amallarni bajarish qoidalari ishlab chiqilgan. Natijada sanoq tizimi yaratilgan. Ushbu soha bilan tanishmay turib, dasturlashtirish sohasida katta yutuqlarga erishish mumkin emas.

Mazkur bobda sanoq tizimlari va ular yordamida arifmetik amallarni bajarilishi haqida umumiy tushunchalar berilgan:



2.1. Sanoq tizimi tushunchasi

Sonlarni tasvirlashda ishlatiladigan belgilar va qoidalar to'plamiga sanoq tizimi deb ataladi. Belgilar to'plami sanoq tizimining alifbosi deyiladi. Amaliyotda ikki xil, yani pozitsion va nopozitsion sanoq tizimlari qo'llaniladi. Sonlarning qiymati undagi raqamlarining sonda joylashgan o'rniga qarab belgilanadigan sanoq tizimi pozitsion sanoq tizimi deb ataladi. Masalan, o'nlik sanoq tizimida o'nta 0,1,2,3,4,5,6,7,8,9 raqamlaridan foydalaniladi, demak uning alifbosi

T_{10}={0,1,2,3,4,5,6,7,8,9} va 52,25,255 sonlarini tasvirlanishida 2 va 5 raqamlaridan foydalanilsada bu sonlarning qiymati turlichadir. Raqamlarning qiymati ularni sonda joylashgan o'rniga bog'liq bo'lmasa, bunday sanoq tizimi nopozitsion sanoq tizimi deyiladi. Masalan, Rim sanoq tizimi nopozitsion sanoq tizimiga misol bo'la oladi. Misol uchun, 20 soni bu tizimda quyidagicha yoziladi *XX*, ya'ni tegishli belgi sonni qanday o'rinda turishidan qat'iy nazar har doim bir xil qiymatni ifoda etadi. Ushbu sanoq tizimi hozirgi paytda turli tarixiy sanalarni yozishda, kitob boblarini, soat raqamlarini belgilashda qo'llaniladi.

Pozitsion sanoq tizimining nopozitsion sanoq tizimidan qulaylik tomoni shundaki, unda katta sonlarni qisqa qilib yozish mumkin.Pozitsion sanoq tizimida istalgan son raqamlar ketma-ketligida yoziladi, butun va kasr qismi vergul bilan ajratiladi. O'z navbatida pozitsion sanoq tizimi ham turli sanoq tizimlariga bo'linadi va ulardan ayrimlari kompyuterlarda keng qo'llaniladi.

Pozitsion sanoq tizimida sonni yozish uchun qo'llaniladigan turli raqamlarning soni *d*-sanoq tizimining asosi deb aytiladi va aksincha sanoq tizimini *d* asosi shu sanoq tizimida qatnashadigan raqamlar sonini bildiradi. Biroq, hech bir sanoq tizimida uning asosi *d* ga teng bo'lgan raqam (A_i) qatnashmaydi, ya'ni doimo quyidagi tengsizlikning bajarilishi shartdir:

$$0 \le A_i < d$$

bu yerda, masalan, agarda d = 2 bo'lsa, A_i =0 yoki 1 bo'lishi mumkin.

Umumiy holda pozitsion sanoq tizimda *d*-asosli istalgan *A* sonni darajali qatorning yig'indisi ko'rinishida quyidagicha tasvirlash mumkin:

$$A_{(d)}=A_m d^m+A_{m-1} d^{m-1}+\ldots+A_1 d^1+A_0 d^0+A_{-1} d^{-1}+\ldots \quad (*)$$

bu yerda

d - sanoq tizimining asosi, m - son razryadining nomeri, A_m - m-nchi razryadda turuvchi koeffitsiyent.

$A_{(d)}$ yozuvida sanoq tizimining asosi d qavslarda indeks ko'rinishida yoziladi. O'nlik sanoq tizimi bundan mustasno.

$A_{(d)}$ sonini yuqoridagi formulaga mos keluvchi ko'rinishdagi qisqartirilgan yozuvini raqamlar ketma-ketligi ko'rinishida quyidagicha yozish qabul qilingan:

$$A_{(d)} = A_m A_{m-1} \ldots A_1 A_0 , A_{-1} \ldots A_{-m}$$

Ushbu ketma-ketlikda sonning butun qismi bilan kasr qismi vergul bilan ajratiladi. Agar manfiy darajalar bo'lmasa vergul tushirib yuboriladi. Verguldan boshlab sanaladigan raqamlar o'rni razryad yoki xona deyiladi. Masalan, 811 soni uch xonali, yoki uch razryadli deyiladi. Razryadlar soni, masalan m uchun, unda ko'pi bilan nechta har xil sonni tasvirlash quyidagicha aniqlanadi

$$N(m)=d^m.$$

Masalan, o'nlik sanoq tizimida m=3 bo'lsa, demak $N=10^3=1000$. Haqiqatan ham, uch xonali sonlar 000 dan boshlanib 999 bilan tugaydi, demak hammasi bo'lib 1000 ta har xil son mavjud bo'ladi.

Pozitsion tizimda d asosli sonning har bir razryadining qiymati undan o'ng tomonda bo'lgan qo'shni razryadni qiymatidan katta hisoblanadi.

Kompyuterlarda o'nli bo'lmagan pozitsion sanoq tizimlari: ikkilik, sakkizlik, o'n oltiliklar sanoq tizimlari ham qo'llaniladi.

Ikkilik sanoq tizimida faqat ikkita 0 va 1 raqamlaridan foydalaniladi, demak uning alifbosi $T_2=\{0,1\}$.

Kompyuterda o'nli songa nisbatan ikkilik sonini tasvirlash uchun ko'proq razryad talab etiladi. Shunga qaramasdan ikkilik tizimini qo'llash kompyuterni loyihalash va ishlatish uchun ko'proq qulaylik tug'diradi, chunki kompyuterda ikkilik sonini tasvirlash uchun istalgan sodda faqat ikki turg'un holatdan birini ifodalovchi elementlardan foydalanilishi mumkin. Ikkilik tizimini boshqa afzalligiga uning ikkilik arifmetikasining soddaligini aytish kifoyadir. Sakkizlik sanoq tizimida sakkizta 0,1,2,3,4,5,6,7 raqamlari ishlatiladi, demak $T_8=\{0,1,2,3,4,5,6,7\}$.

O'n oltilik sanoq tizimida sonni tasvirlash uchun 16 ta ya'ni 0 dan boshlab 15 gacha bo'lgan raqamlar foydalaniladi. Ammo bitta raqamni ikkita belgi bilan ifodalamaslik maqsadida 9 dan katta sonlar uchun maxsus belgilar kiritiladi. Dastlabki o'nta raqamni 0 dan 9 gacha bo'lgan raqamlar bilan, so'nggi katta sonlarni esa lotin harflari, ya'ni o'nni - A, o'n birni - B, o'n ikkini - C, o'n uchni - D, o'n to'rtni - E, o'n beshni - F bilan belgilaymiz, demak $T_{16}=\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$.

Sanoq tizimining asosi kamayishi bilan bitta songa ajratilgan razryadlar soni oshib boradi.

2.2. Sonlarni bir sanoq tizimidan boshqa sanoq tizimiga o'tkazish.

Odatda axborotlarni kompyuterda qayta ishlash uchun ular kompyuterga o'nlik sanoq tizimida kiritiladi, natija ham o'nlik sanoq tizimida chiqarib beriladi. Biroq axborotlar kompyuterda boshqa sanoq tizimlarida qayta ishlanadi.

Sonlarni bir sanoq tizimidan boshqasiga o'tkazishni kompyuterning o'zi quyida keltiriladigan qoidalarga muvofiq ravishda maxsus dasturlar asosida avtomatik tarzda bajaradi.

Sanoq tizimlari			
O'nlik	Sakkizlik	Ikkilik	O'n oltilik
0	0	000	0
1	1	001	1
2	2	010	2
3	3	011	3
4	4	100	4
5	5	101	5
6	6	110	6
7	7	111	7
8	10	1000	8
9	11	1001	9
10	12	1010	A
11	13	1011	B
12	14	1100	C
13	15	1101	D
14	16	1110	E
15	17	1111	F

Sonni d_1-asosli sanoq tizimidan d_2-asosli sanoq tizimiga o'tkazishni ikki xil qoidasi bir-biridan farq qiladi.

Agar $d_1 < d_2$ bo'lsa, u holda sonni d_1 - asosli sanoq tizimidan d_2-asosli sanoq tizimiga o'tkazish uchun o'tkaziladigan son (*) formulaga

asosan yoyib chiqiladi va so'ng qator yig'indisi hisoblanadi. Ushbu jarayonda barcha arifmetik amallar d_2- asosli sanoq tizimining qoidalari bo'yicha amalga oshiriladi.

Misol, $X=100101_{(2)}$ sonini o'nli sanoq tizimiga o'tkazamiz.

$$100101_{(2)} = 1\cdot2^5 + 0\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 = 32+4+1=37.$$

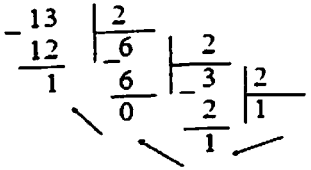
Natija: $100101_{(2)} = 37$

Ushbu qoida asosida, juda osonlik bilan ikkilik va sakkizlik tizimidagi sonlarni o'nlik sanoq tizimiga o'tkazish mumkin. Biroq, yuqoridagi qoidadan farqli ravishda ikkilik tizimidagi sonni sakkizlik tizimiga ham o'tkazish mumkin. Buning uchun (*) qator yig'indisini hisoblashni sakkizlik arifmetika qoidasiga asosan bajarish kerak bo'ladi.

Agar $d_1>d_2$ bo'lsa, u holda butun va kasr sonlarini bir sanoq tizimidan boshqasiga o'tkazishning quyidagi qoidalaridan foydalaniladi. Ya'ni d_1-asosli sanoq tizimidagi butun sonni d_2-asosli sanoq tizimiga o'tkazish uchun u o'tkaziladigan sanoq tizimining asosi d_2 ga ketma-ket bo'linadi. O'tkaziladigan son va bo'linmani bo'lish toki oxirgi qoldiq d_2-1 dan kichik yoki unga teng bo'lguncha davom ettiriladi. d_2-tizimidagi yangi son bo'lish natijasida hosil bo'lgan qoldiq va oxirgi bo'linmalarni teskari yo'nalishida yozish bilan o'qiladi. Oxirgi bo'linma yangi sonning eng katta razryadini beradi. Barcha amallar dastlabki son berilgan d - asosli sanoq tizimida bajariladi. Yangi son asosi eski d_1- asosli tizim raqamlari bilan yoziladi.

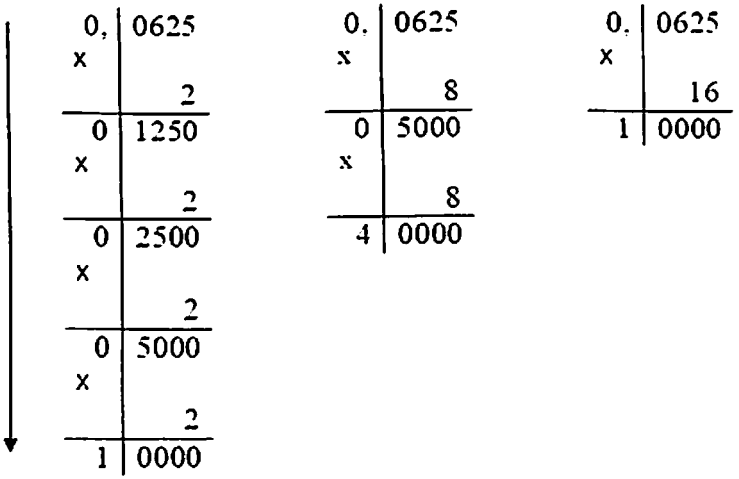
Kasr sonlarni bir sanoq tizimidan boshqa sanoq tizimiga o'tkazish uchun o'tkaziladigan d_1- tizimidagi sonni o'tkazishimiz kerak bo'lgan d_2- tizimining asosiga ketma- ket ko'paytiramiz va har bir ko'paytirishimizdan so'ng uning butun qismini ajratamiz. d_2-tizimidagi yangi son (verguldan keyin) ko'paytmalarning butun qismlarining ketma-ketligi ko'rinishida yoziladi. Ko'paytirish to ko'paytmani kasr qismida nollar hosil bo'lgunga qadar yoki oldindan ko'zda tutilgan aniqlik bajarilgunga qadar davom ettiriladi. Kasr sonni yangi d_2-tizimidagi birinchi ajratilgan butun qismidan boshlab oxirgi butun qismiga pastga qarab o'qiladi. Hisoblashlar o'tkazilayotgan son yozilgan sanoq tizimida amalga oshiriladi.

Misol : $X=13$ sonini o'nlik sanoq tizimidan ikkilik sanoq tizimiga o'tkazamiz.



Natija: $13_{(10)} = 1101_{(2)}$

Endi quyidagi o‘nli kasr sonini ikkilik, sakkizlik va o‘noltilik sanoq tizimlariga o‘tkazamiz. X=0,0625



Natija: $0,0625 = 0,0001_{(2)} = 0,04_{(8)} = 0,1_{(16)}$

Aralash kasr sonni bir sanoq tizimidan boshqa sanoq tizimiga o‘tkazish uchun uning butun qismi alohida va kasr qismi alohida yuqorida qayd etilgan qoidalarga asosan o‘tkaziladi va natijalar qo‘shib yoziladi.

Sonni sakkizlik sanoq tizimidan ikkilik sanoq tizimiga o‘tkazish uchun sakkizlik raqamlarini ikkilikdagi sonlar bilan almashtirib yozish mumkin, chunki sakkizlik sanoq tizimining asosi 2 darajasi 3 ga teng.Ushbu texnologiya o‘n oltilik sanoq tizimiga ham taaluqli. Bu yerda ko‘pincha triada va tetrada so‘zlari ishlatiladi. Triada deganda sakkizlik raqamini ifodalaydigan uchta ikkilik razryad tushuniladi. Tetrada deganda o‘nli raqamni ifodalaydigan to‘rtta ikkilik razryad tushuniladi. Bundan foydalanib, yuqorida keltirilgan sanoq tizimlari jadvali asosida juda osonlik bilan quyidagilarni bajarish mumkin

$11\ 0111\ 1100_{(2)} = 37C_{(16)}$

$110\ 1111\ 1001_{(2)} = 4F9_{(16)}$

$10\ 100\ 010_{(2)} = 242_{(8)}$

$110\ 101_{(2)} = 65_{(8)}$

Ikkilik sanoq tizimida arifmetik amallarni bajarish tartibi quyidagi jadvalda keltirilgan:

0+0=0	0+1=1	1+0=1	1+1=10
0·0=0	1·0=0	0·1=0	1·1=1

Sakkizlik sanoq tizimida qo‘shish arifmetik amalini bajarish tartibini esa quyidagi jadvalda keltirilgan:

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	10
2	3	4	5	6	7	10	11
3	4	5	6	7	10	11	12
4	5	6	7	10	11	12	13
5	6	7	10	11	12	13	14
6	7	10	11	12	13	14	15
7	10	11	12	13	14	15	16

Misol:

$$\begin{array}{r} 25_{(8)} \\ + \\ \underline{47_{(8)}} \\ 74 \\ {\scriptstyle (8)} \end{array}$$

Arifmetik amallarni bajarishni yengillashtirish maqsadida maxsus kodlar ham kiritilgan: teskari kod va qo‘shimcha kod. Teskari kod ikkilik sonda 0 ni 1 ga va 1 ni 0 ga almashtirish yo‘li bilan olinadi. Qo‘shimcha kod esa teskari kodga 1 ni qo‘shish bilan topiladi.

Misol : $01111_{(2)} \rightarrow$ (teskari) $10000_{(2)} + 1 \rightarrow$ (qo‘shimcha) $10001_{(2)}$.

Ko‘pgina amaliy masalalarda uchlik va to‘qqizlik sanoq tizimlaridan foydalanishga to‘g‘ri keladi. Shu bois ushbu tizimdagi o‘zaro bog‘liqlikni quyidagi jadvalda keltiramiz

O‘nlik	Uchlik	To‘qqizlik
0	00	0
1	01	1
2	02	2
3	10	3
4	11	4
5	12	5
6	20	6
7	21	7
8	22	8
9	100	10

2.3. Mantiqiy amallar

Kompyuter arifmetik amallardan tashqari mantiqiy amallarni ham bajarish imkoniga egadir. Dasturlashda bu amallarni ishini va ularning qo‘llanilishini bilish foydalidir. Ularning soni dasturlash tillariga bog‘liq bo‘lib, biz asosan quyidagilarni to‘laroq o‘rganib chiqamiz.

- **And** mantiqiy ko‘paytirish;
- **Not** mantiqiy inkor;
- **Or** mantiqiy qo‘shish;
- **Xor** mantiqiy qo‘shishni inkor etish.

And mantiqiy ko‘paytirish

Bu amal ikkita ifodani mantiqiy ko‘paytirish uchun ishlatiladi. Undan foydalanishning umumiy ko‘rinishi quyidagicha:

natija := ifoda1 And ifoda2

Bu yerda keltirilgan parametrlarning birgalikda bo‘lishi shart. **And** amalining ishlashi quyidagi jadvaldagi qonuniyatlarga amal qiladi:

Ifoda1	Ifoda2	Natija
True	True	True
True	False	False
False	True	False
False	False	False

Bu yerda “**True**” so‘zi “Rost” ma’nosini anglatsa, “**False**” esa “Yolg‘on” ma’nosini anglatadi.

Bu yerdan quyidagi ta'rifni berish mumkin. Mantiqiy ko'paytirish bu ikki ifoda bir paytda rost bo'lgandagina natija rost bo'ladigan amaldir.

And amali sonning bitlarini tekshirish uchun ham ishlatilishini unutmaslik lozim. Bitlar uchun **And** amali quyidagicha ishlatiladi:

1-bit	2-bit	Natija
0	0	0
0	1	0
1	0	0
1	1	1

And amaliga doir misollar quyida keltirilgan:

A	*B*	*C*	Mantiqiy ifoda	Natija
10	8	6	*A > B And B > C*	True
10	8	6	*B > A And B > C*	False
10	8	6	*A And B*	8

Birinchi misolni ko'rib chiqamiz. Bu yerda $A > B$ va $B > C$ tengsizliklar bajarilayapti, shu bois ularning qiymatlari **True** bo'ladi va o'z navbatida natija ham **True** bo'ladi. Xuddi shu yo'l bilan ikkinchi misol natijasi ham isbotlanadi. Lekin 3-misoldagi natijani aniqlash uchun *A* va *B* larni ikkilik sanoq tizimiga o'tkazamiz va yuqoridagi jadval bo'yicha har bir bit uchun amallarni bajaramiz

A	10	1010
B	8	1000
And		1000

Or mantiqiy qo'shish

Mazkur amal ikkita ifodani mantiqiy qo'shish uchun ishlatiladi. Undan foydalanishning umumiy ko'rinishi quyidagicha:

natija = ifoda1 Or ifoda2

Bu yerda keltirilgan parametrlarning birgalikda bo'lishi shart.

Or amalining ishlashi quyidagi jadvaldagi qonuniyatlarga amal qiladi:

Ifoda1	Ifoda2	Natija
True	True	True
True	False	True

False	True	True
False	False	False

Bu yerdan quyidagi ta'rifni berish mumkin. Mantiqiy qo'shish bu ikki ifodadan kamida bittasi rost bo'lganda natija rost bo'ladigan amaldir.

Or amali sonning ma'lum bir bitlarini o'rnatish uchun quyidagicha ishlatiladi:

1-bit	2-bit	Natija
0	0	0
0	1	1
1	0	1
1	1	1

Or operatoriga doir misollar quyida keltirilgan.

A	*B*	*C*	Mantiqiy ifoda	Natija
10	8	6	A > B Or B > C	True
10	8	6	B > A Or B > C	True
10	8	6	A Or 5	15

Birinchi misolni ko'rib chiqamiz. Bu yerda $A > B$ va $B > C$ tengsizliklar bajarilayapti, shu bois ularning qiymatlari **True** bo'ladi va o'z navbatida yuqoridagi jadvalga binoan natija ham **True** bo'ladi. Xuddi shu yo'l bilan ikkinchi misol natijasi ham isbotlanadi. Lekin 3-misoldagi natijani aniqlash uchun *A* va *B* larni ikkilik sanoq tizimiga o'tkazamiz va yuqoridagi jadval bo'yicha har bir bit uchun amallarni bajaramiz

A	10	1010
B	5	0101
Or		1111

Xor mantiqiy qo'shishni inkor etish

Bu amal ikkita ifodani mantiqiy inkor etish uchun ishlatiladi. Undan foydalanishning umumiy ko'rinishi quyidagicha bo'ladi:

natija = ifoda1 Xor ifoda2

Bu yerda keltirilgan parametrlarning birgalikda bo'lishi shart. ***Xor*** amalini qanday ishlashi quyidagi jadvalda keltirilgan:

Ifoda1	Ifoda2	Natija
True	True	False
True	False	True
False	True	True
False	False	False

Bu yerdan, mantiqiy qo'shishni inkor etish bu ikki ifodadan faqatgina bittasi rost bo'lsa, natija rost bo'ladigan amaldir.

Xor amali sonning ma'lum bir bitlarini teskarilashtirish uchun ham ishlatiladi. Bitlar uchun ***Xor*** operatori quyidagicha ishlatiladi:

1-bit	2-bit	Natija
0	0	0
0	1	1
1	0	1
1	1	0

Xor operatori ***Or*** dan shu bilan farq qiladiki, unda ikkala bit ham bir bo'lsa, ***Xor*** operatori ***0*** natijani beradi. ***Xor*** operatori yana shu bilan qiziqarliki, agar u ikki marta qo'llanilsa, berilgan sonni natija sifatida chiqaradi. ***Xor*** operatoriga doir misollar quyida keltirilgan:

A	*B*	*C*	Mantiqiy ifoda	Natija
10	8	6	A > B Xor B > C	False
10	8	6	B > A Xor B > C	True
10	8	6	B > A Xor C > B	False
10	8	6	A Xor B	2

Bu yerda ham keltirilgan misollar deyarli tushunarli, faqatgina oxirgi misolni batafsil ko'rib chiqamiz

A	10	1010
B	8	1000
Xor		0010

Ikkita sonli o'zgaruvchilarning qiymatlarini almashtirishda ***Xor*** amalining qo'llanilishi misoli juda qiziqarli hisoblanadi:

A:=4
B:=7
A:=A Xor B

B:=A Xor B
A:=A Xor B

Bu dasturning bajarilishi natijasida *A* o'zgaruvchi *B* o'zgaruvchining qiymatiga ega bo'ladi va aksincha, ya'ni *A*=7 va *B*=4 .

Not mantiqiy inkor

Ifodani mantiqiy invertlash uchun ishlatiladi. Undan foydalanishning umumiy ko'rinishi quyidagicha bo'ladi:

natija = Not ifoda

Not amalining ishlashi quyidagi jadvalda keltirilgan qonuniyatlarga amal qiladi:

Ifoda	Natija
True	False
False	True

Demak, mantiqiy inkor bu berilgan ifoda rost bo'lganda yolg'on, yoki ifoda yolg'on bo'lganda natija rost bo'ladigan amaldir.

Not operatori «ifoda»ning barcha bitlarini teskarilashtiradi, ya'ni 0 ni 1 ga va 1 ni 0 ga aylantiradi.

Not operatoriga doir misollar quyida keltirilgan:

A	*B*	*C*	Mantiqiy ifoda	Natija
10	8	6	*Not (A > B)*	False
10	8	6	*Not (B > A)*	True

Shunday qilib, barcha mantiqiy amallarni yagona jadvalda joylashtiramiz.

A	*B*	*A OR B*	*A AND B*	*A XOR B*	*NOT A*
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

Qoldiq. N soni k-sanoq tizimida berilgan, ya'ni

$a_n a_{n-1}, \ldots, a_0 = N_{(k)}$

Ushbu sonni m ga bo'lishdan hosil bo'lgan qoldiqni aniqlang. Bu yerda n, k, m va natija o'nlik sanoq tizimida beriladi.

Masala yechimi. Albatta ushbu masalani to'g'ridan-to'g'ri yechish ham mumkin, ya'ni N sonini o'nlik sanoq tizimiga o'tkazish va so'ng qoldiqni aniqlash. Lekin boshqa algoritm ham mavjud ekan va uni katta sonlar uchun ham qo'llash mumkin bo'ladi.

k-chi sanoq tizimida sonni quyidagi shaklda tasvirlash mumkin

$$a[n]*K^n + a[n-1]*K^{n-1} + \ldots + a[0]*K^0.$$

Ushbu yig'indini m ga bo'lganda hosil bo'ladigan qoldiqni quyidagicha hisoblay-miz (a ni b ga bo'lganda hosil bo'ladigan qoldiq matematikada quyidagicha belgilanadi $a \bmod b$)

$$(a[n]*K^n + a[n-1]*K^{n-1} + \ldots + a[0]*K^0) \bmod m =$$

$$\left[\sum_{i=0}^{n} a[i]*K^i\right] \bmod m = \left[\sum_{i=0}^{n} a[i]*(K^i \bmod m)\right] \bmod m$$

Ushbu tenglikni isbotlashda quyidagilarni e'tiborga olish kerak, ya'ni, agar $K^i \bmod m = t$ bo'lsa , u holda $K^i = p*m+t$, va

$(a[i]*K^i) \bmod m = (a[i]*(p*m+t)) \bmod m = = (a[i]*p*m) \bmod m + (a[i]*t) \bmod m == (a[i]*(K^i \bmod m)) \bmod m$

Bu yerda $(a[i]*p*m) \bmod m = 0$ ekanligi e'tiborga olingan va istalgan $b[i]$ lar uchun quyidagi tenglik bajariladi:

$$\left(\sum_{i=0}^{n} b[i]\right) \bmod m = \left(\sum_{i=0}^{n} b[i] \bmod m\right) \bmod m$$

Ushbu tenglikni isbot qilishda $b[i]=p_i m+t_i$ deb qabul qilinsa, qiyinchilik tug'dirilmaydi. $K^i \bmod m$ ni hisoblashda ham quyidagi formuladan foydalanamiz

$K^i \bmod m = [(K^{i-1} \bmod m) * K] \bmod m$,

chunki, agar $K^{i-1} = p*m+t$ bo'lsa, u holda $K^{i-1} \bmod m = t$, demak $K^i = p*m*K+t*K$ va $K^i \bmod m = t*K \bmod m = [(K^{i-1} \bmod m)*K] \bmod m$.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
4 10 15 1 5 0 1 3	13

Bu yerda *input.txt* faylida birinchi qatorda *n, k, m* va ikkinchi qatorda massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida qoldiq qiymati chiqarilgan bo'ladi.
Shundan kelib chiqqan holda quyidagi dasturni tuzamiz:

```
program Qoldiq;
const Nmax=20;
var {o'zgaruvchilarni e'lon qilamiz}
      i, n, k, m, s, t : integer;
      fayl : text;
      A: array[1..Nmax] of integer; {boshlang'ich massiv}
begin
      {fayldan  qiymatlarni kiritish}
            assign(fayl, 'input.txt');
            reset(fayl); readln(fayl, n, k, m);
            for i:=n downto 0 do
                  read(fayl, A[i]);
      {-teskari sikl orqali A[] qiymatlarini kiritamiz}
                  close(fayl);
      s:=0; t:=1;
for i:=0 to n do
begin
s:=(s+a[i]*t) mod m;
t:=(t*K) mod m;
end;
      {chiqarish faylini ochamiz}
       assign(fayl, 'output.txt');
       rewrite(fayl);
            Writeln(fayl, s); {natijani chop etish}
       close(fayl);
end. {dastur tugatildi}
```

Izlangan natija *S* o'zgaruvchida saqlangan bo'ladi.

Birlar soni. Berilgan natural *N* sonining ikkilik sanoq tizimidagi yozuvida nechta bir borligini aniqlang.

Masala yechimi. Bu yerda mantiqiy ko'paytirish amalidan foydalanamiz va tushunarli bo'lish uchun, masalan *N*=6 deb qabul

qilamiz. N ning ikkilikdagi qiymati $110_{(2)}$ ga teng, ya'ni $N=110_{(2)}$. Demak birlar soni 2 ta.
Endi quyidagi amalni bajaramiz *(N and (N-1))* , bu yerda $N-1 = 101_{(2)}$.

N	6	110
N-1	5	101
And		100

ya'ni N ***and*** $(N-1)=100_{(2)}$. O'z-o'zidan ko'rinib turibdiki, bitta 1 ga kamaytirildi. Ushbu jarayonni yana bir bor takrorlaymiz, ya'ni endi $N=100_{(2)}$ va $N-1 = 001_{(2)}$ demak N ***and*** $(N-1)=000_{(2)}$. Shunday qilib, har safar bir raqami yo'qotilib, oxirida nolga tenglashdik.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
15	4

Bu yerda *input.txt* faylida birinchi qatorda n soni berilgan va va *output.txt* faylida natija chiqarilgan bo'ladi.

Endi dasturni qiyinchiliksiz tuzamiz:

```
program Birlar;
var  {o'zgaruvchilarni e'lon qilamiz}
       n, count : integer;
       fayl : text;
begin
   assign(fayl, 'input.txt');
       reset(fayl);
       readln(fayl, n);
       close(fayl);
count:=0;
while (n<>0) do
begin
n:=(n-1) and n;
count := count +1;
end;
{chiqarish faylini ochamiz}
       assign(fayl, 'output.txt');
       rewrite(fayl);
       Writeln(fayl, count); {natijani chop etish}
        close(fayl);
```

end. {dastur tugatildi}

Ikkilik sanoq tizimida *M*+1. Natural *M* soni ikkilik sanoq tizimida yozilgan

$$M = a_1 a_2 \ldots . a_n$$

bu yerda a_i=0 yoki 1. Ikkilik sanoq tizimida *M*+1 raqamlarini aniqlang.

Masala yechimi. Masalani yaxshi tushinib olish uchun quyidagi misolni ko'rib chiqamiz:

10111+1=11000

E'tibor bering, oxiridan kelgan ketma-ket 1 natijada 0 ga aylanib, birinchi uchragan 0 esa 1 ga aylandi. Faqatgina 11111 ga 1 ni qo'shganda, natija 6 ta raqamdan iborat bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 1 0 1 1 1	11000

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va ikkinchi qatorda massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida natijaviy *M*+1 raqamlari chiqarilgan bo'ladi.

Dasturimiz esa quyidagi ko'rinishda bo'ladi.

```
program IkkilikSon;
const Nmax=100;
var
      n, i, j : integer;
      b: boolean;
      A:array[1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl,n);
      for i:=1 to n do
      read(fayl, A[i]);
```

```
      {sikl orqali A[] qiymatlarini kiritamiz}
      close(fayl);
      {chiqarish faylini ochamiz}
       assign(fayl, 'output.txt');
       rewrite(fayl);
b:=True;
for i:=n downto 1 do
  begin
      j:=A[i];
      if b then A[i]:=1-j else A[i]:=j;
      if j=0 then b:=FALSE;
  end;
if b then writeln (fayl, ' 1');
for i:=1 to n do  write(fayl, A[i]);
close(fayl);
end.
```

Bo'linish alomati. Ikkilik natural N sonining 15 ga bo'linishini aniqlang.

Masala yechimi. Bu yerda algebradan bizga ma'lum bo'lgan 9 ga bo'linish alomatini keltiramiz. Agar sondagi raqamlar yig'indisi 9 ga bo'linsa, u holda ushbu son ham 9 ga bo'linadi. Shunga asoslangan holda o'n oltilik sanoq tizimidagi sonning raqamlari yig'indisi 15 ga bo'linsa, ushbu son ham 15 ga bo'linadi. Haqiqatan ham istalgan M uchun

$$M_{(16)} = a[n]*16^n + a[n-1]*16^{n-1} + \ldots + a[1]*16 + a[0]=$$
$$= a[n]*(16^n-1)+a[n] + a[n-1]*(16^{n-1}-1)+a[n-1] + \ldots + a[1]*(16-1)+a[1] + a[0]=$$
$$= (a[n]*(16^n-1)+ a[n-1]*(16^{n-1}-1)+ \ldots + a[1]*(16-1))+ (a[n] + a[n-1] +\ldots+a[1] + a[0]).$$

Bu yerda 16^k - 1 bevosita 15 ga bo'linadi, demak $a[n] + a[n-1] +\ldots+a[1] + a[0]$ yig'indi ham 15 ga bo'linsa, unda M ham 15 ga bo'linadi.

Shundan kelib chiqqan holda ikkilik N sonini tetradalar yordamida o'n oltilik raqamlarga aylantiramiz va bevosita yig'indini hisoblaymiz.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
1111	YES

Bu yerda *input.txt* faylida faqatgina ikkilik son kiritiladi va *output.txt* faylida natijaviy **YES** yoki **NO** so'zlari chiqarilgan bo'ladi.

Dasturimiz esa quyidagi ko'rinishda bo'ladi.

```
program IkkilikDivide;
const
      v : array[0..3] of longint = (1,8,4,2);
var
      n:string;
      s, i, L, nol : longint;
      fayl : text;
begin
{ikkilik sonni  kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
   readln(fayl,n);
      close(fayl);
 s:=0;
 L := length(n);
 for i := 1 to 4 - (L mod 4) do n := '0' + n;
 L := length(n);
 nol := ord('0');
 for i:=1 to L do s:=s + v[i mod 4]*(ord(n[i])-nol);
{chiqarish faylini ochamiz}
       assign(fayl, 'output.txt');
       rewrite(fayl);
if (s mod 15 = 0) then writeln(fayl, 'YES')
else writeln(fayl, 'NO');
      close(fayl);
end.
```

Topshiriqlar

1. Quyidagi ikkilik $100000_{(2)}$, $11111111_{(2)}$ sonlarni o'noltilik sanoq tizimiga o'tkazing
2. Quyidagi o'noltilik $ABCD_{(16)}$, $3FFFF_{(16)}$ sonlarni ikkilik sanoq tizimiga o'tkazing
3. Ikkilik sanoq tizimida yozilgan sonni juft yoki toq son ekanligini aniqlaydigan dastur tuzing.
4. Quyidagi to'qqizlik $888_{(9)}$, $1357_{(9)}$ sonlarni uchlik sanoq tizimiga o'tkazing?
5. Quyidagi sakkizlik $417_{(8)}$, $512_{(8)}$ sonlarni o'noltilik sanoq tizimiga o'tkazing
6. Ushbu ketma-ketlikdagi 110, 20, 12, 11, 10 sonlarini nima birlashtiradi?
7. Qaysi sanoq tizimida quyidagi tengliklar bajarilishini aniqlang:

   21+24=100 20+25 = 100 va 22+44 = 110
8. Ikkilik natural N sonining 7 ga bo'linishini aniqlaydigan dastur tuzing.
9. Agar N natural soni x asosli sanoq tizimiga mansub bo'lsa va M o'nlik sanoq tizimidagi son bo'lsa, unda quyidagi $N_{(x)}=M$ tenglamani yechadigan dastur tuzing.
10. Berilgan natural N sonigacha bo'lgan barcha sonlar ikkilik sanoq tizimida yozilganda, ularda faqatgina K ta nollar bo'lganlarini sonini aniqlaydigan dastur tuzing. Masalan, N=8 va K=1 uchun quyidagi ikkilik sonlar ichida 1, 10, 11, 100, 101, 110, 111 va 1000, faqatgina uchtasida 10, 101 va 110 bittadan nol mavjud.
11. Ikkilik sanoq sistemasida berilgan nomanfiy N ratsional sonini (sonda kasr qismi va davr qatnashishi mumkin va sonning kasr qismi nuqta orqali ajratiladi va ko'pi bilan 4 raqam bo'lishi mumkin) sakkizlik sanoq sistemasiga o'tkazuvchi dastur tuzing. Agar natijada davr qatnashsa, u qavslar orasida ko'rsatilishi shart. Masalan, N=101.10(1) uchun natija 5.5(7) bo'ladi.
12. Rim sanoq tizimida berilgan quyidagi tengliklarni faqatgina bitta cho'pchani o'rnini almashtirish orqali to'g'rilab qo'ying:

   1) VII – V = XI 2) IX – V = VI

   3) VI + I = III 4) VIII – III = X

13. Quyidagi berilgan mulohazalar *A*="8 bit 1 baytga teng" va *B*="1024 bayt 1 Kb ga teng" qanday qiymatga ega va ular ustidagi quyidagi amallarning natijasini aniqlang:

1) *A AND B*
2) *A OR B*
3) *A OR (Not B)*
4) *A AND (Not B)*
5) *(Not A) OR (Not B)*

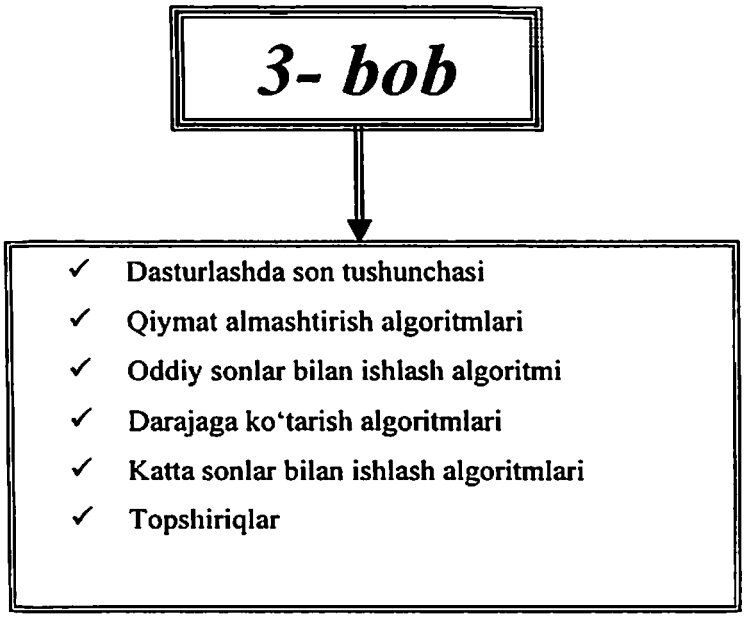
14. Aka-ukalar Iskandar, Jahongir, Baxodirlar kelgusida soatsoz, iqtisodchi va o'qituvchi kasblarini egallashmoqchilar. Kelgusida qaysi kasblarni egallashni istaysizlar degan savolga, aka-ukalardan biri quyidagicha javob qaytaribdi: «Iskandar soatsoz bo'lishni istaydi, Jahongir soatsoz bo'lishni istamaydi, Baxodir o'qituvchi bo'lishni istamaydi». Keyinchalik ma'lum bo'lishicha, ushbu javobda faqatgina bitta mulohaza to'g'ri bo'lib, qolgan ikkitasi noto'g'ri ekan. Aka-ukalar qaysi kasblarni egallashmoqchi ekanligini aniqlang.

15. Berilgan uchta tillo tangalardan bittasi qalbaki ekan. Qalbaki tanganing boshqa tangalarga nisbatan og'ir yoki engilligi noma'lum ekan. Oddiy ikki pallali tarozi yordamida qalbaki tangani aniqlang.

3-bob. SONLAR USTIDA TURLI AMALLAR BAJARISH DASTURLARI

Kompyuterda butun sonlarni tasvirlashda chegara mavjud, masalan **integer** tipli sonlar uchun chegara -32768 dan +32767 gacha. Lekin xattoki 10!= 3628800 ushbu chegaradan chiqib ketadi, shu bois katta sonlar bilan ishlashni o'rganish, ya'ni ularni kompyuterda tasvirlash va ular ustida arifmetik amallarni bajarishni o'rganib olish zaruriryati paydo bo'ladi.

Mazkur bobda sonlarga doir misollar ko'rib chiqilgan bo'lib bevosita katta sonlar ustida amallarni bajarish algoritmlari ham keltirilgan



3.1. Dasturlashda son tushunchasi

Kompyuterda sonlar uchun xotiradan har xil xajmdagi joy ajratiladi. Shu bois sonlar uchun chegara mavjud bo'ladi. Masalan, matematikada butun sonlar $-\infty,\ldots,-2,-1,0,1,2,\ldots,+\infty$ shu chegarada bo'ladi. Lekin kompyuterda cheksiz sonlar bilan ishlab bo'lmaydi. Agar butun sonlar uchun kompyuterda 2 bayt joy ajratilgan bo'lsa, unda -32768 dan +32767 gacha sonlarni tasvirlash mumkin. Quyidagi jadvalda barcha sonlar chegaralari keltirilgan.

Son tipi	Saqlaydigan ma'lumot	Xotira hajmi (bayt)	Qabul qiladigan qiymat chegarasi
Integer	Butun son	2	-32768 dan 32767 gacha
Long	Uzun butun son	4	Taxminan +\- 2.1E9
Float	Haqiqiy son	4	-3.4E38 dan -1.4E-45 gacha manfiy sonlar uchun va 1.4E-45 dan 3.4E38 gacha musbat sonlar uchun
Double	Haqiqiy son (ikki karra aniqlikda)	8	-1.8E308 dan -4.9E-324 gacha manfiy sonlar uchun va 4.9E-324 dan 1.8E308 gacha musbat sonlar uchun
Byte	Butun son	1	0 dan 255 gacha

Natural sonlar deb birdan boshlangan butun sonlarga deyiladi, ya'ni $1,2,\ldots,+\infty$. Har qanday a va $b\neq 0$ butun sonlar uchun $c=a/b$ ratsional son deyiladi. Ratsional sonlar ustida arifmetik amalarni bajarish deyarli qiyinchilik tug'dirmaydi. Masalan $c=x_1/y_1$ va $d=x_2/y_2$ ratsional sonlar ustida arifmetik amallarning soni va uni bajarish quyidagicha bo'ladi:

1) c va d sonlarini qo'shishda amallar soniga e'tibor bering (birinchisida 3 ta va ikkinchisida 5 ta):

$$c+d=\frac{x_1}{y_1}+\frac{x_2}{y_2}=\frac{x_1y_2+x_2y_1}{y_1y_2}$$

2) c va d sonlarini ayirish (3 ta va 5 ta):

$$c-d=\frac{x_1}{y_1}-\frac{x_2}{y_2}=\frac{x_1y_2-x_2y_1}{y_1y_2}$$

3) c va d sonlarini ko'paytirish (3 ta):

$$c \cdot d = \frac{x_1 x_2}{y_1 y_2}$$

4) c va d sonlarini bo'lish (3 ta, 3ta, 3 ta):

$$c : d = \frac{x_1}{y_1} : \frac{x_2}{y_2} = \frac{x_1}{y_1} \cdot \frac{y_2}{x_2} = \frac{x_1 y_2}{y_1 x_2}$$

Lekin kompyuterda ratsional sonlar oddiy kasr sifatida tasvirlanmaydi. Ular o'nlik kasr sonlar bilan yoziladi. Masalan, 1/3 =0,3333..., 1/7=0,1428....

Matematikada juda ko'p sonlar haqiqiy sonlarning irratsional sonlar to'plamidir. Ratsional son sifatida tasvirlash mumkin bo'lmaydigan barcha sonlar irratsional sonlar deyiladi. Misol sifatida quyidagi sonlarni keltirish mumkin, masalan. π=3,1415..., e=2,7182.... Ushbu sonlarni a/b shaklida tasvirlash mumkin emas. Lekin kompyuterlarda ushbu sonlarni to'liq tasvirlash mumkin emas, ya'ni verguldan keyin bir necha sonlar qoldirib, qolganlari tashlab ketiladi. Bu esa ko'pgina muammolarni keltirib chiqarishi mumkin. Masalan, matematikada har qanday ikki a va b sonlari uchun, agar $a < b$ bo'lsa, u holda shunday c soni mavjudki, uning uchun $a < c < b$ tengsizlik bajariladi. Kompyuterlarda esa bu tengsizlik bajarilmay qolishi mumkin, shu bois haqiqiy sonlar bilan ishlaganda ularni qaysi aniqlikda tasvirlash ko'riladigan masalaning shartidan kelib chiqadi.

Shu bois haqiqiy sonlarni yaxlitlash yoki kesib tashlash texnologiyasi qo'llaniladi. Ularning farqini anglash uchun π sonini ko'rib chiqamiz:

$\pi = 3{,}1415926...$

$\pi \approx 3{,}1415$ - kesilgan

$\pi \approx 3{,}1416$ - yaxlitlangan

Haqiqiy sonlar kompyuter xotirasida eksponensial yozuv shaklida yoziladi, shu bois, masalan, a va b haqiqiy sonlarning o'zaro tengligini ($a = b$) hech qachon tekshirish mumkin emas. Buning o'rniga (a-ε, a+ε) intervalga tegishligini aniqlash maqsadga muvofiq, ya'ni

$$a-\varepsilon \le b \le a+\varepsilon$$

bu yerda ε - juda kichik son.

Shunday qilib, haqiqiy sonlar bilan ishlashda ularni yuqori aniqlikda hisoblashda muammolar paydo bo'lishi mumkin, bundan tashqari hisoblanadigan sonlar uchun kompyuterda ajratilgan xotiradagi joy kamlik qilishi mumkin.

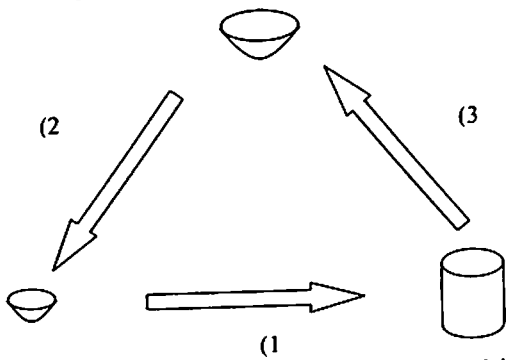
3.2. Qiymat almashtirish algoritmlarini dasturlash

Amaliyotda juda ko'p uchraydigan qiymatlarni o'zaro almashtirish amalida qanday muammolar bo'lishi mumkin? – degan savol tug'iladi. Aniqlik kiritish uchun quyidagi misolni ko'rib chiqamiz.

Almashtirish. Agar a va b o'zgaruvchilarda butun sonlar kiritilgan bo'lsa, ularni qiymatlarini almashtiring.

Masala yechimi. Demak, misolning shartiga binoan a va b larda qiymat kiritilgan, masalan, $a = 10$ va $b = 50$ bo'lsa, natija $a = 50$ va $b = 10$ bo'lishi kerak.

Misolni yechimini hayotda uchraydigan vaziyatlar orqali tushunib olish mumkin. Masalan, piyola va kosada suv solingan bo'lsa, ularni o'zaro almashtirish uchun biz yana bitta idish olamiz va piyoladagi suvni ushbu idishga solamiz (1). Piyolaga esa kosadagi suvni solamiz (2) va yangi idishdagi suvni kosaga solib qo'yamiz (3). Ushbu jarayonni sxematik ravishda quyidagicha tasvirlash mumkin



Shundan kelib chiqqan holda bu yerda biz qo'shimcha yangi o'zgaruvchi - *temp* ni kiritamiz va quyidagi algoritmni tavsiya qilamiz

temp:=a
a:=b
b:= temp

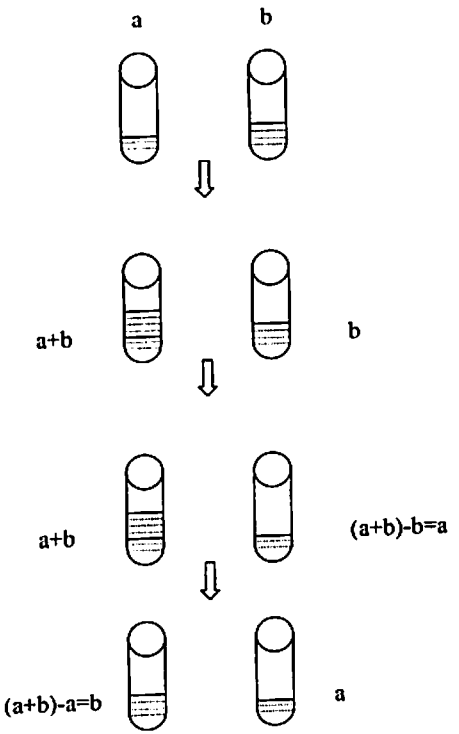
Ushbu g'oyaning to'liq algoritmini quyidagicha dastur shaklida yozamiz

```
program Swap;
      var a, b, temp : integer;
      begin
   readln(a,b);
         temp:=a;
         a:=b;
         b:= temp;
   write(a,b)
end.
```

Bu yerda qo‘shimcha *temp* o‘zgaruvchisi orqali biz maqsadga erishdik. Lekin *a* va *b* o‘zgaruvchilardagi qiymatlar kichik sonlar bo‘lib, ularning yig‘indisi va ayirmasi dasturlash tillarida kelishilgan chegaradan chiqib ketmasa, unda quyidagi algoritmni tavsiya etish mumkin:

```
a:=a+b;
b:=a-b;
a:=a-b;
```

Ushbu algoritm quyidagi rasmda qulay va tushunarli tasvirlangan:



```
program SwapNew;
var
      a, b  : integer;
begin
      readln(a,b);
      a:=a+b;
      b:=a-b;
      a:=a-b;
write(a,b)
end.
```

Mantiqiy operatsiyalar yordamida ham qiymatlar almashtirish mumkin, ya'ni ikkita sonli o'zgaruvchilarning qiymatlarini almashtirishda **Xor** operatorining qo'llanilishi misoli juda qiziqarli hisoblanadi:

```
program SwapXor;
var
      a, b  : integer;
begin
      readln(a,b);
      a:= a xor b;
      b:= a xor b;
      a:= a xor b;
 write(a,b)
end.
```

Shunday qilib ushbu dasturlarning bajarilishi natijasida *a* o'zgaruvchi *b* o'zgaruvchining qiymatiga ega bo'ladi va aksincha.

3.3. Oddiy sonlar bilan ishlash algoritm va dasturlari

Sonlar bilan ishlashni o'rganish maqsadida bevosita oddiy misollardan boshlaymiz.

Ikki natural son ko'paytmasi. Berilgan n va m natural sonlarni ko'paytiring va bunda faqatgina +, -, =, <> amallaridan foydalanish mumkin.

Masala yechimi. Masalan, 5·4 ni hisoblash kerak bo'lsa, biz bu erda arifmetikada qabul qilingan qoidaga asoslanamiz, ya'ni 5·4 = 5+5+5+5. Keltirilgan dastur shu g'oya asosida tuzilgan.

```
program Multiply;
var
    n, m, k, c: integer;
begin
    write ('n,m=');
    readln(n,m);
    k := 0; c := 0;    {natijani  c da saqlaymiz}
while k <> m do
begin
    k := k + 1;
    c := c + n;
end;
{ya'ni har doim c = n * k bo'ladi va oxirida  k = m,}
{demak c = n * m}
writeln(c);
end.
```

Kub tenglama. Berilgan N natural sonini necha variantda $N=i^3+j^3$ yig'indi shaklida yozish mumkin. Faqatgina i va j larning o'rin almashishi bilan yangi variant hosil qilinmaydi va 1/3 darajaga ko'tarish mumkin emas.

Masala yechimi. Ushbu masalaning qiziqarli tomoni shundaki, oddiy usul bilan, ya'ni i , j larning qiymatini 1 dan boshlab oshirib , barcha variantlarni aniqlash mumkin. Lekin ushbu jarayon taxminan N^3 marotaba amallarni bajarishni talab qiladi. Katta N soni uchun bu ko'p vaqtni talab qiladi.

Endi asosiy algoritmga o'tamiz. Oldiniga eng katta N ga yaqin bo'lgan j ni aniqlaymiz, ya'ni

$$j^3 + 1 \geq N$$

Variantlar sonini M da saqlaymiz, i ning qiymatini 1 dan boshlab j gacha davom ettiramiz, ya'ni $i \leq j$. Quyidagi qiymatni hisoblab $K=i^3+j^3$ uni N bilan taqqoslaymiz.

Endi bizda uchta holat mavjud bo'lishi mumkin:

1) Agar $K<N$ bo'lsa, i ning qiymatini oshiramiz, ya'ni $i=i+1$;
2) Agar $K>N$ bo'lsa, j ning qiymatini kamaytiramiz, ya'ni $j=j-1$;
3) Agar $K=N$ bo'lsa, demak bu to'gri variant, shu bois $M=M+1, j=j-1$ va $i=i+1$.

Shu yo'l bilan topilgan barcha variantlar sonini taxminan $N^{1/3}$ qadamda aniqlash mumkin bo'ladi.

```
program KubTenglama;
var
    m,n,i,j,k : integer;
begin
    write ('N=');read(n);
    m:=0; i:=1; j:=1;
while j*j*j+1<n do j:=j+1;
repeat
    k:= i*i*i+ j*j*j;
    if k=n then m:=m+1;
    if k<=n then i:=i+1;
    if k>=n then j:=j-1;
until i>j;
writeln (m)
end.
```

Masalan, quyidagi jadvalda ba'zi-bir N qiymatlari uchun keltirilgan natijalarni tekshirib olish mumkin:

N	Variantlar soni	i va j qiymatlari
9	1	i=1 va j=2
13	0	yo'q
35	1	i=2 va j=3
407	1	i=4 va j=7

Tengsizlikning yechimlar soni. Berilgan n natural soni uchun $x^2 + y^2 < n$ tengsizlikning natural yechimlar sonini aniqlang va bunda haqiqiy sonlar bilan amallar bajarilmasin.

Masala yechimi. Masalan, $n=3$ bo'lganda yechim sifatida quyidagilar qabul qilinadi: 1) $x=1$ va $y=1$. Demak, yechimlar soni bitta bo'ladi,

ya'ni $s=1$. Quyidagi jadvalda har xil n uchun yechimlar sonlari keltirilgan:

N	3	5	7	13	21	36	50
S	1	1	3	6	13	22	30

Asosiy g'oya bu bevosita x va y qiymatlarini birin-ketin oshirib borishda bo'lib, agar tengsizlik buzilsa, u holda jarayon to'xtatiladi va natija chop etiladi. N=13 uchun natijalar quyidagicha bo'ladi:

x=1 va y=1
x=1 va y=2
x=1 va y=3
x=2 va y=1
x=2 va y=2
x=3 va y=1

Asosiy dastur esa quyidagicha bo'ladi:

```
program Tengsizlik;
var
      n, L, t, k, s: integer;
begin
      write ('n=');
      readln(n);
      k:=1; s:= 0;
{ x*x + y*y < n  tengsizligida  x < k bo'lganda  } {yechimlar sonini s
da saqlaymiz}
while k*k < n do
begin
      L:= 1; t:= 0;
{ k*k + y*y < n  tengsizligida  0<=y<l  bo'lganda  } {yechimlar
sonini t da saqlaymiz}
while k*k + L*L < n do
begin
      L:= L + 1;
      t:= t + 1;
 end;
{bu yerda k*k + L*L >= n bo'ladi,  demak }
{k*k + y*y < n  ning yechimlar soni t ga teng bo'ladi}
      k:= k + 1;
      s:= s + t;
end;
```

```
{bu yerda k*k >= n bo'ladi, demak s barcha yechimlar } {soniga
teng bo'ladi}
writeln(s);
end.
```

Sonni siklik teskarilash. Berilgan n natural sonning oxirgi raqamini birinchi raqam o'rniga joylashtiring.

Masala yechimi. Masalaning sharti bo'yicha , agar n=345 bo'lsa, 534 sonini tashkil qilish kerak bo'ladi. Buning uchun quyidagi algoritm tavsiya etiladi:

1. n sonidagi *dig* raqamlar sonini aniqlash, masalan n=345 uchun *dig*=3
2. eng kichik raqamini kesib olamiz, ya'ni *raqam*=5 va *temp*=34
3. ajratib olingan eng kichik raqam (*raqam*=5) ni bevosita 10^{dig-1} ga ko'paytirib razryadini oshiramiz, ya'ni $5 \cdot 10^2$=500
4. olingan natijaga kesib olingan *temp* sonini qo'shamiz, ya'ni 500+34=534.

Ushbu algoritmning dasturi quyidagicha bo'ladi:

```
program DigitMy;
var
      n,i, res,temp,dm: integer;
      raqam, dig: byte;
begin
write ('n=');
 readln(n);
        temp:= n;
        dig := 0;
while  temp > 0 do
begin
        temp := temp div 10;
        dig :=dig + 1;
end;
        temp := n div 10;
        raqam:= n mod 10;
        res:=1;
{bu siklda 10 ning (diq-1) –darajasini hisoblaymiz}
        for i:=1 to dig-1 do
        res:=res*10;
```

```
    dm := raqam * res + temp;
writeln(dm);
end.
```

Tub son. *N* natural sonidan katta bo'lmagan barcha tub sonlarni chiqaring.

Masala yechimi. Tub son - bu natural bo'luvchilari atigi 1 va *p* sonlari bo'lgan har qanday $p>1$ natural son.

Quyida ba'zi-bir tub sonlarga misollar keltirilgan: 2, 3, 5, 7 va h.k.

Shu ta'rifdan kelib chiqqan holda biz faqatgina toq sonlarni ko'rib chiqishimiz yetarli bo'ladi. Bundan tashqari har qanday sonning bo'luvchisi $\sqrt{n}$ gacha aniqlanishi kifoya bo'ladi.

Chunki har qanday natural sonning bo'luvchisi $\sqrt{n}$ gacha mavjud bo'lmasa, uning boshqa bo'luvchisi bo'lmaydi. Haqiqatan ham, faraz qilaylik *n* ning bo'luvchisi *l* uchun $l>\sqrt{n}$

Demak n/l ham natural son bo'ladi, ya'ni $j=n/l$ (*) va $j<\sqrt{n}$. Aksincha, agar $j\geq\sqrt{n}$ bo'lsa, u holda $l\cdot j>\sqrt{n}\cdot\sqrt{n}$, ya'ni $l\cdot j>n$, bu esa (*) bilan ziddiyatga olib keladi. Demak, $j<=\sqrt{n}$, ya'ni har qanday natural sonning $\sqrt{n}$ gacha bo'luvchisi bo'lmasa, uning boshqa bo'luvchisi bo'lmaydi.

Ushbu jarayonni quyidagi algoritm asosida amalga oshirish mumkin bo'ladi:

```
Bolinadi := true;
for i := 2 to trunc(sqrt(x)) do
if x mod i = 0 then Bolinadi := false;
```

Lekin ushbu siklda kamchilik mavjud, ya'ni $\sqrt{n}$ gacha bo'lgan barcha sonlar bo'luvchi ekanligi to'xtovsiz tekshiriladi. Undan tashqari, doimo juft sonlarga bo'lish shart emas. Shulardan kelib chiqqan holda quyidagi algoritmni tavsiya etish mumkin:

```
    Bolinadi := true;
    divisor.: = 2;
    while (divisor <= trunc(sqrt(x))) and Bolinadi do
begin
    if x mod divisor = 0 then
    Bolinadi := false;
    divisor := divisor + 1;
end;
```

Bu yerdagi kamchilik bevosita **while & repeat** sikl operatori bilan bog'liq, unga binoan **trunc(sqrt(x))** doimo hisoblanadi, bu esa ko'p vaqtni talab qiladi. Demak ushbu qiymatni siklgacha hisoblab qo'yish zarur.

Agar massiv elementlaridan foydalansak, unda quyidagicha yo'l tutish mumkin, ya'ni n gacha bo'lgan sonlarni tub son ekanligini aniqlash bilan birgalikda, ularni maxsus massivda saqlab turamiz. Shu orqali tub songa tekshirish shu massiv elementlaridagi sonlar bilan taqqoslanib aniqlanadi.

Misol sifatida n=14 bo'lsa, quyidagi ishlarni amalga oshirish zarur bo'ladi. Boshlang'ich tub sonlar to'plamiga 2 sonini kiritib qo'yamiz, ya'ni r={2}.

i	Tub sonlar to'plami	Natija tub son	Kengaytirilgan tub sonlar to'plami
3	2	+	2,3
4	2,3	-	2,3
5	2,3	+	2,3,5
6	2,3,5	-	2,3,5
7	2,3,5	+	2,3,5,7
8	2,3,5,7	-	2,3,5,7
9	2,3,5,7	-	2,3,5,7
10	2,3,5,7	-	2,3,5,7
11	2,3,5,7	+	2,3,5,7,11

Ushbu algoritm dasturi quyidagicha bo'ladi:

```
program Tub;
const m=200;
label BR,NP;
var
      n,i, j,k,q : integer;
      P:array[1..m] of  integer;
begin
      write ('n=');readln(n);
      if (n>=2) then writeln (2);
      if (n>=3) then writeln (3);
      k:=1; P[k]:=3; i:=5;
while i<n  do
begin
      for j:=1 to k  do
begin
      q:=P[j];
      if (q*q>i) then goto BR;
      if (i mod q=0) then goto NP;
end;
      if (k=m) then
begin
            i:=n-1;
{ Massiv chegarasidan chiqib qolamiz ! }
{ Hisoblashni tugallaymiz}
End;
BR:
begin
      writeln(i);
      if (k<=m-1) then
begin
      k:=k+1;
      P[k]:=i;
end
end;
      NP: i:=i+2
end
end.
```

Tub bo'luvchilar. Natural n sonining barcha tub bo'luvchilarini toping.

Masala yechimi. Quyida ba'zi-bir natural sonlarning tub bo'luvchilari keltirilgan:

N	6	15	51
tub bo'luvchilar	2;3	3;5	3;17

Har qanday n natural son tub bo'luvchilari p_i orqali quyidagicha aniqlanadi:

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$$

Shundan kelib chiqqan holda quyidagi algoritmni ko'rib chiqamiz. Agar p_i soni n ning bo'luvchisi bo'lsa, n ning qiymatini quyidagicha o'zgartiramiz:

$$n=n/p_i$$

va bu bo'luvchini α_i marta takrorlaymiz. Keyingi tub soni aniqlash uchun p_i ni 1 ga oshiramiz va hokazo.

Algoritmdagi jarayonlarni kamaytirish uchun faqatgina $p_i = 2$ juft sonini alohida ko'rib chiqib, keyinchalik faqat toq sonlarni ko'rib chiqish kifoya bo'ladi.

```
program TubB;
var
     n,i,j : integer;
begin
     write('N=');
     readln(n);
     i:=2; j:=0;
 while n>1 do
begin
     while ( n mod i)<>0 do i:=i+1;
     if i<>j then
begin
     writeln (i);
     j:=i
end;
     n:=n div i
```

```
end
end.
```

Tub sonlar. *N* natural soni uchun tub sonlarni Sundaram usuli bilan aniqlash.

Masala yechimi. Yuqorida keltirilgan Tub sonni aniqlashga qaratilgan dasturda berilgan N sonigacha bo'lgan tub sonlarni aniqlash algoritmi keltirilgan edi.

1934-yilda Hindistonlik talaba Sundaram tomonidan tub sonlarni aniqlaydigan algoritm ishlab chiqilgan edi. Unga binoan har qanday N natural soni uchun 2 dan 2N+1 gacha bo'lgan oraliqdagi barcha tub sonlarni chiqarib berish mumkin. Algoritm quyidagi g'oyaga asoslangan. Birinchidan, tub son bevosita toq son bo'lganligi sababli u 2m+1 ko'rinishida bo'ladi. Agar ushbu son ko'paytuvchilarga ajraladigan bo'lsa, unda uni quyidagicha tasvirlash mumkin: 2m+1=(2i+1)(2j+1). Bu yerda i va j natural sonlar bo'lib, bundan quyidagi tenglik bajarilishi kerak bo'ladi: m=2ij+i+j. demak, agar 2ij+i+j (1<=i<=j) ko'rinishidagi barcha sonlarni o'chirib chiqsak, natijada qolgan sonlar tub son bo'ladi!

```
program SundaramTubSon;
const MaxK = 1000;
var a : array [1..MaxK] of longint;
n, k, i, j : longint;
fayl : text;
begin
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, n); close(fayl);
{fayldan sonni o'qib oldik}
      assign(fayl, 'output.txt');
rewrite(fayl);

      for i:=1 to n do a[i]:=0;
i:=1;
while (3*i+1 < n) do
{ i+j+2*i*j < n bo'lsa, j=1 deb olib 3*i+1 < n tengsizlikni hosil
qilamiz}
begin
      j:=1;
```

```
      k := i+j+2*i*j;
      while (k < n) AND (j <= i) do
begin
      a[k] := 1; j:=j+1; k := i+j+2*i*j;
end;
      i:=i+1;
end;

for i:= n-1 downto 1  do
      if a[i] = 0 then
      begin
            k:=2 * i + 1;
            write (fayl,k, ' ');
      end;
close (fayl);
end.
```

Agar N=15 bo'lsa unda quyidagi tub sonlar hosil qilinadi: 29 23 19 17 13 11 7 5 3.

Mukammal son. Mukammal son *n* o'zining natural bo'luvchilari yig'indisiga teng (n ning o'zidan boshqa), masalan, 6=1+2+3. Barcha *n* gacha bo'lgan mukammal sonlarni aniqlang.

Masala yechimi. Quyida ba'zi-bir mukammal sonlar keltirilgan:

6=1+2+3

28=1+2+4+7+14

Har qanday *m* soni uchun uning bo'luvchilarini topishda *i<=sqrt(m)* gacha bo'lgan sonlarni ko'rib chiqsak kifoya. Agar *i* bo'luvchi bo'lsa, u holda *m/i* ham *m* ning bo'luvchisi bo'ladi. Faqatgina bir xil sonlar bo'lish ehtimolini ham e'tiborga olish kerak bo'ladi, masalan, 16/4 = 4.

Haqiqatan ham, agar *m*=6 bo'lsa, u holda *sqrt(6)* ning butun qismi 2 gacha tekshirish yetarli bo'ladi. Bu yerdan 6 soni 2 ga bo'linadi va o'z navbatida 6/2=3 ga ham bo'linadi. Demak shundan kelib chiqqan holda sonning bo'luvchilarini aniqlashda biz *sqrt(m)* gacha bo'lgan sonlarni tekshiramiz.

```
program MukammalSon;
```

```
var
     k, s, m, i, j : integer;
begin
     write('m=');
     readln(m);
for i:=2 to m do
begin
     s:=1;
     j:=1;
     repeat
          j:=j+1;
          k:=i div j;
          if (i=k*j) AND  (j<=k) then
begin
     s:=s+j;
     if j<k then s:=s+k
end
     until j>=k;
     if s=i then   writeln(i)
end
end.
```

Yevklid algoritmi. Ushbu algoritm ikki natural sonning eng katta umumiy bo'luvchisini aniqlashda qo'llaniladi. Uning mazmuni va algoritmi 1-bobda ko'rib chiqilgan edi, dasturi esa quyidagicha bo'lishi mumkin:

```
program EvklidKlassik;
label BR,NP;
var
     n,m : integer;
begin
     write ('n,m=');
     readln(n,m);
     if (n=m) then goto NP;
BR:
     if (n>m) then n:=n-m
     else m:=m-n;
     if (m<>n) then goto BR;
NP:
     writeln(m)
```

end.

Ushbu dasturni quyidagi misollar bilan tekshirib ko'rsak bo'ladi:

n	*m*	EKUB
4	8	4
21	14	7
11	15	1
34	51	17

Ushbu jarayonni tezlashtirish ham mumkin. Faraz qilamiz $m<n$ bo'lsin, u holda n sonidan m ni ayiramiz, toki chiqqan qoldiq m dan kichik yoki teng bo'lsa. Ushbu hisoblashni ($n \ mod \ m$) orqali bajarish mumkin bo'ladi va natijada quyidagi ixcham va tezkor algoritmni keltirish mumkin:

1. n ni m ga bo'lib, r qoldiqni aniqlaymiz, ya'ni $n=mq+r$, $0 \le r<m$
2. agar $r=0$ bo'lsa, u holda m yechim.
3. agar $r \neq 0$ bo'lsa, u holda (n,m) o'rniga (m,r) qiymatlar tahlil qilinadi, ya'ni

1-bandga o'tamiz.

Ushbu algoritm dasturi quyidagicha bo'ladi:

```
program EvklidTezlashgan;
var
    m,n : integer;
begin
    write('n,m=');
    readln(n,m);
while (n<>0) AND (m<>0) do
begin
        if (n>=m) then n:=n mod m
        else m:=m mod n;
end;
    writeln(n+m);
{bu yerda  n yoki m nolga teng bo'ladi}
end.
```

Ushbu algoritmning samaradorligi $f(n,m)=O(5p)$ tartibli bo'ladi, bu yerda p kichik sondagi raqamlar soni.

Yevklid algoritmini yanada tezlashtirish mumkin, buning uchun sonlarni juft va toqligidan foydalanish mumkin, ya'ni quyidagi tengliklarni inobatga olish kerak:
EKUB(2n,2m)=2EKUB(n,m) – bu juft sonlar uchun va *EKUB(2n,m)=EKUB(n,m)* - bu juft va toq sonlar uchun. Shulardan kelib chiqqan holda quyidagi dasturni tavsiya etish mumkin:

```
program EvklidJuftToq;
var
     d,m,n : integer;
begin
write ('n,m=');
     readln(n,m);
     d:=1;
 while not ((m=0) or (n=0)) do
begin
     if (m mod 2 = 0) and (n mod 2 = 0) then
begin {m va n - juft }
     d:= d*2; m:= m div 2; n:= n div 2;
end
     else if (m mod 2 = 0) and (n mod 2 = 1) then
begin {m  - juft }
     m:= m div 2;
end
     else if (m mod 2 = 1) and (n mod 2 = 0) then
begin { n - juft }
      n:= n div 2;
end
     else if (m mod 2=1) and (n mod 2=1) and (m>=n) then
begin {m va n – toq, m ni kamaytiramiz }
      m:= m-n;
end
     else if (m mod 2=1) and (n mod 2=1) and (m<=n) then
begin {m va n – toq, n  ni kamaytiramiz }
      n:= n-m;
end;
end;
     writeln(d*(n+m));
{bu yerda  n yoki m nolga teng bo'ladi}
end.
```

Ushbu algoritm *binar Yevklid algoritmi* deb ataladi.

Yevklid algoritmini imkoniyatini kengaytirish ham mumkin ekan, masalan, *(n,m)* larning eng katta umumiy bo'luvchisidan $d=EKUB(n,m)$ tashqari $nx+my=d$ tenglamasining butun x va y yechimlarini ham aniqlash mumkin ekan. Buning uchun quyidagi algoritm tavsiya etiladi:

1. Agar n<m bo'lsa, u holda
c:=n; n:=m; m:=c;
2. Agar m=0 bo'lsa, u holda
d:=n; x:=1; y:=0;
va (d,x,y) – javob.
3. Agar m>0 bo'lsa, u holda
q:=[n/m]; r:=n-qm; x:=x2-qx1; y:=y2-qy1;
n:=m; m:=r; x2:=x1; y2:=y1; y1:=y;
3-bandga o't.
4. d:=n; x:=x2; y:=y2;
va (d,x,y) – javob.

Masalan, quyidagi jadvaldagi qiymatlarni tekshirib ko'ring.

n	*M*	*d*=EKUB	*nx+my=d yechimi*	
			x	*y*
21	14	7	1	-1
11	15	1	3	-4
34	51	17	1	-1

Paskaldagi dastur quyidagicha bo'ladi:

```
program EvklidTenglama;
label EN;
var
      m,n, d, q, r,x,y, x1, x2, y1, y2, c : integer;
begin
      write ('n,m=');
      readln(n,m);
if (n<m ) then
begin
      c:=n; n:=m; m:=c;
end;
```

```
if (m = 0) then
begin
      d:= n; x := 1; y := 0;    goto EN;
end;
      x2 := 1; x1 := 0; y2 := 0; y1 := 1;
      while (m > 0) do
begin
      q := n div m; r := n - q * m;
      x:= x2 - q * x1; y := y2 - q * y1;
      n := m; m := r;
      x2 := x1; x1 := x; y2 := y1; y1 := y;
end;
      d := n; x := x2; y := y2;
EN:
      writeln('d=',d, ' x=',x, ' y=',y);
end.
```

Ushbu algoritmning samaradorligi $f(n)=O(\log^2 n)$ tartibli bo'ladi.

Qisqartirilmaydigan kasr. Har qanday *k/p* ko'rinishda bo'lgan (*k/p va* $p \le 15$) barcha qisqartirilmaydigan ko'rinishda bo'lgan kasrlarni o'sish tartibi bilan chiqaring.

Masala yechimi. Faraz qilamiz bizda *m/n* kasr aniqlandi, undan keyingi kasrni esa *a/b* deb belgilaymiz. Masalaning shartiga binoan $m/n < a/b$, demak, har qanday *b* uchun quyidagini hisoblaymiz

$$a=mb/n+1$$

bu yerda *b* ning qiymatlari 2 dan 15 gacha o'zgaradi. Masalan, maxraji 4 gacha bo'lgan kasrlar quyidagilar bo'ladi: 0/1; 1/4; 1/3; 1/2; 2/3; 3/4.

Ushbu algoritmning Paskaldagi dasturi quyidagicha bo'ladi:

```
program Kasr;
var
      p,m,a,b,i, j,n : integer;
begin
      write ('P=');
      readln(p);
      m:=0; n:=1;
repeat
      writeln (m:2, ' / ',n:2, ' = ',m/n);
```

```
      i:=1; j:=1;
      for b:=2 to p do
begin
      a:=m*b div n +1;
      if a*j<b*i then
begin
      i:=a; j:=b;
end
end;
      m:=i; n:=j;
until i>=j
end.
```

3.4. Darajaga ko'tarish algoritmlar va dasturlari

Berilgan a sonining k-darajasini ya'ni a^k ni hisoblash talab qilinadi. Bunda dasturlash tilida qabul qilingan daraja funksiyasini qo'llash mumkin emas va k soni juda katta bo'lganligi sababli k marta ko'paytirishni ham qo'llash mumkin emas.

Har qanday a sonining k-chi darajasini a^k hisoblash umumiy holda muammo tug'dirishi mumkin. Bu yerda bizning asosiy maqsadimiz tuzilgan algoritmning samaradorligini oshirishdir. Demak, oddiy holda ushbu a^k ni quyidagicha hisoblash mumkin

```
d:=1
for i:=1 to k
begin
d:=d*a
end
```

Bu yerda zarur bo'lgan hisoblashlar soni $f(k)=O(k)$ tartibli bo'ladi. Uni kamaytirish yo'li quyidagi formulalardan kelib chiqadi. Ya'ni, agar k juft son bo'lsa, u holda $a^k = (a^{k/2})^2$, aks holda esa $a^k = a\,(a^{(k-1)/2})^2$.

Ushbu formulalardan kelib chiqqan holda k=15 uchun natija qanday bo'lishini ko'rsatamiz.

$a^k=a^{15}=a^{1+14}$ $=(a^1)*(a^2)^7$ $=(a^1)*(a^2)^{1+6}$ $=((a^1)*a^2)*(a^2)^6$ $=((a^1)*a^2)*(a^2)^{2*3}$ $=((a^1)*a^2)*((a^2)^2)^3$ $=((a^1)*a^2)*((a^2)^2)^{1+2}$ $=((a^1)*a^2*(a^2)^2)*((a^2)^2)^2$

Shulardan kelib chiqqan holda quyidagi algortmni tavsiya etish mumkin.

Bevosita algoritmni yaratish uchun k=5 bo'lganda ko'rib chiqamiz. Natijani P o'zgaruvchida saqlaymiz va uning boshlang'ich qiymatini P=1 deb qabul qilamiz.

Birin-ketin ko'paytirish algoritmini jadvalda tasvirlaymiz, ya'ni

$k=k-1$	$P=P*a$
4	$P=P*a=a$
3	$P=P*a=a^2$
2	$P=P*a=a^3$
1	$P=P*a=a^4$
0	$P=P*a=a^5$

Demak, k=0 ga teng bo'lsa, ushbu algoritm bo'yicha biz natijaga ega bo'lamiz, faqatgina masalaning sharti bo'yicha k marta ko'paytirishni qo'llash mumkin emas. Lekin k juft son bo'lganda biz a ning qiymatini a^2 ga almashtirib olsak, qadamlar sonini qisqartirish mumkin, ya'ni

$k=k-1$	$P=P*a$	A
4	$P=P*a=a$	
2		$a*a=a^2$
1		$a^2*a^2=a^4$
0	$P=P*a^4=a*a^4=a^5$	

Ya'ni k=5 bo'lganda 4 ta qadamda biz natijaga erishamiz.

Bu yerda asosiy g'oya shundan iboratki, agar k juft son bo'lsa, u holda

$$a^k = (a^2)^{k/2}$$

bo'ladi, demak biz bir marta a^2 ni bajarib, keyinchalik darajani hisoblashni $k/2$ ga kamaytirib qo'ydik.

Keltirilgan algoritm dasturi quyidagicha bo'ladi

```
program Daraja;
var
    a,b : real;
    k,n : integer;
begin
    writeln ('a, k ='');
    readln(a,k); write (a, '^', k, ' =');
```

```
    b:=1;
while k>0 do
begin
    n:=k div 2 ;
    if (n+n<k) then b:=b*a;{toq daraja}
    k:=n; a:=a*a;
end;{ bu erda k=0 bo'lsa, sikldan chiqiladi}
    writeln (b)
end.
```

Ushbu algoritmni bajarish uchun $f(k)=O(logk)$ ta hisoblash talab etiladi. Bu ko'rinishdagi algoritm darajaga ko'tarishning ***dixotomik algoritmi*** deb yuritiladi.

3.5. Katta sonlar bilan ishlash algoritmlar va dasturlari

Dasturlashga doir olimpiadalarda bu ko'rinishdagi sonlar bilan ishlashga ko'p to'g'ri keladi. Muammoni chuqur anglab olish uchun *n!* (*n* faktorial) misoliga nazar tashlasak. Agar *n* uchun 2 bayt ajratilgan bo'lsa, unda ishorali sonning eng katta qiymati 32767 ga teng bo'lishi mumkin. Agar bizga 8! ni hisoblash kerak bo'lsa, ya'ni 8!=40320 ga teng bo'ladi va bu esa xatolikka olib keladi, chunki 2 bayt bilan ushbu sonni ifodalab bo'lmaydi. Agar 4 bayt ajratsak, bunda 12!=479001600 soni xatolikka olib keladi.

Bu ko'rinishdagi katta sonlar uchun arifmetik amallarni bajarish algoritmlari bilan tanishib chiqamiz. Umumiy holda quyidagilarni bilishimiz kerak:

1) dasturga katta sonni kiritish
2) raqamlar sonini aniqlash
3) arifmetik amallarni bajarish
4) katta sonni chiqarish

Birin-ketin ushbu masalalarni ko'rib chiq'amiz va ularni qism dasturlar shaklida tasvirlaymiz. Katta sonni dasturga kiritish uchun uni matn sifatida qabul qilib, barcha raqamlarni massivga kiritamiz. Ushbu algoritmda satrdagi raqamlar sonli massivga teskari kiritiladi. Kiritilayotgan belgilar raqam ekanligini tekshirgan holda, ushbu dasturni barcha hollarda qo'llash mumkin bo'ladi.

```
Const Nmax = 2000;
Type Digit = 0..9;
```

```
UzunSon = Array[1..Nmax] Of Digit;
```

Endi matn qatori ko'rinishida berilgan katta sonni raqamlar massivi shakliga o'tkazamiz. Agarda bu qatorda belgilar raqamlardan iborat bo'lsa, u holda ***OK*** o'zgaruvchi ***True*** qiymatga ega bo'ladi, aks holda esa unga ***False*** qiymati beriladi. Ushbu jarayonni qism dastur shaklida yozamiz, ya'ni

```
procedure Translate(S: String; Var A : UzunSon; Var OK :
Boolean);
var i : Word;
begin
   Zero(A); i := Length(S); OK := True;
while (i >= 1) And OK Do
begin
   if S[i] In ['0'..'9'] Then
     A[Length(S)- i + 1]:= Ord(S[i]) - 48
   Else OK := False;
 i := i - 1
end
end;
```

Bu yerda *Zero(A)* qism dasturi chaqirilyapti, uning vazifasi katta sonning barcha kataklariga boshlang'ich nol qiymatini berishdir, ya'ni

```
procedure Zero(Var A : UzunSon);
var i : Integer;
begin
    For i := 1 To NMax Do A[i] := 0;
end;
```

Shunday qilib, katta son massivga yozildi va massivning oxirida nollar yozilib qolindi. Amallar bajarilishida va natija chop etilishida ushbu nollar e'tiborga olinmaydi. Endi massivga kiritilgan sonning uzunligini aniqlaymiz va uni funksiya sifatida kiritamiz, ya'ni

```
Function Uzunlik(C : UzunSon) : Integer;
Var i : Integer;
begin
     i := NMax;
     While (i > 1) And (C[i] = 0) Do i := i - 1;
     Uzunlik := i
```

```
end;
```

Ushbu funksiyada quyidagi g'oya amalga oshirilgan, ya'ni agarda uchratiladigan birinchi son nol bo'lmasa, uning joylashgan joyi sonning uzunligini bildiradi. Agar son nolga teng bo'lsa, u holda uning uzunligi birga teng bo'ladi. Buni quyidagi misol orqali anglab olamiz. Misol, n=73415608246 va $Nmax$=15 uchun C massivi quyidagi ko'rinishda bo'ladi:

C[]=	6	4	2	8	0	6	5	1	4	3	7	0	0	0	0

Oxiridan kelsak $Nmax$=11 bo'lganda massiv qiymati nol bo'lmaydi, demak sonning uzunligi 11 ga teng bo'ladi.

Yuqorida ishlab chiqilgan qism dasturlar yordamida ko'paytirish amalini bajaramiz. Unda tagma-tag usulidan foydalanamiz, faqatgina bizda ko'paytirish va qo'shish amali bir yo'la bajarilaveradi.

```
{Katta sonlarning ko'paytmasi}
{A, B — ko 'paytiruvchilar, C — natija}
procedure Multiplication(A, B : UzunSon; Var C : UzunSon);
var
      i, j : Integer; P : Digit; VspRez : 0..99;
begin
      Zero(C);
      For i := 1 To Uzunlik(A) Do
{1-sondagi raqamlar bo'yicha sikl}
begin
    P := 0; {Ko'ngildagi son nolga teng}
    For j := 1 To Uzunlik(B) Do
{2-sondagi raqamlar bo'yicha sikl}
begin
      VspRez := A[i] * B[j] + P + C[i + j - 1];
      C[i + j - 1] := VspRez Mod 10;
{ i + j – 1 joydagi raqam}
      P := VspRez Div 10 {Keyingi xonaga o'tkazish}
end;
   C[i + j] := P
{oxirgi o'tkazish noldan farqli bo'lishi mumkin, }
{ shuning uchun uni vaqtincha saqlab qo'yamiz}
end
```

```
end;
```

Endi asosiy dasturni tuzsak ham bo'ladi va u orqali 1000000 ni 123456789 ga ko'paytirib, 123456789000000 natijasini olish mumkin:

```
Program SonlarKopaytmasi;
Const NMax = 2000;
Type Digit = 0..9; UzunSon = Array[1..Nmax] Of Digit;
var
      S : String;
      M, N, R, F : UzunSon;
      i, MaxF : Word;
      Logic : Boolean;
{Katta sonli massivni nol bilan to'ldirish qism dasturi}
procedure Zero(Var A : UzunSon);
var i : integer;
begin
      for i := 1 To NMax Do A[i] := 0;
end;
Function Uzunlik(C : UzunSon) : Integer;
var i : integer;
begin
      i := NMax;
      While (i > 1) And (C[i] = 0) Do i := i - 1;
      Uzunlik := i
end;
Procedure PrintStr(Var A : UzunSon);
var i : integer;
begin
      For i := Uzunlik(A) downto 1 Do write(A[i]);
end;
procedure Translate(S : String; Var A : UzunSon; Var OK :
Boolean);
var i : Word;
begin
      Zero(A); i := Length(S); OK := True;
while (i >= 1) And OK Do
begin
      if S[i] In ['0'..'9'] Then
      A[Length(S)- i + 1]:= Ord(S[i]) - 48
```

```
      else OK := False;
      i := i - 1
end
end;
procedure Multiplication(A, B : UzunSon; Var C : UzunSon);
var
      i, j : Integer; P : Digit; VspRez : 0..99;
begin
      Zero(C);
      for i := 1 To Uzunlik(A) Do
{1-sondagi raqamlar bo'yicha sikl}
begin
      P := 0; {Ko'ngildagi son nolga teng}
      for j := 1 To Uzunlik(B) Do
{2-sondagi raqamlar bo'yicha sikl}
begin
      VspRez := A[i] * B[j] + P + C[i + j - 1];
      C[i + j - 1] := VspRez Mod 10;
{ i + j – 1 joydagi raqam}
      P := VspRez Div 10 {Keyingi xonaga o'tkazish}
end;
      C[i + j] := P
{oxirgi  o'tkazish noldan farqli bo'lishi mumkin, }
{ shuning uchun uni vaqtincha saqlab qo'yamiz}
end
end;
      {Asosiy dastur}
begin
Repeat {Son aniq kiritilgunga qadar takrorlaymiz}
      Write('Birinchi kopaytuvchi: ');
      ReadLn(S); Translate(S, M, Logic)
Until Logic;
Repeat
      Write('Ikkinchi kopaytuvchi: ');
      ReadLn(S); Translate(S, N, Logic)
Until Logic;
      Multiplication(M, N, R);
      PrintStr(R)
end.
```

Ushbu algoritmlardan foydalanib faktorialni hisoblash dasturini keltiramiz. Bevosita tajriba o'tkazib, ushbu algoritm yordamida qaysi katta natural son uchun faktorialni hisoblay olishimizni ham aniqlashimiz mumkin. Bevosita dasturda natural sonlarning faktoriali nechta raqamdan iboratligi chiqariladi.

```
Program Factorial;
Const NMax = 2000;
Type Digit = 0..9;
UzunSon = Array[1..Nmax] Of Digit;
var S : String;
      M,  F : UzunSon;
      i, MaxF : integer;
      Logic : Boolean;
{Katta sonli massivni nol bilan to'ldirish qism dasturi}
procedure Zero(Var A : UzunSon);
var i : integer;
begin
      for i := 1 To NMax Do A[i] := 0;
end;
Function Uzunlik(C : UzunSon) : Integer;
var i : integer;
begin
      i := NMax;
      While (i > 1) And (C[i] = 0) Do i := i - 1;
      Uzunlik := i
end;
Procedure PrintStr(Var A : UzunSon);
var i : integer;
begin
      For i := Uzunlik(A) downto 1 Do write(A[i]);
end;
procedure Translate(S : String; Var A : UzunSon; Var OK :
Boolean);
var i : Word;
begin
      Zero(A); i := Length(S); OK := True;
while (i >= 1) And OK Do
begin
```

```
        if S[i] In ['0'..'9'] Then
        A[Length(S)- i + 1]:= Ord(S[i]) - 48
        else OK := False;
        i := i - 1
  end
  end;
  procedure Multiplication(A, B : UzunSon; Var C : UzunSon);
  var
        i, j : Integer; P : Digit; VspRez : 0..99;
  begin
        Zero(C);
        for i := 1 To Uzunlik(A) Do
  {1-sondagi raqamlar bo'yicha sikl}
  begin
        P := 0; {Ko'ngildagi son nolga teng}
        for j := 1 To Uzunlik(B) Do
  {2-sondagi raqamlar bo'yicha sikl}
  begin
        VspRez := A[i] * B[j] + P + C[i + j - 1];
        C[i + j - 1] := VspRez Mod 10;
  { i + j – 1 joydagi raqam}
        P := VspRez Div 10 {Keyingi xonaga o'tkazish}
  end;
        C[i + j] := P
  {oxirgi  o'tkazish noldan farqli bo'lishi mumkin, }
  { shuning uchun uni vaqtincha saqlab qo'yamiz}
  end
  end;
        {Asosiy dastur}
  begin
        MaxF := 100;
        Zero(F);
        F[1] := 1;
        for i := 1 To MaxF Do
begin
        Str(i, S);
{ i sonini matnli S qatorga aylantiramiz}
        Translate(S, M, Logic);
        Multiplication(F, M, F);
```

```
    Printstr(F);
    WriteLn(i : 4, ' sonining faktorialida ', Uzunlik(F), ' ta raqam
bor');
end
end.
```

Demak, ushbu dasturning natijasi sifatida shuni aniqladikki, 100 sonining faktorialida 158 ta raqam bor ekan.

Ikki son yig'indisi. Shunday qilib, bir ko'rinishda oddiy bo'lgan ikki *a* va *b* natural sonlarni qo'shishni amalga oshiramiz. Bu yerda usulning afzalligi shundaki, kiritilayotgan raqamlar matnli qator sifatida o'qiladi, shu bois alohida maxsus kiritish va chiqarish funksiyalarini tuzish shart emas.

```
{Birinchi nolgacha satrni kesib olamiz}
function KillFirstZeroes (strA: string): string;
var  NChar: LongInt;
begin
    for NChar:=1 to Length (strA)  do
    if strA[NChar]<>'0'  then Break;
KillFirstZeroes:=Copy(strA,NChar,Length(strA)-NChar+1)
end;
{Matn  shaklidagi ikkita raqamlarni qo'shish}
function LongSumCh (A, B: Char): Char;
begin
    LongSumCh:=Chr (Ord (A)-Ord ('0')+Ord (B));
end;
{Bu yerda qo'shishni har bir raqam uchun massiv} {boshidan
boshlab amalga oshiramiz.}
{Agar yig'indi 9 dan katta bo'lsa, unda uni 10 ga } {kamaytirib,
keyingi raqamga 1 ni qo'shamiz.}
function LongSum (strA, strB: string): string;
var
    C: string;
    NChar: LongInt;
begin
    if Length (strA)<Length (strB)
    then LongSum:=LongSum (strB, strA)
    else
```

```
begin
      C:=strA;
      C:='0'+C;
      for NChar:=0 to Length (strB)-1  do
      C[Length(C)-NChar]:=LongSumCh(C[Length(C)-NChar],
strB[Length(strB)-NChar]);
      for NChar:=Length (C) downto 2  do
while C[NChar]>'9'  do
begin
      C[NChar]:=Chr (Ord (C[NChar])-10);
      C[NChar-1]:=Chr (Ord (C[NChar-1])+1);
end;
      LongSum:=KillFirstZeroes (C);
end;
end;
```

Ushbu funksiyalarning qo'llanilishi quyidagi dasturda keltirilgan:

```
Program UzunSum;
var S1,S2,strR : String;
{Birinchi nolgacha satrni kesib olamiz, ushbu nol } {LongSum
funksiyasiga qo'shilgan}
function KillFirstZeroes (strA: string): string;
var
      NChar: LongInt;
begin
      for NChar:=1 to Length (strA)  do
      if strA [NChar]<>'0'   then Break;
 KillFirstZeroes:=Copy(strA,NChar,Length(strA)-NChar+1)
 end;
 {Matn  shaklidagi ikkita raqamlarni qo'shish}
 function LongSumCh (A, B: Char): Char;
 begin
      LongSumCh:=Chr (Ord (A)-Ord ('0')+Ord (B));
 {Ord(ch) – ASCII jadvali bo'yicha tartib son}
 end;
 {Bu yerda qo'shishni har bir raqam uchun massiv }
 {oxiridan boshlab amalga oshiramiz. }
```

```
{ Agar yig'indi 9 dan katta bo'lsa, unda uni 10 ga } {kamaytirib,
keyingi raqamga 1 ni qo'shamiz.}
function LongSum (strA,strB: string): string;
var
      C: string;
      NChar: LongInt;
begin
      if Length (strA)<Length (strB)  then
      LongSum:=LongSum (strB, strA)
{Birinchi qo'shiluvchi ikkinchisiga nisbatan uzunroq}
      else
begin
      C:=strA;
      C:='0'+C;
      for NChar:=0 to Length (strB)-1  do
      C[Length(C)-NChar]:=LongSumCh(C[Length (C)-NChar],
strB [Length(strB)-NChar]);
      for NChar:=Length (C) downto 2  do
while C[NChar]>'9'  do
begin
      C[NChar]:=Chr (Ord (C[NChar])-10);
      C[NChar-1]:=Chr (Ord (C[NChar-1])+1);
end;
      LongSum:=KillFirstZeroes (C);
end;
end;
      {Asosiy dastur}
begin
      Write('Birinchi qoshiluvchi: ');
      ReadLn(S1);
      Write('Ikkinchi qoshiluvchi: ');
      ReadLn(S2);
      strR:=longSum(S1,S2);
      writeln(strR);
end.
```

Endi bevosita katta sonlarni, masalan 62767 va 123456789 sonlarini qo'shib, 123519556 natijasini olish imkoniga ega bo'ldik.

Ikki son ayirmasi. Ikki a va b natural sonlarni ayirish bevosita qo'shish amaliga o'xshash. Bu yerda qatorning oxiridan boshlab raqamlarni ayiramiz. Keyin yana oxiridan boshlab tekshiramiz, agar qiymat noldan kichik bo'lsa, u holda uning qiymatiga 10 ni qo'shamiz va oldingi raqamni birga kamaytiramiz. Misol sifatida, masalan a=215 va b=8 bo'lsin.

a	2	1	5
-			
b	0	0	8
	2	1	-3

Chiqqan natijada oxirgi raqam -3, u noldan kichik, demak unga 10 ni qo'shamiz, ya'ni -3+10=7 va shu bois oldingi raqamdan 1 ni ayiramiz, ya'ni 1-1=0. Birinchi raqam esa noldan katta, shu bois o'zgartirilmaydi. Demak, javob 207 ga teng.

Dasturda biz $a>b$ deb hisoblaymiz.

```
{Ikki raqam ayirmasi}
function LongMinCh (A, B: Char): Char;
begin
    LongMinCh:=Chr (Ord (A)-(Ord (B)-Ord ('0')));
end;
{Ikki son ayirmasi}
function LongMin (strA, strB: string): string;
var
    C: string;
    NChar: LongInt;
begin
    C:=strA;
    for NChar:=0 to Length (strB)-1 do
    C[Length(C)-NChar]:=LongMinCh (C [Length (C)-NChar],
strB [Length (strB)-NChar]);
    for NChar:=Length (C) downto 2  do
while C[NChar]<'0'  do
begin
    C[NChar]:=Chr (Ord (C[NChar])+10);
    C[NChar-1]:=Chr (Ord (C[NChar-1])-1);
```

```
end;
    LongMin:=KillFirstZeroes (C);
end;
```

Ushbu funksiyalarning qo'llanilishi quyidagi dasturda keltirilgan:

```
Program UzunAyirma;
var S1,S2,strR : String;
    {Birinchi nolgacha satrni kesib olamiz}
function KillFirstZeroes (strA: string): string;
var
    NChar: LongInt;
begin
    for NChar:=1 to Length (strA)  do
    if strA [NChar]<>'0'  then Break;
    KillFirstZeroes:=Copy (strA, NChar, Length (strA)-NChar+1)
end;
    {Ikki raqam ayirmasi}
function LongMinCh (A, B: Char): Char;
begin
    LongMinCh:=Chr (Ord (A)-(Ord (B)-Ord ('0')));
end;
    {Ikki son ayirmasi}
function LongMin (strA, strB: string): string;
var
    C: string;
    NChar: LongInt;
begin
    C:=strA;
    for NChar:=0 to Length (strB)-1 do
    C[Length(C)-NChar]:=LongMinCh(C[Length (C)-NChar],
strB[Length(strB)-NChar]);
    for NChar:=Length (C) downto 2  do
while C[NChar]<'0'  do
begin
    C[NChar]:=Chr (Ord (C[NChar])+10);
    C[NChar-1]:=Chr (Ord (C[NChar-1])-1);
end;
    LongMin:=KillFirstZeroes (C);
end;
```

```
    {Asosiy dastur}
begin
    Write('kamayuvchi: ');
    ReadLn(S1);
    Write('ayiruvchi: ');
    ReadLn(S2);
    strR:=Longmin(S1,S2);
    Write(strR);
end.
```

Endi bevosita katta sonlarni, masalan 123456789 va 456789 sonlarini ayirib, 123000000 natijani olish imkoniga ega bo'ldik.

Ikki son ko'paytmasining ikkinchi usuli. Qo'shish amalidan keyin ko'paytirish-ga o'tish yengil amalga oshiriladi. Buning uchun oldiniga sonni bitta raqamga ko'paytirishni bajaramiz, so'ng umumiy masalaga o'tamiz.

```
{Ikkita raqam ko'paytmasi}
function LongMulCh (A, B: Char): Char;
begin
LongMulCh:=Chr((Ord(A)-Ord('0'))*(Ord(B)-Ord('0'))+Ord ('0'));
end;
    {Son bilan raqam ko'paytmasi}
function LongMulOnCh (strA: string; aCh: Char): string;
var
    C: string;
    NChar: LongInt;
begin
    C:='0'+strA;
    for NChar:=1 to Length (C) do
    C[NChar]:=LongMulCh (C[NChar], aCh);
    for NChar:=Length (C) downto 2 do
while C[NChar]>'9' do
begin
    C[NChar]:=Chr (Ord (C[NChar])-10);
    C[NChar-1]:=Chr (Ord (C[NChar-1])+1);
end;
    LongMulOnCh:=KillFirstZeroes (C);
```

```
end;
{Ikkita katta sonlarning ko'paytmasi}
function LongMul (strA, strB: string): string;
var
      C, ZeroOffset: string;
      NChar: LongInt;
begin
      if Length (strA<Length (strB)  then LongMul:=LongMul
(strB, strA)
      else
begin
      ZeroOffset:='';
      C:='0';
      for NChar:=Length (strB) downto 1  do
begin
      C:=LongSum (C, LongMulOnCh (strA, strB
[NChar])+ZeroOffset);
      ZeroOffset:=ZeroOffset+'0';
end;
      LongMul:=C;
end;
end;
```

Ushbu funksiyalarning qo'llanilishi quyidagi dasturda keltirilgan:

```
Program Long_Kupaytma;
var S1,S2,strR : String;
      {Birinchi nolgacha satrni kesib olamiz}
function KillFirstZeroes (strA: string): string;
var
      NChar: LongInt;
begin
      for NChar:=1 to Length (strA)  do
      if strA[NChar]<>'0'  then Break;
KillFirstZeroes:=Copy(strA,NChar,Length(strA)-NChar+1)
end;
function LongSumCh (A, B: Char): Char;
begin
      LongSumCh:=Chr (Ord (A)-Ord ('0')+Ord (B));
```

```
end;
function LongSum (strA,strB: string): string;
var
      C: string;
      NChar: LongInt;
begin
      if Length (strA)<Length (strB)  then
      LongSum:=LongSum (strB, strA)
      else
begin
      C:=strA;
      C:='0'+C;
      for NChar:=0 to Length (strB)-1  do
      C[Length (C)-NChar]:=LongSumCh (C[Length (C)-NChar],
strB [Length (strB)-NChar]);
      for NChar:=Length (C) downto 2  do
while C[NChar]>'9'  do
begin
      C[NChar]:=Chr (Ord (C[NChar])-10);
      C[NChar-1]:=Chr (Ord (C[NChar-1])+1);
end;
      LongSum:=KillFirstZeroes (C);
end;
end;
      {Ikkita raqam ko'paytmasi}
function LongMulCh (A, B: Char): Char;
begin
      LongMulCh:=Chr ((Ord (A)-Ord ('0'))*(Ord (B)-Ord
('0'))+Ord ('0'));
end;
      {Son bilan raqam ko'paytmasi}
function LongMulOnCh (strA: string; aCh: Char): string;
var
      C: string;
      NChar: LongInt;
begin
      C:='0'+strA;
      for NChar:=1 to Length (C)  do
      C[NChar]:=LongMulCh (C[NChar], aCh);
```

```
    for NChar:=Length (C) downto 2  do
while C[NChar]>'9'  do
begin
    C[NChar]:=Chr (Ord (C[NChar])-10);
    C[NChar-1]:=Chr (Ord (C[NChar-1])+1);
end;
    LongMulOnCh:=KillFirstZeroes (C);
end;
    {Ikkita katta sonlarning ko'paytmasi}
function LongMul (strA, strB: string): string;
var
    C, ZeroOffset: string;
    NChar: LongInt;
begin
    if Length (strA)<Length (strB)  then LongMul:=LongMul
(strB, strA)
    else
begin
    ZeroOffset:='';
    C:='0';
    for NChar:=Length (strB) downto 1  do
begin
    C:=LongSum (C, LongMulOnCh(strA,strB
[NChar])+ZeroOffset);
    ZeroOffset:=ZeroOffset+'0';
end;
    LongMul:=C;
end;
end;
    {Asosiy dastur}
begin
    Write('Birinchi kopaytuvchi: ');
    ReadLn(S1);
    Write('Ikkinchi kopaytuvchi: ');
    ReadLn(S2);
    strR:=longmul(S1,S2);
    Writeln(strR);
end.
```

Endi bevosita katta sonlarni, masalan 12767 va 123456789 sonlarini ko'paytirib, 1576172825163 natijasini olish imkoniga ega bo'ldik.

Ikki son butun bo'linmasi. Bo'lish amalini bajarish uchun biz maktabdan ma'lum bo'lgan algoritmni, ya'ni ustunlab bo'lishni qo'llaymiz. Bo'lish natijasida qoldiqli yoki qoldiqsiz natija bo'lishi mumkin.

Masalani to'liq yechish uchun oldin ikkita oddiy dasturlarni yaratamiz. Birinchi dasturda biz bo'luvchi sonni kichikligini aniqlaymiz. Ikkinchi dasturda esa birinchi sonning ichida nechta ikkinchi son borligi aniqlanadi, ya'ni ustunlab bo'lishning ilk boshlanishidagi jarayon ko'riladi.

```
{Birinchi satr kattaligini aniqlash funksiyasi}
function FirstIsGreater (strA,straB: string): Boolean;
begin
     while Length (strA)<Length (strB)  do strA:='0'+strA;
     while Length (strB)<Length (strA)  do strB:='0'+strB;
     FirstIsGreater:=strA>strB;
end;
{ikkita yaqin sonlar  strA  va strB uchun  strA da } {nechta strB
borligini aniqlash}
function LongDivToCh (strA, strB: string): Char;
var
     Ch: Char;
     C: string;
begin
     C:='0';
     for Ch:='1' to '9'  do
begin
     C:=LongSum (C, strB);
     if FirstIsGreater (C, strA)  then
begin
     Dec(Ch);
     Break;
end
     else if C=strA  then Break;
end;
     LongDivToCh:=Ch;
```

```
end;
{ qoldiqsiz bo'lish funksiyasi}
function LongDiv (strA, strB: string): string;
var
      NChar: LongInt;
      C, Buf: string;
      ToChDivResult: Char;
begin
      C:='';
      Buf:='';
      for NChar:=1 to Length (strA)  do
begin
      Buf:=KillFirstZeroes (Buf+strA [NChar]);
      ToChDivResult:=LongDivToCh (Buf, strB);
      C:=C+ToChDivResult;
      if ToChDivResult>'0'  then
      Buf:=LongMin (Buf, LongMulOnCh (strB, ToChDivResult));
end;
      LongDiv:=KillFirstZeroes (C);
end;
```

Ushbu dasturlardan foydalanishning to'liq ko'rinishi quyidagicha bo'ladi:

```
program Uzun_Butun_Bulish;
var S1,S2,strR : String;
{Birinchi  nolgacha  satrni  kesib olamiz}
function KillFirstZeroes (strA: string): string;
var NChar: LongInt;
begin
      for NChar:=1 to Length (strA)  do
      if strA [NChar]<>'0'  then Break;
      KillFirstZeroes:=Copy(strA,NChar,Length(strA)-NChar+1)
end;
function LongSumCh (A, B: Char): Char;
begin
      LongSumCh:=Chr (Ord (A)-Ord ('0')+Ord (B));
end;
function LongSum (strA,strB: string): string;
```

```
var
      C: string;
      NChar: LongInt;
begin
      if Length (strA)<Length (strB)  then LongSum:=LongSum
(strB, strA)
      else
begin
      C:=strA;
      C:='0'+C;
      for NChar:=0 to Length (strB)-1  do
      C[Length(C)-NChar]:=LongSumCh(C[Length (C)-NChar],
strB [Length (strB)-NChar]);
      for NChar:=Length (C) downto 2  do
while C[NChar]>'9'  do
begin
      C[NChar]:=Chr (Ord (C[NChar])-10);
      C[NChar-1]:=Chr (Ord (C[NChar-1])+1);
end;
      LongSum:=KillFirstZeroes (C);
end;
end;
{Ikkita raqam ko'paytmasi}
function LongMulCh (A, B: Char): Char;
begin
      LongMulCh:=Chr ((Ord (A)-Ord ('0'))*(Ord (B)-Ord
('0'))+Ord ('0'));
end;
{Son bilan raqam ko'paytmasi}
function LongMulOnCh (strA: string; aCh: Char): string;
var
      C: string;
      NChar: LongInt;
begin
      C:='0'+strA;
      for NChar:=1 to Length (C)  do
      C[NChar]:=LongMulCh (C[NChar], aCh);
      for NChar:=Length (C) downto 2  do
while C[NChar]>'9'  do
```

```
begin
      C[NChar]:=Chr (Ord (C[NChar])-10);
      C[NChar-1]:=Chr (Ord (C[NChar-1])+1);
end;
      LongMulOnCh:=KillFirstZeroes (C);
end;
{Birinchi satrni  kattaligini aniqlash}
function FirstIsGreater (strA, strB: string): Boolean;
begin
      while Length (strA)<Length (strB)   do strA:='0'+strA;
      while Length (strB)<Length (strA)   do strB:='0'+strB;
{satrlar  uzunligini  tenglashtirdik, endi  } {kattaligini
tenglashtiramiz}
      FirstIsGreater:=strA>strB;
end;
{ikkita yaqin sonlar  strA  va strB uchun  strA da } {nechta strB
borligini aniqlash}
function LongDivToCh (strA, strB: string): Char;
var
      Ch: Char;
      C: string;
begin
      C:='0';
      for Ch:='1' to '9'  do
begin
      C:=LongSum (C, strB);
      if FirstIsGreater (C, strA)  then
begin
      Dec(Ch); {ch ning qiymatini bittaga kamaytiramiz}
      Break;
end
      else
      if C=strA  then Break;
end;
      LongDivToCh:=Ch;
end;
      {Ikki raqam ayirmasi}
function LongMinCh (A, B: Char): Char;
begin
```

```
    LongMinCh:=Chr (Ord (A)-(Ord (B)-Ord ('0')));
end;
    {Ikki son ayirmasi}
function LongMin (strA, strB: string): string;
var C: string;
    NChar: LongInt;
begin
    C:=strA;
    C:=C;
    for NChar:=0 to Length (strB)-1 do
{raqamlarni mos ravishda xonalar bo'yicha ayirib} {chiqamiz}
    C[Length(C)-NChar]:=LongMinCh(C[Length (C)-NChar],
strB[Length (strB)-NChar]);
        for NChar:=Length (C) downto 2  do
while C[NChar]<'0' do
{manfiy bo'lsa,unga 10 ni qo'sib,oldingisidan 1 ni} {ayiramiz }
begin
    C[NChar]:=Chr (Ord (C[NChar])+10);
    C[NChar-1]:=Chr (Ord (C[NChar-1])-1);
end;
    LongMin:=KillFirstZeroes (C);
end;
{ qoldiqsiz bo'lish funksiyasi}
function LongDiv (strA, strB: string): string;
    {bu yerda  strA – bo'linuvchi, strB – bo'luvchi}
var
    NChar: LongInt;
    C, Buf: string;
    ToChDivResult: Char;
begin
    C:='';
    Buf:='';
    for NChar:=1 to Length (strA)  do
begin
    Buf:=KillFirstZeroes (Buf+strA [NChar]);
    ToChDivResult:=LongDivToCh (Buf, strB);
    C:=C+ToChDivResult;
    if ToChDivResult>'0'  then
    Buf:=LongMin (Buf, LongMulOnCh (strB, ToChDivResult));
```

```
end;
    LongDiv:=KillFirstZeroes (C);
end;
    {Asosiy dastur}
begin
    Write('Bolinuvchi: ');
    ReadLn(S1);
    Write('Boluchi: ');
    ReadLn(S2);
    strR:=longDiv(S1,S2);
    Writeln(strR);
end.
```

Endi bevosita katta sonlarni, masalan 123456789 va 1000 sonlarini bo'lib, 123456 natijasini olish imkoniga ega bo'ldik.

Shunga o'xshash dasturlarni quyidagi funksiyalar uchun tekshirishni mustaqil bajaring.

```
{Bo'lishdagi qoldiqni aniqlash funksiyasi}
function LongMod (strA, strB: string): string;
begin
  LongMod:=LongMin (strA, LongMul (strB, LongDiv (strA,
strB)));
end;
{Sonni juft yoki toqligini aniqlash funksiyasi}
function LongOdd (strA: string): Boolean;
begin
    LongOdd:=strA[Length(strA)] in ['1','3','5','7','9'];
end;
{Sonni kvadratini hisoblash funksiyasi}
function LongSqr (const strA: string): string;
begin
    LongSqr:=LongMul (strA, strA);
end;
```

Yuqorida keltirilgan algoritmlar maktabdan bizga ma'lum bo'lgan texnologiyalar asosida amalga oshirildi. Lekin sanoq tizimlarida keltirilgan sonning umumiy ko'rinishidan foydalanib, boshqa ko'rinishdagi algoritmlar ham yaratish mumkin, ya'ni

$$N_{(d)}=A_m d^m+A_{m-1} d^{m-1}+\ldots+A_1 d^1+A_0 d^0 \quad (*)$$

Har qanday N sonining A_i koeffitsiyentlarini massivda saqlab qo'ysak kifoya. Demak, $N=12345_{(10)}$ soni uchun $A=\{A_i\}=\{5,4,3,2,1\}$ bo'ladi. E'tibor bering A massivida raqamlar teskari joylashtirilgan. Shundan kelib chiqqan holda juda katta sonlar uchun d sonini kattalashtirish talab etiladi. Masalan,

20! = 243,2902,0081,7664,0000 soni uchun $d=10000$ deb qabul qilamiz va (*) bo'yicha quyidagini tashkil qilamiz

$$20! = 0\cdot d^0+7664\cdot d^1+81\cdot d^2+2902\cdot d^3+243\cdot d^4,$$

Demak $A=\{A_i\}=\{0, 7664, 81, 2902, 243\}$ bo'ladi.

Misol sifatida quyidagini hisoblaymiz 12!+20! . Sonlarni jadvalda tasvirlaymiz va qo'shish hisoblarini bajaramiz:

$N \setminus A$	A_0	A_1	A_2	A_3	A_4
12!	4	7900	1600		
20!	0	7664	81	2902	243
yig'indi	4	5564	1682	2902	243

Demak, 12!+20! = 243 2902 1682 5564 0004.

Ushbu algoritm yordamida boshqa amallarni ham bajarish mumkin bo'ladi.

Topshiriqlar

1. Barcha 100 dan kichik bo'lgan natural sonlarning kvadratini chop etadigan dasturni yarating.
2. Yuqoridagi misolni faqatgina qo'shish va ayirish arifmetik amallaridan foydalangan holda bajaring.
3. Barcha har xil raqamlardan tashkil topgan 4 xonali sonlarni chop eting.
4. Berilgan N natural sonini ikkilik sanoq tizimiga o'tkazamiz, ya'ni

$$a_1 a_2 a_3 \ldots a_k$$

bu yerda $a_i = 0$ yoki 1. Birin-ketin a_i larni siljitib yangi ketma-ketliklarni, ya'ni

sonlarni tashkil qilamiz, masalan

$$a_k a_1 a_2 \ldots a_{k-1}$$
$$a_{k-1} a_k a_1 \ldots a_{k-2}$$

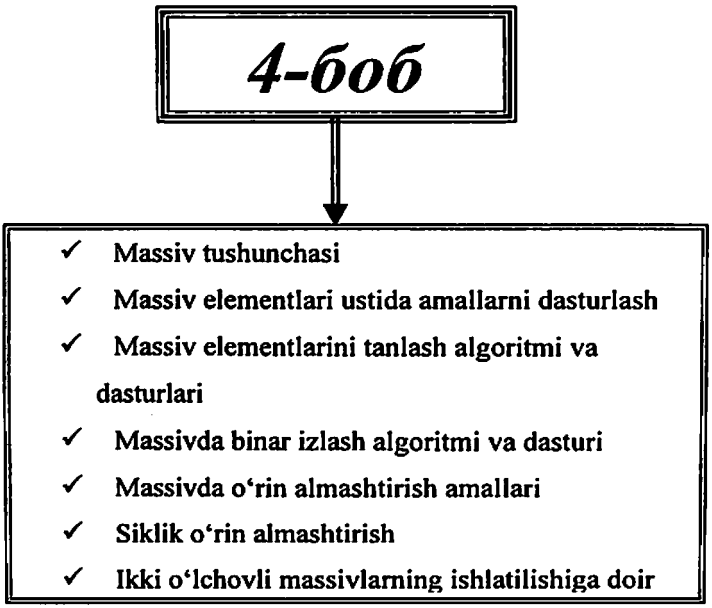
va hokazo. Ushbu sonlardan eng kattasini aniqlang.

5. Berilgan N natural sonining raqamlaridan faqatgina bittasini o'chirish orqali yangi sonlarni tashkil qilamiz. Ushbu sonlardan eng kattasini aniqlang.
6. Berilgan x haqiqiy sonining butun qismini aniqlang.
7. Berilgan N butun sonning juft sonligini aniqlang.
8. Berilgan 4 xonali N natural sonida yuzliklar nechta ekanligini aniqlang.
9. Berilgan N natural son nechta raqamlardan tashkil topganini aniqlang.
10. Berilgan N natural son palindrom son ekanligini aniqlang, ya'ni chapdan o'ngga va teskari bir xil son.
11. Shaxmat taxtasida 2 ta xonaning koordinatalari sonlar bilan berilgan, ya'ni (m,n) va (k,l). Ushbu xonalar rangi bir xilligini aniqlang.
12. Shaxmat taxtasida 2 ta xonada to'ra joylashgan. Ular bir-birini urishlari mumkinligini aniqlang.
13. Berilgan N natural sonining kvadrati N bilan tugallanishini aniqlang, masalan $25^2=625$.
14. Uch xonali N natural son o'z raqamlarining kublari yig'indisiga tengligini aniqlang, masalan $153=1^3+5^3+3^3$.
15. Berilgan N natural son daqiqalarda berilgan bo'lib, uni soat va daqiqalarga ajrating.

4-bob. DASTURLASHDA MASSIV TUSHUNCHASI

Algebra fanida sonlarni matritsa ko'rinishida saqlash undagi qiymatlarni yagona yondashuv orqali qayta ishlash imkoni yaratib berdi. Natijada ma'lumotlarga ishlov berish va uni qayta ishlash bilan bog'liq bo'lgan ko'pgina masalalar yechimi oddiyroq ko'rinishga olib kelindi. Xuddi shunday, dasturlash tillarida ham shunga o'xshash massiv tushunchasi kiritilgan bo'lib, ularsiz dasturlarni tasavvur qilish mumkin emas.

Mazkur bobda ko'p jihatdan zarur bo'lgan massiv tushunchasi va uning elementlari bilan ishlash asoslari, hamda massivlarga doir murakkab masalalar ko'rib chiqilgan.



4.1. Massiv tushunchasi

Agar ma'lumotlar muayyan tartib bo'yicha saqlanilgan bo'lsa, u holda undagi qiymatlarni yagona yondashuv orqali qayta ishlash mumkin bo'ladi. Natijada ma'lumotlarga ishlov berish va uni qayta ishlash bilan bog'liq bo'lgan ko'pgina masalalar yechimi oddiyroq, tezroq va samaraliroq aniqlanishi mumkin. Shu bois dasturlashda massiv orqali tashkil qilingan qiymatlar keng qo'llaniladi. Buning asosiy sabablaridan biri bu massiv elementlariga to'g'ridan-to'g'ri murojaat bo'lsa, ikkinchidan kompyuterda operativ xotiraning tuzilishidir. Albatta massivlarni qo'llashda ma'lum – bir kamchiliklar ham mavjud, masalan, ko'p o'lchamli massivda uning elementini operativ xotiradan manzilini aniqlash juda ko'p arifmetik amallarni bajarishni talab qiladi. Bundan tashqari massivning uzunligi oldindan aniqlangan bo'lishi kerak.

Umumiy nomga ega bo'lgan indeks bo'yicha tartiblangan va bir xil tipdagi elementlar ketma-ketligi ***massiv*** deb ataladi. Massiv elementlarining soni dastur ishlash jarayonida unga o'zlashtirib boriladi. Uning har bir aloxida elementiga murojaat massiv elementlariga mos kelgan indeks bo'yicha bo'ladi. Massiv indeksi xuddi vektorlar indeksi tushunchasi kabi bo'ladi. Massivlarni e'lon qilish uchun "**array of**" tayanch so'zilari ishlatiladi.

Paskal tilida umumiy ko'rinishdagi "**type**" orqali yangi tipni kiritish va keyinchalik u orqali massivni e'lon qilish mumkin. Masalan

```
type
<tip nomi> = array[indeks tipi] of <o'zgaruvchi tipi>;
  var
    < 1-o'zgaruvchi, 2- o'zgaruvchi ...> : < tip nomi>;
```

Massivlarni yangi tiplarni e'lon qilmasdan ham bevosita kiritish mumkin, masalan

```
  var
    MyMassiv :  array[1..60] of integer;
    iMas : array[1..4] of  integer;
```

Paskal tilida ko'p o'lchamli massivni kiritish quyidagi ikki usul bilan amalga oshiriladi. Masalan

```
type
```

```
  Vector = array[1..4] of integer;
  Massiv = array[1..4] of Vector;
var
  Matr : Massiv;
```

Ushbu ikki o'lchamli massivni boshqacha ham hosil qilish mumkin, masalan

```
var
  Matr : array[1..4,1..4] of integer;
```

Shunday qilib, agarda massivlarda bitta indeks qatnashsa, bu massiv bir o'lchovli, ikkita indeks bo'lsa, ikki o'lchovli, *n* ta indeks bo'lsa, u *n* o'lchovli massiv deyiladi.

Misol.

```
var
  MyVector: array[1..40] of real;
{ 40 ta elementdan tashkil topgan 1 o'lchovli massiv }
  MyMatr : array[1..8,1..8] of byte;
{8x8=64 ta elementdan tashkil topgan 2 o'lchovli massiv}
  MyCube : array[1..4,1..5,1..8] of integer;
{ uch o'lchovli massiv }
```

Massivlarni e'lon qilishda konstantalarni ham ishlatilish mumkin, masalan

```
const
  G1 = 4; G2 = 6;
var
  MasY : array[1..G1,1..G2] of real;
```

Massiv elementlari xotirada ketma-ket joylashadi. Indekslari kichkina bo'lgan elementlar xotiraning pastki adreslarida saqlanadi. Ko'p elementli massivlarda eng o'ng tarafdagi indeks birinchi bo'lib o'sib boradi. Masalan, agarda

```
  A : array[1..5,1..5] of integer;
```

bo'lsa, u holda massiv elementlari adreslarning o'sishi bo'yicha joylashadi:

```
  A[1,1]
  A[1,2]
  ...
  A[1,5]
```

A[2,1]
A[2,2]

4.2. Massiv elementlari ustida amallarni dasturlash

Massivni e'lon qilingandan keyin uni elementlarini nomlari kvadrat qavs ichida ko'rsatilgan holda ishlatish mumkin.Masalan: *Mas*[2], *VectorZ*[10] massivning ikkinchi va o'ninchi elementlariga murojaat etishni bildiradi.

Ikki o'lchovli massivlarda ikkita indeks, masalan, *arrMatr*[4,4] bu yozuv *arrMatr* massivining 4-qator va 4-ustunidagi elementini bildiradi. Massiv indeksi 1..*n* bo'ladigan bo'lsa, uni *A*[n] deb yozamiz, indeks chegaralari nostandart bo'lsa, uni bevosita yozib ko'rsatamiz, masalan *A*[0..n]. Agar *A*[] deb yozilsa, ya'ni kvadrat qavs ichida hech qanday son bo'lmasa, bu massivning uzunligi ahamiyatsiz ekanligini bildiradi,

A[] massiv elementlariga qiymat berishni, masalan quyidagicha amalga oshirish mumkin, bu operatsiyani biz **FOR** operatori yordamida bajaramiz:

```
for i := 1 to 4 do A[i] := 0;
```

E'tibor bering, Paskal algoritmik tilida massiv elementlarini birdaniga kiritish - chiqarish imkoniyati yo'qligi sababli elementlar bittadan kiritiladi. Quyidagi misolni ko'rib chiqamiz.

Kvadrat va ildizni hisoblash. 5 ta elementdan tashkil topgan *X*[5] massivining elementlarini istalgan butun sonlar bilan fayldan to'ldiring. Har bir elementning kvadrati va ildizini hisoblab, faylga chiqaring.
Masala yechimi. Bu erda *X*[i] elementlariga qiymat berilib, keyinchalik uning kvadratini va ildizini hisoblab, natijani chop etish talab etiladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
1	1 1
4	16 2
9	81 3
0	0 0
16	256 4

Bu yerda *input.txt* faylida birinchi qatorda 1-element, ikkinchi qatorda 2-element va h.k., ya'ni elementlar vertikal joylashtirilgan, *output.txt* faylida har bir qatorda sonning kvadrati va ildizi chiqarilgan bo'ladi.

Keltirilgan dasturni berilgan izohlar orqali tushunib olish qiyinchilik tug'dirmaydi.

```
program Kvadrat;
var {o'zgaruvchilarni e'lon qilamiz}
      i: integer; {siklni tashkillashtirishda qo'llaymiz}
      X: array[1..5] of integer; {boshlang'ich massiv}
      Ildiz, Kv: array[1..5] of real;
      {- natijaviy massivlar}
      fayl : text;{fayl ko'rsatkichi va uning turi}
begin
{5 ta butun qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      for i:=1 to 5 do
      readln(fayl, X[i]);
{sikl orqali X[] qiymatlarini kiritamiz}
      close(fayl);
{bu erda hisoblash jarayoni}
      for i:=1 to 5 do
begin
      Ildiz[i]:= Sqrt(X[i]);       { ildiz }
      Kv[i]:= Sqr(X[i]);          { kvadrat }
end;
      {chiqarish faylini ochamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i:=1 to 5 do
      Writeln(fayl,Kv[i]:0:0,' ', Ildiz[i]:0:0);
{natijalarni chop etdik}
      close(fayl);
 end. {dastur tugatildi}
```

Yig'indi. O'lchami *n* bo'lgan massiv elementlarini kiriting va ularning yig'indisi-ni hisoblang.

Masala yechimi. Masalan, (3, 4, 0, 5, 1) massivi uchun natija 3+4+0+5+1=13 bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 3 4 0 5 1	13

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va qolgan qatorlarda massiv elementlari bo'ladi va *output.txt* faylida sonlarning yig'indisi chiqarilgan bo'ladi.

```
program Yigindi;
const nmax=50;
var
      A: array[1..nmax] of integer;
      Sum, n, i:  integer;
      fayl : text;
begin
{massiv o'lchamini va  qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl,n);
      for i:=1 to n do
      readln(fayl, A[i]);
{sikl orqali A[] qiymatlarini kiritamiz}
      close(fayl);
      Sum:= 0;
{ yig'indini 0 ga tenglashtiramiz }
      for i:= 1 to n do
      Sum:= Sum+A[i];
{chiqarish faylini ochamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      Write(fayl, Sum); { yig'indini chop etish}
      close(fayl);
end.
```

Ko‘pchilik holatlarda massivda uning qaysidir elementlarini izlashga to‘g‘ri keladi. Masalan, $A[]$ massivining nechta elementi nol qiymatga ega ekanligini bilish talab etiladi. Buning uchun qo‘shimcha o‘zgaruvchi K ni kiritamiz (ko‘pincha ushbu toifadagi o‘zgaruvchini hisoblagich deb aytishadi) va **FOR, IF** operatorlaridan foydalanamiz:

```
K:=0;
for i:=1 to n do
if A[i] = 0 then K : = K+1;
```

Sikl bajarilgandan keyin K o‘zgaruvchi $A[]$ massivining nolga teng bo‘lgan elementlarining soniga teng bo‘ladi. Massiv elementlari qiymatlarini joyini almashtirish massivning bazali tipiga o‘xshash tipdagi yordamchi o‘zgaruvchi yordamida amalga oshiriladi. Masalan, $A[]$ massivining birinchi va beshinchi elementlari qiymatlarini joyini almashtiring:

```
Vs : =A[5]                    { Vs- yordamchi o‘zgaruvchi }
A[5] :=A[1];
A[1]: =Vs;
```

Uchraydi. Butun sonli $A[n]$ massivida eng ko‘p uchraydigan sonni aniqlang.

Masala yechimi. Masalan, quyidagi massivda (2, 5, 3, 1, 5) eng ko‘p uchraydigan son 5 ga teng. Ushbu masala to‘g‘ridan-to‘g‘ri yechiladi, ya’ni har bir element $A[i]$ keyingilari bilan solishtiriladi. Bu yerda birinchi navbatda ushbu son necha marta uchrashishi hisoblab boriladi va ikkinchi navbatda ular ketma-ket $A[i+1]$, $A[i+2]$... yozib boriladi. Massiv oxirigacha ko‘rib chiqilgandan so‘ng, $A[i]$ soni K marta uchragani aniqlanadi, shu bois, keyingi ko‘riladigan elementimiz $A[i+k]$ dan keyingisi bo‘ladi. Yuqorida keltirilgan massiv uchun ushbu algoritmni ishlatib ko‘ramiz. Demak, i=1 bo‘lganda $A[1]$=2 elementi massiv bo‘yicha necha marta uchrashishini tekshirib chiqamiz va natija bu erda k=1 bo‘ladi. Keyingi elementimiz $i = i+1 = 1+1 = 2$, ya’ni 2-o‘rinda joylashgan $A[2]$=5 bo‘ladi. Sikl orqali $(i+1)$ - o‘rindan boshlab ushbu elementga teng bo‘lgan sonlar tekshirilganda k=2 ga teng bo‘ladi va ushbu qiymat oldinroq topilgan uzunlikdan katta bo‘lsa, u holda o‘rin almashtiriladi va (2, 5, 5, 1, 3) massivi hosil bo‘ladi va h.k.

Umumiy holda ushbu algoritmning samaradorlik darajasi $O(n^2)$ tartibli ekanligi ko'rinib turibdi. Agar $A[]$ massivda har xil elementlar soni m ta bo'lsa, u holda samaradorlik darajasi $O(n \cdot m)$ tartibli bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 2 5 3 1 5	5

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va qolgan qatorlarda massiv elementlari bo'ladi va *output.txt* faylida ko'p uchraydigan son chiqarilgan bo'ladi.

Dastur esa quyidagicha bo'ladi.

```
program Uchraydi;
const Nmax=100;
var
      n,m,am,i,j,k,r : integer;
      A:array[1..Nmax] of integer;
      fayl : text;
begin
      {massiv o'lchamini va qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl,n);
      for i:=1 to n do
      readln(fayl, A[i]);
{sikl orqali A[] qiymatlarini kiritamiz}
      close(fayl);
       m:=0; i:=1;
while i+m<=n do
begin
      k:=1;
      for j:=i+1 to n do
      if A[j]= A[i] then
begin
      r:= A[j];
```

```
      A[j]:= A[i+k];
      A[i+k]:=r;
      k:=k+1
end;
      if m<k then
begin
      am:=A[i]; m:=k
end;
      i:=i+k
end;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      Write(fayl, Am); { yig'indini chop etish}
      close(fayl);
end.
```

Uzun ketma-ketlik. Berilgan $A[n]$ massiv butun sonlardan tashkil topgan. Nollardan tashkil topgan eng uzun ketma-ketlikning uzunligini aniqlang.

Masala yechimi. Masalan, quyidagi misolda

$$(1, 0, 2, 0, 0, 1, \underbrace{0, 0, 0}, 5, 0, 1)$$

talab etilgan uzunlik 3 ga teng.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
12 1 0 2 0 0 1 0 0 0 5 0 1	3

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va ikkinchi qatorda massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida uzunlik qiymati chiqarilgan bo'ladi.

Demak algoritmda ushbu ketma-ketlikni uzunligini saqlaydigan o'zgaruvchi bilan birgalikda, ularning eng kattasini ham xotirada saqlab turuvchi o'zgaruvchini kiritishimiz kerak. Quyida keltirilgan dasturni tushunib olish endi qiyinchilik tug'dirmaydi.

```
program EngUzun;
const Nmax=100;
```

```
var
      n,i, t,maxSt : integer;
      A:array[1..Nmax] of  integer;
      fayl : text;
begin
{massiv o'lchamini va  qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl,n);
      for i:=1 to n do
      read(fayl, A[i]);
{sikl orqali A[] qiymatlarini kiritamiz}
      close(fayl);
      t:=0;
{t da ketma-ketlikning joriy uzunligi saqlanadi}
      maxSt:=0;
{maxSt  ketma-ketlikning maksimal uzunligi }
      for i:=1 to n do
begin
      if A[i]<>0  then
begin
      if maxSt<t  then maxSt:=t;
      t:=0
{A[i] noldan farqli son,shu bois t=0 deb oldik}
end
      else    t:=  t+1;
{ nolga  duch keldik, shu bois t ni bittaga oshiramiz}
end;
      if maxSt<t  then maxSt:=t;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      Write(fayl, maxSt); {uzunlikni chop etish}
      close(fayl);
end.
```

4.3. Massiv elementlarini tanlash algoritmi va dasturi

Massivlar bilan ishlashda biz uning elementlari bilan ishlashni nazarda tutamiz. Shu bois massiv elementlarini ketma-ket ko'rib chiqishni bilishimiz zarur. Buning umumiy ko'rinishidagi algoritmi quyidagicha bo'lishi mumkin:

1. index := left
2. Mas[index] elementini tanlash
3. index := index+1
4. if index > right then break {ya'ni chegaraga etsak, tugatamiz}
5. 2 – bandga o't.

Bu yerda *left* va *right* massivning chap va o'ng chegaralari. Ushbu algoritm yordamida massiv elementlariga qiymat berish misolini keltiramiz.

Qiymat berish. Berilgan $A[n]$ massivini natural sonlar bilan teskari tartibda to'ldiring.

Masala yechimi. Demak, bo'sh $A[]$ massivini quyidagi n, n-1, ..., 2, 1 sonlar bilan to'ldirish kerak bo'ladi. Bevosita dasturni keltiramiz.

```
program Inverse;
const N = 10;
var
   i, left, right : integer;
   A: array [1..N] of integer;
  fayl : text;
begin
      left:= 1; right :=n;
for i := left to right do
     A[i] := N – i + 1 ;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i := left to right do
      writeln(fayl, A[i]);
      close(fayl);
end.
```

E'tibor bering, natijalar faylda ustun shaklida chiqarilgan.

Maksimum. Berilgan $A[n]$ massivining eng katta elementini aniqlang.

Masala yechimi. Bu erda, agar $A[]=(4, 1, 10, 7, 13, 6)$ bo'lsa, uning eng katta elementi 13 bo'ladi. Dastur g'oyasi bevosita birinchi elementni katta deb olamizda, boshqa elementlar bilan tekshiramiz. Agar undan katta element topilsa, uni xotirada saqlab, keyingi elementlar bilan tekshiramiz va h.k.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
6 4 1 10 7 13 6	13

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va ikkinchi qatorda massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida eng katta element qiymati chiqarilgan bo'ladi.

Dasturning asosiy qismi quyidagicha bo'ladi.

```
program Maxsimum;
const Nmax = 100;
var
      n,i , temp, ArrayMax: integer;
      A:  array [1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va  qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl,n);
      for i:=1 to n do
      read(fayl, A[i]);
{sikl orqali A[] qiymatlarini kiritamiz}
      close(fayl);
      temp := A[1];
      for i := 2 to N do
begin
      if A[i] > temp then temp := A[i];
end;
      ArrayMax := temp;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
```

```
    rewrite(fayl);
    writeln(fayl, ArrayMax);
    close(fayl);
end.
```

Agarda maksimal qiymatni emas, balkim uning o'rni bizni qiziqtirsa, u holda quyidagi algoritmni tavsiya qilamiz.

```
program MaksimumOrin;
const Nmax = 100;
var
    n, i , temp, IndexMax: integer;
    A: array [1..Nmax] of integer;
    fayl : text;
begin
{massiv o'lchamini va qiymatlarni kiriting}
    assign(fayl, 'input.txt');
    reset(fayl);
    readln(fayl,n);
    for i:=1 to n do
    read(fayl, A[i]);
{sikl orqali A[] qiymatlarini kiritamiz}
    close(fayl);
    temp :=1;
    for i := 2 to n do
    if A[i] > A[temp] then temp := i;
    IndexMax := temp;
{chiqarish fayliga natijani chop etamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
    writeln(fayl, IndexMax);
    close(fayl);
end.
```

Agarda massivda eng katta qiymat bir nechta bo'lsa, bu yerda eng birinchisining o'rni aniqlangan bo'ladi.

Keltirilgan algoritmlarning samaradorlik darajasi $O(n)$ tartibli bo'ladi, chunki massiv elementlari faqatgina bir marotaba ko'rib chiqiladi.

Keyingi, ko'p uchraydigan misol, bu massivda biror-bir qiymatni izlashdan iborat.

Izlash. Berilgan $A[n]$ massivining qaysi elementi berilgan *Sample* qiymatiga tengligini aniqlang. Agarda ushbu element mavjud bo'lsa, uning indeksini aniqlash talab etiladi, aksincha -1 chop etilsin.

Masala yechimi. Masalan, $A[]=(3,4,5,7,4,8)$ va *Sample*=4 bo'lsa, qidirilayotgan indeks 2 ga teng bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
6 4 3 4 5 7 4 8	2

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va qidirilayotgan son joylashgan, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida indeks qiymati chiqarilgan bo'ladi.

Quyidagi algoritmda birinchi topilgan natija bilan jarayon tugatiladi

```
program Qidiruv;
const Nmax = 100;
var
      n, i , Search, sample: integer;
      A:  array [1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va boshqa  qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, n, sample);
      for i:=1 to n do
      read(fayl, A[i]);
{sikl orqali A[] qiymatlarini kiritamiz}
      close(fayl);
      i :=1; Search := -1;
      for i :=1 to n do
      if A[i] = sample then
begin
      Search := i;
      Break;
```

```
end;
{chiqarish fayliga natijani chop etamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
    Write(fayl, Search);
    close(fayl);
end.
```

Agarda izlanayotgan qiymat *sample* massivda mavjud bo'lmasa, algoritm natijasi noaniq bo'lishi mumkin. Ushbu kamchilikni bartaraf etish maqsadida biz natija saqlanayotgan *Search* o'zgaruvchiga massiv chegarasidan chiqadigan qiymatni berayapmiz, ya'ni *Search* := -1. Bu jarayon siklgacha aniqlanishi zarur. Demak natija -1 bo'lsa, izlanayotgan qiymat massivda topilmaganligini bildiradi.

4.4. Massivda binar izlash algoritmi va dasturi

Avvalgi paragrafda massiv elementini izlashning eng oddiy usuli bilan tanishib chiqdik. Xulosa sifatida aytish mumkinki, tartiblanmagan massivda zaruriy elementni izlash jarayoni birmuncha ko'p vaqtni talab qiladi, chunki solishtirishlar soni $O(n)$ tartibli bo'ladi. Ketma-ket tartibga solingan massivlarda izlashni ancha tezlashtirish mumkin. Dixotomik izlash, yoki ikkilangan izlash yoki binar izlash usuli eng tezkor usullardan hisoblanadi.

Demak, massiv oldindan tartiblangan bo'lishi kerak. Ammo tartiblashni o'zi talaygina vaqtni talab qiladi! Lekin massiv bir marotaba tartiblanib, bir necha bor undan izlash jarayonlari qo'llanishi zarur bo'lsa, bu holda , albatta, massivni oldindan tartiblab qo'yish maqsadga muvofiq bo'ladi. Shunday qilib, quyidagi ko'rinishdagi masalani yechish talab etiladi.

Binar izlash. N ta elementdan iborat bo'lgan tartiblangan $A[N]$ massivdan *sample* qiymatiga ega bo'lgan elementni o'rnini aniqlash talab etiladi, agarda bunday element topilmasa bu haqda xabar berilsin.

Masala yechimi. Algoritmni tuzishda massiv elementlari o'sish tartibida joylashgan deb faraz qilamiz, masalan, $A[]$=(3, 6, 7, 10, 13, 16, 19, 25) va *sample* qiymatini qidirishda quyidagi algoritm tavsiya etiladi:

1. massiv o'rtasi va undagi element aniqlanadi.
2. aniqlangan element berilgan *sample* bilan taqqoslanadi. Solishtirish natijasi bo'yicha bundan keyin massivning qaysi qismiga murojaat qilish kerakligi aniqlanadi, ya'ni massivning chap yoki o'ng qismi keyingi izlashda ishtirok etmaydi.
3. izlash olib borilayotgan intervalda elementlar soni 1 dan ortiq bo'lsa , u holda 1-bandga o'tamiz.
4. aniqlangan element *sample* ga tengligi tekshiriladi va jarayon tugatiladi.

Masalan, *sample* = 13 uchun quyidagi hisoblashlar bajariladi:
1.1. Berilgan qiymatlar : $A[]$=(3, 6, 7, 10, 13, 16, 19, 25), n=8 va *sample* = 13 uchun $n/2$=4 va $A[4]$=10;
2.1. *sample* > 10, shu bois massivning ikkinchi , ya'ni o'ng qismini ko'rib chiqamiz;
3.1. qolgan elementlar soni bittadan ortiq;
1.2. Qolgan qiymatlar : $A'[]$=(13, 16, 19, 25), n=4 va *sample* = 13 uchun $n/2$=4 va $A'[2]$=16;
2.2. *sample* < 16, shu bois massivning birinchi , ya'ni chap qismini ko'rib chiqamiz;
3.2. qolgan elementlar soni bittadan ortiq;
1.3. Qolgan qiymatlar : $A''[]$=(13, 16), n=2 va *sample* = 13 uchun $n/2$=1 va $A''[1]$=13;
2.3. *sample* qiymati 13 dan katta emas, demak chap qismini ko'rib chiqamiz;
3.3. qolgan elementlar soni bitta;
4. *sample* = $A''[1]$ bo'lganligi sababli jarayonni tugatamiz.

Shunday qilib, uzunligi n bo'lgan massivda kerakli qiymatni topish uchun $O(n \log n)$ tartibli solishtirishlar talab etiladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
8 13 3 6 7 10 13 16 19 25	5

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va qidirilayotgan sonlar joylashgan, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida indeks qiymati chiqarilgan bo'ladi, agar topilmasa, faylga 'Mavjud emas' so'zlari chiqarilsin.

Ushbu algoritm asosida quyidagi dasturni tavsiya etish mumkin

```
program BinaryQidiruv;
const Nmax = 100;
var
      n, i, l, r, middle , sample: integer;
      Search : boolean;
      A: array [1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va boshqa qiymatlarni kiritamiz}
      assign(fayl,'input.txt');
      reset(fayl);
      readln(fayl, n, sample);
      for i:=1 to n do
      read(fayl, A[i]);
{sikl orqali A[] qiymatlarini kiritamiz}
      close(fayl);
      l :=1; r :=n;
while l<r do
begin
      middle := (l + r) div 2;
      if A[middle] < sample then l := middle + 1
      else r := middle
end;
      if A[l] = sample then Search := true
      else Search := false;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      if Search  then  writeln(fayl, l)
      else writeln(fayl, 'Mavjud emas');
      close(fayl);
end.
```

E'tibor bering, biz birinchi uchratgan qiymatni natija sifatida chop etayapmiz. Aslida bunday qiymatlar massivda bir nechta bo'lishi mumkin.

Paskal tilida massivlar o'lchami aniq ko'rsatilishi lozim. Bu esa massiv chegarasidan chiqib ketishni nazorat qilishni dasturda amalga oshirish shart emasligini bildiradi, chunki bevosita xatolik sodir

bo'ladi. Shunday qilib, massiv elementlari orasiga yangi qiymatni kiritish kerak bo'lsa, massivning bitta elementidan voz kechishga to'g'ri keladi.

4.5. Massivda o'rin almashtirish amallari

Orasiga qo'yish. Berilgan *Mas*[*n*] massivining *k* o'rnidan boshlab o'ngga siljitib, bo'sh o'ringa *Elem* qiymatini yozib qo'ying.

Masala yechimi. Misol uchun, quyidagi *Mas*[]=(2, 4, 3, 1, 7) massivida *k*=2 va *Elem*=13 uchun natija quyidagicha bo'ladi: (2, 13, 4, 3, 1). E'tibor bering, massiv chegarasini o'zgartirib bo'lmasligi tufayli massivning n-elementi o'z qiymatini yo'qotdi. Demak, dasturimiz ikki qadamdan iborat bo'ladi. Birinchi qadamda barcha qiymatlarni *k*-o'rindan boshlab 1 ta elementga o'nga siljitamiz, ikkinchidan berilgan *Elem* qiymatini *k*–o'ringa yozib qo'yamiz.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 2 13 2 4 3 1 7	2 13 4 3 1

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami *n*=5, qo'yiladigan *k*=2 pozisiya va shu o'ringa qo'yiladigan *Elem*=13 sonlari joylashgan, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi massiv elementlari qiymatlari chiqarilgan bo'ladi.

Bevosita ushbu algoritm bo'yicha tuzilgan dastur quyidagicha bo'ladi.

```
program Orasiga;
const Nmax = 100;
var
      n, i, k , Elem: integer;
      Mas: array [1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va boshqa qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n, k, Elem);
```

```
    for i:=1 to n do
    read(fayl, Mas[i]);
{sikl orqali Mas[] qiymatlarini kiritamiz}
    close(fayl);
    for i :=n downto k+1 do      Mas[i] := Mas[i-1];
    Mas[k] := Elem ;
{chiqarish fayliga natijani chop etamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
    for i :=1 to n do
    write (fayl, Mas[i], ' ');
    close(fayl);
end.
```

Ko'p hollarda massiv elementlarini teskarilash talab etiladi, shu bois quyidagi misolni ko'rib chiqamiz.

Teskarilash. Berilgan *Mas*[*n*] massivining elementlarini teskarilab, natijani ushbu massivning o'zida saqlang.

Masala yechimi. Masalan, quyidagi (1,3,4,6,9) massivdan (9,6,4,3,1) massivini tashkil qilish talab etiladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 1 3 4 6 9	9 6 4 3 1

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va ikkinchi qatorda massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi massiv elementlari bo'sh katak orqali ajratilib kiritiladi va chiqarilgan bo'ladi.

```
program Teskarilash;
const Nmax = 100;
var
    i, k, n : integer;
    Mas: array [1..Nmax] of integer;
    fayl : text;
begin
{massiv o'lchamini va qiymatlarni kiriting}
```

```
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl,n);
      for i:=1 to n do
      read(fayl, Mas[i]);
{sikl orqali Mas[] qiymatlarini kiritamiz}
      close(fayl);
      for i := 1 to (n div 2) do
 begin
      k:=Mas[i];
      Mas[i]:= Mas[N-i+1];
      Mas[N-i+1]:=k;
end;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i:=1 to n do
      write(fayl, Mas[i], ' ');
      close(fayl);
end.
```

Berilgan ma'lumotlar faylda joylashgan bo'lsa, unda massivni qo'llash shart bo'lmaydi. Misol sifatida quyidagini ko'rib chiqamiz.

Satrni teskarilash. Faylda lotin harflari bilan yozilgan matnning har bir qatori 255 tagacha belgidan iborat. Matndagi lotin so'zlar teskarilab, qolgan belgilar o'z joyida qolishi shart. Natija boshqa faylga chiqarilsin.

Masala yechimi. Misol sifatida quyidagi matnni keltirish mumkin: "Men 5-talaba, sen olim!", dastur natijasi esa quyidagicha bo'ladi: "neM 5-abalat , nes milo!". Ya'ni masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
Men 5-talaba, sen olim!	neM 5-abalat , nes milo!

Bu erda har bir belgi birin-ketin fayldan o'qilib, uni yo'lma-yo'l teskari qo'shib boramiz, chunki $'a'+'b' \neq 'b'+'a'$ bo'ladi.

Bevosita ushbu algoritm bo'yicha tuzilgan dastur quyidagicha bo'ladi:

```
program TeskariSatr;
var s : string;
      c : char;
      faylin, faylout : text;
begin
      assign(faylin, 'input.txt');
      reset(faylin);
      assign(faylout, 'output.txt');
      rewrite(faylout);
      s := '';
while not eof(faylin) do
begin
      read(faylin, c);
      if c in ['A'..'Z', 'a'..'z'] then
      s := c + s       {ya'ni biz harfni o'qib olsak}
      else
begin
      write(faylout, s);
      s := '';
      write(faylout, c);
end;
end;
      write(faylout, s);
      close(faylin);
      close(faylout);
end.
```

4.6. Siklik o'rin almashtirish

Ko'p hollarda massiv boshida joylashgan elementlarni uning oxiridan joylashtirish talab etiladi. Masalani qo'yilishini aniqlab olamiz. Demak, berilgan *Mas*[*n*] massivi $n>1$ elementdan iborat bo'lib, boshidan turgan $k \geq 1$ tasini massivni oxiridan joylashtirish talab etiladi, bunda k ta elementning tartibi buzilishi mumkin emas. Masalan, (4,1,3,4,5,7) massivi uchun $k=2$ bo'lganda, natijada (3,4,5,7,4,1) massivi tashkil qilinadi.

Ushbu masalaning qiziqarli tomonlaridan biri, bu k ning chegaralanmaganligidir, chunki $k>n$ bo'lganda o'rin almashtirishlar

siklik jarayonli bo'lganligi sababli $k'=k \ mod \ n$ ta elementni siljitish kifoya bo'ladi. Ushbu g'oyani quyidagi algoritmda aks ettiramiz:

1. Massivning 1-elementini xotirada saqlaymiz
2. 2-elementdan boshlab massiv elementlarini chap tomonga siljitamiz
3. Massivning oxirgi elementiga xotiradagi qiymatni o'zlashtiramiz
4. Agar o'rin almashtirishlar k dan kichik bo'lsa 1-bandga o'tamiz, aks holda ishni tugatamiz.

Ushbu algoritm haqida qisqacha ma'lumot bilib olishimiz bevosita g'oyani yaxshiroq anglashimizga va undan kengroq foydalanishimizga yordam beradi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*Output.txt*
6 2 4 1 3 4 5 7	3 4 5 7 4 1

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va siljitish soni va ikkinchi qatorda massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi massiv elementlari bo'sh katak orqali ajratilib chiqarilgan bo'ladi.

Endi ushbu algoritm dasturini keltiramiz

```
program SiklikSiljitish;
const Nmax = 100;
var
      n, i, j, k , temp: integer;
      Mas: array [1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl,n,k);
      for i:=1 to n do
      read(fayl, Mas[i]);
{sikl orqali Mas[] qiymatlarini kiritamiz}
      close(fayl);
      for j := 1 to (k mod n) do
begin
      temp := Mas[1];
```

```
    for i :=2 to n do
    Mas[i-1] := Mas[i];
    Mas[n] := temp;
end;
{chiqarish fayliga natijani chop etamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
    for i :=1 to n do
    write(fayl, Mas[i], ' ');
    close(fayl);
end.
```

Ushbu algoritm bizdan *(k mod n) * (n+1)* ta amallarni bajarishni talab qiladi. Lekin bittalab siljitish o'rniga bir yo'la *k'* qadamga chapga siljitishning o'zi kifoya bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
6 8 4 1 3 4 5 7	3 4 5 7 4 1

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va siljitish soni va ikkinchi qatorda massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi massiv elementlari bo'sh katak orqali ajratilib chiqarilgan bo'ladi.

Bu erda quyidagi dastur tuziladi:

```
program SiklikKtaSiljitish;
const Nmax = 100;
var
    n, i, j, k , k1: integer;
    Mas, temp:  array [1..Nmax] of integer;
    fayl : text;
begin
{massiv o'lchamini va  qiymatlarni kiriting}
    assign(fayl, 'input.txt');
    reset(fayl);
    read(fayl,n,k);
    for i:=1 to n do
    read(fayl, Mas[i]);
```

```
{sikl orqali Mas[] qiymatlarini kiritamiz}
      close(fayl);
      k1:= k mod n;
      for i := 1 to k1 do
      temp[i] := Mas[i];
      for j :=k1+ 1 to n do
      Mas[j-k1] := Mas[j];
      for i := 1 to k1 do
      Mas[n-k1+i] := temp[i];
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i :=1 to n do
      write(fayl, Mas[i], ' ');
      close(fayl);
end.
```

Ushbu algoritmning siklларidagi bajariladigan amallar soni $O(n+k1)$ tartibli bo'ladi, ya'ni ushbu algoritm tezkor hisoblanadi. Lekin e'tibor bering, bu erda biz qo'shimcha *temp*[] massivini kiritdik.

Ushbu sohaga taaluqli yana bir misolni ko'rib chiqamiz.

Demak, berilgan massivda $a[1]..a[m+n]$ birinchi bo'lakni, ya'ni $a[1]..a[m]$ larni oxiriga, ikkinchi bo'lakni $a[m+1]..a[m+n]$ massiv boshidan joylashtirish talab etiladi.

Agar birinchi bo'lakni A va ikkinchi bo'lakni B deb atasak, u holda umumiy berilgan massiv bu $A+B$ bo'ladi. Masalani sharti bo'yicha $B+A$ massivini tuzish talab etiladi. Faraz qilamiz A ning uzunligi B dan kichik bo'lsin. Endi B qismining ichidan A uzunligidagi qismini ajratib olamiz, ya'ni $B=B1+B2$. Demak, $A+B1+B2$ dan $B1+B2+A$ ni tuzamiz va buni quyidagicha amalga oshiramiz, ya'ni

$$A+B1+B2 \rightarrow B1+A+B2$$

Endi A bilan $B2$ larni o'rnini almashtiramiz. E'tibor bering, biz endi A bilan $B2$ bo'laklarini o'rnilarini almashtirishimiz kerak, ya'ni oldingi masalaga qaytdik, faqatgina massivlar uzunligini kamaytirishga erishdik. Endi yana ushbu algoritmni davom ettiramiz. E'tibor bering, ushbu algoritm qaysidir jihati bilan Yevklid algoritmiga o'xshash.

Quyidagi ma'lumotlar uchun dasturni tekshirib ko 'ring:

input.txt	*output.txt*
6 2 4 1 3 4 5 7	3 4 5 7 4 1

```
program SiklikEvklidSiljitish;
const Nmax = 100;
var
      i, j,r ,p,q, s, k , n : integer;
      temp, pnew, qnew, rnew: integer;
      a:  array [1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va  qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl,n, k);
      for i:=1 to n do
      read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
      close(fayl);
      p := 0; q := k; s :=  n; r :=  n;
{a[p+1]..a[q] va a[q+1]..a[s], o'rnini almashtiramiz}
while (p <> q) and (q <> s) do
begin
{ikala qism bo'sh  emas}
      if (q - p) <= (s - q) then
begin
{a[p+1]..a[q] va a[q+1]..a[q+(q-p)]larni almashtiramiz}
      for i :=p+1 to q  do
begin
      temp:= a[i];  a[i]:= a[q+1+(i-p-1)];
      a[q+1+(i-p-1)]:=temp;
end;
      pnew := q; qnew := q + (q - p);
      p := pnew; q := qnew;
end
         else
begin
```

```
{a[q-(r-q)+1]..a[q] va a[q+1]..a[r]larni almashtiramiz}
    for i := q-(r-q)+1 to q  do
begin
    temp:= a[i];  a[i]:= a[q+1+(i- q+(r-q)-1)];
    a[q+1+(i- q+(r-q)-1)]:=temp;
end;
    qnew := q - (r - q); rnew := q;
    q := qnew; r := rnew;
end;
end;
{chiqarish fayliga natijani chop etamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
    for i :=1 to n  do  write (fayl, a[i], ' ');
    close(fayl);
end.
```

Algoritmning samaradorlik darajasi $O(n)$ tartibli bo'ladi, chunki har safar massiv bo'lagining uzunligidagi amallar bajariladi.

Yuqorida keltirilgan *Teskarilash* dasturini *Teskarilash*(*Massiv,n,m*) protsedurasiga aylantirsak, juda qiziq dasturni taklif qilish mumkin.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
6 2 4 1 3 4 5 7	3 4 5 7 4 1

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va siljitish soni va ikkinchi qatorda massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi massiv elementlari bo'sh katak orqali ajratilib chiqarilgan bo'ladi.

```
program TeskarilashNew;
const Nmax = 100;
var
    n, i, k : integer;
    Mas, temp:  array[1..Nmax] of integer;
    fayl : text;
procedure Teskarilash( m, n:integer);
{ Mas[] elementlari m dan boshlab n gacha teskariladi }
var
```

```
    i, k : integer;
begin
    for i := m to ((n+m-1) div 2) do
begin
    k:= Mas[i];  Mas[i]:= Mas[N-i+m]; Mas[N-i+m]:=k;
end;
end;
{Asosiy dastur}
begin
{massiv o'lchamini va  qiymatlarni kiriting}
    assign(fayl, 'input.txt');
    reset(fayl);
    readln(fayl,n,k);
    for i:=1 to n do
    read(fayl, Mas[i]);
{sikl orqali Mas[i] qiymatlarini kiritamiz}
    close(fayl);
    Teskarilash( 1, k);
    Teskarilash( k+1, N);
    Teskarilash( 1, N);
{chiqarish fayliga natijani chop etamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
    for i :=1 to n do
    write (fayl, Mas[i], ' ');
    close(fayl);
end.
```

Haqiqatan, $a_1,a_2,...,a_n$ massivida oldiniga k ta elementni teskarilaymiz $a_k a_{k-1},...,a_1$, $a_{k+1},a_{k+2},...,a_n$. Endi k+1 dan boshlab qolgan elementlarni teskarilaymiz, ya'ni $a_k a_{k-1},...,a_1$, $a_n,a_{n-1},...,a_{k+1}$. Endi to'liq teskarilaymiz, ya'ni $a_{k+1},a_{k+2},...,a_n$, $a_1,..,a_{k-1},a_k$. Bajariladigan amallar soni $O(2n)$ tartibli bo'ladi.

Izlash. Butun sonlardan tashkil topgan va o'sib borish tartibida joylashgan 3 ta massiv berilgan , ya'ni $x[1] \le ... \le x[p]$, $y[1] \le ... \le y[q]$, $z[1] \le ... \le z[r]$.

Ushbu massivlarda bir-biriga teng bo'lgan bitta son mavjud, ya'ni $x[i]=y[j]=z[k]$. Ushbu sonni aniqlang va amallar soni $p + q + r$ tartibli bo'lsin.

Masala yechimi. Agar sikl orqali yechimni qidirmoqchi bo'lsak, unda amallar soni $O(p \cdot q \cdot r)$ tartibli bo'ladi, bu esa masala shartiga to'g'ri kelmaydi. Lekin massivlar o'sish tartibida joylashgan, shu bois ularning chap chegaralarini tekshirish va ularni siljitish orqali masalani yechimini aniqlash mumkin. Ushbu g'oyani quyidagi misolda ko'rib chiqamiz, masalan $x[]=(1,4,5,7)$, $y[]=(3,7,8)$ va $z[]=(2,3,5,6,7)$ bo'lsin, bu erda $x[1]<y[1]$ bo'lganligi sababli, $x[1]$ elementini chetlab o'tamiz.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
4 3 5 1 4 5 7 3 7 8 2 3 5 6 7	7

Bu yerda *input.txt* faylida birinchi qatorda massivlarning o'lchamlari *p, q, r* va ikkinchi qatorda esa mos ravishda massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida teng elementlar qiymati yoki, aksincha, *'Teng element mavjud emas'* deb chiqarilgan bo'ladi.

Dastur esa quyidagi ko'rinishda bo'ladi:

```
program Izlash;
const
      Nmax = 100;
      Mmax = 100;
      Kmax = 100;
var
      i, j,r ,p,q, r1 ,p1,q1: integer;
      x,y,z: array [1..Nmax+Mmax+Kmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, p, q, r);
```

```
      for i:=1 to p do read(fayl, x[i]);
{sikl orqali x[] qiymatlarini kiritamiz}
      for i:=1 to q do read(fayl, y[i]);
{sikl orqali y[] qiymatlarini kiritamiz}
      for i:=1 to r do read(fayl, z[i]);
{sikl orqali z[] qiymatlarini kiritamiz}
      close(fayl);
{chiqarish faylini ochip qo'yamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      p1:=1; q1:=1; r1:=1;
{massivlarning chap chegarasi}
{har bir massivdan quyidagi bo'laklar tekshiraladi }
{  x[p1]..x[p], y[q1]..y[q],z[r1]..z[r]  }
while not ((x[p1]=y[q1]) and (y[q1]=z[r1])) do
begin
      if x[p1]<y[q1] then
begin
      p1:=p1+1;
end
      else if y[q1]<z[r1] then
begin
      q1:=q1+1;
end
      else if z[r1]<x[p1] then
begin
      r1:=r1+1;
end
end;
{Bu erga kelsak, demak x[p1] = y[q1] = z[r1]}
      writeln (fayl, x[p1]);close(fayl);
end.
```

Chegara. Bizga massiv $a[n]$ va b soni berilgan bo'lsin. Massivda elementlar o'rnini almashtirish orqali, qandaydir chegaracha b dan kichik yoki teng, chegaradan o'ng tomonda esa b dan katta yoki teng elementlar joylashgan bo'lsin.

Masala yechimi. Masalan, $a[]=(7,1,4,2,8)$ va $b=6$ bo'lsin, u holda quyidagi natijaga kelish mumkin:

(2, 1, 4, 7, 8)
↑
chegara

Bu yerda, das...ida ko'p uchraydigan g'oyani taklif qilamiz, ya'ni $a[]$ massivini ikki tomondan boshlab tekshirishni amalga oshiramiz.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 6 7 1 4 2 8	2 1 4 7 8 3

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami $n=5$, chegara qiymati $b=6$, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida natijaviy massiv elementlari qiymatlari va chegara chiqarilgan bo'ladi.

Bevosita ushbu algoritm bo'yicha tuzilgan dastur quyidagicha bo'ladi.

```
program Izlash;
const Nmax = 100;
var
      i, b, n ,L,R,temp : integer;
      a: array [1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va boshqa qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n, b);
      for i:=1 to n do
      read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
      close(fayl);
      L:=0; R:=n; {chap va o'ng tomondan kelamiz}
```

```
{shart bo'yicha  a[1]..a[L]<=b; a[R+1]..a[n]>=b va L=R} {bo'lsa
ishni to'xtatamiz}
while L <> R do
begin
      if a[L+1] <= b then
begin
      L:=L+1;
end
      else
      if a[R]>=b then
begin
      R:=R-1;
end
      else
begin
{ya'ni a[L+1]>b  va  a[R]<b  bo'ldi}
{ demak o'rnini almashtiramiz a[L+1] va  a[R] }
      temp:= a[L+1]; a[L+1]:= a[R]; a[R]:= temp;
      L:=L+1; R:=R-1;
end;
end;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i :=1 to n do  write (fayl, a[i], ' ');
      writeln(fayl);
      write(fayl, L);
      close(fayl);
end.
```

Nollar. Sonli $A[n]$ massivda nolga teng bo'lmagan elementlarni, tartibi buzilmagan holda, massivning boshidan yozilib, nollar oxiridan yozilsin. Qo'shimcha massiv kiritish mumkin emas.

Masala yechimi. Demak, (3,1,0,1,2,0) dan (3,1,1,2,0,0) ni tashkil qilishimiz kerak. Buning uchun ketma-ket o'rin almashtirilsa natijaga erishamiz. Haqiqatan ham, massiv bo'yicha o'tib, birinchi nolni uchratib, uni yonidagi element bilan almashtiramiz, ya'ni

(3,1,0,1,2,0) → (3,1,1,0,2,0) → (3,1,1,2,0,0)

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
6 3 1 0 1 2 0	3 1 1 2 0 0

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=6, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida natijaviy massiv elementlari qiymatlari chiqarilgan bo'ladi.

Bevosita ushbu algoritm bo'yicha tuzilgan dastur quyidagicha bo'ladi.

```
program Nollar;
const Nmax=100;
var
      n, i, j : integer;
      A:array[1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va massiv qiymatlarini kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, n);
      for i:=1 to n do read(fayl, A[i]);
{sikl orqali A[] qiymatlarini kiritamiz}
      close(fayl);
      j:=0;
      for i:=1 to n do
      if A[i]<>0  then
begin
      j:=j+1;
      if i<>j  then
begin
      A[j]:= A[i]; A[i]:= 0;
end;
end;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i :=1 to n do  write (fayl, A[i], ' ');
```

```
    close(fayl);
end.
```

Teskarisi teng. Berilgan $A[n]$ massivida eng uzun qism massivni uzunligini aniqlang, unda 1-element k-elementga, 2-element $(k-1)$-elementga va hokazo shu tariqa boshqa elementlar ham bir-biriga teng bo'lsin.

Masala yechimi. Masalan, quyidagi (1,4,5,5,4,0) massiv uchun (4,5,5,4) qism massiv qo'yilgan shartni bajaradi. Masalaning sharti bo'yicha qidiralayotgan qism massiv simmetriya xususiyatiga ega. Demak, ushbu qism massiv uchun "markaz" tushunchasini kiritamiz. Lekin "markazni" bitta son bilan ifodalamaymiz, chunki 2 ta element "markazida" massiv elementi yo'q. Shunday "markazni" 2 ta element nomerlari LN va PN bilan belgilaymiz. Bunda $LN < PN$ va ular quyidagi qiymatlarni qabul qilishi mumkin

$$(LN, PN) = (1,2), (1,3), (2,3), (2,4)\ldots$$

Har bir variantni alohida tahlil qilish uchun chap va o'ng tomon uchun o'zgaruvchilarni L va P ni kiritamiz, ularning qiymati boshida quyidagicha bo'ladi: $L = LN$ va $P = PN$.

Keyin har bir qadamda chapga va o'ngga siljitamiz, ya'ni $L = L-1$ va $P = P+1$. Agarda $A[L] \neq A[P]$ bo'lsa, u holda ushbu jarayonni to'xtatamiz. Bundan tashqari agarda L yoki P massiv chegarasidan chiqib ketsa, bu holda ham jarayonni to'xtatamiz. Natijada (LN, PN) "markaziga" ega qism massiv uzunligi $M=P-L-1$ ga teng bo'ladi. Ushbu M larning ichidan maksimal Max qiymatini aniqlash talab etiladi. Har bir qadamda aniqlangan Max qiymatini, bevosita massiv bo'yicha o'ng tomonga qarab o'sishini quyidagi tengsizlikga binoan chegaralab qo'yish ham mumkin: $2 \cdot (N-PN+1) \leq Max$, ya'ni ko'rilayotgan "markaz" Max qiymatini oshirishga imkon bermaydi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
6 1 4 5 5 4 0	4

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami $n=6$, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida qism massiv uzunligi chiqarilgan bo'ladi.

Bevosita ushbu algoritm bo'yicha tuzilgan dastur quyidagicha bo'ladi.

```
program TeskarisiTeng;
const Nmax=100;
var
      i,Ln,Pn,L,P,m,max,n : integer;
      z: boolean;
      A:array[1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va elementlarini kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, n);
      for i:=1 to n do
      read(fayl, A[i]);
{sikl orqali A[] qiymatlarini kiritamiz}
      close(fayl);
      max:=1;   z:=TRUE;  Ln:=1;   Pn:=2;
while 2*(n-Pn+1)+1>max do
begin
      L:=Ln;   p:=Pn;
while (L>=1) AND (P<=n) AND (A[L]=A[P]) do
begin
      L:=L-1; P:=P+1
end;
      m:=P-L-1;
      if (max<m) then max:=m;
      if z then Pn:=Pn+1 else  Ln:=Ln+1;
      z:=NOT z;
end;
{chiqarish fayliga uzunlikni chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      write (fayl, max);
      close(fayl);
end.
```

4.7. Ikki o'lchovli massivlarning ishlatilishiga doir algoritm va dasturlar

Ikki o'lchovli massivlarni biz oddiy jadval bilan qiyoslaymiz. Jadvallarni to'ldirishda uning qator va ustun kesishmasida qiymat berilishi hammaga ayon. Massivlar bilan xuddi shunday yo'l tutamiz va quyidagi oddiy dasturni keltiramiz.

Ko'paytirish jadvali. Massiv elementlarini to'ldirilishini ko'paytirish jadvali misolida ko'rib chiqamiz. Berilgan bo'sh *MulTable*[*M,N*] massivida qator nomerini ustun nomeriga ko'paytirib, natija ushbu massiv elementiga kiritilsin.

Masala yechimi. Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
2 2	1 2 2 4

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchamlari m=2 va n=2 bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi massiv elementlari qiymatlari chiqarilgan bo'ladi.

Bevosita ushbu algoritm bo'yicha tuzilgan dastur quyidagicha bo'ladi.

```
program MyMulTable;
const
      Mmax=10;
   Nmax=10;
var  i,j,M,N : integer;
      MulTable : array[1..Mmax,1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamlarini kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, m, n);
      close(fayl);
{chiqarish faylini ochamiz va  natijalarni chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
```

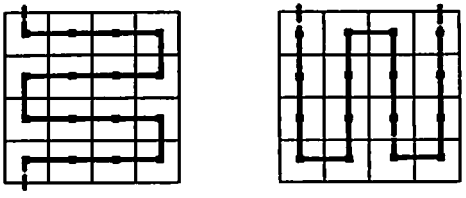
```
    for i := 1 to M do
begin
    for j := 1 to N do
begin
    MulTable[i, j] := i*j;
    write(fayl,  MulTable[i, j], ' ')
end;
    writeln(fayl); {bir qator pastga tushamiz}
end;
    close(fayl);
end.
```

Bajariladigan amallar soni $O(MN)$ (yoki $M=N$ bo'lsa $O(N^2)$) tartibli bo'lishi sikl operatorlari ichma-ich joylashganligidan kelib chiqadi.

Shunday qilib massiv elementlarini chapdan o'ngga va yuqoridan pastga qarab to'ldirib chiqdik. Lekin ushbu aylanib o'tish "uzlukli" hisoblanadi, "uzluksiz" aylanib o'tishlar 1-rasmda keltirilgan. "Uzluksiz" deganda biz bir katakdan yonma-yon turgan katakga o'tishni tushunamiz. Ikki o'lchovli massivlarning aylanib o'tishlarning quyidagi turlari mavjud:

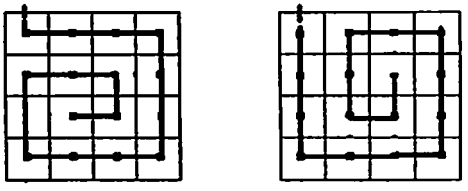
- gorizontal
- vertikal
- bosh diagonal
- ikkinchi diagonal
- spiral

Gorizontal va vertikal aylanib o'tish 1-rasmda keltirilgan. Bunday aylanib o'tishni biz deyarli har kun uchratib turamiz. Masalan, o'quv binosidagi qavatidagi auditoriyalarni bir boshdan chap tomondan ko'rib kelamiz va qaytishda esa o'ng tomondagilarni ko'rib chiqamiz. Bundan tashqari, bir elementdan ikkinchi elementga o'tishda, ushbu koordinatalar faqatgina bittaga farq qiladi.



1-rasm

Spiral aylanib o'tish soat mili yo'nalishida harakat yoki qarama-qarshi yo'nalishda harakatlanishda bo'ladi. Uni quyidagi 2-rasmda keltiramiz.



2-rasm

Spiral. Berilgan $A[n,n]$ massivini 1, 2 ,..., n^2 sonlar bilan spiralning soat mili yo'nalishi bo'yicha to'ldirib chiqing.

Masala yechimi. Ushbu spiralni bo'laklarga bo'lamiz, ya'ni gorizontal bo'yicha chapdan o'ngga, vertikal bo'yicha yuqoridan pastga, gorizontal bo'yicha o'ngdan chapga, vertikal bo'yicha pastdan yuqoriga. Har bir bo'laklangan yo'nalish bo'yicha massivni to'ldirishni alohida bajaramiz. Jarayon, oxirgi kiritilgan k qiymat n^2 ga teng bo'lganga qadar davom etadi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
3	1 2 3 8 9 4 7 6 5

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=2 va *output.txt* faylida yangi massiv elementlari qiymatlari chiqarilgan bo'ladi.

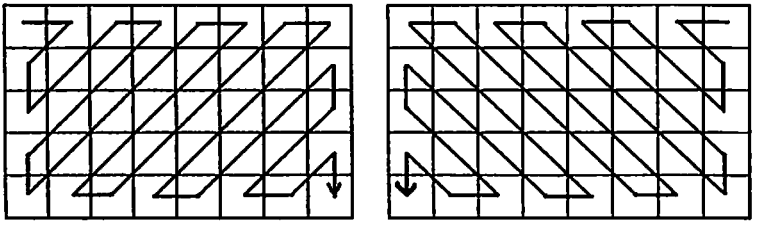
Dastur quyidagi ko'rinishda bo'ladi.

program Spiral;

```
const Nmax=20;
var
      k, i,  j, n : integer;
      A:array[1..Nmax,1..Nmax] of  integer;
      fayl : text;
procedure Keyingisi;
begin
      if k<n*n  then
begin
      k:=k+1;
      A[i,j]:=k;
end;
end;
begin
{massiv o'lchamini  kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      close(fayl);
k:=0; i:=1; j:=1;
repeat
repeat
begin
      Keyingisi;
   j:=j+1;
end
until (i+j>n) ;
repeat
begin
      Keyingisi;
      i:=i+1;
end
until (i>=j);
repeat
begin
      Keyingisi;
   j:=j-1;
end
until (i+j<=n+1);
```

```
repeat
begin
      Keyingisi;
      i:=i-1;
end
until (i<=j+1) ;
until k=n*n;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i:=1 to n do
begin
      for j:=1 to n do write (fayl, A[i,j], ' ');
      writeln(fayl);
end;
      close(fayl);
end.
```

Diagonallar bo'yicha aylanib o'tish 3-rasmda keltirilgan.



3-rasm

Ushbu aylanib o'tishning qiziqarli tomoniga nazar tashlasak. Chapdagi rasmda diagonal bo'yicha yurish qiladigan bo'lsak qator va ustun indekslarning yig'indisi o'zgarmasdir, ya'ni $i+j=const$. O'ng tomondagi rasmdagi variantda $i-j=const$ bo'ladi.

Barcha ko'rib chiqilgan variantlarda aylanib o'tishda bajariladigan amallar soni $O(n^2)$ tartibli bo'ladi.

Diagonal. Berilgan $A[m,n]$ massivida $i-j= k$ tenglikni qanoatlantiruvchi barcha i,j lar uchun $A[i,j]$ larning yig'indisini hisoblang. Umumiy holda k manfiy ham bo'lishi mumkin.

Masala yechimi. Albatta barcha i,j larni sikl orqali ko'rib, shartni qanoatlantirganda yig'indini hisoblashimiz mumkin, lekin bu ko'p vaqt talab etadi. Shu bois, yangi p va q qiymatlarini quyidagicha aniqlaymiz: $p=max(1,1-k)$ va $q=min(n,m-k)$. Natijada sanab o'tiladigan elementlar $A[k+j,j]$ uchun yig'indini hisoblash kerak bo'ladi va bu yerda $j=p, p+1, \ldots, q$ qiymatlarini qabul qiladi. E'tibor bering, birinchi navbatda agar $p>q$ bo'lsa, u holda natija nol bo'ladi va ikkinchidan, biz faqatgina yig'indida ishtirok etadigan $A[i,j]$ elementlarini ko'rib chiqqan bo'lamiz.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
2 2 1 2 4 3 1	3

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami $m=2$, $n=2$ va $k=1$ sonlari joylashgan, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida masala shartidagi yig'indining qiymati chiqarilgan bo'ladi.

Dastur esa quyidagi ko'rinishda bo'ladi.

```
program Diagonal;
const
      Nmax=10;
      Mmax=10;
var
      m,n,s,j,k,p,q : integer;
      A:array[1..Mmax,1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va boshqa qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, m, n, k);
{sikllar orqali A[] qiymatlarini kiritamiz}
      for p:=1 to m do
begin
```

```
      for j:=1 to n do
begin
      read(fayl, A[p,j])
end;
      readln(fayl);
end;
      close(fayl);
      if k>0  then p:=1 else p:=1-k;
      if k+n<m  then q:=n else q:=m-k;
      s:=0;
      for j:=p to q do  s:=s+ A[k+j,j];
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      write (fayl, s);
      close(fayl);
end.
```

Qism kvadrat massiv. Berilgan $A[n,n]$ massivi 0 va 1 lardan tashkil topgan. Uning ichida faqatgina 1 lardan tashkil topgan eng katta kvadrat massivini toping.

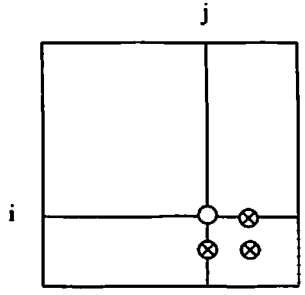
Masala yechimi. Masalan, quyidagi massiv uchun

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

natija chiziqchalar ichiga olingan.

Ushbu misolni yechishda quyidagicha yo'l tutamiz. Massiv elementlarini ko'rib chiqishni qator bo'yicha o'ngdan chapga, ustun bo'yicha pastdan yuqoriga qarab ko'rib chiqamiz. Bu yerda (i,j) koordinatasi topilgan qism kvadratning chap yuqori nuqtasi deb qabul qilamiz. Bundan, $A[i,j]$ elementiga beriladigan qiymat – bu kvadratlarning eng kattasini uzunligini kiritamiz. Demak, $A[i,j]$ elementi ko'rilayotganda i dan pastdagi va j dan o'ng tomondagi barcha elementlar ko'rib chiqilgan bo'ladi. Agar $A[i,j] = 0$ bo'lsa, demak kvadrat tuzib bo'lmaydi, shu bois uning qiymati o'zgarmaydi, ya'ni

$A[i,j]=0$. agar $A[i,j]=1$ bo'lsa, quyidagi nuqtalarni ko'rib chiqamiz $A[i,j+1]$, $A[i+1,j+1]$, $A[i+1,j]$.



Shunday qilib, (i,j) nuqtasidan tuzilgan kvadratning uzunligi belgilangan nuqtalardan tuzilgan kvadratlar uzunligidan ko'pi bilan bittaga ko'p bo'lishi mumkin. Shu bois ushbu nuqtadan tuzilgan kvadrat oldingilarining eng kichigidan bittaga ko'p bo'ladi, ya'ni

$$A[i,j]=min\{A[i,j+1], A[i+1,j], A[i+1,j+1]\}+1.$$

Endi dasturni tuzish muammo tug'dirmaydi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
4 0 0 0 1 0 1 1 0 1 1 1 1 0 1 0 1	2

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami $n=4$, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida faqatgina qism massiv o'lchami chiqarilgan bo'ladi.

```
program QismKvMassiv;
const Nmax=10;
      var n, i, j , MaxDim: integer;
      A : array[1..Nmax,1..Nmax] of integer;
      fayl : text;
function Min3(a,b,c:integer) : integer;
var x : integer;
begin
if (a<=b) then x:=a Else x:=b;
      Min3:=x;
```

```
      if (c<=x) then    Min3:=c;
end;
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl,  n);
{sikllar orqali A[] qiymatlarini kiritamiz}
      for i:=1 to n do
begin
      for j:=1 to n do
begin
      read(fayl, A[i,j])
end;
      readln(fayl); {keyingi qatorga o'tamiz}
end;
      close(fayl);
      MaxDim:=0;{kvadrat tomoni}
      for i:=n-1 downto 1 do
      for j:=n-1 downto 1 do
      if a[i,j]<>0 then
 begin
      a[i,j]:=min3(a[i,j+1],a[i+1,j+1],a[i+1,j])+1;
      if a[i,j]>MaxDim then MaxDim:= a[i,j]
 end;
 {Eng katta kvadratni o'lchamini chop etamiz }
      assign(fayl, 'output.txt');
      rewrite(fayl);
      write (fayl, MaxDim);
      close(fayl);
 end.
```

4 ta element yig'indisi. $A[n,n]$ massiv elementlari 0, 1, 5 yoki 11 ga teng. Ushbu massiv elementlaridan tashkil topgan quyidagi to'rtliklarni ko'rib chiqamiz:

$$A[i,j]\,,\ A[i+1,j]\,,\ A[i,j+1]\,,\ A[i+1,j+1].$$

Agar ushbu to'rtlikdagi sonlar har xil bo'lsa, uni "a'lo to'rtlik" deb ataymiz. Ushbu massivda nechta "a'lo to'rtlik" borligini aniqlang.

Masala yechimi. Quyidagi massivda faqatgina bitta "a'lo to'rtlik" mavjud bo'lib, uni chiziqchalar bilan ajratib qo'yganmiz:

$$\begin{pmatrix} 0 & 1 & 5 \\ 0 & 0 & 5 \\ 0 & 1 & 11 \end{pmatrix}$$

Ushbu masalaning qiziqarli jihati shundan iboratki , shartda keltirilgan qiymatlarga ham e'tibor berish zarur. Berilgan sonlarning yig'indisi 17 ga teng. Hech qanday boshqa kombinatsiya bilan 17 ni tashkil qilish mumkin emas. Shunday qilib, biz faqatgina 4 ta massiv elementlarining yig'indisini tekshirsak kifoya.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
3 0 1 5 0 0 5 0 1 11	1

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=3, ikkinchi qatordan boshlab massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida faqatgina "a'lo to'rtlik" lar soni chiqarilgan bo'ladi.

Dastur esa quyidagi ko'rinishda bo'ladi.

```
program ElementSumma;
const Nmax=10;
var
      n,s,i,j : integer;
      A:array[1..Nmax, 1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va boshqa qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, n);
{sikllar orqali A[] qiymatlarini kiritamiz}
      for i:=1 to n do
begin
      for j:=1 to n do
begin
```

```
      read(fayl, A[i,j])
end;
      readln(fayl); {keyingi qatorga o'tamiz}
end;
      close(fayl);
      s:=0;
      for i:=1 to n-1 do
      for j:=1 to n-1 do
      if (A[i,j]+A[i,j+1]+A[i+1,j]+A[i+1,j+1]=17) then
      s:=s+1;{"a'lo to'rtlik"lar sonini chop etamiz }
      assign(fayl, 'output.txt');
      rewrite(fayl);
      write (fayl, s);
      close(fayl);
end.
```

Ikki marotaba tartiblangan. Berilgan $A[m,n]$ sonli massiv elementlari quyidagicha tartiblangan

$A[i,1] \le A[i,2] \le \ldots$ barcha $i=1,\ldots,m$ lar uchun

va

$A[1,j] \le A[2,j] \le \ldots$ barcha $j=1,\ldots,n$ lar uchun.

Ushbu massivda berilgan k soniga teng element borligini aniqlang va uning indekslarini chop eting. Bunday element topilmasa, u holda "not" so'zini chop eting.

Masala yechimi. Masala shartini qanoatlantiruvchi quyidagi massiv uchun ikki yo'nalish bo'yicha o'sish tartibi mavjud:

$$\begin{pmatrix} 1 & 7 & 7 \\ 3 & 9 & 14 \\ 5 & 11 & 21 \end{pmatrix}$$

Ushbu massivda, masalan, $k=11$ ga teng element mavjudligini aniqlash talab etiladi.

Albatta barcha $A[i,j]$ elementlarni k ga tengligini tekshirish qiyin emas, lekin massivning tartiblanligini ham inobatga olish zarur. Bu yerda quyidagi shartlardan iborat algoritmni tavsiya etish mumkin, ya'ni (bu erda tekshiruvni 1-qatorning oxiridan boshlaymiz)

1) agar $A[i,j] = k$ bo'lsa, u holda natija topildi;

2) agar $A[i,j] < k$ bo'lsa, u holda $i=i+1$;

3) agar $A[i,j] > k$ bo'lsa, u holda $j=j-1$;

Agar $i \le m$ va $j \ge 1$ bo'lsa, u holda jarayon davom etadi. Aks holda bunday son k topilmagan bo'ladi. Ushbu algoritmni quyidagi misolda ko'rib chiqamiz.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 1 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \end{bmatrix} \text{ va } k=13 .$$

Demak, boshida $i=1$ va $j=5$, bunda $A[1,5] = 5 < k=13$, ya'ni 1-qatorda k ga teng element yo'q. 2-shart bo'yicha $i=i+1=1+1=2$. Keyingi qadamga o'tamiz, ya'ni

$A[2,5] = 10 < k=13$, ya'ni 2-qatorda k ga teng element yo'q. 2-shart bo'yicha $i=i+1=2+1=3$. Keyingi qadamga o'tamiz, ya'ni $A[3,5] = 15 > k=13$, ya'ni 3-qatorda k ga teng element bo'lishi mumkin, lekin u 5-ustunda emas, endi 3-shart bo'yicha $j=j-1=5-1=4$. Tekshiramiz $A[3,4] = 14 > k=13$, demak 4-ustunda ham natija yo'q, 3-shart bo'yicha $j=j-1=4-1=3$, bunda $A[3,3] = 13 = k=13$.

Shunday qilib, natija aniqlandi. Ushbu algoritmning samaradorlik darajasi $O(n+m)$ tartibli bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
4 5 13 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	3 3

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchamlari $m=4$, $n=5$ va qidiralayotgan $k=13$ bo'ladi, ikkinchi qatordan boshlab massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida tenglik bajarilgan element indekslari chiqarilgan bo'ladi.

Endi ushbu algorim bo'yicha dasturni tuzamiz.

```
program IkkiMarotabaTartiblangan;
const Mmax=10;
    Nmax=10;
var m,n,i,j,k,r : integer;
    shart : boolean;
```

```
      A:array[1..Mmax, 1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va element qiymatlarini kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, m, n, k);
{sikl orqali A[] qiymatlarini kiritamiz}
      for i:=1 to m do
begin
      for j:=1 to n do
begin
      read(fayl, A[i,j])
end;
      readln(fayl);{keyingi qatorga o'tamiz}
end;
      close(fayl);
      i:=1; j:=n; shart:=true;
      r:=0; {natija belgisi}
while shart do
begin
      shart:=((i<=m) AND (j>=1));
      if A[i,j]=k then
begin
      shart:=false; r:=1;
end;
      if A[i,j]<k then  i:=i+1;
      if A[i,j]>k then  j:=j-1;
end;
{natijani  chop etamiz }
      assign(fayl, 'output.txt');
      rewrite(fayl);
      if r=0 then writeln(fayl, 'not')
           else writeln(fayl, i,j);
      close(fayl);
end.
```

To‘g‘ri to‘rtbukchakli qismiy massiv. Berilgan $A[n,m]$ massivi 0 va 1 lardan tashkil topgan. Uning ichida faqatgina 1 lardan tashkil topgan eng katta to‘g‘ri to‘rtbukchakli massivni toping.

Masala yechimi. Demak , bizda quyidagi ko‘rinishdagi $A[n,m]$ massiv berilgan bo‘lsin:

		j					
		1	2	3	4	5	$= m$
	1	0	0	0	0	0	
	2	0	0	0	0	0	
	3	0	1	1	1	0	
$i =$	4	0	1	1	1	0	
$N =$	5	0	0	0	0	0	

Har bir i-qator uchun $p[1..m]$ massivini kiritamiz, bunda $p[j]$ bu j-ustun bo‘yicha (i,j) elementdan boshlab, yuqoriga qarab bo‘lgan birlar soni bo‘ladi. Masalan
$p[j]=0$ bo‘lsa, u holda $a[i,j]=0$, agarda $p[j]=u>0$ bo‘lsa, u holda
$a[i,j]=a[i,j-1]= \ldots =a[i-u+1,j]=1$
va $a[i-u,j]$ esa nolga teng bo‘ladi (agarda $i-u>0$ bo‘lsa).
Yuqorida keltirilgan massiv uchun $i=4$ va $j=3$ bo‘lganda $p[3]=2=u$ va $a[4,3]=a[3,3]=1$ va $a[2,3]=0$ bo‘ladi.
Shunday qilib, har bir i qatori uchun biz $S\_i(L,R)$ to‘g‘ri to‘rtburchak yuzasini topishimiz kerak (ya’ni birlar soni). Bu yerda (i,R) va (i,L) asos koordinatalari.

	0	1	1	1	1	0
$i \rightarrow$	0	1	1	1	1	0
		↑			↑	
		L			R	

Bundan $S\_i(L,R)$ to‘g‘ri to‘rtburchakning $(R-L+1)$ asosining balandlikga bo‘lgan ko‘paytmasiga teng bo‘ladi. Balandlikni esa quyidagicha aniqlaymiz

$$h(L,R) = \min_{L \le j \le R} p[i]$$

Shunday qilib har bir i uchun $S\_i(L,R)$ ning $1<=L<=R<=M$ oralig'ida eng katta qiymatini aniqlab, keyin qatorlar ichidan eng kattasi masala yechimi bo'ladi. Birinchi qator uchun $p[]$ massivi juda oddiy aniqlanadi, ya'ni $p[j]=a[1,j]$, chunki yuqorida hech qanday element yo'q va shu bois birlar soni $a[i,j]$ ning o'ziga teng bo'ladi. Har qanday i-qator uchun $p[]$ massivi aniqlangan bo'lsa, keyingi $(i+1)$-qator uchun yangi $p[]$ massivi quyidagicha aniqlanadi:

if a[i+1,j]=0 then p[j]:=0 else p[j]:=p[j]+1;

Ushbu shartli ifodani oddiy arifmetik amallar orqali ham aniqlash mumkin:

p[j]:=(p[j]+1)*a[i+1,j];

Shunday qilib biz i-qator uchun $p[]$ massivini tuzdik. Chegaralari $L[j]<=j<=R[j]$ bo'lgan va balandligi $p[j]$ bo'lgan eng katta to'g'ri to'rtburchakni aniqlaymiz. Misol sifatida quyidagi massivni ko'rib chiqamiz

0	1	0	0	0
0	1	1	0	0
0	1	1	1	0
0	1	1	1	0

Bu yerda 4-qator uchun $p[]$ massivi quyidagi qiymatlarga ega bo'ladi: $p[]=[0,4,3,2,0]$. Masalan, $j=3$ bo'lganda $L[j]$ va $R[j]$ qiymatlarini aniqlashni keltiramiz. $j=3$ o'rindan boshlab chapga qarab yursak $p[3]$ dan kichik bo'lgan qiymatni topamiz, bu yerda $p[1]=0<3=p[3]$ va $L(j)$ topilgan indeksdan bittaga ortiq bo'ladi, ya'ni $L[j]=1+1$. shunday qilib, chap chegara topildi. Agar bunday son topilmasa, u holda $L[j]=1$ bo'ladi. Endi $p[3]$ nuqtadan boshlab o'ng tomonga yuramiz. $p[3]$ dan kichik bo'lgan qiymatni topamiz, bu yerda $p[4]=2<3=p[3]$. $R[j]$ topilgan indeksdan bittaga kam bo'ladi. Ya'ni $R[j]=4-1=3$. shunday qilib, o'ng chegara topildi. Agar o'ng tomonda bunday son topilmasa, u holda $R[j]=M$ bo'ladi. Endi $L[j]$ va $R[j]$ qiymatlarini tezkor hisoblash algoritmini yuqorida keltirilgan g'oyaga binoan ko'rib chiqamiz.

R va L massivlarini to'ldirish uchun p massivini bir boshdan ko'rib chiqamiz

```
for j:=1 to m do
begin
temp:=p[j];L[j]:=1;
```

```
for left:=j downto 1 do  begin
if p[left]<temp then
begin L[j]:=left+1; break; end
end;

R[j]:=m;
for right:=j to m do
begin
if p[right]<temp then
begin R[j]:= right -1; break; end
end;
end;
```

Endi $p[j]*(R[j]-L[j]+1)$ ko'paytmani eng katta qiymatini aniqlaymiz. Har bir qator bo'yicha topilgan maksimumlarning eng kattasi masalaning natijasi bo'ladi. Algoritmning samaradorlik darajasi $O(n*m)$ tartibli bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
4 5 0 1 0 0 0 0 1 1 0 0 0 1 1 1 0 0 1 1 1 0	6 4 3

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchamlari n=4 va m=5 bo'ladi, ikkinchi qatordan boshlab massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida birlar soni va 1 lardan tashkil topgan eng katta to'g'ri to'rtbukchakning past va o'ng koordinatalari chiqarilgan bo'ladi.

Shunday qilib, dasturning umumiy ko'rinishi quyidagicha bo'ladi:

```
program Tortburchak;
const  Max=100;
var
      n,m,s,i,j, temp,Smax, iMax, jMax : integer;
      left, right : integer;
      A:array[1..Max, 1..Max] of integer;
      p, L, R:array[1..Max] of integer;
      fayl : text;
begin
{massiv o'lchamini va element  qiymatlarini kiritamiz}
```

```
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, n, m);
{sikl orqali A[] qiymatlarini kiritamiz}
      for i:=1 to n do
begin
      for j:=1 to m do
begin
      read(fayl, A[i,j])
end;
      readln(fayl);{keyingi qatorga o'tamiz}
end;
      close(fayl);
      Smax:=-1;
      for i:=1 to n do
begin
      for j:=1 to m do
begin
      if i=1 then p[j]:=a[1,j]
      else
begin
      if a[i,j]=0 then p[j]:=0
      else p[j]:=p[j]+1;
end;

end;  {j sikli tugadi}
      for j:=1 to m do
begin
      temp:=p[j];L[j]:=1;
      for left:=j downto 1 do
begin
      if p[left]<temp then
begin
      L[j]:=left+1; break;
end
end;
      R[j]:=m;
      for right:=j to m do
begin
```

```
      if p[right]<temp then
begin
      R[j]:= right -1; break;
end
end;
end; {j sikli tugadi}
      for j:=1 to m do
begin
      temp :=p[j]*(R[j]-L[j]+1);
      if Smax<temp then
begin
      Smax:=temp; iMax:=i; jMax:=j;
end;
end;
end; {i sikli tugadi}
{natijani  chop etamiz }
      assign(fayl, 'output.txt');
      rewrite(fayl);
      writeln (fayl, Smax );
      writeln (fayl,  iMax, ' ' , jMax );
      close(fayl);
end.
```

E'tibor bering, ushbu berilgan massiv uchun 2 ta yechim mavjud,

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \quad \text{va} \quad \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

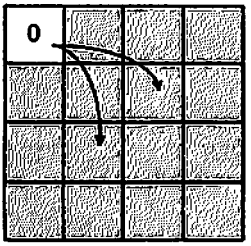
lekin dasturda birinchi uchragan qiymatlar saqlanib qoladi.

Otning yurishi. $N \times M$ $(3 \le N, M \le 10)$ o'lchamli shaxmat taxtasida (X,Y) xonasida ot sipohi joylashgan. Uning (K,L) xonasiga o'tishi uchun kami bilan nechta yurish qilish kerakligini aniqlang, agar mumkin bo'lmasa, bu holda -1 chop etilsin.

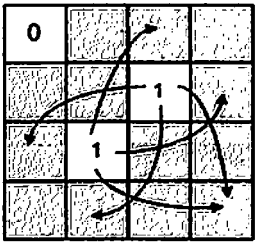
Masala yechimi. Bu yerda masala natijasini *Qadam*$[i,j]$ massivida saqlaymiz. Uning elementi (i,j) xonadan (K,L) xonasigacha bo'lgan qadamlar sonini bildiradi. Ko'p hollarda, ushbu ko'rinishdagi masalalarda, massivning o'lchamlarini ikkitaga ortiq qilib olish kerak

bo‘ladi va u yerda umuman yurish mumkin emasligini belgilab qo‘yish kerak bo‘ladi.

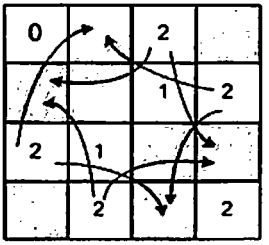
Ushbu masalani oxirgi (K,L) xonadan boshlab, teskari harakat qilib ishlaymiz, ya’ni (K,L) xonasiga qaysi xonadan bir qadam bilan yetib olish mumkin va hokazo. Quyidagi chizmada $N=M=4$ va $K=L=1$ bo‘lganda u yerdan turib qaysi xonalarga yurish mumkinligi har bir chizmada ko‘rsatib o‘tilgan.



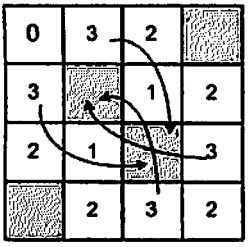
Boshlang‘ich holat



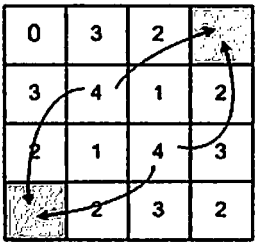
Birinchi qadam



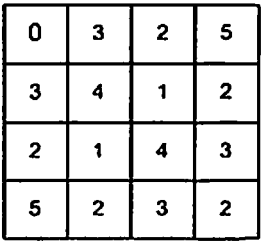
Ikkinchi qadam



Uchinchi qadam



To‘rtinchi qadam



Oxirgi holat

Bu yerda “0”-xonadan qaysi bo‘yalgan (ya’ni bo‘sh) xonaga yurish mumkin bo‘lsa, u yerda “1” ni qo‘yamiz, keyin “1”-xonadan qaysi bo‘yalgan xonalarga yurish qilish mumkin bo‘lsa, u yerda “2” ni qo‘yamiz, shu tariqa barcha xonalar ko‘rib chiqiladi va chizmadagi “Oxirgi holat”ga kelamiz. Ushbu algoritm “to‘lqinli algoritm” nomi bilan mashhur va uni ko‘pgina masalalarda qo‘llash mumkin.

Umumiy ko‘rinishda algoritm mazmunan quyidagicha bo‘ladi:

while <o‘zgartirishlar bo‘lgan> do begin
for i := 1 to N do
for j := 1 to M do
if qadam[i,j] = <joriy qadam> then
<Ot yurishi mumkin bo‘lgan xonalar>:=<joriy qadam> + 1;
<joriy qadamni oshiramiz>;

end;

Ushbu algoritmning samaradorlik darajasi $O((N \cdot M)^2)$ tartibli bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
8 8 2 2 3 1	2

Bu yerda *input.txt* faylida birinchi qatorda taxtaning o'lchamlari n=8 va m=8, ikkinchi qatorda x=1 va y=2, bu ot sipohi joylashgan xona, uchinchi qatorda K=3 va L=1, bu ot sipohi o'tishi kerak bo'lgan xona. Natija *output.txt* faylida bo'lib, unda yurishlar soni chiqarilgan bo'ladi.

Quyida keltirilgan dasturni tushunib olish qiyinchilik tug'dirmaydi.

```
program Ot_Tulqin;
const
     MaxN=12;{Shaxmat taxtasining o'lchami,+2 qo'shilgan}
     plEmpty = -1; {Bo'sh xonani bildiradi}
     plNone  = -2;
{Shaxmat taxtasining yurish mumkin bo'lmagan chakkasi}
{Ot sipohining yurishi mumkin bo'lgan 8 ta yo'nalishi }
dx: array[1..8] of integer=(-1,-1, 1, 1,-2,-2, 2, 2);
dy: array[1..8] of integer=(-2, 2,-2, 2,-1, 1,-1, 1);
var
     qadam: array [-1..MaxN, -1..MaxN] of integer;
{-Shaxmat taxtasi}
     n,m,s,t,x,y,k,l  : integer;
     JoriyQadam   : integer;
     Yurish : Boolean;
{- Yurish qilinganligini bildiradi}
     i, j : integer;
     fayl : text;
{Ot yurishi mumkin bo'lgan xonalarga qiymat berish }
procedure move( x, y : integer);
var i,  xx, yy : integer;
```

```
begin
      JoriyQadam := qadam[x,y] + 1;
      for i := 1 to 8 do
begin
      xx := x+dx[i];
      yy := y+dy[i];
      if qadam[xx,yy] = plEmpty then
begin
      qadam[xx,yy] := JoriyQadam;
      Yurish:=true;
end;
end;
end;
{Qiymatlash qism dasturi}
{ Bu yerda quyidagilarni bajaramiz:}
{- boshlang'ich qiymatlarni o'qib olish;}
{- Shaxmat taxtasini bo'sh deb qiymatlash;}
{- Shaxmat taxtasida chakka xonalarni qiymatlash;}
procedure init;
var i, j : integer;
begin
{boshlang'ich  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, n, m);{ Shaxmat taxtasi }
      readln(fayl, x, y);{ Ot sipohi joylashgan xona }
      readln(fayl, k,l);
{- Ot sipohi o'tishi kerak bo'lgan  xona  }
      close(fayl);
      for i := -1 to m+2 do
      for j := -1 to n+2 do
      if (j<1) or (i<1) or (j>n) or (i>m) then
      qadam[j,i] := plNone
      else
      qadam[j,i] := plEmpty;
      JoriyQadam :=0;
      {-boshlang'ich qadamimiz nolga teng}
      qadam[k,l] := 0;
end;
```

```
{Asosiy dastur}
begin
    init;
    Yurish:=true;
while Yurish do
begin
    Yurish:=false;
    for i := 1 to N do
    for j := 1 to M do
    if qadam[i,j] = JoriyQadam then
begin
    move( i, j);
end;
end;
{endi natijani chop etamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
    if qadam[x,y] <> plEmpty then
        writeln(fayl, qadam[x,y])
    else    writeln(fayl, -1);
    close(fayl);
end.
```

Endi samaradorlik darajasi $O(N \cdot M)$ tartibli bo'lgan algoritmni keltiramiz. Ushbu algoritmning asosiy g'oyasi yurishlarni tartib bilan *qadam*[] massiviga yozish bo'lib, bu yerda quyidagilar amalga oshiriladi:

1) birinchi qadamda biz *qadam*[*k,l*] elementiga 0 qiymatini kiritib qo'yamiz;
2) navbat massivi tashkil qilinib, unga o'tishimiz lozim bo'lgan katak koordinatalari kiritiladi, birinchi bo'lib (*k,l*) kiritiladi;
3) navbat massividan keyingi element olinib, undan ot sipohi yurishi mumkin bo'lgan kataklar uchun *qadam*[] massiviga qiymatlar kiritiladi va ularni navbat massiviga ham kiritib boramiz;
4) oldingi 3-bandni navbat tugaganga qadar davom ettiramiz.

Quyida keltirilgan dastur ushbu g'oyani amalga oshirib beradi:

```
program Ot_Navbat;
```

```
const
      MaxN  = 252;
{-Shaxmat taxtasining o'lchami, +2 qo'shilgan }
      MaxQueue = 10000;
{-Navbat massivining maksimal o'lchami}
      plEmpty  = 253;              {-Bo'sh katakni bildiradi}
      plNone   = 254;
{-Shaxmat taxtasining yurish mumkin bo'lmagan chakkasi}
{Ot sipohining yurishi mumkin bo'lgan 8 ta yo'nalishi }
dx: array[1..8] of integer=(-1,-1, 1, 1,-2,-2, 2, 2);
dy: array[1..8] of integer=(-2, 2,-2, 2,-1, 1,-1, 1);
      type
      TBoard = array [-1..MaxN, -1..MaxN] of byte;
      TPoint = record
       x, y : integer;
       end;
var
      qadam  : ^TBoard;  { Shaxmat taxtasi }
      queue  : array [0..MaxQueue-1] of TPoint;       {-Navbat}
      qr,qw  : integer;
{Navbat massivlaridan o'qish va yozish ko'rsatkichlari}
      n,m,k,l    : integer;
      x, y : integer;
      faylin, faylout : text;
{Ot yurishi mumkin bo'lgan xonalarga qiymat berish:}
{-ot sipohi yurishi mumkin bo'lgan xonalar }
{uchun qadam[] massiviga }
{ qiymatlar bittaga oshirib  kiritiladi;}
{-ko'rilgan xonalarni navbat massiviga kiritiladi;}
{-tezkor xotirani tejash uchun navbat massivini}
{siklik tashkil qiladi;}
procedure move(p : TPoint);
var
      i, tt, x, y : integer;
begin
      tt := qadam^[p.x][p.y] + 1;
      for i := 1 to 8 do
begin
      x := p.x+dx[i];
```

```
      y := p.y+dy[i];
      if qadam^[x][y] = plEmpty then
begin
      qadam^[x][y] := tt;
      queue[qw].x  := x;
      queue[qw].y  := y;
      qw := (qw+1) mod MaxQueue;
end;
end;
end;
{Qiymatlash qism dasturi, bu yerda quyidagilarni } {bajaramiz:}
{- boshlang'ich qiymatlarni o'qib olish;}
{- Shaxmat taxtasini bo'sh deb qiymatlash;}
{- Shaxmat taxtasida chakka xonalarni qiymatlash;}
procedure init;
var i, j : integer;
begin
      new(qadam);
      assign(faylin, 'input.txt');
      assign(faylout, 'output.txt');
      rewrite(faylout);
      reset(faylin);
      readln(faylin, n, m);
      readln(faylin, x, y);
      readln(faylin,  k, l);
      for i := -1 to m+2 do
      for j := -1 to n+2 do
      if (j<1) or (i<1) or (j>n) or (i>m) then
      qadam^[j,i] := plNone
      else
      qadam^[j,i] := plEmpty;
end;
procedure done;
begin
      dispose(qadam);
      close(faylin);
      close(faylout);
      halt(0);
end;
```

```
{Shaxmat taxtasini to'ldirish qism dasturi:}
{- navbatga oxirgi nuqtani (k,l) kiritamiz;}
{- navbat tugamanguncha keyingi element uchun}
{yurish qilish;}
procedure prepare;
begin
      qr := 0;
      qw := 1;
      with queue[0] do
begin
      x := k;
      y := l;
end;
      qadam^[k,l] := 0;
while qw <> qr do
begin
      move(queue[qr]);
      qr := (qr+1) mod MaxQueue;
end;
end;
{Natijani chop etish }
{Agar ot bo'sh katakda joylashgan bo'lsa, demak }
{(k,l) xonasiga o'tib bo'lmaydi, }
{ bunda -1 ni chop etamiz}
procedure solve;
var ss, i : longint;
begin
      ss := 0;
      if qadam^[x,y] <> plEmpty then
      ss:= qadam^[x,y]
      else
begin
      writeln(faylout, -1);exit;
end;
      writeln(faylout, ss);
end;
begin
      init;
      prepare;
```

```
    solve;
    done;
end.
```

Topshiriqlar

1.Tartiblangan ikki massiv

$$x[1] \le \ldots \le x[k]$$

va

$$y[1] \le \ldots \le y[L]$$

elementlari ichidan shunday 2 tasini topish talab etiladiki, ularning yig'indisi berilgan M soniga eng yaqin bo'lsin.

2.Butun sonlardan tashkil topgan va tartiblangan 2 ta massiv

$$x[1] \le \ldots \le x[k]$$

va

$$y[1] \le \ldots \le y[L]$$

elementlari ichida nechta har xil sonlar borligini aniqlang.

3. Berilgan $a[n]$ massivi va $m \le n$ uchun, ketma-ket kelgan barcha m ta massiv elementlari yig'indisini hisoblang .

4. Berilgan $a[n,n]$ massivi va $m \le n$ uchun, barcha m x m kvadrat elementlari yig'indisini hisoblang.

5. Berilgan $a[n]$ massivi elementlari o'zaro bir-biriga teng emas va 0 dan n gacha bo'lgan sonlardan iborat. Tashlab ketilgan sonni aniqlang.

6. Butun sonlardan tashkil topgan ikkita massiv $a[n]$ va $b[m]$ lardan eng uzun umumiy qism massivni aniqlang.

7.Berilgan sonli $a[n]$ massivi nechta har xil sonlardan tashkil topganligini aniqlang .

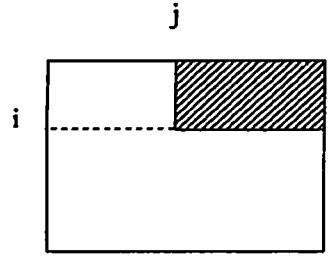
8. Berilgan $X[2n, 2n]$ massivni quyidagicha tasvirlasak

$$X= \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

Undan quyidagi massivni tashkil qiling.

X= D C B A

9.Berilgan $a[n,n]$ sonli massividan $b[n,n]$ massivining b_{ij} elementi quyidagi chizmada shtrix bilan belgilangan a_{kl} elementlarining yig'indisi orqali aniqlanadi.

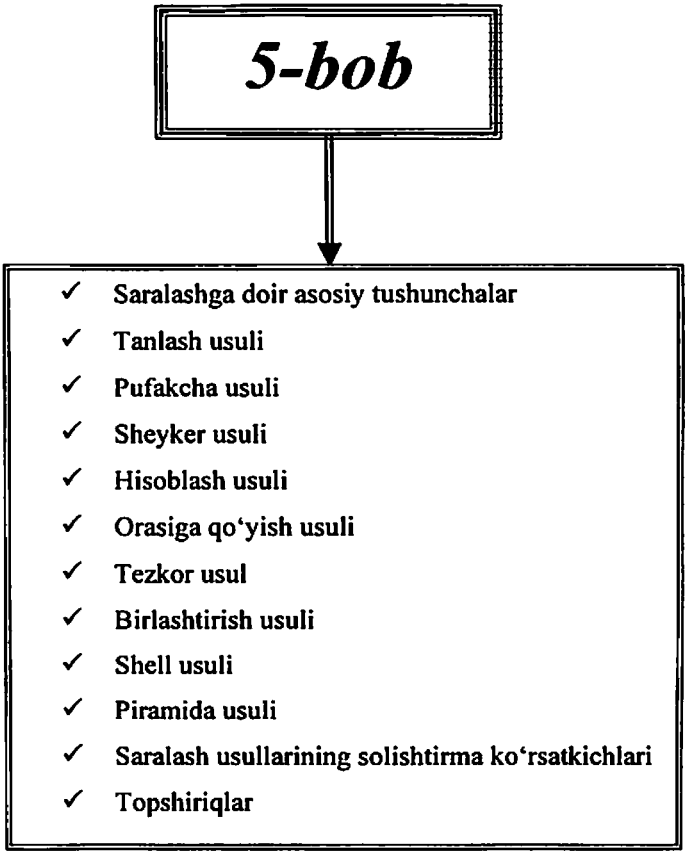


10. Berilgan $a[n,n]$ massivida 1-qatorni n-qator bilan, 2- qatorni $(n-1)$-qator bilan va hokazo barcha qatorlarni o'rnini almashtiring .
11.Berilgan ikki massiv $a[m,n]$ va $b[n,k]$ ko'paytmasini aniqlang.
12. Berilgan $a[n,n]$ sonli massividan $b[n,n]$ massivining b_{ij} elementi i-qatordagi va j-ustundagi sonlar yig'indisiga teng bo'lsin.

5-bob. MASSIVLARNI SARALASH ALGORITMLARI VA DASTURLARI

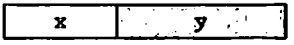
Ko‘pgina masalalarda katta hajmdagi ma’lumotlarni tartibga solish talab etiladi. Agar ma’lumotlar kompyuter xotirasida qandaydir qoida bo‘yicha tartibda saqlanadigan bo‘lsa, u holda ularni izlash, qayta ishlash kabi masalalarni oddiyroq, tez va samaraliroq bajarish mumkin bo‘ladi.

Mazkur bobda massiv elementlarini tartiblash usullari va ularning solishtirma ko‘rsatkichlari haqida umumiy tushunchalar berilgan.



5.1. Saralashga doir asosiy tushunchalar

Avtomatlashtirilgan axborot tizimlarida qayta ishlanishi lozim bo'lgan ma'lumotlar faqatgina bitta maydondan iborat bo'lmaydi. Masalan talabalar to'g'risidagi ma'lumotlarimiz ularning familiyasi, ismi va sharifidan iborat bo'lishi mumkin. Tartibga solish maydoni kalit deb aytiladi. Kalitlar o'z navbatida bir nechta darajali bo'lishi mumkin, asosan birlamchi va ikkilamchi kalitlar yuritiladi. Masalan talabalarni ro'yxat bo'yicha tartiblash familiya maydoni (rasmdagi "*x*" maydoni) bo'yicha amalga oshirilsa, u birlamchi kalit deb hisoblanadi va undan so'ng ismlar bo'yicha (rasmdagi "*y*" maydoni) ham tartibga solsak, u holda ism maydoni ikkilamchi kalit hisoblanadi.



Shunday qilib quyidagi ta'rifni kiritish mumkin: Tartibga solish natijasida yozuvlar kalitlarning qiymati ortib borishi yoki kamayib borish bo'yicha joylashtirish jarayoni saralash deb ataladi.

Kalit maydoni asosan sonli va yozuvli bo'ladi. Sonli maydon bo'yicha saralashni biz ko'p uchratamiz, masalan, guruh jurnallarida talabalar tartib bilan raqamlab chiqilgan. Agar e'tibor bergan bo'lsangiz, ushbu guruh jurnalida talabalar familiyasi alifbo bo'yicha ham tartibga solingan, ya'ni yozuvli maydonni saralash kompyuterda har bir belgi bo'yicha amalga oshiriladi. Ya'ni ikki so'z bo'yicha aytiladigan bo'lsak, dastlab birinchi belgilar taqqoslanadi, keyin ikkinchi belgilar taqqoslanadi va hokazo. Belgilarning katta-kichikligi ularni **ASCII** standarti bo'yicha joylashgan o'rniga bog'liq. Unda harflar bevosita alifbo tartibi bo'yicha keltirilgan bo'ladi va bu leksiografik tartibi deb aytiladi. Lekin o'zbek lotin alifbosi tartibi **ASCII** dagi tartibga mos kelmaydi!

Saralash jarayoni ma'lumotlarning joylashganligi bo'yicha ichki va tashqi saralashga farqlanadi. Agar tartibga solinadigan ma'lumotlar to'laligicha operativ xotirada joylashgan bo'lsa va saralash jarayoni o'sha yerda amalga oshirilsa, bu ichki saralash deb hisoblanadi. Tashqi xotira qurilmalarida saqlanayotgan ma'lumotlarni saralash jarayoni esa tashqi saralash deyiladi.

Barcha saralash jarayonlari bir necha takroran bajariladigan amallardan iborat bo'ladi. Takroriy jarayonning bir iteratsiyasini o'tish dey ataymiz.

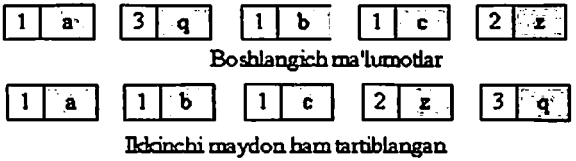
Saralash masalalarida quyidagi ko'rsatkichlar muhim ahamiyatga ega, chunki ular orqali biz algoritmlarni baholashimiz mumkin.

1. Xotira – algoritm uchun zarur bo'lgan xotira hajmi.

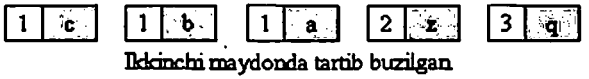
2.Vaqt – asosiy ko'rsatkich, algoritmlarni qay darajada tezkorligini anglatadi.

4. Tabiiylik – boshlang'ich ma'lumotlar saralangan yoki qisman saralangan bo'lganda ham saralash algoritmi samarali bo'lishlik.

3. Turg'unlik – saralash algoritmning turg'unligi undagi teng elementlarning tartibini buzilmasligini anglatadi. Quyidagi ikki maydonli ketma-ketlikda 1-maydon birlamchi deb qabul qilingan. Birlamchi kalit bo'yicha bajarilgan saralashdan so'ng ikkinchi maydonda tartib buzilmagan, demak ushbu saralash turg'un hisoblanadi.



Keyingi saralashda esa ikkinchi maydonda tartib buzilgan, demak bu yerda qo'llanilgan saralash algoritmi turg'un emas.



Saralangan ma'lumotlarning quyidagi afzalliklari mavjud:

- saralangan ma'lumotlardan foydalanish yengil amalga oshiriladi;
- saralangan ro'yxatdan kerakli ma'lumotni qidirish yengilroq;
- saralangan ro'yxatda oraliqda mavjud bo'shlikni aniqlash yengilroq;
- ikkita saralangan jadvallarni taqqoslash oson.

Saralash algoritmlarini chuqur o'rganish orqali biz unda mavjud g'oyalar bilan tanishib chiqamiz. Bevosita bitta masala doirasida algoritmlarni taqqoslashni, ularning sifatini aniqlashni o'rganib, bu

orqali murakkab olimpiada masalalarini yechish uchun poydevor yaratgan b o'lamiz.

Biz ushbu bobda ichki saralash algoritmlari haqida so'z yuritamiz va ularning afzalliklari va kamchiliklari haqida to'xtalib o'tamiz. Saralash usulini baholashda solishtirishlar va o'rin almashtirishlarni eng ko'p va kam qiymatini aniqlash talab etiladi. Biz bu yerda eng ko'p qiymatni baholovchi ko'rsatkichni keltiramiz.

5.2. Tanlash usuli

Shunday qilib, agar bizga $a_1, a_2, \ldots, a_n$ ketma-ketligi berilgan bo'lsa, undagi elementlarni o'rin almashtirish orqali o'sish yoki kamayish tartibi bilan joylashtirish talab etiladi, ya'ni

$$a[1] \le a[2] \le \ldots \le a[n] \text{ yoki } a[1] \ge a[2] \ge \ldots \ge a[n].$$

Ushbu masalani yechimini oddiy algoritmlardan boshlaymiz, ya'ni tanlash usuli asosida yaratilgan algoritmni ko'rib chiqamiz.

Tanlash usulining asosiy g'oyasi quyidagicha, ketma-ketlik orasidan eng kichik element izlab topiladi. Ushbu element birinchi o'rin bilan almashtiriladi. Shundan so'ng qolgan (n-1) uzunlukdagi ketma-ketlik xuddi shu usul bilan tekshiriladi.

Quyidagi jadvalda aniq misolda (28, 17, 73, 47, 22, 10, 65, 54) tanlash usuli ko'rsatilgan.

i o'tish	Massiv $a[i]$							
0	**28**	**17**	**73**	**47**	**22**	**10**	**65**	**54**
1	10	**17**	**73**	**47**	**22**	**28**	**65**	**54**
2	10	17	**73**	**47**	**22**	**28**	**65**	**54**
3	10	17	22	**47**	**73**	**28**	**65**	**54**
4	10	17	22	28	**73**	**47**	**65**	**54**
5	10	17	22	28	47	**73**	**65**	**54**
6	10	17	22	28	47	54	**65**	**73**
7	10	17	22	28	47	54	65	73

Dasturda *IndMin* funksiyasini kiritamiz, unda *a*[] massivning "*start*" indeksidan boshlab *n* gacha bo'lgan sonlar ichidan eng kichigining indeksi aniqlab beriladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*Output.txt*
8 28 17 73 47 22 10 65 54	10 17 22 28 47 54 65 73

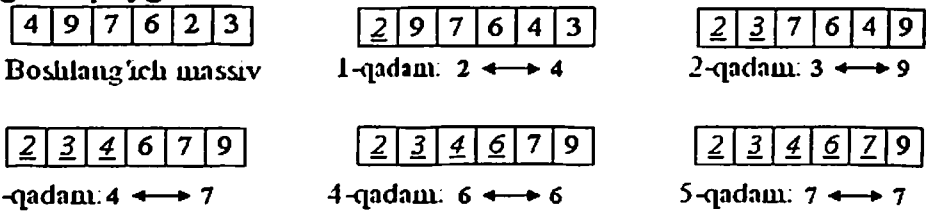
Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi massiv elementlari qiymatlari chiqarilgan bo'ladi.

Bevosita ushbu algoritm bo'yicha tuzilgan dastur quyidagicha bo'ladi.

```
{ Massivni tanlash usuli bilan saralash dasturi }
program Choice_Sort;
const  Nmax = 100;
type
      arrtype = array[1..Nmax] of integer;
var  i, n : integer;
      a : arrtype;
      fayl : text;
{qo'shimcha funksiyalar}
function IndMin(a:arrtype; start : integer ) : integer;
var i, imin : integer;
begin
      imin := start;
      for i := start + 1 to n do
      if a[ i ] < a[ imin ] then imin := i;
      IndMin := imin
end;
{ o'rin almashtirish}
procedure Swap( var a, b : integer );
var c : integer;
begin
      c := a; a := b; b := c
end;
{dasturning asosiy qismi}
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
```

```
    assign(fayl, 'input.txt');
    reset(fayl);
    read(fayl, n);
    for i:=1 to n do read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
    close(fayl);
    for i := 1 to n-1 do
    Swap( a[ IndMin( a, i ) ] , a[ i ] );
{chiqarish fayliga natijani chop etamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
    for i :=1 to n do      write (fayl, a[i], ' ');
    close(fayl);
end.
```

Quyidagi misolda bevosita o‘rin almashtiriladigan elementlarni belgilab qo‘yganmiz:



Demak n ta elementdan iborat massivni saralash uchun $(n\text{-}1)$ ta utish talab etiladi, bir utishda $(n\text{-}i)$ ta taqqoslash talab etiladi. Shu bois algoritmda bajariladigan amallarning umumiy soni quyidagiga teng bo‘ladi:

$$\sum_{i=1}^{n-1}(n-i) = 0{,}5 \cdot n \cdot (n-1)$$

Shundan kelib chiqqan holda, algoritmning samaradorlik darajasi $O(n^2)$ tartibli bo‘ladi va ushbu algoritm qo‘shimcha xotira talab etmaydi.

5.3. Pufakcha usuli

Saralash jarayonida qo‘shni elementlar juftlab solishtiriladi va o‘rinlari bilan almashtiriladi. Agar o‘sish tartibi bilan saralash talab qilinsa, ushbu algoritm qo‘llanilganda, undan eng katta element birin-ketin o‘ngga siljiy boshlaydi. Shu yo‘l bilan ushbu element oxirgi

o'ringa qo'yilgandan so'ng, massivning o'lchami bittaga kamaytiriladi va jarayon davom ettiriladi.

Agar elementlarni vertikal joylashtirsak va eng katta elementni pastdan boshlab joylashtirsak qiziqarli qonuniyatni ko'rish mumkin, ya'ni kichik elementlar har bir o'tishda pufakchaga o'xshab yuqoriga qarab intiladi. Shu bois ushbu algoritm pufakcha usuli deb ham ataladi. Ushbu usulning afzalligi , uni istalgan ko'rinishdagi misollar uchun qo'llash mumkinligida.

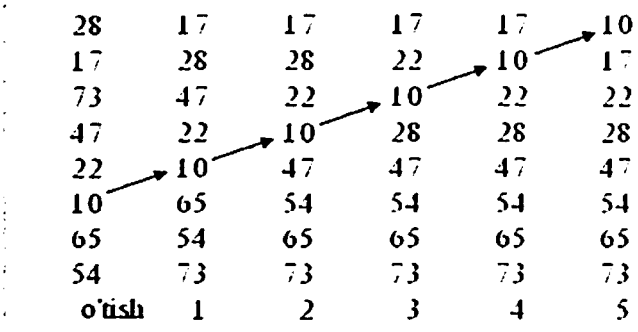
Misol sifatida quyidagi massivni ko'rib chiqamiz:

28, 17, 73, 47, 22, 10, 65, 54

Har bir o'tishda sodir bo'ladigan jarayonlarni quyidagi jadvallarda keltiramiz

Solishtiriladigan elementlar	Almashtirish
1-o'tish:	
28 va 17	bajarildi
28 va 73	bajarilmaydi
73 va 47	bajarildi
73 va 22	bajarildi
73 va 10	bajarildi
73 va 65	bajarildi
73 va 54	bajarildi
Natijaviy massiv: **17, 28, 47, 22, 10, 65, 54, 73**	
2-o'tish:	
17 va 30	bajarilmaydi
30 va 47	bajarilmaydi
47 va 22	bajarildi
47 va 11	bajarildi
47 va 65	bajarilmaydi
65 va 54	bajarildi
Natijaviy massiv: **17, 28, 22, 10, 47, 54, 65, 73**	

Ushbu natijalarni quyidagi jadvalda keltiramiz, e'tibor bering, eng kichik element, ya'ni 10 har bir o'tishda yuqoriga qarab siljiydi.



1-rasm. Saralashning pufakcha usuli.

Quyida keltirilgan dasturdagi o'zgaruvchilarni ma'nosi quyidagicha:

r – o'ng chegara, ya'ni har bir o'tishda 1 dan *r* gacha bo'lgan elementlar ko'rib chiqiladi;

i – ikkita solishtiriladigan elementlarni chap indeksi, ya'ni *a*[*i*] va *a*[*i* + *1*] juftlikning chap elementining indeksi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*Output.txt*
8 28 17 73 47 22 10 65 54	10 17 22 28 47 54 65 73

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi massiv elementlari qiymatlari chiqarilgan bo'ladi.

Bevosita ushbu algoritm bo'yicha tuzilgan dastur quyidagicha bo'ladi.

```
{ Massivni pufakcha usuli bilan saralash dasturi }
program Pufakcha_Sort;
const  Nmax = 100;
type
      arrtype = array[1..Nmax] of integer;
var n, i : integer;
      a : arrtype;
      fayl : text;
{qo'shimcha funksiyalar}
```

```
      { o'rin almashtirish}
procedure Swap( var a, b : integer );
var c : integer;
begin
      c := a; a := b; b := c
end;
      { saralash usuli}
procedure Bubble_Sort( var a : arrtype; n : integer );
var r, i : integer;
begin
      for r := n downto 2 do
      for i := 1 to r - 1 do
      if a[ i ]>a[i + 1] then Swap( a[ i ], a[ i + 1 ] )
end;
      {dasturning asosiy qismi}
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      for i:=1 to n do
      read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
      close(fayl);
      Bubble_Sort( a,n );
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i :=1 to n do      write (fayl, a[i], ' ');
      close(fayl);
end.
```

Pufakcha usuli uchun barcha solishtirishlar soni saralash uchun zarur bo'ladigan o'tishlar soniga bog'liq. Eng yomon variant uchun, ya'ni massiv teskari tartibda bo'lsa, u holda har bir i - o'tishda $(n-i)$ ta almashtirishlarni bajaradi. Barcha o'tishlar soni esa $(n-1)$ ta bo'ladi. Demak, bajariladigan operatsiyalar soni C quyidagiga teng bo'ladi, ya'ni

$$C = \sum_{i=1}^{n-1} (n - i) = 0{,}5 \cdot n \cdot (n - 1)$$

Shunday qilib, ushbu algoritmning ham samaradorlik darajasi $O(n^2)$ tartibli bo'ladi.

Keltirilgan algoritmlarni, oz bo'lsada optimallashtirish mumkin. Shulardan ikkita variantini keltiramiz. Birinchisi bu almashtirishlar sodir bo'lmasa jarayonni to'xtatish. Agar *i*-o'tishda hech qanday ikki element o'rinlari bilan almashtirilmasa, demak elementlarimiz saralash jarayonida tartiblashgan bo'ladi. Ushbu faktni mantiqiy *YesNo* o'zgaruvchisida qayd qilamiz.

Yuqorida keltirilgan *Bubble_Sort* qism dasturigina boshqacha ko'rinishda bo'ladi, ya'ni

```
procedure Bubble_Sort( var a : arrtype; n : integer );
var  r, i : integer;
      YesNo  : boolean;
{almashtirilsa YesNo  = true, aksincha YesNo  = false }
begin
       YesNo  := true;
     r := n;
     while ( r >= 2 ) and YesNo  do
begin
     YesNo  := false;
     for i := 1 to r - 1 do
     if a[ i ] > a[ i + 1 ] then
begin
   Swap( a[ i ], a[ i + 1 ] );
       YesNo  := true;
end;
     r := r - 1
end
end;
```

Ikkinchidan, faqatgina almashtirishlarni emas, balkim uning eng oxirgi almashtirilgan elementlarni o'rnini ham e'tiborga olsak, u holda keraksiz tekshirishlar soni yanada kamayadi.

Bu yerda ham faqatgina yuqorida keltirilgan *Bubble_Sort* qism dasturini o'zgartiramiz, ya'ni

```
procedure Bubble_Sort( var a : arrtype; n : integer );
var r, i : integer;
     YesNo  : boolean;
{almashtirilsa YesNo  = true, aksincha YesNo  = false }
begin
        YesNo  := true;
    r := n;
    while ( r >= 2 ) and YesNo  do
begin
        YesNo  := false;
    for i := 1 to r - 1 do
    if a[ i ] > a[ i + 1 ] then
begin
    Swap( a[ i ], a[ i + 1 ] );
    YesNo  := true;
{i+1  - element  o'z o'rnini egalladi}
          r :=i;
{  - bu massivning yangi chegarasi}
end;
end
end;
```

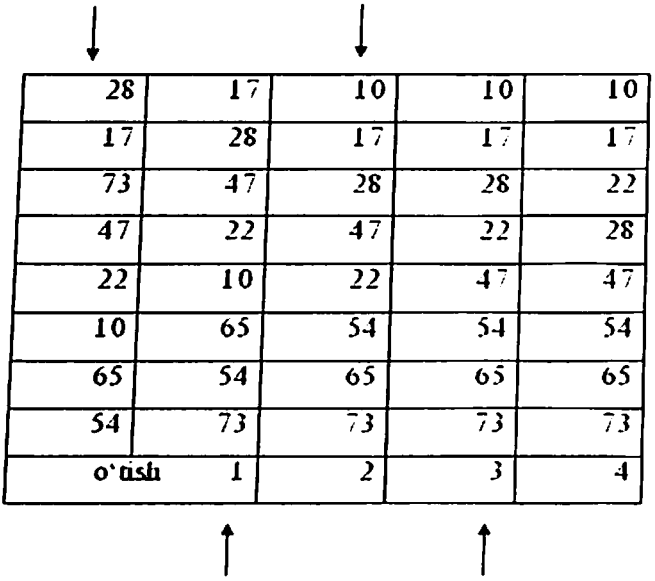
5.4. Sheyker usuli

Yuqorida keltirilgan pufakcha usulida biz faqatgina chapdan o'ngga qarab almashtirishlarni amalga oshirdik. Bu esa quyidagi misolda 5, 7, 12, 28, 36, 85, 2 uncha samarali bo'lmaydi, chunki ushbu massiv deyarli tartiblangan va oxirgi eng kichik element bir o'tishda o'z joyiga kelib tushmaydi. Shu bois jarayonlarni ikki tomonlama amalga oshirish mumkin, ya'ni chapdan o'ngga va o'ngdan chapga. Ushbu usul "sheyker" deb ataladi (inglizchadan **shake** – silkitish degan ma'noni anglatadi).

Misol sifatida yuqorida keltirilgan massivni ko'rib chiqamiz:

28, 17, 73, 47, 22, 10, 65, 54

Har bir o'tishda sodir bo'ladigan jarayonlarni va ularning yunalishini quyidagi jadvallarda keltiramiz



28	17	10	10	10
17	28	17	17	17
73	47	28	28	22
47	22	47	22	28
22	10	22	47	47
10	65	54	54	54
65	54	65	65	65
54	73	73	73	73
o'tish	1	2	3	4

Qism dasturda chap va o'ng tomon chegaralari uchun quyidagi o'zgaruvchilarni kiritamiz: *left* – chap chegara va *right* – o'ng chegara. Saralash jarayonini *left* < *right* bo'lganga qadar davom ettiramiz.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*Output.txt*
8 28 17 73 47 22 10 65 54	10 17 22 28 47 54 65 73

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida saralangan massiv elementlari qiymatlari chiqarilgan bo'ladi.

```
{ Massivni Sheyker usuli bilan saralash dasturi }
program ShakeSort;
const    Nmax = 100;
type
     arrtype = array[1..Nmax] of integer;
var
     n, i : integer;
     a : arrtype;
```

```
      fayl : text;
{qo'shimcha funksiyalar}
procedure Swap( var a, b : integer );
var c : integer;
begin
      c := a; a := b; b := c
end;
procedure Shaker_Sort( var a : arrtype; n : integer );
var
      left, right , i : integer;
      YesNo  : boolean;
begin
      YesNo  := true;
      left := 1; right  := n;
      while (left < right  ) and YesNo  do
begin
      YesNo  := false;
      for i := 1 to right  - 1 do
{ chapdan o'ngga qarab o'tish }
      if a[ i ] > a[ i + 1 ] then
begin
      Swap( a[ i ], a[ i + 1 ] );
      YesNo  := true;
{i+1  - element o'z o'rnini egalladi}
      right  := i
{  - bu massivning yangi o'ng  chegarasi}
end;
      for i := right  downto left + 1 do
{ o'ngdan chapga qarab o'tish }
      if a[ i ] < a[ i - 1 ] then
begin
      Swap( a[ i ], a[ i - 1 ] );
      YesNo  := true;
{i-1  - element o'z o'rnini egalladi}
      left := i
{  - bu massivning yangi chap chegarasi}
end;
end
end;
  {dasturning asosiy qismi}
```

```
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      for i:=1 to n do
      read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
      close(fayl);
      Shaker_Sort ( a,n );
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i :=1 to n do
      write (fayl, a[i], ' ');
      close(fayl);
end.
```

5.5. Hisoblash usuli

Juda oddiy g'oyaga asoslangan usul hisoblanadi. Berilgan massivning birinchi elementi boshqa elementlarning k tasidan katta bo'lsa, demak saralangan massivda u $(k+1)$ – o'rinda joylashtiriladi. Ushbu usulni to'liq tushunib olish uchun quyidagi misolni jadval ko'rinishida keltiramiz

I	1	2	3	4	5	6
$A[i]$	11	5	12	8	7	3
o'tish qadami	1	2	3	4	5	6
K	4	1	5	3	2	0
$B[k+1]$	3	5	7	8	11	12

Massivning birinchi elementi $A[1]=11$ massivning qolgan $k=4$ ta elementidan katta, demak uni $(k+1)$ - o'ringa joylashtiramiz. E'tibor bering, n ta elementli massivni saralash uchun n ta qadam kerak bo'ladi. Har bir qadamda n ta solishtirish talab etiladi, shu bois ushbu algoritmning samaradorlik darajasi $O(n^2)$ tartibi bo'ladi. Ushbu algoritmning dasturi quyidagicha bo'ladi.

```
procedure Count_Sort(a:arrtype; var b : arrtype; n : integer);
var i, j, k: integer;
begin
      for i:=1 to n do {har bir massiv elementi uchun}
 begin          { k ni  aniqlaymiz}
      k:=0;
      for j:=1 to n  do {barchasi bilan solishtiramiz}
      if a[j]<a[i] then k:=k+1;
{ demak, a[i] element   (k+1)-o'rinda joylashtiriladi }
  b[k+1]:=a[i];
end
end;
```

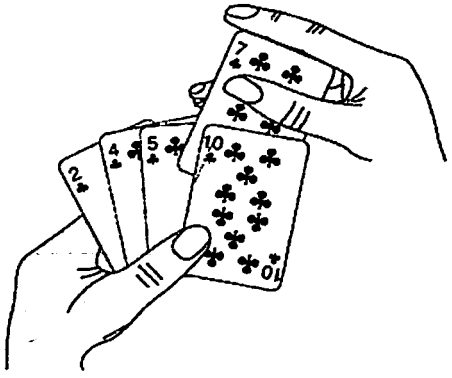
Endi bevosita yuqorida keltirilgan dasturdagi *ShakeSort* qism dasturi o'rniga *Count_Sort* qism dasturini joylashtiramiz va natijani tekshiramiz.

```
{ Massivni hisoblash usuli bilan saralash dasturi }
program CountSort;
const     Nmax = 100;
type
      arrtype = array[1..Nmax] of integer;
var n, i : integer;
      a, b : arrtype;
      fayl : text;
{qo'shimcha funksiyalar}
procedure Count_Sort(a:arrtype; var b : arrtype; n : integer);
var i, j, k: integer;
begin
      for i:=1 to n do {har bir massiv elementi uchun}
 begin          { k ni  aniqlaymiz}
      k:=0;
      for j:=1 to n  do {barchasi bilan solishtiramiz}
      if a[j]<a[i] then k:=k+1;
{demak,  a[i] element   (k+1)-o'rinda joylashtiriladi }
      b[k+1]:=a[i];
end
end;
  {dasturning asosiy qismi}
begin
```

```
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      for i:=1 to n do
      read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
      close(fayl);
      Count_Sort(a,  b , n );
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i :=1 to n do      write (fayl, b[i], ' ');
      close(fayl);
end.
```

5.6. Orasiga qo'yish usuli

Ushbu usuldan hayotda biz juda ko'p foydalanamiz, shulardan biri quyidagi chizmada keltilgan:



Massiv bilan esa quyidagi jarayonni amalga oshiramiz, ya'ni berilgan $a[n]$ massiv elementlari birin-ketin tanlanib, ikkinchi $b[n]$ massivga tartib bo'yicha qo'yiladi. Dastlab $a[1]$ elementi $b[1]$ o'ringa joylashtiriladi. agar $a[2]<b[1]$ bo'lsa, $b[1]$ ikkinchi o'ringa siljitilib, birinchi o'ringa $a[2]$ elementi qo'yiladi. Bevosita ushbu jarayonni quyidagi misolda ko'rib chiqamiz: 11, 5, 12, 8, 7, 3.

a[] massiv	11	5	12	8	7	3
1-o'tish	11					
2-o'tish	5	11				
3-o'tish	5	11	12			
4-o'tish	5	8	11	12		
5-o'tish	5	7	8	11	12	
6-o'tish	3	5	7	8	11	12

Har bir qadamda qo'yiladigan elementlar to'rtburchak bilan belgilangan. Ushbu jarayonda biz *b*[] massivini tartibini buzmagan holda unga elementlarni kiritayapmiz.

Ushbu jadvaldan algoritmning samaradorlik darajasi $O(n^2)$ tartibli bo'lishi ko'rinib turibdi.

Berilgan *a*[] massividagi elementni *b*[] massiviga ikki usul bilan joylashtirish mumkin:

1) tartib bilan saralangan *b* massivi yonidan fikran tanlangan *a*[*i*] elementini qo'yib, chap tomondagi elementlar bilan taqqoslanib *b*[*j*] ≤ *a*[*i*] bo'lgunga qadar o'rinlari almashtiriladi;
2) tanlangan *a*[*i*] elementining *b*[] massivida o'rni aniqlanadi, o'ng tomondagi qolgan elementlar bittaga o'ngga siljitiladi, ya'ni orani ochamiz, endi bo'sh o'ringa *a*[*i*] ni joylashtiramiz.

Umumiy holda *b*[] massivini kiritish shart emas. Faqatgina algoritmni yengil tushunib olish uchun u haqida so'z yuritdik.

Birinchi usulning qism dasturi quyidagi ko'rinishda bo'ladi.

```
procedure Insert_Sort(var a:arrtype; n : integer);
var i, j :integer;
begin
{a massivining ikkinchi elementidan boshlaymiz, chunki}
{birinchisi allaqachon o'z-o'zi bilan tartiblangan }
      for i:=2 to n do
      if a[i]<a[i-1] then
{a[i]-element, tartiblangan elementlarning chegaraviy }
{ a[i-1]- elementi bilan taqqoslanadi }
begin
      j:=i; {tanlangan element indeksi }
repeat
```

```
      Swap(a[j],a[j-1]);
{ikki element o'rinlarini almashtirdik}
      j:=j-1
{ ko'chiriladigan  elementning yangi indeksi }
until (j=1) or (a[j]>=a[j-1])
end
end;
```

Endi bevosita yuqorida keltirilgan dasturdagi *ShakeSort* qism dasturi o'rniga *Insert_Sort* qism dasturini joylashtiramiz va natijani tekshiramiz.

```
{Massivni orasiga qo'yish usuli bilan saralash dasturi}
program InsertSort_1;
const     Nmax = 100;
type
      arrtype = array[1..Nmax] of integer;
var
      n, i : integer;
      a : arrtype;
      fayl : text;
{qo'shimcha funksiyalar}
procedure Swap( var a, b : integer );
var c : integer;
begin
      c := a; a := b; b := c
end;
procedure Insert_Sort(var a:arrtype; n : integer);
var i, j :integer;
begin
{a massivining ikkinchi elementidan boshlaymiz, chunki}
{ birinchi element allaqachon o'z-o'zi bilan tartiblan}
      for i:=2 to n do
      if a[i]<a[i-1] then
{ a[i]-element,tartiblangan elementlarning chegaraviy }
{  a[i-1]- elementi bilan taqqoslanadi }
begin
      j:=i;  {tanlangan  element indeksi }
repeat
      Swap(a[j],a[j-1]);
{ikki element o'rinlarini almashtirdik}
```

```
      j:=j-1
{ ko'chiriladigan  elementning yangi indeksi }
until (j=1) or (a[j]>=a[j-1])
end
end;
  {dasturning asosiy qismi}
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      for i:=1 to n do
      read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
      close(fayl);
      Insert_Sort ( a, n );
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i :=1 to n do
      write (fayl, a[i], ' ');
      close(fayl);
end.
```

Albatta har bir qadamda j=1 shartini tekshirish dasturni bajarilish vaqtini oshiradi. Ushbu kamchilikni bartaraf qilishni muxtaram talabalarga mustaqil ish sifatida topshiramiz.

Ikkinchi usul bilan masalani yechishda biz 2 ta qo'shimcha qism dasturlarni kiritamiz:

1) *Find_Place* — tanlangan $a[i]$ elementning massivning saralangan qismidagi o'rnini aniqlaydi;

2) *Insert* — tanlangan $a[i]$ elementini joylashtirish maqsadida, j – elementdan boshlab o'ngga bir qadam siljitaladi va $a[i]$ elementini qo'yadi.

Asosiy va qo'shimcha qism dasturlarimiz quyidagi ko'rinishda bo'ladi.

```
procedure Find_Place (a:arrtype; i:integer; var j:integer);
begin
 { j indeksini aniqlashga kirishamiz}
```

```
  j:=1;       { massivni  boshidan boshlaymiz }
  while a[j] < a[i] do j:=j+1;
end;

procedure Insert(var a:arrtype; i,j:integer) ;
var tmp,k:integer;
begin
  tmp:=a[i];
{ a[i] elementini vaqtincha xotirada saqlaymiz }
  for k:=i downto j+1 do
{ i dan j+1 gacha elementlar indeksi }
  a[k]:=a[k-1];
{o'ngga bitta katakga siljitamiz}
  a[j]:=tmp;
{ a[i] elementni j-katakda joylashtiramiz}
end;
procedure Insert_Sort(var a:arrtype);
var i,j:integer;
begin
 for i:=2 to N do
   if a[i]<a[i-l] then
begin
       { a[i] elementini o'rnini aniqlaymiz}
       Find_Place(a,i,j);
       { a[i] elementini topilgan o'ringa qo'yamiz}
       Insert(a,i,j)
end
end;
```

Endi bevosita yuqorida keltirilgan dasturdagi *ShakeSort* qism dasturi o'rniga *Insert_Sort* qism dasturini joylashtiramiz (*Find_Place* va *Insert* qism dasturlari bilan birgalikda) va natijani tekshiramiz.

```
{Massivni orasiga qo'yish usuli bilan saralash dasturi }
program InsertSort_2;
const    Nmax = 100;
type
      arrtype = array[1..Nmax] of integer;
var
      n, i : integer;
```

```
      a : arrtype;
      fayl : text;
{qo'shimcha funksiyalar}
procedure Find_Place(a:arrtype; i:integer; var j:integer) ;
begin
 { j indeksini aniqlashga kirishamiz}
  j:=1;       { massivni  boshidan boshlaymiz }
  while a[j] < a[i] do j:=j+1;
end;
procedure Insert (var a:arrtype; i, j:integer) ;
var tmp,k:integer;
begin
  tmp:=a[i];
{ a[i] elementini vaqtincha xotirada saqlaymiz }
  for k:=i downto j+1 do
{ i dan  j+1  gacha elementlar indeksi }
  a[k]:=a[k-1];
{o'ngga bitta katakga siljitamiz}
  a[j]:=tmp;
{ a[i]  elementni  j-katakda joylashtiramiz}
end;
procedure Insert_Sort(var a:arrtype);
var i,j:integer;
begin
 for i:=2 to N do
   if a[i]<a[i-1] then
begin
      { a[i] elementini o'rnini aniqlaymiz}
      Find_Place(a, i , j);
      { a[i] elementini topilgan o'ringa qo'yamiz}
      Insert(a, i, j)
end
end;
{dasturning asosiy qismi}
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
```

```
      for i:=1 to n do
      read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
      close(fayl);
      Insert_Sort ( a );
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i :=1 to n do
      write (fayl, a[i], ' ');
      close(fayl);
end.
```

Ushbu ikki usulning o'zaro taqqoslash maqsadida quyidagi jadvalni keltiramiz.

Usul	Massiv elementlari soni		
	1000	2000	4000
Taqqoslash va o'rin almashtirish	1,3	5,3	21,5
O'rnini izlash va joylashtirish	1,0	4,1	16,1

Bu yerda 1,0 qiymat eng kichik vaqt talab qilingan miqdor bo'lib, qolgan qiymatlar shunga nisbatan keltirilgan. Jadvaldan ko'rinib turibdiki, ikkinchi usuldagi yutuq uncha katta emas.

Yuqorida keltirilgan algoritmda $a[i]$ elementining o'rnini aniqlashda 4-bobda keltirilgan binar izlash algoritmidan foydalanish mumkin.

U holda ko'rib chiqilgan qism dasturlarimiz quyidagi ko'rinishda bo'lishadi.

```
function Binary_Search (last:word; var a:arrtype; m:word) : word;
{ bu yerda a[1...last]  va m – bu izlanayotgan element}
var L,r,mid:word;
begin
   L:=1;
  r:=last;
    while L<=r do
begin
        mid:=(L+r) div 2;
        if a[mid]>m then r:=mid-1
```

```
        else L:=mid+1
end;
 Binary_Search:=L;
{funksiya natijasi, a[] massivida m ning o'rni bo'ladi}
end;
```

Bevosita saralash qism dasturi esa quyidagicha bo'ladi.

```
procedure Binary_Insert_Sort(var a:arrtype);
var i:integer;
begin
 for i:=2 to n do
 if a[i]<a[i-1]  then  { a[i] tanlanildi }
{endi a[i] ni 1...i-1 oraliqga joylashtiramiz}
 Insert(a, i, Binary_Search(i-1,a,a[i]))
end;
```

Endi bevosita yuqorida keltirilgan dasturdagi *ShakeSort* qism dasturi o'rniga *Binary_Insert_Sort* qism dasturini joylashtiramiz (*Search* va *Insert* qism dasturlari bilan birgalikda) va natijani tekshiramiz.

```
{ Massivni binar izlash orqali  saralash dasturi }
program Binary_InsertSort_3;
const    Nmax = 100;
type
     arrtype = array[1..Nmax] of integer;
var
     n, i : integer;
     a : arrtype;
     fayl : text;
{qo'shimcha funksiyalar}
function  Binary_Search  (last:integer;  var  a:arrtype;  m:
integer):integer;
{ bu yerda a[1...last]  va m – bu izlanayotgan element}
var L,r,mid : integer;
begin
  L:=1;
 r:=last;
   while L<=r do
begin
   mid:=(L+r) div 2;
```

```
    if a[mid]>m then r:=mid-1  else  L:=mid+1
end;
 Binary_Search:=L;
{funksiya natijasi, a[] massivida m ning o'rni bo'ladi}
end;
procedure Insert (var a:arrtype; i, j:integer) ;
var tmp,k:integer;
begin
  tmp:=a[i];
{ a[i] elementini vaqtincha xotirada saqlaymiz }
  for k:=i downto j+1 do
{  i  dan  j+1  gacha elementlar indeksi }
  a[k]:=a[k-1];
{o'ngga bitta katakga siljitamiz}
  a[j]:=tmp;
{ a[i]  elementni  j-katakda joylashtiramiz}
end;
procedure Binary_Insert_Sort(var a:arrtype);
var i:integer;
begin
 for i:=2 to n do
 if a[i]<a[i-1]  then   { a[i] tanlanildi }
       {endi a[i] ni 1...i-1 oraliqga joylashtiramiz}
     Insert( a, i, Binary_Search(i-1, a, a[i]))
end;
{dasturning asosiy qismi}
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
       assign(fayl, 'input.txt');
       reset(fayl);
       read(fayl, n);
       for i:=1 to n do
       read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
       close(fayl);
       Binary_Insert_Sort ( a );
{chiqarish fayliga natijani chop etamiz}
       assign(fayl, 'output.txt');
       rewrite(fayl);
```

```
    for i :=1 to n do
    write (fayl, a[i], ' ');
    close(fayl);
end.
```

Oldingi boblarda keltirilgan algoritmlarning samaradorlik darajasidan kelib chiqqan holda, ushbu algoritmning samaradorlik darajasi $O(n \cdot logn)$ tartibli bo'ladi, bu esa $O(n^2)$ ga nisbatan ancha yaxshi hisoblanadi.

Umumiy dastur doirasida esa bari-bir har bir o'rin topilgandan so'ng qolgan elementlarni bittaga siljitish kerak bo'ladi. Shu bois ushbu algoritmning samaradorlik darajasi $O(n^2)$ ga yaqinlashib qoladi.

Quyidagi jadvalda yuqorida keltirilgan algoritmni taqqosiy ko'rsatkichlari keltirilgan.

Usul	Massiv elementlari soni		
	1000	2000	4000
Orasiga qo'yish	1,2	4,5	19,0
Binar orasiga qo'yish	1,0	3,5	14,2

Bu yerda keltirilgan qiymatlar eng yaxshi ko'rsatkichga , ya'ni 1,0 ga nisbatan keltirilgan. Binar izlash algoritmi bevosita orasiga qo'yish algoritmini takomillashtirsada samaradorlik tartibi ko'p o'zgargani yo'q.

5.7. Tezkor usul

Pufakcha usuli eng sust usullardan biri hisoblanadi. Chunki unda faqatgina yonma-yon turgan elementlar taqqoslanib va keyinchalik o'rinlari almashtiriladi. Ushbu jarayonni 1964 yilda Xoar modikatsiyalashtirgan, ya'ni taqqoslash va o'rin almashtirish uzoqda joylashgan elementlar orasida bajariladi. Bu usulni Xoar tezkor saralash deb nomlagan. Uning boshqa nomi – bu bo'laklab saralash usuli deb aytiladi.

Shunday qilib massiv elementlarini ikki qismga ajratamiz. Qandaydir k sonidan kichik elementlarni chap tomonda, undan kattalarini esa o'ng tomonda to'playmiz. Ushbu usulda k soni asosiy element deb nomlanadi.

Misol sifatida quyidagi massivni ko'rib chiqamiz

38, 8, 16, 6, 79, 76, 57, 24, 56, 2, 58, 48, 4, 70, 45, 47

Bu yerda asosiy element sifatida k=38 ni olamiz va quyidagi natijaga kelamiz

4, 8, 16, 6, 2, 24, 38, 57, 56, 76, 58, 48, 79, 70, 45, 47

Ushbu natijaga kelish uchun quyidagi ishlarni amalga oshirish kerak bo'ladi.

Chap tomon uchun *i* va o'ng tomon uchun *j* o'zgaruvchilarini kiritamiz. Boshlang'ich qadamda *i*=1 va *j*=*n* bo'ladi. Keyinchalik *k* bilan *a*[*i*] ni solishtiramiz. Agar *a*[*i*] > *k* bo'lsa, u holda *j*=*j*-*1* va keyingi *a*[*j*] ni *k* bilan solishtiramiz. Shunday qilib *j* ni kamaytirib oldik, ushbu jarayonni *a*[*j*]<=*k* bo'lganga qadar davom etamiz. Agar oxirgi tengsizlik bajariladigan bo'lsa, unda *a*[*j*] va *a*[*i*] larni almashtiramiz, 38 bilan 4 ni. Endi *i*=*i*+1 va *a*[*i*] ni *k* bilan solishtiramiz. *i* ni oshirib *a*[*i*] >= *k* gacha davom ettiramiz. Bu yerda 79 va 38 larni almashtiramiz. Yana *j* ni kamaytirib shu tariqa ishni *j*<=*i* bo'lganga qadar davom ettiramiz. Quyidagi jadvalda ushbu jarayon to'liq keltirilgan.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	j	Jarayon
1	**38**	**8**	**16**	**6**	**79**	**76**	**57**	**24**	**56**	**2**	**58**	**48**	**4**	**70**	**45**	**47**	**16**	j-kamaytiramiz
1	38														45		15	j-kamaytiramiz
1	38													70			14	j-kamaytiramiz
1	38												4				13	4<38
1	4												38				13	Almashtiramiz
1	4												38				13	i-oshiramiz
2		8											38				13	i-oshiramiz
3			16										38				13	i-oshiramiz
4				6									38				13	i-oshiramiz
5					79								38				13	79>38
5					38								79				13	Almashtiramiz
5					38							48					12	j-kamaytiramiz
5					38						58						11	j-kamaytiramiz
5					38					2							10	2<38
5					2					38							10	Almashtiramiz
6						76				38							10	i-oshiramiz
6						76				38							10	76>38
6						38				76							10	Almashtiramiz
6						38			56								9	j-kamaytiramiz
6						38		24									8	j-kamaytiramiz
6						38		24									8	38>24
6						24		38									8	Almashtiramiz
7							57	38									8	i-oshiramiz
7							57	38									8	57>38
7							38	57									8	Almashtiramiz

Natijada quyidagiga kelamiz:

4, 8, 16, 6, 2, 24, 38, 57, 56, 76, 58, 48, 79, 70, 45, 47

Shunday qilib biz massivni ikki bo'lakga ajratib oldik, lekin massiv saralangani yo'q. Tashkil qilingan massivlarning har biri bilan ushbu jarayonni takrorlaymiz. Natijada massivlarning bittadan elementdan tashkil topgunga qadar ushbu algoritmni davom ettiramiz va oxirida massiv saralangan hisoblanadi.

Quyida keltirilgan qism dasturda biz rekursiyadan foydalandik. Bunda massivning chap *L* va o'ng *r* chegaralarini uzatamiz.

```
procedure Partition2(L , r :integer; var a:arrtype);
Var i , j , m , tmp :integer;
begin
     if  L < r then
  {ushbu shart bajarilmasa, rekursiyani to'xtatamiz}
begin
      i := L; j := r; m := a [i] ;
repeat
while m<a[j] do
      j:=j-l;     {j-kamaytiramiz }
      if  i <= j  then
begin
  tmp:=a[i]; a[i]:=a[j]; a[j]:=tmp;  {-almashtirish}
  i := i + l
end;
while a [i] < m do
     i := i + l;   { i-oshiramiz }
    if  i <= j  then
begin
    tmp := a[il;a[i] := a[j] ;a[j] := tmp;
  { almashtirish }
  j := j – 1;
end
until  i > j;
    Partition2( L , j  , a );
{massivning chap qismi bilan ishlaymiz}
    Partition2(i,r,a);
{ massivning o'ng qismi bilan ishlaymiz }
end
end;
```

Ushbu qism dastur quyidagicha chaqiriladi:

```
procedure Quick_Sort(var a:arrtype);
begin
    Partition2( 1 , n , a)
end;
```

Endi bevosita yuqorida keltirilgan dasturdagi *ShakeSort* qism dasturi o'rniga *Quick_Sort* va unda qo'llanilgan qism dasturlarni joylashtiramiz va natijani tekshiramiz.

```
{ Massivni tezkor usuli orqali saralash dasturi }
program QuickSort;
const    Nmax = 100;
type
     arrtype = array[1..Nmax] of integer;
var
     n, i : integer;
     a : arrtype;
     fayl : text;
{qo'shimcha funksiyalar}
procedure Partition2(L , r :integer; var a:arrtype);
var i , j , m , tmp :integer;
begin
     if L < r  then
{ushbu shart bajarilmasa, rekursiyani to'xtatamiz}
begin
     i := L; j := r; m := a [i] ;
repeat
while m<a[j] do
     j:=j-l;     {j-kamaytiramiz }
     if i <= j  then
begin
     tmp:=a[i]; a[i]:=a[j]; a[j]:=tmp;   {-almashtirish}
     i := i + l
end;
while a [i] < m do
     i := i + l;   { i-oshiramiz }
     if i <= j  then
begin
     tmp := a[il;a[i] := a[j] ;a[j] := tmp;
{ almashtirish }
```

```
      j := j – 1;
end
until  i > j;
      Partition2( L , j  , a );
{massivning chap qismi bilan ishlaymiz}
      Partition2(i,r,a);
{ massivning o'ng qismi bilan ishlaymiz }
end
end;
procedure Quick_Sort(var a:arrtype);
begin
      Partition2( 1 , n , a)
end;
      {dasturning asosiy qismi}
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      for i:=1 to n do
      read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
      close(fayl);
      Quick_Sort( a );
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i :=1 to n do
      write (fayl, a[i], ' ');
      close(fayl);
end.
```

5.8. Birlashtirish usuli

Ushbu algoritmni yoritishdan oldin biz quyidagi misolni ko'rib chiqamiz.

Birlashtirish. Ikkita natural son n va m berilgan bo'lsin. Ularga mos quyidagi sonli massivlar berilgan $a[n]$ va $b[m]$, ular o'sish tartibi bilan

saralangan, ya'ni $a[1]<=a[2]<=. \ . \ .<=a[n]$ va $b[1]<=b[2]<=...$ $<=b[m]$. Ushbu ikki massivdan yangi $c[n+m]$ tartiblangan massivni tuzing, ya'ni $c[1]<=c[2]<=.. \ .<=c[n+m]$.

Masala yechimi. Ushbu misolni yechishda yuqorida keltirilgan saralashning qo'yish usulidan foydalanish mumkin, ya'ni masalan, $a[i]$ larni birin-ketin $b[]$ massiv elementlari orasiga qo'yib chiqishimiz mumkin. Lekin bu holda bajariladigan operatsiyalarning soni $n+m$ dan ko'p bo'ladi. Shu bois bu yerda boshqa yondashuvni taklif qilamiz.

Berilgan massivlarning birinchi elementlarini taqqoslaymiz, qaysi biri kichik bo'lsa, shu elementni $c[]$ massiviga kiritamiz va qaysi massiv elementini olsak o'sha massivning chap chegarasini oshirib boramiz. Natijada yana "birinchi" elementlar taqqoslanadi. Har bir taqqoslashda bitta element $c[]$ massiviga kiritiladi, shu bois ushbu algoritmning samaradorlik darajasi $O(n+m)$ tartibli bo'ladi. Shunday qilib quyidagi o'zgaruvchilarni kiritamiz:

i — $a[]$ massivida ko'riladigan element indeksi;
j — $b[]$ massivida ko'riladigan element indeksi;
k — $c[]$ massividagi qiymat beriladigan element indeksi.

Lekin bu yerda bitta massivning elementlarining tugallanishini nazorat qilish kerak bo'ladi, masalan quyidagi algoritmda

```
Agar  a[i ] > b[ j ] bo'lsa,
 u holda  c[ k ] := b[ j ] ; j:=j+1;
  aksincha
     c[ k ] := a[ i ];   i:=i+1;
```

biror-bir massivning elementlari tugab qolsa, natijada algoritm ishlamay qoladi.

Shu bois bu yerda quyidagicha yo'l tutamiz.

Agar $a[\ i\ ] > b[\ j\ ]$ bo'lsa, yoki $a[]$ massivi tugatilsa va $b[]$ massivi tugatilmagan bo'lsa, u holda $c[]$ massiviga $b[j]$ qiymatini kiritamiz, aks holda $a[i]$ elementini kiritamiz, ya'ni

```
Agar j<=m  va  ( i > n yoki a[ i ] > b[ j ] )  bo'lsa,
 u holda   c[ k ] := b[ j ];      j:=j+1;
  aksincha
     c[ k ] := a[ i ];  i:=i+1
```

Ushbu g'oyani quyidagi qism dasturda aks ettiramiz:

```
procedure Plus_1( n,m : integer ; var  a,b :arrtype; var    c :
newarrtype) ;
var i,j,k : integer;
begin
  i:=1;  j:=1;
  for  k:= 1 to n+m do
begin
  if (j<=m) AND ((i>n) OR (a[ i ] > b[ j ])) then
begin
      c[k]:=b[ j ] ;j:=j + 1
end
        else
begin
       c[ k ]:=a[ i ] ; i:=i+1 ;
end
end
end;
```

Endi ushbu g'oyani bevosita massivni saralashda qo'llaymiz. Birinchi o'tishda massivni har bir elementini alohida tartiblangan deb hisoblaymiz. Ikkinchi o'tishda yonma-yon turgan elementlarni guruhlab tartiblangan ikki elementdan iborat guruhlarni tashkil qilamiz. Elementlar soni toq bo'lsa oxirgi guruhda bitta element bo'ladi. Shu yo'l bilan guruhlarni yana birlashtiramiz. Oxir oqibat massivimiz to'liq saralangan bo'ladi. Bu yerda bizga faqatgina qo'shimcha massiv kerak bo'ladi. Demak berilgan *a*[] massivi *b*[] massivga guruhlanib yoziladi, keyin *b*[] massivni guruhlarini *a*[] massiviga yana guruhlab yozamiz va hokazo. Endi yordamchi qism dasturlarni yaratishni boshlaymiz. Birinchi navbatda *a*[] massivining ikki bo'lagini birlashtirib uni *b*[] massivga yozishni dasturini tuzamiz. Quyidagi kelishuvlarni belgilab olamiz, ya'ni *len1* va *len2* bular *a*[] massivining ikki bo'laklarini uzunligi, ular mos ravishda o'z navbatida *f1* va *f2* indekslardan boshlanadi. Guruhlangan elementlar *b*[] massivida *f1* dan boshlab yoziladi. Natijada quyidagi qism dasturni tavsiya etamiz.

```
Procedure Plus_2 (var  a :arrtype ; f1, len1, f2, len2 : integer ; var
b :arrtype);
var i,j,k : integer;
begin
  i:=f1;  j:=f2;
```

```
  for  k:= f1 to f1+len1+len2-1 do
begin
if (j<=f2+len2-1) AND ((i>f1+len1-1) OR (a[i]>a[j]))  then
begin
      b[k]:=a[ j ] ;j:=j + 1
end
        else
begin
       b[ k ]:=a[ i ] ; i:=i+1 ;
end
end;
end;
```

Ishni yengillashtirish maqsadida *n* sonini 2 ning qandaydir darajasi deb qabul qilamiz. Natijada guruhlar uzunligi bir xil bo'ladi. Bu yerda yana quyidagilarni kiritamiz:

len – birlashtiriladigan bo'laklarning uzunligi, *point* – birinchi guruhning birinchi elementining indeksi, *s* – yo'nalishni belgilaydi, ya'ni agar *s*=1 bo'lsa, u holda *a*[] massividan *b*[] massiviga, *s*=-1 bo'lsa, u holda teskaricha.

Shunday qilib saralashning birlashtirish usuli quyidagicha bo'ladi.

```
program PlusSort;
const  Nmax = 100;
type
      arrtype = array[1..Nmax] of integer;
var n, i : integer;
      a, b : arrtype;
      fayl : text;
  {qo'shimcha funksiyalar}
Procedure Plus_2 (var  a :arrtype ; f1, len1, f2, len2 : integer ; var
b :arrtype);
var i,j,k : integer;
begin
      i:=f1;  j:=f2;
      for  k:= f1 to f1+len1+len2-1 do begin
if (j<=f2+len2-1) AND ((i>f1+len1-1) OR (a[i]>a[j])) then
begin
      b[k]:=a[ j ] ;j:=j + 1
end
      else
```

```
begin
      b[ k ]:=a[ i ] ; i:=i+1 ;
end
end;
end;
Procedure Count_Len(point,len:integer;var len1, len2 :integer);
{–len1 va len2 uzunligini aniqlovchi qism dastur}
begin
      If n-point+1>len then
{ ikki bo'lakni birlashtirish }
begin
      Len1:=len;   Len2:=n-(point+len-1)
end
      else
begin
      Len1:=n-point+1;  Len2:=0
end
end;
Procedure Plus_Sort(var a,b:arrtype);
var point,len,len1,len2,s,i:integer;
begin
      Len:=1;    S:=1;
Repeat
      Point:=1;
While n-point + 1 >= 2*len do
{ikki bo'lakni birlashtirish}
      if s=1 then
begin
      Plus_2(a,point,len,point+len,len,b) ;
      point:=point+2*len
end
      else
begin
      Plus_2(b,point,len,point+len,len,a);
      point:=point+2*len
end;
      if point<=n then
{ massiv elementlari tugalanmagan bo'lsa davom etamiz}
begin
```

```
{bo'laklarning uzunligini aniqlaymiz }
      Count_Len(point,len,len1,len2);
      if s=1 then
      Plus_2(a,point,len1,point+len1,len,b)
      else
      Plus_2(b,point,len1,point+len1,len,a);
end;
      Len:=2*len;   S:=-s;
Until len>=n;
        {Saralangan massiv}
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      If s=1 then
      for i:=1 to n do write(fayl, a[i], ' ')
      else
      for i:=1 to n do write(fayl, b[i], ' ');
      close(fayl);
end;
{dasturning asosiy qismi}
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      for i:=1 to n do
begin
      read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
      b[i]:=0;
end;
      close(fayl);
      Plus_Sort( a,b);
end.
```

Ehtiyot bo'ling, dasturda biz *n* sonini 2 ning qandaydir darajasi deb qabul qilganmiz, *n* ning boshqa qiymatlari uchun dastur xato ishlaydi. Ushbu kamchilikni tuzatishni mustaqil hal qilishni tavsiya qilamiz.

5.9. Shell usuli

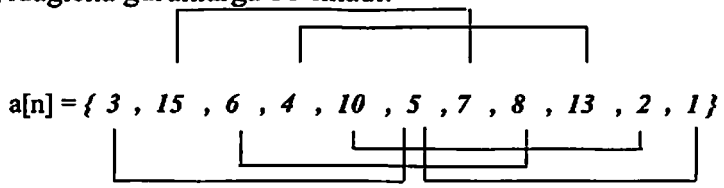
Ushbu usul bevosita o'rin almashtirish algoritmiga asoslangan bo'lib, faqatgina bu yerda yonma-yon turgan elementlar emas, balkim bir-biridan ma'lum masofada joylashgan elementlar solishtiriladi va zarur bo'lsa o'rinlari almashtirilib tartiblanadi. Bu yerda har bir guruh elementlarini tartiblash uchun orasiga qo'yish usuli qo'llaniladi.

Shunday qilib, $a[n]$ massivi $n/2$ ta guruhga , n juft bo'lsa, toq bo'lsa esa
$(n-1)/2$ ta guruhga taqsimlanadi. Demak, har bir guruh ikki elementdan tashkil topgan bo'ladi, n toq bo'lsa, u holda bitta guruhda 3 ta element bo'ladi.

Guruhlar doirasida elementlar bir-biridan $n/2$ uzoqlikda joylashgan bo'ladi. Bu masofa qadam deyiladi. Misol sifatida quyidagi massivni ko'rib chiqamiz ($n=11$)

$$a[n] = \{3, 15, 6, 4, 10, 5, 7, 8, 13, 2, 1\}$$

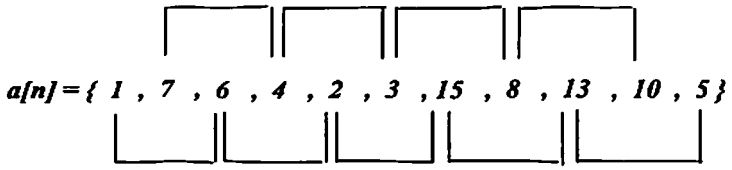
Bu yerda qadam $d=n/2\approx5$ ga teng bo'ladi. Elementlar ushbu qadam bilan quyidagicha guruhlarga bo'linadi:



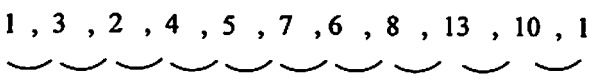
Bu yerda birinchi guruhga mansub elementlar 3, 5, 1 lar bo'ladi. Ularni pufakcha usuli (yoki boshqa usul) bilan tartiblaymiz, natijada 1, 3, 5 ni hosil qilamiz. Ikkinchi guruhga mansub elementlar 15, 7 o'z o'rinlari bilan almashtiriladi. Xuddi shu algoritmni boshqa guruh elementlari uchun qo'llaymiz va natijada 1-o'tishda massiv quyidagi ko'rinishda bo'ladi:

$$a[n] = \{1, 7, 6, 4, 2, 3, 15, 8, 13, 10, 5\}$$

Shell tavsiyasi bo'yicha qadamni $n/2$, $n/4$, $n/8$,... deb olamiz, demak 2- o'tishda qadam $n/4\approx2$ deb qabul qilamiz. Natijada quyidagi ikki guruhni tashkil qilamiz:



Bu yerda hosil qilingan ikki guruh elementlari 1, 6, 2, 15, 13, 5 va 7, 4, 3, 8, 10 alohida tartibga solinadi. Tartibga solingan elementlar o'z guruhi elementlari o'rinlarida joylashtiriladi, natijada quyidagini hosil qilamiz:



Uchinchi o'tishda qadam $n/8 \approx 1$ ga teng deb olinadi. Bu esa massiv bitta yagona guruhni tashkil etadi. Bu yerda birin-ketin solishtirishlar va o'rin almashtirishlar orqali dastlabki massiv elementlari to'la tartibga solingan bo'ladi. Demak uchinchi o'tishda quyidagi natijani olamiz:

$$a[n] = \{1, 2, 3, 4, 5, 6, 7, 8, 10, 13, 15\}$$

Umumiy holda n ta elementdan iborat massivni tartib bilan saralash uchun $log_2 n$ ta o'tish kerak bo'ladi. Har bir o'tishda n ta element bir-biri bilan taqqoslaniladi, demak barcha solishtirishlar ushbu algoritmda $O(n \cdot log_2 n)$ tartibli bo'ladi. Lekin qadamlarni d boshqacha tanlab samaradorlik tartibini optimallashtirish ham mumkin bo'lar ekan.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
11 3 15 6 4 10 5 7 8 13 2 1	1 2 3 4 5 6 7 8 10 13 15

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida saralangan massiv elementlari qiymatlari chiqarilgan bo'ladi.

Paskal tilidagi dasturi esa quyidagicha bo'ladi:

```
{ Shell bo'yicha saralash }
program ShellSort;
const  Nmax = 100;
type
     arrtype = array[1..Nmax] of integer;
```

```
var
      n, i : integer;
      a : arrtype;
      fayl : text;
  {qo'shimcha funksiyalar}
Procedure Sort_Shell( var a : arrtype);
var
      d, i, t : integer;
      k : boolean; { almashtirish belgisi }
begin
      d:=N div 2; { boshlang'ich qadam }
 while d>0 do  { qadamni kamayishi bo'yicha sikl }
begin
       { saralash }
      k:=true;
while k do { almashtirishlar bo'yicha sikl }
begin
      k:=false; i:=1;
      for i:=1 to N-d do
begin
       { ikki elementni taqqoslash }
      if a[i]>a[i+d] then
begin
      t:=a[i];
      a[i]:=a[i+d];
      a[i+d]:=t; { -almashtirish }
      k:=true;
end; { if ... }
end; { for ... }
end; { while k }
      d:=d div 2; { qadamni kamaytiramiz }
end; { while d>0 }
end;
  {dasturning asosiy qismi}
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
```

```
    for i:=1 to n do
    read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
    close(fayl);
    Sort_Shell ( a );
{chiqarish fayliga natijani chop etamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
    for i :=1 to n do
    write (fayl, a[i], ' ');
    close(fayl);
end.
```

5.10. Piramida usuli

Ushbu usul samarali usullardan biri hisoblanadi, uning samaradorlik darajasi $O(nlog\ n)$ deb hisoblanadi. Misol sifatida quyidagi massivni ko'rib chiqamiz:

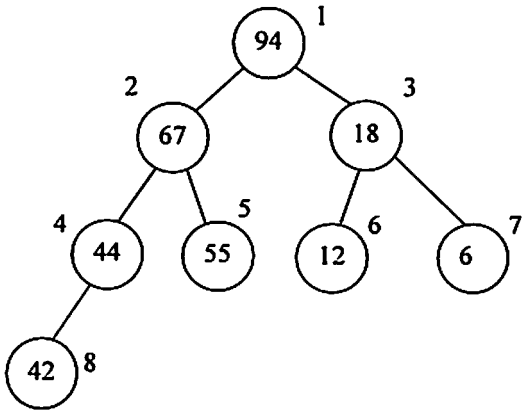
$$a[n] = \{44, 55, 12, 42, 94, 18, 6, 67\}$$

Oldiniga piramida tushunchasini kiritamiz. Har qanday k balandlikga ega bo'lgan ikkilik daraxti piramida deyiladi, agarda unda quyidagilar bajarilsa:

- k-1 pog'onagacha to'ldirilgan bo'lib, k-pogona esa chapdan o'ngga to'ldirilgan;
- har bir tugun qiymati, undan yuqorida joylashgan elementdan kichik.

Piramida bilan massivni quyidagicha bog'lash mumkin:

- $a[0]$ elementida daraxtning ildizini saqlaymiz;
- $a[i]$ elementining "farzandlari" $a[2i+1]$ va $a[2i+2]$ elementlarida saqlanadi.



Demak piramidani massivda saqlaydigan bo'lsak, unda quyidagi xususiyatlar mavjud bo'ladi: $a[i] \geq a[2i+1]$ va $a[i] \geq a[2i+2]$. Piramidani bunday tarzda saqlash quyidagi afzalliklarga ega:

- qo'shimcha o'zgaruvchilarni kiritish shart emas, faqatgina elementlarni joylashuvini tushunib olsak kifoya;
- tugunlar yuqoridan pastga qarab joylashtirilgan;
- bir pog'onadagi tugunlar massivda ketma-ket joylashgan bo'ladi.

Yuqorida keltirilgan piramidani quyidagi massivda saqlash mumkin (faqatgina e'tibor bering, massiv elementlari 0-indeksdan boshlansa, piramidada biz tugunlarni 1 dan sanab boshlaymiz) : 94, 67, 18, 44, 55, 12, 6, 42. Ushbu elementlar piramidan chapdan o'ngga va yuqoridan pastga qarab o'qiladi. Endi massivdan piramidani tiklash qiyinchilik tug'dirmaydi.

Piramidani qurishni $a[k]$ elementdan boshlab $a[n]$ gacha davom etamiz, bu yerda $k=[n/2]$ bo'ladi. Chunki undagi har qanday i-element uchun $2i+1$ va $2i+2$ massiv chegarasidan chiqib ketadi. Kengaytirish jarayonida piramida xossasini saqlab qolish zarur, shu bois quyidagicha yo'l tutamiz, ya'ni $a[i+1]$ ... $a[n]$ piramidaga $a[i]$ ni quyidagi algoritm bo'yicha qo'shib qo'yamiz:

1. $a[i]$ elementining "farzandlari" $a[2i+1]$ va $a[2i+2]$ bo'ladi, ulardan kattasini tanlaymiz
2. agar tanlangan element $a[i]$ dan katta bo'lsa, ularni o'rnini almashtiramiz va yana 2-qadamni bajaramiz, chunki $a[i]$ ning o'rni o'zgardi. Aks holda qism dasturni tugatamiz.

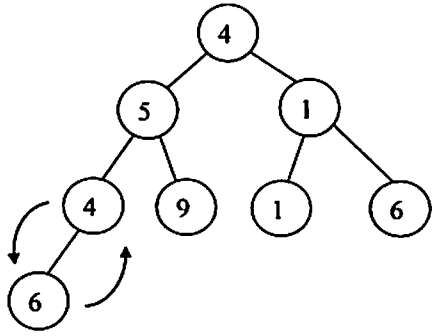
```
procedure downHeap(var a: arrtype; k,n: integer);
{ mavjud a[k+1]...a[n]  piramidadan }
```

```
{yangi a[k]...a[n] - piramidani yaratish qism dasturi }
Var
  new_elem, child: integer;
   flag: boolean;
begin
   new_elem := a[k];
   flag:=true;
while(k <= n/2) and flag do
{ a[k] ning farzandlari bo'lsa }
begin
    child := 2*k;
    {  kattasini tanlaymiz}
    if( child < n) and (a[child] < a[child+1] ) then
     inc(child);
    if( new_elem >= a[child] ) then flag:=false
    else    { aksincha}
begin
      a[k] := a[child];
{ farzandni yuqoriga chiqaramiz}
      k := child;
end;
end;
   a[k] := new_elem;
end;
```

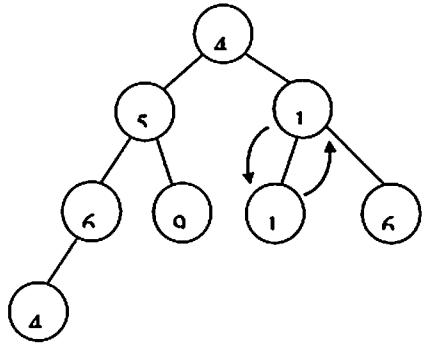
Ushbu jarayonni quyidagi misolda ko'rish mumkin

```
 44 55 12 42 // 94 18 06 67   {bu yerda // belgisidan o'ng tomonda
 44 55 12 // 67 94 18 06 42   turgan elementlar piramida xossasini
 44 55 // 18 67 94 12 06 42   qanoatlantiradi}
 44 // 94 18 67 55 12 06 42   {qolgan elementlar o'ngdan chapga
 // 94 67 18 44 55 12 06 42   qo'shilib boriladi  }
```

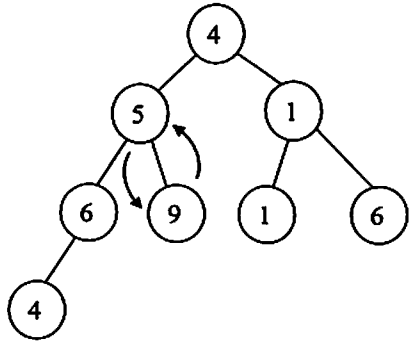
Quyidagi rasmlarda piramidani qadamba-qadam qurish keltirilgan, bu birinchi faza hisoblanadi.



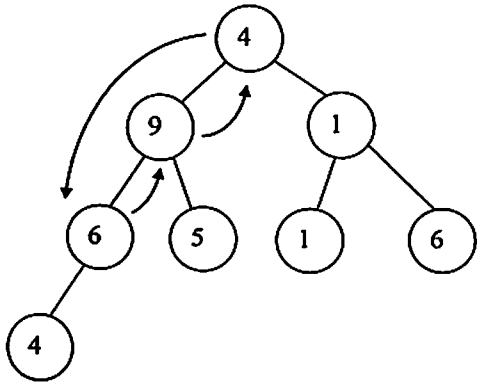
Boshlang'ich qadam.
42 bilan 67 ni taqqoslaymiz va almashtiramiz



12 bilan max(18,6) ni taqqoslaymiz va almashtiramiz

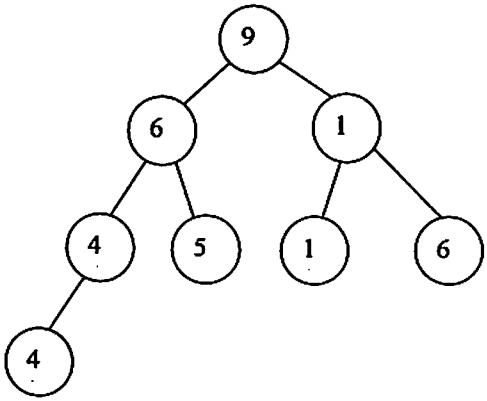


55 bilan max(67,94) ni taqqoslaymiz va almashtiramiz



44 ni 94 va 67 lardan o'tkazib 42 da to'xtaldik

Natijada quyidagi piramida hosil bo'ladi



Shunday qilib berilgan massivdan biz piramidani yaratdik, endi 2-fazani, ya'ni bevosita saralashni amalga oshiramiz.

Piramidaning xususiyati – bu uning ildizida eng katta element joylashganligi bo'ladi.

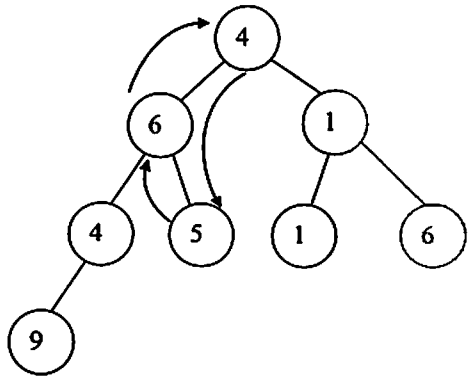
Shundan foydalanib, saralashni quyidagi algoritmini keltiramiz:

1. yangi massivning $a[0]$ ...$a[n]$ birinchi elementi bu ildiz, ya'ni eng kattasi, shu bois uni oxirgi element bilan qiymatlarini o'zaro almashtiramiz. Vaqtincha n-elementni "unutamiz" va quyidagi $a[0]$ ...$a[n-1]$ massivdan piramidani tuzamiz. Buning uchun yuqoridagi yangi birinchi elementni pastga tushirib yuborish kerak bo'ladi;

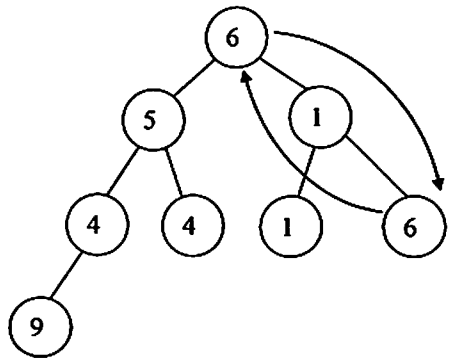
2. bitta element qolgan bo'lsa ishni tugatamiz, aks holda yana 1-qadamga qaytib jarayonni davom ettiramiz.

Natijada har safar piramida hosil bo'laveradi va uning oxirgi elementi maksimal qiymatga ega bo'ladi.

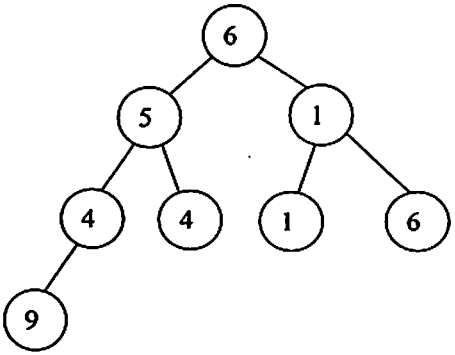
Shu bois tartiblangan ketma-ketlik hosil bo'ladi.



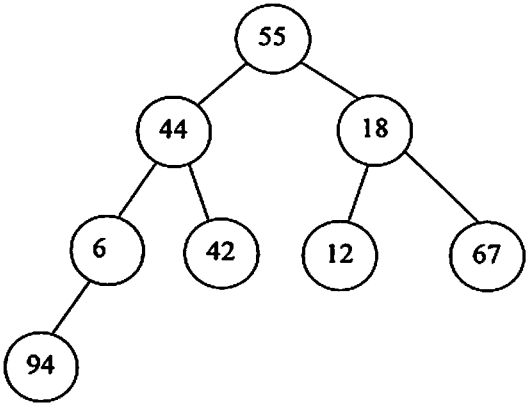
42 ni 67,55 lardan o'tkazib tushiramiz



6 ni 67 bilan almashtiramiz



6 ni 55,44 lardan o'tkazib tushiramiz



Ushbu jarayonni bevosita massivda ko'rish mumkin bo'ladi

94 67 18 44 55 12 06 42 //
67 55 44 06 42 18 12 // 94
55 42 44 06 12 18 // 67 94
44 42 18 06 12 // 55 67 94
42 12 18 06 // 44 55 67 94
18 12 06 // 42 44 55 67 94
12 06 // 18 42 44 55 67 94
06 // 12 18 42 44 55 67 94

Asosiy qism dasturimiz esa quyidagi ko'rinishda bo'ladi:

```
procedure Piramida_Sort(var a: arrtype; size: integer);
var
   i, temp: integer;
```

```
begin
    { piramidani yasaymiz }
    for i:=size div 2 - 1 downto 0 do
    downHeap(a, i, size-1);
{ endi  a[0]...a[size-1] piramida}
     for i:=size-1 downto 1 do
begin
{ birinchi bilan oxirgisini almashtiramiz }
      temp:=a[i]; a[i]:=a[0]; a[0]:=temp;
{  a[0]...a[i-1] ketma-ketligini piramida qilamiz}
      downHeap(a, 0, i-1);
end
end;
```

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
8 44 55 12 42 94 18 6 67	6 12 18 42 44 55 67 94

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida saralangan massiv elementlari qiymatlari chiqarilgan bo'ladi.

Piramida usuli bo'yicha massivni saralaydigan asosiy dasturimiz esa quyidagi ko'rinishda bo'ladi:

```
program PiramidaSort;
const  Nmax = 100;
type
      arrtype = array[0..Nmax-1] of integer;
var
      n, i : integer;
      a : arrtype;
      fayl : text;
  {qo'shimcha funksiyalar}
procedure downHeap(var a: arrtype; k,n: integer);
 { mavjud  a[k+1]...a[n]  piramidadan }
{yangi a[k]...a[n] - piramidani yaratish qism dasturi }
var
```

```
        new_elem, child: integer;
        flag: boolean;
begin
        new_elem := a[k];
        flag:=true;
while(k <= n/2) and flag do
{ a[k] ning farzandlari bo'lsa }
begin
        child := 2*k;
{   kattasini tanlaymiz}
        If ( child < n) and (a[child] < a[child+1] ) then
        inc(child);
        if( new_elem >= a[child] ) then flag:=false
        else    { aksincha}
begin
        a[k] := a[child];
{ farzandni yuqoriga chiqaramiz}
        k := child;
end;
end;
        a[k] := new_elem;
end;
procedure Piramida_Sort(var a: arrtype; size: integer);
var
        i, temp: integer;
begin
{ piramidani yasaymiz }
        for i:=size div 2 - 1 downto 0 do
        downHeap(a, i, size-1);
        { endi  a[0]...a[size-1] piramida}
        for i:=size-1 downto 1 do

begin
{ birinchi bilan oxirgisini almashtiramiz }
        temp:=a[i]; a[i]:=a[0]; a[0]:=temp;
{  a[0]...a[i-1] ketma-ketligini piramida qilamiz}
        downHeap(a, 0, i-1);
end
end;
```

```
  {dasturning asosiy qismi}
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      for i:=0 to n-1 do
      read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
      close(fayl);
      Piramida_Sort ( a, n);
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i :=0 to n-1 do      write (fayl, a[i], ' ');
      close(fayl);
end.
```

E'tibor bering, bu erda *a*[] massivining indeksi 0 dan boshlanadi.

5.11. Saralash usullarining solishtirma ko'rsatkichlari

Yuqorida ko'rib chiqilgan saralash usullarining turli xil modifikatsiyalari ham mavjud. Biz bu yerda tashqi saralash usullariga to'xtalib o'tirmadik. Ushbu usullarni mustaqil o'rganishni tavsiya etamiz.

Saralashning qaysi usulini tanlash va uni baholashda ta'sir etuvchi omillarni takroran eslatib o'tamiz:

1) **Xotira**. Juda katta o'lchamli massiv uchun, qo'shimcha massivlarni kiritilishi tezkor xotiraning hajmi yetmasligiga olib kelishi mumkin;
2) **Vaqt**. Katta hajmdagi massivlar bilan ishlash vaqt muammosini hosil qiladi, shu bois vaqt bo'yicha optimal algoritmlarni qo'llash tavsiya etiladi;
3) **Tabiiylik**. Berilgan massiv elementlarini qay darajada tartibsiz joylashganligi uni tartiblash jarayoniga ta'sirini kuchaytiradi. Ushbu omilni **tabiiylik** deb aytishadi, ya'ni qariyb tartiblangan massiv uchun ko'p vaqt sarflanmasligi zarur.
4) **Turg'unlik**. Massivda mavjud bo'lgan teng elementlar o'zaro ketma-ket joylashuvi o'zgarmas qolishi turg'unlik deb aytiladi.

Albatta yuqorida keltirilgan dasturlar bevosita inson tomonidan yozilgan. Demak ular qay darajada yaxshi tuzilganligiga bog'liq bo'ladi. Ya'ni o'zgaruvchilarni to'g'ri ta'riflashda, oddiy arifmetik amallardan foydalanish, sikl jarayonlarini to'g'ri tashkil etish va hokazo.

Ko'rib chiqilgan saralash algoritmlarini ushbu omillar bo'yicha farqlanishini quyidagi jadvalda keltiramiz:

Usul	Xotira	Vaqt	Tabiiylik	Turg'unlik
Orasiga qo'yish	Oz	$O(n^2)$	+	+
Tanlash	Oz	$O(n^2)$	-	-
Pufakcha	Oz	$O(n^2)$	+	+
Sheyker	Oz	$O(n^2)$	+	-
Shell	Oz	O(n log n)	-	-
Piramida	Oz	O(n log n)	-	-
Birlashtirish	Ko'p	O(n log n)	-	+
Tezkor	Ko'p	O(n log n)	-	-
Hisoblash	Ko'p	$O(n^2)$	+	+

Yuqoridagi keltirilgan usullarni bir-biri bilan taqqoslashda ko'pincha massivning uzunligi va undagi tartib bor-yo'qligi asos qilib olinadi, har xil elementlar soni bilan o'tkazilgan tekshiruvlar natijasi quyidagi jadvalda keltirilgan:

Saralash usuli	Elementlar soni	Oldindan tartiblan	Ixtiyoriy	Teskari tartiblangan
Orasiga qo'yish	Oz	2,4	6,1	19,0
	Ko'p	4,6	24,1	76,6
Tanlash	Oz	97,8	8,5	18,8
	Ko'p	338,4	32,6	72,3
Pufakli	Oz	108,0	17,1	40,3
	Ko'p	433,0	67,6	160,3
Sheyker	Oz	1,0	16,0	43,8
	Ko'p	1,8	60,7	176,2
Shell	Oz	11,6	2,1	4,2
	Ko'p	23,1	5,8	13,3
Piramida	Oz	23,2	1,8	2,8
	Ko'p	50,1	4,0	6,1
Birlashtirish	Oz	19,8	1,7	2,7
	Ko'p	46,8	4,0	6,3
Tezkor	Oz	6,2	1,0	1,0
	Ko'p	18,6	2,4	2,1
Hisoblash	Oz	216,4	18,1	80,4
	Ko'p	461,8	95,6	160,3

Har bir ustundagi 1,0 qiymati uning vaqt bo'yicha eng tezkor usul ekanligini bildiradi. Ushbu qiymatga nisbatan boshqa usullarning nisbiy samaradorligi keltirilgan.

Ushbu jadvaldan eng maqbul usul sifatida tezkor usulni ko'rsatish mumkin, shu bois olimpiada masalalarida vaqt bo'yicha chegara qo'yiladigan bo'lsa, u holda tezkor usulni qo'llash tavsiya etiladi.

Topshiriqlar

1. $a_1,a_2,\ldots,a_n$ natural sonlar ketma-ketligi berilgan. Ular o'sish tartibida ekanligini aniqlang.
2. Natural sonlardan tashkil topgan $A[n]$ massivini B va C massivlarga shunday bo'laklash kerakki, ulardagi sonlar yig'indisi teng bo'lsin.
Izoh. Variantlarni saralamasdan oldin $A[n]$ massivini kamayish bo'yicha tartiblash kerak.
3. Butun sonlardan tashkil topgan massivda $A[M,N]$ elementlar qatorlar va ustunlar bo'yicha o'sib borish bo'yicha tartiblangan. Unda K soni mavjudligini aniqlang va algoritm samaradorlik darajasi $O(M+N)$ tartibli bo'lsin.
4. Futbol komandasining o'yinchilari reytingi $R[N]$ massivida butun sonlar bilan berilgan. Har bir o'yinchining reytingi R_i istalgan ikki o'yinchining reytinglari yig'indisidan kichik, ya'ni $R_i \le R_k+R_m$. O'yinchilarni shunday tanlash kerakki komandaning umumiy reytingi eng katta bo'lsin.
5. Natural sonlardan tashkil topgan $A[n]$ va $B[m]$ massivlaridan bir xil sonlarni o'chirib, yangi $C[n+m]$ massivini tashkil qiling.
6. Natural sonlardan tashkil topgan $A[n]$ massividan nollarni o'chirib, tartiblangan yangi $C[n]$ massivini tashkil qiling.
7. Temiryo'l vokzalida m ta yo'lak bo'lib, ularda n ta vagon bor. Ularni tartib bilan joylashtiring, ya'ni 123456789...n.
8. Natural sonlardan tashkil topgan $A[n]$ massivi har xil sonlardan iborat ekanligini aniqlang.
9. Har xil og'irlikdagi to'rtta tarvuzni, oddiy ikki pallali tarozi yordamida, faqatgina 5 marotaba o'lchash orqali, ularni og'irliklari o'sish tartibi bilan joylashtiring.
10. Natural sonlardan tashkil topgan juftliklar (a,b) to'plami (bu yerda (a,b) teng emas (b,a) ga) tranzitiv ekanligini aniqlang, ya'ni agar (a,b)

Albatta yuqorida keltirilgan dasturlar bevosita inson tomonidan yozilgan. Demak ular qay darajada yaxshi tuzilganligiga bog'liq bo'ladi. Ya'ni o'zgaruvchilarni to'g'ri ta'riflashda, oddiy arifmetik amallardan foydalanish, sikl jarayonlarini to'g'ri tashkil etish va hokazo.

Ko'rib chiqilgan saralash algoritmlarini ushbu omillar bo'yicha farqlanishini quyidagi jadvalda keltiramiz:

Usul	Xotira	Vaqt	Tabiiylik	Turg'unlik
Orasiga qo'yish	Oz	$O(n^2)$	+	+
Tanlash	Oz	$O(n^2)$	-	-
Pufakcha	Oz	$O(n^2)$	+	+
Sheyker	Oz	$O(n^2)$	+	-
Shell	Oz	O(n log n)	-	-
Piramida	Oz	O(n log n)	-	-
Birlashtirish	Ko'p	O(n log n)	-	+
Tezkor	Ko'p	O(n log n)	-	-
Hisoblash	Ko'p	$O(n^2)$	+	+

Yuqoridagi keltirilgan usullarni bir-biri bilan taqqoslashda ko'pincha massivning uzunligi va undagi tartib bor-yo'qligi asos qilib olinadi, har xil elementlar soni bilan o'tkazilgan tekshiruvlar natijasi quyidagi jadvalda keltirilgan:

Saralash usuli	Elementlar soni	Oldindan tartiblan	Ixtiyoriy	Teskari tartiblangan
Orasiga qo'yish	Oz	2,4	6,1	19,0
	Ko'p	4,6	24,1	76,6
Tanlash	Oz	97,8	8,5	18,8
	Ko'p	338,4	32,6	72,3
Pufakli	Oz	108,0	17,1	40,3
	Ko'p	433,0	67,6	160,3
Sheyker	Oz	1,0	16,0	43,8
	Ko'p	1,8	60,7	176,2
Shell	Oz	11,6	2,1	4,2
	Ko'p	23,1	5,8	13,3
Piramida	Oz	23,2	1,8	2,8
	Ko'p	50,1	4,0	6,1
Birlashtirish	Oz	19,8	1,7	2,7
	Ko'p	46,8	4,0	6,3
Tezkor	Oz	6,2	1,0	1,0
	Ko'p	18,6	2,4	2,1
Hisoblash	Oz	216,4	18,1	80,4
	Ko'p	461,8	95,6	160,3

Har bir ustundagi 1,0 qiymati uning vaqt bo'yicha eng tezkor usul ekanligini bildiradi. Ushbu qiymatga nisbatan boshqa usullarning nisbiy samaradorligi keltirilgan.

Ushbu jadvaldan eng maqbul usul sifatida tezkor usulni ko'rsatish mumkin, shu bois olimpiada masalalarida vaqt bo'yicha chegara qo'yiladigan bo'lsa, u holda tezkor usulni qo'llash tavsiya etiladi.

Topshiriqlar

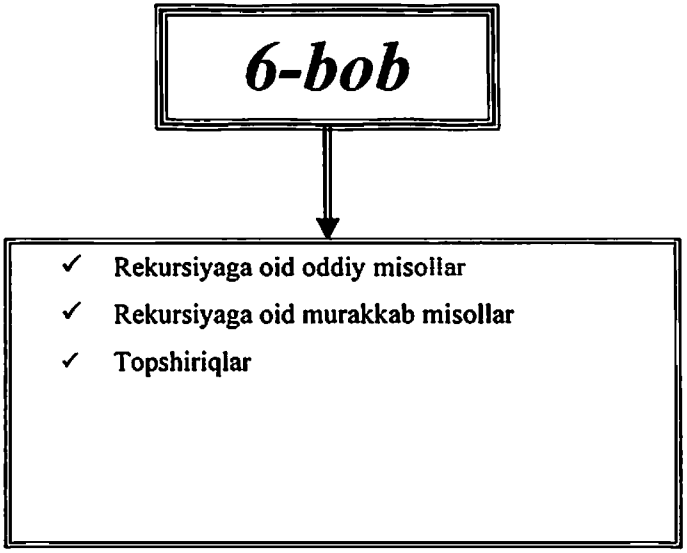
1. $a_1,a_2,\ldots,a_n$ natural sonlar ketma-ketligi berilgan. Ular o'sish tartibida ekanligini aniqlang.
2. Natural sonlardan tashkil topgan $A[n]$ massivini B va C massivlarga shunday bo'laklash kerakki, ulardagi sonlar yig'indisi teng bo'lsin.
Izoh. Variantlarni saralamasdan oldin $A[n]$ massivini kamayish bo'yicha tartiblash kerak.
3. Butun sonlardan tashkil topgan massivda $A[M,N]$ elementlar qatorlar va ustunlar bo'yicha o'sib borish bo'yicha tartiblangan. Unda K soni mavjudligini aniqlang va algoritm samaradorlik darajasi $O(M+N)$ tartibli bo'lsin.
4. Futbol komandasining o'yinchilari reytingi $R[N]$ massivida butun sonlar bilan berilgan. Har bir o'yinchining reytingi R_i istalgan ikki o'yinchining reytinglari yig'indisidan kichik, ya'ni $R_i \le R_k+R_m$. O'yinchilarni shunday tanlash kerakki komandaning umumiy reytingi eng katta bo'lsin.
5. Natural sonlardan tashkil topgan $A[n]$ va $B[m]$ massivlaridan bir xil sonlarni o'chirib, yangi $C[n+m]$ massivini tashkil qiling.
6. Natural sonlardan tashkil topgan $A[n]$ massividan nollarni o'chirib, tartiblangan yangi $C[n]$ massivini tashkil qiling.
7. Temiryo'l vokzalida m ta yo'lak bo'lib, ularda n ta vagon bor. Ularni tartib bilan joylashtiring, ya'ni 123456789...n.
8. Natural sonlardan tashkil topgan $A[n]$ massivi har xil sonlardan iborat ekanligini aniqlang.
9. Har xil og'irlikdagi to'rtta tarvuzni, oddiy ikki pallali tarozi yordamida, faqatgina 5 marotaba o'lchash orqali, ularni og'irliklari o'sish tartibi bilan joylashtiring.
10. Natural sonlardan tashkil topgan juftliklar (a,b) to'plami (bu yerda (a,b) teng emas (b,a) ga) tranzitiv ekanligini aniqlang, ya'ni agar (a,b)

va (b,c) juftliklar to'plamga tegishli bo'lsa , u holda (a,c) ham to'plamga tegishli bo'ladi.

11. Maktab o'quvchilari 2 ta qo'shni mahallada yashashadi. Maktabgacha borish uchun birinchi mahalladan S_1 va ikkinchi mahalladan S_2 vaqt talab etiladi. O'quvchilar soni N ta bo'lsa va har birining uydan chiqish vaqti berilgan bo'lsa, ularni qaysi tartibda maktabga kelishini aniqlang.

6-bob. DASTURLASHDA REKURSIYA TUSHUNCHASI

Mazkur bobda dasturlashtirishda ajoyib ko'rinishga ega bo'lgan dasturlarni yaratish imkonini beruvchi rekursiya haqida so'z olib boriladi. Bunda rekursiya tushunchasi, turlari va tuzilishi kabi ma'lumotlar bo'lib, unda quyidagi bo'limlar berilgan:



6.1. Rekursiyaga oid oddiy misollar

Ko'p hollarda dasturlash jarayonida biz iteratsiyani qo'llaymiz, ya'ni siklik jarayonlarni tashkil qilamiz. Lekin bu rekursiya emas. Rekursiyada ob'ektni ta'riflash unga murojaat qilish orqali amalga oshiriladi, ya'ni biz ta'riflamoqchi bo'lgan ob'ektimiz yana ushbu ob'ekt orqali ta'riflanadi.

Endi ushbu fikrlarni aniq belgilab olamiz. To'plam uchun kiritilayotgan ta'rif rekursiv deb hisoblanadi, agarda to'plam elementlarini aniqlash ushbu to'plamning boshqa elementlari orqali berilgan bo'lsa. Shundan kelib chiqqan holda, agarda ob'ektlar rekursiv berilgan bo'lsa, u holda ob'ekt rekursiv hisoblanadi. Demak, rekursiya – bu rekursiv ta'rifni qo'llash demakdir. Bu yerda misol sifatida, ko'pincha faktorialni hisoblash keltiriladi, ya'ni, har qanday n natural soni uchun $n!=n\cdot(n-1)!$ va $0!=1$ formulalari orqali faktorial ta'riflanadi. Bundan quyidagi qiymatlarni hosil qilish mumkin : $0!=1$, $1=1*0!$, $2=2*1!$, $6=3*2!$ bu esa $\{1,2,6,\ldots\}$ to'plamining elementlari va uning birinchi elementidan tashqari barcha elementlari rekursiv aniqlangan.

Rekursiyaga bog'liq qiziq misollardan yana biri bu "+" amali orqali berilgan va faqat sonlardan va qavslardan iborat arifmetik ifoda (QAI) hisoblanadi. Bu yerda quyidagilar bajariladi:

1) har qanday son bu QAI;
2) agar A va B bevosita QAI tegishli bo'lsa, u holda $(A)+(B)$ ham QAI ga tegishli bo'ladi;

Quyidagi misollar bunday ifodalarga misol bo'la oladi: 1, 2, (1)+(2), ((1)+(2))+(1), bu yerda 1 va 2 dan tashqari barcha ifodalar rekursiv tashkil etilgan.

Rekursiya ta'rifida berilgan element bevosita yoki bilvosita yana ushbu elementga murojaat qilishi mumkin emas. Misol sifatida faktorialni ozgina o'zgartirilgan holda keltiramiz: $n>0$ uchun $n!=n\cdot(n-1)!$ va nol uchun $0!=1!$ deb qabul qilamiz. Natijada $1!$ ni hisoblashda $0!$ ga murojaat qilinadi, u esa o'z navbatida yana $1!$ ga murojaat qiladi. Natijada bu yerda cheksiz jarayon hosil bo'ladi. Bu esa noto'g'ri.

Rekursiya – noma'lumlar qiymatlari biri ikkinchisidan ketma-ket ravishda topilishi mumkin bo'lgan formulalarda berilgan hisoblash jarayonidir. Bunday formulalarga rekurrent formulalar deyiladi. Rekurrent formulalar asosan rekurrent sonli ketma-ketliklarni hisoblashda ishlatiladi.

Matematikada shunga mos rekurrent tushunchasi kiritilgan, ya'ni sonli ketma-ketlik $x_0, x_1, \ldots$ rekurrent deyiladi, agar u quyidagi formula ko'rinishda bo'lsa

$$x_k = f(k, x_{k-1}, x_{k-2}, \ldots, x_{k-p}), k = p, p+1, \ldots$$
$$x_0 = a_0, x_1 = a_1, \ldots, x_{p-1} = a_{p-1}$$

Xususiy holda, p=1 bo'lsa

$$x_k = f(k, x_{k-1}), k = 1, 2, \ldots$$
$$x_0 = a$$

p=2 bo'lsa

$$x_k = f(k, x_{k-1}, x_{k-2}), k = 2, 3, \ldots$$
$$x_0 = a_0, x_1 = a_1$$

rekurrent formulalarga ega bo'lamiz.

E'tibor bering, boshlang'ich qiymatlar rekurrent formula bilan bog'lanmagan. Lekin yuqoridagi keltirilgan misollarda aytilgan cheksiz girdobga duch kelmaslik uchun, umumiy holda rekursiya uchun quyidagi shartlar bajarilishi kerak:

1) to'plam elementlari tartiblangan bo'lishi kerak;
2) har qanday kamayib boruvchi ketma-ketlik minimal element bilan tugallanishi kerak;
3) minimal element rekursiyasiz aniqlanadi;
4) minimal bo'lmagan har qanday element o'zidan katta bo'lgan boshqa elementlar orqali aniqlanadi.

Rekurrent formula bilan berilgan qiymatlarni dasturlash tillarida bevosita sikl orqali hisoblash mumkin. Lekin ularni rekursiya orqali kiritish juda ixcham va tushunarli dasturlarga olib keladi. Shu bois yuqorida keltirilgan faktorial uchun rekursiv funksiyani quyidagicha kiritish mumkin (bu yerda $n!=n*(n-1)!$ barcha $n>1$ va $1!=1$ deb qabul qilingan):

```
function f(n:integer):integer;
begin
     if n=1 then f:=1
     else f:=n*f(n-1)
end;
```

Ushbu funksiyani qo'llash uchun quyidagi asosiy dasturni keltiramiz:

```
program FactRek;
var
    n: integer;
function f(n:integer):integer;
```

```
begin
    if n=1 then f:=1
    else f:=n*f(n-1)
end;
begin
    readln(n);
    write(n, '! = ', f(n));
end.
```

Quyidagi jadvalda ushbu dasturning natijalari keltirilgan:

N	2	3	5	7	8
F(n)	2	6	120	5040	Error

E'tibor bering, *n* ning uncha katta bo'lmagan qiymatlarida dastur xatolikga olib keladi. Chunki biz o'zgaruvchilarni ***integer*** tipli deb kiritdik.

Rekurrent funksiya. Butun musbat son *n* uchun $f(n)$ funksiyasi quyidagicha ta'riflanadi:

$f(0)=0, f(1)=1, f(2n)=f(n), f(2n+1)=f(n)+f(n+1)$

Berilgan N qiymati uchun $f(N)$ qiymatini aniqlang va bu yerda massiv kiritish mumkin emas.

Masala yechimi. Ushbu masalaning ajoyibligi shundaki, bu yerda yangi funksiya kiritish orqali, masalani soddalashtirsh imkoni tug'iladi. Demak, bizni chalkashtiradigan narsa shuki, $f(n)$ har bir qadamda 2 ta funksiyaga ajralsa, uning uchun aniq formulani topish mushkul bo'ladi, lekin aslida bunday emas. Buni aniqlash uchun yangi $g()$ funksiyasini quyidagicha aniqlab kiritiamiz

$g(n,i,j) = if(n)+jf(n+1)$

Ushbu funksiyaning quyidagi xossalarini aniqlash mumkin

$g(2n,i,j)= g(n,i+j,j)$

$g(2n+1,i,j)= g(n,i,i+j)$

Haqiqatan ham

$g(2n,i,j)= if(2n)+jf(2n+1)= if(n)+j(f(n)+f(n+1))= (i+j)f(n)+jf(n+1)$

$g(2n+1,i,j)= if(2n+1)+jf(2n+1+1)= if(2n+1)+jf(2(n+1))=$

$i(f(n)+f(n+1)+jf(n+1)=if(n)+(i+j)f(n+1).$

Shundan kelib chiqqan holda $f(n) = g(n,1,0)$. Bu yerda $g(n,1,0)$ ni hisoblash jarayonida n argumenti kamayib boradi va oxir oqibat quyidagi natijaga kelamiz

$f(n) = g(n,1,0) = \ldots = g(0,i,j) = j$

Demak uchinchi argument j hisoblansa , biz o'z-o'zidan $f(n)$ ni hisoblagan bo'lamiz. Masalan, $n=5$ bo'lsa

$f(5) = g(5,1,0) = g(2*2+1,1,0) = g(2,1,1+0) = g(2,1,1) = g(2*1,1,1) = g(1,1+1,1) = g(1,2,1) = g(2*0+1,2,1) = g(0,2,2+1) = g(0,2,3)$

demak $f(5) = 3$.

Dastur esa quyidagicha bo'ladi:

```
program Rekurrent;
var n, i, j : integer;
begin
     write ('N =');
     readln(n);
     i:=1; j:=0;
while n>0 do
begin
     if (n mod 2) =0  then i:=i+j else j:=i+j;
     n:= n div 2;
end;
     writeln (j);
end.
```

Quyidagi jadvalda ushbu dasturning natijalari keltirilgan:

n	2	3	5	8	13	21
$f(n)$	1	2	3	1	5	8

Shundan kelib chiqqan holda, dasturlash tillarida rekursiya deb protsedura yoki funksiya o'z-o'ziga bevosita yoki bilvosita murojaat etilganda deyiladi.

Yana bitta misolni keltiramiz, 3-bobda keltirilgan Yevklid algoritmi ikki natural sonning eng katta umumiy bo'luvchisini aniqlashda qo'llanilgan edi, endi bu yerda rekursiyani qo'llaymiz.

EKUB. Ikki a va b natural sonlarining eng katta umumiy bo'luvchisini aniqlang.

Masala yechimi. *EKUB(a,b)* ni hisoblashda quyidagi tengliklarni asos qilib olamiz:

- agar *b=0* bo'lsa, u holda *EKUB(a, b)=a*;
- agar *a mod b=0* bo'lsa, u holda *EKUB(a, b)=b*;
- agar *a mod b>0* bo'lsa, u holda *EKUB(a, b)= EKUB(b, a mod b)*.

Natijada quyidagi rekursiv funksiyani keltirish mumkin:

```
function EKUB(a, b:integer):integer;
begin
    if b=0 then EKUB:=a else
    if a mod b=0 then EKUB:=b
    else EKUB:= EKUB (b, a mod b)
end;
```

Ushbu funksiyani qo'llash uchun quyidagi asosiy dasturni keltiramiz:

```
program EKUBRek;
var
    n,m: integer;
function EKUB(a, b:integer):integer;
begin
    if b=0 then EKUB:=a else
    if a mod b=0 then EKUB:=b
    else EKUB:= EKUB (b, a mod b)
end;
begin
    readln(n,m);
    write(n,m, ' larni EKUB = ', EKUB(n,m));
end.
```

Quyidagi jadvalda ushbu dasturning natijalari keltirilgan:

n	2	126	125	216	1453	2221
m	6	48	435	2313	2113	2551
EKUB	2	6	5	9	1	1

Rekursiya jarayonlarida quyidagi 2 ta tushuncha muhim hisoblanadi – bu rekursiyaning uzunligi va chaqiruvlarning umumiy soni. Misol sifatida *n!* rekursiyasini ko'rib chiqamiz. Masalan *f*(4) ni hisoblashda *f*(3) bajarilishi kerak, uning uchun esa *f*(2) ni aniqlash kerak va h.k. Bunday jarayon ichki deyiladi. Demak *f*(4) ni chaqiruvida yana

uchta ichki chaqiruv paydo bo'ladi, ya'ni umumiy holda $f(n)$ uchun rekursiyaning uzunligi $(n-1)$ bo'ladi. Bu yerda chaqiruvlarning ham umumiy soni $(n-1)$ bo'ladi.

Binom koeffitsiyentlarini $C(m,n)$ xossalaridan kelib chiqqan holda, ularni hisoblashni quyidagi rekursiv funksiya orqali amalga oshirish mumkin:

```
function C(m, n:integer):integer;
begin
    if (m<=1) or (n=0) or (n=m) then C:=1
    else C:=C(m-1, n-1)+C(m-1, n)
end;
```

Natijada har bir chaqiruv yana ikkita ichki chaqiruvni sodir qiladi. Albatta bunday vaqtlarda rekursiyadan foydalanish tavsiya etilmaydi.

Rekursiv yechimning iteratsiya usuli bilan farqini Fibonachchi sonlarini aniqlashdagi misolda ko'rib chiqamiz. Fibonachchi sonlari quyidagi formula bo'yicha hisoblanadi:

$$F_n=F_{n-1}+F_{n-2},\ n>1 \text{ bunda } F_0=1 \text{ va } F_1=1$$

Shundan kelib chiqqan holda, quyidagi dasturni keltiramiz:

```
program Fibonacci;
var n : integer;
{ Fibonachchi usuli }
function F(n: integer): longint;
begin
    if (n=0) or (n=1) then
    F:=1
    else F:=F(n-1)+F(n-2); {- bu rekursiya }
end;
{ Iteratsiya usuli }
function Iter(n: integer): longint;
var x,y,t: longint; k: integer;
begin
    x:=1; y:=1;
    for k:=2 to n do
```

```
begin
    t:=y; y:=x+y; x:=t;
end;
    Iter:=y;
end;
{Asosiy dastur}
begin
    write('n='); readln(n);
    writeln('Fibonachchi usuli:F(', n, ')= ', F(n));
    writeln;
    writeln('Iteratsiya  usuli:F(', n, ')= ',Iter(n));
end.
```

Quyidagi jadvalda ushbu dasturning natijalari keltirilgan:

n	0	1	2	3	4	5	6	7
F(n)	1	1	2	3	5	8	13	21

E'tibor bering, rekursiv funksiya orqali yozilgan algoritmni bir ko'rishda tushunib olsa bo'ladi. Iteratsiya usulida hisoblanganda esa dastur uchun ko'p xotira talab etilmaydi va u nisbatan tezkor bajariladi.

Rekursiv chaqiruv bilvosita ham bo'lishi mumkin. Bu yerda "*A*" qism dastur "*B*" qism dasturni chaqiradi va teskari. Ushbu ko'rinishni amalga oshirish uchun qism dastur quyidagicha ta'riflanishi lozim

```
Procedure B(j: Byte); forward;
    Procedure A(i: Byte);
    Begin
        ...
        B(i);
        ...
    end;
procedure B;
begin
    ...
    A(j);
```

...
end;

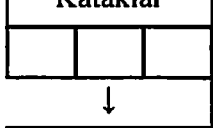
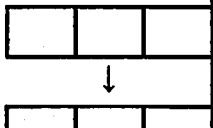
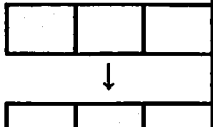
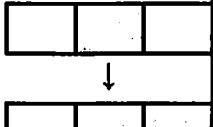
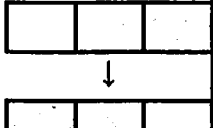
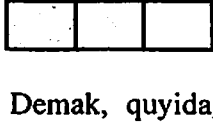
E'tibor bering, "B" qism dasturida parametrlar ko'rsatilmagan.

Kataklarni egallash. Bo'sh kataklar ketma-ket joylashtirilgan va ularning umumiy soni n ta. Agar i-katak bo'sh bo'lsa, uni egallash mumkin (bunda $+i$ yoziladi) yoki uni bo'shatish mumkin (bunda $-i$ yoziladi). Bu erda kataklarni holatini o'zgartirish quyidagi qoidalar asosida amalga oshiriladi:

1. agar katak birinchi bo'lsa;
2. agar egallangan katak boshqa egallangan kataklar ichida eng kichik nomerli bo'lsa, undan keyingi katakni egallash mumkin.

Kataklarni egallash tartibini aniqlang.

Masala yechimi. Masalan, n=3 bo'lganda, kataklarni egallash tartibi quyidagicha bo'ladi: +1+2-1+3+1. Haqiqatan

Kataklar	Jarayon	Izoh
	+1	1-qoidaga asoslanib
	+2	2-qoidaga asoslanib
	-1	1-qoidaga asoslanib
	+3	2-qoidaga asoslanib
	+1	1-qoidaga asoslanib
	natija	Barcha kataklar egallanildi

Demak, quyidagi funksiyalarni kiritamiz, dasturda ularni qism dastur orqali amalga oshiramiz:

1) *fillOnly(n)* – faqatgina n- katakni egallash;
2) *fill(n)* – barcha n ta katakni egallash;
3) *free(n)* – barcha n ta katakni bo'shatish;

Bu yerda *fill*(*n*) funksiyasi quyidagicha aniqlanadi. Agar $n=1$ bo'lsa, u holda natija +1 bo'ladi, $n=2$ bo'lsa, natija +1+2 bo'ladi. Umumiy holda, ya'ni $n\geq 3$ bo'lganda, *n*-katakni egallash uchun (n-1)-katak egallangan va qolgan (n-2) ta katak bo'sh bo'lishi kerak, bu esa *fillOnly*() qismida amalga oshirilgan bo'ladi, shu bois bu erda (n-2) ta katakni bo'shatib o'tirmaymiz. Shundan keyin qolgan (n-2) ta katakni egallash mumkin.

Yuqorida keltirilgan funksiyalar orqali *fill*(*n*) funksiyasida quyidagilar bo'lishi kerak:

```
fillOnly(n-1);
write('+',n);
fill(n-2);
```

Endi *fillOnly*(*n*) funksiyasini tahlil qilamiz. Agar $n=1$ bo'lsa, u holda +1 chiqarilishi lozim, aksincha $n\geq 2$ bo'lsa, oldiniga (n-1)-katakni egallab olishimiz kerak, keyin esa *n*- katakni egallaymiz. Shundan keyin (n-1) ta katakni bo'shatamiz. Demak, *fillOnly*(*n*) funksiyasi quyidagicha bo'lishi mumkin:

```
fillOnly(n-1);
write('+',n);
free (n-1);
```

Endi *free*(*n*) funksiyasini tahlil qilamiz. Agar $n=1$ bo'lsa, natijaga -1 ni chiqarishimiz kerak, $n\geq 2$ bo'lganda esa, oldiniga *fillOnly*(n-1) ni bajarishimiz kerak, keyin *n*-katakni bo'shatamiz, undan keyin (n-1) ta katakni bo'shatib qo'yamiz. Umumiy holda, *free(n)*, $n\geq 2$ bo'lganda quyidagicha bo'ladi:

```
fillOnly(n-1);
write('-',n);
free(n-1);
```

Shunday qilib, *fill*(), *fillOnly*(), *free*() funksiyalari deyarli yozildi. Faqat bu yerda *fillOnly* funksiyasi *free* funksiyasiga murojaat qilyapti, *free* o'z navbatida *fillOnly* funksiyasiga. Yuqorida keltirilgan ushbu funksiyalarni yozish tartibiga amal qilgan holda, quyidagi dasturni tavsiya qilamiz:

```
program Cells;
var n :integer;
procedure free(n : integer) ; forward;
{ fillOnly(n) – faqatgina n- katakni egallash}
procedure fillOnly(n : integer);
```

```
begin
      if n = 1 then write('+1')
      else
begin
      fillOnly(n-1);
      write('+', n); free(n-1)
end
end;
{free(n) – faqatgina n- katakni bo'shatish}
procedure free(n : integer);
begin
  if n = 1 then write('-1')
      else
begin
  fillOnly(n-1);
      write ('-' , n) ; free(n-1)
end
end;
{ fill(n) – barcha n ta  katakni egallash}
procedure fill(n : integer);
begin
      if n = 1 then write('+1')
      else
      if n = 2 then write('+1+2')
      else
begin
      fillOnly(n-1);
      write('+', n);
      fill(n-2)
end
end;
{Asosiy dastur}
begin
      readln(n);
      fill(n);
end.
```

Dasturning n=4 bo'lgandagi quyidagi natijasini +1+2-1+3+1-2-1+4+1+2 bevosita qo'lda tekshirib ko'rishni tavsiya qilamiz.

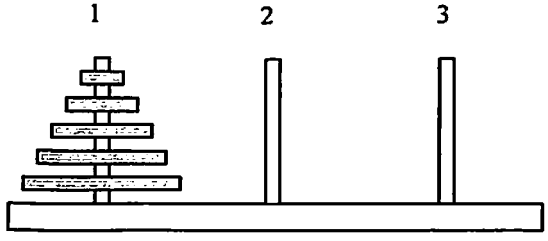
6.2. Xanoy minorasi masalasi haqida

Benares ibodatxonasida olmosli uchta ustun bronza plitalarida joylashgan. Dunyoni yaratish chog'ida xudo ustunga 64 ta oltin diskni joylashtirgan bo'lib, unda eng katta disk bronzali plitada turibdi, qolganlari esa diametri kichrayib borish tartibida joylashib, piramidani tashkil qilgan. Bu Bram minorasidir. Maxsus xizmatkorlar kechayu-kunduz ishlab disklarni bir ustundan ikkinchisiga quyidagi Bram qoidasi bo'yicha olib o'tadilar:

- disklar bir ustundan ikkinchisiga faqat bittalab olib o'tiladi;
- katta diskni kichigining ustiga qo'yish mumkin emas.

Qachonki barcha 64 ta disk bitta ustundan ikkinchisiga xuddi shu holatda o'tkazilsa, o'tkazilgan kuni minora ham, xizmatchilar ham erib ketadi va "oxiri zamon" , ya'ni qiyomat kuni boshlanadi.

Bu qadimiy afsonadan Xanoy minorasi kelib chiqdi: "Bram qonuni" ga ko'ra uchta ustunlardan biridagi n ta disklarni boshqasiga joylashtirish talab etiladi.



Tabiiy tildagi algoritm quyidagi ko'rinishga ega bo'ladi:

1. Agar $m=1$ ga teng bo'lsa, bitta disk o'tkaziladi, jarayon to'xtatiladi.
2. Disklarni o'tkazishda yuqoridagi n-1 ta diskni "1"-ustundan "2"-ustunga , "3"-ustundan foydalangan holda o'tkaziladi.
3. "1"-ustunda qolgan diskni "3"-ustunga olib o'tish.
4. Disklarni o'tkazishda n-1 ta diskni "2"-ustundan "3"-ustunga , "1"-ustundan foydalangan holda o'tkaziladi.

```
Program HanoyRek;
var k:word;
procedure Hanoy(n:integer; one, two, three:char);
begin
    if n=1 then write (one, '-->', two, ' ')
    else
```

```
begin
      Hanoy(n-1,one, three, two);
      Hanoy(1,one, two ,three);
      Hanoy(n-1, three , two, one);
end
end;
{Asosiy dastur}
begin
      readln( k );
      Hanoy(k, '1', '2', '3');
end.
```

n=4 bo'lganda quyidagi natija olinadi:

```
1-->3  1-->2  3-->2  1-->3  2-->1  2-->3 1-->3  1-->2  3-->2  3-->1  2--
>1  3-->2
1-->3  1-->2  3-->2
```

Bu yerda, n=64 bilan tajriba o'tkazishni tavsiya bermaymiz, chunki bu juda ko'p vaqt talab etadi.

Ushbu algoritm bo'yicha jami bo'lib necha marta almatirish zarur bo'lishini aniqlash maqsadida $f(n)$ sonli funksiyani kiritamiz. Ya'ni $f(n)$ bu n ta diskni ko'chirish uchun kerak bo'lgan almashtirishlar soni. Rekursiya algoritmiga binoan $f(n)$ quyidagicha aniqlanishi mumkin:

$f(1)=1$

$f(n+1)=f(n)+f(1)+f(n)$

ushbu rekurrent formulasi uchun $f(n)=2^n-1$ yechim bo'ladi. Haqiqatan, n=1 bo'lganda ushbu formula to'g'ri va n uchun ham to'g'ri deb hisoblasak, u holda

$$f(n+1)= 2f(n)+f(1)=2(2^n-1)+1=2^{n+1}-1$$

Demak, n=64 bo'lsa $f(64)= 2^{64}-1\approx(2^{10})^6\ 10 \approx (10^3)^6\ 10 = 10^{19}$ Albatta buncha o'rin almashtirishda kompyuterni qancha vaqti ketishini tasavvur qilish qiyin.

Daraja. Berilgan a sonining k-darajasini ya'ni a^k ni hisoblash talab qilinadi. Bunda dasturlash tilida qabul qilingan daraja funksiyasini qo'llash mumkin emas va k soni juda katta bo'lganligi sababli k marta ko'paytirishni ham qo'llash mumkin emas.

Masala yechimi. 3-bobda biz ushbu misolni ko'rib chiqqanmiz. Rekursiyani amalga oshirish uchun quyidagilardan foydalanamiz:

- agar $n=1$ bo'lsa, u holda $x^n=x$;
- agar $n>1$ bo'lsa, u holda $x^n = x^{n \bmod 2}(x^{n \operatorname{div} 2})^2$.

Natijada quyidagi funksiyani keltiramiz:

```
function pow(x,n:integer):integer;
    var t:integer;
begin
    if odd(n) then t:=x else t:=1;
    if n=1 then pow:=x else pow:=t*sqr(pow(x,n div 2))
end;
```

Ushbu funksiyani qo'llash uchun quyidagi asosiy dasturni keltiramiz:

```
program DarajaRek;
var a,n: integer;
function pow(x,n:integer):integer;
var t:integer;
begin
      if odd(n) then t:=x else t:=1;
      if n=1 then pow:=x else pow:=t*sqr(pow(x,n div 2))
end;
{Asosiy dastur}
begin
      readln(a,n);
      write(' a darajasi n  = ',pow(a,n));
end.
```

Ushbu dasturni tekshirishda, uncha katta bo'lmagan sonlar uchun ham, masalan 3^{10} qiymati uchun, natija *integer* chegarasidan o'tib ketishini inobatga oling.

6.3. Rekursiyaga oid murakkab misollar

6.3.1. Tekislikdagi Kantor to'plami

Rekursiv jarayonlarni geometrik o'xshash ob'ektlarda bajarish juda samarali hisoblanadi. Shulardan biri tekislikdagi Kantor to'plami hisoblanadi. Ushbu to'plam quyidagi algoritm asosida shakllantiriladi.

1) Bo'yalgan kvadrat chiziladi;
2) Uni 16 ta kvadratlarga bo'lamiz;
3) O'rtada joylashgan 4 ta kvadrat olib tashlanadi, lekin chegaradagi rang saqlanadi;
4) Qolgan kvadratlar bilan ham ushbu jarayon takrorlanadi.

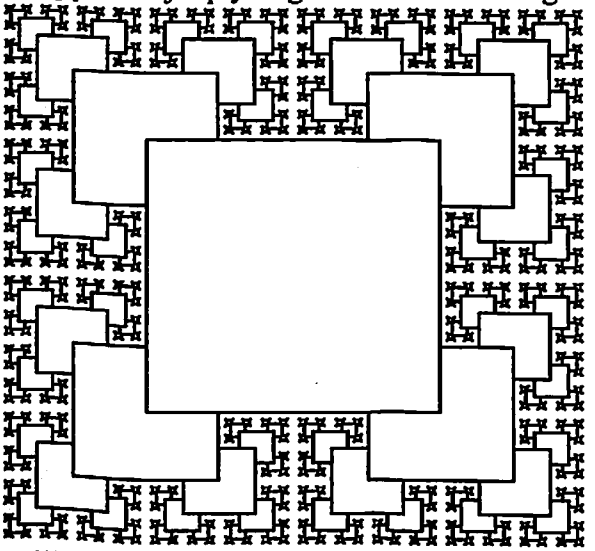
Quyida keltirilgan Kantor dasturida katta kvadratdan kichik kvadratgacha yetib boriladi, so'ng kichigidan kattasigacha chiziqlar chizila boshlanadi.

Dasturning umumiy ko'rinishi quyidagicha bo'ladi:

```
{$S+}
program Kantor;
uses Crt, Graph;
const min_size=1;
var ch: Char;
  Gd,gm: integer;
procedure draw(x,y: integer; size: Word);
      var s: Word;
      var ch: Char;
begin
      if size>min_size then
begin
      s:=size div 2;
      draw(x-size, y+size, s);
      draw(x-size, y-size, s);
      draw(x+size, y+size, s);
      draw(x+size, y-size, s);
end;
      rectangle(x-size, y-size, x+size, y+size);
      bar(x-size+1, y-size+1, x+size-1, y+size-1);
end;
{Asosiy dastur}
begin
```

```
    Gd:=detect;
    InitGraph(gd,gm, 'd:\turbo pascal 7.1\bgi');
    If GraphResult <> grOk then exit;
    SetFillStyle(solidfill, White);
    SetColor(black);
    draw(GetMaxX div 2, GetMaxY div 2, GetMaxY div 4);
    ch:=ReadKey;
{-istalgan tugma bosiladi}
    Closegraph;
end.
```

Ekranda chiqqan natija quyidagi chizmada keltirilgan.



Shunday qilib, rekursiv dasturlarni tuzishda bevosita masalaning qo'yilishidan o'z-o'zidan ayon bo'lgan algoritmlar uchun qo'llash tavsiya etiladi. Bu o'z navbatida dasturni oddiy ko'rinishga olib keladi, lekin xotirani tejashda va tezkorlikni ta'minlashda ehtiyot bo'lish kerak.

6.3.2. Arifmetik ifodaning qiymati.

Arifmetik ifodalar bilan ishlashda biz rekursiya orqali uning qiymatini hisoblashni o'rganib chiqamiz. Bu yerda biz faqatgina qo'shish va ko'paytirish amallaridan va natural sonlardan tashkil topgan quyidagi ko'rinishdagi arifmetik ifodalarni ko'rib chiqamiz, masalan

*(1+2*3)*(4+5)+6*(7+8)+9*

Bunda bizga quyidagi o'zgarmas qiymatlar

```
const
_Plus='+'; {qo'shish belgisi}
_Mul='*'; {ko'paytirish belgisi}
_Open = '('; {ochilgan qavs}
_Close=')'; {yopilgan qavs}
_End=#10; {ifoda tugadi}
```

va quyidagi o'zgaruvchilar kerak bo'ladi

```
var curlex : char; {ifoda belgisi}
    vl : Longint; {qiymat}
  v: longint;{natija}
```

Arifmetik ifodaning elementlarini birin-ketin o'qib olish uchun *nextLexem* qism dasturini yaratamiz.

```
Procedure nextLexem;
begin
    if pos<length(s) then
 begin
      inc(pos);
    curlex:=s[pos];
end
    else
    curlex:=_End; { satr tugadi}
end;
```

Demak, *curlex* o'zgaruvchisini qiymatlash *nextLexem* dasturida amalga oshiriladi. Unda joriy belgi saqlanadi va har bir *nextLexem* chaqirilganda uning qiymati keyingi ifoda belgisiga almashtiriladi.

Endi arifmetik ifodani aniq ta'riflab olamiz. Yuqoridagi misolni ko'rib chiqamiz:

*(1+2*3)*(4+5)+6*(7+8)+9*

Ushbu ifodani hisoblashda, maktabda o'rganganimiz bo'yicha biz oldiniga barcha qavslardagi ifodalarni hisoblab, quyidagini tashkil qilamiz

63 + 90 + 9

va keyin ushbu oddiy ifodani hisoblab natijani hosil qilamiz.

Shundan kelib chiqqan holda arifmetik ifodani quyidagicha ta'riflash mumkin:

<Ifoda>::=<qo'shiluvchi>{+< qo'shiluvchi >}

Bu yerda figurali qavs bo'lmasligi yoki bir necha bor takrorlanishi mumkinligini anglatadi, masalan ifoda 2*3 bitta qo'shiluvchidan iborat.

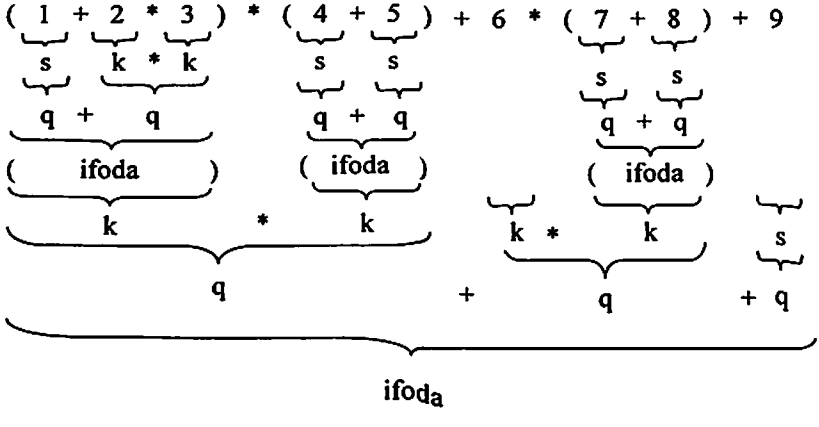
Ushbu misoldan qo‘shiluvchini qanday ta’riflash lozimligi ko‘rinib turibdi, ya’ni

< qo‘shiluvchi >::=<ko‘paytuvchi>{*< ko‘paytuvchi >}

Ko‘paytuvchi esa quyidagi ko‘rinishda bo‘lishi mumkin: 6*(7+8), ya’ni u yoki son yoki yana ifoda bo‘lishi mumkin, demak uni quyidagicha ta’riflash mumkin:

< ko‘paytuvchi >::=<son> | <ifoda >

Ushbu ta’riflashda biz yana ifodaga qaytib keldik, ya’ni bu yerda rekursiya yuzaga keldi, ya’ni ifodani biz yana ifoda orqali ta’rifladik. Quyidagi chizmada bu ifodani to‘liq tahlilini keltiramiz



Bu yerda quyidagi qisqartmalar keltirilgan: *s* – son, *k* – ko‘paytuvchi, *q* – qo‘shiluvchi.

Endi zarur bo‘lgan funksiyalarni ta’riflab, ularning dasturlarini tuzamiz.

Function expr : longint;

Ushbu funksiya bevosita *curlex* o‘zgaruvchi qiymatni qabul qilgandan so‘ng chaqiriladi va ifodaning qiymatini hisoblab, *expr* funksiyasiga qaytaradi, *curlex* o‘zgaruvchisi esa, o‘z navbatida, ifodadan keyingi belgini o‘zlashtirib oladi. Yuqoridagi misolni ko‘rib chiqamiz: *(1+2*3)*(4+5)+6*(7+8)+9*.

Agar *curlex* o‘zgaruvchida birinchi ochiladigan qavs bo‘lsa, u holda ifoda to‘liq hisoblanib, *expr* = 162 bo‘ladi va *curlex* = *_End* bo‘ladi. Agar keyingi belgi bir bo‘lsa, ya’ni *curlex* son bo‘lsa, unda va *vl*=1 va u holda *expr* qiymati 1+2*3=7 bo‘ladi. Bundan so‘ng *curlex* = ‘)’ bo‘ladi.

Keyingi funksiyamiz bu -

Function item : longint;

Ushbu funksiya bevosita *expr* funksiyasiga o'xshash bo'lib, ifodjaning tarkibidan qo'shiluvchining qiymatini hisoblaydi.

Oxirgi funksiyamiz bu -

Function mult : longint;

Ushbu funksiya bevosita ifoda tarkibidagi ko'paytuvchini hisoblaydi.

Endi ushbu funksiyalarni to'liq dasturini keltiramiz

```
Function expr : longint;
var a: longint;
{bu yerda hisoblash jarayonidagi natijani saqlaymiz}
begin
{ifodaning ta'rifidan, u doimo qo'shiluvchidan boshlanadi, {demak uni item funksiyasi orqali hisoblaymiz}
     a:= item;
{shunday qilib,ifodaning qiymati "a" o'zgaruvchida} {saqlanadi. Ushbu infodan keyin keladigan belgi curlex} {o'zgaruvchida saqlanadi. Agar uning qiymati "+" yoki "-"} {bo'lsa, demak undan keyin yana qo'shiluvchi keladi va h.k.} {Agarda "+","-" bo'lmasa, demak ifoda tugamagan:}
while (curlex=_Plus) or(curlex=_Min ) do
begin
     case curlex of
     _Plus:
{ya'ni joriy belgi plyus bo'lsa, demak biz uni xotirada} {saqlaymiz va keyingi belgiga o'tamiz}
begin
     nextlexem;
{endi curlex bu yerda yangi qo'shiluvchining birinchi} {belgisi bo'ldi, demak ushbu qo'shiluvchining qiymatini} {"a" ga qo'shib qo'yamiz}
     a:=a+item
end;
     _Min:
begin
     nextlexem;
  a:=a-item;
end;
end;
end;
```

```
    expr:=a;
end;
```

Endi *item* funksiyasini dasturini tuzamiz. Uning ta'riflangishi bevosita "ifoda"ning ta'riflanishiga o'xshash bo'lganligi uchun ularning dasturlari ham o'xshash bo'ladi:

```
Function item: longint;
var a: longint;
begin
    a:= mult;
while (curlex = _Mul) do
    case curlex of
    _Mul:
begin
    nextlexem;
    a:=a * mult;
end;
end;
    item:=a;
end;
```

Bevosita *mult* funksiyasini ham uning ta'riflanishiga binoan tuzamiz:

```
function mult: longint;
begin
{o'qilgan belgi – bu son , demak ko'paytuvchi bevosita vl}
{o'zgaruvchining qiymatiga teng bo'ladi}
    vl:=0;
while (ord(curlex)>=ord('0'))and(ord(curlex)<=ord('9')) do
begin
    vl:=vl*10+ ord(curlex) - ord('0');
nextlexem; {keyingi belgiga o'tish zarur:}
    mult:=vl;
 end;
    if curlex = _Open then
    {ya'ni joriy belgi – bu ochiladigan qavs}
begin
    nextlexem;
```

```
{endi bu ifodaning birinchi belgisida turibmiz va uning} {qiymati
bevosita ko'paytuvchining qiymati bo'ladi}
      mult:=expr;
{endi biz ifodaning keyingi  belgisida joylashdik.}
{Ushbu belgi yopiladigan qavs bo'lishi kerak}
      if curlex=_Close then nextlexem
else error;
{-ya'ni xato mavjud, ifoda qavs bilan yopilmagan}
end;
end;
```

Shunday qilib, asosiy dasturimiz quyidagi ko'rinishda bo'ladi:

```
program Ifoda;
const
      _Plus='+'; {qo'shish belgisi}
      _Min = '-'; { ayirish belgisi}
      _Mul='*'; {ko'paytirish belgisi}
      _Open = '('; {ochilgan qavs}
      _Close=')'; {yopilgan qavs}
      _End=#10; {ifoda tugadi}
var
      curlex: char ; {ifoda belgisi}
      vl: longint; {qiymat}
      v: longint;{naija}
      pos: longint;{belgi o'rni}
      s: string;
      fayl : text;
function expr  :longint; forward;
Procedure error;
begin
      writeln('Ifodada xatolik mavjud');readln;
      halt(1);
end;
{keyingi belgini o'qib olish}
Procedure nextLexem;
begin
      if pos<length(s) then
 begin
      inc(pos);
      curlex:=s[pos];
```

```
end
      else
      curlex:=_End; { qator tugadi}
end;
function mult: longint;
begin
      vl:=0;
while (ord(curlex)>=ord('0'))and(ord(curlex)<=ord('9')) do
begin
      vl:=vl*10+ ord(curlex) - ord('0');
      nextlexem;
      mult:=vl;
 end;
      if curlex = _Open then
 begin
      nextlexem;
      mult:=expr;
      if curlex=_Close then nextlexem
      else error;
 end;
end;
Function item: longint;
var a: longint;
begin
      a:= mult;
while (curlex = _Mul) do
      case curlex of
      _Mul:
begin
      nextlexem;
      a:=a * mult;
end;
end;
      item:=a;
end;
Function expr : longint;
var a: longint;
begin
      a:= item;
while (curlex=_Plus) or(curlex=_Min ) do
```

```
begin
      case curlex of
      _Plus:
begin
      nextlexem;
      a:=a+item
end;
      _Min:
begin
      nextlexem;
      a:=a-item;
end;
end;
end;
      expr:=a;
end;
      {Asosiy dastur}
begin
      pos:=0;
{ Qatorli ifodani kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl,s);
      close(fayl);
      nextlexem;
{endi curlex o'zgaruvchisida ifodaning birinchi belgisini}
{joylashtirdik, endi ifodani hisoblaymiz}
      v:=expr;
      if (curlex<>_End) then error;
      writeln('Natija=', v);
end.
```

Quyidagi jadvalda ushbu dasturning natijalari keltirilgan:

Ifoda	Natija
(1+2*3)*(4+5)+6*(7+8)+9	162
(11+2+3)*(4+5)+6+(7+8)*9	285
(1+2*3)*4+5+6+(7+8)+9	63
(10+21*3)*5+(5+6)*(7+8+9)	629
(10+21*3)*5-(5+6)*(7-8+9)	277

Ushbu funksiyalar bir-biriga bilvosita murojaat qilganliklari sababli, biz bu yerda rekursiyali funksiyalarni yaratdik. Bunday hollarda **forward** operatori funksiyalar ta'riflanishida qo'llaniladi va ularni quyidagi *mult*, *item*, *expr* tartibda berilganiga e'tibor bering.

Topshiriqlar

1. Berilgan natural N sonining raqamlar yig'indisini hisoblang.
2. Berilgan natural N sonining raqamlarini chop eting.
3. Massiv $a[n]$ elementlarining yig'indisini hisoblang.
4. Quyidagi rekurrent formulada x_n hisoblang

$$x_k = b_0 x_{k-1} + b_1, k = 1,2,\ldots; x_0 = a$$

5. Quyidagi rekurrent formulada x_n hisoblang

$$x_k = qx_{k-1} + rx_{k-2} + b, k = 2,3,\ldots$$
$$x_0 = c, x_1 = d$$

6. Quyidagi Fibonachchi soninini hisoblash formulasida biror h sonidan birinchi katta bo'lgan Fibonachchi sonini aniqlang

$$x_k = x_{k-1} + x_{k-2}, k = 3,4,\ldots$$
$$x_1 = x_2 = 1$$

7. Quyidagi qatorlar ketma-ketligini tuzuvchi rekursiv qism dasturini tuzing. Hammasi bo'lib 26 qator.

A
BB
CCC
DDDD
.....

8. Quyidagi qatorlar ketma-ketligini tuzuvchi rekursiv qism dasturini tuzing. Hammasi bo'lib 9 qator.

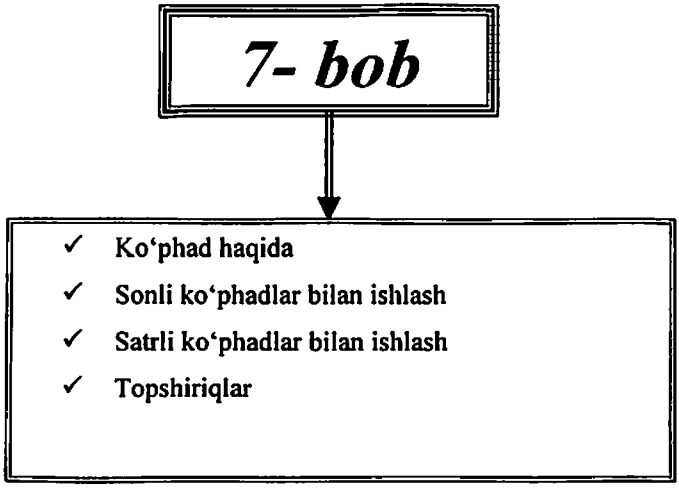
1
22
333
4444
.....

9. Bir o'lchovli massiv elementlarini teskari tartibda yozuvchi rekursiv dastur tuzing
10. Berilgan so'zni teskari tartibda yozuvchi rekursiv dastur tuzing.
11. Quyidagi ko'rinishdagi arifmetik ifodalarni qiymatini hisoblash *(1+12/3)*(4+5)+6*(7+8)-9* dasturini tuzing.

7-bob. KO'PHADLAR USTIDA AMALLAR BAJARISHGA DOIR DASTURLAR TUZISH

Algebra kursidan biz ko'phadlar ustida bajariladigan arifmetik amallarning o'ziga xosligini bilib olganmiz. Bunda bajarilayotgan amallar soni ko'p bo'lishi mumkinligi sababli ular uchun tezkor algoritmlarni tanlash tavsiya etiladi.

Mazkur bobda ko'phadlarga doir asosan misollar keltirilgan bo'lib, nazariy jihatga kam e'tibor berilgan. Ushbu misollar ko'phadlar haqida to'liq tasavvurga ega bo'lishga imkon beradi:



7.1. Ko'phad haqida

Matematikada keng qo'llaniladigan ko'phadlar ichidan quyidagisi juda muhim hisoblanadi:

$$P(x) = a_0 + a_1 x + \ldots + a_n x^n$$

Bu yerda x – o'zgaruvchi, a_i – koeffitsiyentlar va $a_n \neq 0$. Ushbu ko'rinishdagi ko'phadni dasturlash tillarida $(n+1)$ uzunlikdagi massiv orqali tasvirlash keng tarqalgan. Bunday tasvirlash ko'phadlar ustida bajariladigan arifmetik amallarni yengilgina amalga oshirish imkonini beradi. Ko'phadlar ustida amallar bajarilayotganda tezkor algoritmlarni tanlash tavsiya etiladi.

Ko'phadlar ustida quyidagi amallarni bajarish mumkin:

- Hisoblash. Bu yerda berilgan koeffitsiyentlar va x qiymati uchun $P(x)$ ni hisoblash talab etiladi. Agarda chuqur fikrlamasdan, har safar x^i ni hisoblasak, unda algoritmning samaradorlik darajasi $O(n^2)$ tartibi bo'ladi. Vaxolanki, agar biz bu yerda $x^{i+1} = x^i \cdot x$ formulasini qo'llasak, samaradorlik darajasi $O(n)$ tartibi bo'ladli. Albatta bu yerda Gorner usulidan foydalansak, dasturimiz juda ham ixcham bo'ladi.
- Qo'shish va ayirish. Bu amallar hech qanday qiyinchilik tug'dirmaydi. Ushbu amallarni bajarish juda sodda yo'l bilan amalga oshirish mumkin, bu yerda mos hadlar oldidagi koeffitsiyentlar qo'shiladi yoki ayiriladi.
- Ko'paytirish. Bu yerda ikki $P(x)$ va $Q(x)$ ko'phadlarini o'zaro ko'paytirish nazarda tutiladi. Buni bevosita qavslarni ochish bilan amalga oshirish mumkin va unda hosil bo'ladigan yangi ko'phadning koeffitsiyentlari maxsus formula orqali aniqlanadi.
- Bo'lish. Bu yerda $P(x)$ ko'phadini $Q(x)$ ko'phadiga bo'lish kerak bo'ladi. Umumiy holda natija ko'phad emas, shu bois bu yerda faqatgina $P(x) = Q(x) \cdot H(x) + R(x)$ formuladagi $H(x)$ va $R(x)$ ko'phadlarini aniqlash talab etilishi mumkin.

7.2. Sonli ko'phadlar bilan ishlash

Ko'phad qiymati. Berilgan ko'phadni $P(x) = p_0 + p_1 x + \ldots + p_n x^n$ Gorner yondashuvi orqali x_0 nuqtasidagi qiymatini aniqlang.

Masala yechimi. Bu yerda bevosita Gorner formulasi bo'yicha hisoblash ta'kidlab o'tilgan, ya'ni quyidagi formuladan foydalanamiz:

$P(x_0) = (..(p_n x_0 + p_{n-1})x_0 + ... + p_1)x_0 + p_0$

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
2 2 1 1 1	7

Bu yerda *input.txt* faylida birinchi qatorda ko'phadning darajasi *n* va hisoblash nuqtasi x_0, ikkinchi qatorda esa ko'phadning koeffitsiyentlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida natijaviy qiymat chiqarilgan bo'ladi.

Quyida keltirilgan dasturni tushunib olish qiyinchilik tug'dirmaydi.

```
program GornerPolynom;
const Nmax=100;
var
      n, result, i, x0 : integer;
      P:array[0..Nmax] of integer;
      fayl : text;
begin
{polinom darajasini va boshqa qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n, x0);
      for i:=0 to n do
      read(fayl, p[i]);
{sikl orqali p[] qiymatlarini kiritamiz}
      close(fayl);
      result:=0; i:=n;
while  (i>=0) do
begin
      result := result*x0+p[i];
      i := i-1;
end;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      write (fayl, result);
      close(fayl);
end.
```

Ko'phad koeffitsiyentlari. Ko'phadning umumiy ko'rinishi quyidagicha berilgan:

$$P_n(x)=a_0+a_1x+a_2x^2+...+a_{n-1}x^{n-1}+x^n$$

Uning ildizlari $x_1,x_2,...,x_n$ butun sonlarda berilgan bo'lsa, barcha a_i koeffitsiyentlarini aniqlang.

Masala yechimi. Bezu teoremasi bo'yicha

$$P_n(x)=(x-x_1)\ (x-x_2)...\ (x-x_n)$$

To'g'ridan-to'g'ri qavslarni ochib a_i larni aniqlash formulasidan foydalanish mumkin, ammo quyidagilarni e'tiborga olsak, koeffitsiyentlar uchun rekurrent formulani aniqlash mumkin bo'ladi. Yuqorida keltirilgan formula bo'yicha

$$P_n(x)= P_{n-1}(x)\ (x-x_n)$$

Qavsni ochib chap va o'ng tomondagi koeffitsiyentlarni tenglashtiramiz

$$a_{0,n}+a_{1,n}x+a_{2,n}x^2+...+a_{n-1,n}x^{n-1}+x^n =a_{0,n-1}+a_{1,n-1}x+a_{2,n-1}x^2+...+a_{n-1,n-1}x^{n-1}+x^n-$$

$$-a_{0,n-1}\,x_n -a_{1,n-1}x\,x_n -a_{2,n-1}x^2\,x_n -...-a_{n-2,n-1}x^{n-2}\,x_n -x^{n-1}x_n=$$
$$=-a_{0,n-1}\,x_n +(\,a_{0,n-1} - a_{1,n-1}x_n)x+ ...+(\,a_{n-3,n-1}-a_{n-2,n-1}x_n\,)x^{n-2} +(\,a_{n-2,n-1}-x_n\,)x^{n-1}+x^n$$

Demak, umumiy holda har qanday m uchun

$a_{m,m-1}= a_{m-1,m-2}-x_m$

$a_{m,i}= a_{m-1,i-1}- a_{m-1,i}x_m \quad , 1\le i \le m-2$

$a_{m,0}=-a_{m-1,0}x_m$

Ya'ni, koeffitsiyentlarni aniqlash uchun rekurrent formula aniqlandi va shu bo'yicha ketma-ket barcha koeffitsiyentlar aniqlanadi. Boshida n=1 bo'lganda

$$P_1(x)= (x-x_1)$$

bo'ladi, demak ko'phad koeffitsiyentlari bu yerda $A[0]$=-$X[1]$ va $A[1]$=1 bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
2 1 1	1 -2

Bu yerda *input.txt* faylida birinchi qatorda ko'phadning darajasi *n*, ikkinchi qatorda esa uning ildizlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida ko'phad koeffitsiyentlari chiqarilgan bo'ladi.

Quyida keltirilgan dasturni tushunib olish endi qiyinchilik tug'dirmaydi.

```
program CoeffPolynom;
const Nmax=100;
var
      i,m,n : integer;
      X:array[1..Nmax] of integer;
      A:array[0..Nmax-1] of  integer;
      fayl : text;
begin
{polinom darajasini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      for i:=1 to n do
      read(fayl, x[i]);
{sikl orqali x[] qiymatlarini kiritamiz}
      close(fayl);
      A[0]:=-X[1];   A[1]:=1;
      for m:=2 to n do
begin
      A[m-1]:=A[m-2]- X[m];
      i:=m-2;
while i>=1 do
begin
      A[i]:= A[i-1] - A[i]* X[m];
      i:= i-1
end;
      A[0]:= - A[0]* X[m];
end;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i:=0 to n-1 do write (fayl, a[i], ' ');
      close(fayl);
end.
```

Ko'phadni bo'lish. Ikkita ko'phadning umumiy ko'rinishi quyidagicha berilgan:

$Q(x) = q_0 + q_1 x + ... + q_m x^m$, $Q(x) \neq 0$,

$P(x) = p_0 + p_1 x + ... + p_n x^n$.

Quyidagi tenglikni $P(x) = Q(x)H(x) + R(x)$ qanoatlantiruvchi ko'phadlarni aniqlang:

$H(x) = h_0 + h_1 x + ... + h_{n-m} x^{n-m}$,

$R(x) = r_0 + r_1 x + ... + r_k x^k$, $k < m$

Masala yechimi. Bu yerda p_n, q_m nolga teng emas deb qabul qilingan. Agarda $k = -1$ bo'lsa, demak $P(x)$ qoldiqsiz $Q(x)$ ga bo'linadi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*Output.txt*
2 1 1 2 1 1 1	1.00 1.00

Bu yerda *input.txt* faylida birinchi qatorda ko'phadlarning darajalari n va m, ikkinchi va uchinchi qatorlarda P va Q ko'phadlarining koeffitsiyentlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida H va R ko'phadlarining koeffitsiyentlari ikki qatorda chiqarilgan bo'ladi. Agar qoldiqsiz bo'linsa, u holda faqatgina H qatori chiqariladi.

```
program DividePolynom;
const
      Nmax=100;
      epsilon =0.0001;
var
      i,j,m,n,k : integer;
      p,q,h,r:array[0..Nmax] of real;
      fayl : text;
begin
{polinom darajalarini va boshqa qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n, m);
      for i:=0 to n do
      read(fayl, p[i]);
```

```
{sikl orqali p[] qiymatlarini kiritamiz}
      for i:=0 to m do
      read(fayl, q[i]);
{sikl orqali q[] qiymatlarini kiritamiz}
      close(fayl);
      for i:=0 to n-m do   h[i]:=0;
      if n>=m then
begin
      j:=n-m;
while (j>=0) do
begin
      h[j]:=p[j+m]/q[m];
      i:=m;
while (i>=0) do
begin
      p[i+j]:=p[i+j]-h[j]*q[i];
      i:=i-1;
end;
            j:=j-1;
end ;
      k:=m-1;
while (true) do
begin
      if k<0 then break;
      if(abs(p[k]) >=epsilon) then break;
      k:=k-1;
end
end
      else
      k:=n;
      if(k>=0) then
begin
      for i:=0 to k do  r[i]:=0;
      i:=k;
while (i>=0) do
begin
      r[i]:=p[i];
      i:=i-1;
end;
```

```
end;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i:=0 to n-m do  write(fayl, h[i]:2:2, ' ');
      writeln(fayl);
      for i:=0 to k do    write(fayl, r[i]:2:2, ' ');
      close(fayl);
end.
```

Teskari ko'phad. Berilgan $A(x) = a_0 + a_1 x + ... + a_n x^n$ ko'phad uchun unga teskari bo'lgan $B(x) = 1/A(x) = x^{-k}(b_0 + b_1 x + ... + b_m x^m + ...)$ ko'phadning m ta hadini aniqlang. Bu yerda k – bu $A(x)$ ko'phadining nolga teng bo'lgan koeffitsiyentlar soni, ya'ni $a_0 = 0, a_1 = 0, ..., a_k = 0$.

Masala yechimi. Bu yerda bj koeffitsiyentlari quyidagi formulalar bo'yicha hisoblanadi:

$b_0 = 1/a_0;$

$b_j = -b_0 (a_1 b_{j-1} + ... + a_j b_0), j = 1..n;$

$b_j = -b_0 (a_1 b_{j-1} + ... + a_n b_{j-n}), j = n+1, n+2, ...;$

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
2 3 0 1 1	1 1.00 -1.00 1.00 -1.00

Bu yerda *input.txt* faylida birinchi qatorda berilgan ko'phadning darajasi n va qidiralayotgan $B(x)$ ko'phadining darajasi m, ikkinchi qatorda $A(x)$ ko'phadining koeffitsiyentlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida k ning qiymati va $B(x)$ ko'phadining koeffitsiyentlari ikkinchi qatorda chiqarilgan bo'ladi.

Shundan kelib chiqqan holda, dasturimiz quyidagi ko'rinishda bo'ladi:

```
program InvertPolynom;
const Nmax=100;
var i,j,p,m,n,k : integer;
      a : array[0..Nmax] of integer;
      b : array[0..Nmax] of real;
      fayl : text;
begin
```

```
{polinom darajasi va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n, m);
      for i:=0 to n do
      read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
      close(fayl);
      for i:=0 to m do b[i]:=0;
      i := 0;
while ((i<n) and (a[i]=0)) do i:= i+1;
      if ( a[i] <> 0 ) then
begin
      k:= i;
      b[0] := 1.0/a[i];
      if( n-k>=1 ) then
begin
      i := 1;
while(i<=m) do
begin
      b[i] := 0;
      p := i-1;
      if( i-p+k>n ) then
begin
      p:= i+k-n;
end;
      j := 0;
while (j<=p) do
begin
      b[i] := b[i]+b[j]*a[i-j+k];
      j := j+1;
end;
      b[i] := -b[i]*b[0];
      i := i+1;
end;  { while(i<=m) tugadi }
end; { if( n-k>=1 ) tugadi}
end; { if ( a[i] <> 0 ) tugadi}
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
```

```
    rewrite(fayl);
    write (fayl, k); writeln(fayl);
    for i:=0 to m do   write (fayl, b[i], ' ');
    close(fayl);
end.
```

Ko‘phad argumentini chiziqli o‘zgartirish. Berilgan quyidagi ko‘rinishdagi ko‘phadda $P(x) = c_0 + c_1 x + \ldots + c_n x^n$ x ning o‘rniga $x=at+b$ deb olinganda, hosil bo‘lgan ko‘phad koeffitsiyentlarini aniqlang.

Masala yechimi. Bevosita o‘rniga qo‘yish orqali koeffitsiyentlarni aniqlash mumkin bo‘ladi:

$P(at+b) = c_0 + c_1 (at+b) + \ldots + c_n (at+b)^n = d_0 + d_1 x + \ldots + d_n x^n$

Bu yerda

$$d_i = \sum_{k=i}^{n} C_k^i a^i b^{k-i} c_k$$

$$C_k^i = \frac{k(k-1)\ldots(k-i+1)}{k!}$$

Demak d_i larni hisoblash kifoya bo‘ladi.

Masaladagi ma’lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
2 2 -3 1 2 1	4 -8 4

Bu yerda *input.txt* faylida birinchi qatorda ko‘phadning darajasi n, a va b qiymatlari, ikkinchi qatorda esa $P(x)$ ko‘phadining koeffitsiyentlari bo‘sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi ko‘phadning koeffitsiyentlari chiqarilgan bo‘ladi.

Shundan kelib chiqqan holda, dasturimiz quyidagi ko‘rinishda bo‘ladi:

```
program LinePolynom;
const Nmax=100;
var
    i,j,a,b,m,n,k : integer;
    c,d,w,z : array[0..Nmax] of integer;
```

```
      fayl : text;
begin
{polinom darajasi va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n, a, b);
      for i:=0 to n do
      read(fayl, c[i]);
{sikl orqali c[] qiymatlarini kiritamiz}
      close(fayl);
      for i:=0 to n do
begin
      d[i]:=0; w[i]:=0; z[i]:=0;
end;
      d[0] := c[0];
      z[0] := 1;
      w[0] := 1;
      i := 1;
while (i<=n) do
begin
      w[i] := 1;
      z[i] := b*z[i-1];
      d[0] := d[0]+c[i]*z[i];
      i := i+1;
end;
      j := 1;
while (j<=n) do
begin
      w[0] := a*w[0];
      d[j] := c[j]*w[0];
      i := j+1;
      k := i-j;
      w[k] := a*w[k]+w[k-1];
      d[j] := d[j]+c[i]*w[k]*z[k];
      i := i+1;
      j := j+1;
end;   { while (j<=n) sikli tugadi}
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
```

```
    rewrite(fayl);
    write (fayl, k); writeln(fayl);
    for i:=0 to m do   write (fayl, b[i], ' ');
    close(fayl);
end.
```

Ko'phad argumentini chiziqli o'zgartirish. Berilgan quyidagi ko'rinishdagi ko'phadda $P(x) = c_0 + c_1 x + ... + c_n x^n$ x ning o'rniga $x=at+b$ deb olinganda, hosil bo'lgan ko'phad koeffitsiyentlarini aniqlang.

Masala yechimi. Bevosita o'rniga qo'yish orqali koeffitsiyentlarni aniqlash mumkin bo'ladi:
$P(at+b) = c_0 + c_1 (at+b) + ... + c_n (at+b)^n = d_0 + d_1 x + ... + d_n x^n$
Bu yerda

$$d_i = \sum_{k=i}^{n} C_k^i a^i b^{k-i} c_k$$

$$C_k^i = \frac{k(k-1)...(k-i+1)}{k!}$$

Demak d_i larni hisoblash kifoya bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
2 2 -3 1 2 1	4 -8 4

Bu yerda *input.txt* faylida birinchi qatorda ko'phadning darajasi n, a va b qiymatlari, ikkinchi qatorda esa $P(x)$ ko'phadining koeffitsiyentlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi ko'phadning koeffitsiyentlari chiqarilgan bo'ladi.

Shundan kelib chiqqan holda, dasturimiz quyidagi ko'rinishda bo'ladi:

```
program LinePolynom;
const Nmax=100;
var
    i,j,a,b,m,n,k : integer;
    c,d,w,z : array[0..Nmax] of integer;
```

```
      fayl : text;
begin
{polinom darajasi va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n, a, b);
      for i:=0 to n do
      read(fayl, c[i]);
{sikl orqali c[] qiymatlarini kiritamiz}
      close(fayl);
      for i:=0 to n do
begin
      d[i]:=0; w[i]:=0; z[i]:=0;
end;
      d[0] := c[0];
      z[0] := 1;
      w[0] := 1;
      i := 1;
while (i<=n) do
begin
      w[i] := 1;
      z[i] := b*z[i-1];
      d[0] := d[0]+c[i]*z[i];
      i := i+1;
end;
      j := 1;
while (j<=n) do
begin
      w[0] := a*w[0];
      d[j] := c[j]*w[0];
      i := j+1;
      k := i-j;
      w[k] := a*w[k]+w[k-1];
      d[j] := d[j]+c[i]*w[k]*z[k];
      i := i+1;
      j := j+1;
end;   { while (j<=n) sikli tugadi}
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
```

```
    rewrite(fayl);
  for i:=0 to n do  write (fayl, d[i], ' ');
  close(fayl);
end.
```

Ko'phadlar ko'paytmasi. Berilgan quyidagi ko'rinishlardagi 2 ta ko'phadning
$Q(x) = q_0 + q_1 x + ... + q_m x^m$ va $P(x) = p_0 + p_1 x + ... + p_n x^n$ ko'paytmasidan hosil bo'lgan ko'phad koeffitsiyentlarini aniqlang.

Masala yechimi. Demak, masalaning sharti bo'yicha quyidagi ko'phadning h_i koeffitsiyent-larini aniqlash lozim bo'ladi:

$$H(x) = P(x)Q(x) = h_0 + h_1 x + ... + h_{n+m} x^{n+m}$$

Bir qarashda murakkab masalaga o'xshaydi, lekin q_i va p_j koeffitsiyentlar ko'paytmasi bevosita x^{i+j} hadning koeffitsiyenti bo'ladi, demak ularning barchasini yig'indisini hisoblasak kifoya bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

Input.txt	*output.txt*
1 1 1 1 1 1	1 2 1

Bu yerda *input.txt* faylida birinchi qatorda ko'phadlarning darajalari m va n, ikkinchi qatorda esa $Q(x)$ va $P(x)$ ko'phadlarining koeffitsiyentlari ikki qatorda bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi ko'phadning koeffitsiyentlari chiqarilgan bo'ladi.

Dastur esa quyidagicha bo'ladi:

```
program MultiplyPolynom;
const Nmax=100;
var
     i,j,m,n : integer;
     Q, P, H : array[0..Nmax] of integer;
     fayl : text;
begin
{polinom darajasi va boshqa  qiymatlarni kiritamiz}
     assign(fayl, 'input.txt');
     reset(fayl);
```

```
        read(fayl, m, n);
        for i:=0 to m do read(fayl, q[i]);
 {sikl orqali q[] qiymatlarini kiritamiz}
        for i:=0 to n do
        read(fayl, p[i]);
 {sikl orqali p[] qiymatlarini kiritamiz}
        close(fayl);
        for i:=0 to n+m do H[i]:=0;
 {endi hisoblaymiz}
        for i:=0 to m do
        for j:=0 to n do
        H[i+j]:= H[i+j]+ Q[i]* P[j];
 {chiqarish fayliga natijani chop etamiz}
        assign(fayl, 'output.txt');
        rewrite(fayl);
        for i:=0 to n+m  do  write (fayl, H[i], ' ');
        close(fayl);
end.
```

Ko‘phadni tezkor hisoblash. Berilgan quyidagi ko‘rinishdagi ko‘phadning
$f(x)=ax^4+bx^3+cx^2+dx+e$ barcha x=1, ..., 100 qiymatlari uchun qiymati aniqlanib, ularning yig‘indisi hisoblansin va undagi amallar soni katta bo‘lmasin.

Masala yechimi. Qo‘yilgan cheklov, bevosita masalani to‘g‘ridan-to‘g‘ri yechishga imkon bermaydi, shu bois $f(x+1)$ ni $f(x)$ orqali aniqlashga harakat qilamiz:
$f(x+1)=f(x)+(4ax^3+(6a+3b)x^2+(4a+3b+2c)x+(a+b+c+d)=f(x)+g(x)$,
bu yerda

$$g(x)=Ax^3+Bx^2+Cx+D,$$

bu erda $A=4a$, $B=6a+3b$, $C=4a+3b+2c$ va $D=a+b+c+d$.
$g(x)$ funksiyasini ham xuddi shunday aniqlaymiz :
$g(x+1)=g(x)+(3Ax^2+(3A+2B)x+(A+B+C))=g(x)+h(x)$, bundan
$h(x)=A'x^2+B'x+C'$, bu yerda $A'=3A$, $B'=3A+2B$ va $C'=A+B+C$.
Bu yerdan

$$h(x+1)=h(x)+(2A'x+(A'+B'))=h(x)+p(x),$$

Bu yerdan

$$p(x)=A''x+B'' \text{ va } p(x+1)=p(x)+A'',$$

bu yerda $A''=A'$ va $B''=A'+B'$ bo‘ladi.

Shunday qilib quyidagi formulalarni tashkil qilish mumkin:

$$f(x+4)=f(x+3)+g(x+3)$$
$$g(x+3)=g(x+2)+h(x+2)$$
$$h(x+2)=h(x+1)+p(x+1)$$
$$p(x+1)=p(x)+A''.$$

Hisoblashlar $x=1$ dan boshlanadi, shu bois $p(1)$, $h(2)$, $g(3)$ va $f(4)$ larning qiymatlarini oldindan hisoblab olishimiz kerak.

$p(1) = A''+B''$
$h(2) = A' \cdot 2^2 + B' \cdot 2 + C'$
$g(3) = A \cdot 3^3 + B \cdot 3^2 + C \cdot 3 + D$
$f(4) = a \cdot 4^4 + b \cdot 4^3 + c \cdot 4^2 + d \cdot 4 + e$

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
1 1 1 1 1	2076178832

Bu yerda *input.txt* faylida birinchi qatorda ko'phadning *a, b, c, d, e* koeffitsiyentlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yig'indi chiqarilgan bo'ladi.

Asosiy dasturimiz esa quyidagicha bo'ladi:

```
program TezkorPolynom;
var
    x,a,b,c,d,e : integer;
    s, p,h,g,f : longint;
    kA,kB,kC,kD : integer;
    A1,B1,C1,D1 : integer;
    A2,B2,C2,D2 : integer;
    fayl: text;
begin
{polinom koeffitsiyentlarini kiritamiz}
    assign(fayl, 'input.txt');
    reset(fayl);
    read(fayl, a, b, c, d, e);
    close(fayl);
    kA:=4*a; kB:=6*a+3*b; kC:=4*a+3*b+2*c; kD:=a+b+c+d;
    A1:=3*kA; B1:=3*kA+2*kB; C1:=kA+kB+kC;
    A2:=2*A1; B2:=A1+B1;
    p := A2+B2;
```

```
      h := A1*4 + B1*2+ C1;
      g := kA*27+kB*9+kC*3+kD;
      f := a*256+b*64+c*16+d*4+e;
{endi f ni x=5 dan boshlab hisoblaymiz va chop etamiz}
      x:=5; s:=0;
repeat
begin
      P:=P+A2;
      H:=H+P;
      G:=G+H;
      F:=F+G;
      s:=s+f;
      x:=x+1;
end
until x=101;
{chiqarish faylini ochamiz va natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      write (fayl, s);
      close(fayl);
end.
```

Bu erdan bajarilayotgan amallar soni har bir qadamda 5 ta bo'lmoqda, demak jami bo'lib 500 ta bo'ladi, qolgan $f(1), f(2), f(3), f(4)$ larni va qiymatlashlarni $P:=p(1)$; $H:=h(2)$; $G:=g(3)$; $F:=f(4)$; $A_2:=A''$ bajarish uchun ko'p amal talab etilmaydi.

7.3. Satrli ko'phadlar bilan ishlash

Agar ko'phad o'zining koeffitsiyentlari orqali berilgan bo'lsa, u holda uning qiymatini yuqoridagi dastur orqali osongina hisoblab olishimiz mumkin. Lekin ba'zida ko'phad yozuv orqali berilishi mumkin.

Satrli ko'phad qiymati. Ko'phad satr ko'rinishida kiritilgan, masalan

*-2+x^1-3*x^2+x^2+10*x^3-2*x*

va uning qiymatini berilgan x uchun hisoblang.

Masala yechimi. Keltirilgan misoldan ko'rinib turibdiki, ko'phad quyidagi ko'rinishdagi birhadlar yig'indisidan iborat bo'ladi:

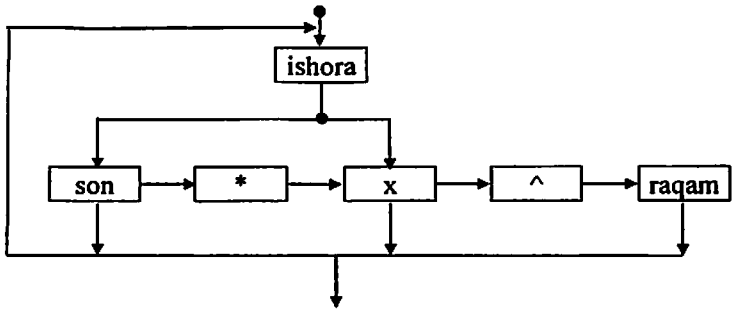
<Ko'phad>::= [-]<Birhad> { (+ | -) <Birhad>}

<Birhad>::= { [<Koeffitsiyent>*]x[<^Daraja>] } | <Koeffitsiyent>

Bu yerda quyidagi cheklovlarni kiritamiz:

1) Koeffitsiyent – bu natural son va u 20 dan kichik;
2) Daraja – bu natural son va u 5 dan kichik;
3) Argument "*x*" – kichik lotin harfida yozilgan bo'lib, uning qiymati kichik butun sonlarda beriladi;
4) Ko'phadda birhadlar soni 10 tadan kam bo'lib, unda bir xil ko'rinishdagi birhadlar ham uchrashishi mumkin.

Ko'phadni quyidagi chizmada keltirilgan tartibda tahlil qilib chiqamiz:



Ko'phadni dasturda s satrli o'zgaruvchida saqlaydigan bo'lsak, yuqoridagi chizmadan u doimo ishoradan boshlanishi kerak, shu bois dasturda quyidagi operatorlarni kiritib qo'yamiz:

if s[1]<>'-' then s := '+' + s;

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
1 -2+x^1-3*x^2+x^2+10*x^3-2*x	5

Bu yerda *input.txt* faylida birinchi qatorda argument *x* ning qiymati va ikkinchi qatorda ko'phad yozilgan, *output.txt* faylida esa ko'phad qiymati chiqarilgan bo'ladi.

Qolgan izohlarni dasturni o'zida berib o'tamiz.

```
program PolynomText;
var
    result, power , coef, sign, i,x : integer;
    s : string;
    fayl : text;
```

```
{Darajani hisoblash funksiyasi}
function Xpow(x, power: integer) : integer;
var
      i,s : integer;
begin
      s:= 1 ;
      for i := 1  to power   do   s:=s*x;
      Xpow:= s ;
end;
begin
{polinom argumenti va satrli ko'phadni  kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl,x);
      readln(fayl,s);
      close(fayl);
      if  s[1]<> '-'  then   s := '+' + s;
      result  := 0;
      i:= 1;       {satrdagi o'rin}
repeat
      coef:=0;  { x oldidagi koeffitsiyent }
      sign:=1;  { koeffitsiyent ishorasi }
      power:=0;  { x darajasi}
      if  s[i]= '-'  then   sign:=-1;
{-ishorani aniqlab oldik}
      inc(i); { keyingi o'ringa o'tdik}
      if  s[i]<> 'x'  then
{bu yerda koeffitsiyentni to'liq aniqlab olamiz}
repeat
      coef:=coef*10+(ord(s[i])-ord('0'));
      inc(i);
until NOT((i<=length(s)) AND (s[i] in ['0'..'9']))
      else coef:=1;
      if (i<=length(s)) AND (s[i]= '*') then inc(i);
{ '*' belgisini o'tkazib yuboramiz }
      if (i<=length(s)) AND (s[i]= 'x') then
begin
      inc(i); { 'x' belgisini o'tkazib yuboramiz }
      if (i<=length(s)) AND (s[i]= '^') then
```

```
begin
      inc(i); { '^' belgisini o'tkazib yuboramiz }
      power:= ord(s[i])-ord('0'); { darajani aniqladik}
      inc(i);
end
      else
      power:=1;
end;
      result:=result+sign*coef*Xpow(x,power);
until i>length(s);
{chiqarish faylini ochamiz va natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      write (fayl, result);
      close(fayl);
end.
```

Satrli ko'phadlarning ko'paytmasi. Berilgan 2 ta ko'phad maxsus ko'rinishdagi satrli yozuv sifatida o'qib olinadi, masalan

*x^2+x+x-2*x^3* va *x+1.*

Ularning koeffitsiyentlari butun son, darajalari esa 10 dan kichik bo'lib, ularning ko'paytmasini yozuv ko'rinishida va darajasi kamayish tartibida chiqarilsin.

Masala yechimi. Oldiniga kiritilayotgan yozuvlarni o'qib olish jarayonini dasturini tuzib olamiz. Bu erda ko'phadni oldingi misolda keltirilgan qoidalar bo'yicha ta'riflab o'tamiz:

<Ko'phad>::= [-]<Birhad> { (+ | -) <Birhad>}

<Birhad>::= { [<Koeffitsiyent>*]x[<^Daraja>] } | <Koeffitsiyent>

Endi, bu yerda har bir qoidani qism dastur orqali aniqlaymiz.

Ushbu ta'riflarni dasturlash tiliga aylantirishdan oldin biz satrlarni o'qib olishimiz va uning har bir belgisini tahlil qilishimiz kerak bo'ladi.

Bunda bizga quyidagi o'zgarmas qiymatlar

```
const
_End=#10; {ifoda tugadi}
```

va quyidagi o'zgaruvchilar kerak bo'ladi:

```
type
Poly = array [0..100] of integer;
```

```
{- ko'phad koeffitsiyentlari }
var
s,s1,s2: string; {o'qib olingan satr}
c : char; {ifoda belgisi}
pos: integer;{belgi o'rni}
vl : integer; {qiymat}
vlDaraja, maxDaraja , vlSon, vlSign: integer;
OneMaxDaraja, TwoMaxDaraja: integer;
arrOnePolynom, arrTwoPolynom, arrResultPolynom : Poly;
faylin, faylout : text;
```

Bu yerda ko'phad koeffitsiyentlari, masalan birinchi kiritilgan ko'phad uchun, *arrOnePolynom*[] massivida saqlanadi. Bunda massiv indeksi darajani va elementi esa koeffitsiyentlarni anglatadi. Masalan, 5*x^2 + 10 – 3*x uchun

arrOnePolynom[0]=10,
arrOnePolynom[1]=-3,
arrOnePolynom[2]=5.

Ifodaning elementlarini birin-ketin o'qib olish uchun *nextChar* qism dasturini yaratamiz.

```
Procedure nextChar;
begin
if pos<length(s) then
begin
inc(pos);
c:=s[pos];
end
else
c:=_End; { satr tugadi}
end;
```

O'zgaruvchilarga boshlang'ich qiymatlarni quyidagi qism dasturda amalga oshiramiz:

```
Procedure MyUnit (var So: string; var maxDaraja :integer);
begin
s:=So;
c:=' ';
pos:=0;
```

```
maxDaraja:=0;
end;
```

Endi yuqorida keltirilgan ta'riflarni to'g'ridan-to'g'ri dasturlashtirishga o'tamiz. Unda kvadrat qavslarga **if** operatori mos keladi, figurali qavslarga esa **while** operatori va oxirgi "|" chizig'i (**or**) **if-else** operatorlariga mos keladi.

```
Procedure Daraja (var MaxDaraja:integer);
begin
vlDaraja:=1;{bu erga kelsak, 'x' da turgan bo'lamiz}
nextchar;
if c='^' then
{keyingi belgi '^' bo'lmasa, vlDaraja:=1 bo'lib qoladi}
begin
nextchar;
vlDaraja:=ord(c) - ord('0');
{'c' belgisini songa aylantiramiz }
nextchar;
end;
if vlDaraja > MaxDaraja then MaxDaraja:= vlDaraja;
{- ko'phadning eng katta darajasini ham aniqlab oldik }
end;
```

Birhadni tahlilini ko'rib chiqamiz. Ushbu dastur faqatgina *c*='+' yoki *c*='-' bo'lgandagina chaqiriladi:

```
Procedure Birhad(var arrayPolinom : Poly; var MaxDaraja
:integer);
begin
vlDaraja:=0;
{- chunki birhad sondan iborat bo'lishi mumkin}
vlSon:=0; {birhad koeffitsiyenti}
if c='+' then vlSign:=1;
if c='-' then vlSign:=-1;
{birhadning birinchi belgisi bo'yicha, uning ishorasini} {saqlab
oldik}
nextchar; {keyingi belgini oldik}
if c in ['1'..'9'] then
begin
```

```
while c in ['0'..'9'] do
begin
vlSon:= ord(c) - ord('0') + vlSon*10;
nextChar;
end;
end;
if vlSon=0 then vlSon:=1;
{Ya'ni, siklda son uchratmadik, u holda koeffitsiyent} {birga teng bo'ladi }
{masalan, x^2 birhadi uchun koeffitsiyent birga teng }
if c = '*' then
begin
nextChar;
end;
if (c='x') then
begin
Daraja(MaxDaraja);
end;
if vlSign=-1 then vlSon:=-vlSon;
{ birhad ishorasini to'g'rilab qo'ydik }
arrayPolinom[vlDaraja]:= arrayPolinom[vlDaraja]+ vlSon;
{bu yerda bir hil birhadlar qo'shib boriladi, masalan,} {4*x+1-x uchun birinchi}
{darajali birhadlar qo'shib olinadi}
end;
```

Endi bevosita ko'phadning tahlilini boshlaymiz.

```
Procedure Kuphad (var arrayPolinom:Poly; var MaxDaraja :integer) ;
Begin
nextchar;
while c <> _End do
begin
if ((c='+') or (c='-')) then
birhad(arrayPolinom , MaxDaraja);
if pos=length(s) then c:=_End;
end;
end;
```

Shunday qilib, agar "*Kuphad*" dasturi ishga tushirilsa, *arrayPolinom*[] massivida ko'phadning koeffitsiyentlari joylashgan bo'ladi.

Ikkala ko'phadning koeffitsiyentlari aniqlangandan so'ng natijaviy ko'phadning koeffitsiyentlarini aniqlash qiyinchilik tug'dirmaydi:

```
Procedure Multiply;
var
i,j : integer;
begin
for i:=0 to OneMaxDaraja+TwoMaxDaraja do
arrResultPolynom[i]:=0;
for i:=0 to OneMaxDaraja do
for j:=0 to  TwoMaxDaraja do
arrResultPolynom[i+j]:=
arrResultPolynom[i+j]+arrOnePolynom[i]*arrTwoPolynom[j]
end;
```

Endi natijani chiqarishni, ya'ni chop etish dasturini ishlab chiqaramiz. Bu yerda quyidagi vaziyatlarni inobatga olish zarur bo'ladi:

- agar birhad nolga teng bo'lsa, u chiqarilmaydi, lekin ko'phad nolga teng bo'lsa, u holda uni chop etamiz;
- birinchi chop etiladigan koeffitsiyentning ishorasi musbat bo'lsa, u chop etilmaydi;
- koeffitsiyent 1 yoki -1 bo'lsa, u chop etilmaydi, faqatgina bo'sh had bo'lsa chop etiladi;
- daraja 1 ga teng bo'lsa, u chop etilmaydi;
- daraja 0 ga teng bo'lsa, daraja ham, '*x*' ham chop etilmaydi;

Oldiniga birhadni chop qilish dasturini tuzamiz, faqatgina musbat koeffitsiyent uchun:

```
Procedure ShowBirhad(coef, pow : integer);
begin
      if coef <=0  then halt(1); {ishni tugatamiz}
      if pow =0  then write(faylout, coef)
      else
begin
      if coef <> 1  then write(faylout, coef);
      write(faylout, 'x');
      if pow <>1  then write(faylout, '^',pow);
end;
end;
```

Endi ko'phadni chop qilish dasturini tuzamiz:

```
Procedure ShowKuphad(P:Poly; MaxDaraja : integer);
var i,j: integer;
begin
     i:= MaxDaraja;
     if p[i] <0  then
begin
     write(faylout, '-');
     ShowBirhad(-p[i],i);
end
     else if p[i] > 0  then   ShowBirhad(p[i],i)
     else   write(faylout, 0);
{ya'ni bizda ko'phad nolga teng, bu esa i=0 demakdir}
     for j:=i-1 downto 0 do
begin
     if p[j] <0  then
begin
     write(faylout, '-');
     ShowBirhad(-p[j],j);
end
     else if p[j] > 0  then
begin
     write(faylout, '+');
     ShowBirhad(p[j],j)
end;
end;
end;
```

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
x^2+x+x-2*x^3 x+1	-2x^4-x^3+3x^2+2x

Bu yerda *input.txt* faylida birinchi va ikkinchi qatorda ko'phadlar yozilgan, ularning yozuvida *, ^, *x* va raqamlar qo'llanilgan, bunda birhadlar tartibsiz bo'lishi ham mumkin, *output.txt* faylida esa ko'phadlar ko'paytmasi tartib bilan chiqarilgan bo'ladi, lekin e'tibor bering '*' belgisisiz.

Shunday qilib, asosiy dasturimiz to'lig'icha quyidagi ko'rinishda bo'ladi:

```
Program QatorliKuphad;
const
      _End=#10; {ifoda tugadi}
type
      Poly = array [0..100] of integer;
{ ko'phad koeffitsiyentlari uchun}
var
      s,s1,s2: string; {o'qib olingan satr}
      c : char; {ifoda belgisi}
      pos: integer;{belgi o'rni}
      vl : integer; {qiymat}
      vlDaraja,  maxDaraja , vlSon, vlSign: integer;
      OneMaxDaraja, TwoMaxDaraja: integer;
      arrOnePolynom,arrTwoPolynom,arrResultPolynom: Poly;
      faylin, faylout : text;
Procedure nextChar;
begin
      if pos<length(s) then
begin
      inc(pos);
      c:=s[pos];
end
else
      c:=_End; { satr tugadi}
end;
Procedure MyUnit (var So: string; var maxDaraja :integer);
begin
      s:=So; c:= ' ';
      pos:=0; maxDaraja:=0;
end;
Procedure Daraja (var MaxDaraja:integer);
begin
      vlDaraja:=1;
{bu erga kelsak, 'x' da turgan bo'lamiz}
      nextchar;
      if c='^' then
begin
```

```
      nextchar;
      vlDaraja:=ord(c) - ord('0');
      nextchar;
end;
      if vlDaraja > MaxDaraja then MaxDaraja:= vlDaraja;
{- ko'phadning eng katta darajasini ham aniqlab oldik }
end;
Procedure Birhad(var arrayPolinom : Poly; var MaxDaraja
:integer);
begin
      vlDaraja:=0;
      vlSon:=0; {koeff}
      if c='+' then vlSign:=1;
      if c='-' then vlSign:=-1;
      nextchar;
      if c in ['1'..'9'] then
begin
while c in ['0'..'9'] do
begin
      vlSon:= ord(c) - ord('0') + vlSon*10;
      nextChar;
end;
end;
      if vlSon=0 then vlSon:=1;
      if c = '*' then
begin
      nextChar;
end;
      if (c='x') then
begin
      Daraja(MaxDaraja);
end;
      if vlSign=-1 then vlSon:=-vlSon;
arrayPolinom[vlDaraja]:= arrayPolinom[vlDaraja]+ vlSon;
end;
Procedure Kuphad (var arrayPolinom: Poly; var MaxDaraja
:integer) ;
begin
      nextchar;
```

```
while c <> _End  do
begin
      if ((c='+') or (c='-')) then
      birhad(arrayPolinom , MaxDaraja);
      if pos=length(s) then c:=_End;
end;
end;
Procedure Multiply;
Var i,j : integer;
begin
      for i:=0 to OneMaxDaraja+ TwoMaxDaraja do
      arrResultPolynom[i]:=0;
for i:=0 to OneMaxDaraja do
for j:=0 to  TwoMaxDaraja do
      arrResultPolynom[i+j]:=arrResultPolynom[i+j]+
      arrOnePolynom[i]*arrTwoPolynom[j];
end;
Procedure ShowBirhad(coef, pow : integer);
begin
      if coef <=0  then halt(1); {ishni tugatamiz}
      if pow =0  then write(faylout, coef)
      else
begin
      if coef <> 1  then write(faylout, coef);
      write(faylout, 'x');
      if pow <>1  then write(faylout, '^',pow);
end;
end;
Procedure ShowKuphad(P:Poly; MaxDaraja : integer);
var i,j: integer;
begin
      i:= MaxDaraja;
      if p[i] <0  then
begin
      write(faylout, '-');
      ShowBirhad(-p[i],i);
end
      else if p[i] > 0  then   ShowBirhad(p[i],i)
      else  write(faylout, 0);
```

```
{ya'ni bizda ko'phad nolga teng, bu esa i=0 demakdir}
      for j:=i-1 downto 0 do
begin
      if p[j] <0  then
begin
      write(faylout, '-');
      ShowBirhad(-p[j],j);
end
      else if p[j] > 0  then
begin
      write(faylout, '+');
      ShowBirhad(p[j],j)
end;
end;
end;
{Asosiy dastur}
begin
 {qiymatlarni kiritamiz}
{polinom argumenti va satrli ko'phadni  kiritamiz}
      assign(faylin, 'input.txt');
      reset(faylin);
      read(faylin,s1); { Birinchi polinom }
      readln(faylin);
      read(faylin,s2);{ Ikkinchi polinom }
      close(faylin);
      if s1[1] <> '-' then s1 := '+' + s1;
      if s2[1] <> '-' then s2 := '+' + s2;
      myunit(s1, OneMaxDaraja);
      kuphad(arrOnePolynom, OneMaxDaraja);
      myunit(s2, TwoMaxDaraja);
      kuphad(arrTwoPolynom, TwoMaxDaraja);
{endi ko'paytiramiz}
      Multiply;
{chop etamiz}
{chiqarish faylini ochamiz va natijalarni qism dasturlardan chop
etamiz}
      assign(faylout, 'output.txt');
      rewrite(faylout);
```

```
ShowKuphad(arrResultPolynom,OneMaxDaraja+
TwoMaxDaraja);
    close(faylout);
end.
```

Topshiriqlar

1. Butun A, B, C, D sonlardan iborat $Ax^3+Bx^2+Cx+D=0$ tenglamasini butun sonlardagi yechimini aniqlash dasturini tuzing.
2. Bir noma'lumli tenglamani yechimini aniqlash lozim. Bunda tenglama 5 ta belgidan iborat bo'lib, ikkinchi belgi '+' (plyus) yoki '-' (minus) bo'ladi, to'rtinchi belgi esa '=' (tenglik) bo'ladi. Birinchi, uchinchi va beshinchi belgilardan 2 tasi 0 va 9 oralig'idagi raqam bo'lib, faqatgina bittasi "x" harfi bilan berilgan, masalan $21-x=12$.
3. Quyidagi ko'rinishdagi uchhad $a + bx + cy$ koeffitsiyentlari bilan berilgan bo'lsa, uni matematikada qabul qilingan qoidalar bo'yicha chop eting:
   - agar noma'lum oldidagi koeffitsiyent +1 yoki -1 bo'lsa, u holda u yozilmaydi;
   - agar koeffitsiyent nolga teng bo'lsa, u holda ushbu had tashlab ketiladi, bunda agar barcha koeffitsiyentlar nol bo'lsa, u holda ko'phad nolga teng bo'lib qoladi;
   - manfiy koeffitsiyent oldidan "+" ishorasi qo'yilmaydi;
   - ifodaning boshida "+" ishorasi qo'yilmaydi;
   - koeffitsiyent bilan noma'lum orasida ko'paytiruv belgisi qo'yilmaydi.

Bundan tashqari, hadlar o'rnini almashtirish mumkin emas.
4. Butun b, c sonlardan iborat $x^2+bx+c=0$ kvadrat tenglamani butun sonlardagi yechimini aniqlab, uni ko'paytuvchilarga ajratib, matematikada qabul qilingan qoidalar bo'yicha yozing, masalan, $x^2+2x+1 = (x+1)(x+1)$.
5. Berilgan ko'phadda $P(x) = c_0+c_1x+...+c_nx^n$ x ning o'rniga $x=at^2+bt+c$ deb olinganda hosil bo'lgan ko'phad koeffitsiyentlarini aniqlang.
6. Birinchi darajali algebraik ifoda satr orqali kiritilgan bo'lsa, uni matematikada qabul qilingan qoidalar bo'yicha soddalashtiring va uni alfavit bo'yicha chop eting, masalan, $2+a+3b-4-2a+b$ ifodasi soddalashtirilsa, natija $-a+4b-2$ bo'ladi.

7. Berilgan ko'phadda $P(x) = c_0 + c_1 x + \ldots + c_n x^n$ har bir x ning 0 dan k gacha bo'lgan qiymatlari uchun ($P(x)$ *mod* m) ni aniqlang.
8. Berilgan ko'phaddan $P(x) = c_0 + c_1 x + \ldots + c_n x^n$ yangi $Q(x) = P(x+1) - P(x)$ ko'phadni aniqlang.
9. Ko'phadlar ketma-ketligi $P_0(x)$, $P_1(x)$,... quyidagicha tashkil etilgan: $P_0(x)=1$, $P_1(x)=x$, $P_k(x)=2xP_{k-1}(x)-P_{k-2}(x)$. Istalgan n uchun $P_n(x)$ ko'phadini aniqlang.
10. Ko'phadning umumiy ko'rinishi quyidagicha berilgan:

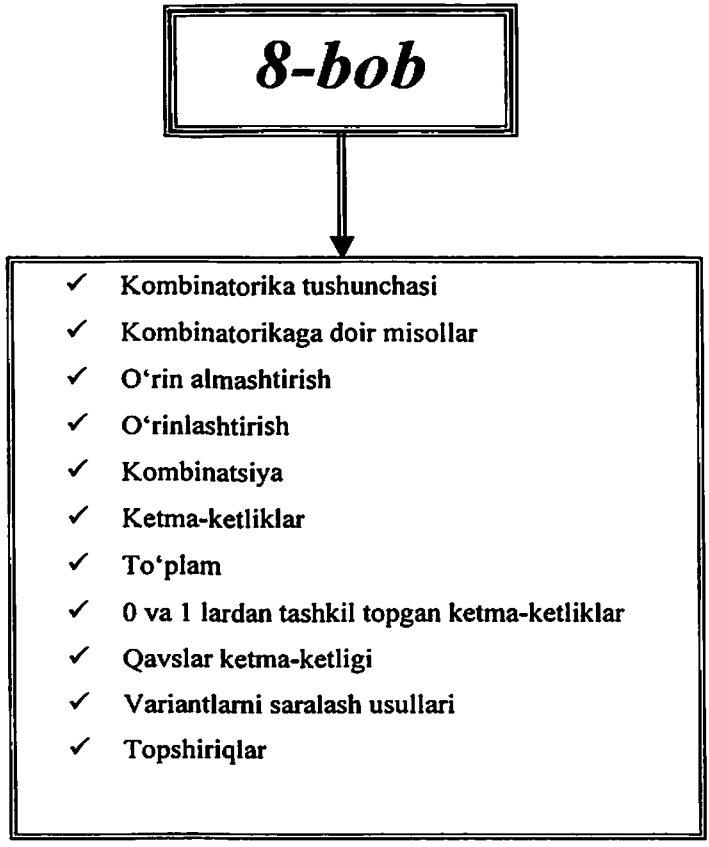
$$P_n(x)=a_0+a_1x+a_2x^2+\ldots+a_{n-1}x^{n-1}+a_nx^n$$

Uning ildizlari $x_1, x_2, \ldots, x_n$ butun sonlarda berilgan bo'lsa, barcha a_i koeffitsiyentlarini aniqlang.

8-bob. KOMBINATORIKAGA DOIR ALGORITMLARNI DASTURLASH

Amaliy misollarning yechimini aniqlashda biz ko'pincha cheklangan to'plam elementlaridan qismiy to'plam tashkil qilamiz. Ushbu qism to'plam elementlari ma'lum bir xususiyatlarga ega bo'lishi yoki ularni qandaydir tartib bo'yicha joylashtirish talab etiladi.

Mazkur bobda dasturlashda qo'llaniladigan kombinatorikaga doir asosiy va murakkab bo'lgan algoritmlar bilan tanishamiz:



8.1. Kombinatorika tushunchasi

Kombinatorikada chekli to'plamlar uchun elementlarning kombinatsiyasi, o'rinlashtirish, o'rin almashtirish va shunga o'xshash tushunchalar o'rganiladi.

O'rin almashtirish deb n ta turli elementlarning bir-biridan faqat joylashishi bilan farq qiluvchi kombinatsiyalariga aytiladi. n ta turli elementlarning o'rin almashtirishlar soni $P_n=n!$ ga teng ($n!=1\cdot2\cdot3\cdot\ldots\cdot n$)).

O'rinlashtirishlar n ta turli elementdan m tadan tuzilgan kombinatsiyalar bo'lib, ular bir-biridan elementlarning tarkibi yoki ularning tartibi bilan farq qiladi. Ularning soni

$$A_n^m = \frac{n!}{(n-m)!}$$

formula bilan topiladi.

Masalan, $\{a, b, s, d\}$ to'plamdan 2 tadan tuzilgan kombinatsiyalari uchun 12 ta o'rinlashtirish variantlari mavjud bo'ladi:

$$A_4^2 = \frac{4!}{(4-2)!} = 3\cdot4 = 12$$

ya'ni ular quyidagilar bo'ladi:

$ab, as, ad, ba, be, bd, ca, cb, cd, da, db, dc.$

E'tibor bering, bu erda $ab\neq ba$, ya'ni o'rinlari har xil bo'lganda, kombinatsiya teng bo'lmaydi, chunki ta'rif bo'yicha tartib bir xil emas.

Xususan, $m=n$ bo'lganda o'rinlashtirishlar bevosita o'rin almashtirishlarga aylanadi:

$$A_n^n = \frac{n!}{(n-n)!} = n!$$

Kombinatsiya deb bir-biridan hech bo'lmaganda, bitta elementi bilan farq qiluvchi n ta elementdan m tadan tuzilgan har qanday qism to'plamiga aytiladi. Ularning soni

$$C_n^m = \frac{n!}{m!(n-m)!}$$

ga teng.

Masalan, $\{a, b, s, d\}$ to'plamidan 2 tadan oladigan bo'lsak, unda 6 ta kombinatsiya variantlari mavjud bo'ladi:

$$C_4^2 = \frac{4!}{2!(4-2)!} = 3 \cdot 4/2 = 6$$

ya'ni ular quyidagilar bo'ladi:

$\{a, b\}, \{a, s\}, \{a, d\}, \{b, s\}, \{b, d\}, \{s, d\}.$

Algebra kursidan biz buni binom koeffitsiyenti deb bilamiz va u Nyuton binomida qo'llaniladi:

$$(x+y)^n = \sum_{k=0}^{n} C_n^k x^k y^{n-k}$$

Bu yerda, agar x=y=1 bo'lsa, qiziqarli natija olsa bo'ladi

$$2^n = \sum_{k=0}^{n} C_n^k$$

O'rinlashtirishlar soni kombinatsiyalar soni bilan quyidagi munosabat orqali bog'langan

$$A_n^m = m!\,C_n^m = \frac{n!}{(n-m)!}$$

Ushbu tushunchalarga asoslangan va variantlar soni faktorial va undan ham ko'p bo'ladigan misollarni kombinatorika masalalariga doir deb ham aytiladi. Kombinatorika masalalarida variantlar sonini kamaytirish asosiy muammolardan biridir.

Kombinatorikaga doir quyidagi klassik bo'lgan masalalarni ko'rib chiqamiz.

8.2. Kombinatorikaga doir misollar

O'rin almashtirish. O'rin almashtirish berilgan to'plam $\{a_1,a_2,...,a_n\}$ elementlarini har xil tartibda joylashtirishdan hosil bo'ladi. Masalan, $\{a,b,c\}$ to'plami uchun quyidagi variantlar hosil qilinadi:

abc, acb, bac, bca, cab, cba.

E'tibor bering, to'plamdagi element takroran qo'llanilmaydi. Umumiy holda to'plam n ta elementdan iborat bo'lsa, u holda variantlar

soni $n!$ bo'ladi. Agar biz $a_1=1$, $a_2=2$, ..., $a_n=n$ deb qabul qilsak, bunda klassik misolga kelamiz, ya'ni $(1,2,...,n)$ boshlang'ich ko'rinish bo'lsa, oxirgisi $(n, n-1,...,1)$ ko'rinishda bo'ladi. Ushbu masalani yechimini $n=5$ bo'lganda batafsil ko'rib chiqamiz.

Birinchi navbatda, taqqoslash tushunchasini kiritamiz. O'rinlashtirishni taqqoslashni quyidagicha kiritamiz. Ikki o'rinlashtirishlar S va P chapdan o'ngga tomon taqqoslaniladi. Qaysi birida birinchi bo'lib katta sonni uchratsak, demak o'sha o'rinlashtirish katta deb hisoblanadi. Masalan, $S=(1,2,3,4,5)$ va $P=(1,2,5,4,3)$ uchun $S<P$ bo'ladi, chunki $S_1=P_1=1$, $S_2=P_2=2$, $S_3=3<P_3=5$. Shunday qilib, bizda boshlang'ich o'rinlashtirish $S=(1,2,3,4,5)$. Faraz qilamiz, ma'lum bir qadamda aniqlangan o'rinlashtirish quyidagicha bo'lsin $P=(3,4,5,2,1)$. Qanday qilib keyingi o'rinlashtirishni tashkil qilamiz. Bu yerda quyidagicha yo'l tutamiz. P o'rinlashtirishda o'ngdan chapga qarab yuramiz va o'sish tartibi buzilgan nuqtani aniqlaymiz, bu yerda 4 sonini, uni ajratib ko'rsatish maqsadida ozgina kattalashtirib yozamiz: $P=(3,\mathbf{4},5,2,1)$. Endi, yana o'ngdan chapga qarab yuramiz va ushbu topilgan sondan, ya'ni 4 dan katta sonni aniqlaymiz, bizda bu 5, uni ham belgilab qo'yamiz, ya'ni $P=(3,\mathbf{4},\mathbf{5},2,1)$.

Endi belgilangan sonlarni o'rinlari bilan almashtirilib, quyidagini hosil qilamiz, ya'ni $P=(3,\mathbf{5},4,2,1)$.

Almashtirilgan o'rindan boshlab o'ng tomondagi sonlarni o'sish tartibi bilan joylashtiramiz. Bu esa qiyinchilik tug'dirmaydi, chunki ular kamayish tartibida edi, shu bois sonlarni teskarilab yozamiz va quyidagini tashkil qilamiz $Q=(3,\mathbf{5},1,2,4)$.

Shunday qilib, P dan keyingi o'rinlashtirish Q bo'ladi, chunki $P<Q$.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
3	1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1

Bu yerda *input.txt* faylida birinchi qatorda elementlar soni $n=3$ va *output.txt* faylida barcha o'rin almashtirishlar chiqarilgan bo'ladi.

Natijada quyidagi dasturni taklif qilish mumkin:

```
program OrinAlmashtirish;
type Pere=array [byte] of byte;
var
      N,i,j:byte;
      X:Pere;
      Yes:boolean;
      fayl : text;
procedure Swap(var a,b:byte);  {o'rin almashtirish}
var c:byte;
begin
      c:=a;a:=b;b:=c
end;
procedure Next(var X:Pere;var Yes:boolean);
var i:byte;
begin
      i:=N-1;
{ i ni qidiramiz}
while (i>0)and(X[i]>X[i+1]) do dec(i);
      if i>0 then
begin
      j:=i+1;
{ j ni qidiramiz }
while (j<N)and(X[j+1]>X[i]) do inc(j);
      Swap(X[i],X[j]);
      for j:=i+1 to (N+i) div 2 do Swap(X[j],X[N-j+i+1]);
      Yes:=true
end
else Yes:=false
end;
{Asosiy dastur}
begin
{n qiymatini fayldan o'qib olamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      close(fayl);
      for i:=1 to N do X[i]:=i;
```

```
{chiqarish faylini ochamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
repeat
    for i:=1 to N do    write (fayl, x[i], ' ');
    writeln(fayl);
    Next(X,Yes)
until not Yes;
    close(fayl);
end.
```

Quyidagi misolda teskari misol keltirilgan. Inversiyalangan o'rinlashtirishdan uni tiklash talab etiladi.

Inversiyalangan o'rinlashtirish. $P=(p_1,p_2,\ldots,p_n)$ bevosita $1,2,\ldots,n$ sonlaridan o'rin almashtirish orqali hosil qilingan. Har qanday P uchun uning inversiyasi $T=(t_1,t_2,\ldots,t_n)$ tuziladi. Bu yerda t_i P ning i sonidan chaproqda joylashgan va i dan katta bo'lgan barcha qiymatlarning soni. Masalan, $n=9$ uchun, agar
$P=(5,9,1,8,2,6,4,7,3)$ bo'lsa , u holda $T=(2,3,6,4,0,2,2,1,0)$.

Har qanday berilgan inversiya T jadvalidan P o'rinlashtirishni tiklab beruvchi dasturni tuzing.

Masala yechimi. Asosiy g'oyani aniqlash uchun masalani chuqurroq o'rganamiz. Agar $i=1$ bo'lsa yuqoridagi misolga binoan $t_i = 2$, lekin $i=1$ eng kichik son va undan chapda $t_i = 2$ ta son bor, demak $i=1$ ni p_3 da yozib qo'yishimiz kerak, ya'ni $t_i + 1$ da. Aks holda , masalan, 1 ni p_4 da yozsak, undan chapda turgan sonlarning hammasi katta bo'lganligi sababli $t_1=3$ bo'ladi, bu esa noto'g'ri. Shunday qilib, $p_3=1$ ga . Keyinchalik , $i=2$ va $t_i = 3$. demak i ni biz P da shunday joyda qo'yishimiz kerakki, undan chapda $t_i = 3$ ta undan katta son bo'lsin. Birni p_3 ga joylashtirganimiz sababli $i=2$ ni $t_2 + 1 + 1 = 3+2=5$ ga joylashtiramiz, ya'ni $p_5=2$.

Ushbu algoritmni bir yo'lakay qo'llash uchun quyidagicha yo'l tutamiz. P ning barcha elementlariga 0 berib chiqamiiz , ya'ni $P=(0,0,0,0,0,0,0,0,0)$. $i=1$ uchun $t_i + 1 = 3$ ta P dan nollarni sanab, oxirgisiga i ning qiymatini beramiz, ya'ni 3-o'ringa $i=1$ qiymati beriladi, ya'ni $p_3=1$. demak, quyidagini hosil qilamiz
$P=(0,0,1,0,0,0,0,0,0)$. $i=2$ uchun $t_i + 1 = 4$ ta P dan nollarni sanab, oxirgisiga $i = 2$ qiymatini beramiz, ya'ni 5-o'ringa , demak $p_5=2$.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
9 2 3 6 4 0 2 2 1 0	5 9 1 8 2 6 4 7 3

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=9, ikkinchi qatorda esa T massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi P massiv elementlari qiymatlari chiqarilgan bo'ladi.

Ushbu algoritm quyidagi dasturda keltirilgan.

```
program InversiyaP;
const Nmax=100;
var
      n,i,j,k : integer;
      P,T:array[1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va boshqa qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      for i:=1 to n do
begin
      read(fayl, T[i]);
{-sikl orqali T[] qiymatlarini kiritamiz}
      P[i]:=0
end;
      close(fayl);
      for i:=1 to n do
begin
      j:=0; k:=0;
repeat
      k:=k+1;
      if P[k]=0 then  j:=j+1;
until j>T[i];
      P[k]:=i;
end;
{chiqarish fayliga natijani chop etamiz}
```

```
    assign(fayl, 'output.txt');
    rewrite(fayl);
    for i :=1 to n do      write (fayl, P[i], ' ' );
    close(fayl);
end.
```

Ko'p hollarda massivlardagi elementlarni ma'lum bir shart bo'yicha o'rinlarini almashtirish talab etiladi. Quyidagi misolda shunga doir oddiy misolni keltiramiz.

Tartibli o'rin almashtirish. $A[n]$ massiv elementlari 0,1 yoki 2 ga teng. Massiv elementlarini shunday o'rin almashtiringki, oldin 0 lar, keyin 1 lar va oxirida 2 lar yozilgan bo'lsin (qo'shimcha massiv kiritish mumkin emas).

Masala yechimi. Masalaning sharti bo'yicha biz, masalan, (1,0,1,1,0,0,2,0,2) massividan (0,0,0,0,1,1,1,2,2) massivini tuzishimiz talab etiladi. Albatta, bevosita o'rin almashtirish orqali, yangi massiv kiritmasdan, algoritmni tuzish mumkin. Lekin massiv elementlari aniq sonlar bilan berilganligi ishni osonlashtiradi. Ya'ni , A massivida nechta 0,1,2 borligini bilsak, shunga qarab A massivini yangidan qo'yilgan talab bo'yicha to'ldirib qo'yamiz.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
9 1 0 1 1 0 0 2 0 2	0 0 0 0 1 1 1 2 2

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=9, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yangi massiv elementlari qiymatlari chiqarilgan bo'ladi.

Ushbu algoritm quyidagi dasturda keltirilgan.

```
program TartibliAlmashtirish;
const Nmax=100;
type el=0..2;
var
    n,i : integer;
    A:array[el] of 0..Nmax;
```

```
    X:array[1..Nmax] of el;
    fayl: text;
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
    assign(fayl, 'input.txt');
    reset(fayl);
    read(fayl, n);
    for i:=1 to n do
    read(fayl, X[i]);
{sikl orqali X[] qiymatlarini kiritamiz}
    close(fayl);
    A[0]:=0; A[1]:=0;A[2]:=0;
    for i:=1 to n do     A[X[i]]:= A[X[i]]+1;
    for i:=1 to n do
    if i<= A[0] then X[i]:=0
    else    if i<= n-A[2] then X[i]:=1 else X[i]:=2;
{chiqarish fayliga natijani chop etamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
    for i :=1 to n do  write (fayl, X[i], ' ');
    close(fayl);
end.
```

Eng katta ko'paytma. Berilgan N ta sondan ko'paytmasi eng katta bo'lgan 3 tasini tanlang.

Masala yechimi. Ushbu misolni yechishda eng oddiy algoritm – bu barcha variantlarni birin-ketin ko'rib chiqishdir. Quyidagi dasturda ushbu g'oya amalga oshirilgan.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
9 1 0 1 1 0 0 2 0 2	2 2 1

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=9, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida natijaviy qiymat chiqarilgan bo'ladi.

Dastur esa quyidagicha bo'ladi.

```
program EKK;
const Nmax=20;
Maxlongint=32000; {katta son}
var
     a : array [1..Nmax] of integer;
     n, i, j, k : longint ;
     t, max : longint;
     i1,i2, i3 : longint;
{-topilgan elementlar indekslari}
     fayl: text;
begin
{massiv o'lchamini va boshqa qiymatlarni kiritamiz}
     assign(fayl, 'input.txt');
     reset(fayl);
     read(fayl, n);
     for i:=1 to n do
     read(fayl, a[i]);
{sikl orqali a[] qiymatlarini kiritamiz}
     close(fayl);
     max := -Maxlongint -1;
     for i:= 1 to n do
     for j:= 1 to n do
     for k:= 1 to n do
     If (i<>j) AND (j<>k) AND (i<>k) then
begin
     t:= a[i] * a[j] * a[k];
     if t>= max then
begin
     max:= t;
     i1:= i; i2 := j; i3:= k;
end;
end;
{chiqarish fayliga natijani chop etamiz}
     assign(fayl, 'output.txt');
     rewrite(fayl);
     write (fayl, a[i1] , ' ', a[i2], ' ' , a[i3]);
     close(fayl);
end.
```

Ushbu dasturda quyidagi kamchiliklar mavjud. Birinchidan, bajariladigan amallar soni $f(n)=O(N^3)$ tartibli, bu esa ko'p vaqtni talab qiladi. Ikkinchidan, agar sonlar katta bo'lsa, ularning ko'paytmasi xotira hajmidan katta bo'lib, xatolikka olib kelishi mumkin. Shu bois misolni yechishda boshqacha yondashuvini ko'rib chiqamiz. Misolning shartidan kelib chiqqan holda, agarda biz 3 ta katta sonlarni topib olsak kifoya bo'ladi. Musbat sonlar uchun haqiqatan shunday. Masalan,

3	1	5	0	9	4	6	2	6	6

sonlari uchun maksimal qiymat $9 \cdot 6 \cdot 6 = 324$ ga teng bo'ladi.
Massivdan eng katta elementni qidirish muammo tug'dirmaydi. Faqatgina biz yo'l-yo'lakay 3 ta katta elementni xotirada saqlab olishimiz kerak bo'ladi.

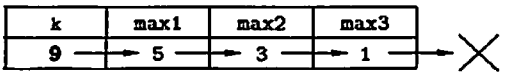
Quyidagi *max1*, *max2*, *max3* o'zgaruvchilarida maksimal qiymatlarni saqlaymiz. Yuqoridagi misol uchun birinchi 3 ta qiymat quyidagicha joylashgan bo'ladi:

Max1	*max2*	*Max3*
5	3	1

Keyingi element $k=0$ da hech narsa o'zgarmaydi. Undan keyingi $k=9$ elementi uchun quyidagi tekshiruvni amalga oshirgandan so'ng qiymatlar o'ngga qarab siljiydi:

```
if k > max1 then
begin
    max3:=max2; max2:=max1; max1:=k;
end;
```

ya'ni



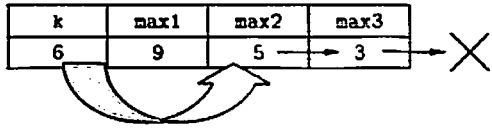
Natija esa quyidagicha bo'ladi

Max1	*Max2*	*Max3*
9	5	3

Agar $k \leq max1$ bo'lsa, u holda keyingi maksimum bilan taqqoslaymiz, ya'ni

```
if k > max2 then
begin
    max3 := max2; max2 := k;
end;
```

Bunga k=6 bo'lganda duch kelamiz



Bunda *max1* o'zgarmay, *k* ning qiymati *max2* ga beriladi, *max3* ga esa oldingi *max2* ning qiymati beriladi. Oxirida, *max1* va *max2* bilan tekshirilganda shart bajarilmasa *k* ni *max3* bilan taqqoslaymiz

```
if k > max3 then max3 :=k;
```

Ushbu oddiy misolda biz faqatgina musbat sonlarni ko'rib chiqdik. Agarda sonlar orasida manfiylari ham bo'lsa, u holda algoritmga qo'shimchalar kiritishga to'g'ri keladi. Ikki manfiy sonning ko'paytmasi musbat bo'lganligi sababli, biz 2 ta eng kichik manfiy sonlarni qidiramiz, ya'ni *min1* va *min2* .

Endi ikki ko'paytmani tekshiramiz: *min1*min2* va *max2*max3*. Keyin ulardan eng kattasini tanlab olamiz. Albatta dasturda uchalasini ham ko'paytirish shart emas.

Boshqa hollarni ham ko'rib chiqamiz. Agarda barcha sonlar manfiy bo'lsa-chi? Bunday holda 3 ta maksimal son yechimi bo'ladi, ya'ni *max1*max2*max3*. Aralash massivda (-3, -1, -5, 0, -9) nol bo'lsa-chi? Bu yerda 0 qiymati *max1* ga tushib qoladi va shu bois natija to'g'ri bo'ladi. Agarda hech bo'lmaganda bitta musbat son bo'lsa (3, 1, -5, 0, -9), u holda max1 hech vaqt 0 ga teng bo'lmaydi, agarda *max2=0* va *max3=0* bo'lsa, u holda eng katta ko'paytma quyidagi elementlardan *min1*min2* va *max1* lardan tashkil topgan bo'ladi.

Quyidagi keltirilgan dasturda biz qo'shimcha massiv kiritib o'tirmaymiz va hisoblashlar qayta takrorlanmasdan to'g'ridan-to'g'ri bajariladi, shu bois amallar soni $f(n)=O(n)$ tartibli bo'ladi.

```
program Makc;
```

```
var
      n : longint;  { elementlar soni}
      k : longint ; {ko'riladigan element}
      i : longint ;
      max1, max2, max3  :  longint;
      min1, min2  :  longint;
      faylin, faylout: text;
begin
      max1 := -30000;  max2  := max1 ;
      max3:= max1;
      min1 := 30000;
      min2  := min1 ;
{fayllarni ochamiz}
      assign(faylin, 'input.txt');
      reset(faylin);
      assign(faylout, 'output.txt');
      rewrite(faylout);
      readln(faylin, n);
{sonlarni bittalab kiritib, qayta ishlaymiz}
      for i:= 1  to n do
begin
      read(faylin, k);
      if  k > max1 then
begin
    max3 := max2;  max2 := max1;
    max1 := k;
end
      else
      if  k > max2 then
begin
      max3 := max2;
      max2 := k;
end
      else
      if  k > max3 then  max3 := k;
      if  k < min1 then
begin
      min2 := min1;
```

```
      min1 := k;
end
      else
      if  k < min2 then
      min2:= k;
end;
      if  (max1>0)  and ( min1* min2 > max2*max3) then
      writeln(faylout, max1, min1  , min2)
      else
      writeln(faylout, max1 , ' ',  max2, ' '  , max3);
      close(faylin);
      close(faylout);
end.
```

Ekskursiya. Samarqandda o'tkaziladigan olimpiada ishtirokchilari uchun tashkiliy qo'mita ekskursiya tashkillashtirgan. Lekin Samarqandning ko'hna mahallalari-dagi ko'chalar juda ham tor va shu bois ulardan avtobuslarni o'tishi mumkin emas. Ekskursiya *N* ta ko'chadan o'tadi va ularning kengligini tashkiliy ko'mita aniqlab olgan. Avtobus ko'chadan o'tishi uchun ko'chaning kengligi 300 sm dan katta bo'lishi kerak. Ekskursiyani amalga oshirish yoki oshiraolmasligini aniqlang va bevosita qaysi ko'chadan o'ta olmasligini aniqlang.

Masala yechimi. Bevosita qiymatlarni birin-ketin kiritib ularni masala shartini qanoatlantirishligini aniqlaymiz.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 1011 402 350 250 490	4

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=9, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida natijaviy qiymat, ya'ni '0' agarda o'tish mumkin bo'lsa va aksincha o'ta olmaydigan ko'cha tartib raqami chiqarilgan bo'ladi.

Quyida dastur keltirilgan.

```
program Ekskursiya;
```

```
Label L;
var
      i,j,n,k : integer;
      faylin, faylout: text;
begin
{fayllarni ochamiz}
      assign(faylin, 'input.txt');
      reset(faylin);
      assign(faylout, 'output.txt');
      rewrite(faylout);
      readln(faylin, n); {ko'chalar soni}
{sonlarni bittalab kiritib, qayta ishlaymiz}
      for i:= 1 to n do
begin
      read(faylin, k); {ko'cha kengligi}
      if k<=300 then
begin
      j:=i; goto L;
end;
end;
L:
      if k>300 then write(faylout, '0') else
      write(faylout, j);
      close(faylin);
      close(faylout);
end.
```

Ketma-ketliklar. 1,2,...,M sonlardan tashkil topgan va uzunligi N bo'lgan barcha ketma-ketliklarni aniqlash talab etiladi, ya'ni (1,1,...,1) dan (M,M,...,M) gacha bo'lgan ketma-ketliklar.

Masala yechimi. Qo'yilgan shartni chuqur anglash uchun quyidagi misolni ko'ramiz, N=4 va M=3 uchun

(1,1,1,1) -> (1,1,1,2) ...
...(1,1,1,3) -> (1,1,2,1)...
...(3,1,3,3) -> (3,2,1,1)...

Bu yerda, e'tibor bersak, «son» ga 1 ni qo'shish natijasida yangi «son» paydo bo'lmoqda, ya'ni (3,1,3,3) +1= (3,2,1,1). Bu yerda nolsiz M-sanoq tizimida amal bajarilayapti. Boshqacha aytganda, agar sonlar

0,1,..,*M*-1 bo'lsa, keyingi ketma-ketlikni aniqlash bu oldingisiga *M*-sanoq tizimida 1 ni qo'shishni anglatadi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
3 2	1 1 1 2 1 3 2 1 2 2 2 3 3 1 3 2 3 3

Bu yerda *input.txt* faylida birinchi qatorda m=3 va n=2 sonlari beriladi va *output.txt* faylida barcha o'rin almashtirishlar chiqarilgan bo'ladi.

Natijada quyidagi dasturni taklif qilish mumkin:

```
program Sequences;
type Sequence=array [byte] of byte;
var
     M,N,i:byte;
     X:Sequence;
     Yes:boolean;
     fayl: text;
procedure Next(var X:Sequence;var Yes:boolean);
var i:byte;
begin
     i:=N;
{ i ni izlash}
while (i>0) and (X[i]=M) do
begin
     X[i]:=1;dec(i)
end;
     if i>0 then
begin
     inc(X[i]);Yes:=true
end
     else Yes:=false
```

```
end;
begin
{m va n  qiymatlarini kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, m, n);
      close(fayl);
      for i:=1 to N do X[i]:=1;{boshlang'ich vaziyat}
{chiqarish faylini ochamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
repeat
      for i:=1 to N do
      write(fayl, X[i], ' ');writeln(fayl);
{-natijani chop etamiz}
      Next(X,Yes)
until not Yes;
      close(fayl);
end.
```

To'plam. M to'plami quyidagi shartlar bilan aniqlanadi:
1) 1 soni M ga tegishli;
2) agar k soni M ga tegishli bo'lsa, u holda $2k+1$ va $3k+1$ sonlari ham M to'plamiga tegishli bo'ladi.

Ushbu M to'plamining N ta ($N<1000$) sonini o'sish tartibi bilan chiqaring.

Masala yechimi. Ushbu to'plamdagi sonlarni ko'rib chiqamiz

Boshlang'ich M dagi son	Hosil bo'ladigan sonlar
1	3 , 4
3	7 , 10
4	9 , 13
7	15 , 22
...	...

Demak M dagi sonlar bular - 1, 3, 4, 7, 10, 9, 13 ...

Ko'rinib turibdiki o'rtada o'sish tartibi buzilyapti. Demak to'g'ridan-to'g'ri masala shartini bajarish mumkin emas ekan. Hosil

qilinadigan sonlarni o'sish tartibi bilan $A[1: N]$ massivida saqlab qo'yamiz, ya'ni

$A[1]<A[2]<...<A[i]$ (1)

bo'ladi. Ushbu sonlardan 3 ta eng kichik indeks sonlari $k2$ va $k3$ uchun $A2=2A[k2]+1$ va $A3=3A[k3]+1$

sonlari (1) to'plamga tegishli bo'lmasin. Hosil bo'lgan $A2$ va $A3$ larni to'plamga (1) quyidagicha qo'shib qo'yamiz:

Agar $A2 < A3$ bo'lsa , u holda $A[i+1]=A2$ va $k2= k2+1$, aks holda, ya'ni bunda $A3<A2$ bo 'ladi, $A[i+1]=A3$ va $k3= k3+1$.

Agarda $A=A2$ bo'lsa, u holda $A[i+1]=A2$ va $k2= k2+1$, $k3= k3+1$.

O'z-o'zidan ko'rinib turibdiki boshlang'ich qadamda, ya'ni $i=1$ bo'lganda $A[1]=1$, $k2=1$ va $k3=1$ bo'ladi. Shundan foydalanib algoritmni bajarilishiga e'tibor beraylik

			A massiv elementlari	
i	*k2*	*k3*	boshlang'ich	qo'shiladigan
1	1	1	1	3
2	2	1	1, 3	4
3	2	2	1, 3, 4	7
4	3	2	1, 3, 4, 7	9
5	4	2	1, 3, 4, 7, 9	10
6	4	3	1, 3, 4, 7, 9, 10	
...				

Albatta, M to'plamdagi sonlarni A massiviga joylashtirib, keyinchalik tartib bilan saralasak bo'lardi, lekin-da bu ko'p vaqtni talab qilar edi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
7	1 3 4 7 9 10 13

Bu yerda *input.txt* faylida birinchi qatorda $n=7$ soni beriladi va *output.txt* faylida n ta to'plam elementlari chiqarilgan bo'ladi.

Natijada quyidagi dasturni taklif qilish mumkin:

```
program Toplam;
const Nmax=100;
var
      i,k2,k3,a2,a3,n : integer;
      A:array[1..Nmax] of integer;
      fayl: text;
begin
{n  qiymatini kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      close(fayl);
      k2:=1;  k3:=1;  A[1]:=1;
{chiqarish faylini ochopamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      writeln (fayl, 1);
      for i:=2 to n do
 begin
      a2:=2*A[k2]+1;
      a3:=3*A[k3]+1;
      if a2<=a3  then
 begin
      A[i]:=a2;
      k2:= k2+1
 end;
      if a3<=a2  then
 begin
      A[i]:=a3;
      k3:= k3+1
 end;
      writeln (fayl, A[i])
 end;
      close(fayl);
 end.
```

8.3. Variantlarni saralash usullari

Faqatgina dasturlash tillarini o'rganish bevosita dasturlashtirishni o'rganishga olib kelmaydi. Bu yerda maxsus algoritmlarni yaratish uchun talabalarimizning fikrlash qobiliyatini tubdan o'zgartirish kerak bo'ladi. Buning uchun esa dasturlashga doir masalalarni yechish uchun unga doir usullarni chuqur o'rganish talab etiladi. Variantlarni usullari xuddi shunday usullardan biri hisoblanadi, ularni chuqur o'rganib olgan talabalarimiz keyinchalik mustaqil ravishda dasturlar yaratish imkoniga ega bo'lishishadi.

Qo'yilgan masala doirasida uning yechimini aniqlash, bevosita barcha variantlarni birin-ketin ko'rib chiqish bilan bog'liq bo'ladi. Demak biz variantlarni saralab masala shartini qanoatlantiruvchi variantlarni aniqlash imkoniga ega bo'lamiz. Bu yerda quyidagi to'plam tushunchasini kiritamiz. Variantni bevosita ko'rilayotgan masalaning qandaydir holati deb tushunsak, ushbu holatlarni jamlanmasini to'plam deb qabul qilamiz va uni X deb nomlaymiz. X to'plamiga joylashtirilgan variantlarni uning elementlari sifatida qaraladi. Demak, variantlarni saralab masala yechimini topish – bu X to'plamidagi har bir element x uchun $K(x)$ sharti bo'yicha tekshirib ko'rilishiga aytiladi.

Agar X to'plamidagi elementlar soni $|X|$ bo'lsa va $K(x)$ shartini tekshirish uchun $|K|$ ta qadam zarur bo'lsa, demak masalani yechish uchun $|X|\cdot|K|$ ta qadam kerak bo'ladi.

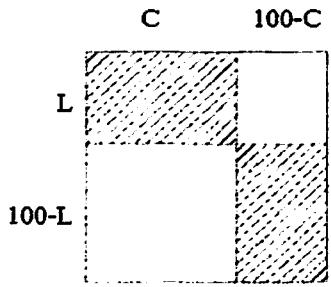
Ko'p hollarda X to'plamining elementi o'z navbatida ko'p parametrli $X_1,...,X_n$ (ya'ni har bir element $x=(x_1,...,x_n)$ ko'rinishda bo'lib, unda $x_i \square X_i$). Bu yerdan qiziqarli natija olish mumkin, ya'ni agarda elementlar sonini 2 barobarga qisqartirsak, u holda variantlar soni 2^n martaga kamayadi.

Albatta, variantlarni kamaytirishning yagona algoritmi mavjud emas. Shu bois aniq misol doirasida variantlar sonini kamaytirishni ko'rib chiqamiz.

Shaxmat taxtasi. Shaxmat taxtasi 100×100 xonadan iborat. Undagi barcha 10000 xonalar qora donalar bilan band. Har bir harakatda qator yoki ustun bo'yicha barcha donalar qoradan oqga va oqdan qoraga almashtiriladi. Cheklangan qadamlardan so'ng taxtada 1990 ta oq donalar paydo bo'lishini aniqlang.

Masala yechimi. Masalaning shartidan kelib chiqqan holda quyidagi xulosalarni keltirish mumkin:

1. Agar biror-bir qatorda operatsiya 2 marta bajarilsa vaziyat o'zgarmaydi.
2. Operatsiyalar bajarilish ketma-ketligi ahamiyatsizdir.
3. Qatorda $2n+k$ ta operatsiya bajarilishi bevosita k ta operatsiya bajarilishi bilan bir xil vaziyatga olib keladi.
4. Masalaning sharti bo'yicha oq donalar qayerda joylashishi ahamiyatsiz.
5. Oxirgi 4-banddan kelib chiqqan holda, qator va ustunlarni o'rin almashtirish orqali shaxmat taxtasini quyidagi ko'rinishga olib kelamiz:



Natijada faqatgina (L,C) juftliklarni ko'rib chiqish kifoya bo'ladi, bu esa variantlar xajmini $100 \cdot 100=10000$ taga olib keladi.

6. Agar (L,C) juftlik yechim bo'lsa, u holda (C,L) ham yechim bo'ladi, ya'ni simmetriyalik mavjud.
7. Bundan tashqari, agar (L,C) juftlik yechim bo'lsa, u holda $(100-C, 100-L)$ ham yechim bo'ladi. Natijada L va C larni quyidagi tengsizliklar bilan bog'lash mumkin $L \leq C$ va $L \leq 100-C$. Demak variantlar soni 2500 tadan kam bo'ladi.

Shunday qilib variantlar sonini biz 10000 dan 2500 tagacha kamaytirdik. Lekin, eng qiziqarligi , masalani yana yengillashtirish mumkin ekan.

Oq donalar soni yuqorida $L \cdot (100-C)$ ta va pastda $C \cdot (100-L)$ ta bo'ladi. Ularning yig'indisi $100(L+C)-2LC=1990$ natijani berishi kerak.

Bu yerdan $C=(1990-100L)/(100-2L)$, demak L ni 50 gacha o'zgartirib biz C ning qiymatini topishimiz mumkin bo'ladi. Lekin butun sonlarda yechimini qidirsak, undan ham qiziqarli yechimga ega bo'lamiz. Buning uchun l va c o'zgaruvchilarini quyidagicha kiritib $L=l+50$ va $C=c+50$ tenglamani soddalashtiramiz, ya'ni $l \cdot c=1505$. Demak 1505 sonini ko'paytuvchilarga ajratib, masalan $1505=35 \cdot 43$, yechim mavjudligini aniqlash mumkin. Bu yerdan esa $l=35$, $c=43$ bo'ladi , demak $L=85$, $C=93$ bo'ladi. (bu yerdan simmetrik yechimni ham taklif qilish mumkin , ya'ni $L=15$, $C=7$).

Shunday qilib, shaxmat taxtasida 1990 ta oq donalar paydo qilish mumkin ekan. E'tibor bering, masala yechimini aniqlashda barcha variantlarni ko'rib chiqish shart bo'lmay qoldi.

8.4. Variantlar sonini qisqartirish yo'llari

Yuqoridagi misolda biz masala yechimini aniqlashda barcha variantlarni ko'rib chiqishimiz shart bo'lmay qoldi, ya'ni masalamiz oddiy ko'rinishga keltirildi. Shu bois uni kombinatorikaga doir masala deb aytaolmaymiz. Umumiy holda kombinatorika masalarida barcha variantlarni birin-ketin ko'rib chiqishga majbur bo'lamiz. Bu yerda asosiy muammo bu - variantlar sonini qisqartirish hisoblanadi. Shunga doir algoritmlar bilan tanishib chiqamiz.

8.4.1. Keraksiz variantlarni e'tibordan chiqarish

Ushbu algoritm bo'yicha yechim bo'lmagan variantlarni e'tibordan chiqarish orqali holatlar to'plamini kamaytirishga olib kelish zarur bo'ladi. Keraksiz variantlarning paydo bo'lish sababi bu variantlarni saralash jarayeni "noto'g'ri" tashkil etilganidadir. Bu yerda asosan oddiy yo'l tanlanib, algoritmda sikl operatorlaridan keng foydalaniladi. Masalani qo'yilishidan kelib chiqqan holda undagi ichki vaziyatlar o'rganilib, holatlar to'plamini kamaytirish mumkin bo'ladi. Endi ba'zi variantlarni qisqartirishni farzin to'g'risidagi misolda ko'rib chiqamiz.

***N* ta farzinni joylashtirish.** $n \times n$ o'lchamli shaxmat taxtasida n farzinni joylashtiring va ular bir-birini urishmasin.

Masala yechimi. Har qanday farzin joylashuvini *(x, y)* deb olamiz. Agar $x_i \neq x_j$ $(y_i \neq y_j)$ bo'lsa, demak i va j farzinlar bitta vertikalda (gorizontalda) bo'lishmaydi.

Agar $x_i+y_i \neq x_j+y_j$ va $x_i-y_i \neq x_j-y_j$ shartlari bajarilsa, u holda i va j farzinlar bitta diagonalda bo'lishmaydi. Shunday qilib, quyidagi masalaga kelamiz: Barcha $i, j \in \{1,...,n\}$ lar uchun $x_i \neq x_j$, $y_i \neq y_j$, $x_i+y_i \neq x_j+y_j$, $x_i-y_i \neq x_j-y_j$ shartlari bajariladigan $(x_1,y_1,x_2,y_2,...,x_n,y_n)$ to'plamni aniqlang. Agar farzinlarni nomerini x koordinatasi deb olsak, masala quyidagi ko'rinishda bo'ladi $(y_1,y_2,...,y_n)$, $y_i \in \{1,...,n\}$, bu yerda $y_i \neq y_j$, $i+y_i \neq j+y_j$, $i-y_i \neq j-y_j$ barcha $i, j \in \{1,...,n\}$ lar uchun.

Birinchi navbatda, masalaning shartini to'g'ridan-to'g'ri bajarishga harakat qilamiz, ya'ni birin-ketin variantlarni saralab chiqamiz. Albatta bu yerda barcha variantlarni ko'rib chiqqan bo'lamiz. Demak, umumiy holda oldiniga birinchi farzinni joylashtirish uchun variantlarni saralaymiz, har bir farzinning holati uchun keyingi farzinni joylashtirish uchun variantlarni saralaymiz va h.k. Ushbu jarayonni n ta sikl yordamida amalga oshirishda quyidagi algoritmni keltirish mumkin:

```
for i1 := 1 to n do
begin
  ferz[1] := i1;
   for i2 := 1 to n do
      begin
     ferz[2] := i2;
      for i3 := 1 to n do
      begin
       ferz[3] := i3;
       ...
       ... <tuzilgan variantni tekshiramiz>
      end;
   end;
end;
```

Ushbu algoritm oddiy bo'lib, juda ko'p vaqtni talab qiladi. Lekin biz barcha kerakli, ya'ni masala shartini qanoatlantiruvchi vaziyatlarni ajratib, keraksizlarini oldindan kesib tashlasimiz mumkin. Haqiqatan ham, farzinni 1-xonaga qo'ygandan so'ng , ikkinchi ustunda farzinni qo'yishda barcha xonalarni ko'rib chiqish shart emas, masalan, 1- va 2-xonalarda farzinni qo'yish mumkin emas. Shu bois $2 \cdot n^{n-2}$ ta variantni

umuman tahlil qilish kerak emas. Shundan kelib chiqqan holda quyidagi algoritmni tavsiya qilsa bo'ladi

```
for i1 := 1 to n do
begin
 ferz[1] := i1;
 for i2 := 1 to n do
 begin
  ferz[2] := i2;
  { tuzilgan variantni tekshiramiz }
  if <ferz[2] bevisita ferz[1] ni urmaydi> then
   for i3 := 1 to n do
   begin
    ferz[3] := i3;
    { tuzilgan variantni tekshiramiz }
    if <ferz[3] oldingilarini urmaydi> then
     for i4 := 1 to n do
          begin
          ...
       if <ferz[n] oldingilarini urmaydi > then
            <yechim topildi>
     end;
   end;
 end;
end;
```

Algoritmni umumiy ko'rinishda yozishimizdan asosiy maqsad, uni boshqa masalalarda ham qo'llash mumkinligida. Bu yerda keltirilgan algoritm variantlarni saralashdagi qaytish usuli deb ataladi. Faqatgina bu yerda biz ushbu qaytish algoritmini sikllar yordamida amalga oshirdik. Albatta algoritmni tuzishda variantlarni saralashdagi holatlarni kamaytirish uning umumiy ishlash vaqtini va uni tushunarsiz ko'rinishga olib kelishini oldini olish kerak.

8.4.2. Simmetriyadan foydalanish

Variantlar sonini kamaytirish yo'llaridan biri – bu simmetriyadan foydalanishdir. Masalan, yuqoridagi misolda farzinni (1,1) xonadagi yechimi bilan $(n,1)$ xonadagi yechimi simmetriyalik xususiyatga ega bo'ladi. Faqatgina shu orqali variantlar sonini 2 martaga kamaytirish mumkin bo'ladi.

8.4.3. Elementlarni guruhlash

Variantlarni ko'rib chiqayotganda ko'p operatsiyalar takrorlanadi. Shu bois ushbu operatsiyalarni guruhlangan elementlar uchun bajarish orqali variantlarni qisqartirish mumkin bo'ladi. Yuqoridagi misolda buni *ferz*[1] := *i1*; ... operatorlari orqali amalga oshirilgan edi.

8.5. Variantlarni saralashda rekursiyadan foydalanish

Yuqorida keltirilgan algoritmni rekursiya ko'rinishida yozilishi qulayroq hisoblanadi. Chunki umumiy holda sikllar soni farzinlar soniga teng va 100 ta farzin uchun sikllarni ichma-ich yozib, faqatgina shu songa moslashtirib olishimiz mumkin. Variantlarni saralashdagi qaytish usulida qo'yiladigan qadamlar bir oldinga va bir orqaga qo'yiladi. Ushbu o'tishlarsiz ham algoritmni tuzish mumkin va buni rekursiya orqali amalga oshirish mumkin. Barcha variantlarni to'liq ko'rib chiqadigan dastur algoritmining umumiy ko'rinishi quyidagicha bo'ladi.

```
procedure Variantlarni_toliq_saralash (m : integer);
var i : integer;
begin
   if m > n then
     <tuzilgan holatni tekshirish>
   else
     for i := 1 to n do
     begin
      ferz[m] := i;
      Variantlarni_toliq_saralash (m+1);
     end;
end;
```

Bu yerda algoritmni qulayroq tasvirlashda uni qism dastur shaklida keltiryapmiz. Qiymatlarni kiritish va chiqarish operatorlarini keltirib o'tirmaymiz.

Variantlarni saralashdagi qaytish usulida rekursiyani kiritish esa quyidagi qism dasturda keltirilgan. Ko'pgina masalalarda ushbu algoritmni qo'llash mumkinligi sababli, uning umumiy ko'rinishini keltiramiz.

```
procedure Variantlarni_qaytib_saralash (m : integer);
var i : integer;
```

```
begin
   if m > n  then
    <yechim topildi>
   else
     for i := 1  to n  do
     begin
       ferz[m] := i;
       if <ferz[m] oldingilarini urmasa>  then
       Variantlarni_qaytib_saralash (m+1);
     end;
end;
```

Bu ikki algoritmlarning farqi quyidagicha: birinchi algoritmda tekshiruv eng oxirgi holat tuzilganda tekshirilsa, ikkinchi algoritmda esa har bir qadamda qisman tekshirish amalga oshiriladi. Natijada qaytish usulida ko'pgina variantlarni to'liq saralashga nisbatan qisqartirish mumkin bo'ladi.

Paskaldagi to'liq dasturi esa quyidagicha bo'ladi:

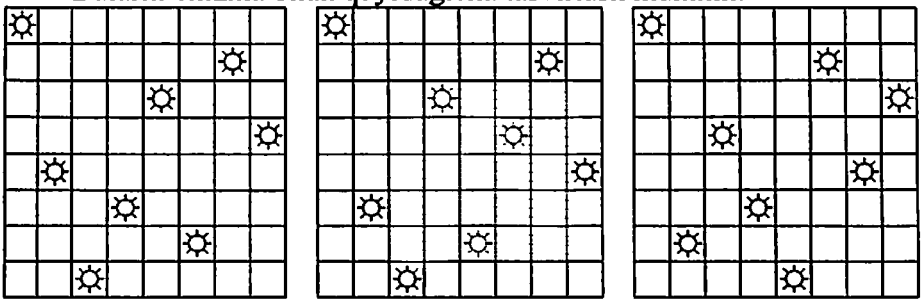
```
{n ta farzin uchun Variantlarni qaytib saralash}
program BacktrackQueens;
var
      n : integer;
      faylin, faylout : text;
      queen : array[1..100] of integer;
{-farzinlar joylashuvini saqlovchi massiv}
{ Natijani chop etish }
procedure WriteSolution;
var
      i : integer;
begin
      for i := 1  to n  do
      Write(faylout, queen[i],' ');
      WriteLn(faylout);
end;
{  m-farzinni oldingilari bilan tekshirish funksiyasi }
function Check(m : integer) : boolean;
var
      i : integer;
begin
      for i := 1  to m-1  do
```

```
      if (queen[i] = queen[m])
{- bitta gorizontalda joylashgan}
Or (i+queen[i] = m+queen[m]) { chap diagonalda }
Or (i-queen[i] = m-queen[m]) { o'ng diagonalda }  then
begin
      Check := False; Exit;
end;
      Check := True;
end;
{ Variantlarni saralash qism dasturi }
procedure Backtrack(m : integer);
var
      i : integer;
begin
      if m > n  then { yechim topildi }
      WriteSolution
      else
      for i := 1  to n  do
begin
      queen[m] := i;
      if Check(m) then
      {- m-farzin oldingilari bilan kesishmaydi }
      Backtrack(m+1);
end;
end;
begin
{n  qiymatini kiritamiz}
      assign(faylin, 'input.txt');
      reset(faylin);
      read(faylin, n);{ Farzinlar soni}
      close(faylin);
{chiqarish faylini ochopamiz}
      assign(faylout, 'output.txt');
      rewrite(faylout);
      Backtrack(1);
      close(faylout);
end.
```

Bu erda n=8 bo'lganda jami variantlar soni 92 ta bo'ladi, birinchi 3 ta natija sifatida quyidagilar chiqariladi:

1 5 8 6 3 7 2 4
1 6 8 3 7 4 2 5
1 7 4 6 8 2 5 3

Bularni chizma bilan quyidagicha tasvirlash mumkin:



Bu yerda, simmetrik variantlar olib tashlansa, u holda faqatgina 12 ta variant qoladi.

8.6. Variantlar tartibini o'zgartirish algoritmi

Ushbu algoritmni bevosita farzin masalasi bo'yicha n=8 bo'lganda ko'rib chiqamiz. Birinchi uchta farzin joylashtirilganda holat quyidagicha bo'ladi

					3	2	1
				3	2	1	
			3	2	1		
		3	2	1	3	3	3
		2	1				
	2	1	2	2	2	2	2
	1	2			3		
1	1	1	1	1	1	1	1
1	2	3	4	5	6	7	8

Agar shu yo'l bilan davom ettirsak, biz to'rtinchi, beshinchi va oltinchi farzinlarni birin-ketin qo'yib chiqishimiz kerak. Lekin e'tibor

bering oltinchi ustunda faqatgina bitta xona bo‘sh qolgan. Demak to‘rtinchi farzinni shu yerdan joylashtirsak, natijada quyidagiga kelamiz

					3	2	1
				3	2	1	
			3	2	1		4
		3	2	1	3	3	3
		2	1	4	4	4	4
	2	1	2	2	2	2	2
	1	2	4		3		4
1	1	1	1	1	1	1	1
1	2	3	4	5	6	7	8

E’tibor bering, sakkizinchi ustunda keyingi farzinni qo‘yishga bitta xona qolgan. Shu yondshuv bilan oltinchi va yettinchi farzinlar joylashtirilsa natija quyidagicha bo‘ladi:

			6	6	3	2	1
			5	3	2	1	5
			3	2	1	5	4
		3	2	1	3	3	3
		2	1	4	4	4	4
	2	1	2	2	2	2	2
	1	2	4	7	3	7	4
1	1	1	1	1	1	1	1
1	2	3	4	5	6	7	8

E’tibor bering, sakkizinchi farzinni qo‘yishga joy qolmagan, ushbu holni oldindan tekshirib, orqaga qaytish mumkin bo‘ladi.

Variantlar tartibini o‘zgartirishni optimallashtirish maqsadida quyidagi algoritmni taklif qilish mumkin. Holatlar fazosidan $X_k \times \ldots \times X_l$ shunday i ni tanlash kerakki, u uchun X_i eng kichik o‘lchamga (ya’ni eng kam elementlar soniga) ega bo‘lsin. Natijada, agar elementlar soni nol bo‘lsa, demak algoritmni to‘xtatib, orqaga qaytib yangi holatlarni yaratamiz. Shu bilan biz har bir qadamda variantlar sonini keskin kamaytirib boramiz va natijada umumiy variantlar soni ham kamayadi.

```
procedure Oldinga_qarash(m : integer);
var i, qm : integer;
begin
    if m > n  then
      <yechim topildi>
    else
begin
{ eng kam variantlarga ega bo'lgan  farzinni tanlaymiz }
  minq := n+1;
  for i := m  to n  do
  if variantlar_soni [qadamda [i]] < minq  then
begin
  l := i; qm := qadamda [l];
  minq := variantlar_soni[qm];
end;
  qadamda [l] := qadamda [m]; qadamda [m] := qm;
  for i := 1  to n  do
  if shaxmat_taxtasi[qm,i] = 0  then
begin
  ferz[qm] := i;
  holatlar_fazosini_qisqartirish(qm,i,m);
  Oldinga_qarash (m+1);
  holatlar_fazosini_tiklash(qm,i,m);
end;
end;
end;
```

Ushbu algoritm "oldinga qarash" (**forward checking**) deb aytiladi. Biz uni faqat psevdo koddagi dasturini keltirdik. Uni dolzarbligini inobatga olib undagi o'zgaruvchilarga izoh berib o'tamiz. Qo'yilmagan va hozirda ko'rilayotgan farzinlarni "*qadamda*" massivida saqlaymiz. Qo'yiladigan qadamlarni *m* o'zgaruvchida saqlaymiz, *qm* o'zgaruvchida esa ko'rilayotgan farzinning tartib raqamini saqlaymiz. Bu erda "*holatlar_fazosini_qisqartirish*" va "*holatlar_fazosini_tiklash*" qism dasturlari shaxmat taxtasidagi o'zgarishlardan tashqari, har bir farzin uchun "*variantlar_soni*" massivini ham to'ldirishi kerak bo'ladi.

Tekshiruvlar, ushbu algoritmni bajarish uchun sarflanadigan vaqt juda ozligini ko'rsatadi.

Paskaldagi to'liq dasturi esa quyidagicha bo'ladi:

```
{n ta  farzin uchun variantlarni kamaytirish va } {"oldinga_qarash"
algoritmi asosidagi dasturi  }
program PropagateQueens;
var
      n : integer; { taxtaning o'lchami }
      queen : array[1..160] of integer;
      {-farzinlar joylashuvini saqlovchi massiv}
      space : array[1..160,1..160] of integer;
      {-variantlar fazosi}
      step : array[1..160] of integer;
      {- farzinlarni tanlash tartibi }
      cases : array[1..160] of integer;{variantlar soni}
      faylin, faylout : text;
{ Natijani chop etish }
procedure WriteSolution;
var
      i : integer;
begin
      for i := 1  to n  do
      Write(faylout, queen[i],' ');
      WriteLn(faylout);
end;
{ "holatlar_fazosini_qisqartirish" qism dasturi }
procedure Prune(qm,queen,m : integer);
      var i, nq : integer;
procedure ExcludeField(queen : integer);
begin
      if (queen >= 1) and (queen <= n)  then
      if space[nq,queen] = 0  then begin
      Dec(cases[nq]); space[nq,queen] := m;
end;
end;
begin
      for i := m+1  to n  do
begin
      nq := step[i];
      ExcludeField(queen);
{- bitta gorizontalda joylashgan  }
```

```
      ExcludeField(queen+nq-qm);
{- bitta diagonalda joylashgan }
      ExcludeField(queen-nq+qm);
end;
end;
{ "holatlar_fazosini_tiklash" qism dasturi }
procedure Restore(qm,queen,m : integer);
var i, nq : integer;
procedure IncludeField(queen : integer);
begin
      if (queen >= 1) and (queen <= n)  then
      if space[nq,queen] = m  then
begin
      Inc(cases[nq]); space[nq,queen] := 0;
end;
end;
begin
      for i := m+1  to n  do
begin
      nq := step[i];
      IncludeField(queen);
{- bitta gorizontalda joylashgan  }
      IncludeField(queen+nq-qm);
{ -bitta diagonalda joylashgan }
      IncludeField(queen-nq+qm);
end;
end;
procedure Propagate(m : integer);
var i, qm, minq, l : integer;
begin
      if m > n  then
      WriteSolution
      else
begin
{eng kichik variantlar soniga ega bo'lgan}
{farzin tanlanadi}
      minq := n+1;
      for i := m  to n  do
      if cases[step[i]] <= minq  then
```

```
begin
      l := i; qm := step[l]; minq := cases[qm];
end;
      step[l] := step[m]; step[m] := qm;
      for i := 1  to n  do
      if space[qm,i] = 0  then
begin
      queen[qm] := i;
      Prune(qm,i,m);
      Propagate(m+1);
      Restore(qm,i,m);
end;
end;
end;
{Asosiy dastur}
      var i, j : integer;
begin
{n  qiymatini kiritamiz}
      assign(faylin, 'input.txt');
      reset(faylin);
      read(faylin, n);{ Farzinlar soni}
      close(faylin);
{chiqarish faylini ochopamiz}
      assign(faylout, 'output.txt');
      rewrite(faylout);
      for i := 1  to n  do
begin
      for j := 1  to n  do
      space[i,j] := 0;
      step[i] := i; cases[i] := n;
end;
      Propagate(1);
      close(faylout);
end.
```

Bu yerda ham n=8 bo'lganda jami variantlar soni 92 ta bo'ladi va birinchi 3 ta natija sifatida quyidagilar chiqariladi:

3 5 2 8 6 4 7 1
3 6 4 2 8 5 7 1
4 2 7 3 6 8 5 1

Natural sonning yoyilmasi. Har qanday natural n sonini boshqa natural sonlar yig'indisi bilan yozish mumkin. Barcha takrorlanmaydigan variantlarni aniqlang.

Masala yechimi. Tushunarli bo'lish uchun quyidagi n=4 misolini keltiramiz

4=4
4=3+1
4=2+2
4=2+1+1
4=1+1+1+1

Bu yerda 3+1 va 1+3 bitta variant deb qabul qilingan, aks holda takrorlanish paydo bo'ladi. Demak, masalaning sharti bo'yicha variantlar takrorlanmasligi zarur, buning uchun har qanday n uchun

$n=m_1+m_2+...$

Yoyilmada $m_1 \geq m_2 \geq ....$ deb qabul qilamiz. Har qanday ikki yoyilmadan

$n = m_1^{(1)} + m_2^{(1)} + ...$

$n = m_1^{(2)} + m_2^{(2)} + ...$

Biri oldin, biri keyin keladi, agarda barcha j uchun , agar $j<i$ bo'lsa va $m_j^{(1)} = m_j^{(1)}$ va $m_j^{(1)} > m_j^{(1)}$ bo'lsa, birinchi yoyilma oldin keladi.

Yuqorida keltirilgan n=4 misoli xuddi shunday tarzdagi yoyilmalari keltirilgan.

Har qanday yoyilmadan

$n=m_1+m_2+...+m_i$ (*)

yangisini tuzish quyidagicha amalga oshiriladi. Keltirilgan (*) yoyilmaning oxirgi m_i hadidan boshlab m_i, m_{i-1}, ...m_k larni ko'rib chiqamiz. Bu yerda m_k eng birinchi $m_k>1$ bo'lgan son. Agar bunday sonlar topilmasa, demak bu oxirgi variant bo'ladi. Demak bizda $m_k>1$ bo'lsin, biz m_k ning qiymatini 1 kamaytiramiz va m_k gacha bo'lgan birlarni yo'qotamiz. Natijada n qiymati $s=1+i-k$ ga kamayadi. Endi yangi m_j $(j=k+1,k+2,...,i)$ larning qiymatlarini aniqlash kerak. Agar $s>m_k$ bo'lsa , u holda $m_j = m_k$ deb qabul qilamiz va s ning qiymatini m_k ga kamaytiramiz. Agar $s \leq m_k$ bo'lsa, $m_k = s$ deb olamiz va shu bilan yangi variant tuzildi deb qabul qilamiz.

n=4 misolida ushbu algoritmni qadamba-qadam ko'rib chiqamiz.

1. n=4, demak bu yerda i=1, m_1=4

2. Oxiridan kelsak, k=1 bo'lganda m_1>1 bo'lganligi sababli uni 1 ga kamaytiramiz, ya'ni m_1=3 , bundan s=1+i-k =1+1-1=1.
$s \le m1$ bo'lganligi sababli m_2=s, ya'ni m_2=1 , shunday qilib ikkinchi variant
hosil bo'ldi:
4=3+1
Bu yerda i=2, m_1=3 va m_2=1.
3. Oxiridan kelsak, k=1 bo'lganda m_1>1 demak uni 1 ga kamaytiramiz, ya'ni m_1=2 . bundan s=1+i-k =1+2-1=2.
$s \le m_1$, ya'ni $2 \le 2$ bo'lganligi sababli m_2=s, ya'ni m_2=2 , shunday qilib uchinchi variant hosil bo'ldi:
4=2+2
Bu yerda i=2, m_1=2 va m_2=2.
4. Oxiridan kelsak, k=2 bo'lganda m_2>1 demak uni 1 ga kamaytiramiz, ya'ni m_2=1 . bundan s=1+i-k =1+2-2=1.
$s \le m_2$, ya'ni $1 \le 1$ bo'lganligi sababli m_3=s, ya'ni m_3=1 , shunday qilib yangi variant hosil bo'ldi:
4=2+1+1
Bu yerda i=3, m_1=2 , m_2=1 va m_3=1.
5. Oxiridan kelsak, k=1 bo'lganda m_1>1, demak uni 1 ga kamaytiramiz, ya'ni m_1=1 . bundan s=1+i-k =1+3-1=3.
$s > m_1$ bo'lganligi sababli keyingi m_2=m_1=1, va s=3-m_1=3-1=2. bu yerda s yana katta m_1 dan , ya'ni 2>1 , demak keyingi m_3=m_1=1 va s=s-m_1=2-1=1.
Bu yerda $s \le m_1$ shu bois keyingi m_4=s , ya'ni m_4=1 shu bilan jarayon tugatildi va quyidagi hosil bo'ldi
4=1+1+1+1
Demak barcha yoyilmalar aniqlandi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
4	4 3 1 2 2 2 1 1 1 1 1 1

Bu yerda *input.txt* faylida birinchi qatorda n=4 soni beriladi va *output.txt* faylida barcha yoyilmalar chiqarilgan bo'ladi.

Dastur esa quyidagicha bo'ladi:

```
program Yoyilma;
const Nmax=100;
label RA, BR;
var
      n,i,t,k,s : integer;
      M:array[1..Nmax] of integer;
      fayl: text;
begin
{n qiymatini kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      close(fayl);
{chiqarish faylini ochamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      writeln (fayl, n);
      M[1]:=n; k:=1; i:=1;
RA:
      t:= M[k]-1; s:=t+i-k+1;
      for i:=k to n do
begin
      if s>t then
begin
      M[i]:=t; s:=s-t
end
      else
begin
      M[i]:=s; goto BR
end;
end;
BR:
      for k:=1 to i do write (fayl, M[k], ' ');
      writeln(fayl);
      for k:=i downto 1 do if M[k]>1 then goto RA;
      close(fayl);
```

```
end.
```

Agarda shartsiz o‘tish operatoridan voz kechmoqchi bo‘lsak, quyidagi algoritmni keltirsak bo‘ladi:

```
program Yoyilma_New;
type Razb=array [byte] of byte;
var
      N,i,L:byte;
      X:Razb;
      fayl: text;
procedure Next(var X:Razb;var L:byte);
      var i,j:byte;
      s:word;
begin
      i:=L-1;s:=X[L];
{ i ni qidiramiz}
while (i>1)and(X[i-1]<=X[i]) do
begin
      s:=s+X[i];dec(i)
end;
      inc(X[i]);
      L:=i+s-1; { yoyilmadagi qo‘shiluvchilar soni }
      for j:=i+1 to L do X[j]:=1
end;
begin
{n  qiymatini kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      close(fayl);
      L:=N; for i:=1 to L do X[i]:=1;
{chiqarish faylini ochamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i:=1 to L do
      write(fayl, X[i], ' ');
      writeln(fayl);
repeat
```

```
    Next(X,L);
    for i:=1 to L do
    write(fayl, X[i], ' ');
    writeln(fayl)
until L=1;
    close(fayl);
end.
```

E'tibor bering, natijalar bu yerda eng kichik sonlardan boshlanadi:

1 1 1 1
2 1 1
2 2
3 1
4

Raqamlar yig'indisi. Uch xonali sonning raqamlar yig'indisi berilgan N soniga teng bo'lganlarini toping.

Masala yechimi. Ushbu masala oddiyligi bilan va bajariladigan amallarni sonini qanday qilib kamaytirish mumkinligi bilan diqqatga sazovordir. Har qanday 3 xonali sonni quyidagicha yozish mumkin $\overline{kji} = x$, bu yerda $1 \le k \le 9$, $0 \le j \le 9$ va $0 \le i \le 9$.

Masalaning sharti bo'yicha $N=k+j+i$ tenglik bajarilsa X sonini chop etish talab etiladi. Oddiyroq yondashuv bo'yicha, albatta, biz har bir raqamni k, j, i larning ko'rsatilgan intervalda o'zgartirib, barcha sonlarni tashkil qilishimiz mumkin bo'ladi. Buning uchun hammasi bo'lib 900 ta sikl bajariladi. Lekin j,i bo'yicha sikl tuzib, k ni quyidagicha aniqlashamiz mumkin $k= N-j-i$ va faqatgina k ning chegarasi $1 \le k \le 9$ tekshirilsa kifoya. Demak jami bo'lib 100 ta sikl bajariladi.

Agar N soni 9 dan kichik bo'lsa, ushbu jarayonni yanada tezlashtirish mumkin bo'ladi.

Dastur esa quyidagi ko'rinishda bo'ladi:

```
program SumRaqam;
var n,i, j,k : integer;
begin
    write ('N=');
    readln(n);
    if n in [1..27] then
    for i:=0 to 9 do
```

```
    for j:=0 to 9 do
begin
    k:=n-i-j;
    if k in [1..9] then  writeln (i+10*j+100*k);
end;
end.
```

Masalan, N=25 bo'lganda quyidagi sonlar chiqariladi:

997 ; 988 ; 898 ; 979 ; 889 ; 799 .

Sonlar yig'indisi. Natural sonlar massivi berilgan $N[k]$ bo'lsa, shunday minimal natural M sonini aniqlash kerakki, uni N dagi hech qanday sonlar yig'indisi shaklida yozish mumkin bo'lmasin.

Yig'indida faqatgina bitta element bo'lishi ham mumkin va yig'indida N massivining elementi faqatgina bir marotaba qo'llanilishi mumkin.

Masala yechimi. To'g'ridan-to'g'ri ushbu masalani yechish algoritmi quyidagicha bo'lishi mumkin:

1. natural son M tanlanadi;
2. N massividan har xil to'plamlar tuziladi;
3. Yig'indi hisoblanib M bilan taqqoslanadi va h.k.

Albatta ushbu algoritm juda ko'p vaqt talab etadi. Shu bois boshqa algoritm tavsiya etiladi. M qiymatini natural sonlar tartibi bo'yicha o'zgartirib boramiz, ya'ni boshida M=1. Agar N massivida biror-bir element $P[j] \le M$ bo'lsa, $M=M+P[j]$ deb qabul qilib, massivdan $P[j]$ ni o'chiramiz va yana yangi $P[j]$ ni qidira boshlaymiz. Agarda shunday element topilmasa, demak M soni masala javobi bo'ladi.

Algoritmni yaxshi tushunib olish uchun misol sifatida quyidagini olamiz:

N : 2, 4, 1, 3, 6

Demak, M=1 bo'lganda j=3 uchun $N[j] \le M$, shu bois $M=M+N[3]=1+1=2$. Ya'ni $N[3]$ ni o'chiramiz

N : 2, 4, ~~1~~, 3, 6

Hozir M=2, bu yerda j=1 uchun $N[j] \le M$, shu bois $M=M+N[1]=2+2=4$. Ya'ni $N[1]$ ni o'chiramiz

N : ~~2~~, 4, ~~1~~, 3, 6

Hozir M=4, bu yerda j=2 uchun $N[j] \le M$, shu bois $M=M+N[2]=4+4=8$. Ya'ni $N[2]$ ni o'chiramiz

N: ~~2~~, ~~4~~, ~~1~~, 3, 6

Hozir M=8, bu yerda j=4 uchun $N[j] \le M$, shu bois $M=M+N[4]=8+3=11$. Ya'ni $N[4]$ ni o'chiramiz

N: ~~2~~, ~~4~~, ~~1~~, ~~3~~, 6

Hozir M=11, bu yerda j=5 uchun $N[j] \le M$, shu bois $M=M+N[5]=11+6=17$. Ya'ni $N[5]$ ni o'chiramiz

N: ~~2~~, ~~4~~, ~~1~~, ~~3~~, ~~6~~

Boshqa elementlar N da qolmadi, demak M=17 eng minimal natural son.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 2 4 1 3 6	17

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami k=5, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida natijaviy qiymat chiqarilgan bo'ladi.

Dasturimiz quyidagi ko'rinishda bo'lishi mumkin:

```
program SumSon;
const Nmax=10;
label PL, WL;
var
    k, i , j, s : integer;
    N:array[1..Nmax] of integer;
    fayl: text;
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
    assign(fayl, 'input.txt');
    reset(fayl);
    read(fayl, k);
    for i:=1 to k do
    read(fayl, n[i]);
{-sikl orqali n[] qiymatlarini kiritamiz}
    close(fayl);
    s:=1;
```

```
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      for i:=1 to k do
begin
      for j:=1 to k do
      if ( N[j]<=s) and (N[j]>0)  then goto PL;
      goto WL;
PL:
      s:=s+ N[j];
      N[j]:= -N[j]; {-manfiy qilib,ro'yxatdan chiqaramiz}
end;
WL:
      writeln (fayl, s);
      close(fayl);
end.
```

Katta-kichik. Musbat sonlardan tashkil topgan $A[n]$ massividan yangi massiv quyidagi yo'l orqali tashkil topgan. Har qanday $A[i]$ elementi, undan keyin turgan massiv elementlaridan $A[i]$ dan kattalari orasidan eng kichik j indeksi uchun $A[i]$ ning qiymati $A[j]$ ga almashtiriladi. Agarda bunday elementlar bo'lmasa $A[i]$ ga nol qiymati beriladi. $A[]$ massivini aniqlang.

Masala yechimi. Masala shartini anglash uchun quyidagi misolni keltiramiz:
$A[]$= (2, 9, 8, 5, 9, 3, 4, 5, 2) dan (9, 0, 9, 9, 0, 4, 5, 0, 0) hosil bo'ladi.

Masala sharti bo'yicha $A[]$ massivini o'ng tomondan tahlil qilishni boshlash kerakligi ko'zga tashlanib turibdi. Undan tashqari yana bitta massiv $B[]$ ni tashkil qilamiz. Ushbu massivda asl $A[i]$ elementlarini saqlab qo'yamiz, chunki ular yo'lma-yo'l o'zgaradi.

Demak ishni $k=n$ dan boshlaymiz, bu yerda, o'ng tomonda hech qanday elementlar bo'lmaganligi sababli: $B[k]=A[n]$ va $A[n]=0$.

Keyingi $A[i]$ elementni qaraydigan paytimizda undan o'ngdagi $A[i+1 \ldots n]$ elementlar allaqachon o'zgartirilgan, lekin ular $B[k:n]$ massivida joylashgan. Shu bois $B[k:n]$ elementlari ko'rib chiqiladi. Undan birinchi uchragan $B[j]>A[i]$ aniqlanadi. Agarda shunday j indeksi mavjud bo'lsa, unda quyidagilarni bajaramiz

$k:=j-1;\ B[k]:=A[i];\ A[i]:=B[j];$

agar j mavjud bo'lmasa, unda

$k:=n;\ B[k]:=A[i];\ A[i]:=0;$

Ushbu algoritmda bajarilayotgan operatsiyalar soni $O(n)$ tartibli bo'ladi. Chunki har bir $A[i]$ elementi uchun u $B[j]$ bilan taqqoslanadi va $A[i]$ ni B ga kiritamiz yoki $B[j]$ ni uchiramiz. Demak , taqqoslashlarini soni $2n$ dan kam bo'ladi va shu bois algoritm $O(n)$ tartibli.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
9 2 9 8 5 9 3 4 5 2	9 0 9 9 0 4 5 0 0

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=9, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida natijaviy massiv qiymatlari chiqarilgan bo'ladi.

Dastur esa quyidagicha bo'ladi.

```
program KattaKichik;
const Nmax=100;
var
      i, j, k, m, n : integer;
      A,B:array[1..Nmax] of integer;
      fayl: text;
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n);
      for i:=1 to n do
      read(fayl, a[i]);
{-sikl orqali a[] qiymatlarini kiritamiz}
      close(fayl);
      B[n]:=A[n]; A[n]:=0;k:=n;
      for i:=n-1 downto 1 do
begin
      j:=k;
while (j<=n) and  (a[i]>=b[j] ) do j:=j+1;
      k:=j-1 ;
```

```
    B[k]:=A[i];
    if j<=n then A[i]:=B[j] else A[i]:=0;
end;
{chiqarish fayliga natijani chop etamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
    for i:=1 to n do write(fayl, A[i], ' ') ;
    close(fayl);
end.
```

Ko‘pgina masalalarda maxsus belgilarni ketma-ket o‘rin almashtirish orqali har xil holatlar keltirib chiqariladi va uning qandaydir qiymatlari hisoblanadi. Unda qo‘llaniladigan algoritmlar umumiy ko‘rinishga ega bo‘ladi. Quyidagi misol ham shu yo‘nalishdagi misollardan biridir.

Qavsli arifmetik ifoda. Quyidagi ifodada

$$((((1?2)?3)?4)?5)?6$$

? belgilari o‘rniga arifmetik amallar belgilarini qo‘yib *M* qiymatini hosil qiling. Bo‘linmalarda faqatgina butun qismi olinadigan bo‘lsa, bitta yechimni topish bilan kifoyalansa bo‘ladi.

Masala yechimi. Berilgan ifodani quyidagicha belgilab olamiz:

$$B=((((1\ a_1\ 2)\ a_2\ 3)\ a_3\ 4)\ a_4\ 5)\ a_5\ 6$$

Bu yerda a_i elementlari arifmetik amallarni anglatadi, masalan, +, -, *, / amallarni 0,1,2,3 deb belgilaymiz. Demak *a*[] massivi elementlari qiymatlari {0;0;0;0;0} dan boshlanib {3;3;3;3;3} gacha davom etadi. Lekin bular biz o‘rgangan 00000 sonidan boshlanib 33333 bilan tugagan jarayon va har bir yangi variantni 1 ni qo‘shish bilan yaratamiz. Faqatgina qo‘shish amalida javob 0 dan 3 gacha bo‘lishi kerak, ya’ni 3+1=10 (ya’ni 1 ko‘ngilda bo‘ladi) , bu esa 4-lik sanoq tizimi ekanligidan dalolat beradi. Ifodani hisoblash quyidagi ketma-ketlikda bajariladi:

$B_1=1,\ B_2=B_1\ a_1\ 2,\ B_3=B_2\ a_2\ 3,\ B_4=B_3\ a_3\ 4,\ B_5=B_4\ a_4\ 5,\ B_6=B_5\ a_5\ 6$

Har bir variantda B_1 , B_2, B_3, B_4 , B_5 larni bir marta hisoblab, faqatgina , B_6 hisoblanadi. Keyinchalik esa a_4 amali o‘zgarganda B_5 ni hisoblab, yana faqatgina B_6 ni hisoblaymiz. Shu usul bilan hisoblash variantlarini kamaytirib, tezroq natijaga erishish mumkin bo‘ladi.

Masaladagi ma'lumotlar quyidagicha chiqariladi:

input.txt	*output.txt*
13	((((1+2)+3)-4)+5)+6 ((((1+2)+3)*4)-5)-6 ((((1+2)*3)*4)%5)+6 ((((1+2)*3)%4)+5)+6 ((((1*2)*3)-4)+5)+6 ((((1*2)*3)*4)-5)-6 ((((1%2)+3)*4)-5)+6 7

Bu yerda *input.txt* faylida faqatgina M=13 qiymati bo'lib, *output.txt* faylida barcha natijaviy arifmetik ifodalar va oxirida ifodalar soni chiqarilgan bo'ladi.

Dasturimiz esa quyidagi ko'rinishda bo'ladi.

```
program Qavs;
const
    N=6;
label R;
var
    i, j, k, m, y : integer;
    A,B : array[1..N] of integer;
    fayl : text;
begin
{ifoda qiymatini kiritamiz}
    assign(fayl, 'input.txt');
    reset(fayl);
    read(fayl, m);
    close(fayl);
{chiqarish faylini ochip qo'yamiz}
    assign(fayl, 'output.txt');
    rewrite(fayl);
    B[1]:=1; k:=0; A[2]:=0;
    for i:=3 to N do A[i]:=4;
R:
    for i:=N downto 2 do
```

```
      if (A[i]=4) then A[i]:=1
      else
begin
      A[i]:= A[i]+1;y:= B[i-1];
      case (A[i]) of
            1: B[i]:=y+i;
            2: B[i]:=y-i;
            3: B[i]:=y*i;
            4: B[i]:=y div i;
      end;
      for j:=i+1 to N do B[j]:= B[j-1]+j;
      if (B[N]<>M) then  goto R; k:=k+1;
      for j:=2 to N-1 do
      write (fayl , '(' );
      write (fayl , '1');
      for j:=2 to N do
begin
      if (j>2) then   write(fayl , ')' );
      case a[j] of
            1: write (fayl , '+');
            2: write (fayl , '-');
            3: write (fayl , '*');
            4: write (fayl , '%');
      end;
      write (fayl , j);
end;
      writeln (fayl); goto R;
      {-boshqa variantlarni ham ko'ramiz}
end;
      writeln (fayl, k);{ ifodalar soni}
      close(fayl);
end.
```

Topshiriqlar

1. Tekis taxtada n ta mix urilgan. Ularning koordinatalari $X[n]$, $Y[n]$ massivlari orqali berilgan. Barcha mixlar o'zaro iplar bilan bog'langan. Bitta mix kamida bitta ip bilan bog'langan. Mixlarni bog'lanishdan hosil bo'ladigan holatidagi iplar uzunligining minimal variantini aniqlang va mixlarni juftliklar bo'yicha bog'lanishini chiqaring.
Izoh. Masalani variantlarni saralab va saralamasdan yeching.
2. Dengizda n ta orol mavjud. Ularning koordinatalari $X[n]$, $Y[n]$ massivlari orqali berilgan. Orollarni o'zaro ko'priklar bilan bog'lash talab etiladi. Har bir ko'prik bir juft orolni o'zaro bog'laydi va istalgan oroldan istalgan orolgacha ushbu ko'priklar orqali o'tish mumkin. Ko'priklar uzunligi minimal bo'lishi uchun nechta ko'prik qurishni aniqlang.
Izoh. Masalani variantlarni saralab va saralamasdan yeching.
3. Do'konga kelgan xaridorda har xil k_1, k_2,..., k_m tangalaridan har biridan ikki donadan bo'lsa, u N summali mahsulotlarni qaytimsiz to'la olishligini aniqlang.
4. 0 va 1 lardan tashkil topgan ketma-ketlikning uzunligi n bo'lsa, ulardan faqatgina k ta birlardan tashkil topganlarini chop eting.
5. Agar natural son N bevosita $m<6$ ta raqamdan tashkil topgan bo'lsa, ya'ni $N=a_1a_2...a_m$. ulardan faqatgina quyidagi tenglikni $N=a_1^m+a_2^m+...+a_m^m$ qanoatlantiruvchi sonlarni chop eting.
6. Barcha tub sonlarni 3, 5, 7, 11, 13, ... birin-ketin yozib quyidagi ketma-ketlikni 23571113171923... tashkil qilamiz. Berilgan N-o'rinda joylashgan raqamni aniqlang.
7. Berilgan N natural soni K asosli sanoq tizimida berilgan bo'lsa, unda ushbu sanoq tizimida $N!$ nechta nol bilan tugallanishini aniqlang.
8. Faqatgina -1, 0 yoki +1 dan tashkil topgan ketma-ketlik x_1, x_2, ..., x_N elementlarining yig'indisi S ga teng bo'lgan variantlar sonini aniqlang.
9. Samarqanddan Jumagacha 2 ta yo'l, Jumadan Buxorogacha esa 4 ta yo'l olib borsa, Samarqanddan Buxorogacha nechta yo'l olib borishini aniqlang.
10. Avtomashinalar uchun davlat nomerlarida uchta xona raqamlar uchun va uchta xona harflar uchun ajratilgan bo'lsa, har xil nomerlar sonini aniqlang.
11. Kimyoviy zavodda ikkita mahsulot A va B ishlab chiqiladi. Ular maxsus qutilarda ketma-ket joylashtiriladi. Birinchi mahsulot juda xavfli, ikkinchisi esa xavfsiz bo'lib, agarda A mahsuloti qutida ketma-

ket kelsa, u holda quti portlashi mumkin. Agarda qutiga N ta mahsulot joylashtirish mumkin bo'lsa, ulardan nechta usul bilan xavfsiz quti tuzish mumkinligini aniqlang.

12. Niqob quyidagi belgilardan {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *a*, *b*, *c*, *d*, *e*, *f*, *g*, ?} tashkil topgan bo'lib, uzunligi $n<9$ bilan chegaranlan. Niqobdan quyidagi qoidalar asosida yangi satrlarni yaratish mumkin:

- raqamlar o'zgartirilmaydi;
- qolgan har bir belgi keltirilgan raqamlarning istalgan biri bilan almashtirilishi mumkin, ya'ni

'a'	0,1, 2, 3
'b'	1, 2, 3, 4
'c'	2, 3, 4, 5
'd'	3, 4, 5, 6
'e'	4, 5, 6, 7
'f'	5, 6, 7, 8
'g'	6, 7, 8, 9
'?'	0,...,9

Ikkita berilgan p_1 va p_2 satrlardan, mos ravishda S_1 va S_2 satrlar to'plami yuqoridagi qoidalar bo'yicha yaratilgan bo'lsa, ikala to'plamda mavjud bo'lgan bir xil satrlar sonini aniqlash talab etiladi.

9-bob. GRAFLARGA DOIR ALGORITMLAR VA DASTURLAR

Bu yerda biz graflarga doir algoritmlar bilan tanishib chiqamiz. Keltirilgan tushunchalar soddalashtirilgan bo'lib, ularni o'rganish talabalarimizda qiyinchilik tug'dirmaydi. Graflar uchun algoritmlarni yaratishni o'rganib olish ko'pgina amaliy masalalarni yechishni imkonini yaratib beradi.

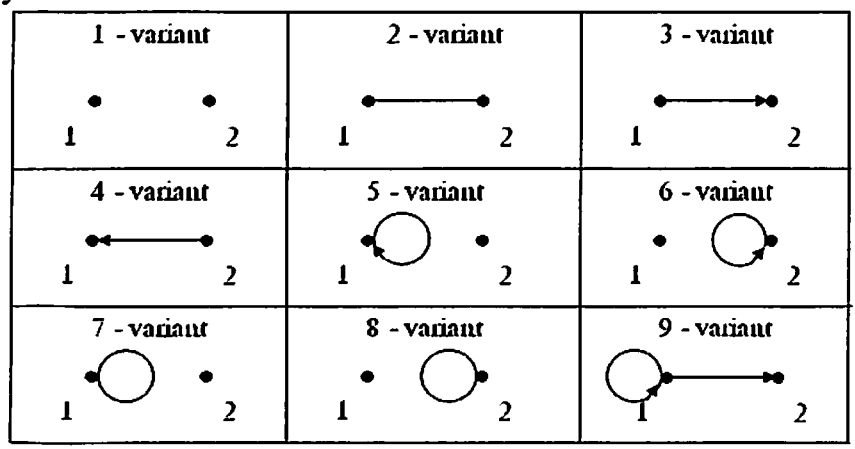
Mazkur bobda graflar va ularda bajariladigan algoritmlar tahlili berilgan bo'lib, unda quyidagi bo'limlar ko'rib chiqilgan:

9.1. Graf tushunchasi

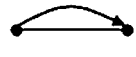
Umumiy holda graf tushunchasi oliy matematikada keng yoritilgan bo'lib, u yerdagi ta'riflarni bu yerda keltirsak, talabalarimizga qiyinchilik tug'dirishi mumkin. Shu bois graf va unga bog'liq boshqa tushunchalarni soddalashtirilgan holda keltiramiz.

Dasturlashtirishda graflar algebraik ifodalarni yechish algoritmlarini tuzishda, saralash va izlash masalalarida va boshqa misollarda qo'llaniladi. Bu yerda biz asosan oddiy graflar haqida so'z yuritamiz. Xo'sh, graf – bu o'zi nima? Oddiy misollardan boshlaymiz. Shaharda binolar va yo'llar mavjud. Har bir yo'lning uzunligi bo'ladi. Ko'pincha *A* nuqtadan *B* nuqtaga yuk tashish kerak bo'ladi, shu bois biz eng qisqa masofani qidirishimiz talab etiladi. Ushbu tuzilma graf deyiladi. Bunda binolar graf uchlari bo'lsa, yo'llar uning qirralari bo'ladi. Endi grafik tasvirlashga o'tamiz. Ikkita nuqtani belgilab, ularni 1 va 2 deb raqamlaymiz:

Endi ularni bir-biri bilan qanday tutashtirish mumkinligini asosiy variantlarini:



va qo'shimcha variantlarini ko'rib chiqamiz:

10 - variant 	11 - variant 	12 - variant 
13 - variant 	14 - variant 	15 - variant 

Boshqa variantlarni talabalarga mustaqil ko'rib chiqishni tavsiya etamiz.

1-variantda keltirilgan nuqtalar grafning uchlari (ba'zi adabiyotlarda tugun, cho'qqi, bo'g'in) deyiladi.

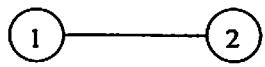
2-variantda keltirilgan tutashish chizig'i qirra deyiladi. Bu yerda 1- va 2 – uchlar o'zaro tutashgan hisoblanadi va qo'shni deb ataladi.

3-variantda keltirilgan tutashishda qirraning yo'nalishi bor, uni yoy deb aytishadi. Bu yerda 1 va 2 uchlar tutashgan, lekin 2 va 1 uchlar tutashmagan hisoblanadi, ya'ni 3- bilan 4- variantlar har xil hisoblanadi.

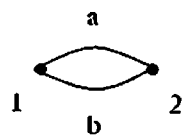
5-9 - variantlarda keltirilgan yoy (xuddi shunday qirra ham) bir uchdan chiqib, yana o'sha uchga kiryapti, u sirtmoq deb ataladi.

10-15 - variantlarda keltirilgan yoylar (xuddi shunday qirralar ham) qo'shsha hisoblanadi.

Ba'zida graf uchlarini aylana bilan belgilash qulayroq bo'ladi, masalan

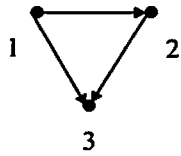


Grafdagi qirralarni harflar bilan belgilash mumkin:

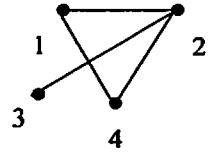


Bu yerda *a*, *b* qirralar 1-uchga insident deb aytiladi. O'z navbatida 1-uch bevosita *a*, *b* qirralarning har biriga insidentdir.

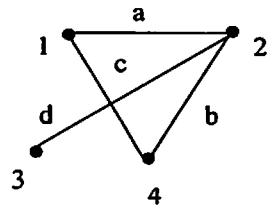
Faqatgina yoylardan tashkil topgan graf yo'naltirilgan, yoki orgraf deb ataladi.



Faqatgina qirralardan tashkil topgan graf yo‘naltirilmagan deb ataladi.

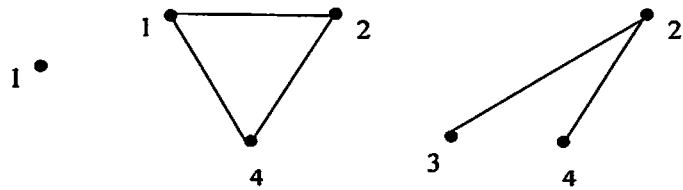


Umumiy uchga ega bo‘lgan qirralar qo‘shni deb ataladi. Xuddi shunday umumiy qirraga ega bo‘lgan uchlar ham qo‘shni bo‘ladi.



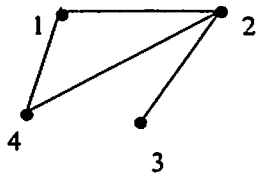
Bu yerda *a*, *b*, *d* qirralar qo‘shni hisoblanadi.

Qism graf bu bevosita grafdan kamida bitta uch va unga insident bo‘lgan qirralarni olib tashlash orqali paydo bo‘lgan grafga aytiladi. Yuqoridagi graf uchun quyidagilar qism graf bo‘ladi:

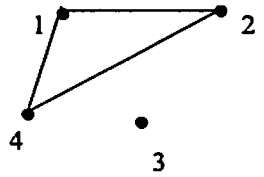


Agar faqatgina qirralarni olib tashlasak sugraf hosil qilinadi, ya’ni yuqoridagi graf uchun quyidagilar sugraf bo‘ladi:

Dasturlashga doir misollarda grafning bog'liqligiga ko'p e'tibor beriladi. Agar grafda istalgan 2 ta uch o'zaro yo'l bilan tutashgan bo'lsa, u bog'liqli graf deb aytiladi, aksincha bog'liqmas.



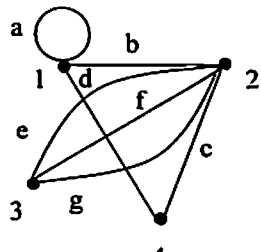
bog'liqli graf



bog'liqmas graf

Agar graf o'zining $v_1, v_2, \ldots, v_n$ uchlari va $e_1, e_2, \ldots, e_m$ qirralari bilan berilgan bo'lsa, graf uchun quyidagi tushunchalarni ham kiritsa bo'ladi.

Qo'shnilik matritsasi $A=(a_{ij})$, i=1,..,n , j=1,...n, bu yerda har bir a_{ij} elementi i va j uchlarni tutashtiruvchi qirralar (yoki yoylar) soniga teng bo'ladi. Quyidagi graf uchun



$$A = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 3 & 1 \\ 0 & 3 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

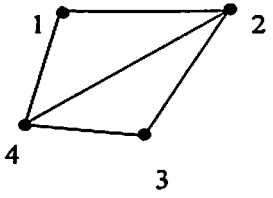
Insedentlik matritsasi $B=(b_{ij})$, i=1,..,n , j=1,...m, bu yerda har bir b_{ij} elementi 1 ga teng bo'ladi agarda v_i uch e_j qirraga insident bo'lsa, aks holda 0 ga teng. Yuqoridagi graf uchun insedentlik matritsasi quyidagicha bo'ladi:

$$B = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Oddiy graf bu sirtmoq va qo'sh qirralar bo'lmagan grafga aytiladi. Keyinchalik graf deyilganda biz oddiy grafni nazarda tutgan bo'lamiz.

v_i va v_j uchlarni tutashtiruvchi qirralar $(v_i, v_{i+1}), \ldots, (v_{j-1}, v_j)$ ketma-ketligi marshrut yoki yo'l deyiladi. Undagi qirralar soni yo'lning uzunligi deyiladi.

Qirralar soni nol bo'lmaganda va $v_i = v_j$ bo'lsa, ushbu yo'l siklik hisoblanadi, masalan,



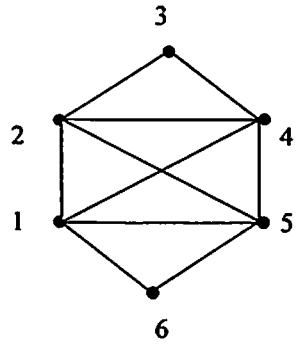
- (1,2), (2,3) - yo'l;
- (1,4), (4,2), (2,3), (3,4), (4,1) – siklik yo'l.

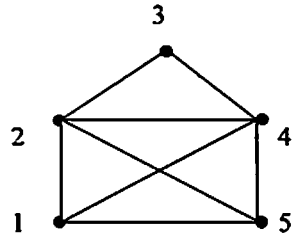
Agarda yo'l har xil qirralardan tashkil topgan bo'lsa - bu yo'l zanjir deb aytiladi, ya'ni (1,4), (4,2), (2,3).

Siklik yo'l sikl deb aytiladi, ya'ni (1,4), (4,3), (3,2), (2,1) .

Bu yerda Eyler siklini ham ta'kidlash lozim – bu barcha qirralarni qamrab olgan siklga aytiladi. Grafda Eyler sikli mavjud bo'lishi uchun uning uchlarida qirralar soni juft bo'lishi kifoya. Yo'naltirilgan graf uchun esa har bir uchga keladigan yoylar soni undan chiqadigan yoylar soniga teng bo'lishi kifoya. Bunga quyidagicha izoh berish mumkin, ya'ni qalamni qog'ozdan uzmasdan turib, barcha qirralarni aylanib o'tsak – bu Eyler sikli bo'ladi.

Misol

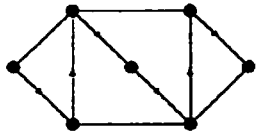


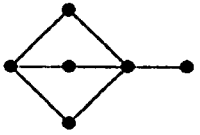


Eyler sikli bor. Eyler sikli yo'q.

1→2→3→4→5→6→1→4→2→5→1

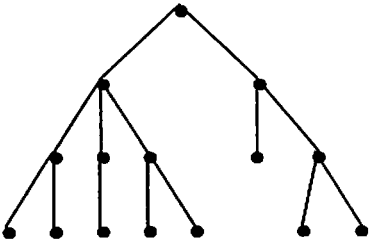
Bundan tashqari Gamilton sikli tushunchasi mavjud. Bunda grafning barcha uchlarini qamrab olgan siklga aytiladi. Grafda Gamilton sikli mavjud bo'lishi haqida shu vaqtgacha hech qanday ma'lumot yo'q.





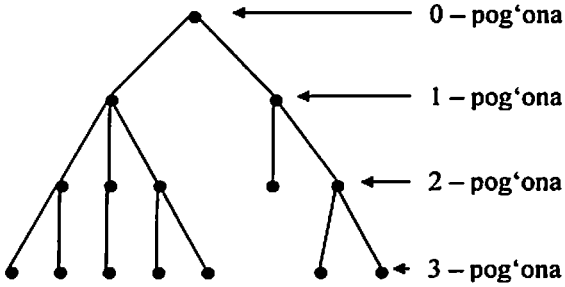
Gamilton sikli bor Gamilton sikli yo'q

Bundan tashqari grafning ajoyib tushunchalaridan biri – bu daraxt tushunchasi. Umumiy holda daraxt – bu siklsiz bog'liqli grafdir. Quyidagi chizmada daraxt keltirilgan:

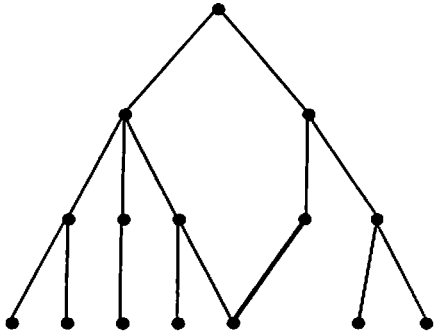


Bu yerdan daraxtning istalgan 2 ta uchi yagona zanjir bilan bog'langanligini ko'rish mumkin. Daraxtlar uchun ildiz va pog'ona

tushunchasi kiritilgan. Daraxtning istalgan uchini tanlab olib uning ildizi deb ataymiz va uni 0-pog'ona deb belgilaymiz. Ildizga qo'shni bo'lgan barcha uchlarni birinchi pog'ona uchlari deb olamiz va hokazo.



Daraxtlarning ajoyib xususiyatlari mavjud, ya'ni unda qirralarning soni uchlari sonidan bittaga kam bo'ladi va istalgan ikki uchni qirra bilan ulasak natijada sikl hosil bo'ladi.



Daraxtning ildizidan, masalan 1 dan, uning istalgan n uchigacha yo'l mavjud bo'ladi. Ushbu yo'ldagi uchlarni *n* ning ajdodi deb aytiladi. Va aksincha, 1 dan *n* gacha bo'lgan uchlar 1 ning avlodi bo'ladi. Faqatgina ildizning ajdodi bo'lmaydi.

Har qanday daraxtning *i* uchini ildiz deb olsak, uning avlodi qism daraxt hisoblanadi.

Ba'zi-bir adabiyotlarda (*i,j*) qirrada *i* uchi "otasi" va *j* uch "farzand" deyiladi. Farzandi bo'lmagan daraxt uchi esa barg deyiladi.

Daraxt ildizidan *i* uchiga bo'lgan yo'l uzunligi *i* ning chuqurligi deyiladi. Eng katta chuqurlik daraxtning balandligi deyiladi.

Ikkilik daraxti – bu har bir uch ko'pi bilan 2 ta qirraga ega bo'lgan daraxtdir.

9.2. Grafda izlash algoritmlari

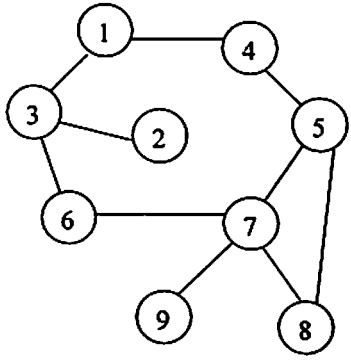
Graflarga doir algoritmlarning g'oyasini anglab yetish va ularni amaliyotda qo'llash oson ish emas. Shu bois ham faqatgina misollar doirasida ishlab chiqilgan algoritmlar bilan tanishib chiqamiz.

Ichkarilab izlash. Bu yerda asosiy g'oya quyidagidan iborat. Grafning v uchidan izlashni boshlaymiz. Ushbu v uchga qo'shni bo'lgan u uchga o'tamiz va uni tanlaymiz. Shu tariqa davom ettirib q uchga kelib qolamiz. Agar q uchida boshqa ko'rilmagan bo'sh uch bo'lmasa, unda q uchidan oldingi uchga qaytamiz. Jarayonni davom ettiramiz. Qaytishdagi uch bevosita v bo'lsa, u holda jarayonni to'xtatamiz.

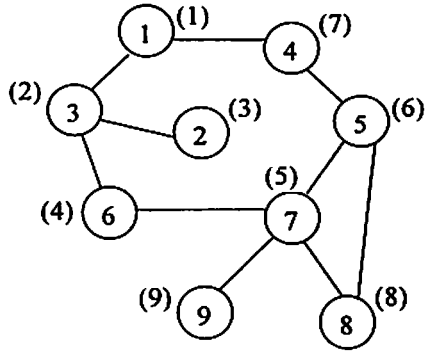
Graf uchlarini ko'rib chiqilganini belgisini quyidagi mantiqiy massivda saqlash mumkin:

Nnew : array[1..N] of Boolean;

Misol sifatida quyidagi grafni ko'rib chiqamiz:



Yuqorida keltirilgan algoritm bo'yicha graf uchlarini aylanib o'tish tartibi quyidagi chizmada keltirilgan:



Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
9 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0	1 3 2 6 7 5 4 8 9

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=9, ikkinchi qatordan boshlab qo'shnilik massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida aylanib o'tilgan graf uchlari chiqarilgan bo'ladi.

Ushbu algoritmning rekursiya asosida yaratilgan dasturi quyidagicha bo'ladi:

```
program Ichkarilab_izlash_Rek;
const MaxSize=10;
var
      Matrix: array [1..MaxSize, 1..MaxSize] of integer;
      Nnew : array[1.. MaxSize] of Boolean;
      i,n: integer;
      faylout: text;
{qo'shimcha funksiyalar}
      {Boshlang'ich qiymatlar }
Procedure Init;
var
      fayl: text;
      i, j: integer;
begin
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, n);
      for i:=1 to n do
begin
```

```
      For j:=1 to n do Read(fayl, Matrix[i,j]);
      { Matrix[]  qo'shnilik massivi}
      Readln(fayl)
 end;
      close(fayl);
{chiqarish faylini ochamiz}
      assign(faylout, 'output.txt');
      rewrite(faylout);
end;
{ Ichkarilab izlashning rekursiyali  algoritmi}
procedure Pg(v:integer);
var j:integer;
begin
      Nnew[v]:=false; write(faylout, v:2);
      for j:=1 to n do
      if (Matrix [v,j]<>0) and Nnew[j] then Pg(j);
end;
{Asosiy dastur}
begin
      Init;
      FillChar(Nnew,SizeOf(Nnew),true);
      for i:=1 to n do if Nnew[i] then Pg(i);
      close(faylout);
end.
```

Ushbu algoritmning ko'p hollarda qo'llanishini e'tibor olgan holda uning norekursiv variantini ham keltiramiz, bu yerda *St* – ko'rib chiqilgan uchlar saqlanadigan massiv va *yk* – uning indeksi:

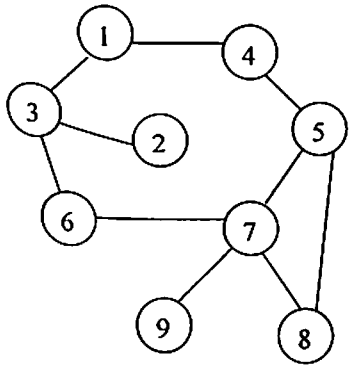
```
program Ichkarilab_izlash;
const MaxSize=10;
var
      Matrix: array [1..MaxSize, 1..MaxSize] of integer;
      Nnew : array[1.. MaxSize] of boolean;
      i,n: integer;
      faylout: text;
{qo'shimcha funksiyalar}
      {Boshlang'ich qiymatlar }
Procedure Init;
Var
      fayl: text;
```

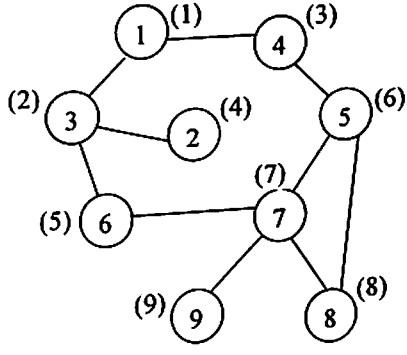
```
      i, j: integer;
begin
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, n);
      for i:=1 to n do
begin
      For j:=1 to n do Read(fayl, Matrix[i,j]);
      { Matrix [] qo'shnilik massivi}
      Readln(fayl)
end;
      close(fayl);
{chiqarish faylini ochamiz}
      assign(faylout, 'output.txt');
      rewrite(faylout);
end;
{ Ichkarilab izlash algoritmi}
procedure PgOddiy(v:integer);
var
      St:array[1.. MaxSize] of integer;
      yk:integer;
      t,j:integer;
      pp:boolean;
begin
      FillChar(St,SizeOf(St),0); yk:=0;
      Inc(yk);St[yk]:=v;
      write(faylout, v:2);
      Nnew[v]:=false;
while yk<>0 do
begin {St massivi bo'sh emas}
      t:=St[yk];  {massivdan eng oxirgi uch tanlanadi}
      j:=0;
      pp:=false;
repeat
      if (Matrix [t,j+1] <>0) and Nnew[j+1] then pp:=true
      else Inc(j);
until pp or (j>=N);
{ushbu siklni to'xtatamiz agarda yangi uch topilsa yoki} {ushbu uch
bilan bog'liq uchlar ko'rib chiqilgan bo'lsa }
```

```
    if pp then
begin
    Inc(yk);
St[yk]:=j+1;
    write(faylout, j+1 :2);
    Nnew[j+1]:=false;
{uni St massiviga kiritdik}
end
    else Dec(yk); { aksincha bitta orqaga qaytamiz}
end;
end;
{Asosiy dastur}
begin
    Init;
    FillChar(Nnew,SizeOf(Nnew),true);
    for i:=1 to n do
    if Nnew[i] then PgOddiy(i);
    close(faylout);
end.
```

Yonma-yon izlash. Bu yerda tanlangan uchdan qo'shni bo'lgan barcha uchlar ko'rib chiqiladi. Keyinchalik ulardan birinchisi tanlanib, unga qo'shni barcha uchlar birma-bir ko'rib chiqiladi.

Quyida keltirilgan graf misolida ushbu algoritm natijasi keltirilgan





Bu yerda qavsning ichida uchlarga qaysi qadamda kelishlar keltirilgan.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
9 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0	1 3 4 2 6 5 7 8 9

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=9, ikkinchi qatordan boshlab qo'shnilik massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida aylanib o'tilgan graf uchlari chiqarilgan bo'ladi.

Ushbu algoritmning dasturi quyida keltirilgan:

```
program Yonma_yon_izlash;
Const MaxSize=10;
var
      Matrix: array [1..MaxSize, 1..MaxSize] of integer;
      Nnew : array[1.. MaxSize] of boolean;
      i,n: integer;
      faylout : text;
{qo'shimcha funksiyalar}
{Boshlang'ich qiymatlar }
procedure Init;
var f   : text;
      i, j: integer;
begin
      assign(f, 'input.txt');
      reset(f);
      Readln(f, n);
      For i:=1 to n do
begin
      For j:=1 to n do Read(f, Matrix [i,j]);
{ Matrix[]  qo'shnilik massivi}
      Readln(f)
```

```
end;
    close(f);
{chiqarish faylini ochamiz}
    assign(faylout, 'output.txt');
    rewrite(faylout);
end;
{ Yonma-yon izlash algoritmi}
procedure PwOddiy(v:integer);
var
    Og:array[1.. MaxSize] of 0.. MaxSize;
{ushbu massivga navbatma-navbat uchlar kiritiladi va} {undan
o'qiladi}
    yk1,yk2:integer;
{Og[]massivdan, yk1 – kiritiladigan, }
{yk2 –o'qiladigan elementlar indekslari}
    j:integer;
begin
    FillChar(Og,SizeOf(Og),0);yk1:=0;yk2:=0;
{boshlang'ich qiymatlar}
    Inc(yk1); Og[yk1]:=v; Nnew[v]:=false;
{ Og[] massivga  v uchni kiritamiz}
while yk2<yk1 do
begin
{massivdan elementlar navbati bilan o'qiladi}
    Inc(yk2);
    v:=Og[yk2];
    write(faylout, v:2);
{ massivdan navbatdagi uchni tanlaymiz }
    for j:=1 to n do
{v uch bilan qo'shni bo'lgan uchlarni ko'rib chiqamiz}
    if (Matrix [v,j]<>0) and Nnew[j] then
{ bu uch oldinroq ko'rilmagan bo'lsa davom ettiramiz}
begin
    Inc(yk1);Og[yk1]:=j;
    Nnew[j]:=false;{navbat ro'yxatiga kiritamiz}
end;
end;
end;
{Asosiy dastur}
```

```
begin
    Init;
    FillChar(Nnew,SizeOf(Nnew),true);
    for i:=1 to n do
    if Nnew[i] then PwOddiy(i);
    close(faylout);
end.
```

9.3. Grafning bog'liq qismini aniqlash

Faqatgina qirralardan tashkil topgan yo'naltirilmagan grafda bog'liqli qismini aniqlash talab etiladi.

Oldiniga graf uchlarini quyidagicha tasniflaymiz:

1. Ular haqida hech qanday ma'lumot yo'q.
2. Ularga boshlang'ich uchdan yo'l mavjud lekin ular qayta ishlanmagan.
3. Ularga boshlang'ich uchdan yo'l mavjud lekin va ular qayta ishlangan.

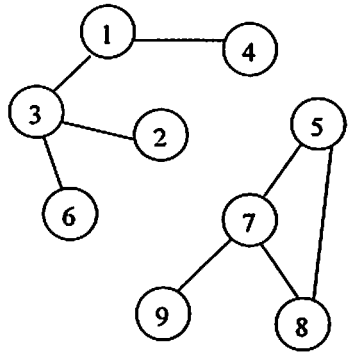
Shundan kelib chiqqan holda ushbu algoritm 3 bosqichdan tashkil topgan bo'ladi: 1) boshlang'ich belgilash; 2) qo'shni uchlarni belgilash; 3) natijaviy xulosalarni chiqarish.

Har bir bosqichni batafsil yoritamiz.

1. Boshlang'ich belgilash. Grafning barcha uchlarini bir bilan belgilaymiz, chunki ular haqida biz hozircha hech narsa bilmaymiz. Grafning birinchi uchini 2 deb belgilaymiz, chunki u o'zi bilan bog'langan hisoblanadi.
2. Bo'sh uchlarni belgilash. Agar 2 bilan belgilangan uchlar bo'lmasa, u holda 3-bosqichga o'tamiz. Aksincha , 2 bilan belgilangan uchni tanlaymiz. Uni 3 bilan belgilaymiz. Ushbu uch bilan qo'shni bir bilan belgilangan uchlarni 2 bilan belgilaymiz. Ushbu jarayonni takrorlaymiz.
3. Natijaviy xulosalarni chiqarish. Tanlangan uchni o'z ichiga oladigan bog'liq qism grafning uchlari 3 bilan belgilangan bo'ladi. Shu bois ularni alohida massivga joylashtiramiz Bog'liq qism grafga kirmaydigan uchlar 1 bilan belgilangan bo'ladi, ularni ham alohida massivga kiritamiz.

Grafning o'zini bog'liqligini aniqlash uchun 1 bilan belgilangan uchlarni aniqlaymiz. Agar ularning soni nolga teng bo'lsa, demak graf bog'liq bo'ladi.

Quyida keltirilgan graf misolida ushbu algoritm natijasi keltirilgan



Bu yerda qavsning ichida uchlarga qaysi qadamda kelishlar keltirilgan.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
9 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0	1 2 3 4 6 5 7 8 9 Graf yaxlitmas

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=9, ikkinchi qatordan boshlab qo'shnilik massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylining birinchi qatorida *first*=1 uchini o'z ichiga olgan bog'liq qism graf uchlari, ikkinchi qatorida esa unga kirmaydigan uchlar va oxirgi qatorda '*Graf yaxlit*' yoki '*Graf yaxlitmas*' qiymatlari chiqarilgan bo'ladi.

Asosiy dastur esa quyidagicha bo'ladi.

```
program BogliqGraf;
uses crt;
const MaxSize =10;
type uchlar=1..3;{ uchlarni belgisi}
```

```
var  about:array[1.. MaxSize] of uchlar;
     Matrix:array[1.. MaxSize,1.. MaxSize] of integer;
     first,next,i,j,n:integer;
     b:boolean;
   faylout : text;
{Boshlang'ich qiymatlar }
Procedure Init;
var  f: text;
     i, j: integer;
begin
     assign(f, 'input.txt');
     reset(f);
     readln(f, n);
     for i:=1 to n do
begin
     For j:=1 to n do Read(f, Matrix [i,j]);
{ Matrix []  qo'shnilik massivi}
     Readln(f)
end;
     close(f);
{chiqarish faylini ochamiz}
     assign(faylout, 'output.txt');
     rewrite(faylout);
end;
begin
fillchar(matrix,sizeof(matrix),0);{0-lar bilan to'ldiramiz}
     fillchar(about,sizeof(about),1);
{- birlar bilan to'ldiramiz}
     Init;
{- Qo'shnilik matrix massiviga qiymatlarni kiritish }
     first:=1;{masalan, birinchi uchni tanladik}
     about[first]:=2;
     b:=true;{2 bilan belgilangan uchlar uchun}
while b do
begin
     b:=false;
     for i:=1 to n do
     if about[i]=2 then
{-massividan 2 bilan belgilangan uchlarni qidiramiz...}
begin
```

```
      about[i]:=3;{...topilsa, uni 3  bilan belgilaymiz}
      b:=true;{siklni davom ettirish uchun}
      for j:=1 to n do
      if (matrix[i,j]>0) and (about[j]=1) then          about[j]:=2;
end;
end;
{Asosiy ishlar bajarildi, endi natijaviy  xulosalarni }
{ tashkillashtiramiz}
{first-uchni o'z ichiga olgan bog'liq qism graf uchlari}
      for i:=1 to N do
      if about[i]=3 then write(faylout, i ,' ');
write(faylout);
{ first  -ni o'z ichiga olgan bog'liq qism grafga}      {kirmaydigan
uchlar}
      for i:=1 to N do
      if about[i]=1 then write(faylout, i ,' ');
write(faylout);
{Grafni bog'liqligini aniqlaymiz}
      j:=0;
        for i:=1 to N do
        if about[i]=1 then   j:=j+1;
        if j=0 then writeln(faylout, 'Graf yaxlit')
        else writeln(faylout, 'Graf yaxlitmas');
        close(faylout);
end.
```

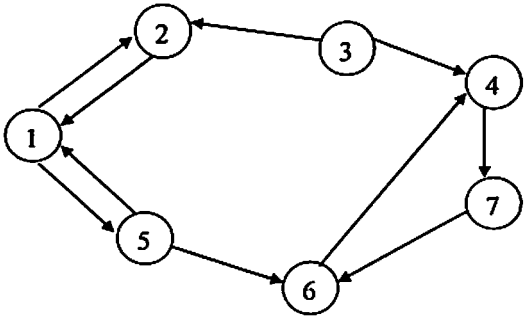
Agarda to'rtinchi va beshinchi uchlarni tutashtirib qo'ysak, u holda natija quyidagicha bo'ladi:

input.txt	*output.txt*
9 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0	1 2 3 4 5 6 7 8 9 Graf yaxlit

Yetishish. Yo'naltirilgan grafda v uchdan u uchgacha yo'l mavjud bo'lsa, u holda u uchiga v uchdan yetib kelish mumkin hisoblanadi. Bu yerda yetishish matritsasi $R[v,u]$ quyidagicha kiritiladi

$$R[v,u] = \begin{cases} 1, \text{ agar } u \text{ uchiga } v \text{ uchdan yetib kelish mumkin bo lsa} \\ 0, \text{ aksincha} \end{cases}$$

Quyidagi graf uchun



qo'shnilik matritsasi quyidagicha bo'ladi

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$R[]$ yetishish matritsasini hisoblash dasturining natijasi quyidagicha bo'ladi

$$R = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
7 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0	1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1 1 1 0 1 1 1 1 0 0 0 1 0 1 1 0 0 0 1 0 1 1

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=7, ikkinchi qatordan boshlab qo'shnilik massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida yetishish massivi elementlarining qiymatlari chiqarilgan bo'ladi.

Asosiy dastur esa quyidagicha bo'ladi.

```
program Yetishish;
const MaxSize =10;
var
      Matrix,R:array[1..MaxSize,1..MaxSize] of integer;
      n, i , j :integer;
      faylout : text;
{Boshlang'ich qiymatlar }
procedure Init;
var
      f: text;
      i, j: integer;
begin
      assign(f, 'input.txt');
      reset(f);
      Readln(f, n);
      for i:=1 to n do
begin
      for j:=1 to n do Read(f, Matrix [i,j]);
{ Matrix[]  qo'shnilik massivi}
      Readln(f)
end;
      close(f);
end;
procedure  Reach;
```

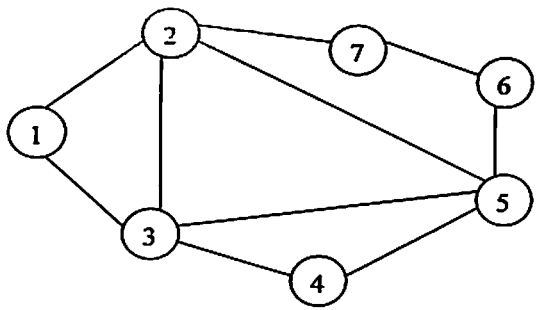
```
{-Yerishish  matritsasini R topish,}
{ Matrix[]- qo'shnilik matritsasi}
var
      S,T:Set  Of  1.. MaxSize;
      i, j, L:Integer;
begin
      FillChar (R,sizeof(R),0);
      For i:=1  To n Do
begin   { i uchdan yetishish}
      T := [i];
repeat
      S:=T;
      For L :=1 To n Do
      If  L   In   S  Then
{Matrix[] ning qatorlari bo'yicha ,  S ga tegishli}
      For j:=1 To n Do
      If Matrix [L,j]=1 Then  T:=T+[j];
Until  S=T;
{  Agar T o'zgarmasa u holda i uchga yetishuvchi barcha} {uchlar
topildi }
      For j :=1    To n Do
      If  j In T Then  R[i,j]:=1;
 end;
end;
{Asosiy dastur}
begin
      Init;
      Reach;
{chiqarish faylini ochamiz}
      assign(faylout, 'output.txt');
      rewrite(faylout);
      for i:=1 to n do
begin
      For j:=1 to n do
      Write(faylout, R[i,j], ' ');
      Writeln(faylout );
end;
      close(faylout);
end.
```

9.4. Sikllarni aniqlash

Eyler sikli. Bu yerda bog'liqli grafda Eyler siklini aniqlash masalasini ko'rib chiqamiz.

Masala yechimini ichkarilab izlash algoritmidan foydalanib topamiz. Faqatgina bu yerda har bir qadamda qirra olib tashlanadi. Aylanib o'tiladigan uchlar xotirada saqlanib boriladi. Agar biror-bir uchdan chiqadigan qirra bo'lmay qolsa, biz uni asosiy ro'yxatda saqlab boramiz va ushbu uchni o'chiramiz va oldingi uchdan ishni davom ettiramiz. Ushbu jarayonni qirra qolgunga qadar davom ettiramiz. Natijada asosiy ro'yxatda Eyler siklini tashkil qiladigan graf uchlari hosil bo'ladi.

Quyidagi graf uchun



Eyler sikli 1 → 3 → 5 → 6 → 7 → 2 → 5 → 4 → 3 → 2 → 1 bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
7 0 1 1 0 0 0 0 1 0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 1 0 1 0 0 0 1 1 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0	1 3 5 6 7 2 5 4 3 2 1

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=7, ikkinchi qatordan boshlab qo'shnilik massiv elementlari bo'sh

katak orqali ajratilib kiritiladi va *output.txt* faylida sikl uchlari chiqarilgan bo'ladi.

Agar grafda Eyler sikli mavjud bo'lmasa, u holda *output.txt* faylida qanday qiymat chop etilishini alohida tahlil qiling.

Asosiy dastur esa quyidagicha bo'ladi.

```
program Eyler_sikli_Rek;
uses crt;
const MaxSize =10;
var
      Matrix,R:array[1.. MaxSize,1.. MaxSize] of integer;
      St:array[1.. MaxSize] of integer;
      n, yk , i :integer;
      faylout: text;
{Boshlang'ich qiymatlar }
Procedure Init;
var
      f: text;
      i, j: integer;
begin
      Assign(f, 'input.txt');
      Reset(f);
      Readln(f, n);
      For i:=1 to n do
begin
      For j:=1 to n do Read(f, Matrix [i,j]);
{ Matrix[]  qo'shnilik massivi}
      Readln(f)
 end;
      close(f);
end;
Procedure Eyler (v: Integer);
{St–stek,yk–stek ko'rsatkichi}
var
      j : integer;
begin
      for j:=1 to n do
      if Matrix [v,j]<> 0 then
begin
      Matrix [v,j]:=0;
      Matrix [j,v]:=0;
```

```
        Eyler (j)
end;
        Inc(yk);
        St[yk]:=v;
end;
{Asosiy dastur}
begin
        Init;
        yk:=0;
        Eyler(1);
{chiqarish faylini ochamiz}
        assign(faylout, 'output.txt');
        rewrite(faylout);
        for i:=1 to yk do
        Write(faylout, St[i], ' ');
        close(faylout);
end.
```

Rekursiyasiz ishlaydigan dasturimiz quyida keltirilgan. Faqat kiritishda biz har bir uchning qirralarini *Qirra_soni*[] massivida sanab boramiz va *Bogliq*[] massivida qatorlar bevosita uchlarni bildirsa, ustunlar qirralarni anglatadi.

Masaladagi ma'lumotlar yuqoridagi graf uchun quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
7 10 1 2 1 3 2 3 2 5 2 7 3 4 3 5 4 5 5 6 6 7	1 3 5 6 7 2 5 4 3 2 1

Bu yerda *input.txt* faylida birinchi qatorda grafdagi uchlar soni n=7, ikkinchi qatorda qirralar soni m=10 , uchinchi qatordan boshlab uchlarning o'zaro bog'langanligi kiritiladi va *output.txt*

faylida sikl uchlari chiqarilgan bo'ladi. Agar grafda Eyler sikli mavjud bo'lmasa, *'Sikl mavjud emas'* deb faylga chiqaramiz.

Asosiy dastur esa quyidagicha bo'ladi.

```
program Eyler_sikli;
const  MaxSize = 10;
var
Bogliq,was:array[1..MaxSize,1..MaxSize]of integer;
      Qirra_soni:array[1.. MaxSize]of integer;
      stack:array[1.. MaxSize * MaxSize]of integer;
      i,n,m,u,v,top: integer;
      ok:boolean;
      f: text;
begin
      Assign(f, 'input.txt');
      Reset(f);
      readln(f,n);    {uchlar soni}
      readln(f,m);    {qirralar soni}
      fillchar(Qirra_soni,sizeof(Qirra_soni),0);
{- nollar bilan to'ldiramiz}
      fillchar(Bogliq,sizeof(Bogliq),0);
{- nollar bilan to'ldiramiz}
      for i:=1 to m do
begin
      read(f, u,v); {qaysi uchlar ulangan}
      inc(Qirra_soni [u]);
      Bogliq [u, Qirra_soni [u]]:=v;
      inc(Qirra_soni [v]);
      Bogliq [v, Qirra_soni [v]]:=u;
end;
      close(f);
{chiqarish faylini ochamiz}
      assign(f, 'output.txt');
      rewrite(f);
      for i:=1 to n do
      if Qirra_soni [i] mod 2=1 then
begin
      writeln(f, 'Sikl mavjud emas');
      close(f);
```

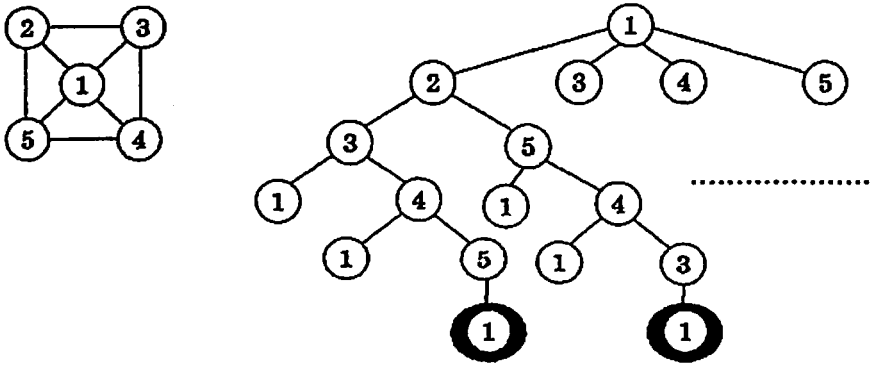
```
      exit;
end;
      top:=1;
      stack[1]:=1;
while top>0 do
begin
      u:=stack[top];
      ok:=true;
      for i:=1 to Qirra_soni [u] do
begin
      v:= Bogliq [u,i];
      if was[u,v]=0 then
begin
      inc(top);
      stack[top]:=v;
      was[u,v]:=1;
      was[v,u]:=1;
      ok:=false;
      break;
end;
end;
      if ok then
begin
      dec(top);
      write(f,u,' ');
end;
end;
      close(f);
end.
```

Gamilton sikli. Yo'naltirilmagan bog'liqli grafda barcha Gamilton sikllarini aniqlash talab etiladi

Bu yerda yechimni aniqlashda barcha variantlarni qaytib saralash usulidan foydalanamiz. Demak, masalan birinchi uchdan boshlaymiz, agar yechimning *K* ta qismi topilgan bo'lsa, keyinchalik oxirgi uchdan chiqadigan qirralarni ko'rib chiqamiz. Oldinroq ko'rib chiqilmagan uch mavjud bo'lsa, uni yechimlar ro'yxatiga kiritamiz va uni ko'rib chiqilganligini belgilab qo'yamiz. Agar bunday uch mavjud bo'lmasa, orqaga qaytib, undan boshqa uchni qidirib

ko'ramiz. Agar barcha uchlar ko'rib chiqilsa va oxirgisidan birinchi uchga o'tish mumkin bo'lsa, demak yechim topilgan hisoblanadi. Uni chop yetib, keyingi siklni qidirishni davom ettiramiz.

Quyidagi keltirilgan graf uchun barcha variantlarni ko'rib chiqish sxemasi keltirilgan.



Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 0 1 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 0 1 1 1 0 1 0	1 2 3 4 5 1 1 2 5 4 3 1 1 3 2 5 4 1 1 3 4 5 2 1 1 4 3 2 5 1 1 4 5 2 3 1 1 5 2 3 4 1 1 5 4 3 2 1

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=5, ikkinchi qatordan boshlab qo'shnilik massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida sikl uchlari chiqarilgan bo'ladi.

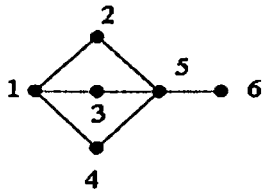
Asosiy dastur esa quyidagicha bo'ladi.

```
program Gamilton_sikli_Rek;
{ Gamilton sikli rekursiya yordamida }
```

```
const MaxSize =10;
var
Matrix, R :array[1.. MaxSize,1.. MaxSize] of integer;
      St:array[1.. MaxSize] of integer;
      Nnew : array[1.. MaxSize] of boolean;
      n, yk, sikl :integer;
      faylout: text;
{Boshlang'ich qiymatlar }
procedure Init;
var
      f: text;
      i, j: integer;
begin
      assign(f, 'input.txt');
      Reset(f);
      Readln(f, n);
      For i:=1 to n do
begin
      For j:=1 to n do Read(f, Matrix [i,j]);
{ Matrix[]  qo'shnilik massivi}
      Readln(f)
end;
      close(f);
{chiqarish faylini ochamiz}
      assign(faylout, 'output.txt');
      rewrite(faylout);
      sikl:=0;
      FillChar(Nnew, SizeOf(Nnew), true);
end;
procedure Gm (k: Integer);
{k- iteratsiya qadami, Matrix – qo'shnilik massivi,} {St – stek, yk
– stek ko'rsatkichi, Nnew – uchlarni} {ko'rilganligini belgilashda
qo'llaniladigan massiv}
var
      i, j, v : integer;
begin
      v:=St[k-1]; {oxirgi uch raqami}
      for j:=1 to n do
      if Matrix [v,j]<> 0 then
```

```
{ v va j uchlari orasida qirra bor }
      if (k=N+1) And (j=1) Then {siklni chop etamiz}
begin
      sikl:=1;
      for i:=1 to yk do
      Write(faylout , St[i], ' ');
      write(faylout, St[1] ); writeln(faylout);
end
      else
      If Nnew[j] Then
begin {uch ko'rilmagan}
      St[k]:=j;yk:=k;
      Nnew[j]:=False;
      Gm(k+1);
      Nnew[j]:=True;
end;
end;
{Asosiy dastur}
begin
      Init;
      yk:=0;
      St[1]:=1;
      Nnew[1]:=False;
      Gm(2);
      if sikl=0 then
      write(faylout,'Gamilton sikli mavjud emas');
      close(faylout);
end.
```

Quyidagi grafda gamilton sikli mavjud emas:



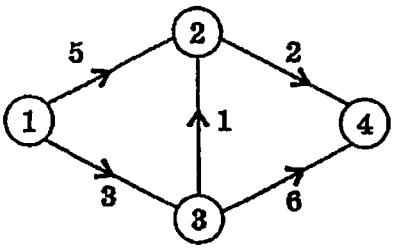
Ushbu grafning go'shnilik massiv elementlari quyidagicha *input.txt* faylida kiritiladi va natija *output.txt* chop etiladi:

input.txt	*output.txt*
6 0 1 1 1 0 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 1 1 0 1 0 0 0 0 1 0	*Gamilton sikli mavjud emas*

9.5. Qisqa yo'lni aniqlash algoritmlari

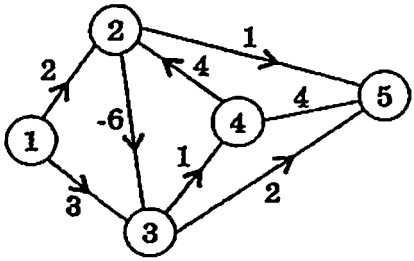
Graflar bilan amallar bajarilganda ko'p hollarda ikki uchlar orasidagi yo'llarni aniqlash va ular orasidan eng qisqasini, ya'ni kam qirralardan iborat bo'lganini yoki optimallik shartlarini bajaradigan yo'llarni aniqlash talab etiladi. Bu yerda graf o'zining qo'shnilik $A[]$ matritsasi orqali berilgan bo'ladi. Unda $A[i,j]$ element i va j uchlarini birlashtirsa bevosita ushbu qirraning vazni kiritilgan bo'lishi mumkin, aksincha $A[i,j]$ elementiga "*cheksiz*" qiymatni kiritamiz. Shunday qilib s va t uchlari orasidagi yo'l $\sum A[i,j]$ yig'indi bilan aniqlanadi. Ushbu yig'indini minimal qiymatini ta'minlaydigan yo'lni qidirish talab etiladi.

Quyidagi graf uchun



1 va 4 uchlar uchun eng qisqa yo'l: $1 \rightarrow 3 \rightarrow 2 \rightarrow 4$ bo'ladi.

Vaznlar manfiy bo'lsa konturli graflarda muammo paydo bo'lishi mumkin, masalan quyidagi graf uchun



$2 \rightarrow 3 \rightarrow 4 \rightarrow 2$ kontur bo'lib, uni cheksiz aylanib o'tsak natijada istalgan manfiy sonni tashkil qilish mumkin bo'ladi.

Deykstr algoritmi. Har qanday kontursiz yo'naltirilgan graf uchun berilgan boshlang'ich uchdan ko'rsatilgan oxirgi uchgacha bo'lgan minimal yo'lni aniqlang.

Bu yerda har bir qirraning vazni, ya'ni yo'l uzunligi berilgan bo'lib, ular manfiy emas. Bundan tashqari, agar i va j bevosita bog'lanmagan bo'lasa, u holda $A[i,j]=\infty$ deb olamiz.

Ushbu masalani Deykstr 1959 yilda yechimini taklif qilgan. Unga binoan quyidagi massivlarni kiritamiz:

- *Visited*[1..*n*] massivi, u ikki qiymat qabul qiladi, ya'ni *False* – graf uchi ko'rib chiqilmagan, *True* - graf uchi ko'rilgan.
- *Len*[1..*n*] massivi, bu boshlang'ich uchdan qolganlaricha bo'lgan masofani saqlash uchun.
- *Path*[1..*n*] massivi, bu yerda graf uchlari saqlanadi, uning *Path*[*k*] qiymati quyidagicha aniqlanadi, boshlang'ich uchdan *k*-uchgacha bo'lgan eng qisqa yo'ldagi *k* dan oldin turgan uch.
- *Matrix*[1..*n*,1..*n*] massivi , bu i va j uchlari orasidagi masofalardan iborat massiv.

Asosiy g'oya nimadan iborat? Berilgan uchdan qolgan uchlarning qaysi biriga o'tish qiymati minimal bo'lsa, o'sha uchga o'tamiz va undan qolgan uchlargacha bo'lgan minimal bo'lgan uch aniqlanadi va h.k.

Endi Deykstr algoritmini keltiramiz.

1. *Visited*[1..*n*] massivini *False* qiymati bilan to'ldiramiz, *Path*[] massivining *Path*[*i*]:=*Start* deb olamiz, Matrix massividan *Start* qatorini *Len*[] ga ko'chiramiz. Keyinchalik tanlangan *Start* uchun *Visited*[*Start*]:=*True* va *Path*[*Start*]:=0 deb olamiz.

2. Belgilanmagan uchlar ichidan (*Visitid*[*k*]=*False*) minimal qiymatni aniqlaymiz, masalan *j* bo'lsin, ya'ni *Len*[*j*] ≤ *Len*[*k*];
Keyin quyidagilar bajariladi:

Visited[i]:=True;
Agar Len[k]>Len[j]+Matrix[j, k] bo'lsa, u holda
(Len[k]:=Len[j]+Matrix[j, k]; Path[k]:=j)
z:= Path[k];
z ni chop etamiz
z:= Path[z].
Agar z =0 bo'lsa tamom, aksincha 2-bandga o'tamiz.

Masaladagi ma'lumotlar yuqorida keltirilgan birinchi graf ychun quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
4 1 4 0 5 3 1000 1000 0 1000 2 1000 1 0 6 1000 1000 1000 0	4 2 3 1

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami *n*=4, boshlangich uch *Start*=1 va oxirgi uch *Finish*=4, ikkinchi qatordan boshlab qo'shnilik massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida chop etilayotgan qisqa yo'l uchlari teskari tartibda chiqariladi.

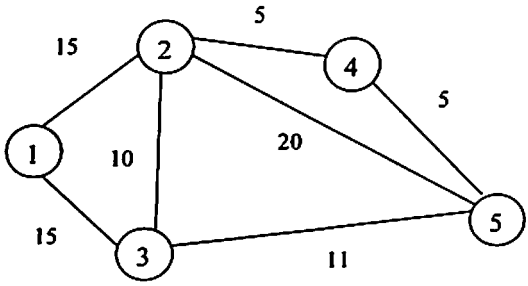
Dastur esa quyidagicha bo'ladi:

```
program Minimal_Yol;
Const MaxSize=10;
Infinity=1000; {katta son}
var
    Matrix: array [1..MaxSize, 1..MaxSize] of integer;
    Visited: array [1..MaxSize] of boolean;
    Len,Path: array [1..MaxSize] of integer;
    n, Start, Finish, k, i: integer;
    faylout: text;
{qo'shimcha funksiyalar}
{Boshlang'ich qiymatlar }
Procedure Init;
Var
```

```
      f: text;
      i, j: integer;
begin
      Assign(f, 'input.txt');
      Reset(f);
      Readln(f, n, Start, Finish );
      For i:=1 to n do
begin
      For j:=1 to n do Read(f, matrix[i,j]);
      Readln(f)
 end;
      close(f);
{chiqarish faylini ochamiz}
      assign(faylout, 'output.txt');
      rewrite(faylout);
      write(faylout, Finish, ' ');
      For i:=1 to n do
begin
      Visited[i]:=False;
      Len[i]:=Matrix[Start, i];
      Path[i]:=Start
end;
      Path[Start]:=0;
      Visited[Start]:=True
end;
{ko'rib chiqilmagan uchni bor-yo'qligini aniqlaydi }
Function Possible: Boolean;
var
      i: integer;
begin
      Possible:=True;
      For i:=1 to n do If not Visited[i] then Exit;
      Possible:=False
end;
{ko'rib chiqilmagan uchlar ichidan Len[i] larning eng } {kichik
qiymatining indeksini aniqlovchi funksiya }
Function MinInLen: Integer;
var
      i, minvalue, currentmin: integer;
```

```
begin
      Minvalue:=Infinity;
      For i:=1 to n do
      If not Visited[i] then
      If Len[i]<minvalue then
begin
      currentmin:=i;
      minvalue:=Len[i]
end;
      MinInLen:=currentmin
end;
{dasturning asosiy qismi}
begin
      Init;
While Possible do
begin
      k:= MinInLen;
      Visited[k]:=True;
      For i:=1 to n do
      If Len[i]>Len[k]+Matrix[k, i] then
begin
      Len[i]:=Len[k]+Matrix[k, i];
      Path[i]:=k;
end
end;
      finish:=Path[Finish];
While Finish<>0 do
begin
      Write(faylout,  Finish, ' ');
      Finish:=Path[Finish];
end;
      close(faylout);
end.
```

Ushbu dasturni yo‘naltirilmagan graf uchun ham qo‘llash mumkin. Masalan, quyidagi vaznlari berilgan graf va qo‘shnilik matritsasi uchun natijani keltiramiz:

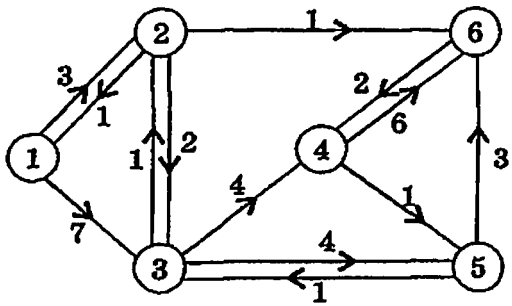


$$Matrix = \begin{pmatrix} 0 & 15 & 15 & \infty & \infty \\ 15 & 0 & 10 & 5 & 20 \\ 15 & 10 & 0 & \infty & 11 \\ \infty & 5 & \infty & 0 & 5 \\ \infty & 20 & 11 & 5 & 0 \end{pmatrix}$$

Yuqorida keltirilgan dastur natijasi *Start*=1-uchdan *Finish*=5-uchgacha eng qisqa yo'l: 5 - 4 - 2 - 1 bo'ladi, e'tibor bering, natija teskari chop etiladi.

Quyidagi yo'naltirilgan va vaznlari berilgan graf va qo'shnilik matritsasi uchun boshqa algotmni va natijalarni tahlili bilan keltiramiz.

Endi barcha uchlargacha bo'lgan algoritmni keltiramiz. Demak, berilgan yo'naltirilgan grafda *Start* uchidan boshqa uchlargacha bo'lgan minimal yo'llarni aniqlaymiz, ushbu natijalarni *D*[] massivida saqlaymiz. Bu erda *T* da uchlar to'plamini kiritamiz va unda faqatgina minimal yo'llar aniqlanmagan uchlarni saqlaymiz.



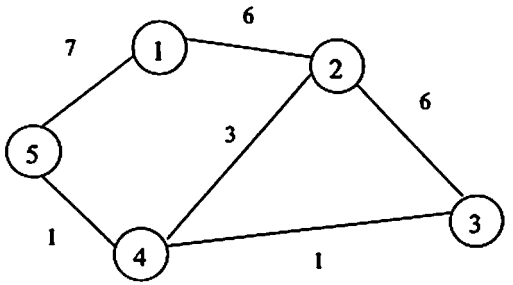
$$Matrix = \begin{pmatrix} \infty & 3 & 7 & \infty & \infty & \infty \\ 1 & \infty & 2 & \infty & \infty & 1 \\ \infty & 1 & \infty & 4 & 4 & \infty \\ \infty & \infty & \infty & \infty & 1 & 5 \\ \infty & \infty & 1 & \infty & \infty & 3 \\ \infty & \infty & \infty & 2 & \infty & \infty \end{pmatrix}$$

Ushbu misol doirasida algoritmni qadamba-qadam keltiramiz. Uchlar to'plami T=[2,3,4,5,6] bo'ladi. Faraz qilamiz *Start*=1 bo'lsin, u holda *Start* uchidan eng minimal yo'l ikkinchi uchda erishiladi, ya'ni D[] massiv elementlarining ichida minimal qiymat ikkinchi uchga to'g'ri keladi. Keyin ushbu uchni T to'plamdan chiqarib tashlaymiz. Qolgan uchlargacha bo'lgan hisoblashlarni davom ettiramiz va uni minimallashtiramiz, faqatgina qolgan [3,4,5,6] uchlargacha minimal qiymatni ikkinchi uchdan boshlab hisoblaymiz. Natijalarni quyidagi jadvalda keltiramiz:

Qadam №	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	T to'plami
1	0	3	7	∞	∞	∞	[2,3,4,5,6]
2	0	3	5	∞	∞	4	[3,4,5,6]
3	0	3	5	6	∞	4	[3,4,5]
4	0	3	5	6	9	4	[4,5]
5	0	3	5	6	7	4	[5]

Izoh. Qator bo'yicha ochiq rangli katakchalardagi qiymatlar orasidan minimal qiymat aniqlanadi.

Quyidagi yo'naltirilmagan va vaznlari berilgan graf uchun ham natijalarni keltiramiz:



Ushbu grafning qo‘shnilik matritsasi quyidagicha bo‘ladi:

$$Matrix = \begin{pmatrix} \infty & 6 & \infty & \infty & 7 \\ 6 & \infty & 6 & 3 & \infty \\ \infty & 6 & \infty & 1 & \infty \\ \infty & 3 & 1 & \infty & 1 \\ 7 & \infty & \infty & 1 & \infty \end{pmatrix}$$

Start=1 uchun natijalar quyidagi jadvalda keltirilgan:

Qadam №	*D*[1]	*D*[2]	*D*[3]	*D*[4]	*D*[5]	T to‘plami
1	0	6	∞	∞	7	[2,3,4,5]
2	0	6	12	9	7	[3,4,5]
3	0	6	12	8	7	[3,4]
4	0	6	9	8	7	[3]

Masaladagi ma’lumotlar yuqorida keltirilgan graf ychun quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
1 1000 6 1000 1000 7 6 1000 6 3 1000 1000 6 1000 1 1000 1000 3 1 1000 1 7 1000 1000 1 1000	0 6 9 8 7

Bu yerda *input.txt* faylida birinchi qatorda boshlangich uch *Start*=1, ikkinchi qatordan boshlab qo‘shnilik massiv elementlari bo‘sh katak orqali ajratilib kiritiladi (massivning o‘lchami n=5 deb qabul qilingan) va *output.txt* faylida chop etilayotgan minimal qisqa yo‘llar uchlari teskari tartibda chiqariladi.

Demak, ushbu masalalarda barcha uchlargacha bo‘lgan minimal yo‘llarni aniqlash talab etilsa, unda quyidagi dastur orqali natijani olsak bo‘ladi:

```
program Minimal_Yollar;
Const
```

```
      N=5;
      MaxSize=10;
Infinity=1000; {katta son}
var
      Matrix: array [1..MaxSize, 1..MaxSize] of integer;
      D: array [1..MaxSize] of integer;
      { Minimal yollarni ushbu massivda saqlaymiz }
      Start,  i: integer;
      faylout: text;
{qo'shimcha funksiyalar}
      {Boshlang'ich qiymatlar }
procedure Init;
var
      f: text;
      i, j: integer;
begin
      Assign(f, 'input.txt');
      Reset(f);
      Readln(f, Start);
      For i:=1 to n do
begin
      For j:=1 to n do Read(f, matrix[i,j]);
      Readln(f)
end;
      close(f);
{chiqarish faylini ochamiz}
      assign(faylout, 'output.txt');
      rewrite(faylout);
end;
Function Min (a,b : integer ): Integer;
begin
      if a<b then Min:=a else Min:=b;
end;
Procedure Dist ;   { matrix, D, Start, N   asosiy dasturda
aniqlangan bo'ladi}
var
      i, u, L : Integer;
      T: Set of 1..N;
begin
```

```
      For i:=1 to N do D[i]:= matrix [Start,i];
      D[Start] := 0 ; T:=[1..N] - [Start];
While T<> [] Do
begin
      For i:=1 to N do
      if (i in T) then u:=i;
{-T to'plam ichidan bitta uchni tanladik}
      For i:=1 to N do
      if (i in T) and (D[i] < D[u]) then u:=i ;
{-minimum topildi}
{
bu erda tahlil uchun D[] larni chop etish mumkin
      for i:=1 to n do
      writeln('i=',i,'  d[i]=',d[i]);  readln;
}
      T:=T-[u];
      For i:=1 to N do
      If i in T then  D[i]:=Min(D[i],D[u]+matrix[u,i] );
end;
end;
{dasturning asosiy qismi}
begin
      Init;
      Dist;
      For i:=1 to N do Write (faylout, D[i], ' ') ;
      close(faylout);
end.
```

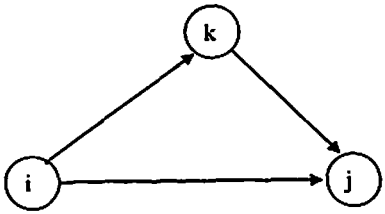
Ushbu algoritmning samaradorlik darajasi $O(N^2)$ tartibli bo'ladi.

Floyd algoritmi.

Yo'naltirilgan graf uchun berilgan har qanday uchdan barcha qolgan uchlargacha bo'lgan minimal yo'lni aniqlang.

Algoritmning asosiy g'oyasi quyidagidan iborat. $D^m[i,j]$ massiv elementi i dan j gacha bo'lgan va $[k,..,m]$ uchlardan iborat minimal yo'lning qiymatini aniqlaydi. Demak, $D^0[i,j] = A[i,j]$ bo'ladi va $D^{m+1}[i,j] = Min(D^m[i,j], D^m[i,m+1] + D^m[m+1,j])$, chunki i va j oralig'idagi minimal yo'l $[1,..,m+1]$ uchlar to'plamining ba'zi-bir uchlaridan tashkil topgan bo'lsin. Agar ushbu yo'l $(m+1)$ uchni o'z ichiga olmasa, demak $D^{m+1}[i,j] = D^m[i,j]$ bo'ladi, aksincha esa yo'lni

ikki qismga bo'lamiz, ya'ni i dan $(m+1)$ gacha va $(m+1)$ dan j gacha. Natijada yuqorida keltirilgan formulaga kelamiz. Sxematik ravishda ushbu algoritmni quyidagicha tasvirlash mumkin:

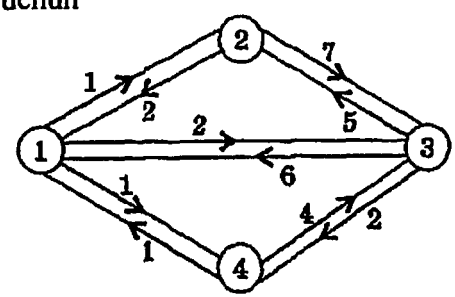


Agar $d_{ik} + d_{kj} < d_{ij}$ bo'lsa $i \rightarrow j$ yo'ldan emas, balkim $i \rightarrow k \rightarrow j$ yo'ldan o'tish optimal bo'ladi.

Ushbu algoritmning samaradorlik darajasi $O(N^3)$ tartibli bo'ladi.

```
Procedure Dist ;
{A va D massivlar asosiy dasturda aniqlangan bo'ladi}
var
    m, i, j : Integer;
begin
    For i:=1 to N do
    For j:=1 to N do D[i,j] := A[i,j] ;
    For i:=2 to N do D[i,i] := 0 ;
    For m:=1 to N do
    For i:=1 to N do
    For j:=1 to N do
    D[i,j]  :=Min( D[i,j],  D[i,m] + D[m,j] );
end;
```

Quyidagi graf uchun



D[] massivining har bir qadamdagi qiymatlari quyidagilarga teng bo'ladi:

$$D^0 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 7 & \infty \\ 6 & 5 & 0 & 2 \\ 1 & \infty & 4 & 0 \end{pmatrix} \qquad D^1 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 6 & 5 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{pmatrix}$$

$$D^2 = D^1 \qquad D^3 = D^2 \quad D^4 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 3 & 4 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{pmatrix}$$

Shunday qilib, ushbu natijada ikki uch orasidagi masofa aniqlanadi. Endi yo'lni aniqlash uchun quyidagi qism dasturni keltiramiz. Bu yerda *Marsh*[*i,j*] massivini kiritamiz, unda $i - j$ optimal yo'lining oxirgisidan bitta oldingi uchni kiritib boramiz. Natijada *Dist* qism dasturi quyidagicha o'zgaradi:

```
Procedure Dist ;
Var
     m, i, j : Integer;
begin
     For i:=1 to N do
     For j:=1 to N do
begin
     D[i,j] := A[i,j] ; Marsh[i,j]:=i;
end;
     For i:=2 to N do D[i,i] := 0 ;
     For m:=1 to N do
     For i:=1 to N do
     For j:=1 to N do
begin
     If ( D[i,j] >  D[i,m] + D[m,j] ) then
begin
     D[i,j]:=D[i,m]+D[m,j];
     Marsh[i,j] := m;
```

```
end;
end;
end;
```

Dastur natijasida *Marsh*[] massivi esa har bir qadamda quyidagicha bo'ladi:

$$Marsh^0 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{pmatrix} \qquad Marsh^1 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 1 & 1 \\ 3 & 3 & 3 & 3 \\ 4 & 1 & 1 & 4 \end{pmatrix}$$

$$Marsh^2 = Marsh^1 \qquad Marsh^3 = Marsh^2$$

$$Marsh^4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 1 & 1 \\ 4 & 4 & 3 & 3 \\ 4 & 1 & 1 & 4 \end{pmatrix}$$

Bu yerdan, masalan 3 dan 2 gacha bo'lgan yo'lni quyidagicha aniqlaymiz: *Marsh*[3,2] qiymati 4 ga teng, shu bois 3 dan 4 gacha va 4 dan 2 gacha bo'lgan yo'lni aniqlaymiz. Demak, *Marsh*[3,4]=3 bo'lganligi uchun 3 → 4 bo'ladi (*Marsh*[3,4] qiymati bu 3 dan 4 ga o'tishdagi 4 dan oldingi uchni bildiradi, bizda *Marsh*[3,4]=3, ya'ni 3 dan bevosita 4 ga o'tamiz). *Marsh*[4,2]=1 bo'lganligi sababli, 4 dan 1 gacha va 1 dan 2 gacha bo'lgan yo'lni aniqlaymiz. Demak, *Marsh*[4,1]=4 bo'lganligi uchun 4 → 1 bo'ladi. *Marsh*[1,2] esa 1 ga teng, ya'ni 1 → 2 bo'ladi. Shunday qilib, eng qisqa yo'l bu 3 → 4 → 1 → 2 bo'ladi.

Ushbu yo'lni chop etish qism dasturini quyidagicha taklif etamiz:

```
Procedure Way(i,j: integer) ;
begin
    If Marsh[i,j] =i then
            if i=j then Write(i)
            else Write(i, ' ', j)
    Else
begin
    Way(i, Marsh[i,j]);
```

```
    Way( Marsh[i,j] , j);
end;
end;
```

Masaladagi ma'lumotlar yuqorida keltirilgan birinchi graf ychun quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
4 0 1 2 1 2 0 7 1000 6 5 0 2 1 1000 4 0	1->2 1>2 1->3 1>3 1->4 1>4 2->1 2>1 2->3 2>1 1>3 2->4 2>1 1>4 3->1 3>4 4>1 3->2 3>4 4>1 1>2 3->4 3>4 4->1 4>1 4->2 4>1 1>2 4->3 4>1 1>3

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=4, ikkinchi qatordan boshlab qo'shnilik massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida chop etilayotgan eng minimal qisqa yo'l uchlari chiqariladi, masalan 3 dan 2 ga o'tish quyidagicha chiqarilgan:

3->2
3>4 4>1 1>2

Endi dasturimizning umumiy ko'rinishi quyidagicha bo'ladi:

```
program Floyd ;
Const MaxSize=10;
Infinity=1000; {katta son}
var
Mattr,Marsh,D:array[1..MaxSize, 1..MaxSize] of integer;
      n, Start,  i, j: integer;
      faylout: text;
{qo'shimcha funksiyalar}
{Boshlang'ich qiymatlar }
Procedure Init;
var
      f: text;
      i, j: integer;
begin
            Assign(f, 'input.txt');
            Reset(f);
            Readln(f, n);
            For i:=1 to n do
begin
            For j:=1 to n do Read(f, mattr[i,j]);
            Readln(f)
end;
            close(f);
{chiqarish faylini ochamiz}
      assign(faylout, 'output.txt');
      rewrite(faylout);
end;
procedure Dist ;
var
      m, i, j , ii, jj: Integer;
begin
      For i:=1 to N do
      For j:=1 to N do
begin
      D[i,j] := mattr[i,j] ; Marsh[i,j]:=i;
end;
```

```
      For i:=2 to N do D[i,i]:=0 ;
      For m:=1 to N do
begin
      For i:=1 to N do
begin
      For j:=1 to N do
begin
      If ( D[i,j] >  D[i,m] + D[m,j] ) then
begin
      D[i,j]:=D[i,m]+D[m,j];
      Marsh[i,j] := m;
end;
end;
end;
{ D[] va Marsh[] massivlarini ko'rib olishda } {qo'llaniladigan
qism}
{*
      For ii:=1 to N do
begin
      For jj:=1 to N do
begin
      write(D[ii,jj],' '); write( Marsh[ii,jj], ' ');
end;      writeln;
end;
readln;
*}
end;
end;
Procedure Way(i,j: integer) ;
begin
      If  Marsh[i,j] =i then
            if i=j then Write(faylout , i , ' ')
            else Write(faylout, i,'>', j,' ')
      Else
begin
      Way(i, Marsh[i,j]);
      Way( Marsh[i,j] , j);
end;
end;
```

```
{dasturning asosiy qismi}
begin
      Init;
      Dist;
          For i:=1 to n do
          For j:=1 to n do
      If i<>j then
begin
      if (i=1) and (j=2) then writeln
      else writeln(faylout);
      writeln(faylout,i, '->',j);
      Way(i,j);
end;
      close(faylout);
end.
```

9.6. Graflarni qo'llashga doir misollar

Uzun zanjir. Berilgan $A=\{A_1, A_2, ..., A_n\}$ to'plami elementlaridan tashkil topgan (A_i,A_j) takrorlanmaydigan juftliklar berilgan. Ushbu juftliklardan eng uzun zanjirli ketma-ketlik quyidagi qoida bo'yicha tuzilsin

$$(A_i,A_j)+(A_j,A_k)=(A_i,A_j,A_k).$$

Bu yerda (A_i,A_j) juftlik faqatgina bir marotaba qo'llanishi mumkin bo'ladi.

Masala yechimi. Grafni tasvirlashda biz qo'shnilik matritsasini kiritgan edik, bu yerda xuddi shunday matritsani tuzamiz, ya'ni bu yerda $C[1...n,1..n]$ elementlari quyidagicha aniqlanadi: agar (A_i,A_j) berilgan bo'lsa $C[i,j]=1$ deb olamiz, aksincha esa $C[i,j]=0$ bo'ladi.

Endi zanjirli ketma-ketlikni tuzishga kirishamiz. "A" to'plamidan istalgan A_i ni tanlaymiz. $C[]$ massividan i-qatorni tekshiramiz, agarda $C[i,j]=1$ bo'lsa demak biz juftlikni belgilab oldik. Agarda bu hol bajarilmasa boshqa i ni olamiz. Demak, (A_i,A_j) juftlikni biz qo'lladik, shu bois $C[i,j]=-1$ deb olamiz. Endi j-qatordan keyingi (A_j,A_k) juftlikni qidiramiz, ya'ni $C[j,k]=1$ bo'lishini tekshiramiz. Agar topilsa A_k ni oldingi zanjirga qo'shib $C[j,k]=-1$ deb olamiz. Endi k-qatorga o'tib, u yerda ham xuddi shunday yo'l tutamiz. Qaysidir qadamda biz quyidagi

$$A_i A_j A_k ... A_s A_L A_p$$

zanjirni tashkil qilamiz. Agar ushbu zanjirning uzunligi oldingilaridan katta bo'lsa, uni xotirada saqlaymiz. Endi undan oxirgi A_p ni olib tashlaymiz va jarayonni davom ettiramiz, faqatgina jarayonni L qatorning o'zidan davom ettiramiz va u yerda p indeksdan keyin ham birga teng qiymat borligini $C[]$ dan tekshiramiz. Topilmasa zanjirdan A_L ni olib tashlaymiz va s qatordan davom ettiramiz va u yerda L indeksdan keyin ham birga teng qiymat borligini $C[]$ dan tekshiramiz va hokazo. Ushbu jarayonni A_i ni olib tashlashgacha davom ettiramiz.

Masaladagi ma'lumotlar yuqoridagi misolda keltirilgan graf uchun quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
3 1 2 2 3 3 4	1 2 3 4

Bu yerda *input.txt* faylida birinchi qatorda juftliklar soni beriladi L=3, ikkinchi qatordan boshlab juftliklar bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida eng uzun zanjirli ketma-ketlik raqamlari chiqariladi.

Ushbu algoritm bo'yicha dastur quyidagicha bo'ladi:

```
Program Zanjir;
const N=4; { A to'plamdagi elementlar soni}
{soddalashtirish maqsadida  A =(1,2,...,N) deb olamiz }
var
      c:array[1..N,1..N] of integer;
      curstr, maxstr: array[0..N] of integer;
{zanjirlar ro'yxati va eng uzun zanjir}
{Nolinchi elementda zanjir uzunligini saqlaymiz}
      E : integer; {E – juftliklar soni}
      i,j,k : integer;
      f: text;
{qism dasturlar}
procedure find;
var
      L,j : integer;
begin
      L:=curstr[curstr[0]];
```

```
{L = zanjirning oxirgi elementi}
      for j:=1 to N do { L qatorni ko'rib chiqamiz}
      if C[L,j]=1 then
begin
      curstr[0]:=curstr[0]+1;
      curstr[curstr[0]]:=j; {j ni zanjirga kiritamiz}
      c[L,j]:=-1; {juftlikdan foydalandik}
      find;
      c[L,j]:=1; { juftlikdan yana foydalansa bo'ladi }
      curstr[0]:=curstr[0]-1;
end;
      if curstr[0]>maxstr[0] {uzunroq zanjir topilsa}
      then maxstr:=curstr {uni saqlab qo'yamiz}
end;
      {Asosiy dastur}
begin
      Assign (f,'input.txt');
      reset(f);
      readln(f, E);
      for i:=1 to N do
      for j:=1 to N do
      C[i,j]:=0;
      for k:=1 to E do
begin
      read(f, i, j);
      { juftliklarni fayldan o'qib olamiz,}
      c[i,j]:=1
end;
      close(f);
      for i:=1 to N do
begin
      curstr [0]:=1;
      {i dan boshlanuvchi zanjirlarni qidiramiz}
      curstr [1]:=i;
      find;
end;
{chiqarish faylini ochamiz}
      assign(f, 'output.txt');
      rewrite(f);
```

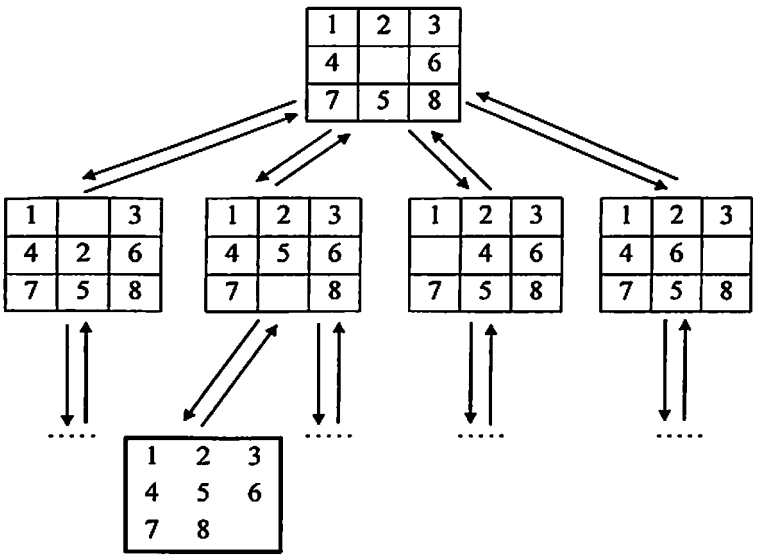
```
    for i:=1 to maxstr[0] do
    write(f, maxstr[i], ' '); {maksimal zanjir}
    close(f);
end.
```

8 o'yini haqida. 15 o'yinida kubiklar tartib bilan raqamlangan bo'lib, maxsys qutichada tartibsiz joylashtirilgan bo'ladi. Demak, ushbu sonlarni betartib joylashgan holatdan, quyidagi chizmada keltirilgan kabi tartib bilan joylashtirish kerak bo'ladi, ya'ni

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Bunda, faqatgina bo'sh katakga gorizontal yoki vertikal qo'shni kataklardan raqamlangan kubiklarni bittaga siljitib o'tkazish mumkin.

Bu erda keltirilgan 15 ta kubiklarning o'rniga, uning kichraytirilgan variantini, ya'ni 8 ta kubikni ko'ramiz va undagi yurishlarni ko'rib chiqamiz, masalan



Ushbu chizmadan xulosa qilib quyidagini ta'kidlash mumkin, ya'ni kubiklarning joylashuvi – bu grafning cho'qqilari bo'lsa, kubik bilan yurish qilish – bu grafning qirrasi bo'ladi.

O'yinning boshidagi kubiklarning joylashuvi bu graf bo'ylab yurishdagi boshlang'ich cho'qqi bo'lib hisoblanadi. O'yinda qabul qilingan qoidalar bo'yicha biz grafning boshqa cho'qqi va qirralarini aniqlashimiz mumkin. Grafdan izlanayotgan natijaviy kubiklar joylashuvi yuqoridagi chizmada qalin chiziq bilan ko'rsatilgan.

E'tibor bering, graf oldindan berilmagan, albatta yuqoridagi qoida bo'yicha grafni to'liq tashkil qilishimiz mumkin, faqatgina undagi cho'qqilar soni juda ham katta bo'ladi. Bu esa uni oddiy usullar orqali, masalan Deykstr usuli bilan, yechimini qisqa vaqt ichida aniqlashga imkon bermaydi.

Shunday qilib, to'liq tuzilmagan grafda optimal yo'lni aniqlash talab etiladi. Bu erda har bir cho'qqilarni tashkil qilib, ularni aylanib o'tish kerak bo'ladi. Lekin biz yonma-yon izlash algoritmini tanlaymiz. Chunki ushbu grafda sikllar mavjud bo'lib, unda ichkarilab izlash algoritmini qo'llash mumkin emas. Faqatgina bu erda yonma-yon izlash algoritmini ikki tomonlama amalga oshiramiz, ya'ni boshlang'ich va oxirgi holatlardan yurishni boshlaymiz. Ular uchrashib qolishsa, demak yechim mavjud bo'ladi.

Endi dasturda qo'llaniladigan o'zgaruvchilar keltiramiz:

```
Type GameState = array[0..2, 0..2] of integer;
{-o'yin maydonining holati}
Type Vertex = record   {graf uchlari}
State : GameState;   {o'yin maydoni}
PrevVertex : Integer;
{-joriy uchga kelturuvchi uch nomeri}
end;
var
StartingState : GameState;      {boshlang'ich holat}
Neighbours: array[0..3] of GameState;
{-qo'shni holatlar massivi}
L : array[0..2999] of Vertex;
{-ko'rilayotgan uchlar ro'yxati}
TailIdx : Integer;
{- ro'yxatdagi oxirgi element ko'rsatkichi }
faylout:text;
```

Bu erda L ning uzunligi 3000 gacha deb olingan, bu esa ba'zi-bir boshlang'ich holatlar uchun yetarli emas, natijada massiv chegarasidan chiqib ketish holati yuzaga kelishi mumkin. Lekin, biz uchun, algoritmni tushunib olish uchun yetarli hisoblanadi.

Endi boshlang'ich holatni fayldan o'qib kiritamiz. Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
4 1 3 2 0 6 7 5 8	1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 0 8 1 2 3 4 0 6 7 5 8 1 0 3 4 2 6 7 5 8 0 1 3 4 2 6 7 5 8 4 1 3 0 2 6 7 5 8 4 1 3 2 0 6 7 5 8 Holatlar soni: 632

Bu yerda *input.txt* faylida massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida natijaviy holatga o'tishdagi har

bir qadamdagi holat chop etiladi va oxirida barcha ko'rib chiqilgan holatlar soni chop etiladi. E'tibor bering, natijalar oxirgi holatdan boshlanib chop etiladi.

Endi kerakli funksiyalarni ishlab chiqamiz. Oldiniga ma'lumotlarni kiritish funksiyasini yaratamiz:

```
Procedure Initialize;
var
i,j:integer;
fayl: text;
begin
      Assign(fayl, 'input.txt');
      Reset(fayl);
            for i:=0 to 2 do
begin
      for j:=0 to 2 do
      read(fayl, StartingState[i,j]);
      Readln(fayl)
end;
      close(fayl);
end;
```

Har doim joriy uchga qo'shni bo'lgan uchlarni bilish talab qilinadi, shu bois *GetNeighbours*() funksiyasini kiritamiz. Bu erda *Neighbours* massivi to'ldiriladi va funksiya qo'shni uchlar sonini qaytaradi, ular 4 tadan oshmaydi.

```
Function GetNeighbours(s : GameState) : integer;
var i, j, k, zi, zj, idx : integer;
const
di :  array[0..3] of integer = (-1, 0, 1, 0) ;
dj :  array[0..3] of integer = (0, -1, 0, 1) ;
begin
 for i:=0 to 2 do
 for j:=0 to 2 do
 if s[i,j]=0 then    {bo'sh katakni topamiz}
 begin
  zi:=i;   {va katak koordinatalarini saqlaymiz}
     zj:=j;
 end;
 idx:=0;  {joriy qo'shni katakning tartib raqami}
```

```
 for k:=0 to 3 do
 begin
       i:=zi+di[k];
{demak, i,j  - bo'sh katakga qo'shni katak  }
      j:=zj+dj[k];    {koordinatalari}
       if (i>=0) and (j>=0) and (i<=2) and (j<=2) then
  begin
{qo'shni katak o'yin maydonida bo'lishi kerak}
        Neighbours[idx]:=s;
{ Neighbours massiviga navbatdagi elementni kiritamiz}
    Neighbours[idx][i,j]:=0;          {bo'sh katak}
    Neighbours[idx][zi,zj]:=s[i,j];
{-qo'shni katak bilan almashtirildi}
    idx:=idx+1;
  end;
 end;
 GetNeighbours:=idx; {qo'shni uchlar soni}
end;
```

Har bir yangi holat bitta yurish bilan hosil bo'ladi. Yurishlar soni 2 tadan kam va 4 tadan ortiq bo'lmaydi. Ushbu yurishlar kubikni chapga, o'ngga, yuqoriga va pastga siljitish bilan amalgam oshiriladi. Yuqorida keltirilgan *GetNeighbours*() funksiyasi bo'sh katak atrofidagi kataklarni o'rganib chiqadi va har safar bo'sh katakni o'rnini almashtirib yurish qiladi, yani bo'sh katak bilan yonidagi kataklar o'rin almashishadi va natijada har safar yangi holat yuzaga keladi.

Maqsadga erishilganligimizni *IsGoal*() funksiyasi tekshirib turadi, uning kodi quyidagicha bo'ladi:

```
Function IsGoal(s : GameState) : Boolean;
{ s holatni yechimga  mosligini tekshiramiz}
var i, j: integer;
const
goal  :  GameState  = ((1,2,3), (4,5,6), (7,8,0)) ;
{-bu  yechim}
begin
      for i:=0 to 2 do
 for j:=0 to 2 do
 if s[i,j] <> goal[i,j] then
```

```
{-qaysidir katak mos kelmadi}
begin
       IsGoal:=false;
      Exit;
end;
 IsGoal:=true;
{-bu erda, s to'liq  yechimga  mos keldi}
end;
```

Endi holatni faylga chiqarishni tashkil qilamiz, buning uchun oldiniga *s* massivini satrga aylantiramiz:

```
Function StateToString(s : GameState) : String;
var
 i, j: integer;
      r : string;
begin
      r:= '' ;
      for i:=0 to 2 do
  begin
    for j:=0 to 2 do
    r:=r+IntToStr( s[i,j] ) + ' ';
    r:=r+chr( 13 ) + chr( 10 ) ;
{-qatorning oxiridan yangi qatorga o'tish belgilarini } {qo'ydik}
  end;
  StateToString:=r;
end;
```

Aniqlangan *s* massivini satrga aylantirib, qulay bo'lishi uchun uni matritsa ko 'rinishida chop etish zarur.

Endi asosiy dasturni keltiramiz:

```
Program Game8;
Type GameState = array[0..2, 0..2] of integer;
{-o'yin maydonining holati}
Type Vertex = record   {graf uchlari}
State : GameState;   {o'yin maydoni}
PrevVertex : Integer;
{-joriy uchga kelturuvchi uch nomeri}
end;
```

```
var
StartingState : GameState;      {boshlang'ich holat}
Neighbours: array[0..3] of GameState;
{-qo'shni holatlar massivi}
L : array[0..2999] of Vertex;
{-ko'rilayotgan uchlar ro'yxati}
TailIdx : Integer;
{ -ro'yxatdagi oxirgi element ko'rsatkichi }
faylout:text;
function IntToStr(n:integer):string;
var s : string;
begin
      str(n,s);
      inttostr:=s;
end;
Procedure Initialize;
var
      i,j:integer;
      fayl: text;
begin
      Assign(fayl, 'input.txt');
      Reset(fayl);
      for i:=0 to 2 do
begin
      for j:=0 to 2 do
      read(fayl, StartingState[i,j]);
      Readln(fayl)
 end;
      close(fayl);
end;
Function GetNeighbours(s : GameState) : integer;
var i, j, k, zi, zj, idx : integer;
const
      di :  array[0..3] of integer = (-1, 0, 1, 0) ;
      dj :  array[0..3] of integer = (0, -1, 0, 1) ;
begin
      for i:=0 to 2 do
      for j:=0 to 2 do
      if s[i,j]=0 then          {bo'sh katakni topamiz}
```

```
begin
      zi:=i;      {va katak koordinatalarini saqlaymiz}
      zj:=j;
end;
idx:=0;  {joriy qo'shni katakning tartib raqami}
      for k:=0 to 3 do
begin
      i:=zi+di[k];
      {-demak, i,j  - bo'sh katakga qo'shni katak  }
      j:=zj+dj[k];    {koordinatalari}
      if (i>=0) and (j>=0) and (i<=2) and (j<=2) then
begin   {qo'shni katak o'yin maydonida bo'lishi kerak}
      Neighbours[idx]:=s;
{ Neighbours massiviga navbatdagi elementni kiritamiz}
      Neighbours[idx][i,j]:=0;          {bo'sh katak}
      Neighbours[idx][zi,zj]:=s[i,j];
      {-qo'shni katak bilan almashtirildi}
      idx:=idx+1;
end;
end;
      GetNeighbours:=idx; {qo'shni uchlar soni}
end;
Function IsGoal(s : GameState) : Boolean;
{ s holatni yechimga  mosligini tekshiramiz}
var i, j: integer;
const
goal : GameState = ((1,2,3),(4,5,6),(7,8,0)); {-bu  yechim}
begin
      for i:=0 to 2 do
      for j:=0 to 2 do
      if s[i,j] <> goal[i,j] then
      {-qaysidir katak mos kelmadi}
begin
      IsGoal:=false;
      Exit;
end;
      IsGoal:=true;
      {-bu erda, s to'liq  yechimga  mos keldi}
end;
```

```
Function StateToString(s : GameState) : String;
var
      i, j: integer;
      r : string;
begin
      r:= '' ;
      for i:=0 to 2 do
begin
      for j:=0 to 2 do
      r:=r+IntToStr( s[i,j] ) + ' ';
      r:=r+chr( 13 ) + chr( 10 );
{qatorning oxiridan yangi qatorga o'tish belgilarini } {qo'ydik}
end;
      StateToString:=r;
end;
Procedure Solve;
var
      HeadIdx : integer; {L ro'yxatining ko'rsatkichi }
      N : integer; {qo'shni holatlar soni}
      i : integer;  { sikl o'zgaruvchisi}
      v : VerTex; {joriy ko'riladigan cho'qqi}
      c : integer; {ko'rib chiqilgan cho'qqilar soni}
begin
L[0].State:=StartingState;
{-L ro'yxatiga boshlang'ich cho'qqini joylshtiramiz}
L[0].PrevVertex:=-1;
{-boshlang'ich cho'qqidan oldinroq cho'qqi yo'q}
      HeadIdx:=0;
      TailIdx:=1;
      c:=0;
repeat
      v:=L[HeadIdx]; { v - ro'yxatning birinchi elementi}
      c:=c+1;
      If IsGoal(v.State) Then
{Agar joriy cho'qqi yechim bo'lsa}
begin
   {u holda natijani faylga chiqaramiz}
    Writeln(faylout, StateToString(v.State));
   {endi qadamba-qadam cho'qqilarini ko'rib chiqamiz}
```

```
repeat
      v:=L[v.PrevVertex];
    Writeln(faylout, StateToString(v.State));
Until v.PrevVertex=-1;
{-boshlang'ich cho'qqiga yetib keldik}
   Writeln(faylout, 'Holatlar soni: ', c);
   Exit;
end;
     n:=GetNeighbours(v.State);
     {-qo'shni cho'qqilarni aniqlaymiz}
for i:=0 to n-1 do {va ularni L ro'yxatiga kiritamiz}
begin
      L[TailIdx].state:=Neighbours[i];
      L[TailIdx].PrevVertex:= HeadIdx;
      TailIdx:= TailIdx+1;
end;
     HeadIdx:= HeadIdx+1; {ro'yxatni bittaga oshiramiz}
until HeadIdx= TailIdx; {ro'yxat tugamangunga qadar}
     writeln(faylout, ' Yechim topilmadi');
end;
{dasturning asosiy qismi}
begin
{chiqarish faylini ochamiz}
     assign(faylout,'output.txt');
     rewrite(faylout);
     Initialize;
     Solve;
     close(faylout);
end.
```

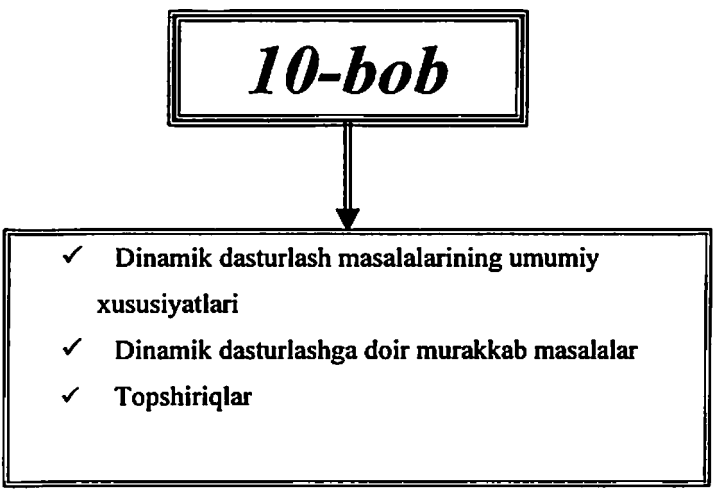
Topshiriqlar

1. To'yga N ta mehmon taklif etilgan. Agar i –mehmon j-mehmon bilan tanish bo'lsa $A[N,N]$ massivida $A[i,j]=A[j,i]=1$ deb kiritamiz, aksincha esa $A[i,j]= A[j,i]=0$ bo'ladi. Mehmonlarni ikki guruhga shunday taqsimlash kerakki, undagi mehmonlar bir-birini tanimaydigan bo'lishsin.

2. Informatikadan o'tkaziladigan olimpiadaga N ta talaba taklif etildi. Ularning ba'zi-birlari o'zaro tanish. Bilvosita, ya'ni umumiy tanish talaba orqali, barcha talabalarni bir-biri bilan tanishtirish mumkinligini aniqlang.
3. Daraxtning berilgan 2 ta uchining eng yaqin umumiy ajdodini aniqlang.
4. Shaharlar o'zaro yo'llar bilan bog'langan, ya'ni istalgan shahardan istalgan shaharga o'tish mumkin. Har bir yo'ldan o'tkazilishi mumkin bo'lgan yukning eng katta og'irligi berilgan. Birinchi shahardan boshqa barcha shaharlarlargacha olib o'tish mumkin bo'lgan eng katta yuk og'irligini aniqlang.
5. Shahardagi ko'chalar bir-biriga perpendikulyar qilib qurilgan. Ko'chalar berk bo'lishi ham mumkin. Berilgan ikki nuqta orasidagi eng yaqin yo'lni aniqlang.
6. Labirintdan chiqishni aniqlashda eng kam burilishlar bo'lgan yo'lni aniqlang.
7. Labirintda devor ichidan o'tishga bir marotaba ruxsat berilsa, undan chiqishning eng qisqa yo'lini aniqlang.
8. Tekislikda N ta nuqta koordinatalari bilan berilgan. Ularni o'zaro kesmalar bilan shunday ulash kerakki, istalgan nuqtadan istalgan nuqtagacha o'tish mumkin bo'lsin va kesmalar yig'indisi minimal qiymat qabul qilsin.
9. Natural sonlardan tashkil topgan $N \times M$ jadvalda har bir katakcha jarimani bildiradi. Jadval diagonalining bir uchidan ikkinchi uchigacha shunday o'tish kerakki jarimalar yig'indisi minimal bo'lsin.
10. Shaxmat taxtasida 5 ta *farzinni* shunday joylashtirish kerakki, ular barcha kataklarni o'z nazoratlariga olishsin.

10-bob. DINAMIK DASTURLASHGA DOIR MISOL VA MASALALAR

Dinamik dasturlashga doir masalalar juda murakkab bo'lib, bu erda masalaning optimal yechimini topish zarur bo'ladi va bunda masalani yechish bosqichma - bosqich amalga oshiriladi. Dinamik dasturlash juda kuchli usullardan bo'lib, uni tartiblangan ko'rinishga ega bo'lgan tarkiblarni optimallashtirishda qo'llash samarali hisoblanadi. Ushbu usulni amalga oshirish qiyin bo'lmasada, uni o'rganib olish ko'pchilik uchun qiyinchilik tug'diradi. Shu bois mazkur bobda dinamik dasturlashni qo'llashga doir ko'pgina misollar keltirilgan:



10.1. Dinamik dasturlash masalalarining umumiy xususiyatlari

Ko'pgina masalalarni yechishda biz masalani kichik masalalarga taqsimlab, ularning har birining yechimini aniqlab keyin umumiy masalaning yechimini shakllantiramiz. Kichik masalalarning yechimlari jadvalda saqlanib borilsa, ularni bir marotaba hisoblash kifoya bo'ladi. Masalani bunday yechish dinamik dasturlash usullariga mansub. Dinamik dasturlash – bu joriy natijalarni saqlab turish orqali rekursiya algoritmlarini samaraliroq amalga oshiruvchi usul hisoblanasdi. Ya'ni rekursiya jarayonida olinadigan natijalarni maxsus jadvalda saqlab, keyinchalik har bir qadamda rekursiyadan emas, balkim birinchi navbatda jadvaldan, oldinroq hisoblangan qiymatlarni, olishni taqozo etadi. Ushbu usulning mohiyatini to'liq anglab olish uchun quyidagi misollarni ko'rib chiqamiz.

Zinapoya. Har bir qadamda bitta zinaga yoki birdan ikki zinaga chiqish mumkin. Nechta usul bilan zinapoyaning 10-zinasiga yetib olish mumkinkiligini aniqlang.

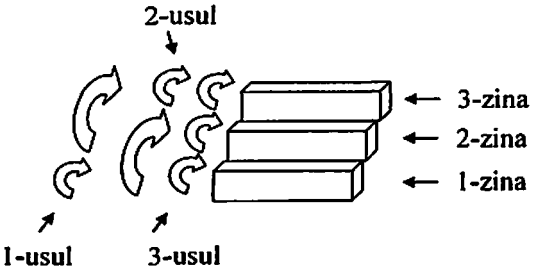
Masala yechimi. Bu erda i-zinaga chiqish sonini $K(i)$ deb belgilaymiz, e'tibor bering, bizdan faqatgina chiqish sonini aniqlash talab etilmoqda, barcha yo'llarni emas. Shunday qilib, bizga $K(10)$ qiymatini aniqlash talab etiladi. Masalaning sharti bo'yicha 10-zinaga 8-zinadan yoki 9-zinadan chiqish mumkin, demak

$$K(10) = K(8) + K(9).$$

Haqiqatan, 8-zinaga chiqish yollari, 9-zinaga chiqish yollari bilan mos emas va ulardan 10-zinaga chiqish usullari soni birga teng, demak 10-zinaga chiqish usullari soni quyidagi yig'indiga teng bo'ladi

$$K(10) = 1 \cdot K(8) + 1 \cdot K(9) = K(8) + K(9).$$

Tekshirish uchun $K(3)$ ni ko'ramiz.



Ushbu rekurrent formula istalgan *i*-zina uchun ham bajariladi, ya'ni

$$K(i) = K(i - 2) + K(i - 1).$$

Endi $K(1)$ va $K(2)$ larni aniqlaymiz, masalaning sharti bo'yicha va yuqoridagi chizmadan 1-zinaga chiqish usuli bitta, ya'ni $K(1) = 1$ va 2-zinaga chiqish usuli ikkita, ya'ni $K(2) = 2$ bo'ladi.

Demak, kichik masalalar natijalarini saqlash uchun 10 ta elementdan iborat massiv kifoya bo'ladi. Shundan kelib chiqqan holda, quyida ushbu g'oyani dasturiy shaklini taklif qilamiz:

```
K[1] := 1;
K[2] := 2;
For i := 3 to 10 do
K[i] := K[i - 1] + K[i - 2];
```

E'tibor bering, *K*[] massivining bir o'lchamli bo'lishi kichik masalani aniqlovchi funksiyaning argumentlar soniga teng, massivdagi elementlar soni esa argumentning qabul qilishi mumkin bo'lgan qiymatlar soniga teng bo'ladi. Dasturning umumiy ko'rinishi quyidagicha bo'ladi:

```
program Zinapoya;
var
    i: integer;
    K: array[1..10] of integer;
begin
  K[1] := 1;
  K[2] := 2;
  For i := 3 to 10 do
begin
    K[i] := K[i - 1] + K[i - 2];
end;
  Write('Usullar soni: ', K[10]);
end.
```

Shunday qilib, zinapoyaning 10-zinasiga yetib olishning 89 ta usuli mavjud ekan.

Maksimal ketma-ketlik. Berilgan $A[N]$ butun sonli massivdan teng sonlar ketma-ketligining maksimal uzunligini aniqlang.

Masala yechimi. Bu erda $L[i]$ massivini kiritamiz. Berilgan $A[]$ massivining i-indeksigacha bo'lgan eng uzun ketma-ketlik sonini $L[i]$ deb olamiz. Endi, qanday qilib $L[i+1]$ ni hisoblash mumkinligini ko'rib chiqamiz. Agar $A[i+1] = A[i]$ bo'lsa, u holda $L[i+1]=L[i]+1$ bo'ladi. Aks holda, ya'ni $A[i+1] \neq A[i]$ bo'lsa, $L[i+1]$ birga teng bo'ladi, ya'ni $L[i+1] = 1$, chunki $A[i+1]$ elementi oxirgi ketma-ketlikda bitta bo'ladi. Shunday qilib, $L[i]$ larni hisoblab olamiz va ularning eng kattasi masala javobi bo'ladi.

Quyidagi misol uchun $A[]=\{1,4,4,4,11,3,3,3,3,1,1,4,4\}$ natija $\{3,3,3,3\}$ bo'lishi kerak.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
13 1 4 4 4 11 3 3 3 3 1 1 4 4	4

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va ikkinchi qatorda massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida maksimal uzunlik qiymati chiqarilgan bo'ladi.

Dasturning umumiy ko'rinishi quyidagicha bo'ladi:

```
program EngUzun;
const Nmax=100;
var
      i, n, IndL : integer;
      L, A: array[1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl,n);
      for i:=1 to n do
      read(fayl, A[i]);
{-sikl orqali A[] qiymatlarini kiritamiz}
      close(fayl);
```

```
      L[1]:=1;
      For i:=2 to N do
begin
      if A[i-1]=A[i]  then
      L[i]:=L[i-1]+1
      else
      L[i]:=1;
end;
      IndL:=1;
      For i:=2 to N do
begin
      if L[i]>L[IndL]  then
      IndL:=i;
end;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      Write(fayl, L[IndL]);
{ - Maksimal uzunlikni chop etish}
      close(fayl);
end.
```

Ushbu g'oyani yana bir misolda ko'rib chiqamiz.

Maksimal qism massiv. Berilgan $A[N]$ butun sonli massivdan o'sib borish tartibi bilan joylashgan eng uzun qism massivni aniqlang. Qism massiv elementlari $A[]$ massivida ketma-ket kelishi shart emas.

Masala yechimi. Oldingi misolga asoslanib, $L[i]$ massivini quyidagicha kiritamiz. Berilgan $A[]$ massivining i-indeksi bilan tugaydigan eng uzun qism massivining uzunligini $L[i]$ deb olamiz. $L[i]$ ning qiymati undan oldingi qiymatlardan, ya'ni $L[j]$, $j=1,...,i-1$ larning biri uchun, agar $A[j]<=A[i]$ bajarilsa, u holda $L[i]$ bittaga ko'p bo'ladi. Buning ma'nosi, $A[i]$-element qism ketma-ketlikni kengaytirishi mumkin. Shunday qilib, $L[i]$ larni hisoblab olamiz va ularning eng kattasi masala javobi bo'ladi.

Quyidagi misol uchun $A[]=\{1,4,2,7,11,3,5,8,13\}$ natija $\{1,2,3,5,8,13\}$ bo'lishi kerak, ya'ni eng uzun qism massivning uzunligi 6 ga teng bo'ladi. Quyidagi jadvalda L[] massivining har bir qadamdagi qiymatlari keltirilgan:

Qadam	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
1	1	1	1	1	1	1	1	1	1
2	1	2	1	1	1	1	1	1	1
3	1	2	2	1	1	1	1	1	1
4	1	2	2	3	1	1	1	1	1
5	1	2	2	3	4	1	1	1	1
6	1	2	2	3	4	3	1	1	1
7	1	2	2	3	4	3	4	1	1
8	1	2	2	3	4	3	4	5	1
9	1	2	2	3	4	3	4	5	6

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
9 1 4 2 7 11 3 5 8 13	6

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami va ikkinchi qatorda massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida maksimal qism uzunlik qiymati chiqarilgan bo'ladi.

Dasturning umumiy ko'rinishi quyidagicha bo'ladi:

```
program EngUzunQism;
const Nmax=100;
var
      i, j, n, IndL : integer;
      L, A: array[1..Nmax] of integer;
      fayl : text;
begin
{massiv o'lchamini va qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl,n);
      for i:=1 to n do
```

```
      read(fayl, A[i]);
{-sikl orqali A[] qiymatlarini kiritamiz}
      close(fayl);
      For  i:=1 to N do
      L[i] := 1;
      For  i:=2 to N do
begin
      For  j:=1 to i-1 do
      if (A[j]<=A[i])and(L[i]<=L[j])then
      L[i]:=L[j]+1;
end;
      IndL:=1;
      For  i:=2 to N do
begin
      if L[i] > L[IndL]  then
      IndL:=i;
end;
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      Write(fayl, L[IndL]);
{ - Maksimal qism uzunlikni chop etish}
      close(fayl);
end.
```

Shunday qilib, dinamik dasturlash usuli optimallashtirish masalarida qo'llaniladi. Umumiy holda ushbu toifadagi masalaning yechimi bir nechta bo'lishi mumkin, ulardan esa faqatgina optimal yechimni tanlash kerak bo'ladi.

Yuqoridagi misollarda nega biz shu tariqa yo'l tutdik? Nega ushbu usulda olingan natija samarali bo'ladi? Chunki ushbu toifaga mansub masalalar ma'lum-bir xususiyatlarga ega bo'lishadi va biz ko'rib chiqqan misollarda ushbu xususiyatlar namoyon bo'lgan.

Dinamik dasturlashga oid algoritmlarni yaratishda quyidagi xususiyatlarni hal etish talab etiladi:

1. optimal yechimlarni tarkibini yoritish;
2. kichik masalalarni rekurrent formula bilan o'zaro bog'lash;
3. pastdan yuqoriga qarab kichik masalalarning optimal yechimlarini hisoblash;

4. olingan natijalarga asoslanib umumiy masalaning optimal yechimiga qaysi holatlarga erishish mumkinligini aniqlash.

Har bir bandni qisqa ko'rib chiqamiz.

1. Ko'riladigan masala optimallik xususiyatiga ega bo'lishi uning kichik masalalarining optimal yechimga ega bo'lishidadir. Masalalada ushbu xususiyat mavjud bo'lgan holdagina dinamik dasturlashni qo'llash mumkin. Kichik masala uchun optimallik aniqlangan bo'lsa, bevosita algoritmda qanday ko'rsatkichlar to'plamini yaratish ayon bo'ladi.

2. Kichik masalalarning o'zaro bog'liqligi bevosita rekurrent formula yordamida berilib, kichik masalalarning to'plami katta bo'lmasligini talab qiladi. Natijada rekurrent formula bilan ularni bog'lab bir xil ko'rinishdagi kichik masalani yechimini aniqlash kifoya bo'ladi. Ko'pgina masalalarda kichik masalalar soni juda ko'p bo'ladi. Dinamik dasturlashda ushbu kichik masalalar bir marotaba yechilib, natijasi maxsus jadvalda saqlanadi. Rekursiyali algoritmlarda esa har bir qadamda paydo bo'ladigan kichik masalalar yana boshqatdan hisoblanadi. Shu bois rekursiyali algoritmlar ko'p vaqt talab qiladi.

3. Pastdan yuqoriga qarab harakatlanishda biz bevosita kichik masalalarning ketma-ket yechimlari aniqlanib, so'ng yechimlar jadvalda saqlanib boriladi. Jarayon davom etilganda, oldinroq ko'rib chiqilgan kichik masalaga yana duch kelinsa, uning optimal yechimi bevosita jadvaldan olinadi.

Ba'zi-bir hollarda yuqoridan pastga qarab optimal yechimni tezroq aniqlash mumkin bo'ladi.

4. Olingan natijalarga asoslanib umumiy masalaning optimal yechimiga qaysi holatlarga erishish mumkinligini aniqlash uchun ushbu optimal yechim qaysi qiymatlarda erishilganini aniqlash yetarli bo'ladi.

10.2. Dinamik dasturlashga doir murakkab masalalar

Aniq misollar bilan tanishishni davom etamiz.

Uchburchakdagi sonlar. Uchburchak shaklida joylashgan butun sonlarni yuqoridan pastga qarab, bu yerda har bir qatordan bitta son olib, ularning yig'indisi hisoblanadi.

```
        7
    3       8
8       1       0
```

```
      2     7     4     4
4     5     2     6     5
```

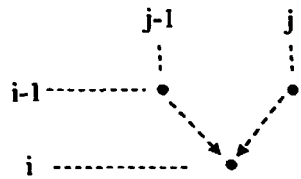
Bu yerda, pastga tushishda chapdagi yoki o'ngdagi songa o'tish mumkin. Kiritilgan sonlar 0 va 99 oralig'idan bo'lishi kerak. Uchburchakdagi qatorlar soni 100 dan kam.

Qaysi yo'ldan yursak, yig'indi maksimal qiymatga ega bo'lishini aniqlang.

Masala yechimi. Uchburchakdagi sonlarni *D*[] massivda quyidagicha joylashtiramiz:

```
7 0 0 0 0
3 8 0 0 0
8 1 0 0 0
2 7 4 4 0
4 5 2 6 5
```

Endi uchburchak bo'yicha qanday harakatlanishni, ya'ni maksimal qiymatga chiqish yo'lini aniqlash uchun *Ray*[1..*MaxN*,0..*MaxN*+1] massivini kiritamiz. Uning elementlarini aniqlash uchun quyidagi sxemaga e'tibor beramiz:



Masalaning sharti bo'yicha (*i*,*j*) nuqtaga ikki joydan, ya'ni (*i*-1,*j*-1) va (*i*-1,*j*) nuqtalaridan kelishimiz mumkin. Qaysi biri maksimal bo'lsa, uni qabul qilamiz. Natijada *Ray*[] massivi elementlari quyidagicha hisoblanadi:

Ray[1,1]=D[1,1]

Ray[i,j]=max(D[i,j]+Ray[i-1,j],D[i,j]+Ray[i-1,j-1])

bu erda $i=2..N$ va $j=1..i$.

Bizdagi uchburchakdagi qiymatlar uchun, *Ray*[] massivi quyidagi ko'rinishda bo'ladi:

0	7	0	0	0	0	0
0	10	15	0	0	0	0
0	18	16	15	0	0	0

0	20	25	20	19	0	0
0	24	30	27	26	24	0

Endi, hosil bo'lgan *Ray*[] massivining oxirgi qatoridan eng katta elementni aniqlaymiz xolos.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 7 0 0 0 0 3 8 0 0 0 8 1 0 0 0 2 7 4 4 0 4 5 2 6 5	5<--7<--8<--3<--7

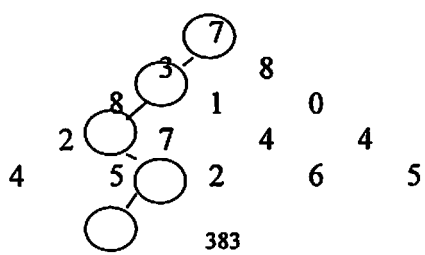
Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=5, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida maksimal yig'indini tashkil qiluvchi sonlar chiqarilgan bo'ladi.

Dasturning umumiy ko'rinishi quyidagicha bo'ladi:

```
program EngUzunQism;
const MaxN=100;
var
      i, j, Lmax, n : integer;
      D,Ray: array[1..MaxN, 1..MaxN] of integer;
      fayl : text;
function Max(a,b: integer) : integer;
begin
if a>b then Max:=a else Max:=b
end;
begin
{massiv o'lchamini va boshqa  qiymatlarni kiritamiz}
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl,  n);
 {sikllar orqali A[] qiymatlarini kiritamiz}
      for i:=1 to n do
begin
      for j:=1 to n do
      read(fayl, D[i,j]);
```

```
      readln(fayl); {keyingi qatorga o'tamiz}
end;
      close(fayl);
      for i:=1 to n do
      for j:=0 to n+1 do  Ray[i,j]:=0;
      Ray[1,1]:= D[1,1];
      for i:=2 to n do
      for j:=1 to i do
begin
      Ray[i,j]:=max(D[i,j]+Ray[i-1,j], D[i,j]+ Ray[i-1,j-1]);
end;
      Lmax := 1;
      For  j:=2 to N do
begin
      if Ray[n,j]> Ray[n, Lmax]  then
      Lmax:=j;
end;
{chiqarish faylini  ochamiz }
      assign(fayl, 'output.txt');
      rewrite(fayl);
{ Endi pastdan yuqoriga qarab yolni chiqaramiz}
      Write(fayl, D[N, Lmax], '<--' );
      For  i:=N-1  downto 2 do
begin
      if Ray[i, Lmax-1] > Ray[i, Lmax]  then
      Lmax:= Lmax -1;{aksincha esa Lmax o'zrarmay qoladi}
      Write( fayl, D[i, Lmax], '<--' );
end;
      Write( fayl, D[1,1] );
      close(fayl);
end.
```

Bu yerda chiqqan 5<--7<--8<--3<--7 natijasini quyidagi sxemada keltiramiz:



Chipta. N ta futbol ishqibozlari chipta olish maqsadida bitta kassa yonida navbatda turishibdi. Har bir kishiga bitta chipta kerak. Navbatning siljishi juda sust. Lekin bir nechta chiptani bir kishi olsa, u holda navbat tezroq siljishi mumkin ekan. Shu bois navbatda turganlar kelishib olishdi, ya'ni ketma-ket turganlar guruhlanib, birinchisiga pulni berib, hamma uchun chipta olishga kelishdilar. Lekin kassir bir kishiga 3 tadan ko'p chipta sotishi mumkin emas ekan. Shu bois navbatda turganlardan 2 kishi yoki 3 kishi o'zaro kelishishlari mumkin bo'ladi.

Navbatda turgan i-ishqibozga bitta chiptani sotishda A_i, ikkita chiptani sotishda B_i va uchta chiptani sotishda C_i vaqt talab etilishi berilgan bo'lsa, navbatda turganlarning barchasiga xizmat ko'rsatish uchun qancha vaqt talab etilishini aniqlash talab etiladi.

Masala yechimi. Masalani oldingi misollar kabi kichik masala yechimidan boshlaymiz va ketma-ket murakkablashtiramiz. Demak, faraz qilamiz, chipta uchun navbatda bir kishi bor. Masalaning sharti bo'yicha bir kishiga bitta chipta kerak, shu bois javobi A_1 bo'ladi. Chunki navbatda undan tashqari hech kim yo'q. Endi navbatda ikki kishi bo'lsin. Ular alohida- alohida chipta xarid qilishlari mumkin, bunda xarid qilish vaqti $A_1 + A_2$ bo'ladi. Agar ular birlashib, chipta xarid qilishsa, u holda masalaning sharti bo'yicha chiptani faqatgina birinchi kishi xarid qilishi mumkin va unga B_1 vaqt talab etiladi. Ikkala variantdan kam vaqt talab qilingan variantni tanlaymiz va uni $D[2]$ da saqlaymiz. Shunday qilib, $D[2]$ – bu ikki kishi tomonidan xaridga ketadigan minimal vaqt. Endi navbatda 3 kishi bo'lishini ko'ramiz. Bu yerda quyidagi variantlar mavjud bo'ladi:

1) Uchinchi shaxs chiptani mustaqil xarid qiladi. Demak, birinchi va ikkinchi shaxslarning chipta xarid qilishlari mustaqil amalga oshiriladi. Lekin biz bu masalani oldinroq ko'rib chiqqan edik va unda natija $D[2]$ ga eng. Umumiy vaqt esa, bu variantda, $D[2] + A_3$ bo'ladi;
2) Ikkinchi va uchinchi shaxslar birgalikda chiptani xarid qilishsin. Bu variantda natija $B_2 + A_1$ bo'ladi, chunki ikkinchi shaxs ikkita chipta oladi (vaqtdan yutush uchun) va birinchi shaxs chiptani mustaqil xarid qiladi;
3) Uchala shaxs birlashsa, ular C_1 vaqt ichida chipta xarid qilishadi, chunki xaridni birinchi shaxs amalga oshiradi va u uchta chipta oladi.

Uchala variantdan eng kam vaqt talab etadigan natijani $D[3]$ ga kiritib qo'yamiz – bu 3 kishilik navbatda talab etiladigan minimal vaqt bo'ladi.

Yuqoridagi mulohazalarni jadval shaklida ko'rsatib o'tamiz. Masalaning sharti bo'yicha quyidagi qiymatlar bizga berilgan bo'ladi, masalan N=5 va

i	A_i	B_i	C_i
1	5	10	15
2	2	10	15
3	5	5	5
4	20	20	1
5	20	1	1

$D[]$ massivini boshlang'ich qiymatlar bilan to'ldirishni boshlaymiz, avvaliga $D[1]= A_1$, bo'ladi va qolgan elementlar hozircha noma'lum hisoblanadi:

D[1]	*D[2]*	*D[3]*	*D[4]*	*D[5]*
5	??	??	??	??

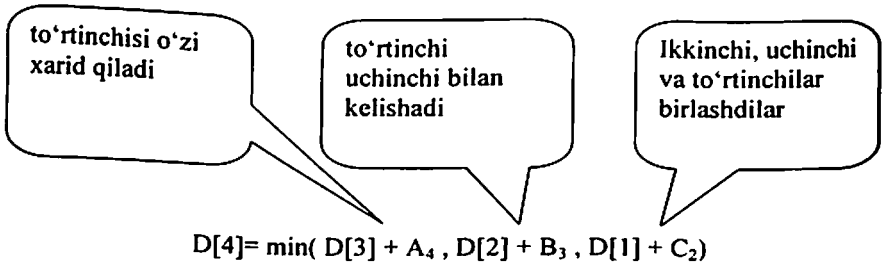
$D[2]$ elementini hisoblaymiz. Yuqorida ko'rilgan variantlarga binoan alohida xarid qilinganda $A_1 + A_2 = 5 + 2 = 7$ bo'ladi va birgalikda xarid qilinganda $B_1 =10$, demak $D[2]= min(7, 10) = 7$ bo'ladi. Shunday qilib, $D[]$ quyidagicha bo'ladi:

D[1]	*D[2]*	*D[3]*	*D[4]*	*D[5]*
5	7	??	??	??

$D[3]$ elementini hisoblaymiz. Navbatga uchinchi kishini qo'shamiz. Agar u alohida xarid qilsa, u holda $D[2] + A_3 = 7+5=12$ vaqt taslab etiladi. Agar u ikkinchi bilan kelishsa, u holda $D[1] + B_2 = 5+10=15$ vaqt taslab etiladi. Agar uchalasi kelishsa, u holda $C_1 = 15$ vaqt taslab etiladi. Demak, $D[3]= min(12, 15, 15) = 12$ bo'ladi. Jadval esa quyidagicha bo'ladi:

D[1]	*D[2]*	*D[3]*	*D[4]*	*D[5]*
5	7	12	??	??

Keyingi $D[4]$ elementini ham shu usulda hisoblaymiz.



Ushbu formulaga qiymatlarni jadvallardan olib qo'yamiz:
D[4]= min(12+20, 7+5, 5+15) = min(32, 12, 20) =12

D[1]	*D[2]*	*D[3]*	*D[4]*	*D[5]*
5	7	12	12	??

Shu tariqa davom ettirib, umumiy holda quyidagi rekurrent formulani hosil qilamiz, ya'ni

$$D[i]= min(D[i-1] + A_i , D[i-2] + B_{i-1} , D[i-3] + C_{i-2})$$

Qo'yilgan masala uchun biz $D[5]$ ni hisoblashimiz kerak, demak
D[5]= min(D[4] + A5 , D[3] + B4 , D[2] + C3)=min(12+20, 12 + 20, 7 + 5) =
min(32, 32, 12) =12.

D[1]	*D[2]*	*D[3]*	*D[4]*	*D[5]*
5	7	12	12	12

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

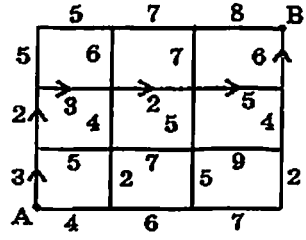
input.txt	*output.txt*
5 5 10 15 2 10 15 5 5 5 20 20 1 20 1 1	12

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=5, ikkinchi qatorda esa A, B, C massivlar elementlari $a[i],b[i],c[i]$ bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida talab etilgan vaqt qiymati chiqarilgan bo'ladi.

Ushbu algoritm dasturining to'liq ko'rinishi quyidagicha bo'ladi:

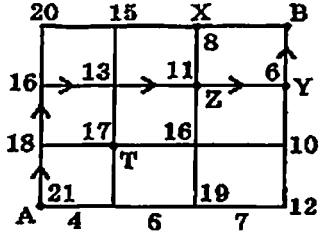
```
program Chipta;
const MaxN=5000;
var
      n : word ; {navbatdagilarning soni}
      a, b, c : array[0..MaxN] of integer;
      d : array[0..MaxN] of longint;
      i : integer;
      f: text;
function MyMin(a,b : longint ):longint;
begin
      if a<b then MyMin:=a else MyMin:=b
end;
begin
      assign(f, 'input.txt');
      reset(f);
      readln(f, n);
      for i:=1 to n do
      readln(f,a[i],b[i],c[i]);
      close(f);
      d[0]:=0;   d[1]:=a[1];
      d[2]:=MyMin(a[1]+a[2], b[1]);
      for i:=3 to n do
d[i]:=MyMin(d[i-1]+a[i],MyMin(d[i-2]+b[i-1],d[i-3]+c[i-2]));
{chiqarish faylini ochamiz }
      assign(f, 'output.txt');
      rewrite(f);
      writeln (f, d[n]);
      close(f);
end.
```

Toshbaqa. Har bir yo'lning vazni vaqtda berilgan quyidagi chizmada toshbaqa A nuqtadan yurishni boshlab B nuqtaga yetib kelishi kerak.



Toshbaqa yuqoriga yoki o'ngga yurishi mumkin. Uning eng qisqa vaqt ichida B nuqtaga yetib kelish yo'lini aniqlang.

Masala yechimi. Yuqorida keltirilgan chizmada ko'rsatilgan yo'l bo'yicha yurilsa 21 birlik vaqt kerak bo'ladi, ya'ni



Lekin barcha yo'llarni aylanib chiqishga to'g'ri keladi. Masalan, n=8 bo'lsa, yo'llar soni 12870 ta bo'ladi, bu esa hisoblash uchun ko'p vaqt talab etadi.

Shu bois, toshbaqaning optimal yo'lini aniqlashni B nuqtadan boshlaymiz. Har bir nuqta uchun, ushbu nuqtadan B nuqtasigacha zarur bo'lgan minimal vaqtni aniqlaymiz. X va Y nuqtalaridan B nuqtasiga o'tadigan minimal yo'l Y nuqtasiga to'g'ri keladi. Z nuqtasidan B nuqtasiga X yoki Y nuqtalari orqali amalga oshirilishi mumkin. Agar X nuqtasi orqali o'tsak 15 birlik vaqt kerak, agar Y nuqtasi orqali o'tsak, unda 11 birlik vaqt kerak bo'ladi. Demak, ulardan eng kichigini olsak, Z nuqtasining vazni 11 bo'ladi. Shu usulda uchlarning vaznlari hisoblanib boriladi. Masalan, agar T nuqtadan B nuqtaga o'tish kerak bo'lsa, bunga 17 birlik vaqt talab etiladi. Masala shartida qo'yilgan A nuqtaning vazni esa 21 birlik bo'ladi.

Yuqorida keltirilgan misol uchun vazn massivini ikkiga bo'lamiz, ya'ni chapdan va pastdan kelishda ketadigan vaqtlar bo'yicha massivlarni quyidagicha kiritamiz va undagi "cheksiz" sonini dasturda 100 deb belgilaymiz:

$$VaznLeft = \begin{pmatrix} 5 & 7 & 8 & \infty \\ 3 & 2 & 5 & \infty \\ 5 & 7 & 9 & \infty \\ 4 & 6 & 7 & \infty \end{pmatrix} \qquad VaznDown = \begin{pmatrix} \infty & \infty & \infty & \infty \\ 5 & 6 & 7 & 6 \\ 2 & 4 & 5 & 4 \\ 3 & 2 & 5 & 2 \end{pmatrix}$$

Bu yerda asosiy muammolardan bittasi bu – massivlarni to'g'ri shakllantirish hisoblanadi. Yuqoridagi chizmadan ko'rinib turibdiki, bevosita har bir tugundan yuqoriga yoki o'ngga yurish mumkin. Demak, kiritilayotgan massivlarni 4x4 deb olamiz, ya'ni *VaznLeft*[1..4, 1..4] va *VaznDown*[1..4, 1..4]. Oldiniga *VaznLeft*[1..4, 1..4] massivini to'ldiramiz:

Massivning birinchi qatoriga izoh berib o'tamiz. Bu yerda *VaznLeft*[1,1]=5, ya'ni [1,1] tugundan [1,2] tugunga o'tish uchun sarflanadigan vaqt, *VaznLeft*[1,2]=7, ya'ni [1,2] tugundan [1,3] tugunga o'tish uchun sarflanadigan vaqt, *VaznLeft*[1,3]=8, ya'ni [1,3] tugundan [1,4] tugunga o'tish uchun sarflanadigan vaqt va *VaznLeft*[1,4]=∞, ya'ni [1,4] tugundan o'ngga o'tish mumkin emas. Xuddi shu yo'l bilan qolgan qatorlarni to'ldirib chiqamiz.

Endi *VaznDown*[1..4, 1..4] massivini to'ldiramiz:

Massivning birinchi ustuniga izoh berib o'tamiz. Bu yerda *VaznDown*[4,1]=3, ya'ni [4,1] tugundan [3,1] tugunga o'tish uchun sarflanadigan vaqt, *VaznDown*[3,1]=2, ya'ni [3,1] tugundan [2,1] tugunga o'tish uchun sarflanadigan vaqt, *VaznDown*[2,1]=5, ya'ni [2,1] tugundan [1,1] tugunga o'tish uchun sarflanadigan vaqt va *VaznDown*[1,1]=∞, ya'ni [1,1] tugundan yuqoriga o'tish mumkin emas. Xuddi shu yo'l bilan qolgan ustunlarni ham to'ldirib chiqamiz.

Har bir nuqta uchun natijani *Vazn*[] massivida saqlaymiz va ushbu massivdan eng yaqin yo'lni *Ray*[] massivida saqlaymiz.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
4 5 7 8 100 3 2 5 100 5 7 9 100 4 6 7 100 100 100 100 100 5 6 7 6 2 4 5 4 3 2 5 2	21 (4,1)->(3,1)->(2,1)->(2,2)->(2,3)->(2,4)->(1,4)

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=5, ikkinchi qatorda esa oldiniga *VaznLeft* va undan keyin *VaznDown* massivlar elementlari bo'sh katak orqali ajratilib ketma-ket kiritiladi va *output.txt* faylida talab etilgan minimal vaqt qiymati va yo'l chiqarilgan bo'ladi.

Dasturning umumiy ko'rinishi quyidagicha bo'ladi:

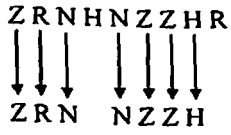
```
program Toshbaqa;
const MaxN=10;
   Infinity=100;
var
      i, j,n,k  : integer;
VaznLeft, VaznDown:array[0..MaxN, 0..MaxN] of integer;
Vazn : array[0..MaxN, 0..MaxN] of integer;
      Ray: array[1..2, 1..MaxN] of integer;
{bu erda marshrutni qatorini 1-da saqlaymiz,  ustunni } {esa 2-da
saqlaymizi}
      Bnuqtasi : boolean;
      fayl : text;
function Min(a,b : longint ):longint;
begin
      if a<b then Min:=a else Min:=b
end;
begin
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, n);
 {chegaralarni cheksiz son bilan toldiramiz}
      for j:=0 to n+1 do
begin
      VaznDown[0,j]:= Infinity; Vazn[0,j]:= Infinity;
      VaznLeft[j,n+1]:= Infinity; Vazn[j,n+1]:= Infinity;
end;
      Vazn[1,4]:=0;
      for i:=1 to n do
begin
      for j:=1 to n do   begin read(fayl, VaznLeft[i,j]);
end;
      readln(fayl);{keyingi qatorga o'tamiz}
end;
      for i:=1 to n do
begin
```

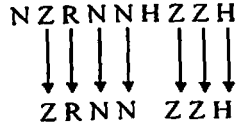
```
      for j:=1 to n do begin   read(fayl, VaznDown[i,j]);
end;
      readln(fayl);{keyingi qatorga o'tamiz}
end;
      close(fayl);
{chiqarish faylini  ochamiz }
      assign(fayl, 'output.txt');
      rewrite(fayl);
{ Vazn massivini toldiramiz}
      Vazn[1,n]:=0; {bu  B nuqtasining vazni }
      for  i:=1 to n do
      for j:=n downto 1 do
begin
      Bnuqtasi:=((i=1) and (j=n));
      if  Not Bnuqtasi  then
Vazn[i,j]:=min(VaznDown[i,j]+Vazn[i-1,j],VaznLeft[i,j]+ Vazn[i,j+1]);
end;
      Writeln(fayl , Vazn[n,1]   );
      {-Minimal vaqtni chop etamiz}
{ Endi yolni aniqlaymiz}
      i:=n; j:=1;
      Ray[1,1]:=n; Ray[2,1]:=1; {bu A nuqtasi}
      k:=1;
{2*n-1 bu B nuqtadan A nuqtagacha bo'lgan qadamlar soni}
while (k <2*n-1) do
begin
      k:=k+1;
      if Vazn[i-1,j]< Vazn[i,j+1] then
begin
      i:=i-1;  Ray[1,k]:=i; Ray[2,k]:=j;
end
      else
begin
      j:=j+1; Ray[1,k]:=i; Ray[2,k]:=j;
end
end;

      Ray[1, 2*n-1]:=1; Ray[2, 2*n-1]:=n; {bu B nuqtasi}
{ Endi yolni chiqaramiz}
{Minimal vaznli yo'l:}
      For  i:=1  to 2*n-2 do
Write(fayl, '(', Ray[1,i], ',',Ray[2,i], ')','->' );
```

```
Write(fayl, '(', 1, ',',n, ')' );
    close(fayl);
end.
```

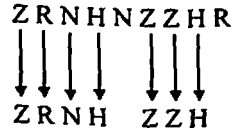
DNK molekulasi. Biologiya fanidan bilamizki, DNK molekulasida genetik axborot mavjud bo'ladi. Faraz qilamiz, DNK faqatgina 4 ta harflardan (*N, Z, R, H*) tashkil topgan bo'lsin. Ikkita DNK berilgan bo'lib, ular uzunligi *M* va *N* bo'lsa, ularning umumiy qismini tashkil qiluvchi qism satrni aniqlang.

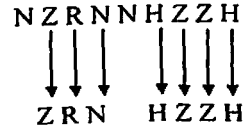
Masala yechimi. Masalan, qyuifagi *ZRNHNZZHR* va *NZRNNHZZH* molekulalar uchun natija 2 ta bo'ladi: *ZRNNZZH* va *ZRNHZZH*, haqiqatan ham, birinchi javob uchun





ikkinchi javob uchun esa



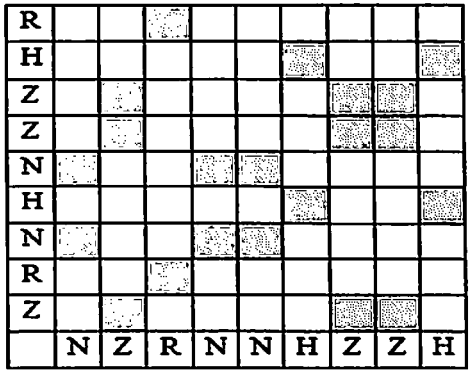


Bu yerda keltirilgan yechim Nudelman-Vunsh algoritmi asosida yaratilgan. Ushbu algoritmga binoan berilgan ikkita so'zdan quyidagi jadvalni tuzamiz

R									
H									
Z									
Z									
N									
H									
N									
R									
Z									
	N	Z	R	N	N	H	Z	Z	H

E'tibor bering, ustun bo'yicha satrni pastdan yuqoriga qaratib joylashtirdik. Bunday joylashturuv dasturda ham e'tiborga olingan.

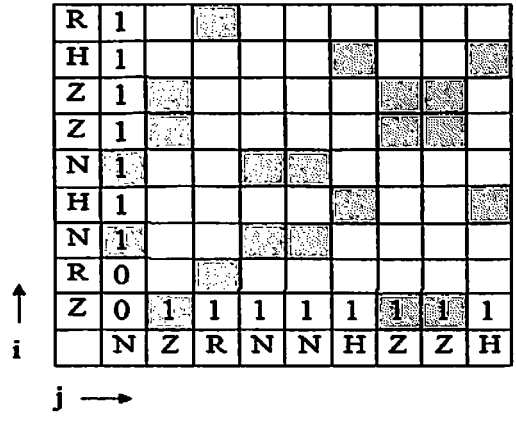
Qator va ustunda bir xil harflar joylashgan bo'lsa, ular kesishgan katakni belgilaymiz.



Endi kataklarni to‘ldirishni boshlaymiz, ya’ni $W[i,j]$ ga quyidagicha qiymat beriladi

$$W[i,j] = \begin{cases} \text{Max}(W[i+1,j], W[i,j-1]), \text{ agar } (i,j) \text{ belgilanmagan bo‘lsa} \\ W[i+1,j-1]+1, \text{ agar } (i,j) \text{ belgilangan bo‘lsa} \end{cases}$$

Ushbu jarayonni boshlashdan oldin birinchi qator va ustunni to‘ldiramiz, bunda birinchi bo‘yalgan katakdan boshlab 1 va ungacha 0 qiymatlar beriladi



Endi $W[i,j]$ larni hisoblab, kataklarni to‘ldiramiz va quyidagi ko‘rinishga kelamiz

R	1	2	3	3	4	5	5	6	7
H	1	2	2	3	4	5	5	6	7
Z	1	2	2	3	4	4	5	6	6
Z	1	2	2	3	4	4	5	5	5
N	1	1	2	3	4	4	4	4	4
H	1	1	2	3	3	4	4	4	4
N	1	1	2	3	3	3	3	3	3
R	0	1	2	2	2	2	2	2	2
Z	0	1	1	1	1	1	1	1	1
	N	Z	R	N	N	H	Z	Z	H

i ↑

j →

Masala shartini qanoatlantiruvchi natijaviy qatorni aniqlash uchun eng katta qiymatdan boshlab, pastga qarab kelamiz. Bu erda katakdan o'tishda ularning qiymati birga farq qilishi kerak va harflar belgilangan kataklardan olinadi.

R	1	2	3	3	4	5	5	6	7
H	1	2	2	3	4	5	5	6	7
Z	1	2	2	3	4	4	5	6	6
Z	1	2	2	3	4	4	5	5	5
N	1	1	2	3	4	4	4	4	4
H	1	1	2	3	3	4	4	4	4
N	1	1	2	3	3	3	3	3	3
R	0	1	2	2	2	2	2	2	2
Z	0	1	1	1	1	1	1	1	1
	N	Z	R	N	N	H	Z	Z	H

i ↑

j →

Algoritmning samaradorlik darajasi $O(nm)$ tartibli bo'ladi. Eng uzun ketma-ketlikning uzunligi esa $W[1,m]$ da saqlanadi.

Bizdagi misolda javoblar quyidagilar bo'ladi: *ZRNNZZH* va *ZRNHZZH* , faqat natijaviy faylda bittasi chiqariladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
9 ZRNHNZZHR 9 NZRNNHZZH	ZRNHZZH

Bu yerda *input.txt* faylida birinchi qatorda 1-DNK ning uzunligi n=9, ikkinchi qatorda esa DNK beriladi, keyingi qatorda 2-DNK ning uzunligi m=9, undan keyin esa DNK beriladi va *output.txt* faylida *DNK* larning umumiy qismini tashkil qiluvchi qism satr chiqarilgan bo'ladi.

Dasturning umumiy ko'rinishi quyidagicha bo'ladi:

```
program DNK;
const MaxN=100;
var i, j,n,m,k : integer;
      W : array[1..MaxN, 1..MaxN] of integer;
      lcsPos : string;
      S1, S2: string;
{bu erda       S1, S2 berilgan satrlar}
      MaxUzun,pos : integer;
      fayl : text;
function Max(a,b : longint ):longint;
begin
      if a<b then Max:=b else Max:=a
end;
begin
      assign(fayl, 'input.txt');
      reset(fayl);
{ 1- DNK uzunligi } readln(fayl, n);
{ 1- DNK } readln(fayl, s1);
{2- DNK uzunligi } readln(fayl, m);
 {2- DNK } readln(fayl, s2);
      close(fayl);
{s1 va s2 boyicha W massivini 1-ustunini toldiramiz}
      k:=n+1;
      for i:=n downto 1 do
begin
      W[i,1] :=0;
```

```
      If s1[n-i+1]=s2[1]  Then
 begin
      k:=i; break;
 end;
 end;
      for i:=1 to k Do W[i,1] :=1;
 {s1 va s2 boyicha W massivini 1-qatorini toldiramiz}
      k:=m+1;
      for j:=1 to m Do
 begin
      W[n,j] :=0;
      If s1[1]=s2[j]  Then
 begin
      k:=j; break;
 end;
 end;
      for i:=k  to  m do W[n,i] :=1;
      for i:=n-1 downto 1 do
      for j:=2 to m do
 begin
      W[i,j] :=Max( W[i+1,j]  , W[i, j-1]) ;
      If s1[n-i+1]=s2[j]  Then
      W[i ,j]:=Max (W[i ,j] ,W[i+1,j-1] +1)
 end;
 {endi yuqoridan orqaga pastga qarab kelamiz}
      MaxUzun := W[1,m];
      pos := MaxUzun;
      i := 1; j:= m;
 while (pos>0 ) do
 begin
      if (s1[n-i+1] = s2[j]) then
 begin
      lcsPos[pos]:=s2[j];
      pos:=pos-1;  i:=i+1;  j:= j-1;
 end
      else
 begin
      if (W[i+1,j] >= W[i,j-1]) then
      i:=i+1
```

```
        else
        j:=j-1
end
end;
{chiqarish faylini ochamiz }
      assign(fayl, 'output.txt');
      rewrite(fayl);
{ endi  lcsPos[] ni chop etamiz   }
      for i:= 1 to MaxUzun do
      write(fayl, lcsPos[i]);
      close(fayl);
end.
```

Toshlar og'irligi. Og'irliklari $t_1,t_2, \ldots, t_n$ bo'lgan toshlar yordamida W vaznli yukni o'lchash mumkinligini aniqlang. Aqar mumkin bo'lmasa, unga pastdan yaqin vaznni chiqaring.

Masala yechimi. Algoritmning g'oyasini quyidagi misolda ko'rib chiqamiz. Agar bizga 5 ta tosh berilgan bo'lsa, ya'ni $n=5$ uchun, $t_1=5$, $t_2=7$, $t_3=9$, $t_4=11$, $t_5=13$ va $W=19$ bo'lsin. Endi $A[1..n, 0..W]$ massivini, uni yechim massivi deb ataymiz, quyidagicha tashkil qilamiz: Qator raqami bu – toshning tartib raqami, ustun raqami esa to'plangan vaznni anglatadi.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1																				
2																				
3																				
4																				
5																				

Dastlab, birinchi ustun bilan, birinchi qatorni to'ldiramiz:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	0	0	0	0	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
2	0																			
3	0																			
4	0																			
5	0																			

Bu yerda nolinchi ustunni nol bilan to'ldirdik, chunki hech qaysi tosh bilan bu vaznni tashkil qilib bo'lmaydi. Birinchi qatorda beshinchi

ustungacha nol qo'ydik, chunki birinchi tosh bilan undan kichik bo'lgan vaznni o'lchab bo'lmaydi. Birinchi qatorning beshinchi ustunidan boshlab 5 bilan to'ldirdik, chunki bizda hozircha bitta tosh bor.

Endi ikkinchi qatorga o'tamiz.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	0	0	0	0	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
2	0	0	0	0	0	5	5	7	7	7	7	7	12	12	12	12	12	12	12	12
3	0																			
4	0																			
5	0																			

Buy erda birinchi tosh qatoridagi vazn bilan ikkinchi toshni qo'shib, qanday vaznni o'lchashni aniqlab olamiz.

Xuddi shunday, uchinchi qatorni to'ldirishda faqatgina ikkinchi qatordagi vaznlarni inobatga olamiz va hokazo, ya'ni har bir qatorni to'ldirishda faqatgina oldingi qator elemenlar qiymatlari yetarli bo'ladi.

Natijada jadvalimiz quyidagi ko'rinishga keladi.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	0	0	0	0	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
2	0	0	0	0	0	5	5	7	7	7	7	7	12	12	12	12	12	12	12	12
3	0	0	0	0	0	5	5	7	7	9	9	9	12	12	14	14	16	16	16	16
4	0	0	0	0	0	5	5	7	7	9	9	11	12	12	14	14	16	16	18	18
5	0	0	0	0	0	5	5	7	7	9	9	11	12	12	14	14	16	16	18	18

Bu yerdan W=19 ni o'lchab bo'lmasligini va unga eng yaqin vazn bu 18 bo'lishini ko'rsa bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 19 5 7 9 11 13	7 11

Bu yerda *input.txt* faylida birinchi qatorda toshlar soni n=5 va W=19, ikkinchi qatorda esa massiv elementlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida W ga eng yaqin bo'lgan toshlar chiqarilgan bo'ladi.

Dasturning umumiy ko'rinishi quyidagicha bo'ladi:

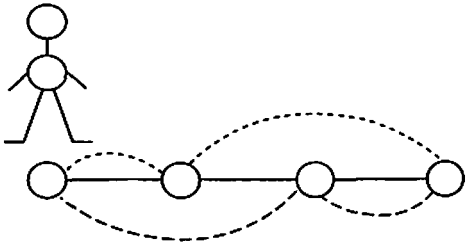
program Toshlar;

```
const MaxN=100;
var
      i, j,n,m,k,W  : integer;
      A : array[1..MaxN, 0..MaxN] of integer;
      T: array[1..MaxN] of integer;
{bu erda   T  berilgan toshlar}
      fayl : text;
{ yechim massivini to'ldiruvchi qism dasur}
Procedure    Solve;
var      i,j:integer;
function Max(a,b : longint ):longint;
begin
      if a<b then Max:=b else Max:=a
end;
begin
      for i:=2 to N Do
      for j:=1 to  W Do
begin
      If j-T[i]>=0   Then
      A[i,j]:=   Max (A[i-1 , j], A[i-1, j-T[i]]+T[i])
      Else  A[i,j] :=A[i-1,j] ;
end;
end;
{yechim qaysi toshlardan tashkil topganligini } {aniqlovchi qism
dastur}
Procedure    Way (i,j: Integer) ;
Begin
      If (i=1) And  (A[i,j]=0)    Then Exit
      Else If i=1 Then
Begin
      Way (i, j-T [i]) ; Write(fayl, T[i],' ');
End
      Else   If A[i,j] <> A[i-1,j]  Then
Begin
      Way (i, j-T[i]);Write(fayl, T[i] ,' ')  ;
end
      Else    Way (i-1, j) ;
end;
{Asosiy dastur}
```

```
Begin
      assign(fayl, 'input.txt');
      reset(fayl);
{Toshlar soni  n  va Yuk vazni W}
      readln(fayl, n, W);
      for i:=1 to n Do    read(fayl, T[i]);
      {-Toshlar  T[i]}
      close(fayl);
{ A  massivini 1-qatorini to'ldiramiz}
      for i:=0 to W Do
      If i<T[1]  Then  A[1,i]:=0 else A[1,i]:= T[1];
{endi qism dasturlarni qo'llaymiz}
       Solve;
{chiqarish faylini  ochamiz }
      assign(fayl, 'output.txt');
      rewrite(fayl);
      Way (n,W);
      close(fayl);
end.
```

Yo'llar soni. N qadam birlikdagi yo'l berilgan. Robotning qadami bir birlikdan to K birlikgacha bo'lishi mumkin. Robotning ushbu yo'lni boshidan to oxirigacha har xil o'tish usullari sonini aniqlang.

Masala yechimi. Masalaning mohiyatini tushunib olish uchun quyidagi qiymatlarni ko'rib chiqamiz: N=3 va K=2. Demak, ushbu yo'lni o'tish uchun quyidagi variantlar mavjud: $1 \to 2$, $1 \to 1 \to 1$ va $2 \to 1$, quyidagi chizmada ushbu yo'llar ko'rsatilgan.



Demak, jami bo'lib 3 ta usul bilan o'tish mumkin ekan. E'tibor bering, 2→2 va birdaniga 3 qadam bilan o'tishlar masala shartini qanoatlantirmaydi.

Bu yerda *i* birlik yo'lni o'tish usullari sonini *S(i)* deb belgilab olamiz. Agar *S*(*i*) aniqlangan bo'lsa, undan *S*(*i*+1) ni hisoblashni o'rganamuiz. Demak, (*i*+1) –o'ringa o'tish quyidagi nuqtalardan mumkin bo'ladi: *i*, *i*-1, ..., *i-k*. Bu esa quyidagi rekurrent formulaga olib keladi:

$$S(i+1) = S(i) + S(i-1) + \ldots S(i-k), \text{ bu erda } i>k.$$

Shu bois, *S*(1), *S*(2), ..., *S*(*N*) birin-ketin hisoblash orqali biz *S*(*N*) qiymatini aniqlab olamiz. Yuqoridagi misol uchun ushbu hisoblashlarni bajarib ko'ramiz.

Bu yerda $j<=k$ gacha bo'lganda *S*(*j*) larni alohida hisoblash kerak bo'ladi. Shu bois, bevosita bitta yo'lni o'tish usullari soni *S*(1)=1, agar yo'llar 2 ta bo'lsa, u holda *S*(2)=2 ga teng bo'ladi, demak *S*(3) o'tish usullari soni teng bo'ladi:
S(1)+ *S*(2)=1+2=3 ga.

Umumiy holda, *N*=13 va *K*=10 bo'lganda

k\n	1	2	3	4	5	6	7	8	9	10	11	12	13
2	1	2	3	5	8	13	21	34	55	89	144	233	377
3	1	2	4	7	13	24	44	81	149	274	504	927	1705
4	1	2	4	8	15	29	56	108	208	401	773	1490	2872
5	1	2	4	8	16	31	61	120	236	464	912	1793	3525
6	1	2	4	8	16	32	63	125	248	492	976	1936	3840
7	1	2	4	8	16	32	64	127	253	504	1004	2000	3984
8	1	2	4	8	16	32	64	128	255	509	1016	2028	4048
9	1	2	4	8	16	32	64	128	256	511	1021	2040	4076
10	1	2	4	8	16	32	64	128	256	512	1023	2045	4088

E'tibor bering, belgilangan kataklarda $S(i)$ ikkiga oshib boryapti. Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
13 10	4088

Bu yerda *input.txt* faylida birinchi qatorda massivning o'lchami n=13 va k=10 va *output.txt* faylida har xil o'tish usullari soni chiqarilgan bo'ladi.

Dasturning umumiy ko'rinishi quyidagicha bo'ladi:

```
program YollarRekursiya;
var
      n,k:integer;
      fayl : text;
{rekursiv funksiya}
function s(i:integer):integer;
var
      r,j,t:integer;
begin
      r:=0;
      if i<0 then s:=0;
      if (i=0) or (i=1) then s:=1;
      if i>1 then
      for j:=i-1 downto i-k do
begin
      r:=r+s(j);
      s:=r;
end;
end;
{Asosiy dastur}
begin
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n, k);
      close(fayl);
{chiqarish faylini ochamiz }
      assign(fayl, 'output.txt');
      rewrite(fayl);
      write(fayl, s(n)); { Yollar soni chop etildi }
```

```
    close(fayl);
end.
```

Ushbu masalaning rekursiyasiz oddiy dasturi quyidagicha bo'ladi:

```
program Yollar;
const Nmax=100;
var
    n,k,j,i,r:integer;
    s:array[1..Nmax] of integer;
begin
    write('n='); readln(n);
    write('k='); readln(k);
    s[1]:=1;
    for i:=2 to k do s[i]:=2*s[i-1];
    for i:=k+1 to n do
begin
    for j:=i-1 downto i-k do
    r:=r+s[j];
    s[i]:=r;r:=0;
end;
    writeln('Yollar soni=', s[n]);
    readln;
end.
```

Yuqoridagi qiymatlar n=13 va k=10 uchun natija 4088 bo'lishini tekshirib ko'ring!

Eng kam pul. Berilgan $M[n]$ massivi tartiblangan $M[1] \le M[2] \le ... \le M[n]$ bo'lib, u pul qiymatlaridir. Berilgan S qiymatni $M[1]$, ... , $M[n]$ lar bilan to'lashda zarur bo'lgan eng kam pul sonlarini aniqlang.

Masala yechimi. Bir qarashda oddiy misolga o'xshaydi, masalan, quyidagi qiymatlar $M[n]$=1, $M[n]$=5, $M[n]$=15 va S=17 berilgan bo'lsin. Bunda, oldiniga eng katta pulni tanlaymiz, ya'ni 15 ni, qoldiqda 17-15=2 qoldi, uni esa ikkita 1 bilan qoplaymiz. Lekin, quyidagi qiymatlar uchun $M[n]$=1, $M[n]$=10, $M[n]$=23 va S=50 natija, ushbu algoritm asosida, 23+23+1+1+1+1 bo'ladi, ya'ni bunda 6 ta pul kerak bo'ladi. Ammo to'g'ri javob 10+10+10+10+10 bo'ladi.

Bu erda quyidagi massivlarni kiritamiz:

1) *A*[] massivi – bu quyidagicha aniqlanadi, agar *i* summasini to'la olsak, u holda *A*[*i*] = 1 bo'ladi, aks holda *A*[*i*] = 0.
2) *Sonlari*[] massivi – bunda *Sonlari*[*i*] ning qiymati bevosita *i* summasini olish uchun zarur bo'lgan *M*[] elementlarining minimal sonidir.
3) *OldingiSumma*[] massivi – bunda *OldingiSumma*[*i*] bu oldinroq hisoblangan yig'indi bo'lib, unga bitta elemetni qo'shish orqali *i* summasi hosil bo'ladi.
4) *B*[] massivi – bu *OldingiSumma*[] massivining nusxasi hisoblanadi.

Agarda *A*[*S*]=1 bo'lsa, demak *S* qiymatni tuzish mumkin bo'ladi va buning uchun *Sonlari*[*S*] ta pul kerak bo'ladi. Ushbu *S* qiymati qaysi pullardan tashkil topgan degan savol tug'iladi. E'tibor bering, *S* dan oldinroq to'plangan yig'indi bu *S'=OldingiSumma*[*S*], demak (*S-OldingiSumma*[*S*]) bu oxirgi qo'shilgan pul bo'ladi. Xuddi shu yo'l bilan qolgan pullar ham aniqlanadi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
3 17 1 5 15	15 1 1

Bu yerda *input.txt* faylida birinchi qatorda pullar soni n=8 va S=32, keyingi qatorda esa pul qiymatlari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida berilgan *S* summani qoplab bo'lmasa '0' ni, aks holda esa *S* ni qoplaydigan pullarni chop etamiz.

Dasturning umumiy ko'rinishi esa quyidagicha bo'ladi:

```
program EngKamPul;
const MaxN=100;
type
arrtype = array[0.. MaxN] of integer;
var
      i, j,n,k,P,S, PS  : integer;
      A, B, M, Sonlari, OldingiSumma: arrtype;
{bu erda    M berilgan pul qiymatlari}
      fayl : text;
begin
{Berilgan ma'lumotlar}
      assign(fayl, 'input.txt');
      reset(fayl);
      read(fayl, n, S);
```

```
      P:=0;
      for i:=1 to n do
begin
      read(fayl,  M[i]); P:= P+M[i];
end;
      close(fayl);
PS:=S; If P>S then PS:=P;{ Kattasini tanladik}
{ MaxN >Max( P,S)  bo'lish kerak}
      A[0]:=1;
      for i:=1 to MaxN do  A[i]:=0;
      {-Qolgan barchasini nolga tenglashtirdik}
      Sonlari[0]:=0;
      for i:=1 to MaxN do Sonlari[i]:=254;
{ya'ni  Sonlari[i]=254 bo'lsa, i summani to'plab } {bo'lmaydi,bunda
254 katta son }
      for i:=1 to MaxN do OldingiSumma[i]:=0;
{Asosiy ishlarni boshlaymiz}
      for i:=1 to N do {barcha pullar bo'yicha sikl}
begin
      for j:=M[i] to PS  do
{ M[i] ishtiroki bilan bo'lgan barcha summalar }
{bo'yicha sikl}
if (A[j-M[i]]<>0) and (Sonlari[j]> Sonlari[j-M[i]]+1)
{ (j-M[i]) summani to'plash mumkin bo'lsa }
{ va  M[i] ni  qo'shib biz j summasini kamroq pullar } {orqali hosil
qilsak }
      then
begin
      Sonlari[j]:= Sonlari[j-M[i]]+1;
{unda bu yangi qiymatni xotirada saqlaymiz va  B[j] da} {ham
saqlaymiz}
      B[j]:=j-M[i];{endi bu oldingi summa bo'ldi}
      A[j]:=1;{va  j summasini tashkil qilsa bo'lar ekan}
end
      else
      B[i]:= OldingiSumma[j]; {aks holda, eski axborotni qoldiramiz }
      for j:=M[i] to PS do
{ OldingiSumma massivida B massivini saqlab qo'yamiz}
      OldingiSumma [j]:=B[j];
end;
```

```
{Endi natijani chiqaramiz, chiqarish faylini ochamiz }
      assign(fayl, 'output.txt');
      rewrite(fayl);
      if A[S]=0 then writeln(fayl, '0')
{- Berilgan S summani qoplab bo'lmaydi}
      else
begin
      i:=S;
{Berilgan S summani quyidagi pullar bilan qoplaymiz}
      for j:= Sonlari[S] downto 1 do
{ Sonlari[S]-bu S ni qoplash uchun zarur bo'lgan}
{ pullar soni}
begin
      write(fayl, i-OldingiSumma[i], ' ');
      i:=OldingiSumma[i];
end;
end;
      close(fayl);
end.
```

Yuqoridagi qiymatlar uchun *Sonlari*[*i*] massivi elementlari quyidagi jadvalda keltirilgan:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
S[i]	0	1	2	3	4	1	2	3	4	5	2	3	4	5	6	1	2	3

Bu yerdan, *S*[17]=3, demak 3 ta pul bilan *S* ni qoplash mumkin. Bunda qaysi pullar kerak bo'lishini aniqlash uchun *OldingiSumma*[] elementlarini jadvalda keltiramiz:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
OldingiSumma[i]	0	0	1	2	3	0	1	2	3	4	5	6	7	8	9	0	1	2

Shunday qilib, birin-ketin pullarni aniqlaymiz:

Qadam	*i*	Formula	Pul
1	*i=S*=17	*17-OldingiSumma[17]*	15
2	*i=OldingiSumma[17]=2*	*2-OldingiSumma[2]*	1
3	*i=OldingiSumma[2]=1*	*1-OldingiSumma[1]*	1

Masofa. Belgilardan iborat $a=(a_1,a_2,...,a_m)$ va $b=(b_1,b_2,...,b_n)$ satrlari uchun $d(a,b)$ masofa tushunchasini quyidagicha kiritamiz. Bittadan harflarni o'chirish, almashtirish va kiritishlar orqali a dan b hosil qilingan qadamlar soni deb $d(a,b)$ qabul qilamiz. Har qanday ikki a va b satrlar uchun $d(a,b)$ sonini aniqlang.

Masala yechimi. Bu yerda quyidagi misolni keltiramiz:

"p" o'chiriladi "g" kiritildi "f" almashtirildi

ptslddf ⟶ tslddf ⟶ tsglddf ⟶ tsglds

Demak $d(ptslddf, tsglds)=3$ ga teng bo'ladi.

Bizga berilgan a va b satrlarni $a[1..m]$ va $b[1..n]$ massivlar deb kiritamiz.

Yangi $d[0..m,0..n]$ massivining elementlarini esa quyidagicha aniqlaymiz:

$$d[i,j] = d(a(1)...a(i), b(1)...b(j)),\ 0<=i<=m,\ 0<=j<=n$$

Ushbu massivning nolinchi qator va ustuni uchun bevosita quyidagilar bajarilgan bo'ladi:

$$d[0,j]=j \quad \text{barcha} \quad 1<=j<=n$$
$$d[i,0]=i \quad \text{barcha} \quad 1<=i<=m$$

Endi $d[i,j]$ larni hisoblash uchun quyidagi rekurrent formulani kiritsak bo'ladi:

$$d[i,j]=min\{d[i-1,j]+1,d[i,j-1]+1,d[i-1,j-1]+P_{ij}\}$$

bu yerda $P_{ij}=1$ agar $a(i)\neq b(i)$ bo'lsa, aks holda $P_{ij}=0$ bo'ladi.

Mazmunan ushbu formulaga izoh beramiz.

$d[i-1,j]$ – bu $a(1)...a(i-1)a(i)$ qatordan $a(i)$ ni olib tashlab, shu bois yana +1, keyin $a(1)...a(i-1)$ dan $b(1)...b(j)$ qatorni tashkil qilish soni.

$d[i,j-1]$ – bu $a(1)...a(i-1)a(i)$ qatordan $b(1)...b(j-1)$ qatorni tashkil qilish soni. Lekin oxiridan $b(j)$ ni qo'shadigan bo'lsak +1.

$d[i-1,j-1]+P_{ij}$ – bu almashtirish, ya'ni a_i ni b_j ga almashtiramiz, agar $a(i)\neq b(i)$ bo'lsa.

Shundan kelib chiqqan holda, masala yechimi bu $d(x,y)=d[m,n]$.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
ptslddf tsglds	3

Bu yerda *input.txt* faylida birinchi qatorda *A* va ikkinchi satrda esa *B* satrlari kiritiladi va *output.txt* faylida masofa qiymatini chop etamiz.

Dasturning umumiy ko'rinishi esa quyidagicha bo'ladi:

```
program Masofa;
const MaxN=100;
var i, j,n,m,Pij : integer;
      d : array[0..MaxN, 0..MaxN] of integer;
      A, B: string;
{bu erda   A, B berilgan satrlar}
      fayl : text;
function Min(a,b : longint ):longint;
begin
      if a<b then Min:=a else Min:=b
end;
begin
{Berilgan ma'lumotlar}
      assign(fayl, 'input.txt');
      reset(fayl);
{A va B qatorlarining uzunligi }
      readln(fayl, A); m:=length(A);
      readln(fayl, B); n:=length(b);
      close(fayl);
{Asosiy ishlarni boshlaymiz}
      for i:=1 to m do d[i,0]:=i;
      for j:=1 to n do d[0,j]:=j;
      for i:=1 to m do
      for j:=1 to n do
begin
      if A[i]<>B[j] then Pij:=1 else Pij:=0;
d[i,j]:=min(d[i-1,j]+1,min(d[i,j-1]+1,d[i-1,j-1]+Pij));
end;
{chiqarish faylini ochamiz }
      assign(fayl, 'output.txt');
      rewrite(fayl);
{ Masofa d[a,b] }
      writeln(fayl, d[m,n]);
      close(fayl);
end.
```

Teng qism satr. Belgilardan iborat $a=(a_1,a_2,...,a_m)$ va $b=(b_1,b_2,...,b_n)$ satrlardan belgilar o'chirilsa, u holda yangi a' va b' qism satrlar hosil

bo'ladi. Bir-biriga teng bo'lgan a' va b' qism satrlarlarning eng uzunini aniqlang.

Masala yechimi. Bu yerda oldingi masalada keltirilgan algoritmni qo'llaymiz.

Bizga berilgan a va b satrlarni $a[1..m]$ va $b[1..n]$ massivlar deb kiritamiz.

Yangi $Uzun[0..m,0..n]$ massivini kiritamiz, unda $Uzun[i,j]$ elementi – bu $a_1,a_2,...,a_i$ va $b_1,b_2,...,b_j$ larning teng bo'lgan qismlarining eng uzuni deb hisoblanadi.

Agar m yoki n nolga teng bo'lsa, u holda a va b satrlarning umumiy qismi bo'lmaydi, shu bois

$$Uzun[0,j]=0 \quad \text{barcha} \quad 0<=j<=n$$
$$Uzun[i,0]=0 \quad \text{barcha} \quad 0<=i<=m$$

Agar $a_i = b_j$ bo'lsa, u holda qism ketma-ketlikning uzunligi oldingisiga nisbatan bittaga oshadi, ya'ni $Uzun[i,j]= Uzun[i-1,j-1]+1$.

Agar $a_i \neq b_j$ bo'lsa, u holda $Uzun[i,j]$ uchta variantdan biri bo'lishi mumkin:

1) $a_1,a_2,...,a_{i-1}$ va $b_1,b_2,...,b_j$
2) $a_1,a_2,...,a_i$ va $b_1,b_2,...,b_{j-1}$
3) $a_1,a_2,...,a_{i-1}$ va $b_1,b_2,...,b_{j-1}$

Ushbu uchta variantdan kattasini tanlaymiz, ya'ni

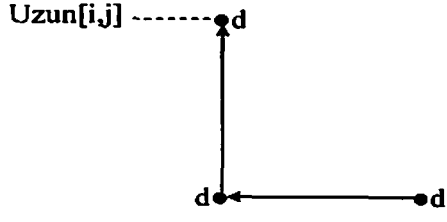
$$Uzun[i,j]=max\{ Uzun[i-1,j], Uzun[i,j-1], Uzun[i-1,j-1]\}$$

Lekin 3-variant uchun doimo $Uzun[i-1,j-1] \leq Uzun[i,j-1]$ bo 'ladi, shu bois

$$Uzun[i,j]=max\{ Uzun[i-1,j], Uzun[i,j-1] \}$$

Barcha $Uzun[i,j]$ lar hisoblanib bo'lingandan keyin, $Uzun[m,n]$ bevosita masala yechimi bo'ladi.

Ushbu $Uzun[i,j]$ massividan hosil qilinadigan qism ketma-ketlikni ham aniqlash mumkin bo'ladi. Faraz qilamiz $Uzun[m,n]=d$ deb, unda oxirgi nuqtadan chapga qarab d qiymatni uchratguncha o'tamiz, undan keyin yuqoriga qarab yuramiz va yana d ni qidiramiz. Sxematik ravishda quyidagicha bo'ladi



Demak, $a_i = b_j$ bo'ladi. Endi d-1 deb olamiz va unga mos *Uzun*[i-1,j-1] nuqtadan yuqorida keltirilgan algoritm bo'yicha ish tutamiz.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
aaaabbb daaabc	4 Baaa

Bu yerda *input.txt* faylida birinchi qatorda *A* va ikkinchi qatorda *B* satrlari kiritiladi va *output.txt* faylida eng uzun qism satr uzunligi va satrning o'zi teskari tartibda chop etilgan bo'ladi.

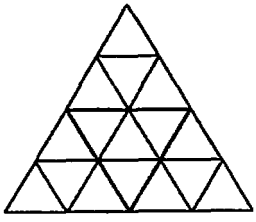
Dasturning umumiy ko'rinishi esa quyidagicha bo'ladi:

```
program TengQismQatorlar;
const MaxN=100;
var  i, j,n,m,d  : integer;
      A, B:string;
{bu erda  A, B berilgan satrrlar}
      Uzun : array[0..MaxN, 0..MaxN] of integer;
      fayl: text;
function Max(a,b : longint ):longint;
begin
      if a<b then Max:=b else Max:=a
end;
begin
{Berilgan ma'lumotlar}
      assign(fayl, 'input.txt');
      reset(fayl);
{A va B satrlarining uzunligi }
      readln(fayl, A); m:=length(A);
      readln(fayl, B); n:=length(b);
      close(fayl);
{Asosiy ishlarni boshlaymiz}
      for i:=0 to m do Uzun[i,0]:=0;
      for j:=0 to n do Uzun[0,j]:=0;
      for i:=1 to m do
      for j:=1 to n do
      if a[i]=b[j] then Uzun[i,j]:= Uzun[i-1,j-1]+1
      else Uzun[i,j]:=max(Uzun[i-1,j], Uzun[i,j-1]);
{chiqarish faylini  ochamiz }
```

```
      assign(fayl, 'output.txt');
      rewrite(fayl);
{Ketma-ketlik uzunligini chop qilamiz}
      writeln(fayl, Uzun[m,n]);
{Endi ushbu ketma-ketlikni ainqlaymiz}
      d:= Uzun[m,n]; i:=m; j:=n;
while (d<>0) do
begin
while Uzun[i,j-1]=d do j:=j-1;
while Uzun[i-1,j]=d do i:=i-1;
      write(fayl, a[i]); { Ketma-ketlikning d-hadi}
      i:=i-1; j:=j-1; d:=d-1;
      {bir qadam orqaga o'tib, ishni davom etamiz }
end;
      close(fayl);
end.
```

Topshiriqlar

1. Chipta $2n$ uzunlikdagi son bilan berilgan. Chipta baxtli hisoblanadi, agar birinchi n ta raqam yig'indisi, ikkinchi n ta raqam yig'indisiga teng bo'lsa. Barcha baxtli chiptalarni aniqlang.
2. Berilgan sonlar $a_1, a_2, \ldots, a_n$ ketma-ketligidan eng kam sonlarni o'chirish orqali, o'sib borayotgan $b_1, b_2, \ldots, b_m$ ketma-ketlik yarating, ya'ni unda $b_i<b_{i+1}$ bo'lsin.
3. Teng tomonli uchburchak quyidagi chizmada ko'rsatilganidek tomonlari K ga bo'laklanib birlashtirilgan.



Natijada bir necha uchburchaklar hosil qilinadi, qalin chiziq bilan belgilangan uchburchak ham inobatga olinadi. Berilgan K uchun barcha uchburchaklar sonini aniqlang.

4. Berilgan S satri quyidagi belgilardan, qavslardan tashkil topgan: { , }, [,] , (,). S satri to'g'ri qavsli ifoda hisoblanadi, agarda quyidagilardan biri bajarilsa:

1) S – bo'sh satr;

2) $S = S_1 + S_2 + ... + S_n$ ($n>1$) bo'lsa, bunda S_i bo'sh bo'lmagan to'g'ri qavsli ifoda;

3)Har qanday to'g'ri qavsli ifoda C uchun quyidagilardan biri $S =$ '(' + C + ')' yoki $S =$ '[' + C + ']' yoki $S =$ '{' + C + '}' bajarilsa.

Berilgan S uchun, uni to'g'ri qavsli ifodaga aylantirish uchun kami bilan nechta qavs kerak ekanligini aniqlang.

5. Avtobuslar faqatgina bir tomonga qatnaydi. Har bir i bekatdan j ($i<j$) bekatgacha bo'lgan yo'l narhi berilgan. Yo'lni boshidan to oxiricha shunday o'tish kerkki, yo'l harajati eng kam bo'lsin.

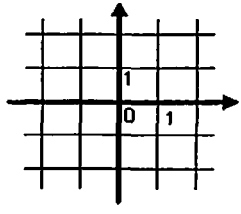
6. Berilgan musbat butun $x_1,x_2,...,x_n$ sonlar ichidan ba'zilarini tanlab, ular yig'indisi M bo'lishi mumkinligini aniqlang.

7. Ikkilik sonlardan $a=(a_1,a_2,...,a_m)$ va $b=(b_1,b_2,...,b_n)$ (a_i, $b_i = 0$ yoki 1) qatorlardan raqamlarni o'chirib, yangi bir-biriga teng bo'lgan $a' = b'$ sonlar tashkil bo'ladi. Ulardan qiymati maksimal bo'lgan sonni aniqlang.

8. Palindrom – bu ikki tomonlama bir xil o'qiladigan satr. Berilgan satr orasiga kami bilan nechta belgi joylashtirilsa, u palindromga aylanadi.

9. Berilgan $N \times N$ jadvalda kataklar raqamlar bilan to'ldirilgan. Bir katakdan ikkinchisiga faqatgina umumiy tomon orqali o'tish mumkin. Unda (1,1) katakdan (N,N) katakgacha eng minimal yo'lni va kataklardagi raqamlar yig'indisi maksimal bo'lgan variantni aniqlash talab etiladi.

10. Koordinatalar sistemasining boshida robot joylashtirilgan bo'lib, u to'rt tomonga yurish qilishi mumkin va har bir yurish bitta qadamga teng.

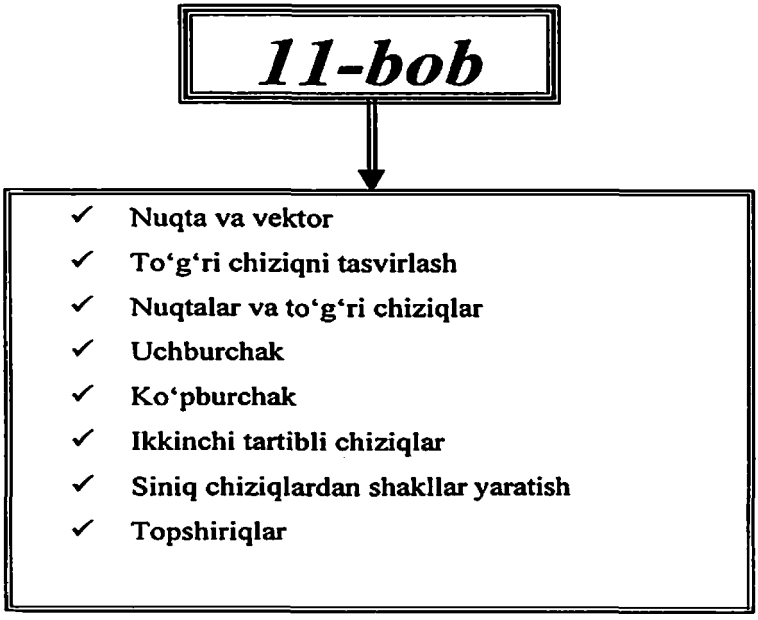


Robotni (X,Y) nuqtasiga K ta qadam bilan kelish yo'llar sonini aniqlang ($0 \le K \le 16$ va $|X|, |Y| \le 16$).

11-bob. DASTURLASHNING GEOMETRIYAGA OID MASALALARI

Ushbu bobda geometriyada mavjud tushunchalar va ulardan dasturlashda foydalanishga doir algoritmlar ko'rib chiqilgan. Bu yerda geometriyaga taaluqli tayanch tushunchalar berilgan bo'lib, ularni algoritmlarni yaratishda qo'llash bo'yicha misollar keltirilgan. Ushbu misollarni o'rganib chiqish orqali talabalarimiz geometriyaga taaluqli asosiy tushunchalarni qanday qilib dasturda tasvirlashni o'rganib olishadi.

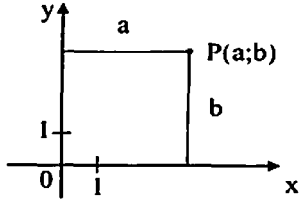
Mazkur bobda keltirilgan mavzular quyidagi tartibda berilgan:



11.1. Nuqta va vektor

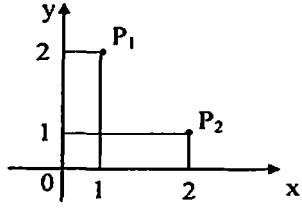
Tekislikda nuqtalar asosiy geometrik obyektlar hisoblanadi. Nuqtalar *A, B, C, ...* kabi bosh lotin harflari bilan belgilanadi.

Ma'lumki, o'zaro perpendikulyar bo'lgan gorizontal va vertikal sonlar o'qi Dekart to'g'ri burchakli koordinatalar sistemasini tashkil qiladi. Bu sistema orqali tekislikdagi nuqta bilan bir juft haqiqiy son o'rtasida bir qiymatli moslik o'rnatiladi. Tekislikda nuqta $P(a;b)$ bilan belgilansa, undagi *a va b* sonlarga *P* ning koordinatalari deyiladi.



"Nuqta berilgan" degan ibora uning koordinatalarining berilganligini, „Nuqtani toping" degan ibora esa, shu koordinatalarni topishni tushuniladi. Koordinatalar sistemasi orqali o'rnatilgan bunday moslikka koordinatalar usuli deyiladi.

Masalan, ikki $P_1(1;2)$ va $P_2(2;1)$ nuqtalari koordinatalar sistemasida quyidagicha joylashgan bo'ladi:

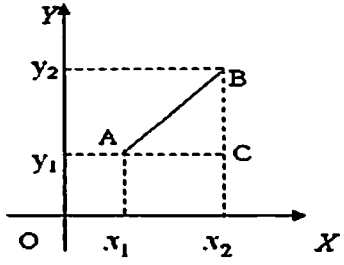


Bu usulni geometriyaning sodda masalalarini yechishga qo'llaymiz.

1) Tekislikda berilgan $A(x_1;y_1)$ va $B(x_2;y_2)$ nuqtalar orasidagi masofani topish talab etilsin. Ma'lumki, $AC=x_2-x_1$, $BC=y_2-y_1$. ABC to'g'ri burchakli uchburchakdan, $AB^2=(x_2-x_1)^2+(y_2-y_1)^2$, shunday qilib,

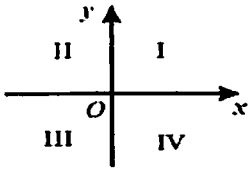
$$AB = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

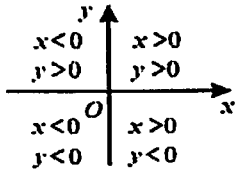
bo‘ladi. Bu formulaga tekislikda berilgan ikkita nuqta orasidagi masofani topish formulasi deyiladi.



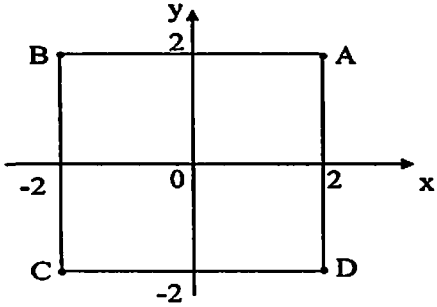
Bunda x koordinata V nuqtaning abssissasi, u koordinata esa ordinatasi deyiladi. Mos ravishda, Ox o‘q — abssissa o‘qi, Oy o‘q esa ordinata o‘qi deyiladi. Koordinata o‘qlarining kesishish nuqtasi O — yasalgan to‘g‘ri burchakli koordinatalar sistemasining boshi deyiladi.

Koordinatalar o‘qlari tekislikni choraklar (ba’zida kvadrantlar) deb ataladigan to‘rtta qismga bo‘ladi. Choraklar soat mili harakati yo‘nalishiga teskari tartibda raqamlanadi.





Quyidagi $A(2;2)$, $B(-2;2)$, $C(-2;-2)$ va $D(2;-2)$ nuqtalar o‘z choraklarida quyidagicha tasvirlanadi:



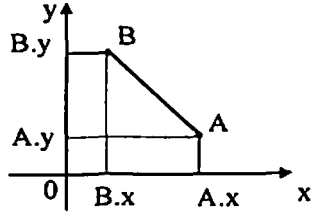
Ushbu bobda nuqtani dasturda tasvirlash uchun quyidagi ko‘rinishdagi qiymat kiritiladi

```
Type
TPoint=Record  x, u: Real;
End;
```

Agar koordinatalar butun sonlarda berilishi zarur bo‘lsa, unda qiymat quyidagicha kiritiladi

```
Type
TPoint=Record  x, u: Integer;
End;
```

Yuqorida aniqlangan ikki nuqta orasidagi masofani



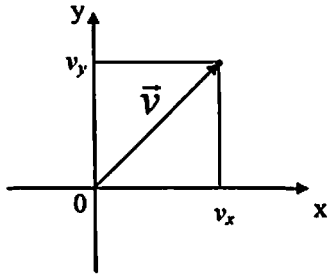
aniqlashda *Dist* qism dasturini tavsiya etish mumkin bo‘ladi, uni qo‘llash esa quyidagi dasturda keltirilgan:

```
program Masofa;
Type
      TPoint=Record  x, u: Real;
end;
var
      A, B :  TPoint;
      Result : real;
{qo‘shimcha funksiyalar}
Function   Dist (Const A, V:  TPoint):  Real;
      {Ikki nuqta orasidagi masofa}
begin
      Dist:=Sqrt (Sqr(A.x-B.x)+ Sqr (A.u-V.u));
end;
{dasturning asosiy qismi}
begin
      Writeln('A nuqtaning X va Y koordinatalari: ');
      Readln(A.x, A.y);
      Writeln('B nuqtaning X va Y koordinatalari: ');
      Readln(B.x, B.y);
      Result:= Dist ( A,V);
      Write('Masofa=',Result );
```

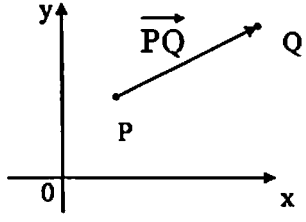
```
    Readln;
end.
```

Ushbu dasturni har xil ikki nuqtalar uchun tekshirib ko'rinishni tavsiya qilamiz.

Vektorlar ham nuqtalar kabi o'z koordinatalari bilan aniqlanadi, masalan $\vec{v} = (v_x, v_y)$. Uni tasvirlash esa quyidagicha bo'ladi



Istalgan ikki nuqta P va Q uchun $\overrightarrow{PQ}$ vektori quyidagicha bo'ladi

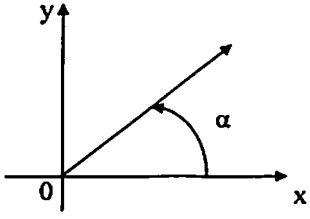


Ushbu vektorning koordinatalari $(Q_x-P_x,\ Q_y-P_y)$, shu bois har qanday vektorni (0;0) nuqtadan chiqadi deb qabul qilishimiz mumkin.

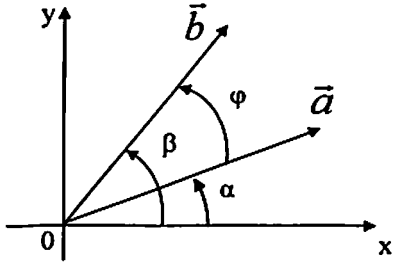
Har qanday 3 ta $\vec{a}, \vec{b}, \vec{c}$ vektorlar uchun quyidagilarni bilishimiz shart:

- yig'indi: $\vec{c} = \vec{a} + \vec{b}, c_x = a_x + b_x, c_y = a_y + b_y$
- ayirma:
- skalar kc $\vec{c} = \vec{a} - \vec{b}$ $c_x = a_x - b_x$ $c_y = a_y - b_y$

Vektorning og'ma t $\vec{a} \cdot \vec{b} = a_x b_x + a_y b_y$ dan vektorgacha bo'lgan burchakga aytiladi, ya'ni



Ikki $\vec{a}$ va $\vec{b}$ vektorlari orasidagi burchak $\vec{a}$ dan $\vec{b}$ ga qarab o'lchanadi, ya'ni



Agar ushbu harakat soat miliga qarshi bo'lsa, ushbu burchak musbat ishorali bo'ladi, soat mili bo'yicha bo'lsa, uning ishorasi manfiy bo'ladi. Yuqoridagi chizmada $\vec{a}$ va $\vec{b}$ vektorlar orasidagi burchakni o'lchashda quyidagi ayniyatdan foydalanamiz

$$cos(\beta-\alpha) = cos\alpha \cdot cos\beta + sin\alpha \cdot sin\beta$$

va uni $\vec{a}$ va $\vec{b}$ vektorlarining koordinatalari va uzunliklari orqali ifodalaymiz, ya'ni

$$cos(\beta - \alpha) = \frac{a_x b_x + a_y b_y}{l_a l_b}$$

Bu yerda keltirilgan $a_xb_x+a_yb_y$ skalar ko'paytma deyiladi va $\vec{a}\cdot\vec{b}$deb yoziladi. Bu yerdan, agar vektorlar perpendikulyar bo'lsa $a_xb_x+a_yb_y=0$ ga teng bo'ladi.

Vektorlar orasidagi burchak 0 yoki π bo'lsa, ular kollinear deyiladi.

Quyidagi ayniyatdan

$$sin(\beta-\alpha) = cos\alpha \cdot sin\beta - sin\alpha \cdot cos\beta$$

esa

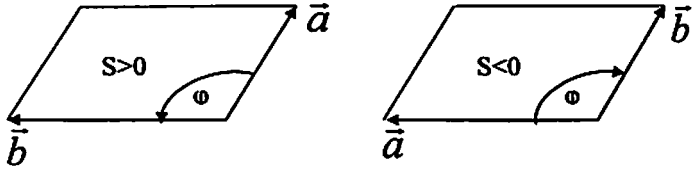
$$sin(\beta - \alpha) = \frac{a_x b_y - a_y b_x}{l_a l_b}$$

kelib chiqadi.

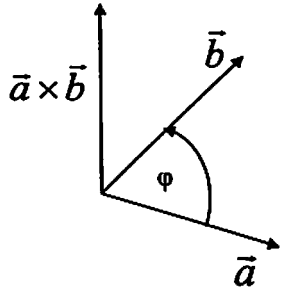
Ushbu formulani boshqacha yozsak, ya'ni

$$a_x b_y - a_y b_x = l_a l_b \sin(\beta - \alpha)$$

undan qiziqarli xossaga duch kelamiz, ya'ni $\vec{a}$ va $\vec{b}$ vektorlardan tashkil topgan parallelogrammning S "yuzasi" kelib chiqadi. Faqatgina u manfiy yoki musbat bo'lishi mumkin:



Uch o'lchamli fazoli Dekart sistemasida

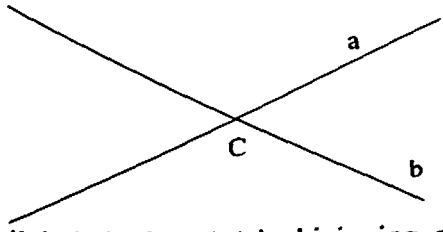


$\vec{a}$ va $\vec{b}$ vektorlarining vektorli ko'paytmasini kiritish mumkin va $\vec{a} = (a_x, a_y, 0)$ va $\vec{b} = (b_x, b_y, 0)$ lar uchun $\vec{c} = \vec{a} \times \vec{b} = (0,0, a_x b_y - a_y b_x)$ ga teng bo'ladi.

Bu yerdan quyidagi xulosani qilsak bo'ladi, ya'ni $\vec{c}$ vektori $\vec{a}$ va $\vec{b}$ vektorlari joylashgan tekislikga perpendikulyar bo'ladi.

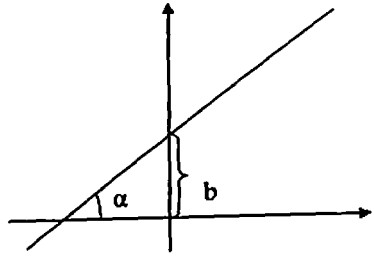
11.2. To'g'ri chiziqni tasvirlash

Tekislikda to'g'ri chiziqlar asosiy geometrik obyektlardan biri hisoblanadi. To'g'ri chiziqlar *a, b, c,* ... kabi kichik lotin harflari bilan belgilanadi. S nuqta ham a to'g'ri chiziqqa, ham b to'g'ri chiziqqa tegishlidir. Bu holda a va b to'g'ri chiziqlar C nuqtada *kesishadi* deyiladi, S nuqta esa a va b to'g'ri chiziqlarning kesishish nuqtasi bo'ladi.



Quyida har xil holatlarda to'g'ri chiziqning analitik ifodalarini, ya'ni tenglamalarini keltirib chiqaramiz va ular yordamida to'g'ri chiziqning tekislikdagi vaziyatlarini o'rganamiz.

To'g'ri chiziqning Ox o'qi musbat yo'nalishi bilan hosil qilgan burchagi α va to'g'ri chiziqning ordinatlar o'qidan ajratgan kesmasining kattaligi b berilganda, uning tekislikdagi holati aniq bo'ladi.



Keltirilgan qiymatlar bo'yicha to'g'ri chiziqning tenglamasi quyidagicha bo'ladi:

$$y = kx + b$$

bu yerda $k = tg\,\alpha$ to'g'ri chiziqning burchak koeffitsiyenti deyiladi va bunday to'g'ri chiziqni burchak koeffitsiyentli tenglamasi deyiladi.

Agar $A(x_1;y_1)$ nuqtasi berilgan bo'lsa to'g'ri chiziq A nuqtadan o'tishi uchun A nuqtaning koordinatlari to'g'ri chiziq tenglamasini qanoatlantirishi kerak, ya'ni $y_1 = kx_1 + b$. Demak burchak koeffitsiyentli tenglamasidan ushbu tenglikni ayirsak:

$$y - y_1 = k(x - x_1)$$

hosil bo'ladi. Ushbu tenglamaga berilgan bitta nuqtadan o'tuvchi to'g'ri chiziqlar dastasining tenglamasi deyiladi.

Agar to'g'ri chiziq $B(x_2;y_2)$ ikkinchi nuqtadan ham o'tsa, u holda

$$y_2 - y_1 = k(x_2 - x_1)$$

bo'lib,

$$k = \frac{y_2 - y_1}{x_2 - x_1}$$

bo'ladi. k ning qiymatini yuqoridagi tenglamaga qo'yib,

$$\frac{y-y_1}{y_2-y_1}=\frac{x-x_1}{x_2-x_1}$$

tenglamani hosil qilamiz. Bu tenglama berilgan ikki $A(x_1;y_1)$ va $B(x_2;y_2)$ nuqtalardan o'tuvchi to'g'ri chiziq tenglamasi deyiladi. Ushbu tenglamani umumiy holda quyidagi ko'rinishda tasvirlash maqsadga muvofiq:

$$x\cdot(y_1-y_2)+y\cdot(x_2-x_1)+(x_1y_2-x_2y_1)=0$$

Ikki noma'lumli

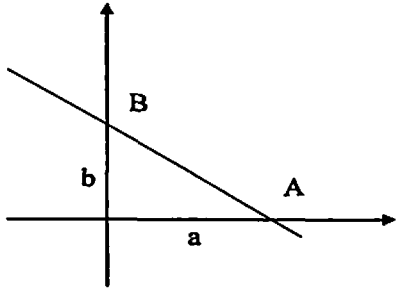
$$Ax+By+C=0$$

tenglamani qaraymiz.

Bundan, $By=-Ax-C$, $y=-\frac{A}{B}x-\frac{C}{B}$ bo'lib, $k=-\frac{A}{B}$, $b=-\frac{C}{B}$ bilan belgilasak, $y=kx+b$ tenglama hosil bo'ladi. Shunday qilib, $Ax+By+C=0$ tenglama ham to'g'ri chiziq tenglamasi ekanligi kelib chiqadi. Bundan esa quyidagi xulosani keltirish mumkin:

Har qanday ikki o'zgaruvchili $Ax+By+S=0$ chiziqli tenglama tekislikda to'g'ri chiziqni ifodalaydi va u to'g'ri chiziqning umumiy tenglamasi deyiladi. Agar $\sqrt{A^2+B^2}=1$ va $A\geq 0$ bo'lsa, ushbu tenglama normal deyiladi.

To'g'ri chiziq koordinata o'qlaridan mos ravishda a va b kesmalar ajratib o'tsin.



Ya'ni to'g'ri chiziq $A(a;0)$ va $B(0;b)$ nuqtalardan o'tadi. Berilgan ikki nuqtadan o'tuvchi to'g'ri chiziq tenglamasiga asosan quyidagi

$$\frac{x}{a}+\frac{y}{b}=1$$

tenglama hosil bo'ladi. Bu tenglamaga to'g'ri chiziqning kesmalarga nisbatan tenglamasi deyiladi.

Berilgan $(x_0;y_0)$ nuqtadan va (a_x,a_y) vektoriga parallel to'g'ri chiziqning tenglamasi quyidagicha bo'ladi

$$x=x_0+a_x\cdot t,\quad y=y_0+a_y\cdot t$$

bu yerda t ning qiymati $-\infty$ dan $+\infty$ gacha o'zgaradi. Ushbu tenglamalar to'g'ri chiziqning yo'naltiruvchi vektorli tenglamasi deyiladi.

Agar to'g'ri chiziq $(x_0;y_0)$ va $(x_1;y_1)$ nuqtalar bilan berilgan bo'lsa, unda $(x_1-x_0;y_1-y_0)$ ni yo'naltiruvchi vektor deb qabul qilamiz va natijada quyidagini hosil qilamiz

$$x=x_0+(x_1-x_0)\cdot t,\quad y=y_0+(y_1-y_0)\cdot t$$

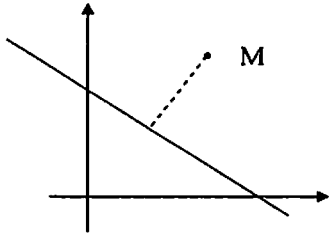
Agar bu yerda t ning qiymati -1 dan +1 gacha o'zgarsa, u holda ikki nuqta orasidagi kesma nuqtalarining tenglamasi hosil bo'ladi.

11.3. Nuqtalar va to'g'ri chiziqlar

$M(x_0;y_0)$ nuqta va $x\cos\alpha+y\sin\alpha-p=0$ tenglamasi bilan to'g'ri chiziq berilgan bo'lsin. Berilgan nuqtadan, berilgan to'g'ri chiziqqacha bo'lgan masofa

$$d=|x_0\cos\alpha+y_0\sin\alpha-p|$$

formula yordamida aniqlanadi.



To'g'ri chiziq tenglamasi umumiy

$$Ax+By+C=0$$

ko'rinishda berilgan bo'lsa, nuqtadan to'g'ri chiziqqacha bo'lgan masofa,

$$d=\frac{|Ax_0+By_0+C|}{\sqrt{A^2+B^2}}$$

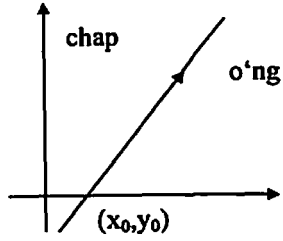
formula bilan aniqlanadi.

Ko'p hollarda nuqtaning to'g'ri chiziqga nisbatan joylashuvi bizni qiziqtirishi mumkin. Agar to'g'ri chiziq $(x_0;y_0)$ nuqtasi va yo'naltiruvchi (b_x,b_y) vektori bilan berilgan bo'lsa, u holda $(A_x;A_y)$ nuqtasi ushbu to'g'ri chiziqning qaysi tomonidan joylashganligini aniqlash uchun quyidagini hisoblaymiz:

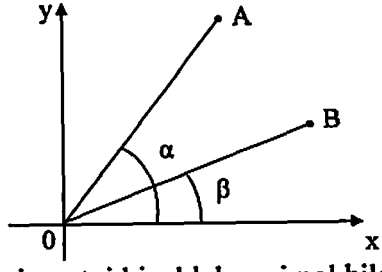
$$S=b_x\cdot(A_y-y_0)-b_y\cdot(A_x-x_0)$$

Ushbu ifoda nolga teng bo'lsa, nuqta to'g'ri chiziqda joylashgan bo'ladi, agar noldan katta bo'lsa, u holda nuqta to'g'ri chiziqning

chap tomonida joylash-gan bo'ladi va aks holda chiziqning o'ng tomonida joylashgan bo'ladi. "Chap" va "o'ng" tomonlar quyidagi chizmada ko'rsatilgan, unda chiziqda turib vektor yo'nalishi tomon qarash kerak bo'ladi.



Ikki nuqta $A(x_1; y_1)$ va $B(x_2; y_2)$ berilgan bo'lsa, OA va OB chiziqlarining qaysi biri Ox o'qi bilan katta burchak tashkil qilishini aniqlaymiz.



Bu yerda quyidagi qiymatni hisoblab, uni nol bilan taqqoslaymiz:

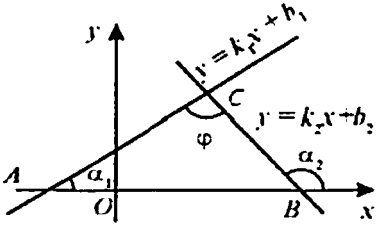
$$D=x_1 \cdot y_2 - x_2 \cdot y_1$$

Agar $D=0$ bo'lsa, u holda burchaklar teng, agar $D<0$ bo'lsa, u holda $\alpha>\beta$ bo'ladi, agar $D>0$ bo'lsa, u holda $\alpha<\beta$ bo'ladi.

To'g'ri chiziqlarning o'zaro joylashishinini aniqlashda tekislikda

$$y = k_1 x + b_1 \ , y = k_2 x + b_2$$

ko'rinishda bo'lgan ikkita to'g'ri chiziqlarni ko'rib chiqamiz. Bu tenglamalarni sistema sifatida qarab, berilgan to'g'ri chiziqlarning umumiy nuqtasini topish mumkin.



Ular orasidagi burchak esa quyidagicha aniqlanadi:

$$tg\phi = tg(\alpha_2 - \alpha_1)$$

Burchak koeffitsiyentlari orqali esa

$$tg\phi = \frac{k_2 - k_1}{1 + k_1 k_2}$$

bo'lishi kelib chiqadi. Bu ikki to'g'ri chiziq orasidagi burchak formulasi deyiladi. Agar to'g'ri chiziqlar o'zaro parallel bo'lsa, $\alpha_1=\alpha_2$ va u vaqtda $k_1=k_2$ bo'ladi. Bu tenglik ikki to'g'ri chiziqning o'zaro parallellik sharti deyiladi.

Agar to'g'ri chiziqlar bir-biriga perpendikular bo'lsa, tgφ aniqmas bo'ladi. Bu shart $k_1 \cdot k_2 = -1$ bo'lganda bajariladi. Bu ikki to'g'ri chiziqning perpendikulyarlik sharti deyiladi.

Agar 2 ta to'g'ri chiziqlar umumiy tenglamalar orqali berilgan bo'lsa, ya'ni

$A_1x + B_1y + C_1 = 0$ va $A_2x + B_2y + C_2 = 0$, u holda ularning kesishgan nuqtasi bo'lishi uchun $A_1 \cdot B_2 - A_2 \cdot B_1 \neq 0$ bo'lishi kerak. Shunda kesishgan nuqta koordinatalari quyidagicha aniqlanadi

$x=(-C_1 \cdot B_2 + C_2 \cdot B_1) / (A_1 \cdot B_2 - A_2 \cdot B_1)$, $y=(-A_1 \cdot C_2 + A_2 \cdot C_1) / (A_1 \cdot B_2 - A_2 \cdot B_1)$.

Ushbu nuqtani aniqlovchi qism dasturni va undan foydalanishni quyidagi dasturda keltiramiz:

```
program Kesishgan_Nuqta;
Type
    TPoint=Record  x, u: Real;
End;
Type
    Tline = Record A, B, C : Real;
End;
var
```

```
    A,B,P: TPoint;
    L1,L2:  TLine;
{qo'shimcha funksiyalar}
{Kesishgan nuqta koordinatalarini aniqlash q/dasturi}
Procedure  Line2ToPoint (Const L1,L2:  TLine);
var
    st:  Real;
begin
    st:=L1.A*L2.B-L2.A*L1.V;
    P.X:=(-L1.C*L2.B+L2.C*L1.B)/st;
    P.Y:=(-L1.A*L2.C+L2.A*L1.C)/st;
end;
    {dasturning asosiy qismi}
begin
Writeln('Birinchi chiziq:A,B va C koeffitsiyentlari:');
    Readln(L1.A, L1.B, L1.C);
Writeln('Ikkinchi chiziq:A,B va C koeffitsiyentlari:');
    Readln(L2.A, L2.B, L2.C);
    Line2ToPoint ( L1,  L2);
    Write(P.X, P.Y :8:2);
    Readln;
end.
```

Ushbu dasturni har xil qiymatlar bilan tekshirib ko'ring.

Agar $A_1 \cdot B_2 - A_2 \cdot B_1 = 0$ bo'lsa, to'g'ri chiziqlar parallel bo'ladi.

Agar to'g'ri chiziq umumiy tenglama bilan $Ax+By+C=0$ berilgan bo'lsa va (x_0,y_0) nuqtasi to'g'ri chiziqqa tegishli bo'lsa, quyidagi tenglama

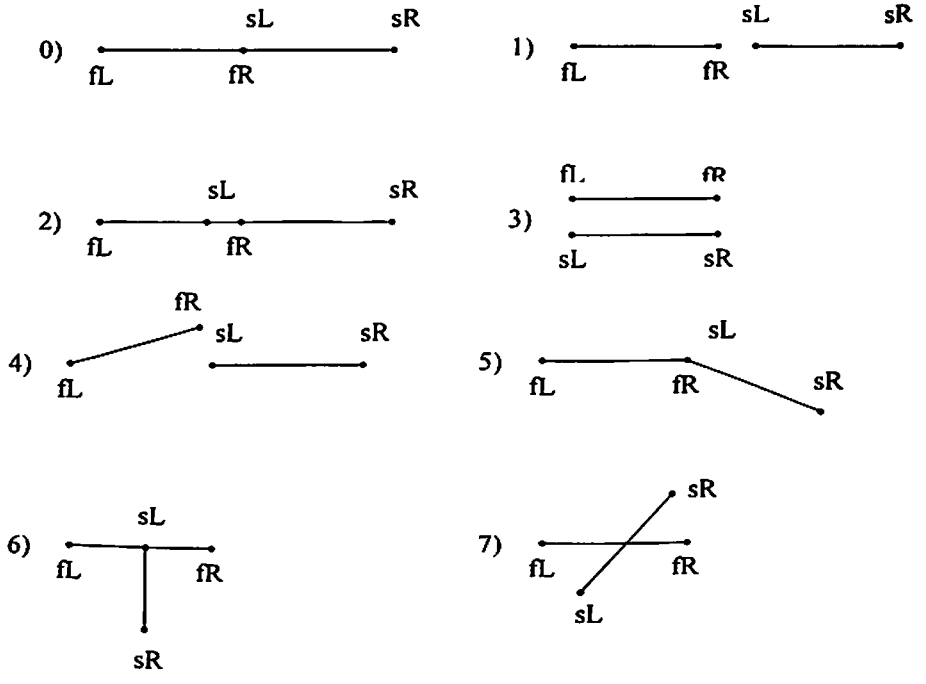
$$A(x-x_0) + B(y-y_0) = O$$

o'rinli bo'ladi. Bundan quyidagi xulosalarni chiqarish mumkin: - (A,B) vektor $Ax + By + C = 0$ to'g'ri chiziqga perpendikular bo'ladi; - $Ax + By + C = 0$ tenglamasi bilan berilgan to'g'ri chiziq $(-B,A)$ vektoriga parallel bo'ladi.

Ikki to'g'ri chiziqlar tenglamalaridagi koeffitsiyentlar bo'yicha ular haqidagi xulosalar quyidagi jadvalda keltirilgan:

Ikki chiziq munosabati	Tenglama		
	Burchak koeffitsiyentli	Umumiy	yo'naltiruvchi vektorli
Parallel yoki teng	$K_1 = k_2$	$A_1 \cdot B_2 - A_2 \cdot B_1 = 0$	$a_x \cdot b_y - a_y \cdot b_x = 0$
Oldingi shart bajarilganda teng	$D_1 = d_2$	$A_1 \cdot C_2 - A_2 \cdot C_1 = 0$	$a_x \cdot (y_2 - y_1) - a_y \cdot (x_2 - x_1) = 0$
Perpendikulyar	$K_1 \cdot k_2 = -1$	$A_1 \cdot A_2 + B_1 \cdot B_2 = 0$	$a_x \cdot b_x + a_y \cdot b_y = 0$

Tekislikda ikkita kesma berilgan bo'lsa, ularning o'zaro joylashuvlari quyidagicha bo'lishi mumkin:



Ushbu joylashuvlarni aniqlab beruvchi *SegmCross* funksiya qism dasturini keltiramiz. Unda funksiya natijalari quyidagicha bo'ladi:
0 – kesmalar bitta to'g'ri chiziqda joylashgan va bitta nuqtada kesishadi;
1 – kesmalar bitta to'g'ri chiziqda joylashgan va kesishmaydi;

2 – kesmalar bitta to‘g‘ri chiziqda joylashgan va kesishgan nuqtalari ko‘p;
3 – kesmalar parallel chiziqlarda joylashgan;
4 – kesmalar bitta to‘g‘ri chiziqda emas va umumiy nuqtaga ega emas;
5 – kesmalar bitta to‘g‘ri chiziqda emas va chetki nuqtalarining birida kesishadi;
6 – kesmalar bitta to‘g‘ri chiziqda emas va kesishgan nuqta bittasining chetki nuqtasi;
7 – kesmalar bitta to‘g‘ri chiziqda emas va kesishgan nuqta chetki nuqta emas.

Ushbu kesmalar joylashuvlarini aniqlovchi qism dasturni va unga bog‘liq qolgan funksiyalarni ham keltirgan holda, ulardan foydalanishni quyidagi dasturda keltiramiz:

```
program Kesmalar_joylashuvi;
Type
      TPoint=Record  x, u: Real;
End;
Type
      Tline = Record A, B, C : Real;
End;
const eps=0.0000000001;
{-haqiqiy sonlarni taqqoslashda ishlatiladi}
var
      fL, fR, sL, sR: TPoint;
      Result : Byte;
      fayl : text;
   {qo‘shimcha funksiyalar}
{ Chiziqlarning kesishgan nuqtasi koordinatalarini } {aniqlash qism
dasturi}
Procedure  Line2ToPoint (Const L1,   L2:   TLine; var  P: Tpoint);
var
      st:   real;
begin
      st:=L1.A*L2.B-L2.A*L1.V;
      P.X:=(-L1.C*L2.B+L2.C*L1.B)/st;
      P.Y:=(-L1.A*L2.C+L2.A*L1.C)/st;
end;
Procedure Point2ToLine(Const v,w:TPoint; var L:TLine);
```

```
{Ikki nuqta bo'yicha to'g'ri chiziq tenglamasidagi A,B,C} {larni
aniqlash}
begin
      L.A:=v.y-w.y;
      L.B:=w.x-v.x;
      L.C:= v.x* w.y - v.y* w.x;
end;
Function  CrossLine(Const L1, L2 : TLine): Boolean;
      {Ikki chiziq kesishganligini aniqlash}
var
     st: real;
begin
      st:= L1.A * L2.B – L2.A * L1.B;
      CrossLine:=not (abs(st) < eps);
end;
Function EqNumber(Const a, b :real): Boolean;
      {Ikki sonni tengligini aniqlash}
begin
      EqNumber:=(abs(a- b)<eps) ;
end;
Function EqPoint(Const A, B : TPoint): Boolean;
      {Ikki nuqtani tengligini aniqlash}
begin
EqPoint:=(abs(A.x- B.x)<eps) And (abs(A.y-B.y)<eps);
end;
Function  AtSegm(Const A, B, P : TPoint): Boolean;
      {P nuqtasi kesmada ekanligini aniqlash}
begin
      If EqPoint(A,B) then AtSegm:= EqPoint (A,P)
      {- A va B nuqtalari teng}
      Else
{AP va PB vektorlar ko'paytmasidan }
AtSegm:= EqNumber ( ((B.x-A.x)*(P.y-A.y)), ((B.y-A.y)*(P.x-A.x)) )
And (( (P.x>=A.x) And (B.x>=P.x)) or ( (P.x>=B.x) And
(A.x>=P.x)));
{P nuqtasi o'rtada bo'lishi kerak}
end;
Function SegmLineCross( const fL, fR, sL, sR : Tpoint) : Byte;
{Ikki kesma bir chiziqda}
```

```
Var Minf, Maxf, Mins, Maxs: real;
Begin
if fl.x*fl.x+fl.y*fl.y < fr.x*fr.x+fr.y*fr.y then minf:=fl.x*fl.x+fl.y*fl.y
else minf:=fr.x*fr.x+fr.y*fr.y;
{fL yoki fR nuqtalardan (0;0) ga yaqinini aniqladik}
if fl.x*fl.x+fl.y*fl.y > fr.x*fr.x+fr.y*fr.y then
maxf:=fl.x*fl.x+fl.y*fl.y else maxf:=fr.x*fr.x+fr.y*fr.y;
{fL yoki fR nuqtalardan (0;0) dan  uzog'ini  aniqladik}
if sl.x*sl.x+sl.y*sl.y < sr.x*sr.x+sr.y*sr.y then
mins:=sl.x*sl.x+sl.y*sl.y else mins:=sr.x*sr.x+sr.y*sr.y;
{sL yoki sR nuqtalardan (0;0) ga yaqinini aniqladik}
if sl.x*sl.x+sl.y*sl.y > sr.x*sr.x+sr.y*sr.y then
maxs:=sl.x*sl.x+sl.y*sl.y else maxs:=sr.x*sr.x+sr.y*sr.y;
{sL yoki sR nuqtalardan (0;0) dan  uzog'ini  aniqladik}
If Eqnumber(Minf, Maxs) or  Eqnumber(Maxf, Mins) then
SegmLineCross :=0
{kesmaning bitta uchi ikkinchi kesmaning uchiga teng}
Else
If  (Mins>= Maxf) or  (Minf>= Maxs) then SegmLineCross :=1
{ bitta nuqta ikkinchi kesmaning ichida bo'lmaydi }
Else SegmLineCross :=2;
{-bitta nuqta ikkinchi kesmaning ichida bo'ladi}
End;
Function SegmCross (Const fL, fR, sL, sR: Tpoint): Byte;
var
      rs:TPoint;
      L1,L2:Tline;
      b, bRS: Boolean;
begin
      Point2ToLine (fL,  fR, L1) ;
      Point2ToLine (sL,  sR,  L2);
      If CrossLine (L1,L2)   Then
      {-ya'ni ikki chiziq kesishsa, u holda}
begin
Line2ToPoint(L1,L2,rs);
{-ikki chiziq,kesma emas,kesishgan nuqtasini rs } {aniqladik }
bRS:=EqPoint (rs,fL)  Or EqPoint (rs, fR)  Or EqPoint (rs,sL)  Or
EqPoint (rs,sR);
```

```
{ya'ni kesishgan nuqta kesmalar uchlarining biriga } {tengligini
tekshiramiz}
b:=EqPoint(sL,fL) Or EqPoint(sR, fR) Or EqPoint(sR,fL) Or
EqPoint(sL,fR);
{ya'ni ikki kesma chetki nuqtalarini tengligini } {tekshiramiz}
      If b Then SegmCross:=5
      Else
If AtSegm(fL,fR,rs) And AtSegm(sL,sR,rs) and ( not bRS)
      Then SegmCross:= 7
      Else
      If AtSegm(fL,fR,rs) and AtSegm (sL,sR,rs) and bRS  Then
SegmCross:=6
      Else SegmCross:=4;
end
      Else
      If EqNumber (L1.A*L2.B, L2.A*L1.B) And Not (EqNumber
(L1.C,L2.C))
      Then SegmCross:=3
      Else
      SegmCross := SegmLineCross ( fL,  fR, sL,  sR);
end;
      {dasturning asosiy qismi}
begin
   {massiv o'lchamini va qiymatlarni kiriting}
      assign(fayl, 'input.txt');
      reset(fayl);
      {Birinchi kesmaning fL, fR nuqtalari koordinatalari}
      Readln(fayl, fL.x, fL.y, fR.x, fR.y);
{Ikkinchi kesmaning sL, sR nuqtalari koordinatalari}
      Readln(fayl, sL.x, sL.y, sR.x, sR.y);
      close(fayl);
      Result := SegmCross (fL, fR, sL, sR);
{chiqarish fayliga natijani chop etamiz}
      assign(fayl, 'output.txt');
      rewrite(fayl);
      Write(fayl, Result);
      close(fayl);
end.
```

Ushbu dasturda ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
0 0 0 2 0 0 2 0	5

Bu yerda *input.txt* faylida birinchi va ikkinchi qatorlarda mos ravishda kesmalar nuqtalarining koordinatalari bo'sh katak orqali ajratilib kiritiladi va *output.txt* faylida ularning o'zaro joylashuvlarini belgilovchi raqam chiqarilgan bo'ladi.

11.4. Uchburchak

Shunday qilib, biz tekislikda nuqta va to'g'ri chiziqlar, asosiy geometrik obyektlar bilan tanishib chiqdik. Endi geometrik shakllardan biri bo'lmish uchburchakka doir misollarni ko'rib chiqamiz.

Keng tarqalgan misollardan biri bu quyidagi misol.

Uchta kesma uzunliklari *a*, *b*, *c* bilan berilgan bo'lsa, ulardan uchburchak tuzish mumkinligini aniqlash talab etiladi.

Har qanday uchburchak tomonlari uchun quyidagi tengsizliklar bajarilishi shart:

$$a+b>c \text{ va } a+c>b \text{ va } b+c>a$$

Shundan kelib chiqqan holda, quyidagi *IsTrian* funksiya qism dasturini va undan foydalanish dasturini taklif etamiz, uning natijasi *true* yoki *false* bo'ladi.

```
program Uchburchak;
var
      a, b, c : Real;
      Result : Boolean;
      {qo'shimcha funksiyalar}
Function    IsTrian(Const a,b,c: Real): Boolean;
begin
      IsTrian:=(a+b> c) And   (a+c> b) And (b+c>  a);
end;
      {dasturning asosiy qismi}
begin
      Writeln('Uchburchakning A , B va C tomonlari: ');
      Readln(A,B,C);
      Result := IsTrian( a, b, c);
      If Result then  Write('Uchburchak tuzish mumkin')
```

```
    else Write('Uchburchak tuzish mumkin emas');
    Readln;
end.
```

Ushbu dasturni quyidagi qiymatlar uchun tekshirib ko'rishni tavsiya qilamiz:

A	*B*	*C*	Natija
10	3	7	*Uchburchak tuzish mumkin emas*
3	4	5	*Uchburchak tuzish mumkin*
5	7	13	*Uchburchak tuzish mumkin emas*
5	5	5	*Uchburchak tuzish mumkin*

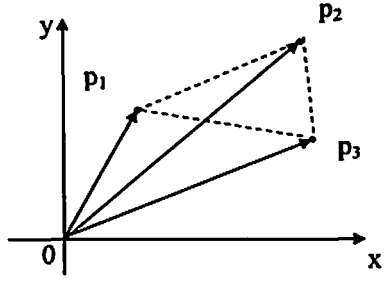
Agar uchburchakning tomonlari berilgan bo'lsa, u holda ko'p hollarda uning yuzasini ham aniqlash talab etiladi. Bunda biz Geron formulasidan foydalanib, quyidagi *Sq_Geron* funksiya qism dasturini va undan foydalanishni taklif etamiz:

```
program Uchburchak_yuzasi_Geron;
var a, b, c : Real;
    Result : Real;
    {qo'shimcha funksiyalar}
Function Sq_Geron(a,b,c: Real) : Real;
var
    p: Real;
begin
    p:= (a+b+c)/2;
    Sq_Geron:=Sqrt(p* (p-a) * (p-b) * (p-c) ) ;
end;
    {dasturning asosiy qismi}
begin
    Writeln('Uchburchakning A , B va C tomonlari: ');
    Readln(A,B,C);
    Result := Sq_Geron ( a, b, c);
    Write('Uchburchak yuzasi:', Result);
    Readln;
end.
```

Ushbu dasturni quyidagi qiymatlar uchun tekshirib ko'rishni tavsiya qilamiz:

A	*B*	*C*	Natija
5	5	6	12
3	4	5	6
12	10	10	48
10	6	8	24

Agar uchburchak uchlarining koordinatalari berilgan bo'lsa, u holda uning yuzasini vektorli ko'paytma orqali ham aniqlash mumkin.



Bu yerdan ikkita vektor tashkil qilamiz: p_1p_2 va p_1p_3 . Endi ularni vektorli ko'paytiramiz va faqatgina ishorasini musbat qilib olish esdan chiqmasin. Shundan kelib chiqqan holda, quyida ushbu g'oyaga asoslangan funksiya qism dasturini va undan foydalanish dasturini taklif etamiz:

```
program Uchburchak_yuzasi_Vector;
Type
      TPoint=Record x,   u:  Real;
end;
var
      p1,p2,p3 : TPoint;
      Result : Real;
      {qo'shimcha funksiyalar}
Function  Sq_Vector(p1,p2,p3: TPoint) : Real;
begin
Sq_Vector:=( (p2.x-p1.x) * (p3.y-p1.y) -(p2.y-p1.y) * (p3.x-p1.x)  )/2 ;
end;
      {dasturning asosiy qismi}
```

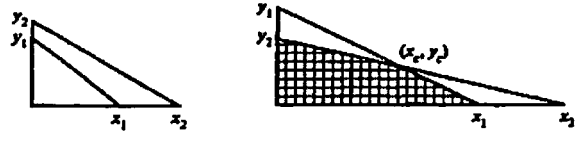
```
begin
    Writeln('Birinchi p1 uchining koordinatalari: ');
    Readln(p1.x,p1.y);
    Writeln('Ikkinchi p2 uchining koordinatalari: ');
    Readln(p2.x,p2.y);
    Writeln('Uchinchi p3 uchining koordinatalari: ');
    Readln(p3.x,p3.y);
    Result := Sq_Vector (p1,p2,p3);
    Write('Uchburchak yuzasi:', Abs( Result));
    Readln;
end.
```

Ushbu dasturni quyidagi qiymatlar uchun tekshirib ko'rishni tavsiya qilamiz:

1-nuqta		*2-nuqta*		*3-nuqta*		Natija
x	*y*	*x*	*y*	*x*	*Y*	
0	0	4	0	0	3	6
0	0	0	-8	-6	0	24

Keyingi misolda ikkita to'g'ri burchakli uchburchaklar berilgan bo'lib, ularning katetlari koordinata o'qlarida joylashgan. Uning to'g'ri burchakli uchi koordinatalar sistemasining boshida bo'lsa, ularning umumiy kesishgan qismining maydon yuzasini aniqlash talab etiladi. Bu yerda uchburchaklarning qolgan uchlari koordinata o'qlarida bo'lganligi sababli quyidagicha beriladi: x_1 va y_1 birinchi uchburchak uchun va x_2 va y_2 ikkinchi uchburchak uchun.

Uchburchaklarning o'zaro joylashuvi quyidagi chizmada keltirilgan



a) б)

Birinchi a) variantda natija $0{,}5 \cdot x_1 \cdot y_1$ bo'ladi, ikkinchi variant esa murakkabroq bo'ladi. Bu yerda gipotenuzalarni burchak koeffiyentli to'g'ri chiziq tenglamasi orqali ifodalab, ularning kesishgan nuqtasini aniqlaymiz:

Chizmadan $k_1=y_1/x_1$ va $k_2=y_2/x_2$ deb qabul qilamiz va bundan $x_c=(y_1-y_2)/(k_1-k_2)$ va $y_c=y_2-k_2\cdot(y_1-y_2)/(k_1-k_2)$ bo'ladi.

Chizmadan ko'rinib turibdiki, ushbu maydonni ikki qismga, ya'ni trapesiya va to'g'ri burchakli uchburchakga ajratib, ularning maydonlari alohida hisoblanadi $0,5\cdot(y_2+y_c)x_c$ va $0,5\cdot(x_1-x_c)y_c$.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
6 6 12 4	15

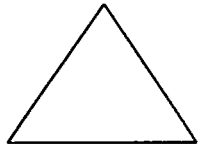
Bu yerda *input.txt* faylida *x1, y1, x2, y2* qiymatlari kiritiladi va *output.txt* faylida faqatgina uchburchak yuzasi chiqarilgan bo'ladi.

Yuqoridagi g'oya asosida tuzilgan quyidagi dasturni taklif etamiz:

```
program TrianglesIntersection;
var
      fayl : text;
      x1, u1, x2, u2,t, x_cross, y_cross,k1, k2, S : real;
begin
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, x1,  y1,  x2,  y2);
      close(fayl);
      if x1 > x2 then
begin
      t:=x1; x1:=x2; x2:=t;
      t:=y1; y1:=y2; y2:=t;
end;
      if y1 <= y2 then S:=0.5* x1 * y1
      else
begin
      k1 := y1 / x1;
      k2 := y2 / x2;
x_cross:=(y1-y2)/(k1-k2); y_cross := y1 -  k1* x_cross;
S:=0.5*x_cross*(y2+y_cross)+0.5*(x1-x_cross)*y_cross
end;
      assign(fayl, 'output.txt');
      rewrite(fayl);
      writeln(fayl, S:0:0);
      close(fayl)
```

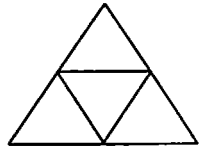
end.

Serpinskiy uchburchaklari. Har qanday teng tomonli uchburchakning 0-tartibli deb qabul qilsak



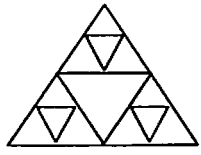
0-tartibli uchburchak

va uning o'rta chiziqlari o'tkazilsa, unda quyidagi 1-tartibli uchburchaklar paydo bo'ladi:



1-tartibli uchburchak

Shu tartibda paydo bo'lgan uchburchaklarni, faqatgina markazda joylashgan uchburchakdan tashqari, bo'laklash orqali n-tartibli uchburchaklar sodir qilinadi:



2-tartibli uchburchak

Ushbu algorimni dasturini yaratishda o'z-o'ziga o'xshash geometrik shakllarni $(n-1)$-tartibidan n–tartibga o'tishni amalga oshirsak kifoya bo'ladi. Bu yerda, n-tartibli $(n\geq 1)$ uchburchakni (x,y) nuqtasidan boshlab chizish uchun $(n-1)$-tartibli uchburchakni uch marotaba chaqirish kerak bo'ladi va unda uchburchak tomoni $d/2$ bo'ladi. Ushbu uchburchaklarning chap pastki nuqtasi quyidagi koordinatalarda joylashgan bo'ladi:

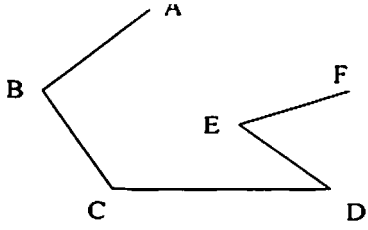
$$(x,y),(x+d/2,y),\left(x+d/4,y+d\sqrt{3}/4\right).$$

Ushbu protsedurada d,n o'zgaruvchilaridan tashqari x,y o'zgaruvchilari ham parametr sifatida ko'rsatilishi lozim. Shunday qilib, dasturimiz quyidagi ko'rinishda bo'ladi:

```
Program SerpUchburchaklari;
uses graph, crt;
var n, Gd,gm: integer;
Procedure SerpUchburchak(x,y,d:real; n:integer);
begin
     If n=0 then
begin
{bu yerda 0-tartibli, ya'ni oddiy uchbuchak chiziladi}
Line(round(x),round(y),round(x+d/2),round(y-d*sqrt(3)/2));
{e'tibor bering, (y-d*sqrt(3)/2) ifodada ayirish amali} {
bajarilyapti,}
{chunki ekranda Oy o'qi yuqoridan pastga qarab } {yo'naltirilgan }
     Line(round(x+d/2), round(y-d*sqrt(3)/2), round(x+d),
round(y));
     Line(round(x), round(y), round(x+d), round(y));
end else
begin
     SerpUchburchak(x,y,d/2, n-1);
     SerpUchburchak(x+d/2,y,d/2, n-1);
     SerpUchburchak(x+d/4, y-d*sqrt(3)/4,d/2, n-1);
end;
end;
{Asosiy dastur}
begin
     Readln(n);
     Gd:=detect;
     InitGraph(gd,gm, 'd:\turbo pascal 7.1\bgi');
     If GraphResult <> grOk then exit;
     SerpUchburchak(75,475,500, n);
     Readkey;
     Closegraph;
end.
```

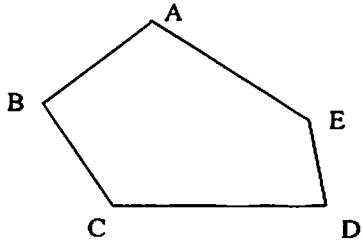
11.5. Ko'pburchak

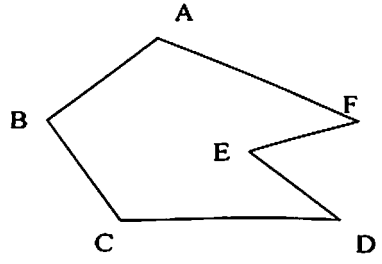
Birinchining uchi ikkinchisining oxiri bilan ketma-ket tutashtirilgan *AB*, *VS*, *CD*, *DE*, *EF* kesmalardan tuzilgan shakl *ABCDEF* siniq chiziq deyiladi.



Bunda *A*, *B*, *C*, *D*, *E*, F nuqtalar siniq chiziqning uchlari, *AB*, *VS*, *CD*, *DE*, *EF* kesmalar uning bo'g'inlari, *A* va *F* nuqtalar esa siniq chiziqning oxirlari deyiladi. Agar siniq chiziqning hech qanday uchta nuqtasi to'g'ri chiziqda yotmasa va uning hech qanday bo'g'inlari ichki nuqtalarda kesishmasa, u sodda siniq chiziq deyiladi. Siniq chiziqning bo'g'inlari uzunliklarining yig'indisi uning perimetri deyiladi. Ravshanki, *ABCDE* siniq chiziqning perimetri uning oxirlari orasidagi *AE* masofadan kichik emas.

Agar siniq chiziqning oxirlari ustma-ust tushsa, u yopiq siniq chiziq deyiladi.

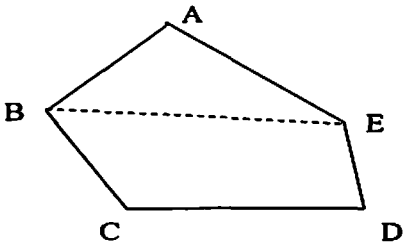




Tekislikning sodda yopiq siniq chiziq bilan chegaralangan qismi ko'pburchak deyiladi.

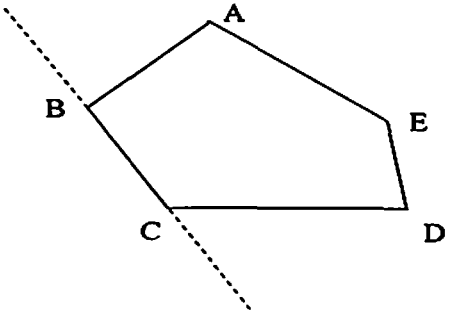
Siniq chiziqning uchlari va bo'g'inlari, mos ravishda, ko'pburchakning uchlari va tomonlari deyiladi. Tomonlari soni eng kam bo'lgan ko'pburchak uchburchakdan iborat. Ko'pburchakning nomi uning tomonlari soniga bog'liq ravishda aytiladi.

Ko'pburchakning bitta tomonida yotmagan ikkita uchini tutashtiruvchi kesma uning diagonali deyiladi.

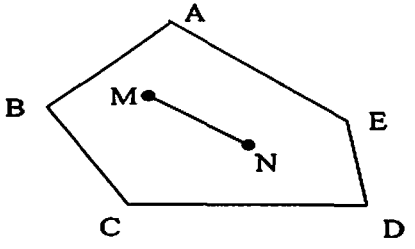


Har qanday ko'pburchak tekislikni ikki qismga bo'ladi: ko'pburchak tomonlari bilan chegaralangan qism ko'pburchakning ichki sohasi, ko'pburchakdan tashqarida yotgan qism uning tashqi sohasidir.

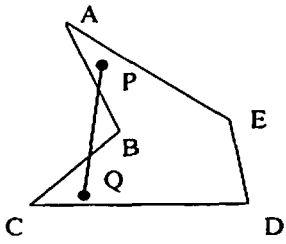
Bizga *ABCDE* ko'pburchak berilgan bo'lsin. Uning tomonlaridan istalgan bittasini, masalan, *BC* ni davom ettiramiz. Agar ko'pburchak shu *BC* to'g'ri chiziqning bir tomonida yotsa, u qavariq ko'pburchak deyiladi.



Qavariq ko'pburchakda uning ichki sohasidagi istalgan ikkita *M* va *N* nuqtani tutashtiruvchi *MN* kesma shu sohada to'liq yotadi.



Quyidagi ko'pburchakda esa uning ichki *P* va *Q* nuqtalarini tutashtiruvchi *PQ* kesma ko'pburchakning ichki sohasida ham, tashqi sohasida ham yotadi. Shu sababli u qavariq bo'lmagan ko'pburchak deyiladi.

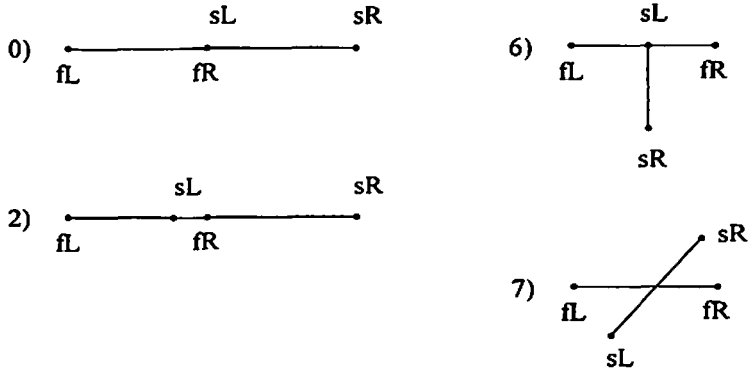


Agar qavariq ko'pburchakning: a) barcha tomonlari; b) barcha ichki burchaklari o'zaro teng bo'lsa, u muntazam ko'pburchak deyiladi.

Endi misollarga o'tamiz.

Oddiy ko'pburchak. Ko'pburchakning uchlari ketma-ket kiritiladi. Uning yonma-yon bo'lmagan istalgan ikki tomonlari umumiy nuqtaga ega bo'lmasa, u holda uni oddiy ko'pburchak deymiz. Berilgan ko'pburchakni oddiyligini aniqlash talab etiladi.

Masala yechimi. Bu yerda tomonlar kesishmasa va bir-biriga ustma-ust tushmasa, demak bu oddiy ko'pburchak. Bu yerda biz oldin ishlab chiqilgan *SegmCross* qism dasturidan foydalanamiz. Bu yerda uchlar *A* massivida soat mili yo'nalishi bo'yicha berilgan va unda quyidagi holatlarni mavjud emasligini tekshirib chiqamiz.



Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*

5 0 0 0 2 2 4 3 3 4 0	Poligon oddiy

Bu yerda *input.txt* faylida nuqtalar soni $n=5$ va *x1, y1* ko‘pburchakning uchlari koordinata qiymatlari kiritiladi va *output.txt* faylida ko‘pburchak qanday ekanligi haqida ma’lumot chiqarilgan bo‘ladi.

```
program Poligon_oddiy;
const Nmax=10;
      eps=0.0000000001;
      {-haqiqiy sonlarni taqqoslashda ishlatiladi}
Type
      TPoint=Record  x,   u:  Real;
end;
type
      arrtype = array[1..Nmax] of TPoint;
Type
      Tline = Record A, B, C : Real;
end;
Var  A: Arrtype ;
      n, i : integer;
      fayl:text;
  {qo‘shimcha funksiyalar}
{ Chiziqlarning kesishgan nuqtasi koordinatalarini } {aniqlash qism
dasturi}
Procedure  Line2ToPoint (Const L1,   L2:   TLine; var  P: Tpoint);
var st:   real;
begin
      st:=L1.A*L2.B-L2.A*L1.V;
      P.X:=(-L1.C*L2.B+L2.C*L1.B)/st;
      P.Y:=(-L1.A*L2.C+L2.A*L1.C)/st;
end;
Procedure Point2ToLine(Const v,w:TPoint; var L:TLine);
{Ikki nuqta bo‘yicha to‘g‘ri chiziq tenglamasidagi }
{A,B,C larni aniqlash}
```

```
begin
      L.A:=v.y-w.y;
      L.B:=w.x-v.x;
      L.C:= v.x* w.y - v.y* w.x;
end;
Function EqNumber(Const a, b :real): Boolean;
      {Ikki sonni tengligini aniqlash}
begin
      EqNumber:=(abs(a- b)<eps) ;
end;
Function EqPoint(Const A, B : TPoint): Boolean;
      {Ikki nuqtani tengligini aniqlash}
begin
EqPoint:=(abs(A.x- B.x)<eps) And (abs(A.y-B.y)<eps);
end;
Function SegmLineCross( const fL, fR, sL, sR : Tpoint) : Byte;
{Ikki kesma bir chiziqda}
Var Minf, Maxf, Mins, Maxs: real;
Begin
if fl.x*fl.x+fl.y*fl.y < fr.x*fr.x+fr.y*fr.y then minf:=fl.x*fl.x+fl.y*fl.y
else minf:=fr.x*fr.x+fr.y*fr.y;
{fL yoki fR nuqtalardan (0;0) ga yaqinini aniqladik}
if fl.x*fl.x+fl.y*fl.y > fr.x*fr.x+fr.y*fr.y then
maxf:=fl.x*fl.x+fl.y*fl.y else maxf:=fr.x*fr.x+fr.y*fr.y;
{fL yoki fR nuqtalardan (0;0) dan  uzog'ini  aniqladik}
if sl.x*sl.x+sl.y*sl.y < sr.x*sr.x+sr.y*sr.y then
mins:=sl.x*sl.x+sl.y*sl.y else mins:=sr.x*sr.x+sr.y*sr.y;
{sL yoki sR nuqtalardan (0;0) ga yaqinini aniqladik}
if sl.x*sl.x+sl.y*sl.y > sr.x*sr.x+sr.y*sr.y then
maxs:=sl.x*sl.x+sl.y*sl.y else maxs:=sr.x*sr.x+sr.y*sr.y;
{sL yoki sR nuqtalardan (0;0) dan  uzog'ini  aniqladik}
If Eqnumber(Minf, Maxs) or  Eqnumber(Maxf, Mins) then
SegmLineCross :=0
{kesmaning bitta uchi ikkinchi kesmaning uchiga teng}
Else
If Mins>=Maxf) or  (Minf>= Maxs) then SegmLineCross :=1
{ bitta nuqta ikkinchi kesmaning ichida bo'lmaydi }
Else SegmLineCross :=2;
{-bitta nuqta ikkinchi kesmaning ichida bo'ladi}
```

```
End;
Function  CrossLine(Const L1, L2 : TLine): Boolean;
      {Ikki chiziq kesishganligini aniqlash}
var st: real;
begin
      st:= L1.A * L2.B – L2.A * L1.B;
      CrossLine:=not (abs(st) < eps);
end;
Function  AtSegm(Const A, B, P : TPoint): Boolean;
      {P nuqtasi kesmada ekanligini aniqlash}
begin
If EqPoint(A,B) then AtSegm:= EqPoint (A,P)
      {- A va B nuqtalari teng}
      Else
{AP va PB vektorlar ko'paytmasidan }
AtSegm:= EqNumber ( ((B.x-A.x)*(P.y-A.y)), ((B.y-A.y)*(P.x-A.x)) )
And (( (P.x>=A.x) And (B.x>=P.x)) or ( (P.x>=B.x) And
(A.x>=P.x)));
{P nuqtasi o'rtada bo'lishi kerak}
end;
Function SegmCross (Const fL, fR, sL, sR: Tpoint): Byte;
var rs:TPoint;
      L1,L2:Tline;
      b, bRS: Boolean;
begin
      Point2ToLine (fL,  fR, L1) ;
      Point2ToLine (sL,  sR,  L2);
      If CrossLine (L1,L2)   Then
      {-ya'ni ikki chiziq kesishsa, u holda}
begin
Line2ToPoint(L1,L2,rs);
{-ikki chiziq,kesma emas,kesishgan nuqtasini rs } {aniqladik }
bRS:=EqPoint (rs,fL)  Or EqPoint (rs, fR)  Or EqPoint (rs,sL)  Or
EqPoint (rs,sR);
{ya'ni  kesishgan  nuqta kesmalar uchlarining biriga  } {tengligini
tekshiramiz}
b:=EqPoint(sL,fL)  Or EqPoint(sR, fR)  Or EqPoint(sR,fL)  Or
EqPoint(sL,fR);
{ya'ni ikki kesma chetki nuqtalarini tengligini } {tekshiramiz}
```

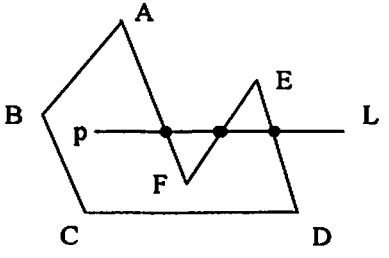
```
      If b Then SegmCross:=5
      Else
If AtSegm(fL,fR,rs)  And AtSegm (sL, sR, rs) and ( not bRS)  Then
SegmCross:= 7
      Else
      If AtSegm(fL,fR,rs)  and  AtSegm (sL,sR,rs) and bRS   Then
SegmCross:=6
      Else SegmCross:=4;
end
      Else
      If  EqNumber (L1.A*L2.B, L2.A*L1.B) And Not (EqNumber
(L1.C,L2.C))
      Then SegmCross:=3
      Else
      SegmCross :=  SegmLineCross ( fL,  fR, sL,  sR);
end;
Function IsPoligonSimple(Const A: Arrtype ; Const N:Word)
:Boolean;
{ ko'pburchak oddiy bo'lsa natija True bo'ladi, aks }
{holda False}
var    i, j: integer;
      pp:boolean;
begin
      pp:=True; i:=1;
while  (i<=N-1)  And pp Do
begin
      j:=i+1;
while  (j<=N) And pp Do
begin
Case SegmCross (A[i] ,A[i+1] ,A[j] ,A[ (j+1) Mod N]) Of
{ikki kesmaning o'zaro joylashuvi, [ (j+1)  Mod N] – bu}
{1-nuqtani tutashtiradi}
      0,2,6,7:      pp:=False;
end;
      Inc(j) ;
end;
      Inc(i);
end;
      IsPoligonSimple:=pp;
```

```
end;
  {dasturning asosiy qismi}
begin
      assign(fayl, 'input.txt'); reset(fayl);
      readln(fayl,n); {nuqtalar soni}
      for i:=1 to n do readln(fayl, A[i].x, A[i].y);
      close(fayl);
      assign(fayl, 'output.txt'); rewrite(fayl);
      If IsPoligonSimple( A, N) then
      Write(fayl, 'Poligon oddiy') else
      Write(fayl, 'Poligon oddiy emas');
      close(fayl);
end.
```

Keyingi misolimiz juda ko'p uchraydigan misollardan biri.

Ichki nuqta. Berilgan p nuqtani oddiy Q ko'pburchak ichida joylashganligini aniqlash talab etiladi.

Masala yechimi. Bu yerda p nuqtadan Ox o'qiga parallel bo'lgan L to'g'ri chiziqni o'tkazamiz. Agar L chizig'i Q ko'pburchakni kesib o'tmasa, demak p nuqtasi Q ga tashqi bo'ladi. Agar p nuqtadan bir tomonga qarab to'g'ri chiziq o'tkazsak, uning ko'pburchak bilan kesishgan nuqtalar W soni toq bo'ladi.



Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 1 1 0 0 0 2 2 4	*Nuqta ichkari*

3 3 4 0	

Bu yerda *input.txt* faylida ko'pburchak uchlarining soni n=5, berilgan nuqta koordinatalari va ko'pburchakning uchlari qatorlab ketma-ket kiritiladi va *output.txt* faylida nuqta qayerda joylashganligi haqida ma'lumot chiqarilgan bo'ladi.

Shulardan kelib chiqqan holda quyidagi *IncludPoint* qism dasturni va undan foydalanish dasturini taklif etamiz:

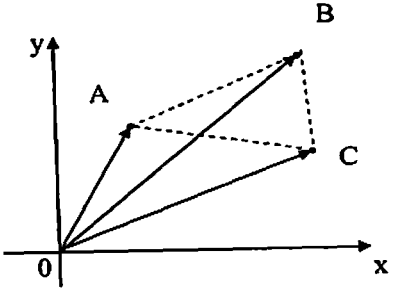
```
program Poligon_nuqta;
const Nmax=10;
eps=0.0000000001;
{-haqiqiy sonlarni taqqoslashda ishlatiladi}
Type
TPoint=Record  x, y: Real;
end;
type
arrtype = array[0..Nmax] of TPoint;
var
A: Arrtype;
p : TPoint;
n,i : integer;
fayl:text;
{qo'shimcha funksiyalar}
Function EqNumber(Const a, b :real): Boolean;
{Ikki sonni tengligini aniqlash}
begin
EqNumber:=(abs(a- b)<eps) ;
end;
Function EqPoint(Const A, B : TPoint): Boolean;
{Ikki nuqtani tengligini aniqlash}
begin
EqPoint:=(abs(A.x- B.x)<eps) And (abs(A.y-B.y)<eps);
end;
Function  AtSegm(Const A, B, P : TPoint): Boolean;
{P nuqtasi kesmada ekanligini aniqlash}
begin
If EqPoint(A,B) then AtSegm:= EqPoint (A,P)
```

```
{- A va B nuqtalari teng}
Else
{AP va PB vektorlar ko'paytmasidan }
AtSegm:= EqNumber ( ((B.x-A.x)*(P.y-A.y)), ((B.y-A.y)*(P.x-A.x)) )
And (( (P.x>=A.x) And (B.x>=P.x)) or ( (P.x>=B.x) And
(A.x>=P.x)));
{P nuqtasi o'rtada bo'lishi kerak}
end;
Function IncludPoint(Const A: Arrtype; Const N: Word; Const  P:
TPoint) :  Boolean;
var
i, nxt,  sc:   Integer;
Lf,  rg:  TPoint;
nx:  Real;
begin
sc:=0;      IncludPoint:=True;
For i:=0 To N-1 Do
begin
{ ko'pburchakning har bir tomoni uchun L chiziqning } {kesishgan
nuqtasining koordinatalarini aniqlaymiz}
nxt:=(i+1)  Mod N;
If AtSegm(A[i] ,  A[nxt],  P)  Then  Exit;
{ P nuqtani (A[i] ,A[nxt]) kesmada bo'lishini aniqladik}
Lf:=A[i];  rg:=A[i];
If (A[i] .y <  A[nxt].y)  Then  rg:=A[nxt]
Else  Lf:=A[nxt];
If  (Lf.y <  p.y) And   (p.y<=  rg.y)    Then
begin
Nx:=((-rg.x+Lf.x)*P.y-Lf.x*rg.y+Lf.y*rg.x)/ (-rg.y+Lf.y);
{Ikki nuqtadan V va W lardan o'tuvchi chiziq  tenglamasiga} { P.y
ni qo'yib, kesishgan nuqtaning x }
{koordinatasini aniqladik }
If (nx>P.x) Then Inc(sc);
{Ya'ni P dan o'ng tomonga qarab }
end;
end;
IncludPoint:=Odd(sc) ;{sc ni toqligini aniqladik }
end;
{dasturning asosiy qismi}
```

```
begin
assign(fayl, 'input.txt');
reset(fayl);
readln(fayl, n);
readln(fayl, p.x, p.y); {P nuqta koordinatalari: x, y}
writeln(p.x, p.y);
for i:=1 to n do
readln(fayl, A[i-1].x, A[i-1].y);
close(fayl);
{Endi hisoblab, natijani chop etamiz }
assign(fayl, 'output.txt');
rewrite(fayl);
If IncludPoint( A, N, P) then
Writeln(fayl, 'Nuqta ichkari') else
Writeln(fayl, 'Nuqta tashqari');
close(fayl);
end.
```

Ko'pburchak yuzasi. Berilgan oddiy Q ko'pburchak yuzasini hisoblash talab etiladi.

Masala yechimi. Koordinatalar sistemasining boshini ko'pburchakni uchlari bilan tutashtiramiz. Birin-ketin uchburchaklarni vektorli ko'paytma orqali ishorali yuzasini hisoblab chiqamiz. Quyidagi chizmada uchburchak misolida yuzani hisoblashni keltiramiz.



Bu yerda OAB va OBC uchburchaklar yuzalari (S_0 va S_1) hisoblanganda, ularning ishorasi bir xil bo'ladi. Undan keyin esa OCA yuzasi S_2 hisoblanib, oldingi yig'indining keraksiz qismini ayirib tashlaydi, natijada faqat ABC uchburchak yuzasi qoladi.

$$S=S_0+S_1+S_2$$

Vektorli ko'paytma orqali quyidagilarni hosil qilamiz

$S_0 = x_0 \cdot y_1 - x_1 \cdot y_0$

$S_1 = x_1 \cdot y_2 - x_2 \cdot y_1$

$S_2 = x_2 \cdot y_0 - x_0 \cdot y_2$

Dasturlashda yengil bo'lishi uchun, ushbu yig'indini quyidagicha guruhlab yozamiz

$S=S_0+S_1+S_2 = x_0 \cdot y_1 - x_1 \cdot y_0 + x_1 \cdot y_2 - x_2 \cdot y_1 + x_2 \cdot y_0 - x_0 \cdot y_2 =$

$= x_1 \cdot (y_2 - y_0) + x_2 \cdot (y_0 - y_1) + x_0 \cdot (y_1 - y_2)$

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
4 0 0 0 4 4 4 4 0	-16

Bu yerda *input.txt* faylida ko'pburchak uchlarining soni $n=4$ va ko'pburchakning uchlari qatorlab ketma-ket kiritiladi va *output.txt* faylida ko'pburchakning ishorali yuzasi chiqarilgan bo'ladi. Agar nuqtalar teskari o'qib olinsa, u holda natija musbat bo'ladi.

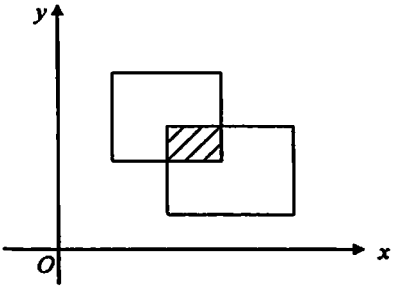
Shundan kelib chiqqan holda quyidagi *Square* qism dasturini va undan foydalanish dasturini taklif qilamiz:

```
program Poligon_Ishorali_yuza;
const Nmax=10;
Type
     TPoint=Record x,  u:  Real;
End;
type
     arrtype = array[0..Nmax] of TPoint;
var
     A: Arrtype;
     result : real;
     n,i : integer;
     fayl:text;
  {qo'shimcha funksiyalar}
Function Square (Const A:  Arrtype;  Const  N: Word): real;
{ Ishorali yuza}
var
     i: Word;
```

```
      sc:   real;
begin
      if N<3  Then Square:=0 Else
begin
      Sc:=A[0] .x* (A[1] .y-A[N-1] .y) +A[N-1] . x*(A[0] .y-A[N-2] .y)
;
{- birinchi va oxirgi nuqtalar orqali aniqlangan yuza}
      For i:=1   To N-2 Do
       sc:=sc+A[i]. x*(A[i+1] .y-A[i-1] .y) ;
       Square:=sc/2;
end;
end;
  {dasturning asosiy qismi}
Begin
      assign(fayl, 'input.txt'); reset(fayl);
      readln(fayl, n);
      for i:=1 to n do
      readln(fayl, A[i-1].x, A[i-1].y);
      close(fayl);
{Endi hisoblab, natijani chop etamiz }
      assign(fayl, 'output.txt'); rewrite(fayl);
      result:= Square ( A,  N);
      Writeln(fayl, result); {Ko'pburchak  yuzasi}
      close(fayl);
end.
```

Kesishmalar yuzasi. Tekislikda N ta to'g'ri burchakli to'rtburchaklar chap pastki va o'ng yuqoridagi uchlari koordinatalari orqali butun sonlarda berilgan. Ularning tomonlari koordinata o'qlariga parallel bo'lsa, ularning umumiy kesishgan maydonining yuzasini aniqlang.

Masala yechimi. Bu yerda Ox va Oy o'qlari bo'yicha kesmalarning umumiy kesishgan nuqtalarini aniqlash orqali maydon yuzasini hisoblashimiz mumkin.



Agar to'rtburchaklar kesishmasa, bu haqida ham dasturimiz xabar berishi lozim.

Qiymatlarni kiritishda to'g'ri to'rtburchakning chap yuqori va o'ng pastki burchaklari koordinatalarini butun sonlarda beriladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
2 0 2 4 4 3 3 6 6	1

Bu yerda *input.txt* faylida ko'pburchaklarning soni n=2 va ko'pburchakning ikki uchlari koordinatalari *x1, y1, x2, y2* qatorlab ketma-ket kiritiladi va *output.txt* faylida ko'pburchaklarning umumiy kesishgan maydonining yuzasi chiqarilgan bo'ladi, agar ular kesishmasalar natijada nolni chop etamiz.

Shundan kelib chiqqan holda quyidagi *Kesishma* dasturini taklif qilamiz:

```
Program Kesishma;
Const Nmax=10;
Type
      Segm=Record L, R: Real;
end;
      SegmArray=Array[ 1.. Nmax] of Segm;
Var OnX, OnY: SegmArray; ObX, ObY: Segm;
      N:integer;
      fayl:text;
  {qo'shimcha funksiyalar}
Procedure     Init;
var i :  Integer;
```

```
      x1, y1,  x2, y2: real;
begin
      assign(fayl,'input.txt');
      reset(fayl);
      readln(fayl, n);
      For i:=1 To N   Do
begin
      readln(fayl, x1, y1,  x2, y2);
      OnX[i].L:=x1; OnX[i].R:=x2;
      OnY[i].L:=y1; OnY[i].R:=y2;
end;
      close(fayl);
end;
Function    Cross  (A,B:Segm;Var New:Segm) : Boolean;
      {a va b kesmalarning kesishini aniqlash}
begin
      If B.L<A.L  Then  New.L:=A.L  Else  New.L:=B.L;
      If  A.R<B.R  Then  New.R:=A.R Else New.R:=B.R;
      Cross:=New.L<New.R;
end;
Function SolSegm(Const On:SegmArray;Var Ob:Segm): Boolean;
  { kesmalarning kesishgan  qismini  aniqlash}
var  i : Integer;
begin
      Ob:=On[1]; i:=2;
      While   (i<=N) And Cross (On [i] ,Ob,    Ob)   Do   Inc(i);
      SolSegm:=i>N;
end;
      {dasturning asosiy qismi}
begin
      Init;
      {Endi hisoblab, natijani chop etamiz }
      assign(fayl, 'output.txt');
      rewrite(fayl);
      If SolSegm(OnX, ObX)  And SolSegm(OnY,  ObY) Then
      WriteLn(fayl, Abs ( (ObX.L-ObX.R)*(ObY.L-ObY.R)) :0:0)
Else
      WriteLn (fayl, '0');
      close(fayl);
```

end.

Qavariq ko'pburchak. Berilgan Q ko'pburchakni qavariq ekanligini aniqlash talab etiladi.

Masala yechimi. Ko'pburchakni qavariq ekanligini tekshirish ko'pgina amaliy ishlanmalarda qo'llaniladi. Buni aniqlashda biz ko'pburchak uchlarini uchtadan birin-ketin tartib bilan ko'rib chiqamiz va ishorali maydon yuzasini hisoblaymiz. Uchburchaklar ushbu tartib bo'yicha bitta ishorali yuzaga ega bo'lishlari kerak, aks holda ko'pburchak noqavariq bo'ladi.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
5 0 0 0 4 4 4 4 0 2 2	qavariq emas

Bu yerda *input.txt* faylida ko'pburchakning uchlari soni n=5 va uning uchlari koordinatalari qatorlab ketma-ket kiritiladi va *output.txt* faylida ko'pburchakning "*qavariq*" yoki "*qavariq emas*"ligini chop etamiz.

Shundan kelib chiqqan holda quyidagi *IsPoligonConvex* qism dasturini va undan foydalanish *Qavariq* dasturini taklif qilamiz:

```
Program Qavariq;
const Nmax=10;
      eps=0.0000000001;
      {-haqiqiy sonlarni taqqoslashda ishlatiladi}
Type
      TPoint=Record  x, u: Real;
end;
type
      arrtype = array[0..Nmax] of TPoint;
var
      A: Arrtype ;
      n:integer;
      fayl:text;
```

```
  {qo'shimcha funksiyalar}
Procedure  Init;
var  i : Integer;
begin
      assign(fayl, 'input.txt');
      reset(fayl);
      readln(fayl, n);
      for i:=0 to n-1 do
      readln(fayl, A[i].x, A[i].y);
      close(fayl);
end;
Function  Sq_Vector(p1,p2,p3: TPoint) : Real;
{uchburchakning ishorali yuzasi}
begin
      Sq_Vector:=( (p2.x-p1.x) * (p3.y-p1.y) -(p2.y-p1.y) * (p3.x-
p1.x)  )/2 ;
end;
Function EqNumber(Const a, b :real): Boolean;
      {Ikki sonni tengligini aniqlash}
begin
      EqNumber:=(abs(a- b)<eps) ;
end;
Function  PoligonQavariq(Const  A:Arrtype;  Const  N:  Integer):
Boolean;
{ko'pburchak qavariq bo'lsa, natija True bo'ladi}
var
      nw, bn:  Byte;
{nw – yuzaning ishorasi uchun, bn – ishorani }
{o'zgarishini nazorat qilish uchun}
      i: Integer;
      rp:  Real; {maydon yuzasi uchun}
begin
      PoligonQavariq:=False;
      bn:=1;
      For i:=1 To N-1   Do
begin
      rp:=Sq_Vector(A[i-1],A[i], A[(i+1) Mod N]) ;   {ishorali
maydon}
      if EqNumber (rp, 0)  Then    nw:=1
```

```
    {- nuqtalar bitta chiziqda joylashgan }
    Else if (rp<0) Then nw:=0 Else nw:=2;
    If (bn=1) Then bn:=nw
    Else If (nw<>1) And (nw<>bn) Then Exit;
{uchlar bitta chiziqda emas, maydon ishoralari har }
{xil, shu bois ko'pburchak qavariq emas}
end;
    PoligonQavariq:=True;
end;
  {dasturning asosiy qismi}
begin
    Init;
    {Endi hisoblab, natijani chop etamiz }
    assign(fayl, 'output.txt');
    rewrite(fayl);
    If PoligonQavariq (A, N) Then
    WriteLn(fayl, 'qavariq') Else
    WriteLn (fayl, 'qavariq emas');
    close(fayl);
end.
```

11.6. Ikkinchi tartibli chiziqlar

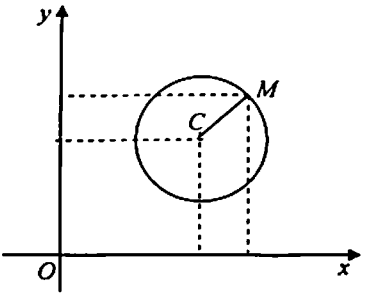
Tekislikda to'g'ri chiziq x va y o'zgaruvchi koordinatalarga nisbatan birinchi darajali bo'ladi. Endi tekislikda ikkinchi tartibli chiziqlarni o'rganamiz. Ikkinchi tartibli chiziqlar x va y o'zgaruvchi koordinatalarga nisbatan ikkinchi darajali tenglama bilan ifodalanadi. Ikkinchi darajali tenglamaning umumiy ko'rinishi

$$Ax^2 + 2Bxy + Cy^2 + Dx + Ey + F = 0$$

bo'ladi. Bu tenglamaga ikkinchi tartibli chiziqning umumiy tenglamasi deyiladi. Quyida muayyan hollarda, ikkinchi tartibli chiziqlarning analitik ifodalarini topib, ularning xususiyatlarini o'rganamiz.

Tekislikda biror $C(a,b)$ nuqtadan teng uzoqlikda joylashgan nuqtalar geometrik o'rniga aylana deyiladi.

$M(x,y)$ aylanaga tegishli ixtiyoriy nuqta bo'lsin. Aylana ta'rifiga ko'ra CM masofa o'zgarmas, bu masofani R bilan belgilaylik.



Ikki nuqta orasidagi masofani topish formulasiga asosan,

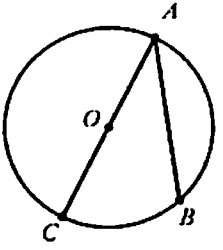
$$CM = \sqrt{(x-a)^2 + (y-b)^2} \text{ёки} \sqrt{(x-a)^2 + (y-b)^2} = R$$

bo'ladi. Oxirgi tenglikning ikkala tomonini kvadratga ko'tarib,

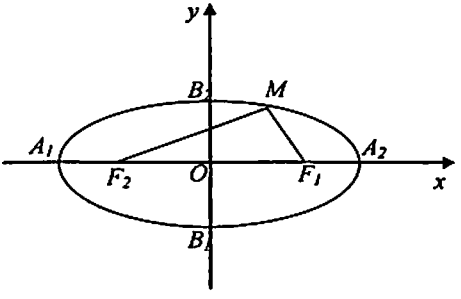
$$(x-a)^2 + (y-b)^2 = R^2$$

tenglamaga kelamiz. Bu tenglamaga markazi $C(a,b)$ nuqtada, radiusi R ga teng aylananing kanonik (qonuniy) tenglamasi deb aytiladi.

Aylananing ikkita A, B nuqtasini tutashtiruvchi AB kesma aylananing vatari deyiladi. Aylananing markazidan o'tuvchi AC vatar diametr deyiladi.



Tekislikda, har bir nuqtadan berilgan ikkita nuqtalargacha bo'lgan masofalar yig'indisi o'zgarmas miqdordan iborat bo'lgan nuqtalar geometrik o'rniga ellips deyiladi. Berilgan nuqtalar F_1 va F_2 bo'lsin.



Bu nuqtalarga ellipsning fokuslari deyiladi. O'zgarmas miqdorni $2a$, fokuslar orasidagi masofani $2c$ bilan belgilab, koordinatalar sistemasini shunday olamizki, Ox o'qi fokuslardan o'tsin va koordinatalar boshi F_1F_2 masofaning o'rtasida bo'lsin. $M(x,y)$ ellipsga tegishli ixtiyoriy nuqta bo'lsa, ta'rifga ko'ra

$$F_1M + F_2M = 2a$$

bo'ladi. Ma'lumki, $F_1(+c;0)$ va $F_2(-c;0)$ bo'lib, ikki nuqta orasidagi masofani topish formulasiga asosan:

$$\sqrt{(x-c)^2+(y-0)^2} + \sqrt{(x+c)^2+(y-0)^2} = 2a$$

tenglamani hosil qilamiz. Bu tenglamadan irratsionallikni yo'qotib,

$$x^2(a^2-c)^2 + a^2y^2 = a^2(a^2-c^2)$$

ko'rinishga keltiramiz. $a^2-c^2=b^2$ bilan belgilaymiz (chunki, $a>c$). Bu holda

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

tenglamani hosil qilamiz. Bu tenglamaga ellipsning kanonik tenglamasi deyiladi.

Koordinatalar boshi, ellipsning simmetriya markazi, koordinatalar o'qi simmetriya o'qlari bo'ladi.

$A_1(-a,0), A_2(a,0), B_1(0,-b), B_2(0,+b)$

nuqtalar ellipsning uchlari, $a=OA_2$ va $b=OB_2$ masofalar mos ravishda ellipsning katta va kichik yarim o'qlari deyiladi.

Tekislikda, har bir nuqtadan berilgan ikkita (fokus) nuqtalargacha bo'lgan masofalar ayirmasi o'zgarmas miqdordan iborat bo'lgan nuqtalar geometrik o'rniga giperbola deyiladi (ko'rsatilgan ayirma absolyut qiymati bo'yicha olinib, u fokuslar orasidagi masofadan kichik va noldan farqli).

O'zgarmas miqdorni $2a$, fokuslar orasidagi masofani $2c$ va koordinata o'qlarini ellipsdagidek olib, $c^2-a^2=b^2$ belgilash kiritib,

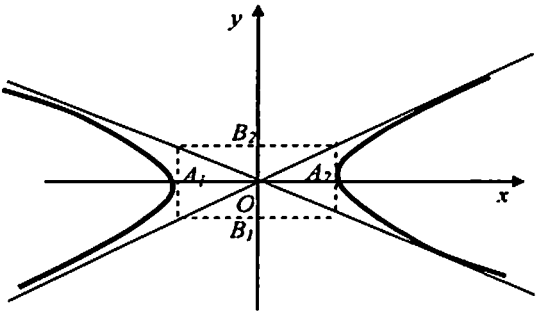
$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$$

tenglamani hosil qilamiz. Bu tenglamaga giperbolaning kanonik tenglamasi deyiladi. Giperbolaning fokuslari $F_1(+c;0)$ va $F_2(-c;0)$ bo'ladi. Koordinatlar o'qi simmetriya o'qlari va koordinatalar boshi $O(0;0)$ simmetriya markazidir. Giperbola koordinata o'qlarini $A_1(-a;0)$ va $A_2(a;0)$ nuqtalarda kesib o'tib, bu nuqtalarga haqiqiy uchlari va $a=OA_2$ masofa haqiqiy yarim o'qi deyiladi. $B_1(0;-b)$ va $B_2(0;b)$ nuqtalar giperbolaning mavhum uchlari, $b=OB_2$ - mavhum yarim o'qi deyiladi.

Giperbola ikkita asimptotalarga ega bo'lib, uning tenglamalari

$$y = \pm \frac{b}{a} x$$

bo'ladi.



Giperbola o'qlari $a=b$ bo'lsa, unga teng tomonli giperbola deyiladi va uning tenglamasi

$$x^2 - y^2 = a^2$$

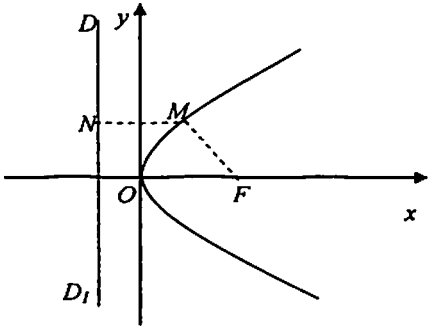
bo'ladi.

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1, -\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

giperbolalarga o'zaro qo'shma giperbolalar deb ataladi.

Tekislikda, har bir nuqtadan berilgan nuqta(fokus)gacha va berilgan to'g'ri chiziq (direktrisa)gacha masofalari o'zaro teng bo'lgan nuqtalar geometrik o'rniga parabola deyiladi.

Koordinatalar sistemasini shunday olamizki, OX o'qi F fokusdan o'tib, DD_1 direktrisaga perpendikulyar, OY o'qi esa fokus va direktrisaning o'rtasidan o'tsin.



$M(x,y)$ parabolaga tegishli ixtiyoriy nuqta bo'lsin. F nuqtadan DD_1 to'g'ri chiziqqacha bo'lgan masofani p ($p>0$) bilan belgilaymiz. Bunda $F(p/2; 0)$ bo'lib, direktrisaning tenglamasi

$$x = -\frac{p}{2}$$

bo'ladi. Ta'rifga asosan, $MN=MF$, $N(-p/2; y)$.
Ikki nuqta orasidagi masofa formulasiga asosan,

$$x + p/2 = \sqrt{(x - p/2)^2 + y^2}.$$

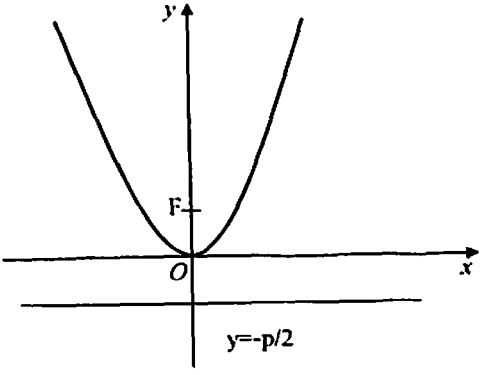
Bu tenglamadan irratsionallikni yo'qotib,

$$y^2 = 2px$$

tenglamani hosil qilamiz. Bu absissalar o'qiga simmetrik parabolaning kanonik tenglamasi bo'ladi. Ordinatlar o'qi simmetriya o'qi bo'lsa, parabola tenglamasi

$$x^2 = 2py(p > 0)$$

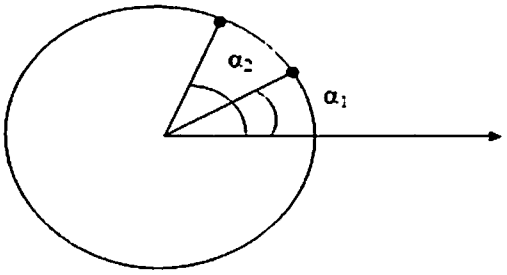
ko'rinishda bo'ladi. Bu holda $y=-p/2$ direktrisa tenglamasi, $F(0; p/2)$ nuqta fokus bo'ladi



$M(x,y)$ nuqtadan $F(p/2;0)$ fokusgacha masofaga fokal radius deyiladi va $r=x+p/2$.

Aylanadagi masofa. Aylanada N ta nuqta berilgan va ularning hech biri ustma-ust tushmagan. Ularning joylashuvi α burchak ($1 \leq \alpha \leq 360$ va butun sonlar) bilan berilgan va u gradusda o'lchanadi. Shunday nuqtani aniqlash kerakki, undan barcha nuqtalargacha bo'lgan masofalar yig'indisi minimal bo'lsin va unga tegishli nuqtalar o'sish tartibida chop etilsin. Masofa aylana bo'yicha bo'lib, burchakda o'lchanadi va faqatgina eng yaqin yo'l inobatga olinadi.

Masala yechimi. Masalaning sharti bo'yicha ikki nuqta orasidagi masofa bevosita ular orasidagi burchak bilan o'lchanadi.



Shundan kelib chiqqan holda ushbu masofani quyidagicha aniqlaymiz:

$$min(|\alpha_2 - \alpha_1|, 360 - |\alpha_2 - \alpha_1|)$$

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
3 180 270 360	1 2

Bu yerda *input.txt* faylida birinchi qatorda nuqtalar soni va qolgan qator-larda ushbu nuqtalar burchagi berilgan bo'ladi, *output.txt* faylida esa birinchi qatorda masalani shartini qanoatlantiruvchi nuqtalar soni va qolgan qatorlarda ushbu nuqtalar tartib raqamlari o'sish tartibida chiqarilgan bo'ladi.

Dastur esa quyidagi ko'rinishda bo'ladi:

```
program AylanaMin;
var
    n, i, j, k, t, min, sum : longint;
    a, b : array [1..360] of integer;
    fayl:text;
function Minimum(a, b : longint) : longint;
begin
    if a < b then Minimum := a else Minimum := b;
end;
begin
    assign(fayl, 'input.txt');
```

```
        reset(fayl);
        readln(fayl, n);
        for i := 1 to n do
begin
        readln(fayl, a[i]);
end;
        close(fayl);
        k := 0;
        min := 360*360;
        for i := 1 to n do
begin
        sum := 0;
        for j := 1 to n do
        if i<>j then
inc(sum,Minimum(360-abs(a[i]-a[j]),abs(a[i]-a[j])));
        if (sum < min) and (sum > 0) then
begin
        min  := sum;
        k := 1;
        b[k] := i;
end
        else
        if sum=min then
begin
        inc(k);
        b[k] := i;
end;
end;
        if n = 1 then
begin
        k := 1;
        b[k] := 1;
end;
        assign(fayl, 'output.txt');
        rewrite(fayl);
        writeln(fayl, k);
        for i := 1 to k do
        writeln(fayl, b[i]);
        close(fayl);
```

end.

E'tibor bering, dasturda nuqtalar soni bitta bo'lgan holat alohida ko'rib chiqilgan. Olimpiada masalalarida shunga o'xshash vaziyatlar e'tiborga olinmasa, testga yuborilgan dastur to'g'ri deb qabul qilinmaydi!

Topshiriqlar

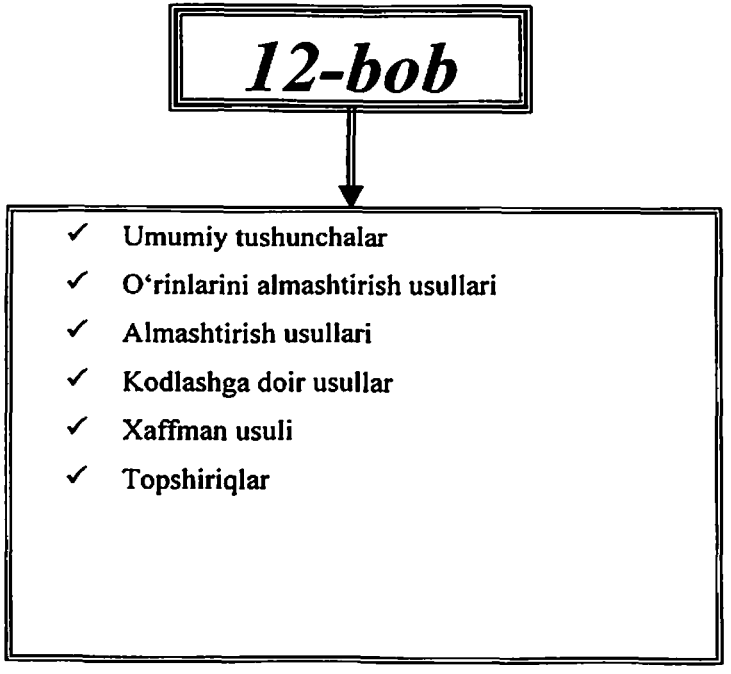
1. Tekislikda N ta nuqta berilgan bo'lsin. Barcha nuqtalarni o'z ichiga oladigan eng kichik qavariq ko'pburchakni aniqlang.
2. Ox o'qida N ta nuqta berilgan. Absissa o'qida shunday z nuqtani topingki, undan barcha N ta nuqtagacha bo'lgan masofalar yig'indisi minimal qiymatga ega bo'lsin.
3. Berilgan uchta nuqtadan o'tadigan aylana markazining koordinatalarini aniqlang.
4. Berilgan N ta nuqtani o'z ichiga oladigan aylananing minimal radiusini aniqlang.
5. Tekislikda N ta to'g'ri to'rtburchak tomonlari koordinata o'qlariga parallel bo'lib , ular bir-biri bilan kesishgan. Ularning chap yuqori va o'ng pastki uchlari koordinatalari butun sonlarda berilgan bo'lsa, ularning birlashmasidan hosil bo'lgan ko'pburchakning perimetrini aniqlang.
6. Tekislikda N ta uchdan iborat oddiy ko'pburchak berilgan bo'lib , uning tomonlari koordinata o'qlariga parallel berilgan. Uchlar koordinatalari butun sonlarda bo'lib, soat mili aylanishi bo'yicha berilgan. Ushbu ko'pburchakning ichidan barcha uchlarni ko'rish mumkin bo'lgan nuqta borligini aniqlash talab etiladi.
7. Tekislikda 2 ta uchburchak berilgan. Ulardan birini siljitish va aylantirish orqali ikkinchisining ichiga joylashtirish mumkinligini aniqlang.
8. Tekislikda qavariq ko'pburchak va 2 ta A va B nuqtalar berilgan. Ushbu nuqtalarni eng yaqin masofa bilan tutashtiring, bunda ko'pburchakni kesib o'tish mumkin emas.
9. Tekislikda 2 ta ta to'g'ri to'rtburchak berilgan. Ulardan birini siljitish va aylantirish orqali ikkinchisining ichiga joylashtirish mumkinligini aniqlang.
10. Tekislikda N ta nuqta berilgan bo'lib, ularning ichidan bir-biridan eng uzoqda joylashgan ikkita nuqtani toping.

11. Tekislikda N ta nuqta berilgan bo'lib, ularni kami bilan nechta to'g'ri chiziqda joylashtirish mumkinligini aniqlang.
12. Tekislikda A, B, C va D nuqtalaridan tashkil topgan $ABCDA$ yopiq siniq chiziq qavariq to'rtburchak ekanligini aniqlang.
13. Tekislikda ikkita aylananing kesishgan nuqtalarini aniqlang.
14. Tekislikda doira o'zining markazi va radiusi bilan berilgan. Berilgan kesmaning doiraga tegishli qismini aniqlang.
15. Tekislikda uchburchak uchlari bilan berilgan. Ushbu uchburchakni o'z ichiga oladigan eng kichik kvadratning tomonini aniqlang.
16. Tekislikda qavariq ko'pburchak uchlaridan $A_1, A_2,\ldots,A_n$ ikkitasini A va A_j larni belgilab, eng katta yuzaga ega bo'lgan $A_iA_jA_k$ uchburchakni aniqlovchi A_k uchini toping.

12-bob. KRIPTOGRAFIYA ALGORITMLARI VA ULARNI DASTURLASH

Ushbu bobda kriptografiyada mavjud asosiy tushunchalar va tarixiy ma'lumotlar keltirilgan bo'lib, bevosita ularga doir algoritmlar misollarda keltirilgan. Hozirgi kunda ushbu yo'nalishda olib boriladigan izlanishlar natijasida juda murakkab algoritmlar yaratilgan bo'lib, ularni o'rganishda ushbu bob ilk qadam hisoblanadi. Keltirilgan misollarni o'rganib chiqish orqali talabalarimiz kriptografiya sohasida mavjud asosiy g'oyalarni tushunib olishadilar va bu kelgusida zamonaviy dasturlarni yaratishda asos bo'lib xizmat qiladi.

Mazkur bobda keltirilgan mavzular quyidagi tartibda berilgan:



12.1. Umumiy tushunchalar

Islom dunyosida ilm-fanda o'rta asrlarda erishilgan yutuqlar albatta kriptografiya sohasini chetlab o'tib ketishi mumkin emas edi. Ushbu sohada yaratilgan va bizgacha yetib kelgan asarlardan biri Abu Bakr Axmad ibn Ali ibn Vaxshiya an-Nabati qalamiga mansub bo'lib (olimlarning ismlari Internet manbalaridan olingan), u 855-yillarda yaratilgan va qadimiy qo'lyozmalarni o'qish muammolariga bag'ishlangan. Ushbu kitobda ikki alifboli kriptografiya usullari ham yoritilgan va unda har xil shifrlash tizimlari keltirilgan va ular XIX asrlargacha qo'llanilib kelingan.

Chastotali kriptotahlilga bag'ishlangan asar 855-yilda yaratilgan va Abu Yusuf al-Kindi qalamiga mansub bo'lib, unda kriptografik xabarlarni deshifrlash muaamolari o'rganib chiqilgan.

Al-Kindining kriptotahlilga bag'ishlangan qo'lyozmasi bizgacha yetib kelgan va unda qadimiy xalqlarning alifbosiga asoslangan shifrlash usullari ham berilgan. Ushbu algoritm XIX asrgacha qo'llanilib kelingan.

Ushbu sohada to'plangan ma'lumotlar misrlik matematik Shaxob Kalkashandi tomonidan 1412-yilda yozilgan 14 jildlik «Shauba al-Asha» asarida jamlangan. Unda shifrlashning 7 ta usuli bilan birga, deshifrlash masalalari ham ko'rib chiqilgan. Bunda arab tiliga xos statistika va lingvistika qonuniyatlariga asoslanib noma'lum shifrlangan xabarlarni o'qish yo'llari keltirilgan. Ushbu qomusiy asarning ajoyib jihatlaridan biri bu ilk bor kriptotahlilda chastotali tahlilni qo'llanilganligi. Ushbu usul orqali oddiy o'rinlarini almashtirish usullari orqali shifrlangan matnlari o'qilgan. "Shifr" so'zining fandagi o'rni ushbu asarlardan kelib chiqqan.

Maxfiy xatlarni o'qishda qabul qilingan tartib qoidalarga asoslanib tayanch so'zlar negizida matnni deshifrlashni ilk bor VIII asrda Xalil al-Faraxidi e'tibor bergan. Masalan, xatni biz "salom" so'zidan boshlaymiz, demak matndagi beshta harfni qanday belgilab olinganlini aniqlash mumkin bo'ladi. Ushbu g'oyani Xalil al-Faraxidi «Kitab al-Maumma» asarida yoritgan.

Shu bois Shaxob Kalkashandi kriptografik tahlilning asoschisi sifatida e'tirof etilgan, ammo milliy adabiyotlarda bu haqida nimagadir so'z yuritilmaydi.

Axborotni himoyalash uchun kodlashtirish va kriptografiya usullari qo'llaniladi.

Kodlashtirish deb axborotni bir tizimdan boshqa tizimga ma'lum bir belgilar yordamida belgilangan tartib bo'yicha o'tkazish jarayoniga aytiladi.

Kriptografiya deb maxfiy xabar mazmunini shifrlash, ya'ni ma'lumotlarni maxsus algoritm bo'yicha o'zgartirib, shifrlangan matnni yaratish yo'li bilan axborotga ruxsat etilmagan kirishga to'siq qo'yish usuliga aytiladi.

Kriptografiya nuqtayi nazaridan shifr – bu kalit demakdir va ochiq ma'lumotlar to'plamini yopiq (shifrlangan) ma'lumotlarga o'zgartirish kriptografiya o'zgartirishlar algoritmlari majmuasidir.

«Kriptografiya» so'zi dastlab «yashirish, yozuvni berkitib qo'ymoq» ma'nosini bildirgan. Birinchi marta u yozuv paydo bo'lgan davrlardayoq aytib o'tilgan. Hozirgi vaqtda kriptografiya deganda har qanday shakldagi, ya'ni diskda saqlanadigan sonlar ko'rinishida yoki hisoblash tarmoqlarida uzatiladigan xabarlar ko'rinishidagi axborotni yashirish tushuniladi. Kriptografiyani raqamlar bilan kodlanishi mumkin bo'lgan har qanday axborotga qo'llash mumkin. Maxfiylikni ta'minlashga qaratilgan kriptografiya kengroq qo'llanilish doirasiga ega. Aniqroq aytganda, kriptografiyada qo'llaniladigan usullarning o'zi axborotni himoyalash bilan bog'liq bo'lgan ko'p jarayonlarda ishlatilishi mumkin.

Kriptografiyaning vazifasi xabarlarning maxfiyligini va haqiqiyligini ta'minlashdan iboratdir.

Kriptografiya – ma'lumotlarni o'zgartirish usullarining to'plami bo'lib, ma'lumotlarni himoyalash bo'yicha quyidagi ikkita asosiy muammolarni hal qilishga yo'naltirilgan:

- **maxfiylik;**
- **yaxlitlik.**

Maxfiylik orqali yovuz niyatli shaxslardan axborotni yashirish tushunilsa, **yaxlitlik** esa yovuz niyatli shaxslar tomonidan axborotni o'zgartira olmaslik haqida dalolat beradi.

Kalit – kriptografiya o'zgartirishlar algoritmining ba'zi-bir parametrlarining maxfiy holati bo'lib, barcha algoritmlardan yagona variantni tanlaydi. Kalitlarga nisbatan ishlatiladigan asosiy ko'rsatgich bo'lib kriptomustahkamlik hisoblanadi.

Kriptografiya himoyasida shifrlarga nisbatan quyidagi talablar qo'yiladi:

- yetarli darajada kriptomustahkamlik;
- shifrlash va qaytarish jarayonining oddiyligi;

- axborotlarni shifrlash oqibatida ularning hajmining ortib ketmasligi;
- shifrlashdagi kichik xatolarga ta'sirchan bo'lmasligi.

Ushbu talablarga quyidagi tizimlar javob beradi:

- o'rinlarini almashtirish;
- almashtirish;
- gammalashtirish;
- analitik o'zgartirish.

O'rinlarini almashtirish shifrlash usuli bo'yicha boshlang'ich matn belgilarining matnning ma'lum bir qismi doirasida maxsus qoidalar yordamida o'rinlari almashtiriladi.

Almashtirish shifrlash usuli bo'yicha boshlang'ich matn belgilari foydalanilayotgan yoki boshqa bir alifbo belgilariga almashtiriladi.

Gammalashtirish usuli bo'yicha boshlang'ich matn belgilari shifrlash gammasi belgilari, ya'ni tasodifiy belgilar ketma-ketligi bilan birlashtiriladi.

Analitik o'zgartirish usuli bo'yicha boshlang'ich matn belgilari analitik formulalar yordamida o'zgartiriladi, masalan, vektorni matritsaga ko'paytirish yordamida. Bu yerda vektor matndagi belgilar ketma-ketligi bo'lsa, matritsa esa kalit sifatida xizmat qiladi.

12.2. O'rinlarini almashtirish usullari

Ushbu usul eng oddiy va eng qadimiy usuldir. O'rinlarini almashtirish usullariga misol sifatida quyidagilarni keltirish mumkin: **shifrlovchi jadval** va **sehrli kvadrat.**

Shifrlovchi jadval usulida kalit sifatida quyidagilar qo'llaniladi:

- jadval o'lchovlari;
- so'z yoki so'zlar ketma-ketligi;
- jadval tarkibi xususiyatlari.

Misol. Quyidagi matn berilgan bo'lsin:

TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI SAMARQAND FILIALI

Ushbu axborot ustun bo'yicha ketma-ket jadvalga kiritiladi:

T	N	R	N	Y	U	S	S	A	I
O	T	O	O	A	N	I	A	N	A
S	A	T	L	L	I	T	M	D	L
H	X	T	O	A	V	E	A	F	I
K	B	E	G	R	E	T	R	I	*
E	O	X	I	I	R	I	Q	L	*

Natijada, 6x10 o'lchovli jadval tashkil qilinadi. Endi shifrlangan matn qatorlar bo'yicha aniqlanadi, ya'ni o'zimiz uchun 4 tadan belgilarni ajratib yozamiz:

TNRN EUSS AIOT OOAN IANA SATL LITM DLHX TOAV EAFI KBEG RETR I*EO XIIR IQL*

Bu yerda kalit sifatida jadval o'lchovlari xizmat qiladi.

Ushbu usulni murakkablashtirish maqsadida tayanch so'zni kiritsa bo'ladi. Yuqoridagi misol uchun quyidagi

TEZLASHMOQ

so'zini olamiz va oldingi jadvalga joylashtiramiz:

T	E	Z	L	A	S	H	M	O	Q
9	2	10	4	1	8	3	5	6	7
T	N	R	N	Y	U	S	S	A	I
O	T	O	O	A	N	I	A	N	A
S	A	T	L	L	I	T	M	D	L
H	X	T	O	A	V	E	A	F	I
K	B	E	G	R	E	T	R	I	*
E	O	X	I	I	R	I	Q	L	*

Ikkinchi qatordagi raqamlar harflarning alifbo harflari ketma-ketligi bo'yicha tartiblashdan kelib chiqadi. Shu qatordagi raqamlar bo'yicha ustunlarni tartiblaymiz:

A	E	H	L	M	O	Q	S	T	Z
1	2	3	4	5	6	7	8	9	10
Y	N	S	N	S	A	I	U	T	R
A	T	I	O	A	N	A	N	O	O

L	A	T	L	M	D	L	I	S	T
A	X	E	O	A	F	I	V	H	T
R	B	T	G	R	I	*	E	K	E
I	O	I	I	Q	L	*	R	E	X

Shifrlangan matn quyidagi ko'rinishda bo'ladi:

YNSN SAIU TRAT IOAN ANOO LATL MDLI STAX EOAF IVHT RBTG RI*E KEIO IIQL *REX

Sehrli kvadrat deb katakchalariga 1 dan boshlab sonlar yozilib, undagi har bir ustun, satr va diagonal bo'yicha sonlar yig'indisi bitta songa teng bo'lgan kvadrat shaklidagi jadvalga aytiladi.

Sehrli kvadratga sonlar tartibi bo'yicha belgilar kiritiladi va bu belgilar satrlar bo'yicha o'qilganda matn hosil bo'ladi.

Misol. 4x4 o'lchovli sehrli kvadratni olamiz, bu yerda sonlarning 880 ta har xil kombinatsiyasi mavjud. Quyidagicha ish yuritamiz:

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Boshlang'ich matn sifatida quyidagi matnni olamiz:

DASTURIY VOSITASI

va jadvalga joylashtiramiz:

I	S	A	T
U	O	S	Y
V	R	I	I
T	S	A	D

Shifrlangan matn jadval elementlarini satrlar bo'yicha o'qish natijasida tashkil topadi: **IUVT SORS ASIA TYID**

12.3. Almashtirish usullari

Almashtirish usullari sifatida quyidagi usullarni keltirish mumkin:

- Sezar usuli;
- Affin tizimidagi Sezar usuli;
- Tayanch so'zli Sezar usuli va boshqalar.

Sezar usulida almashtiruvchi harflar k ta siljish bilan aniqlanadi. Sezar bevosita k=3 bo'lganda ushbu usuldan foydalangan.

k=3 bo'lganda va alifbodagi harflar m=26 ta bo'lganda quyidagi jadval hosil qilinadi:

A→D	F→I	K→N	P→S	U→X
V→E	G→J	L→O	Q→T	V→Y
S→F	H→K	M→P	R→U	W→Z
D→G	I→L	N→Q	S→V	X→A
E→H	J→M	O→R	T→W	Y→B
				Z→C

Matn sifatida **SAMARQAND** so'zini oladigan bo'lsak, Sezar usuli natijasida quyidagi shifrlangan yozuv hosil bo'ladi: **VDPDUTDQG**.

Sezar usulining kamchiligi bu bir xil harflarning, o'z navbatida, bir xil harflarga almashishidir.

Masaladagi ma'lumotlar quyidagicha kiritiladi va chiqariladi:

input.txt	*output.txt*
3 SAMARQAND	VDPDUTDQG

Bu yerda *input.txt* faylida birinchi qatorda *k* va ikkinchi qatorda shifrlanadigan matn kiritiladi va *output.txt* faylida shifrlangan matn chiqarilgan bo'ladi.

Shundan kelib chiqqan holda quyidagi dasturni tuzamiz:

```
program Cesar;
const n=26; {lotin alifbosidagi harflar soni }
var
i, k: integer;
p: string;
fayl : text;
begin
      assign(fayl, 'input.txt');
      reset(fayl);
readln (fayl , k);
readln (fayl , p);
{siljitish sonini va matnni o'qib oldik}
close (fayl);
for i:=1 to length(p) do
```

```
if p[i] in ['A'..'Z'] then
begin
p[i]:= chr((ord(p[i])-ord('A')+k) mod n+ord('A'));
{bu yerda mod n bevosita harflar chegasidan o'tib ketmaslikni
ta'minlaydi}
end;
      assign(fayl, 'output.txt');
rewrite(fayl);
writeln (fayl ,p);
close (fayl);
end.
```

Affin tizimidagi Sezar usulida har bir harfga almashtiriluvchi harflar maxsus formula bo'yicha aniqlanadi: at+b (mod m), bu yerda a,b- butun sonlar, 0≤ a, b<m, EKUB(a,m)=1. Quyida m=26, a=3 va b=5 bo'lganda hosil bo'lgan jadval keltirilgan va shunga mos ravishda harflar almashtiriladi:

T	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
alifbo	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
3t+5	5	8	11	14	17	20	23	0	3	6	9	12	15	18	21	24	1	4	7	10	13	16	19	22	25	2
Harf	F	I	L	O	R	U	X	A	D	G	J	M	P	S	V	Y	B	E	H	K	N	Q	T	W	Z	C

Natijada yuqorida keltirilgan **SAMARQAND** matni quyidagicha shifrlanadi: **HFPFEBFSO.**

Tayanch so'zli Sezar usulida siljitish bilan birgalikda tayanch so'z qo'llaniladi. Tayanch so'zni qo'llashdan maqsad hosil qilinadigan alifboda harflar ketma – ketligini o'zgartirishdir.

Misol. k=5 va **DIPLOMAT** tayanch so'zini olamiz va bu so'z k – o'rindan yoziladi:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
					D	I	P	L	O	M	A	T													

Ushbu tayanch so‘z alifbodagi ko‘rsatilgan joyda joylashtiriladi, undagi harflar inobatga olinmasdan,

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	B	C		E	F	G	H		J	K			N			Q	R	S		U	V	W	X	Y	Z
					D	I	P	L	O	M	A	T													

qolgan harflar alifbodagi tartib bo‘yicha tayanch so‘zdan keyin ketma – ket yoziladi va natijada quyidagi hosil qilinadi:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
V	W	X	Y	Z	D	I	P	L	O	M	A	T	B	C	E	F	G	H	J	K	N	Q	R	S	U

Yuqorida ko‘rib chiqilgan **SAMARQAND** so‘zi esa mazkur usul yordamida **HVTVGFVBY** ga o‘tkaziladi.

12.4. Kodlashga doir usullar

Kodlash bilan bog‘liq bo‘lgan misollardan birini ko‘rib chiqamiz. Berilgan quyidagi natural sonlarni sonli faylda saqlash talab etiladi. Bu yerda har bir songa ikki bayt ajratilgan bo‘lsin:

1020 1012 1013 1013 1013 1008 1010 1020 1032 1050 1036 ...

Ushbu sonlarga nazar tashlaydigan bo‘lsak, ular bir-biridan katta farq qilmasligini kuzatish mumkin. Ya’ni bizda ushbu farq 128 dan oshmaydi. Demak, faylda barcha sonlarni saqlash shart emas, faqatgina ularning farqini yozib qo‘yish kifoya bo‘ladi. Faqat bunda birinchi elementni belgilab olish kerak bo‘ladi. Agar bizda bu 1000 bo‘ladigan bo‘lsa, unda quyidagi natijani olamiz:

1000 +20 -8 +1 0 0 -5 +2 +10 +12 +18 -14 ...

Shunday qilib, ushbu algoritm orqali sonlarni kodlab oldik va natijada ushbu faylning hajmini qariyb 50 foizga kamaytirdik. Ma’lumotlarni bunday kodlash usuli keng qo‘llaniladi, masalan, raqamlashtirilgan audio yozuvlarda.

Keyingi kodlash usullaridan biri bu RLE (Repeated Running Length Encoding, yoki qisqacharoq– Run Length Encoding deb yuritiladi). Bunda asosiy maqsad takrorlanuvchi belgilarni ixchamlash hisoblanadi. Masalan, matn bevosita bir baytli belgilardan iborat bo'lsin:

ABBCCCDDDDEEEEE ...

Bunda RLE usuli natijasi quyidagicha bo'ladi:

1A2B3C4D5E ...

Umumiy holda bu yerda siqilish koeffitsiyentini oshirish faqatgina maxsus fayllarda amalga oshirilishi mumkin.

12.5. Xaffman usuli

ASCII jadvalida keltirilgan asosiy matn belgilaridan tashkil topgan M to'plamidagi belgilarni ikkilik ko'rinishiga optimal kodlash talab etiladi. Bunda optimallikga erishish uchun M to'plamidagi har bir belgini maxsus algoritm orqali nol va birlar bilan kodlash zarur bo'ladi. M to'plamiga mos kelmagan belgilarni kodlash va natijaviy fayldan joylashtirish talab etilmaydi. Kodlangan natijadan birlamchi matn dekodlash orqali aniq bo'lishi kerak. Optimallik esa bu yerda kodlangan natijada nol va birlarning umumiy soni minimal bo'lishi kerakligini anglatadi. Bu yerda optimal algoritm quyidagi tamoyil asosida quriladi: ko'p uchraydigan belgilar kichik uzunlikdagi kod orqali belgilanishi shart. Buning uchun quyidagi qadamlar bajariladi:

1. Matnda uchraydigan barcha belgilar ro'yxatini shakllantiramiz va unda ushbu belgidan nechta borligini yozamiz.
2. Ushbu ro'yxatdan ikkita eng kam uchraydigan belgilarni tanlab olamiz. Ularning sonini yig'indisini olib, uni ro'yxatga qo'shamiz va ularni birlamchi ro'yxatdan uchiramiz.
3. Ro'yxatda ikkita son qolguncha 2-bandni davom ettiramiz.
4. Ro'yxatda qolgan birinchi songa 0 beramiz va ikkinchisiga esa 1.
5. Bu yerda orqaga qaytish jarayonini boshlaymiz, ya'ni yig'indini ajratib va ro'yxatga elementni qo'shib boramiz. Bunda, agar biz ajratadigan sonimiz $X=a_1a_2..a_m$ (a_i= 0 yoki 1) bo'lsa, unda ajratib olingan birinchi songa nol qo'shilgan X0 va ikkinchi songa esa bir

qo‘shilgan X1 to‘g‘ri kelishi kerak (keyingi misollarda batafsil tushuntirilgan).

6. Oxirgi 5-bandni barcha yig‘indilarni ajratib olguncha davom ettiramiz.

Natijada ketma-ketlikdagi belgilar uchun ularning ikkilik ko‘rinishidagi kodlari shakllanadi.

Endi Xaffman (ingl. Huffman) algoritmini murakkab misolda ko‘rib chiqamiz. Umumiy holda ushbu algoritm orqali faylni siqishda birinchi navbatda faylni to‘liq o‘qib olamiz va unda ASCII jadvalidagi har bir belgi necha marta takrorlanishini hisoblab chiqish talab etiladi. Shu bois ushbu algoritm orqali oddiy matnli va EXE fayllarni siqish bir xil amalga oshiriladi.

Har bir belgining chastotasini aniqlab, ularni kamayish tartibi bilan joylashtiramiz. Aniq qadamlarni bajarishni quyidagi misolda ko‘rib chiqamiz. Tarixiy manbalarda keltirilishicha, buyuk Amir Temurning uzukida forscha “rosti-rusti” degan so‘zlar bitilgan ekan. Keling, shu so‘zlarni kodlaymiz. Demak, bizga 11 baytdan iborat matn ROSTI-RUSTI berilgan va unda jami bo‘lib 7 ta har xil belgilar mavjud. Belgilarning chastotasini hisoblab, quyidagi jadvalni shakllantiramiz:

Belgi	R	U	S	T	I	O	-
Chastotasi	2	1	2	2	2	1	1

Bu yerdan eng kichik chastotalarni o‘ng tomonda yig‘ib quyidagi jadvalni hosil qilamiz:

Belgi	R	I	S	T	U	O	-
Chastotasi	2	2	2	2	1	1	1

Ushbu jadvaldan eng kichik chastotali ikkita belgini tanlaymiz, masalan, ‘O’ (1) va ‘-‘ (1) bo‘lsin. Ularning chastotalarini yig‘indisini olamiz:

Chastotasi	2	2	2	2	1	1	1
Belgi	R	I	S	T	U	O	-
						2=1+1	

Ikkita belgini birlashtirish natijasida hosil bo‘lgan yangi belgining chastotasi 1+1=2 bo‘ldi (albatta, bu son chastotani anglatmaydi, ammo mazmunan chastota deb yuritsak xato bo‘lmaydi). Keyingi qadamda ushbu ‘O’ va ‘-‘ belgilari inobatga olinmaydi, faqatgina ularning yig‘indisi jarayonda ishtirok etadi. Yana ushbu jadvaldan eng kichik

chastotali ikkita belgini tanlaymiz, masalan, 'U' va 'O-' bo'lsin. Ularning chastotalarini yig'indisini olamiz:

Chastotasi	2	2	2	2	1	1	1
Belgi	R	I	S	T	U	O	-
						2=1+1	
					3=1+2		

Yana ushbu jadvaldan eng kichik chastotali ikkita belgini tanlaymiz, masalan, 'R' va 'I' bo'lsin. Ularning chastotalarini yig'indisini olamiz:

Chastotasi	2	2	2	2	1	1	1
Belgi	R	I	S	T	U	O	-
						2=1+1	
					3=1+2		
	4=2+2						

Yana ushbu jadvaldan eng kichik chastotali ikkita belgini tanlaymiz, bunda faqatgina 'S' va 'T' bo'lishi mumkin. Ularning chastotalarini yig'indisini olamiz:

Chastotasi	2	2	2	2	1	1	1
Belgi	R	I	S	T	U	O	-
						2=1+1	
					3=1+2		
	4=2+2						
			4=2+2				

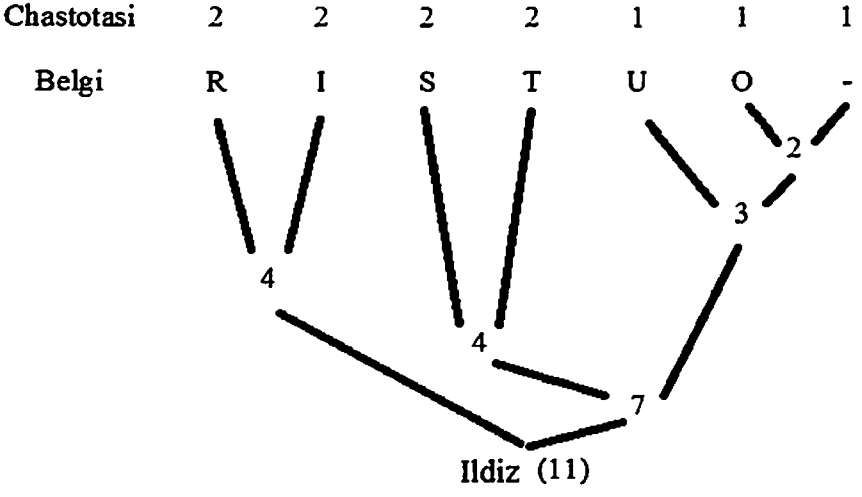
Yana ushbu jadvaldan eng kichik chastotali ikkita belgini tanlaymiz, bunda faqatgina 'ST' va 'UO-' bo'lishi mumkin. Ularning chastotalarini yig'indisini olamiz:

Chastotasi	2	2	2	2	1	1	1
Belgi	R	I	S	T	U	O	-
						2=1+1	
					3=1+2		
	4=2+2						
			4=2+2				
					7=4+3		

Endi ushbu jadvaldan eng kichik chastotali ikkita belgi bular faqatgina 'RI' va 'STUO-' bo'ladi. Ularning chastotalarini yig'indisini olamiz:

Chastotasi	2	2	2	2	1	1	1
Belgi	R	I	S	T	U	O	-
						2=1+1	
					3=1+2		
	4=2+2						
			4=2+2				
			7=4+3				
	Ildiz (11)						

Endi jarayonni yaxshiroq tasavvur qilish uchun ushbu jsdvalni daraxt shaklida namoyon qilamiz:



Daraxt yaratildi va uning eng oxirgi nuqtasini **Ildiz** deb belgiladik. Daraxtning R nuqtasiga (yoki barg deb yuritiladi) o'tish uchun biz 'chap' yoki 'o'ng' tomonlarga burilishim kerak bo'ladi. Agar 'chap' tomonga burilish bajarilsa, unda 0 bitni yozamiz. Agar 'o'ng' tomonga burilish bajarilsa, unda 1 bitni yozamiz. Demak, R bargiga o'tish uchun **Ildiz** nuqtasidan 'chap' tomonga o'tamiz, ya'ni (4)

nuqtasiga va 0 bitni yozamiz, undan yana 'chap' tomonga o'tamiz va yana 0 bitni yozamiz. Demak, R belgisining Xaffman kodi 00 bo'ladi. Keyingi I belgisi uchun 'chap' dan 'o'ng' ga o'tamiz. Demak, I belgisining Xaffman kodi 01 bo'ladi. Shu algoritm orqali barcha belgilarning aniqlangan Xaffman kodlari va ularga ajratilgan bitlar soni quyidagi jadvalda jamlangan:

Belgi	R	I	S	T	U	O	-
Kod	00	01	100	101	110	1110	1111
Bitlar soni	2	2	3	3	3	4	4

Ushbu jadval orqali ROSTI-RUSTI matni quyidagi Xaffman kodiga aylantiriladi:

00111010010101111100110100101 01

Quyida keltirilgan dasturdan olingan natijada Xaffman kodlari jadvalda keltirilgan natijadan ozgina farq qiladi. Bu farq harflarni tartiblab yozishdagi farqdan yuzaga keladi, ya'ni

Belgi	-	I	O	R	S	T	U
Kod	010	101	011	110	111	00	100
Bitlar soni	3	3	3	3	3	2	3

Ushbu jadval orqali ROSTI-RUSTI matni quyidagi Xaffman kodiga aylantiriladi: 1100111110010101011010011100101. E'tibor bering, ikkala variantda ushbu kodlarning uzunligi o'zgarmas qoldi.

Ya'ni, jami bo'lib 31 ta bit orqali matn kodlandi, demak siqilish koeffitsiyenti 2,8 bo'ladi (=11*8\31= 2,8).

Ushbu koddan asl matnni tiklash uchun oxirgi keltirilgan jadvaldan foydalanamiz. E'tibor bering, har bir matn uchun ushbu jadval har xil shakllanishi mumkin, shu bois, arxivlash va dearxivlash masalalari uchun universal jadvalga ega bo'lish kerak bo'ladi.

Endi masalani murakkablashtirib, kattaroq matnni misol sifatida ko'rib chiqamiz:

SAMARQAND SAYQALI RO'YI ZAMIN AST

Belgi	S	A	M	R	Q	N	D	Y	L	I	Z	T	O	'	space
Chastotasi	3	7	2	2	2	2	1	2	1	3	1	1	1	1	4

Ushbu jadvaldan eng ko'p uchraydigan belgi – bu 'A' , demak eng qisqa kodlanadigan belgi ham 'A' bo'ladi.

Yuqorida keltirilgan algoritm asosida Xaffman dasturini tuzishda kirish faylida bosh harflarda matn beriladi va chiqishda matnning bitlardagi hajmi, kodlangan variantidagi hajmi va siqilish koeffitsiyenti verguldan keyin bitta raqam aniqlikda chiqariladi.

Quyidagi dastur asosida olingan natijalar matndagi belgilarni qanday kodlanishini pastdagi jadvalda keltirilgan :

Belgi	Chastotasi	Bitlar
A	7	00
Space	4	010
I	3	1110
S	3	1111
M	2	0110
N	2	0111
Q	2	1000
R	2	1001
Y	2	1010
D	1	10110
L	1	10111
O	1	11000
T	1	11001
Z	1	11010
'	1	11011

```
program Xaffman;
const MaxK = 1000;
var  k,a,b : array [1..MaxK] of longint;
bits : array [0..MaxK] of string[20];
sk : array [1..MaxK] of char;
free : array [1..MaxK] of boolean;
res : array [0..255] of string[20];
kj,  m, kk1, kk2 : longint;
str : string;
fayl : text;
procedure InputData;
{boshlang'ich qiymatlarni kirituvchi qism dastur}
var c:char;
```

```
  j, ij, nj :integer;
s: array [0..255] of longint;
begin
      assign(fayl, 'input.txt');
      reset(fayl);
      for ij:=0 to 255 do s[ij]:=0;
      {s[] indeksi ij  ASCII kodini bildiradi va uning chastotasi
saqlanadi}
      readln(fayl, str);
{fayldan qatorni o'qib oldik}
      for nj:=1 to length(str) do
            begin
            c:=str[nj]; {bitta belgini tanladik}
            inc(s[ord(c)]);
            {c belgisiga mos keluvchi kod indeksini s[ord(c)] bittaga
oshiramiz}
{natijada  s[] massivida har bir belgining chastotasi saqlanadi}
            end;
      j:=0;
      for ij:=0 to 255 do  {0...255 bu ASCII kodlari}
      if s[ij]<>0 then begin inc(j); k[j]:=s[ij]; sk[j]:=chr(ij) end;
{ k[j] massivida   matnda mavjud belgi chastotasi saqlanadi}
{ sk[ij] massivida   matnda mavjud belgi saqlanadi}
kj:=j; {kj - bu kiritilgan qatordagi jami takrorlanmas belgilar soni}
Close (fayl);
end;
function MinK:  longint;
{chastotalar ichidan eng kichigini aniqlaymiz va }
{uning o'rnini  m o'zgaruvchida saqlaaymiz va}
{uni free[] ro'yxatda false deb belgilaymiz}
var min, ij :  longint;
begin  ij:=1;
      while (not free[ij])  do inc(ij);
min:=k[ij]; m:=ij;
for ij:=m+1 to kk2 do
      if free[ij] and (k[ij]<min) then begin min:=k[ij]; m:=ij end;
MinK:=min; free[m]:=false
End;
Procedure SumUp;
```

```
{ Yig'indilarni hisoblaydigan qism-dastur}
Var s1, s2, m1, m2, ij,ii : longint;
Begin if  kj=1 then begin
            assign(fayl, 'output.txt');
      rewrite(fayl);
      Writeln(fayl,8*length(str),' ' ,length(str),' ',8);
      close(fayl);
      Exit; {faylda bitta belgili qator bo'lsa unda ishni tugatamiz}
      End;
For ij:=1 to kj do
      Begin free[ij]:=true; a[ij]:=0; b[ij]:=0 end;
kk1:=kj; kk2:=kj;
While kk1>2 do {iktadan elemetni olamiz, shu bois oxirida 1 ta yoki
2 ta element qoladi}
      Begin s1:=MinK; m1:=m; s2:=MinK; m2:=m;
      {chastotasi kichik bo'lgan ikkita qiymat va}
      { ularming o'rni tanlanildi, takrorlanmas ro'yxatdan}
      Inc(kk2);
      K[kk2]:=s1+s2; a[kk2]:=m1; b[kk2]:=m2;
 {ikki minimal chastotani qo'shdik va }
 {ularming o'rnini alohida a[] va b[] massivida saqlab qo'ydik}
      Free[kk2]:=true;
      Dec(kk1)
      End;
end;
Procedure BuildBits;
{ Kiritilgan belgilarni ikkilik raqamlar bilan kodlaydigan qism-
dastur}
var ij : integer;
Begin
      bits[kk2]:= '1'; free[kk2]:=false;
      bits[a[kk2]]:= bits[kk2]+'0';
      bits[b[kk2]]:= bits[kk2]+'1';
      ij:=MinK;
      bits[m]:= '0'; free[m]:=true;
      bits[a[m]]:= bits[m]+'0';
      bits[b[m]]:= bits[m]+'1';
      for ij:=kk2-1 downto 1 do
            if not free[ij] then begin
```

```
                    bits[a[ij]]:= bits[ij]+'0';
                    bits[b[ij]]:= bits[ij]+'1';
                end
end;
procedure OutputData;
{ Matnni kodlaydigan va chiqarib beradigan qism-dastur}
var b8, bh, ij : longint;
begin
     assign(fayl, 'output.txt');
rewrite(fayl);
     for ij:= 1 to kj do
       begin
         res[ord(sk[ij])]:=bits[ij];
     {bits[ij] massivda matndagi harflarning Xaffman kodlari
saqlanadi}
{    writeln (fayl,sk[ij], ' ', k[ij], ' ',bits[ij] );}
{agar barcha qiymatlarni chiqarish talab etilsa,unda ushbu
operatorni qo'llang}
       end;
     b8:= 8*length(str);
     bh:=0;
          for ij:=1 to length(str) do
     begin
     inc(bh, length(res[ord(str[ij])] ));

     write (fayl, res[ord(str[ij])] );
     end;
     writeln (fayl);
     writeln (fayl, b8, ' ' , bh, ' ' , b8/bh : 0:1);
 close (fayl);
end;
begin
          InputData;
          SumUp;
          {faylda bitta belgi bo'lsa unda ishni tugatamiz}
     If kj<>1then BuildBits;
     If kj<>1then OutputData;
end.
```

Bunda siqilish koeffitsiyenti 1 ga 2,2 ga (=33*8\120 = 2,2) teng bo'ladi. Ya'ni koddagi bir bit axborot matnning 2,2 bitli axborotiga teng bo'ladi.

Topshiriqlar

1. «ZOTAN» tayanch so'zli shifrlovchi jadval (4x5 o'lchovli jadval) asosida quyidagi kodni aniqlang ("QARDSAMQSALIAAMZNNYA").
2. «MAGISTR» tayanch so'zli shifrlovchi jadval (4x7 o'lchovli jadval) asosida quyidagi kodni aniqlang ("RDISTINQISAUYOSAQOMTTTNTDAIII").
3. «KOTIBA» tayanch so'zli shifrlovchi jadval (4x6 o'lchovli jadval) asosida quyidagi kodni aniqlang ("QGIARAIIZXOVDHLBTFAAIOTS").
4. «TAMOYIL» tayanch so'zli shifrlovchi jadval (4x7 o'lchovli jadval) asosida quyidagi kodni aniqlang ("RNJAABIUAALVISVTSAITNMIIKYII").
5. «BARSELONA» tayanch so'zli shifrlovchi jadvaldan (3x9 o'lchovli jadval) foydalanib quyidagi kodni asl ma'nosini aniqlang ("AESLUSVADRTAANIENAQIMTITRDV").
6. 4x4 o'lchovli sehrli kvadratdan foydalanib quyidagi xabarning mazmunini aniqlang "*ELTTALNKROIK**E".
7. 4x4 o'lchovli sehrli kvadratdan foydalanib quyidagi xabarning mazmunini aniqlang "*IRATFIRAOGYP**K"
8. 4x4 o'lchovli sehrli kvadratdan foydalanib quyidagi xabarning mazmunini aniqlang "IMOMYTAERUTRPG'OK".
9. 4x4 o'lchovli sehrli kvadratdan foydalanib quyidagi xabarning mazmunini aniqlang "*SANUNJYIRIJTRED".
10. 4x4 o'lchovli sehrli kvadratdan foydalanib quyidagi xabarning mazmunini aniqlang "*BXIRVTXAOTSO*RA".
11. Lotin alifbosida berilgan o'zbek tilidagi matnni shifrlashda birinchi harfni ikkinchi harfga, ikkinchi harfni uchinchi harfga va hokazo, oxirgi harfni birinchi harfga almashtirish orqali kodlash dasturini tuzing.
12. Birinchi misolda shakllangan kodlangan matndan asl matnni aniqlash dasturini tuzing.
13. Yuqorida keltirilgan dasturlarda bitta harfga siljitish orqali tuzilgan kodlash dasturini istalgan n ta harfga siljitish asosida umumiy dasturini tuzing.

14. RLE usulidan foydalanib matnni kodlash dasturini tuzing.
15. Alfavit harflari chastotasi Fibonachchi sonlariga mos keladi, ya'ni
a : 1 b : 1 c : 2 d : 3 e : 5 f : 8 g : 13 h : 21.
Ushbu harflarning Xaffman kodlarini aniqlang.
16. Bahouddin Naqshbandning bosh g'oyasi bo'lmish "Dil-ba yor-u, dast-ba kor" matnni Xaffman kodlari bilan belgilang.
17.Berilgan matnni 4 tadan belgi olib bloklarga ajratib, har bir blokni alohida quyidagi sxema bo'yicha kodlash dasturini tuzing:
1-belgini 3- o'ringa, 3-belgini 2- o'ringa, 2-belgini 4- o'ringa, 4-belgini 1- o'ringa,
18. Yuqoridagi misolda olingan kodli matnni dekodlash dasturini tuzing.
19.Matnda 25 harf bo'lib, uni 5x5 matritsaga qatorlab yozing va ustunlab chiqarish orqali kodlash dasturini tuzing.
20. Yuqoridagi misolda olingan kodli matnni dekodlash dasturini tuzing.
21. Bitlardan tashkil topgan $a_1, a_2, \ldots, a_n$ ketma-ketligi quyidagicha kodlanadi:

$$b_1 = a_1 \text{ va}$$

$$b_i = \begin{cases} 1, \text{ agar } a_i = a_{i-1}, \\ 0, \text{ aksincha} \end{cases}$$

barcha i=2,...,n uchun. Ushbu algoritmga asoslangan kodlash dasturini tuzing.
22. Yuqoridagi misolda olingan kodli bitlar ketma-ketligini tiklaydigan dekodlash dasturini tuzing.
23.Xatoliklarni tuzatish. Berilgan bitlar ketma-ketligini uzatishda har bir bit uch marta uzatiladi. Masalan, 1,0,1 kodlari quyidagicha uzatiladi: 1,1,1,0,0,0,1,1,1. Agar uchtalikda qaysi bit ikki marta takrorlansa, o'sha bit natija sifatida qabul qilinadi. Ya'ni yuqoridagi misolda qabul qilingan bitlar quyidagicha buzilgan bo'lsada: 1,1,1,0,1,0,1,1,1 natija baribir 1,0,1 bo'ladi.
Ushbu algoritmga asoslangan dekodlash dasturini tuzing.
24. Berilgan ketma-ketlikdagi harflarni har xil qadamga siljitishni quyidagicha amalga oshiramiz. Masalan $a_1, a_2, \ldots, a_n$ harflarni 5 tadan bloklarga ajratamiz. Har bir blok doirasida harflarni nechtaga siljitishda quyidagi jadvaldan foydalanamiz:

1 2 3 4 5
7 5 11 8 9

Ushbu algoritmga asoslangan kodlash va dekodlash dasturlarini tuzing.

25. Kodlangan matnda so'zlar ajratib yozilgan va unda quyidagi so'zlar mavjudligi aniqlangan:

SAMARQAND, AXBOROT, TEXNOLOGIYALARI, FILIAL, UNIVERSITET, KESH.

Har bir harf faqatgina bitta harf bilan kodlangan deb faraz qilib asl matnni tiklaydigan dastur tuzing.

26. Sezar usuli berilgan dasturda kichik harflarni ham inobatga olish imkonini shakllantirib dasturni qayta tuzing.

XULOSA

Keyingi yillarda dasturlashtirishga qaratilgan web-manzillar soni keskin oshib bormoqda. Ushbu manbalarda keltirilgan algoritmlarni yoshlarimizga yetkazib berish uchun ularni mantiqan bog'liq mavzularga birlashtirib, ushbu kitob yuzaga keldi. Bu yerda keltirilgan masalalar sinchiklab o'rganilgan bo'lib, misollar bilan boyitilgan. Keltirilgan dasturlar nafaqat **PascalABC** tilida, balkim **Delphi 7** muhitida ham tekshirilgan. Yaratilgan dasturlarning barchasi tekshirilgan bo'lsa-da, lekin xatolik va kamchiliklardan to'liq xolis bo'lmasligi ham mumkin. Shularga qaramasdan ushbu kitob yosh dasturlovchilarimizga haqiqiy sovg'a bo'ladi deb umid qilamiz.

Ko'p yillik dastur yozish tajribasidan kelib chiqqan holda dasturlovchilarga quyidagi foydali maslahatlarni va qanday sifatlarga ega bo'lishlari haqida quyidagi takliflarni keltiramiz:

1) Hushyorlik – har bir operatorni yozishda uning imkoniyatlarini va qo'llash chegarasini bilish;
2) Qadamba-qadam – har bir operatorni bajarilishini nazorat qilib turish orqali, noxush vaziyatlarga tushishning oldini olish;
3) Tajriba – dasturlovchi masalaga doir algoritmlar bilan tanish bo'lishi va shu orqali noo'rin tajribalarni kamaytirish;
4) Munozara – algoritmlarni boshqalar bilan birgalikda davra suhbatlarida tahlil qilish;
5) Qiymatlash – dasturni har xil boshlang'ich qiymatlar bilan tekshirish;
6) Tarqatish – yaratilgan algoritmni boshqalar bilan baham ko'rish;
7) Ehtiyotkorlik – dasturda har xil vaziyatlarni chetlab o'tish uchun ehtiyotkorlik choralarini ishlab chiqish;
8) Mualliflik – qo'llanilgan algoritmlar mualliflarini e'tirof etish;
9) Rivojlanish – texnik va dasturiy vositalarni rivojlanishi dasturni qo'llashga to'siq bo'lmasligini ta'minlash;
10) Kengaytirish – dasturni qo'llash chegarasini kengaytirish;
11) Foydalik – dastur insonlar uchun foydali bo'lishi;
12) Buzg'inchilik – dasturni insonga, jamiyatga, dasturiy va texnikaviy vositalarga zarar etkazmasligini ta'minlash.

Mazkur kitobda keltirilgan materiallar o'quvchilarga algoritmlar olamiga sayohatni amalga oshirish uchun imkon yaratadi. Bu yerda keltirilgan algoritmlarni maxsus kurslarda o'rganishni tashkil qilish maqsadga muvofiq bo'ladi.

Bundan tashqari kitobda keltirilgan algoritmlar bugungi kunda ishlab chiqarish uchun zarur bo'lgan dasturiy ta'minotlarni yaratishda ham foydali bo'ladi deb umid qilamiz.

Ushbu kitob yosh dasturlovchilarni tayyorlash yo'lida kichik bir qadam bo'lib, kelgusida yanada mukammal ishlangan kitoblar yaratishga asos bo'ladi deb umid qilamiz. Shu sabab, kelgusida jahonshumul dasturlovchilar bizning ham yurtimizdan yetishib chiqishiga umid qilamiz. Biz, avvalo, yurtimiz qadimda barcha ilmlarga ulkan hissa qo'shgan va boshqalarga ustoz, hamda o'rnak bo'lgan yurt ekanini hech qachon unutmasligimiz kerak. Yurtimiz va xalqimiz tarixiga nazar tashlaydigan bo'lsak, ilm-ma'rifat sohasida ijod qilgan buyuk ajdodlarimiz yurtimizni shon-shuhratini oshirishda ulkan xizmatlar qilganligini yodda saqlashimiz lozim. Shu bois ham, rivojlangan davlatlar zabt etgan yuqori cho'qqilarga bizlar ham yetib olishimiz mumkin va buning uchun esa o'qish, o'qish va yana o'qish bilan birgalikda izlanishlar olib borish uchun yaratilgan shart-sharoitlardan unumli foydalanishimiz lozim.

Tayanch so'zlar ko'rsatkichi

Bundan tashqari kitobda keltirilgan algoritmlar bugungi kunda ishlab chiqarish uchun zarur bo'lgan dasturiy ta'minotlarni yaratishda ham foydali bo'ladi deb umid qilamiz.

Ushbu kitob yosh dasturlovchilarni tayyorlash yo'lida kichik bir qadam bo'lib, kelgusida yanada mukammal ishlangan kitoblar yaratishga asos bo'ladi deb umid qilamiz. Shu sabab, kelgusida jahonshumul dasturlovchilar bizning ham yurtimizdan yetishib chiqishiga umid qilamiz. Biz, avvalo, yurtimiz qadimda barcha ilmlarga ulkan hissa qo'shgan va boshqalarga ustoz, hamda o'rnak bo'lgan yurt ekanini hech qachon unutmasligimiz kerak. Yurtimiz va xalqimiz tarixiga nazar tashlaydigan bo'lsak, ilm-ma'rifat sohasida ijod qilgan buyuk ajdodlarimiz yurtimizni shon-shuhratini oshirishda ulkan xizmatlar qilganligini yodda saqlashimiz lozim. Shu bois ham, rivojlangan davlatlar zabt etgan yuqori cho'qqilarga bizlar ham yetib olishimiz mumkin va buning uchun esa o'qish, o'qish va yana o'qish bilan birgalikda izlanishlar olib borish uchun yaratilgan shart-sharoitlardan unumli foydalanishimiz lozim.

Tayanch so'zlar ko'rsatkichi

Ilova. Misollarning C++ dasturlash tilidagi kodlari.

\\ Birlar soni

```
#include <iostream>
using namespace std;
int main()
{
	int n, count = 0;
	cin>> n;
	while (n!=0)
	{
		n = (n - 1) & n;
		count++;
	}
	cout<< count;
}
```

\\ Qoldiq

```
#include <iostream>
using namespace std;
int main()
{
	constintNmax = 20;
	int n, k, m, s, t;
	int A[Nmax] = { 0 };
	cin>> n >> k >> m;
	for (inti = 0; i<= n; i++)
	{
		cin>> A[n-i];
	}
	s = 0; t = 1;
	for (inti = 0; i<= n; i++)
	{
		s = (s + A[i] * t) % m;
		t = (t*k) % m;
	}
	cout<< s <<endl;
}
```

\\ Ikkilik sanoq tizimida M+1

```
#include <iostream>
using namespace std;
int main()
{
      const int Nmax = 100;
      int n = 0, j = 0;
      bool b = false;
      int A[Nmax];
      cin>> n;
      for (int i = 0; i< n; i++)
      {
            cin>> A[i];
      }
      b = true;
      for (int i = n-1; i>= 0; i--)
      {
            j = A[i];
            if (b)
                  A[i] = 1 - j;
            else A[i] = j;

            if (j == 0)
                  b = false;
      }
      if (b)
            cout<< 1;
      for (inti = 0; i< n; i++)
      {
            cout<< A[i];
      }
}
```

\\ Bo'linish alomati

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
```

```
        constint v[4] = {1,8,4,2 };
        string n = "";
        int s = 0, L, nol;
        cin>> n;
        L = n.length();
        for (int stop = 4 - (L % 4), i = 1; i< stop; i++)
        {
                n = "0" + n;
        }
        L = n.length();
        nol = int('0');
        for (int stop = L, i = 1; i< stop; i++)
        {
                s = s + v[i % 4] * (int(n[i]) - nol);
        }
        if (s % 15 == 0)
        {
                cout<< "YES";
        }
        else {
                cout<< "NO";
                }
}
```

\\ Tub

```
#include <iostream>
using namespace std;
constint m=201;
int main()
{
        int n,k,q,i;
        int p[m]={0};
        cout<<"N=";
        cin>>n;
        if(n>=2) cout<<2<<" ";
        if(n>=3) cout<<3<<" ";
        k=0;
        p[k]=3;
        i=5;
```

```
    while(i< n) {
        for(int j=0; j <= k; j++)
        {
            q = p[j];
            if(q * q >i) {
                goto BR;
            }
            if(i % q == 0){
                goto NP;
            }
        }
        if(k == m-1)
            i=n-1;
BR:
        cout<<i<<" ";
        if(k <= m-1)
            p[++k]=i;
NP:     i+=2;
    }
}
```

\\ **TubB**

```
#include <iostream>
using namespace std;
int main(){
    intn,i,j;
    cout<<"N=";
    cin>>n;
    i=2;j=0;
    while(n>1){
        while(n % i!=0) i++;
        if(i!=j){
            cout<<i<<" ";
            j=i;
        }
        n/=i;
    }
}
```

```
\\ SundaramTubSon
#include<iostream>
#include<fstream>
using namespace std;
constintMaxK=1001;
intn,k,i,j;
long a[MaxK];
int main()
{
      ifstream fin("input.txt");
      ofstream fout("output.txt");
      fin>>n;
      for(i=1; i<=n; i++) a[i]=0;
      i=1;
      while(3*i+1<n){
            j=1;
            k=i+j+2*i*j;
            while(k<n && j<=i){
                  a[k]=1;
                  j++;
                  k=i+j+2*i*j;
            }
            i++;
      }
      for(i=n-1; i>=1; --i)
            if(a[i]==0){
                  k=2*i+1;
                  fout<<k<<" ";
            }
}

\\ Mukammal son
#include<bits/stdc++.h>
#define LL long long
using namespace std;
int main ()
{
  LL son, boluvchi=2, sum=0; cin>>son;
  for (int i=2; i<=son; i++)
```

```
    {
      while(boluvchi*boluvchi<=i)
      {
       if (i%boluvchi==0 && boluvchi*boluvchi!=i)
          sum=sum+boluvchi+i/boluvchi;
        else if (i%boluvchi==0 && boluvchi*boluvchi==i)
          sum=sum+boluvchi;
     boluvchi++;
      }
      if(sum+1==i)
         cout<<i<<" mukammal son."<<endl;
      boluvchi=2;  sum=0;
    }
}
```

\\ Yevklid algoritmi

```
#include<iostream>
using namespace std;
int main ()
{
 int a, b; cin>>a>>b;
 while(a != b)
   {
      if(a > b)
         a -= b;
      else
         b -= a;
   }
   cout << a;
}
```

\\ Qisqartirilmaydigan kasr

```
#include<iostream>
using namespace std;
int main ()
{
   int p, m, a, b, i, j, n;
   cin>>p;
   m=0, n=1;
```

```
   do
   {
      cout<<m<<"/"<<n<<endl;
      i=1; j=1;
      for (b=2; b<p; b++)
      {
         a=m*b/n+1;
         if (a*j<b*i)
         {
            i=a;
            j=b;
         }
      }
      m=i;
      n=j;
   }
   while (i<j);
}
```

\\ **Daraja**

```
#include<iostream>
#define LL long long
using namespace std;
LL binar (LL a, LL b)
{
   LL res = 1;
   while (b)
   {
      if (b & 1)
         res = res*a;
      b=b>>1;
      a*=a;
   }
   return res;
}
int main ()
{
   LL  a, b;
       cin>>a>>b;
```

```
   cout<<binar(a, b);
}

// Katta sonlar bilan ishlash algoritmlari
// Ikki son yig'indisi
#include<bits/stdc++.h>
using namespace std;
int main ()
{
   string a, b;
   cin>>a>>b;
   int n, m;
   n=a.size();
   m=b.size();
   if (n>m)
   {
      for (int i=m; i<n; i++)
         b='0'+b;
   }
   else if (m>n)
   {
      for (int i=n; i<m; i++)
         a='0'+a;
   }
   int x=0;
   string c="";   int k;
   k=max(n,m);    int q;
   for (int i=k-1; i>=0; i--)
   {
      x+=(a[i]-48)+(b[i]-48);
      if (x>9)
      {
        q=x%10; x=1;
        c=char(q+48)+c;
      }
      else
      {
         c=char(x+48)+c;
         x=0;
```

```
      }
    }
    if (x==1) c='1'+c;
    cout<<c;
}

// Katta sonlarning ko'paytmasi
#include<bits/stdc++.h>
using namespace std;
int c[10000];
int main ()
{
 int i, j;
 string a, b; cin>>a>>b;
 int butun=0, temp, qoldiq;
 if (a[0]==48 || b[0]==48)
 {
     cout<<0;
  return 0;
 }
 for (i=a.size()-1; i>=0; i--)
 {
  butun=0;
  for (j=b.size()-1; j>=0; j--)
  {
   temp=(a[i]-48)*(b[j]-48)+c[1+i+j]+butun;
   butun=temp/10;
   c[1+i+j]=temp%10;
  }
  c[i]=butun;
 }

 if (c[0]==0)
 {
  for (int i=1; i<a.size()+b.size(); i++)
    cout<<c[i];
 }
 else
  {
```

```
   for (int i=0; i<a.size()+b.size(); i++)
    cout<<c[i];
   }
}

// Ikki son ayirmasi (umumiy holdagi yechimi)
#include<bits/stdc++.h>
using namespace std;
bool done;
bool smaller(string a, string b)
{
   int n1 = a.length(), n2=b.length();
   if (n1<n2)
    return true;
   if (n2<n1)
    return false;
   for (int i=0; i<n1; i++)
   {
    if (a[i]<b[i])
       return true;
    else if (a[i]>b[i])
       return false;
  }
   return false;
}
string subs(string a, string b)
{
   if (smaller(a, b))
   {
      swap(a, b);
      done=true;
   }
   string c="";
   int m=a.length(), n=b.length();
   if (m>n)
   {
      for (int i=n; i<m; i++)
         b='0'+b;
   }
```

```
    else if (n>m)
    {
       for (int i=m; i<n; i++)
          a='0'+a;
    }
    int x = 0;
   int k=max(m, n);
    for (int i=k-1; i>=0; i--)
    {
       x+=a[i]-b[i];
       if (x>=0)
       {
          c=char(x+48)+c;
          x=0;
       }
       else
       {
          c=char(x+58)+c;
          x=-1;
       }
    }
    int sanoq=0;
    for (int i=0; i<k; i++)
    {
       if (c[i]=='0')
         sanoq++;
       else
      break;
    }
    if (sanoq>0 && sanoq!=k)
       c.erase(0, sanoq);
    else if (sanoq==k)
       c.erase(0, sanoq-1);
    if (done)
      c="-"+c;
    return c;
}
int main()
{
```

```
  string a, b;
  cin>>a>>b;
  cout<<subs(a. b);
}

// Uzun ketma-ketlik
#include<bits/stdc++.h>
using namespace std;
int main ()
{
 ifstream in_file;
 ofstream out_file;
 in_file.open("read.txt");
 out_file.open("write.txt");
 int time=0, maxi=0;
  int n, m;
  in_file>>n;
  while (n)
  {
   in_file>>m;
   if (m==0)
      time++;
   else
    time=0;
   if (maxi<time)
      maxi=time;
   n--;
  }
  out_file<<maxi;
}

// Binar izlash
#include<bits/stdc++.h>
using namespace std;
int binary(int ar[], int n, int sample)
{
 int left=0, right=n-1;
 while (left<right)
 {
```

```
   int middle=(left+right)/2;
   if (ar[middle]==sample)
     return middle;
   else if (ar[middle]<sample)
     left=middle;
   else
     right=middle;
  }
  return -1;
}
int main ()
{
  int n, sample; cin>>n>>sample;
  int ar[n+1];
  for (int i=0; i<n; i++)
    cin>>ar[i];
  if (binary(ar, n, sample)==-1)
    cout<<"massivda bunday element uchramaydi";
  else
    cout<<sample<<" element "<<binary(ar, n, sample)<<" indeksda
uchraydi";
}
```

// Orasiga qo'yish

```
#include<bits/stdc++.h>
using namespace std;
int main ()
{
  ifstream in_file;
  ofstream out_file;
   in_file.open("read.txt");
   out_file.open("write.txt");
   int n, k, element, i=1;
  in_file>>n>>k>>element;
  int massiv[n];
while(!in_file.eof())
{
       in_file>>massiv[i];
       i++;
```

```
}
    for (int i=n; i>=k+1; i--)
    massiv[i]=massiv[i-1];
 massiv[k]=element;
    for (int i=1; i<=n; i++)
    out_file<<massiv[i]<<" ";
}
```

// Nollar

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
int main ()
{
    ll n; cin>>n;
    ll ar[n+1];
    for (int i=1; i<=n; i++)
        cin>>ar[i];
    for (int i=1; i<=n; i++)
    {
        if (ar[i]==0)
        {
            for (int j=i+1; j<=n; j++)
            {
                if (ar[j]!=0)
                {
                    swap(ar[i], ar[j]);
                    break;
                }
            }
        }
    }
    for (int i=1; i<=n; i++)
        cout<<ar[i]<<" ";
}
```

// Chegara

```
#include<bits/stdc++.h>
```

```
#define ll long long
using namespace std;
int main ()
{
	ll n, b; cin>>n;
	ll ar[n+1];
	for (int i=1; i<=n; i++)
		cin>>ar[i];
	cin>>b;
	ll left=1, right=n;
	while (left<right)
	{
		if (ar[left]<=b)
			left++;
		else
		{
			if (ar[right]>=b)
				right--;
			else
				swap(ar[left], ar[right]), left++, right--;
		}
	}
	for (int i=1; i<=n; i++)
		cout<<ar[i]<<" ";
}
```

// TeskarisiTeng

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
int main ()
{
	ll n; cin>>n;
	ll ar[n+1];
	for (int i=1; i<=n; i++)
		cin>>ar[i];
	ll LN=1, PN=2, max=1;
	ll total=0;
	if (n>1)
```

```
        total=2*(n-2)+1;
    else
    {
        cout<<1;
        return 0;
    }
    while (total)
    {
        ll l=LN, p=PN;
        while (l>=1 && p<=n && ar[l]==ar[p])
        {
            l--;
            p++;
        }
        ll m=p-l-1;
        if (m>max)
            max=m;
        if (total%2==1)
            PN++;
        else
            LN++;
        total--;
    }
    cout<<max;
}
```

// Qism kvadrat massiv

```
#include<bits/stdc++.h>
using namespace std;
int main ()
{
   int n, m, maxi=0; cin>>n>>m;
   int ar[n+1][m+1];
   for (int i=1; i<=n; i++)
   {
      for (int j=1; j<=m; j++)
          cin>>ar[i][j];
      }
      for (int i=1; i<=n; i++)
```

```
    {
        for (int j=1; j<=m; j++)
        {
            if (ar[i][j]==1)
            {
                ar[i][j]=min(min(ar[i-1][j-1], ar[i-1][j]), ar[i][j-
1])+1;
                maxi=max(ar[i][j], maxi);
            }
        }
    }
    cout<<maxi;
}
```

// 4 ta element yig'indisi

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
int main ()
{
    ll n, res=0; cin>>n;
    ll ar[n+1][n+1];
    for (int i=1; i<=n; i++)
    {
        for (int j=1; j<=n; j++)
            cin>>ar[i][j];
    }
    for (int i=1; i<=n-1; i++)
    {
        for (int j=1; j<=n-1; j++)
        {
            if (ar[i][j]+ar[i+1][j]+ar[i][j+1]+ar[i+1][j+1]==17)
                res++;
        }
    }
    cout<<res;
}
```

// Ikki marotaba tartiblangan

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
int main ()
{
	ll n, m, k, cnt=0; cin>>n>>m>>k;
	ll ar[n+1][m+1];
	for (int i=1; i<=n; i++)
	{
		for (int j=1; j<=m; j++)
			cin>>ar[i][j];
	}
	for (int i=1; i<=n;)
	{
		for (int j=m; j>=1;)
		{
			if (ar[i][j]==k)
			{
				cnt++;
				cout<<i<<" "<<j<<endl;
				return 0;
			}
			if (ar[i][j]>k)
				j--;
			else if (ar[i][j]<k)
				i++;
		}
	}
	if (cnt==0)
		cout<<"BUNDAY ELEMENT YO'Q";
}
```

// Qism to'g'ri to'rtbukchakli massiv

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
int main ()
{
```

```
ll n, m, temp; cin>>n>>m;
ll a[n+1][m+1], iMax, jMax;
ll p[m+1], L[m+1], R[m+1];
for (int i=1; i<=n; i++)
{
      for (int j=1; j<=m; j++)
            cin>>a[i][j];
}
ll Smax=-1;
for (int i=1; i<=n; i++)
{
      for (int j=1; j<=m; j++)
      {
            if (i==1)
            {
                  p[j]=a[1][j];
            }
            else
            {
                  if (a[i][j]==0)
                  {
                        p[j]=0;
                  }
                  else
                  {
                        p[j]=p[j]+1;
                  }
            }
      }
      for (int j=1; j<=m; j++)
      {
            temp=p[j];
            L[j]=1;
            for (int left=j; left>=1; left--)
            {
                  if (p[left]<temp)
                  {
                        L[j]=left+1;
                        break;
```

```
                }
            }
            R[j]=m;
            for (int right=j; right<=m; right++)
            {
                if (p[right]<temp)
                {
                    R[j]=right-1;
                    break;
                }
            }
        }
        for (int j=1; j<=m; j++)
        {
            temp=p[j]*(R[j]-L[j]+1);
            if (Smax<temp)
            {
                Smax=temp;
                iMax=i;
                jMax=j;
            }
        }
    }
    cout<<Smax<<" "<<iMax<<" "<<jMax;
}
```

// Otning yurishi

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
struct cell {
    ll x, y;
    ll dis;
    cell() {}
    cell(ll x, ll y, ll dis)
        : x(x), y(y), dis(dis)
    {
    }
};
```

```
bool isInside(ll x, ll y, ll n, ll m)
{
    if (x >= 1 && x <= n && y >= 1 && y <= m)
        return true;
    return false;
}

ll minStepToReachTarget(ll knightPos[], ll targetPos[], ll n, ll m)
{
    ll dx[] = { -2, -1, 1, 2, -2, -1, 1, 2 };
    ll dy[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
    queue<cell> q;
    q.push(cell(knightPos[0], knightPos[1], 0));
    cell t;
    ll x, y;
    bool visit[n + 1][m + 1];

    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            visit[i][j] = false;
    visit[knightPos[0]][knightPos[1]] = true;
    while (!q.empty()) {
        t = q.front();
        q.pop();
        if (t.x == targetPos[0] && t.y == targetPos[1])
            return t.dis;
        for (int i = 0; i < 8; i++) {
            x = t.x + dx[i];
            y = t.y + dy[i];

            if (isInside(x, y, n, m) && !visit[x][y]) {
                visit[x][y] = true;
                q.push(cell(x, y, t.dis + 1));
            }
        }
    }
}
```

```
int main()
{
    ll n, m; cin>>n>>m;
    ll x1, y1, x2, y2;
    cin>>x1>>y1;
    cin>>x2>>y2;
    ll knightPos []= {x1, y1};
    ll targetPos []= {x2, y2};
    cout << minStepToReachTarget(knightPos, targetPos, n, m);
    return 0;
}
```

// Tanlash usuli

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
int main ()
{
    ll n; cin>>n;
    ll a[n+1];
    for (int i=1; i<=n; i++)
        cin>>a[i];
    for (int i=1; i<=n-1; i++)
    {
        ll minIndex=i;
        for (int j=i+1; j<=n; j++)
        {
            if (a[j]<a[minIndex])
                swap(a[j], a[minIndex]);
        }
    }
    for (int i=1; i<=n; i++)
        cout<<a[i]<<" ";
}
```

// Pufakcha usuli

```
#include<bits/stdc++.h>
```

```
#define ll long long
using namespace std;
int main ()
{
	ll n; cin>>n;
	ll a[n+1];

	for (int i=1; i<=n; i++)
		cin>>a[i];
	for (int i=1; i<n; i++)
	{
		for (int j=1; j<=n-i; j++)
		{
			if (a[j]>a[j+1])
				swap(a[j+1], a[j]);
		}
	}
	for (int i=1; i<=n; i++)
		cout<<a[i]<<" ";
}
```

// Sheyker usuli

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
void sheyker(ll a[], ll m)
{
  for(int i=0; i<m;)
	{
	for(int j=i+1; j<m; j++)
		{
	  if(a[j]<a[j-1])
		swap(a[j], a[j-1]);
	}
	m--;
	for(int k=m-1; k>i; k--)
		{
	  if(a[k]<a[k-1])
		swap(a[k], a[k-1]);
```

```
        }
        i++;
    }
}
int main ()
{
        srand(time(NULL));
        ll n; cin>>n;
    ll a[n];
        for(int i=0; i<n; i++)
                cin>>a[i];
    sheyker(a, n);
    for (int i=0; i<n; i++)
      cout<<a[i]<<" ";
}

// Quick_Sort
#include<bits/stdc++.h>
#define ll long long
using namespace std;
ll PartitionIndex(ll a[], ll start, ll end)
{
        ll pivot=a[end];
        ll pIndex=start;
        for (int i=start; i<end; i++)
        {
                if (a[i]<=pivot)
                {
                        swap(a[i], a[pIndex]);
                        pIndex++;
                }
        }
        swap(a[end], a[pIndex]);
        return pIndex;
}

void QuickSort(ll a[], ll start, ll end)
{
        if (start<end)
```

```
    {
        ll pIndex=PartitionIndex(a, start, end);

        QuickSort(a, start, pIndex-1);
        QuickSort(a, pIndex+1, end);
    }
}

int main ()
{
    srand(time(NULL));
    ll n; cin>>n;
    ll a[n+1];
    for (int i=1; i<=n; i++)
        cin>>a[i];
    QuickSort(a, 1, n);
    for (int i=1; i<=n; i++)
        cout<<a[i]<<" ";
}
```

// PlusSort

```
#include <bits/stdc++.h>
#include<bits/stdc++.h>
#define ll long long
using namespace std;
void Merge(ll a[], ll l, ll m, ll r)
{
    ll i, j, k;
  ll n1 = m - l + 2;
  ll n2 = r - m + 1;
  int L[n1], R[n2];
  for (i = 1; i <= n1-1; i++)
    L[i] = a[l + i - 1];
  for (j = 1; j <= n2-1; j++)
    R[j] = a[m + j];
  i = 1;  j = 1; k = l;
  while (i <= n1-1 && j <= n2-1) {
    if (L[i] <= R[j]) {
      a[k] = L[i];
```

```
        i++;
      }
      else {
        a[k] = R[j];
        j++;
      }
      k++;
    }
    while (i <= n1-1) {
      a[k] = L[i];
      i++;
      k++;
    }
    while (j <=n2-1) {
      a[k] = R[j];
      j++;
      k++;
    }
}
void MergeSort(ll a[], ll l, ll r)
{
      if (l < r) {
      int m = l + (r - l) / 2;
      MergeSort(a, l, m);
      MergeSort(a, m + 1, r);
      Merge(a, l, m, r);
    }
}
int main ()
{
      ll n; cin>>n;
      ll a[n+1];
      for (int i=1; i<=n; i++)
            cin>>a[i];
      MergeSort(a, 1, n);
      for (int i=1; i<=n; i++)
            cout<<a[i]<<" ";
}
```

```
// ShellSort
#include<iostream>
#define ll long long
using namespace std;
void ShellSort(ll a[], ll n)
{
	bool k;
	ll d=n/2;
	while (d)
	{
		k=true;
		while (k)
		{
			k=false;
			for (int i=1; i<=n-d; i++)
			{
				if (a[i]>a[i+d])
				{
					k=true;
					swap(a[i], a[i+d]);
				}
			}
		}
		d/=2;
	}
}
int main ()
{
	ll n; cin>>n;
	ll a[n+1];
	for (int i=1; i<=n; i++)
		cin>>a[i];
	ShellSort(a, n);
	for (int i=1; i<=n; i++)
		cout<<a[i]<<" ";
}

// Piramida usuli
#include<bits/stdc++.h>
```

```
#define ll long long
using namespace std;
void heapify(ll a[], ll n, ll i)
{
	ll largest=i;
	ll l=2*i+1;
	ll r=2*i+2;
	if (l<n && a[l]>a[largest])
		largest=l;
	if (r<n && a[r]>a[largest])
		largest=r;
	if (largest!=i)
	{
		swap(a[i], a[largest]);
		heapify(a, n, largest);
	}
}
void heapSort(ll a[], ll n)
{
	for (int i=n/2-1; i>=0; i--)
		heapify(a, n, i);
	for (int i=n-1; i>0; i--)
	{
		swap(a[i], a[0]);
		heapify(a, i, 0);
	}
}
int main ()
{
	ll n; cin>>n;
	ll a[n];
	for (int i=0; i<n; i++)
		a[i]=rand()%100;
	heapSort(a, n);
	for (int i=0; i<n; i++)
		cout<<a[i]<<" ";
}
```

```
// Xanoy minorasi
#include<bits/stdc++.h>
#define ll long long
using namespace std;
void Hanoi(ll n, char from, char to, char aux)
{
	if (n==1)
	{
		cout<<"1 "<<from<<"-> "<<to<<endl;
		return;
	}
	Hanoi(n-1, from, aux, to);
	cout<<n<<" "<<from<<"-> "<<to<<endl;
	Hanoi(n-1, aux, to, from);
}
int main ()
{
	ll n; cin>>n;
	Hanoi(n, '1', '2', '3');
}
```

Foydalanilgan adabiyotlar ro'yxati

1. Akbarov D.Ye. Axborot xavfsizligini ta'minlashning kriptografik usullari va ularning qo'llanilishi.- Toshkent, «O'zbekiston markasi» nashriyoti, 2009-432 bet.
2. Aripov A.N., Mirzaxidov X.M., Shermatov Sh.X., Saidxodjayev S.R., Hasanov P.F., Amirov D.M., Bakirov O.A. Axborot – kommunikatsiya texnologiyalari. Izohli lug'at. Toshkent-2004.-499 bet.
3. Aripov M.M., Irmuhamedova R.M., Sagatov M.V., Haydarov A.X., Yakubov A.X., Imamov T. Informatika axborot texnologiyalari. O'quv qo'llanmasi. 1-qism. Toshkent, "Universitet", 2007.-264 bet
4. Axborot texnologiyasi operatsion tizimlari terminlarining inglizcha-ruscha-o'zbekcha izohli lug'ati// Yo.Amedova, R. Sayfulin, A.Mirazizov, A.Obidov, A.Gafurov// M.Muxitdinovning umumiy tahriri ostida. O'zbekiston Respublikasi Fanlar akademiyasi-«Fan» nashriyoti, 2009.-495 b.
5. Axmedov A.B., Taylaqov N.I. Informatika. Akademik litsey va kasb-hunar kollejlari uchun o'quv qo'llanma.- T.: "O'zbekiston", 2002-272 bet.
6. Azamatov A.R. Algoritmlash va dasturlash asoslari: kasb-hunar kollejlari uchun o'quv qo'llanma. T.: Cho'lpon nomidagi nashriyot-matbaa ijodiy uyi, 2010. - 232 b.
7. Buyuk ajdodlarimiz / nashrga tayyorlovchi va mas'ul muharrirlar: M.Aminov, F.Hasanov. – T.: «O'zbekiston milliy ensiklopediyasi» Davlat ilmiy nashriyoti, 2010. – 208 b.
8. G'ulomov S.S., Zaynalov N.R., Begalov B.A., Dadabayeva R.A., Davronov A.E. Dasturlash texnologiyalari (Oliy o'quv yurtlari uchun o'quv qo'llanma) Toshkent. TDIU, 2006. –191 b.
9. Isroilov I., Pashayev Z. Geometriya. Akademik litseylar uchun darslik . 2-nashri. – Toshkent : "O'qituvchi" NMIU, 2010. I qism. – 224 b.
10. Kurosh A.G. Oliy algebra kursi. Ruschadan tarjima. – T.: "O'qituvchi", 1976. – 461 b.
11. Narzullayev U.X., Qarshiyev A.B., Boynazarov I.M. Ma'lumotlar tuzilmasi va algoritmlar. O'quv qo'llanma. –Toshkent, 2013. -192 b.
12. Manturov O.V., Solnsev Yu.K., Sorkin Yu.I., Fedin N.G. Matematika termin-larining ruscha-o'zbekcha izohli lug'ati.Tarjima. /Prof. V.A.Ditkin tahriri ostida. O'qituvchi, T.-1974-550.

13. Nazarov R.N. va boshq. Algebra va sonlar nazariyasi: Ped.in-t va un-t fiz.-mat. fak. talabalari uchun o'quv qo'llanma/ R.N.Nazarov, B.T.Toshpo'latov, A.D.Do'sumbetov, 2 qismli. K.I. – T.: O'qituvchi, 1993 – 320 b.
14. Ne`matov A., Kulmuradov M., Tangirov A., Akbarova N. Algoritmik tillarda dasturlashdan misol va masalalar. Amaliy va mustaqil mashg'ulotlar uchun electron qo'llanma. Toshkent-2010.-92 bet.
15. O'zbek tilining izohli lug'ati: 80000 dan ortiq so'z va so'z birikmasi. J. III. N-Tartibli / Tahrir hay'ati: T.Mirzayev (rahbar) va boshq.: O'zR FA Til va adabiyot in-ti.- T.: "O'zbekiston milliy ensiklopediyasi" Davlat ilmiy nashriyoti, 2006.-688 b.
16. Qarshiyev A.B., Yusupov R.A. *C++ Builder*da masalalar va topshiriqlar. Uslubiy qo'llanma. – Samarqand, TATU Samarqand filiali nashri, 2017. -146 b.
17. Qosimjonov R. Shaxmat – sehrli olam. -T.: «infoCOM.UZ» nashriyoti, 2008. - 213 b.
18. Qosimov S.S. Axborot texnologiyalari. O'quv qo'llanma. Toshkent- "Aloqachi"-2006.-370b.
19. Raisov M.R., Xo'jayorov B.X., Toshov N.T. Matematik programmalash. Ma'ruzalar matni. Samarqand, SamISI, 2006. –170 b.
20. Rajabov F., Masharipov S., Madrahimov R. Oliy matematika. O'quv qo'llanma. Toshkent- "Turon-Iqbol"-2007.-400b.
21. Sattarov 3. A., Xujayeva G.. "EHMda programmalashni bilasizmi?", T.-1989.
22. To'rayev H.T., Azizov I., Otaqulov S. Kombinatorika va graflar nazariyasi. T., ILM ZIYO, 2009
23. Xaldjigitov A.A., Madraximov Sh.F., Ikromov A.M., Rasulov S.I. Pascal tilida programmalash bo'yicha masalalar to'plami. O'quv qo'llanma. O'zMU. Toshkent, 2005.-94 bet.
24. Zaynalov N.R. Pascal tilida murakkab masalalarni dasturlash. Uslubiy ko'rsatma. SamISI, Samarqand,2007. –24 b.
25. Zaynalov N.R., Begalov B.A., Davronov A.E. Visual Basicda dasturlash texno-logiyasi. Uslubiy ko'rsatma. SamISI, Samarqand,2006. –244 b.
26. Zaynalov N.R., Davronov A.E., Temirov O.G'.Visual Basicda ishlash asoslari. Uslubiy ko'rsatma.SamKI, Samarqand,2001. -32 bet.

27. Zaynalov N.R., Sarimsoqov U., Eshmurodov K., Salomova M.A. Izlash, tanlash va saralash misol va masalalari. Uslubiy ko'rsatma. SamDAQI. Samarqand, 1997-38 bet
28. Zaynalov N.R., Suvonkulov A.M., Qurbonov M.M. Katta sonlar bilan amal-larni bajarish algoritmlari. Uslubiy ko'rsatma. SamISI, Samarqand,2011. –56 b.
29. Zaynalov N.R., Suvonkulov A.M., Qurbonov M.M. MS Excel muhitida Visual Basic imkoniyatlari. Uslubiy ko'rsatma. SamISI, Samarqand,2011. –50 b.
30. Босова Л.Л., Босова А.Ю., Коломенская Ю.Г. Занимательные задачи по информатике. 3-изд. – М.: БИНОМ. Лаборатория знаний, 2007. – 119 с.
31. Долинский М.С. Решение сложных и олимпиадных задач по программированию: Учебное пособие. – СПб.: Питер, 2006. – 366 с.
32. Зайналов Н.Р., Давронов А.Э. Преподавание информатики в высшей школе: проблемы и решения/ InfoCOM.UZ ,2004, № 5, С.66-67.
33. Зубов В.С. Справочник программиста. Базовые методы решения графовых задач и сортировки. – М.: Информационно-издательский Дом «Филинъ», 1999. – 256 с.
34. Кнут Д. Искусство программирования для ЭВМ. – Т.1. Основные алгоритмы. – М.: Мир, 1976. 735 с.
35. Кнут Д. Искусство программирования для ЭВМ. – Т.3. Сортировка и поиск. – М.: Мир, 1978. 844 с.
36. Красиков И.В., Красикова И.Е. Алгоритмы. Просто как дважды два. – М.: Эксмо, 2007. – 256 с.
37. Меньшиков Ф.В. Олимпиадные задачи по программированию. – СПб.: Питер, 2006. – 315 с.
38. Макконнелл Дж. Основы современных алгоритмов. 2-е доп. изд. М.: Техносфера, 2004. – 368 с.
39. Мозговой М.В. Занимательное программирование: Самоучитель. – СПб.: Питер, 2005. – 208 с.
40. Нарзуллаев У. Алгебра и теория чисел. Сборник задач и упражнений: Часть1.–LAP LAMBERT Academic Publishing, GmbH & Co.KG.: Saarbrucken, 2012. – 217 с.
41. Окулов С.М. Программирование в алгоритмах. – М.: БИНОМ. Лаборатория знаний, 2002. – 341 с.

42. Першиков В.И., Марков А.С., Савинков В.М. Русско-английский толковый словарь по информатике. 3-е перераб. изд. – М.: Финансы и статистика, 1999. – 368 с.
43. Порублев И.Н., Ставровский А.Б. Алгоритмы и программы. Решение олимпиадных задач. – М.: ООО "И.Д.Вильямс", 2007. – 480 с.
44. Скиена С.С., Ревилла М.А. Олимпиадные задачи по программированию. Руководство по подготовке к соревнованиям/ Пер. с англ. – М.: КУДИЦ-ОБРАЗ, 2005. – 416 с.
45. Холматов Т.Х., Зайналов Н.Р., Хайдаров Р.Н. Программирование алгорит-мов с использованием подпрограмм на языке Pascal. Методическое указание. СамГАСИ, Самарканд,2003.
46. Шень А.Программирование: теоремы и задачи. 2-е изд., испр. и доп. – М.: МЦНМО, 2004. – 296 с.
47. Шень А. Программирование: теоремы и задачи. 6-е изд., дополненное. М.: МЦНМО, 2017. – 320 с.
48. Кормен Т.Х. Алгоритмы:вводный курс.: Пер. с англ. – М.: ООО "И.Д. Вильямс", 2014. – 208 с.
49. Левитин А.. Левитина М. Алгоритмические головоломки. Пер. с англ. Ж.А.Меркуловой, Н.А.Меркулова.- М.: Лаборатория знаний, 2018. - 325 с.
50. А.Лааксонен. Олимпиадное программирование. / пер. с англ. А.А.Слинкин. – М.: ДМК Пресс, 2018. – 300 с.
51. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, 3-е изд. : Пер. с англ. – М.: ООО "И.Д. Вильямс", 2013. – 1328 с.
52. Рафгарден Тим. Совершенный алгоритм. Основы. – СПб.: Питер, 2019. – 256 с.
53. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. – СПб.: Питер, 2017. – 288 с.
54. Юркин А.Г. Задачник по программированию. – СПб.: Питер, 2002. – 192 с.
55. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to algorithms. 3rd ed. Cambridge.: The MIT Press, 2009.-1290 p.
56. Dickson E.A. Computer program design. – N.-Y.: Irwin/McGraw-Hill, 1996.-424 p.

57. Neapolitan R., Naimipour K. Foundations of Algorithms Using C++ Pseudocode, Third Edition. N.-Y.: Jones and Bartlett Publishers, 2004.- 618 p.
58. http://acm.dvpion.ru/ - Dasturchi maktabi.
59. http://acm.sgu.ru/ - Saratov Davlat Universiteti, dasturlarni tekshiruvchi portal.
60. http://acm.timus.ru/ - Timus Online Judge, dasturlarni tekshiruvchi portal.
61. http://acm.tuit.uz/ - Masalalar yechimi avtomatik tekshiruvchi tizimi
62. http://acmp.ru/ - Oddiy masalalar to'plami
63. http://codeforces.ru/ - Dasturlash bo'yicha on-line musobaqa portali
64. http://diofant.ru / - Dasturlash bo'yicha musobaqa portali.
65. http://dl.gsu.by/ - Boshlang'ich ta'limot uchun dasturlash bo'yicha musobaqa portali.
66. http://ejudge.btty.su/bmstu/2007-2008/docs/ - Dinamik dasturlashga doir ma'ruzalar
67. http://e-maxx.ru/ - Dasturlashga doir algoritmlar portali
68. http://g6prog.narod.ru/din_kotov.rar/- Dinamik dasturlashga doir ma'ruzalar
69. http://informatics.mccme.ru/moodle/course/ - Dinamik dasturlashga doir ma'ruzalar
70. http://intuit.ru/video/tree/catalog/algorithms/ - Algoritmlarga doir video-kurslar
71. http://lektorium.tv/course/ - Algoritmlarga doir video-kurslar
72. http://neerc.ifmo.ru/school / - Informatika bo'yicha olimpiada portali.
73. http://olympiads.ru/mosolymp/- Informatika bo'yicha olimpiada portali.
74. http://programming-challenges.com/ - Dasturlash bo'yicha olimpiada portali.
75. http://snarknews.info/ - Dasturlashga doir olimpiadalar haqidagi portal
76. http://topcoder.com/ - Dasturlash bo'yicha musobaqa portali

Xaldjigitov A.A., Zaynalov N.R.,
Qarshiyev A.B., Yakubdjanova D.K.

DASTURLASHDAN MISOL VA MASALALAR TO'PLAMI

O'quv qo'llanma

Nashr uchun mas'ul: B. Mavlonov

Muharrir: U. Yunusov

Badiiy muharrir: F. Sobirov

Dizayner-sahifalovchi: L. Abdullayev

Nashriyot ro'yxat raqami № 1043191. 24.09.2021-y.
Bichimi 60x84 1/16 Offset qog'ozi.
Times New Roman garniturasi.
Shartli bosma tabog'i 32,75. Nashr hisob tabog'i 13,9.
Adadi 100 nusxada. Buyurtma № 10-12.

1940

100000, Toshkent shahri, Mirzo Ulug'bek tumani,
M.Ismoiliy ko'chasi 1-G uy.

«ZUXRA BARAKA BIZNES» MChJ bosmaxonasida chop etildi.
Toshkent shahri Bunyodkor shoh ko'chasi 27 A–uy.