CS206 Data Structures

Recursion

Sung-eui Yoon (윤성의)
Department of Computer Science
KAIST

http://sglab.kaist.ac.kr/~sungeui

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Class Objectives

- ☐ Understand the concept of recursion
- ☐ Can write a recursion function for addressing problems
  - Base case
  - Recursive calls
- ☐ Get to know that a program can run fast or slow depending on its algorithm

The Recursion Pattern

- **Recursion**: when a method calls itself
- Classic example: the factorial function:
  - n! = 1· 2· 3· ··· · (n-1)· n
- Recursive definition:

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot f(n-1) & \text{else} \end{cases}$$

- As a Java method:

```
// recursive factorial function
public static int  recursiveFactorial(int n) {
    if  (n  ==  0)  return  1;     // basis case
    else return  n  *  recursiveFactorial(n- 1);      // recursive case
}
```

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Content of a Recursive Method

```
// recursive factorial function
public static int  recursiveFactorial(int n) {
    if  (n  ==  0)  return  1;    // basis case
    else return  n  *  recursiveFactorial(n- 1);      // recursive case
}
```

☐ Base case(s)

- Return values w/o recursive calls; there should be at least one base case.

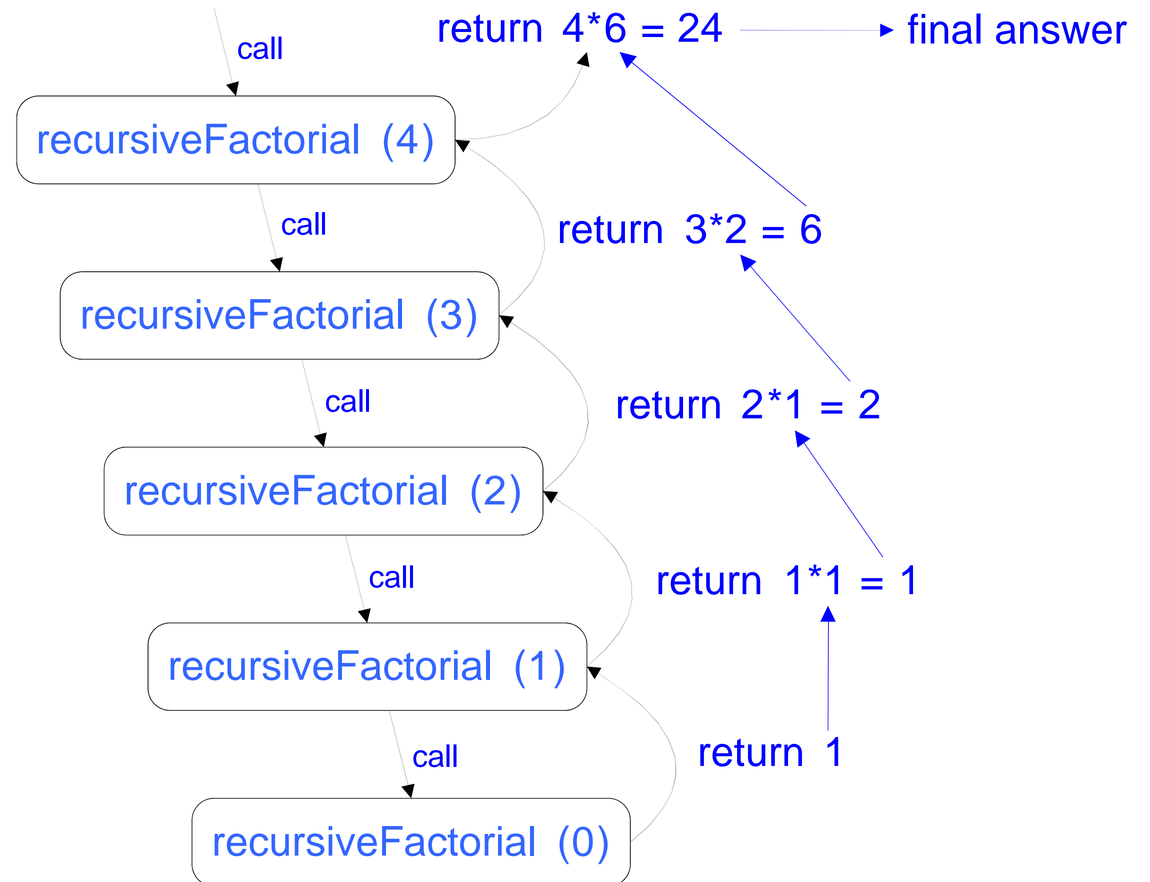
☐ *Recursive calls*

- Calls to the current method that makes progress towards a base case.

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Visualizing Recursion

- Recursion trace
- A box for each recursive call
- An arrow from each caller to callee
- An arrow from each callee to caller showing return value



KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Linear Recursion

- Begin by testing for a set of base cases and recur once

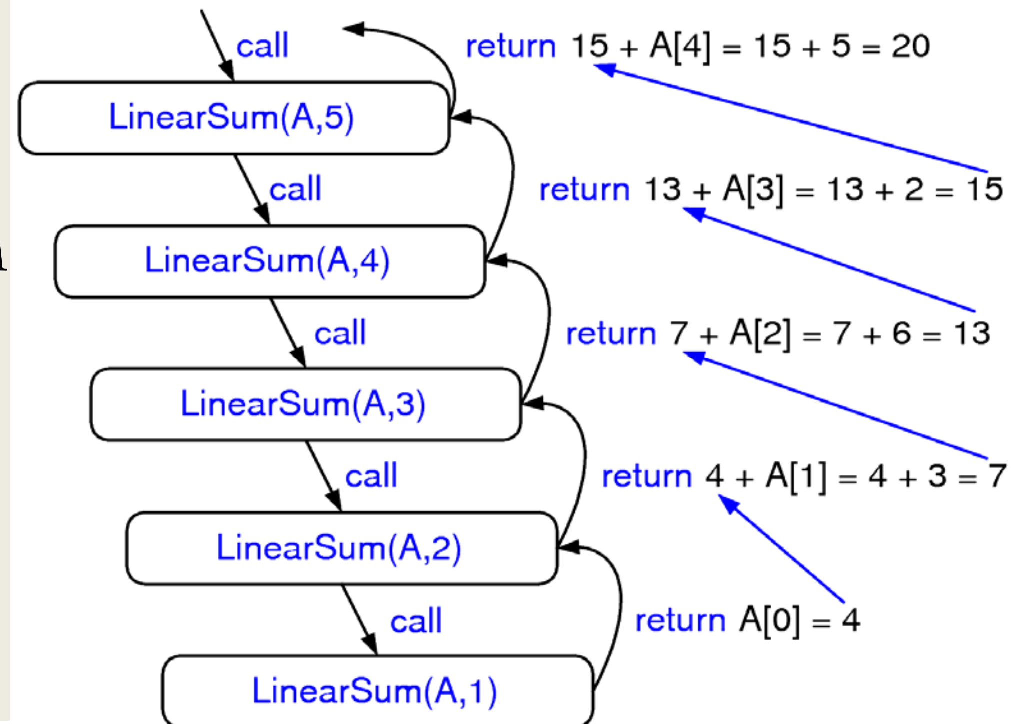
Algorithm LinearSum(A, n)

Input: integer array A and integer $n \geq 1$ s.t. A has at least n elements

Output: Sum of first n integers in A

if $n = 1$ **then**
  return $A[0]$
else
  return LinearSum(A, $n-1$) $+ A[n-1]$
end if

Example recursion trace:



- Iterative approach (using "for") should be easier for this problem
  - But this linear recursion is useful when the first (or last) plus a remaining set that has the same structure

Reversing an Array (HW: Self study)

Algorithm ReverseArray(A, i, j)

Input: array A and non-negative integer indices i and j

Output: Reversal of elements in A starting at index i and ending at j

if $i < j$ **then**

Swap $A[i]$ and $A[j]$

ReverseArray(A, $i + 1$, $j - 1$)

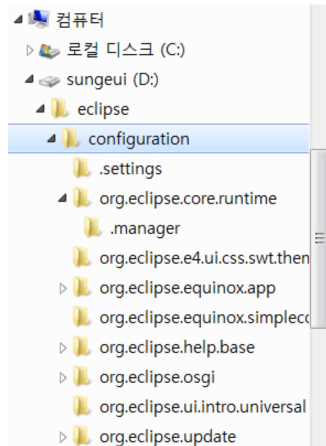
end if

return

Defining Arguments for Recursion

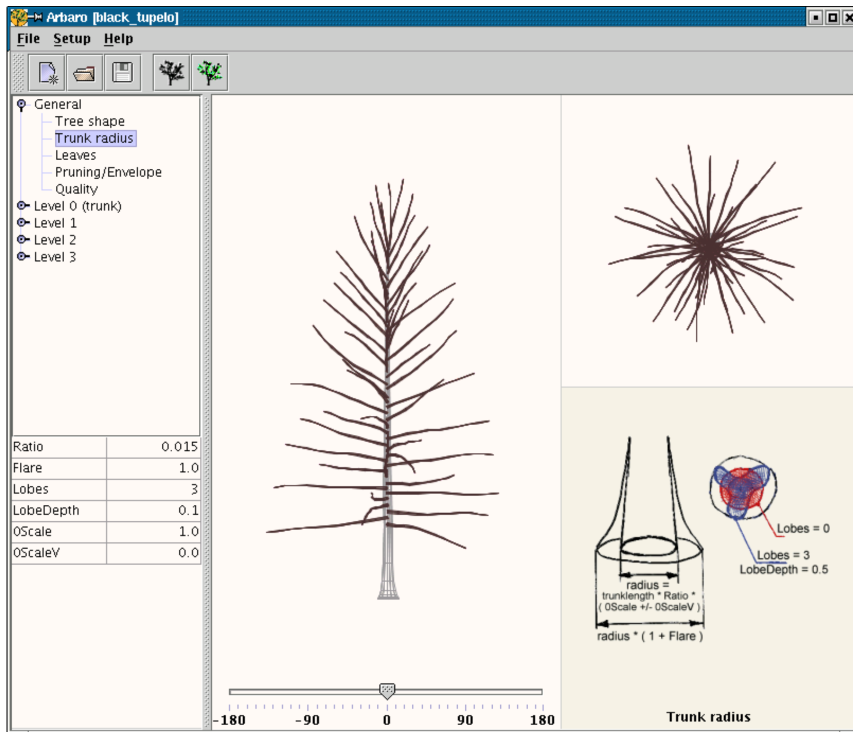
- In creating recursive methods, it is important to define the methods in ways that facilitate recursion
- This sometimes requires to use additional parameters that are passed to the method
  - For example, we defined the array sum method as LinearSum(A, n), not LinearSum (A)

Recursive structures are everywhere!



Google images

- Tree generation tool in computer graphics
  - Generate recursively



Computing Powers

- The power function, p(x,n)=x^n, can be defined recursively:

$$p(x,n) = \begin{cases} 1 & \text{if } n = 0 \\ x \cdot p(x, n-1) & \text{else} \end{cases}$$

- This leads to an power function that invokes n times of multiplication (for we make n recursive calls).
  - = runs linearly as a function of n

- We can do better than this, however.

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Recursive Squaring

- We can derive a more efficient linearly recursive algorithm by using repeated squaring:

$$p(x,n) = \begin{cases} 1 & \text{if } n = 0 \\ p(x, \frac{n}{2})^2 & \text{if } n > 0 \text{ is even} \\ x \cdot p(x, \frac{n-1}{2})^2 & \text{if } n > 0 \text{ is odd} \end{cases}$$

- For example,
  - $2^4 = 2^{(4/2)2} = (2^{4/2})^2 = (2^2)^2 = 4^2 = 16$
  - $2^5 = 2^{1+(4/2)2} = 2(2^{4/2})^2 = 2(2^2)^2 = 2(4^2) = 32$
  - $2^6 = 2^{(6/2)2} = (2^{6/2})^2 = (2^3)^2 = 8^2 = 64$
  - $2^7 = 2^{1+(6/2)2} = 2(2^{6/2})^2 = 2(2^3)^2 = 2(8^2) = 128.$

A Recursive Squaring Method

Algorithm Power(x, n):

Input: A number x and integer $n = 0$

Output: The value x^n

if $n = 0$ **then**

  return 1

if n is odd **then**

  $y = \text{Power}(x, (n - 1)/2)$

  return $x \cdot y \cdot y$

else

  $y = \text{Power}(x, n/2)$

  return $y \cdot y$

Analyzing Recursive Squaring Method

Algorithm Power(x, n):

Input: A number x and integer $n = 0$

Output: The value x^n

if $n = 0$ **then**

  return 1

if n is odd **then**

  $y = \text{Power}(x, (n - 1)/2)$

  return $x \cdot y \cdot y$

else

  $y = \text{Power}(x, n/2)$

  return $y \cdot y$

Each time we make a recursive call we halve the value of n; hence, we make log n recursive calls. That is, this method runs in log n time.

It is important that we used a variable twice here rather than calling the method twice.

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Binary Recursion

- Binary recursion occurs whenever there are **two** recursive calls for each non-base case.
- Define a recursive way to print the ticks and numbers like an English ruler:

```
---- 0
-
--
-
---
-
--
-
---- 1
-
--
-
---
-
--
-
---- 2
```

(a)

2 inch ruler w/ major tick length 4

```
----- 0
-
--
-
---
-
--
-
----
-
--
-
---
-
--
-
----- 1
```

(b)

1 inch ruler w/ major tick length 5

```
--- 0
-
--
-
--- 1
-
--
-
--- 2
-
--
-
--- 3
```

(c)

3 inch ruler w/ major tick length 3

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Recursive Method for Ruler Drawing

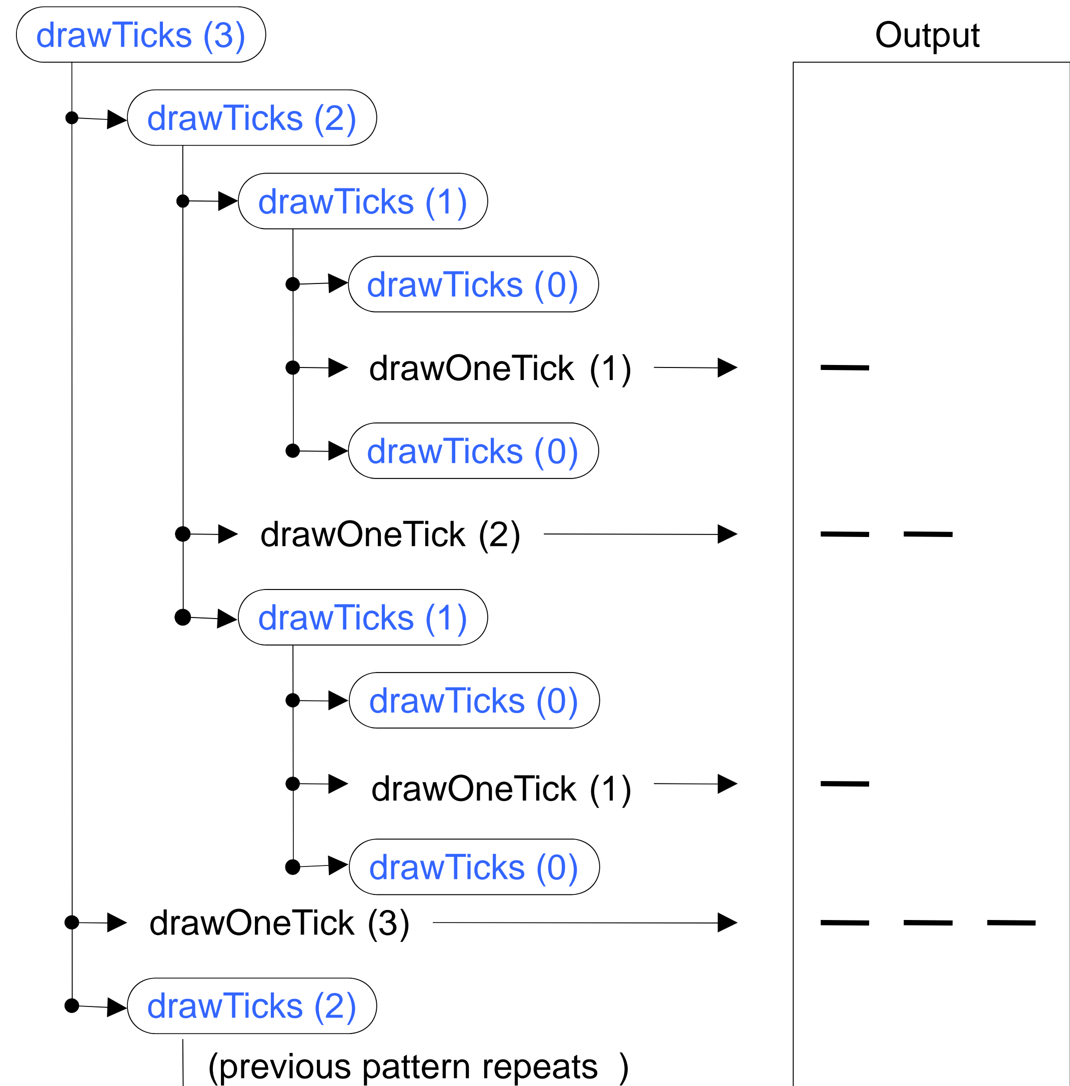
```java
// draw one tick
public static void drawOneTick(int tickLength, int tickLabel)
{
    for (int i = 0; i < tickLength; i++)
        System.out.print("-");
    if (tickLabel >= 0)
        System.out.print(" " + tickLabel);
    System.out.print("\n");
}

public static void drawTicks(int tickLength)
{ // draw ticks of given length
    if (tickLength > 0)
    { // stop when length drops to 0
        drawTicks(tickLength - 1); // recursively draw left ticks
        drawOneTick(tickLength); // draw center tick
        drawTicks(tickLength - 1); // recursively draw right ticks
    }
}

public static void drawRuler(int nInches, int majorLength)
{ // draw ruler
    drawOneTick(majorLength, 0); // draw tick 0 and its label
    for (int i = 1; i <= nInches; i++)
    {
        drawTicks(majorLength - 1); // draw ticks for this inch
        drawOneTick(majorLength, i); // draw tick i and its label
    }
}
```

Visualizing the DrawTicks Method

- An interval with a central tick length $L \geq 1$ is composed of the following:
  - an interval with a central tick length L-1,
  - a single tick of length L,
  - an interval with a central tick length L-1.



KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Another Binary Recusive Method

- Problem: add all the numbers in an integer array A:

Algorithm BinarySum(A, i, n)

Input: array A and non-negative integer indices i and n

Output: sum of n elements in A starting at index i

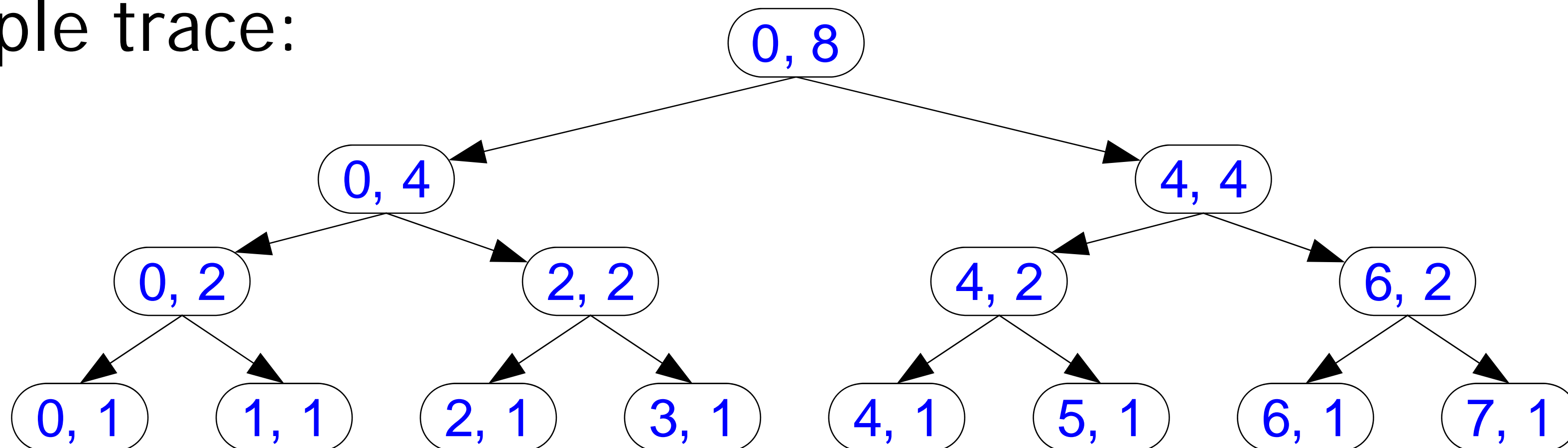
if $n = 1$ **then**

  return $A[i]$

end if

return BinarySum(A, i, $n/2$) $+$ BinarySum(A, $i + n/2$, $n/2$)

- Example trace:



Computing Fibanacci Numbers

- Fibonacci numbers are defined recursively:
  - $F_0 = 0$
  - $F_1 = 1$
  - $F_i = F_{i-1} + F_{i-2}$ for $i > 1$.
- As a recursive algorithm (first attempt):

Algorithm BinaryFib(k)

Input: non-negative integer k
Output: k-th Fibonacci number F_k
if $k \le 1$ **then**
  return k
else
  return BinaryFib($k-1$) + BinaryFib($k-2$)
end if

KAIST Korea Advanced Institute of Science and Technology 한국과학기술원

Analyzing the Binary Recursion Fibonacci Algorithm

- Let n_k denote number of recursive calls made by BinaryFib(k). Then
  - $n_0 = 1$
  - $n_1 = 1$
  - $n_2 = n_1 + n_0 + 1 = 1 + 1 + 1 = 3$
  - $n_3 = n_2 + n_1 + 1 = 3 + 1 + 1 = 5$
  - $n_4 = n_3 + n_2 + 1 = 5 + 3 + 1 = 9$
  - $n_5 = n_4 + n_3 + 1 = 9 + 5 + 1 = 15$
  - $n_6 = n_5 + n_4 + 1 = 15 + 9 + 1 = 25$
  - $n_7 = n_6 + n_5 + 1 = 25 + 15 + 1 = 41$
  - $n_8 = n_7 + n_6 + 1 = 41 + 25 + 1 = 67$.
- Note that the value at least doubles for every other value of n_k. That is, $n_k > 2^{k/2}$. It is exponential!

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

A Better Fibonacci Algorithm

- Use linear recursion instead:

Algorithm LinearFib(k)

Input: non-negative integer k

Output: Pair of Fibonacci numbers (F_k, F_{k-1})

if $k \leq 1$ **then**

return $(k, 0)$

else

$(i, j) = \text{LinearFib}(k-1)$

return $(i+j, i)$

end if

- The number of recursive calls is linear in k
  - = Runs in O(k) time.

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

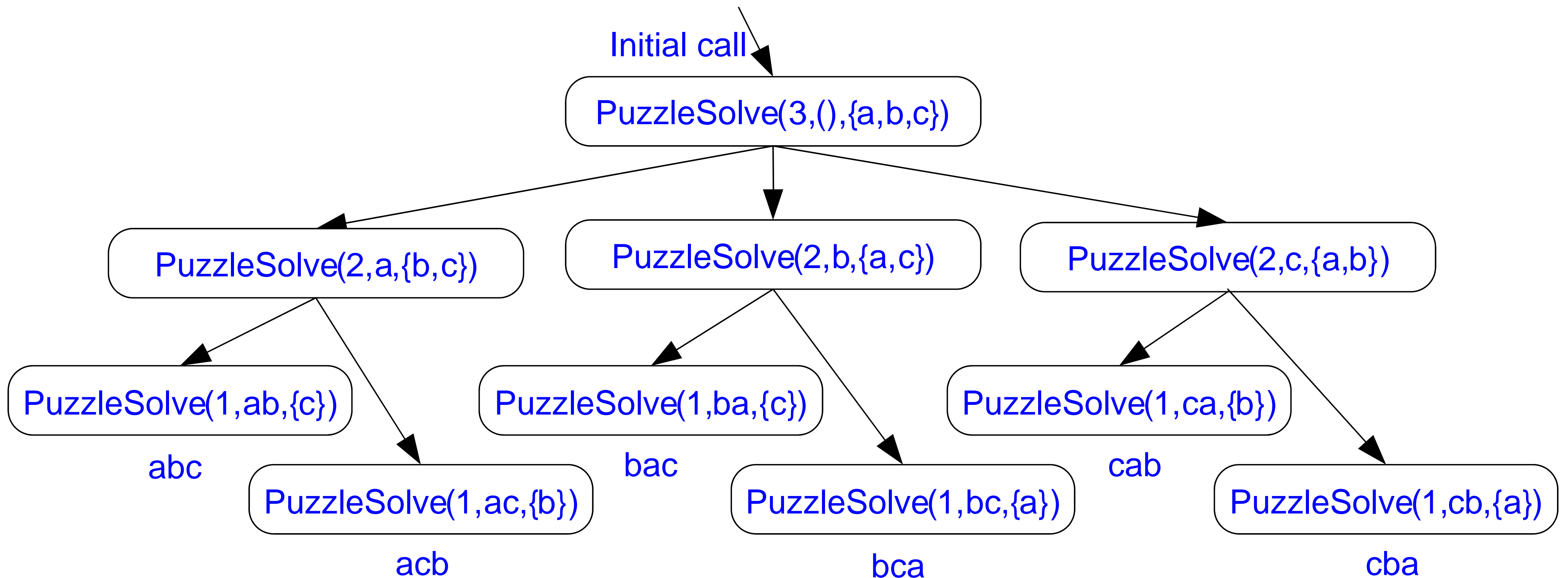
Multiple Recursion

- ☐ Motivating example: summation puzzles
  - Assign a unique digit (0,1,…,9) to each letter so that
    - pot + pan = bib
    - dog + cat = pig
    - boy + girl = baby
- ☐ A simple (but naive) solution: enumerate all possibilities (=permutations) and test each one
  - Multiple recursion: makes potentially many recursive calls (not just one or two)

Algorithm for Multiple Recursion

```
Algorithm PuzzleSolve(k, S, U)
  Input: integer k, sequence S, and set U
  Output: enumeration of all k-length extensions to S
    using elements in U without repetitions
    for each e in U do
      remove e from U // e is now being used
      add e to the end of S
      if k = 1 then
        Test whether S is a configuration that solves the puzzle
        if S solves the puzzle then
          return "Solution found: " S
        end if
      else
        PuzzleSolve(k − 1, S, U)
      end if
      add e back to U // e is now unused
      remove e from the end of S
    end for
```

Visualizing PuzzleSolve

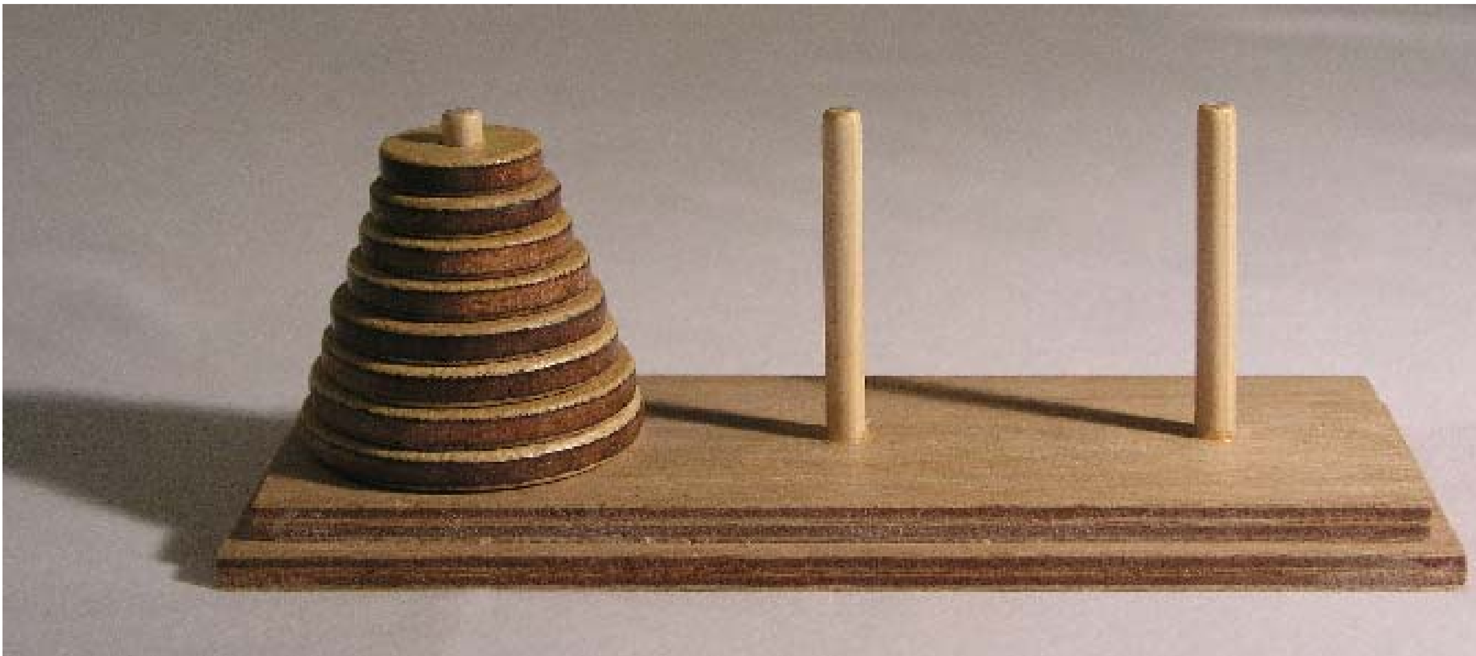


Class Objectives were:

- ☐ Understand the concept of recursion
- ☐ Can write a recursion function for addressing problems
  - Base case
  - Recursive calls
- ☐ Get to know that a program can run fast or slow depending on its algorithm

PA1

- Implement the tower of Hanoi using recursion



Next Time

☐ Analysis tools

☐ HW:

- Go over the next lecture slides before the class
- Just 10 min ~ 20 min should be okay

☐ Any questions:

- Come up with one question on what we have discussed in the class and submit at the end of the class
- 1 for typical questions or 2 for questions with thoughts or that surprised me

- Write questions at least 4 times in this semester
- You can type at KLMS