Recursion (part 3)

October 29, 2007

ComS 207: Programming I (in Java)
Iowa State University, FALL 2007
Instructor: Alexander Stoytchev

Recursive Programming

- Consider the problem of computing the sum of all the numbers between 1 and any positive integer N
- This problem can be recursively defined as:

$$\sum_{i=1}^{N} i = N + \sum_{i=1}^{N-1} i$$

$$= N + N-1 + \sum_{i=1}^{N-2} i$$

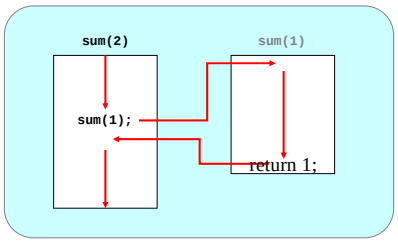
$$= N + N-1 + N-2 + \sum_{i=1}^{N-3} i$$

Recursive Programming

```
// This method returns the sum of 1 to num
public int sum (int num)
{
    int result;

    if (num == 1)
        result = 1;
    else
        result = num + sum (num-1);

    return result;
}
```

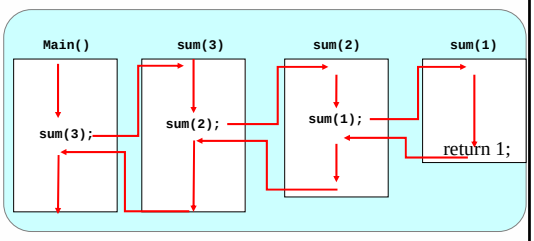

Recursive Control Flow

- In Recursive calls methods can call themselves, but typically with different arguments each time



Recursive Control Flow

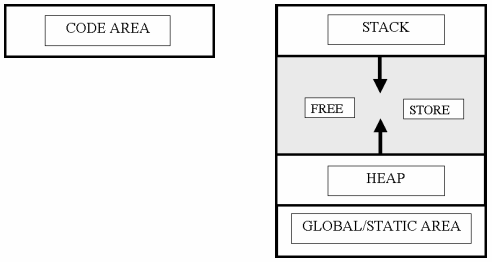
- In Recursive calls methods can call themselves, but typically with different arguments each time



Stack Animation

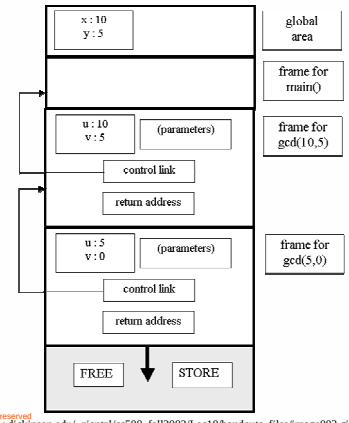
- http://acc6.its.brooklyn.cuny.edu/~cis22/animations/tsang/html/STACK/stack1024.html

Memory Organization



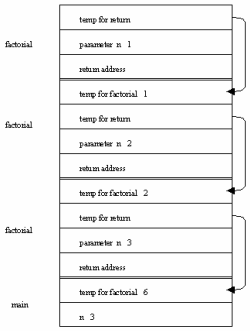
[http://www.dickinson.edu/~ziantzl/cs500_fall2002/Lec10/handouts_files/]

The stack during a recursive call to gcd



[http://www.dickinson.edu/~ziantzl/cs500_fall2002/Lec10/handouts_files/image003.gif]

The stack during a recursive call to factorial

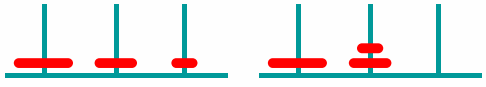


Towers of Hanoi

- The *Towers of Hanoi* is a puzzle made up of three vertical pegs and several disks that slide on the pegs
- The disks are of varying size, initially placed on one peg with the largest disk on the bottom with increasingly smaller ones on top
- The goal is to move all of the disks from one peg to another under the following rules:
  - We can move only one disk at a time
  - We cannot move a larger disk on top of a smaller one

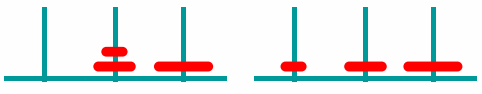
Towers of Hanoi

Original Configuration | Move 1



Move 2 | Move 3

Towers of Hanoi



Move 4 | Move 5

Move 6 | Move 7 (done)

Animation of the Towers of Hanoi

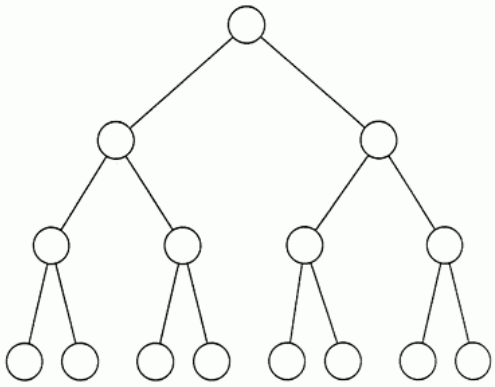
http://www.cs.concordia.ca/~twang/WangApr01/RootWang.html

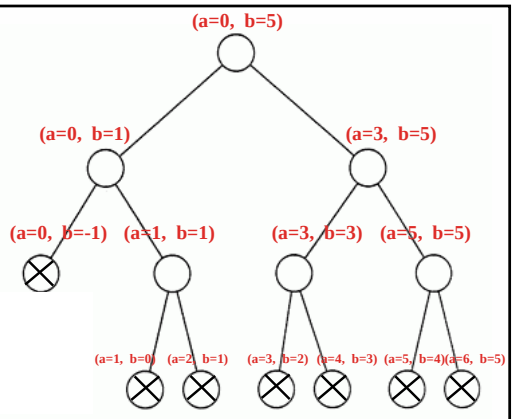
Mystery Recursion on HW8

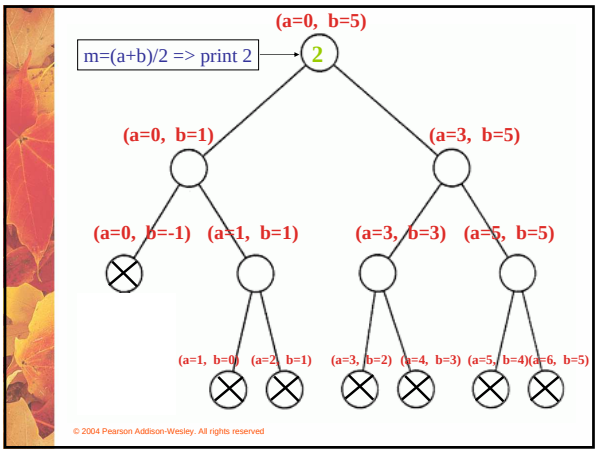
```
public static void mystery1(int a, int b)
{
        if (a <= b) {
            int m = (a + b) / 2;
            System.out.print(m + " ");
            mystery1(a, m-1);
            mystery1(m+1, b);
        }
}

public static void main(String[] args) {
        mystery1(0, 5);
        System.out.println();
}
```

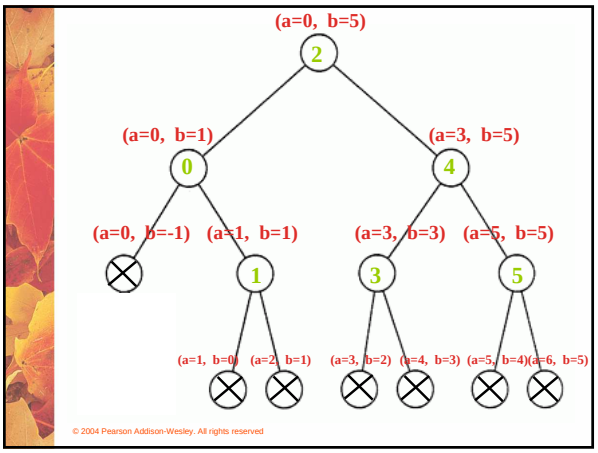

Think of recursion as a tree …

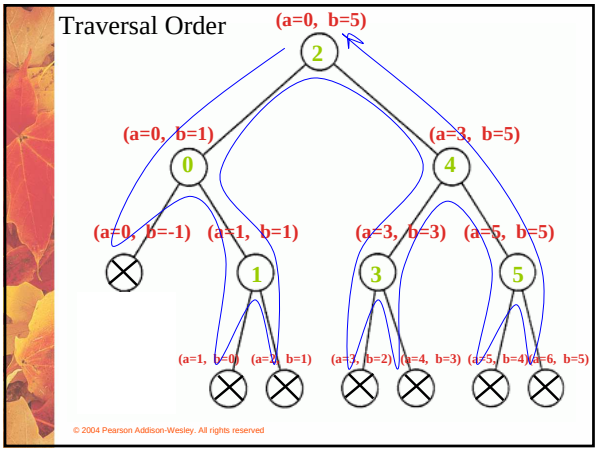
… an upside down tree











Example: Recursion_Debug.java

Recursion: Fibonacci Numbers

$$F_n = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ F_{n-1} + F_{n-2}, & n \geq 2 \end{cases}$$

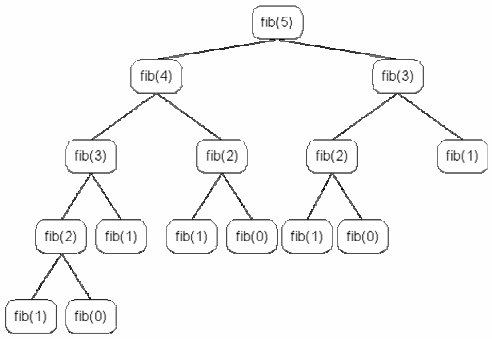
The sequence: {0,1,1,2,3,5,8,13,...}

Mathematical notation v.s. java code

$$F_n = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ F_{n-1} + F_{n-2}, & n \geq 2 \end{cases}$$

```
public static int fib(int n)
{
    if(n <= 1) return n; //base case
    else return fib(n-1) + fib(n-2);
}
```

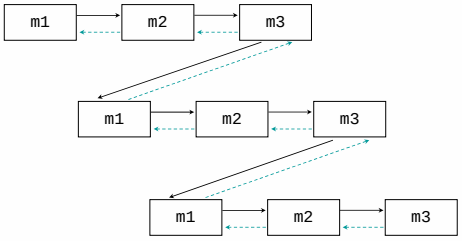
Execution Trace



Indirect Recursion

- A method invoking itself is considered to be *direct recursion*
- A method could invoke another method, which invokes another, etc., until eventually the original method is invoked again
- For example, method `m1` could invoke `m2`, which invokes `m3`, which in turn invokes `m1` again
- This is called *indirect recursion*, and requires all the same care as direct recursion
- It is often more difficult to trace and debug

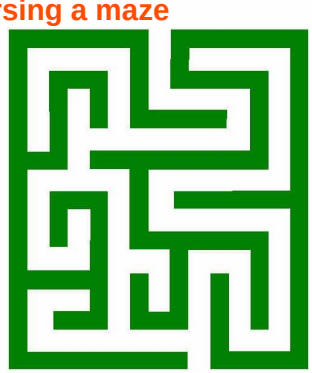
Indirect Recursion

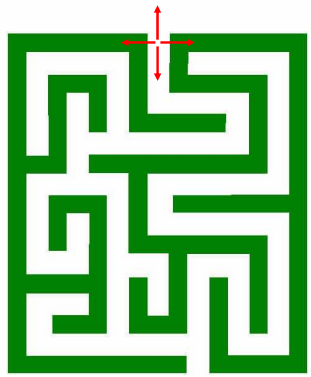


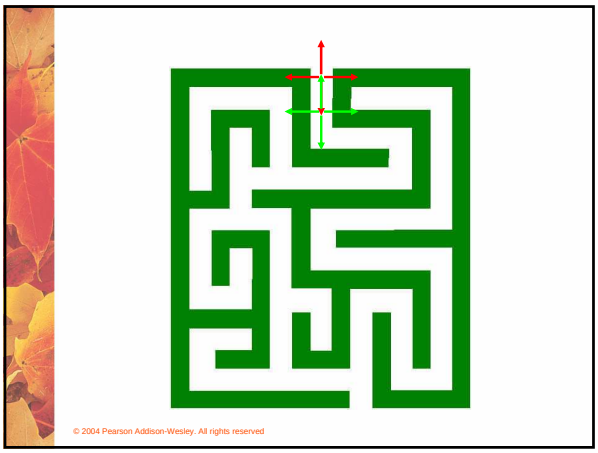
Maze Traversal

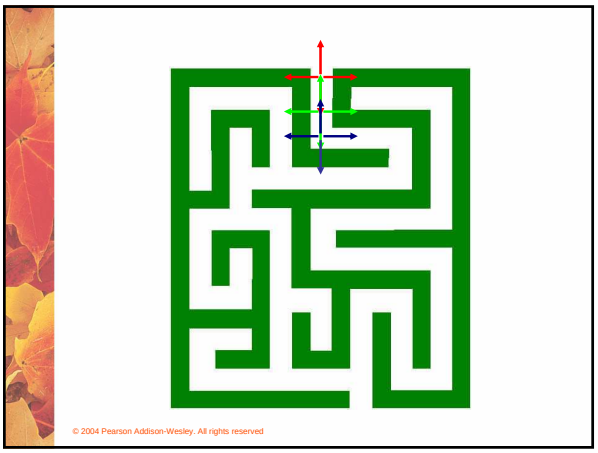
- We can use recursion to find a path through a maze
- From each location, we can search in each direction
- Recursion keeps track of the path through the maze
- The base case is an invalid move or reaching the final destination
- See `MazeSearch.java` (page 583)
- See `Maze.java` (page 584)

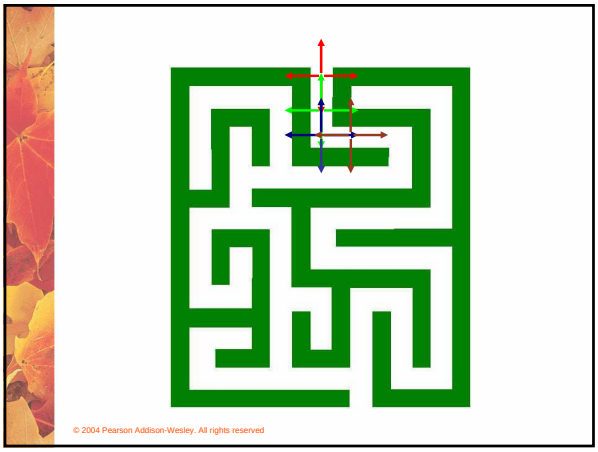
Traversing a maze











THE END