Recursion Revisited

Chapter 8
Fundamentals

• A method is **recursive** if it calls itself.
• A recursive method should contain a **stopping case** or **base case** that is not recursive.
  (Without it, the method would be infinitely recursive, never ending.)
• The recursive call(s) should be for simpler versions of the same problem.
Activation Records

• When a method calls another method (even itself), an activation record is stored on the system stack (of the O.S.).

• An activation record contains:
  - where to return when the called method ends
  - parameter(s) passed to the called method
  - values of the method’s local variables

• When a method returns, it uses the top activation record on the system stack to restore the conditions before the method call.
public int factorial(int n) {
    if (n == 0)
        return 1;
    int x = factorial(n - 1);
    return n * x;
}

result = factorial(4);  // return location A
System.out.println(result);  // return location B
Factorial

Activation Record

<table>
<thead>
<tr>
<th>x</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td></td>
</tr>
<tr>
<td>Return location</td>
<td></td>
</tr>
</tbody>
</table>
if (n=0) return 1;
x = factorial(n-1);
return n * x;
public int factorial(int n) {
    if (n == 0) return 1;
    int x = factorial(n - 1);
    return n * x;
}

factorial 4          return 4 * 6 = 24
factorial 3          return 3 * 2 = 6
factorial 2          return 2 * 1 = 2
factorial 1          return 1 * 1 = 1
factorial 0          return 1
Activation Records (summary)

- Hold return location.
- Temporary storage for local variables including parameters, if any.
- Basis for re-entrant code.
public int fib(int n) {
    if (n == 0 || n == 1)
        return n;
    int x = fib(n-1);  \(\text{← return location B}\)
    int y = fib(n-2);  \(\text{← return location C}\)
    return (x + y);
}

result = fib(4);  \(\text{← return location A}\)
System.out.println(result);
Fibonacci Numbers

Activation Record

<table>
<thead>
<tr>
<th>y</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td></td>
</tr>
<tr>
<td>Return location</td>
<td></td>
</tr>
</tbody>
</table>
Backtracking

• An exhaustive search is a technique of generating a solution from all combinations of partial solutions.
• If any step leads to an invalid solution or infeasible solution, we backtrack to the most recent partial solution and try a different path to a full solution until we find the best solution.
Example: Searching a maze
findPath(x,y): The trick

x, y

findPath(x-1,y)  findPath(x,y-1)  findPath(x+1,y)  findPath(x,y+1)
Initial Algorithm

• Start from position x,y such that Maze[x][y] = o
• if findPath(x,y)
    output TARGET FOUND
else
    output TARGET NOT FOUND
findPath(x,y)
  if Maze[x][y] = T
    output x,y
    return true
  else if Maze[x][y] = X
    return false
  else (must be an O)
    Maze[x][y] = X
    if findPath(x-1,y) OR findPath(x,y-1)
      OR findPath(x+1,y) OR findPath(x,y+1)
      output x,y
      return true
    else return false

This is a simplification in the algorithm. What’s missing?
Dynamic Programming

• Reduce the number of recursive calls by saving the return values of recursive calls as they are determined.
• Use the saved value in place of an identical recursive call later in the execution.
• Example: let d[ ] be a global array that holds the previously determined Fibonacci values. Give these initial values of –1.
Example : Matrix Chain Multiplication

Let $A_1$ be a 100 x 10 matrix
Let $A_2$ be a 10 x 20 matrix
Let $A_3$ be a 20 x 30 matrix

How many operations are required to multiply $A_1 \times A_2 \times A_3$?
Fibonacci Numbers Revisited
Using Dynamic Programming

```java
public int fib(int n) {
    if (n == 0 || n == 1) {
        return n;
    }
    if (d[n-1] == -1)
        d[n-1] = fib(n-1);
    if (d[n-2] == -1)
        d[n-2] = fib(n-2);
    return (d[n-1]+d[n-2]);
}
```
Tail Recursion

• If a method is defined such that it has one recursive call as the last computational statement, then the method is called tail recursive.
• Every tail recursive method can be re-written as an equivalent method without recursion using a loop.
• Example: factorial is tail recursive.
Factorial Revisited

public int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n-1);
}

public int factorial(int n) {
    int product = 1;
    int i;
    for (i = n; i >= 1; i--)
        product = i * product;
    return product;
}

Why should we try to eliminate tail recursion?
public static void reversePrint(int n) {
    if (n > 0) {
        System.out.println(n);
        reversePrint(n-1);
    }
}

public static void reversePrint(int n) {
    L: if (n > 0) {
        System.out.println(n);
        Set up new parameters
        Jump to L
    }
}

while (n > 0) {
    System.out.println(n);
    n--;
    // no code necessary
}
Towers of Hanoi

public static void move(int n, int src, int dest, int aux) {
    if (n > 0) {
        move(n-1, src, aux, dest);
        System.out.println(src+"   "+dest);
        move(n-1, aux, dest, src);
    }
}

public static void move(int n, int src, int dest, int aux) {
    while (n > 0) {
        move(n-1, src, aux, dest);
        System.out.println(src+"   "+dest);
        n = n-1; exchange(aux, src);  
    }
}