Recursion

15-121

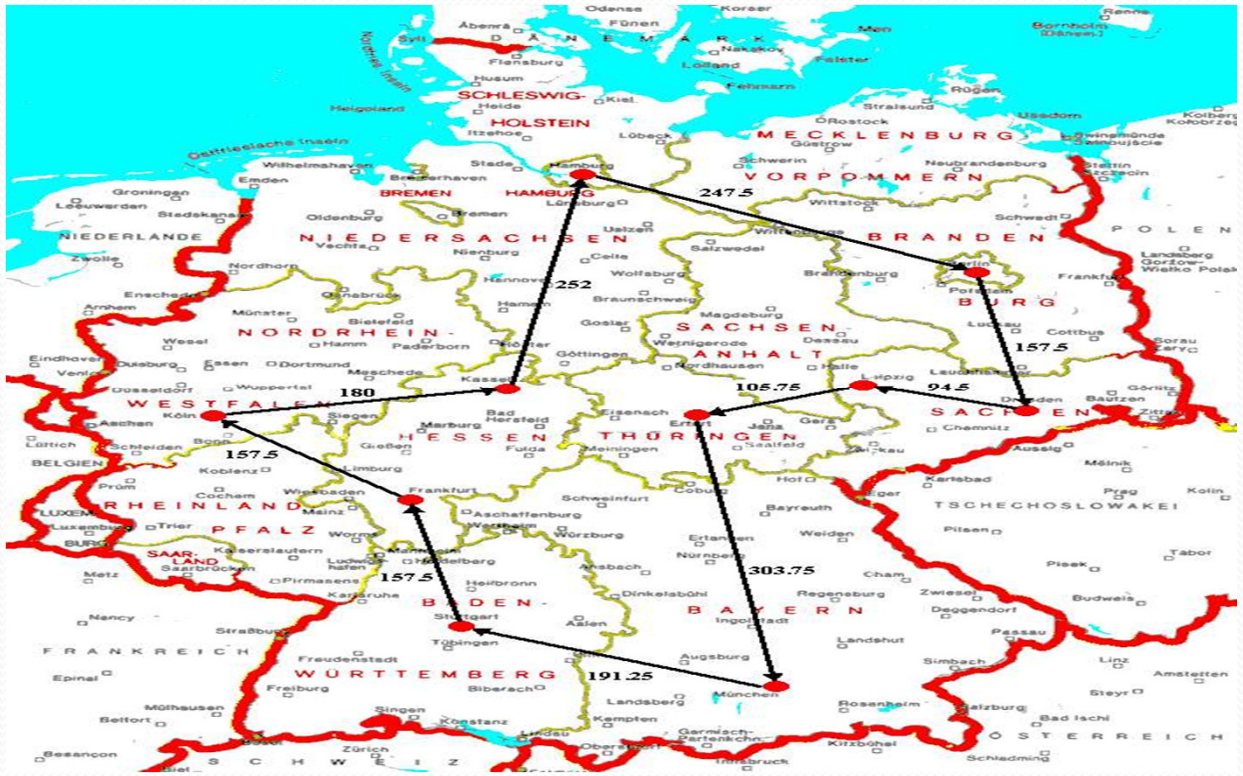
Data Structures

Ananda Gunawardena

Problem Solving Techniques

Divide and Conquer

Exhaustive Search



Greedy Algorithms

Dynamic Programming

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0											
G	0											
A	0											
T	0											
C	0											
G	0											
A	0											

applicable to problems that exhibit the properties of overlapping sub problems and optimal substructure

Recursion

The promise of recursion

- Suppose you can express the solution to a bigger problem using solution to a sub problem
  - F(n) = n * F(n-1)
- The express the solution to what is called base case
  - F(0) = F(1) = 1
- Question: What is the closed form of this function?

Change problem

- Problem: Given n cents in change, find the least number of coins to provide the change
- Solution:
  - Iterative: Keep subtracting highest coin until the balance is zero
  - Recursive: Assume you know how to express the solution using solution to a sub problem
    - $C(n) = 1 + C(n-25)$ if $(n > 25)$

What problems can be solved using Recursion

- Problems that lend themselves to recursive solution have the following characteristics
  - **one or more simple cases of the problem (stopping cases) have a straightforward, non-recursive solution**
  - **For the other cases, there is a process (using recursion) for substituting one or more reduced cases of the problem closer to a stopping case**
  - **Eventually the problem can be reduced to stopping cases only**

General Form of a recursive Function

- The recursive functions we work with will generally consist of an **`if`** statement with the form shown below

```
if the stopping case or base case is reached
     solve the problem
else
     reduce the problem using
     recursion
```

Recursion

- Examples

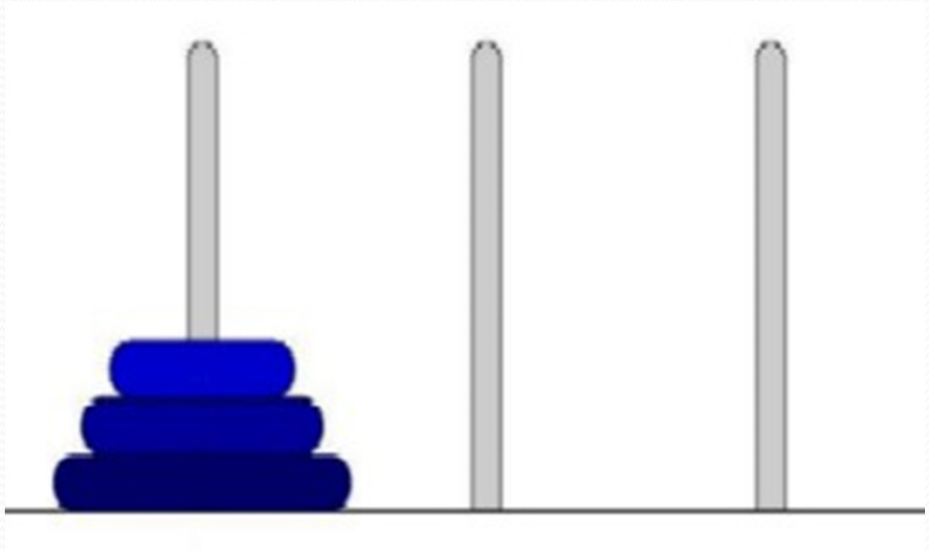
GCD(a,b) = a if a=b

= o if a=o or b=o

= GCD(a%b, b) if a > b

= GCD(a, b%a) if b > a

Tower of Hanoi Problem



Rules of the game

- Move one disk at a time
- Cannot place a larger disk on the top of a smaller disk
- Find
  - Moves you need to solve Hanoi(n) problem

Thinking about the game

- Consider small cases
  - N = 1 trivial
  - N = 2
  - N = 3

Generalizing Hanoi

Suppose you need to move n disks from a **origin** to **Destination** using **intermediate**

- **You break the problem into parts**
  - move first (n-1) disks from origin to intermediate using destination
  - move the n-th disk from origin to destination
  - move the (n-1) disks from intermediate to destination using origin

Tracing Hanoi

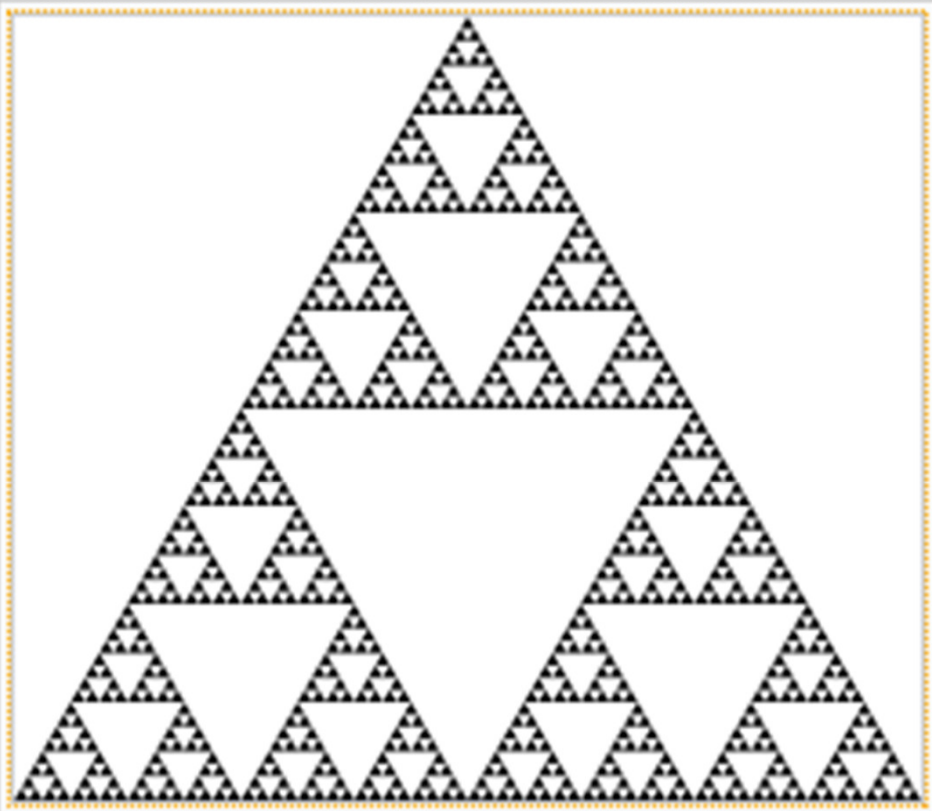
Tracing Recursion

```
public void foo(n) {
    if  (n == 0) return 0;
    else return n+foo(n-1);
}
```

Recursive Solutions

- Implementing a recursive solution is generally less efficient in terms of system overhead, due to the overhead of extra function calls;
- however recursive functions
  - allow us to think of solutions to problems that may be recursive in nature
  - allow us to work with data structures that are best accessed recursively
    - *Eg: binary search trees*

Famous Recursive Solutions



A Sierpinski triangle—a confined recursion of triangles to form a geometric lattice.

Types of Recursion

- Head recursion

```
public void foo(n){
    if (n>0)  foo(n-1);
    System.out.println(n);
}
```

- Tail Recursion

```
public void foo(n){
   if (n>0) System.out.println(n);
    else foo(n-1);
}
```

What is the output?

```
public void printPattern(int n){
    if  (n > 0) {
        printPattern(n-1);
        printStars(n);
        printPattern(n-1);
    }
```

Where printStars(n) prints n stars.

Eg: printStarts(3) ➔ ***

Next

- List and Recursion
- Binary Search using Recursion
- Maze Solver
  - Thinking recursively