DATA STRUCTURES (ITPC-203)

Principles of Recursion

Mrs. Sanga G. Chaki

Department of Information Technology

Dr. B. R. Ambedkar National Institute of Technology, Jalandhar

Contents

Recurrence Relations

1. A sequence is a discrete structure used to represent an ordered list.
2. A **recurrence relation** for the sequence $\{a_n\}$ is an equation that expresses a_n in terms of one or more of the **initial terms** of the sequence, namely, $a_0, a_1, \ldots, a_{n-1}$, for all non-negative integers n
3. A sequence is called a **solution of a recurrence relation** if its terms satisfy the recurrence relation.
4. A recurrence relation is said to **recursively define a sequence**.
5. The **initial conditions** for a recursively defined sequence specify the terms that precede the first term where the recurrence relation takes effect.

Recurrence Relations – Fibonacci Sequence

1. The Fibonacci sequence, $f_0, f_1, f_2, \ldots,$ is defined by the **initial conditions**
   - $f_0 = 0, f_1 = 1$ → The first two terms in the sequence
2. The **recurrence relation**

$$\mathbf{f_n = f_{n-1} + f_{n-2} \text{ for } n = 2, 3, 4, \ldots .}$$

   - **Each term is defined using the previous two terms of the sequence.**
   - **So the sequence is defined in terms of itself.**

Recurrence Relations

1. We say that we have solved the recurrence relation together with the initial conditions when we find an **explicit formula**, called a **closed formula**, for the terms of the sequence. – this formula does not use previous terms in the sequence to find each successive terms.

Recurrence Relations

1. **Example**: Determine whether the sequence $\{a_n\}$, where $\mathbf{a_n = 3n}$ for every nonnegative integer n, is a solution of the recurrence relation

$$\mathbf{a_n = 2a_{n-1} - a_{n-2}} \text{ for } n = 2, 3, 4\ldots$$

Closed formula

Recurrent formula

- Suppose that $a_n = 3n$ for every nonnegative integer n.
- Then, for any $n \geq 2$, we see that
- $\mathbf{a_n = 3n,}$
- $\mathbf{a_{n-1} = 3(n-1)}$
- $\mathbf{a_{n-2} = 3(n-2)}$ → replace them in the recurrent relation

$$2a_{n-1} - a_{n-2} = 2(3(n-1)) - 3(n-2) = 3n = a_n.$$

- **Therefore, the sequence $\{a_n\}$, where $a_n = 3n$, is a solution of the given recurrence relation**

Principles of recursion

Recursion

1. In computer science, recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem.
2. Recursion solves such recursive problems by using functions that call themselves from within their own code.

Recursion

1. A process by which a function calls itself repeatedly is called recursion
2. In the below example, the factorial of n is computed using the results of smaller instances of the same problem – namely, the factorial of (n-1).
3. This again, would be computed using the results of fact(n-2) and so on.

- Used for repetitive computations in which each action is stated in terms of a previous result.

```
fact(n) = n * fact (n-1)
```

Recursion

- For a problem to be written in recursive form, two conditions are to be satisfied:
  - It should be possible to express the problem in recursive form.
  - The problem statement must include a stopping condition

```
fact(n)  =  1,                 if  n = 0
         =  n * fact(n-1),     if  n > 0
```

Recursion

Some example problems which can be solved using recursion:

1. Factorial – previous example.
2. Fibonacci Series (0, 1, 1, 2, 3, 5, 8, 13, ….) – first example
3. GCD – finding the greatest common divisor – express this problem in recursive way.
4. Finding the length of a string - express this problem in recursive way.
5. Find an example problem which cannot be solved using recursion.

Recursion – Mechanism of Execution

```
long  int  fact (n)
int  n;
{
    if    (n == 0)
        return (1);
    else
        return  (n * fact(n-1));
}
```

- **Mechanism of execution**
  - When a recursive program is executed, the recursive function calls are not executed immediately.
    - They are kept aside (on a stack) until the stopping condition is encountered.
    - The function calls are then executed in reverse order.

Recursion – Mechanism of Execution

Example :: Calculating `fact(4)`

- First, the function calls will be processed:

```
fact(4) = 4 * fact(3)
fact(3) = 3 * fact(2)
fact(2) = 2 * fact(1)
fact(1) = 1 * fact(0)
```

- The actual values return in the reverse order:

```
fact(0) = 1
fact(1) = 1 * 1 = 1
fact(2) = 2 * 1 = 2
fact(3) = 3 * 2 = 6
fact(4) = 4 * 6 = 24
```

Recursion – Fibonacci Numbers

- Fibonacci number f(n) can be defined as:

```
f(0)  =  0
f(1)  =  1
f(n)  =  f(n-1) + f(n-2),   if  n > 1
```

  - The successive Fibonacci numbers are:

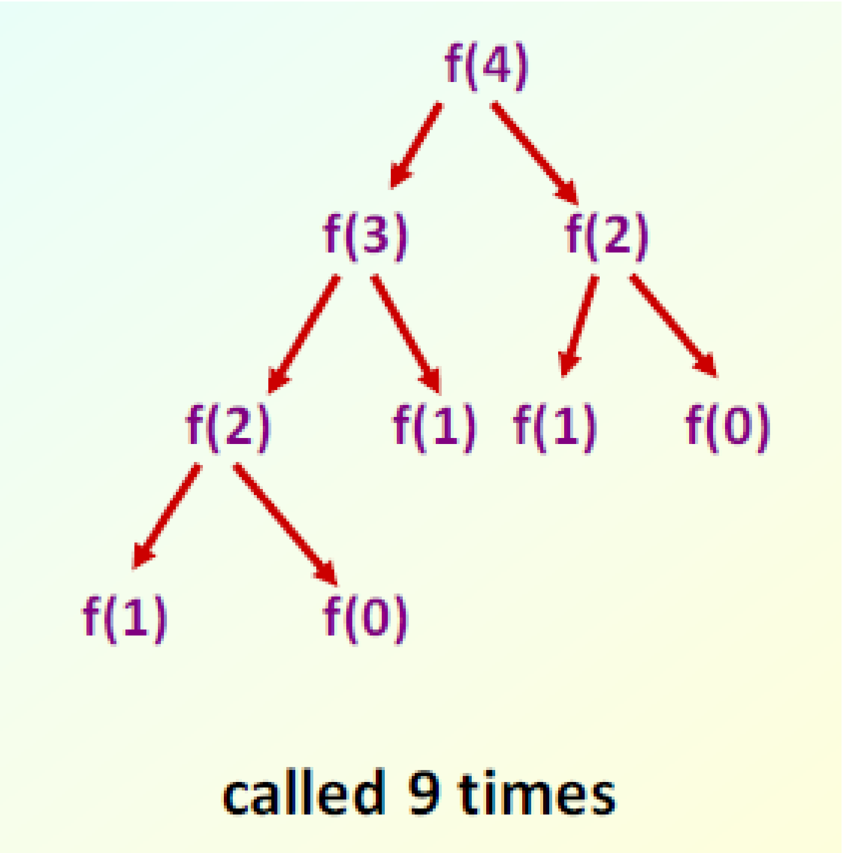
    0, 1, 1, 2, 3, 5, 8, 13, 21, ...

- Function definition:

```
int  f (int n)
{
      if  (n  < 2)   return (n);
      else  return (f(n-1) + f(n-2));
}
```

Recursion – Fibonacci Numbers

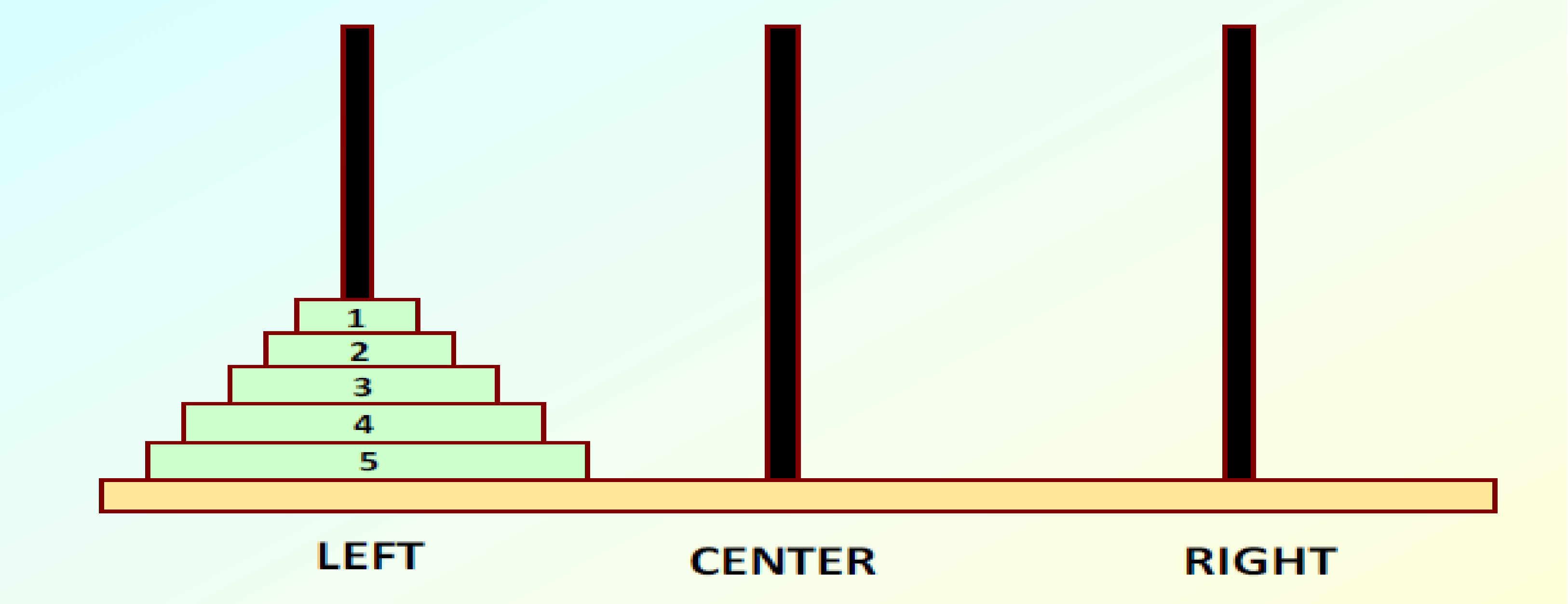
How many times the function is called when evaluating f(4) ?



1. **Same thing is calculate multiple times – inefficient.**
2. **Exponential "explosion" of calls**
3. **Takes more time and memory**

Recursion – Tower of Hanoi

- **The problem statement:**
  - Initially all the disks are stacked on the LEFT pole.
  - Required to transfer all the disks to the RIGHT pole.
    - Only one disk can be moved at a time.
    - A larger disk cannot be placed on a smaller disk.
  - CENTER pole is used for temporary storage of disks.



Recursion – Tower of Hanoi

- Recursive statement of the general problem of n disks.
  - Step 1:
    - Move the top (n-1) disks from LEFT to CENTER.
  - Step 2:
    - Move the largest disk from LEFT to RIGHT.
  - Step 3:
    - Move the (n-1) disks from CENTER to RIGHT.

Recursion – Tower of Hanoi

Moved disk 3 from L to R

Moved disk 2 from L to R

Moved disk 1 from L to R

```
3
Move disk 1 from L to R
Move disk 2 from L to C
Move disk 1 from R to C
Move disk 3 from L to R
Move disk 1 from C to L
Move disk 2 from C to R
Move disk 1 from L to R
```

Recursion – Tower of Hanoi

```
4
Move disk 1 from L to C
Move disk 2 from L to R
Move disk 1 from C to R
Move disk 3 from L to C
Move disk 1 from R to L
Move disk 2 from R to C
Move disk 1 from L to C
Move disk 4 from L to R
Move disk 1 from C to R
Move disk 2 from C to L
Move disk 1 from R to L
Move disk 3 from C to R
Move disk 1 from L to C
Move disk 2 from L to R
Move disk 1 from C to R
```

Moved disk 4 from L to R

Moved disk 3 from L to R

Moved disk 2 from L to R

Moved disk 1 from L to R

Thanks!