Chapter 6

Recursion

Data Structure & Algorithm

Compiled by :

Ankit Bhattarai
Email: ankit.bca@kathford.edu.np

Syllabus

Unit	Contents	Hours	Remarks
1.	Introduction to Data Structure	2	
2.	The Stack	3	
3.	Queue	3	
4.	List	2	
5.	Linked Lists	5	
6.	Recursion	4	
7.	Trees	5	
8.	Sorting	5	
9.	Searching	5	
10.	Graphs	5	
11.	Algorithms	5	

Credit : 3

Outlines

- Introduction
- Principle of recursion
- Recursion Vs Iteration
- Advantages/Disadvantages of Recursion
- Recursion Example: TOH & Fibonacci Series
- Application of Recursion

Introduction: Recursion

- ✓ Recursion is the process of expressing a function in terms of itself.
- ✓ Definition:

A function is said to be recursively defined, if a function containing either a Call statement to itself or a Call statement to a second function that may eventually result in a Call statement back to the original function.

A recursive functions must have the following properties:

1. **Base condition**: There must be at least one base criteria or condition, such that, when this condition is met the function stops calling itself recursively.
2. **Progressive approach**: The recursive calls should progress in such a way that each time a recursive call is made it comes closer to the base condition.

Introduction: Recursion

- Calculating factorial of an number using recursion

```
#include<stdio.h>
#include<conio.h>
long int factorial(int n);
int main() {
   int n;
   printf("Enter a positive integer: ");
   scanf("%d",&n);
   printf("Factorial of %d = %ld", n, factorial(n));
   return 0;
}

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

Introduction: Recursion

- ✓ One of the most critical aspects of writing either a recursive function is to have an exit clause (or "base case" for recursions) that is checked at every recursion and is guaranteed to be reached at some finite point, given any input. Otherwise you will get either:
- **Infinite Recursion**: More and more functions being spawned, never closing, using up resources until there are none left, possibly crashing the system.

Why use recursion ?

- Recursion is made for solving problems that can be broken down into smaller, repetitive problems.
- It is especially good for working on things that have many possible branches and are too complex for an iterative approach.

Principle of Recursion

Principle of Recursion

- Recursion is implemented through the use of functions.
- A function that contains a function call to itself or a function call to a second function which eventually calls the first function is known as a recursive function.

Two important **conditions** must be satisfied by any recursive function:

1. There must be a decision criteria for stopping a process or computation
2. Each time a function calls itself it must be closer, in some sense to a solution.

Principle of Recursion

For designing the **good recursive program** we must take certain assumptions such as:

1. **Base case** : Base case is the terminating condition for the problem while designing any recursive function.
2. **If conditions**: If condition in the recursive algorithm defines the terminating condition.
3. When a recursive program is subjected to for execution, the recursive function calls will not be executed immediately.
4. The initial parameter input value pushed on to the stack.

Principle of Recursion

5. Each time a function is called a new set of local variable and formal parameters are again pushed on the stack and execution starts from the beginning of the function using the changed new value. This process is repeated till a base condition (stopping condition) is reached.
6. Once a base condition or stopping condition is reached the recursive function calls popping elements from stack & returns a result to the previous values of the function.
7. A sequence of returns ensures that the solution to the original problem is obtained.

Advantages/ Disadvantages of Recursion

Advantages of Recursion

1. Recursion is very flexible in data structure like stacks, queues, linked list & quick sort.
2. Recursion may lead to simpler, shorter, easier-to-understand functions, especially for mathematicians who are comfortable with induction formulae.
3. The length of the program can be reduced using recursion.

Disadvantages of Recursion

1. It uses more memory resources, because of the extra function calls.
2. It consumes more time to get the desired result & may be slower.
3. If recursive calls are not checked, the computer may run out of memory.

Recursion Vs Iteration

Recursion

It is the technique of defining anything in terms of itself.

For example:

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

Iteration

The process of executing a statement or a set of statements repeatedly, until some specified condition is satisfied.

For example:

```
int factorial(int n)
{
    int i;
    int prod = 1;
    for (i=1;i<n; i++)
       {
       prod = prod * i;
       }
    return (prod);
}
```

Recursion	Iteration
There is a base condition inside the recursive function, specifying stopping condition.	It involves four clear-cut steps like initialization, condition, execution & updating.
It uses selection structures such as if statement, if – else or switch statement and achieves repetition through function calls itself repeatedly.	It uses repetitions structures such as for loop, while - loop and do – while and explicitly uses repetition structures.
Not all problems have recursive solution.	Any recursive problem can be solved iteratively.
It is generally a worse option to go for simple problems, or problems not recursive in nature.	Iterative counterpart of a problem is more efficient in terms of memory, utilization & execution speed.

Tower of Hanoi (TOH)

Recursion Example: TOH (Tower of Hanoi)

- Tower of Hanoi is a mathematical puzzle which consists of three towers (pegs) and a number of disks of different sizes, which can slide onto any peg.
- The problem has a historical basis in the ritual of the ancient Tower of Brahman.
- This is the problem which is not specified in terms of recursion and we have to see how we can use recursion to produce logical and elegant solution.

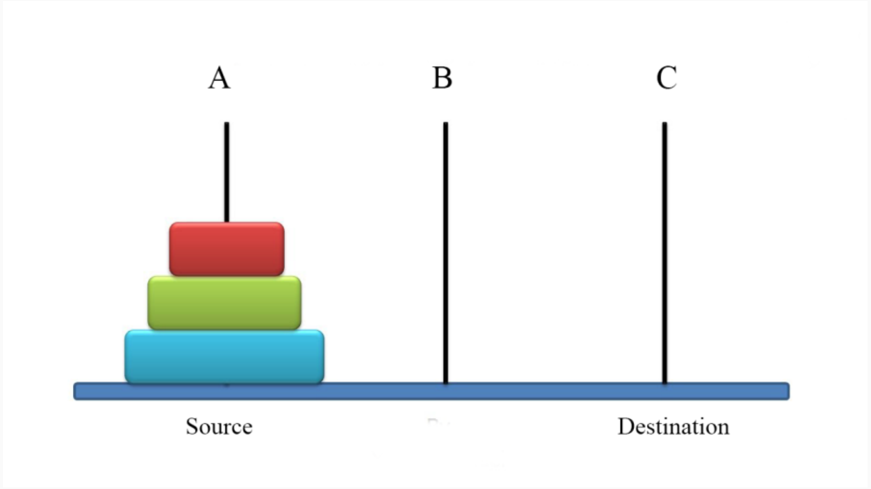


Fig. Tower of Hanoi

Recursion Example: TOH (Tower of Hanoi)

The mission is to transfer all the disks from source peg A to the destination peg C by using an intermediate peg B.

A few **rules** to be followed for Tower of Hanoi are:

- Transfer the disks from the source peg to the destination peg such that at any point of the transformation no large size disk is placed on the smaller one.
- Only one disk may be moved at a time.
- Each disk must be stacked on any one of the pegs.

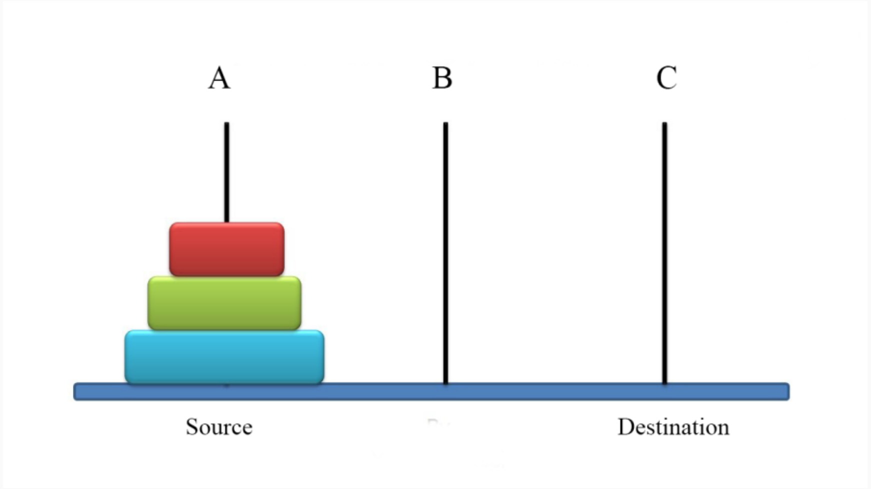


Fig. Tower of Hanoi

Recursion Example: TOH (Tower of Hanoi)

Generalize solution of tower of Hanoi problem recursively as follows:

To move n disks from source to destination, using a auxiliary peg.

1. If n=1,move the single disk from source to destination & stop.
2. Move the top(n-1) disks from the source to the auxiliary.
3. Move n^{th} disk from source to destination.
4. Now move n-1 disk from auxiliary to destination.

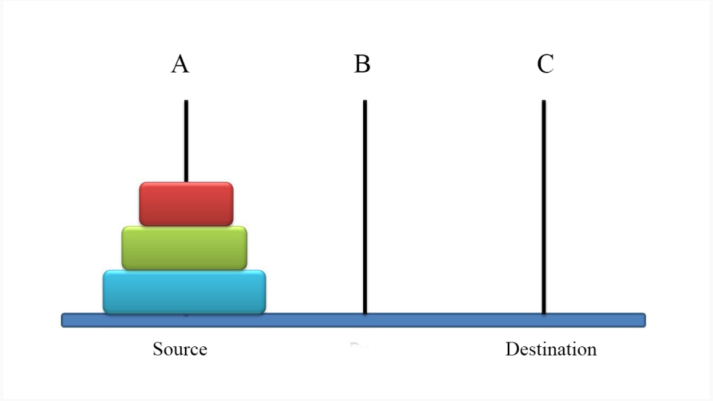


Fig. Tower of Hanoi

Recursion Example: TOH (Tower of Hanoi)

Tower of Hanoi puzzle with n disks can be solved in minimum 2^n-1 steps. This presentation shows that a puzzle with 3 disks has taken $2^3 - 1 = 7$ steps.

Now following could be the steps to solve the tower of Hanoi problem for 3 disks.

Move disc 1 from A to C

Move disc 2 from A to B

Move disc 1 from C to B

Move disc 3 from A to C

Move disc 1 from B to A

Move disc 2 from B to C

Move disc 1 from A to C

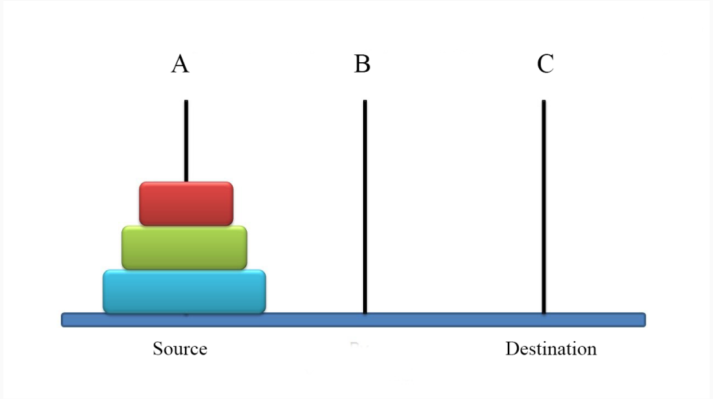
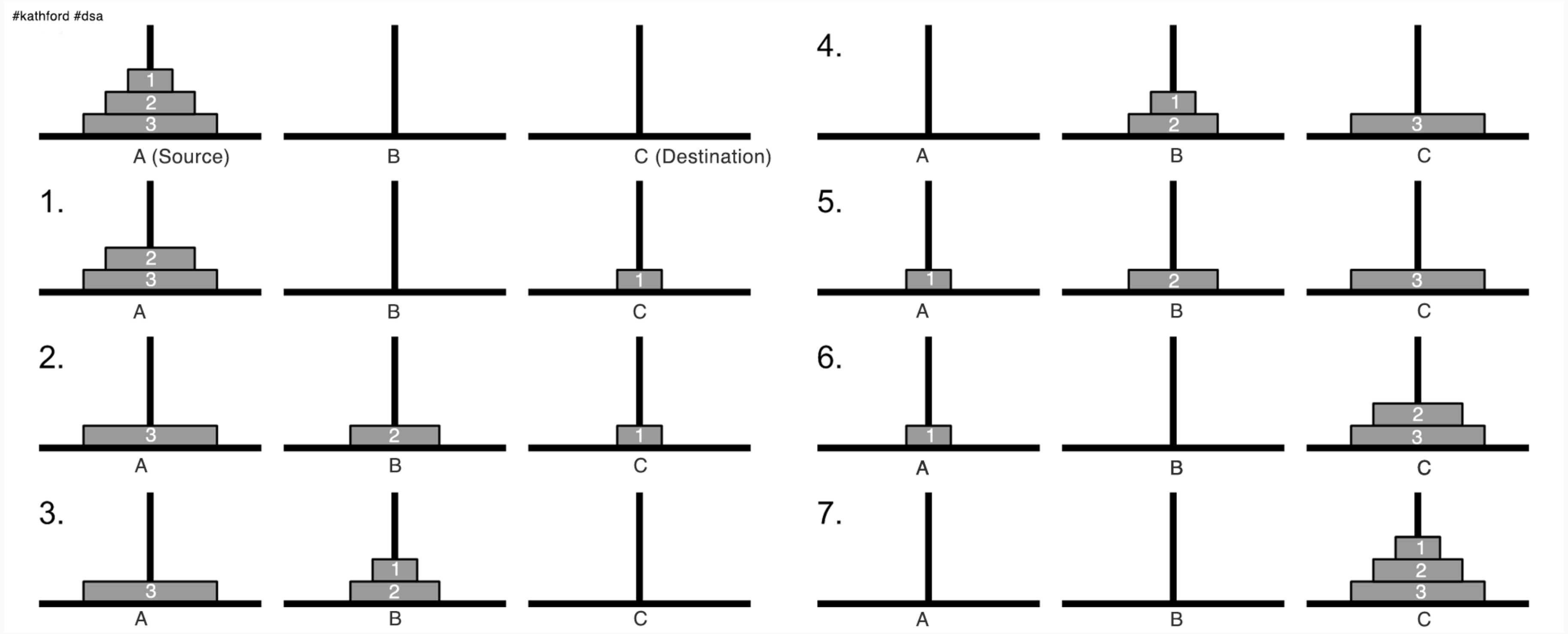


Fig. Tower of Hanoi

Recursion Example: TOH (Tower of Hanoi)



Recursion Example: Fibonacci Series

Fibonacci can be defined recursively as follows. Each number is the sum of the previous two numbers in the sequence. The nth Fibonacci number is

Fib(n) = Fib(n-1) + Fib(n-2)

Fib(0) = 1,

Fib(1) = 1.

That is, the nth number is formed by adding the two previous numbers together. The equation for Fib(n) is said to be the recurrence relation and the terms for Fib(0) and Fib(1) are the base cases.

Recursion Example: Fibonacci Series

The code for the recursive Fibonacci sequence is provided in below.

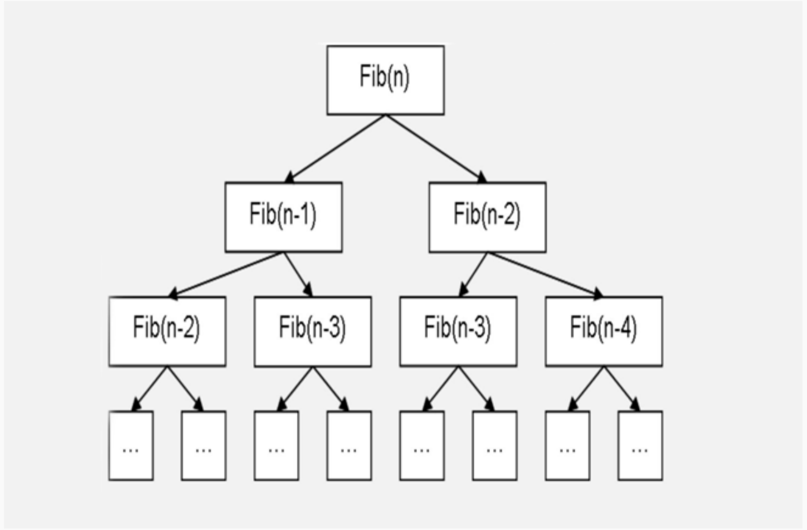
```
int fibonacci(int n)
{
   if(n == 0 || n == 1)
       return n;
  else
  return(fibonacci(n-1) + fibonacci(n-2));
}
```

```
#include<stdio.h>
#include<conio.h>>
int fibonacci(int);
int main()
{
                int n, m= 0, i;
                printf("Enter Total terms:");
                scanf("%d", &n);
                printf("Fibonacci series terms are: \n");
                for(i = 1; i <= n; i++)
                     {
                      printf("%d\n", fibonacci(m));
                      m++;
                      }
                return 0;
}

int fibonacci(int n)
{
                if(n == 0 || n == 1)
                       return n;
                else
                     return(fibonacci(n-1) + fibonacci(n-2));
}
```

Recursion Example: Fibonacci Series

- To find Fib(*n*) you need to call Fib two more times: once for Fib(*n*-1) and once more for Fib(*n*-2); then you add the results.
- But to find Fib(*n*-1), you need to call Fib two more times: once for Fib(*n*-2) and once more for Fib(*n*-3); then you need to add the results. And so on!



✓ You can see the recursion tree for Fib(*n*) in Figure above. The recursion tree shows each call to Fib() with different values for *n*.

Recursion Example: Fibonacci Series

- For example, consider Fib(5), the fifth term in the sequence.
- To find Fib(5), you would need to find Fib(4) + Fib(3). But to find Fib(4), you would need to find Fib(3) + Fib(2).
- Luckily, Fib(2) is a base case; the answer to Fib(2) is 1. And so on.
- Look at Figure above for an example recursion tree for Fib(5) : it lists all of the computations needed to carry out the calculation recursively

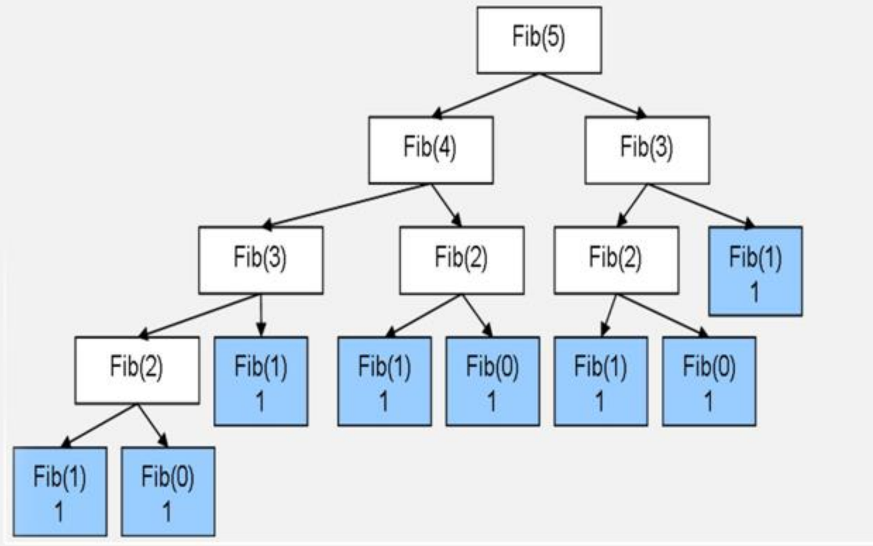


Fig. Recursion tree for Fib(5), Base condition are highlighted

Application of Recursion

1. Recursion is used to compute the factorial function, to determine whether a word is a palindrome, to compute powers of a number, to find the Fibonacci series, and to solve the ancient Towers of Hanoi problem
2. It is used in applications like Candy Crush in which Fibonacci sequence and the Ackermann to group tiles combination.
3. It is being implemented in many search algorithm for artificial intelligence.
4. It is used in sorting algorithm like quick sort & merge sort.
5. Recursion is used in things like BSP(Binary Space Partitioning) trees for collision detection in game development.

Questions to focus

1. What is the base condition of the recursive function? Compare recursion and iterative solution with example. Which one would you choose?
2. Define recursion. Discuss Tower of Hanoi (TOH) problem with its solution for 3 discs.
3. Discuss Tower of Hanoi (TOH) problem with its solution for 4 discs.
4. Distinguish between recursion & iteration with the help of Fibonacci series example for both the case.
5. What is recursion? What are its advantages and disadvantages?
6. Give the difference between recursion and iteration by taking reference of factorial function.
7. Differentiate recursion with iteration.
8. Write down the algorithm to solve the tower of Hanoi using recursion.
9. Compare recursion and iterative solution with example. Which one would you choose?
10. Explain the recursive procedure to solve the TOH problem.

THANK YOU

Any Queries ?

ankit.bca@kathford.edu.np