UNIT- SIX
RECURSION

Topics to be Covered

- Introduction
- Principle of recursion
- Recursion Vs Iteration
- Recursion Example: TOH and Fibonacci
- Application of Recursion
- Search Tree

Introduction: Recursion

- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.
- A recursive function solves a particular problem by calling a copy of itself and solving smaller sub problems of the original problems.
- Many more recursive calls can be generated as and when required. It is essential to know that we should provide a certain case in order to terminate this recursion process.
- Using a recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi(TOH), Inorder / Preorder/ Postorder Tree Traversals, DFS of Graph, etc.

Recursion Basics:

- The program should call itself
- There should be a stopping criteria for a program to stop.

Example: A Simple C program to find the factorial of a number provided by user

```
#include <stdio.h>
int fact(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fact(n-1);
}

int main()
{
    int n,f;
    printf("Enter any number: ");
    scanf("%d", &n);
    f = fact(n);
    printf("factorial = %d",f);
}
```

Types of Recursion

- Recursion is of two types depending on whether a function calls itself from within itself, or whether two functions call one another mutually.

i) Direct recursion: the function calls itself

ii) Indirect recursion: two function calls one another mutually.

- Recursion may be further categorized as:

a) Linear recursion

b) Binary recursion

c) Multiple recursion

a) Linear recursion:

- linear recursion begins by testing set of base cases there should be at least one. Every possible chain of recursive calls must eventually reach base case, and the handling of each base case should not use recursion.
- **In linear recursion we follow:**

  -Perform a single recursive call: In this process the recursive step involves a test which decide out of all the several possible recursive calls which one is make, but it should ultimately choose to make just one of these calls each time we perform this step.

  -Define each possible recursive call, so that it makes progress towards a base case.

A simple example of linear recursion.

Input: An integer array A and an integer n=1, such that A has at least n elements.

Function: LinearSum(A,n)

Output: The sum of first n integer in A

```
    If n==1 then
    return A[0]
    else
    return Linear Sum (A, n-1) + A[n-1]
```

b) Binary recursion: Binary recursion occurs whenever there are two recursive calls for each non base case. Example: the problem to add all the numbers in an integer array **A**.

Algorithm:

```
BinarySum (A, i, n)
Input: An array A and integers i and n.
Output: The sum of the integers in A starting from index i,
If n==1 then
return A[i]
else
return BinarySum [A, i, n/2] + BinarySum [A, i+n/2, n/2]
```

c) Multiple recursion: In multiple recursion we make not just one or two calls but many recursive calls. One of the motivating examples is summation puzzles.

Pot + pan = bib

Dog + cat = pig

Boy + girl = baby

To solve such a puzzle, we need to assign a unique digit (that is 0, 1, 2,…..9) too each letter in the equation, in order to make the equation true. Typically, we solve such a puzzle by using out human observation of the particular we are trying to solve to eliminate configurations (that is, possible partial assignment of digit to letters) until we can work through the feasible configurations left, testing for the correctness of each one.

Iteration

- In Iteration, there is the usage of loops to execute the set of instructions repetitively until the condition of the iteration statement becomes false. It is comparatively faster than recursion. It has a larger code size than recursion. The termination in iteration happens when the condition of the loop fails.

```
#include <stdio.h>
int fact(int num)
{
  int res = 1, i;
   for (i = 2; i <= num; i++)
      { res *= i;  }
   return res;
}
int main()
{
   int num, f;
   printf("Enter any number: ");
   scanf("%d", &num);
   f = fact(num);
   printf("factorial = %d", f);
}
```

Recursion VS Iteration

- Recursion uses **selection structure**.
- **Infinite recursion** occurs if the recursion step does not reduce the problem in a manner that converges on some condition (**base case**) and Infinite recursion can crash the system.
- Recursion terminates when a **base case** is recognized.
- Recursion is usually **slower than iteration** due to the overhead of maintaining the stack.
- Recursion uses **more memory than iteration**.
- Recursion makes the **code smaller**.

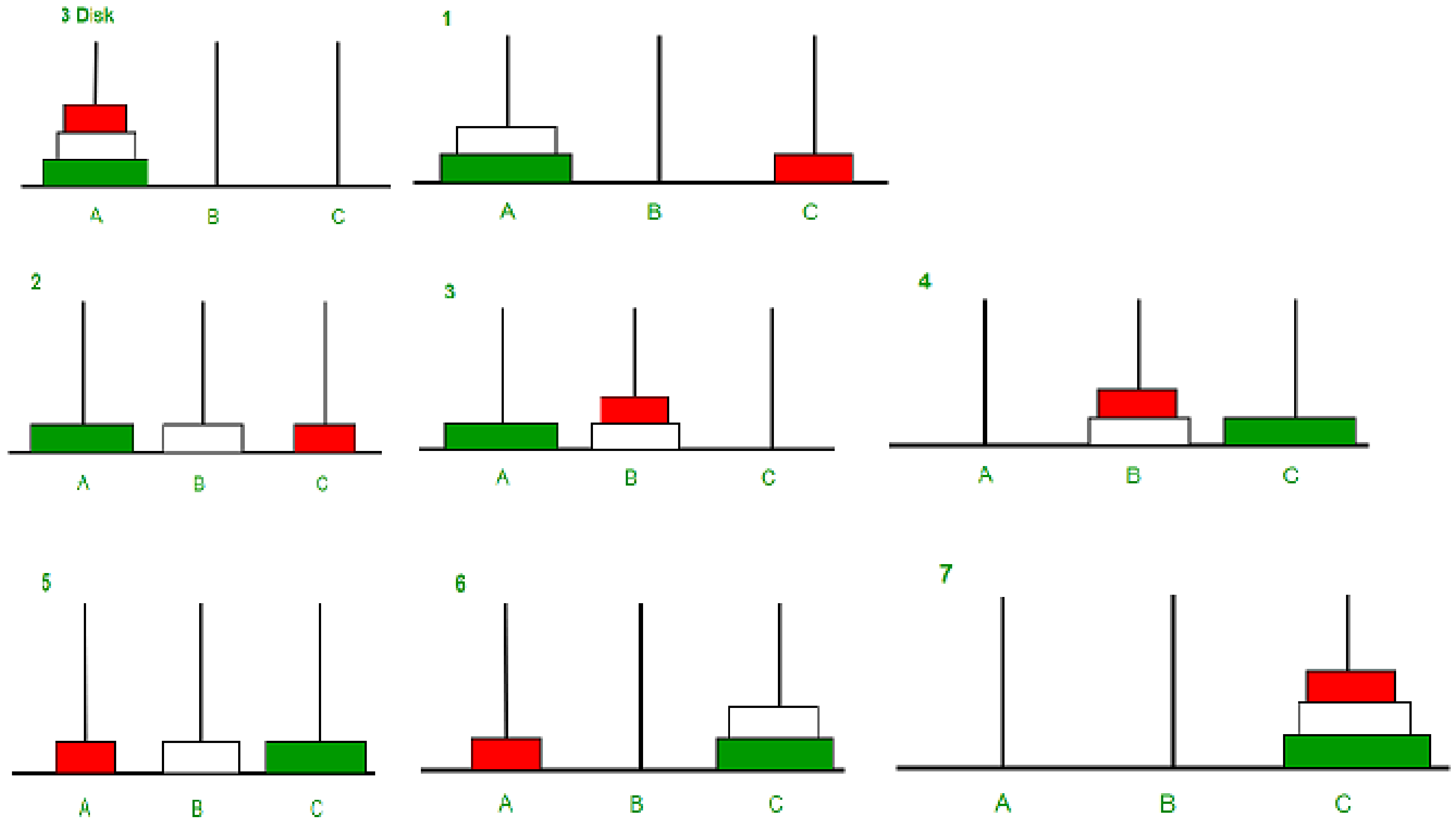
- Iteration uses **repetition structure**.
- An infinite loop occurs with iteration if the loop condition test never becomes false and Infinite looping uses CPU cycles repeatedly.
- An iteration **terminates** when the **loop condition fails**.
- An iteration does not use the **stack** so it's **faster than recursion**.
- Iteration consumes **less memory.**
- Iteration makes the **code longer**.

Recursion Example: Tower of Hanoi (TOH)

Tower of Hanoi is a mathematical puzzle where we have three rods (**A**, **B**, and **C**) and **N** disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod **A**. The objective of the puzzle is to move the entire stack to another rod (here considered **C**), obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

Graphical illustration of TOH Algorithm



Example

- ***Input***: *2 Disks*
  Output: *Disk 1 moved from A to B*
  Disk 2 moved from A to C
  Disk 1 moved from B to C
- ***Input:*** *3 disks*
  Output: *Disk 1 moved from A to C*
  Disk 2 moved from A to B
  Disk 1 moved from C to B
  Disk 3 moved from A to C
  Disk 1 moved from B to A
  Disk 2 moved from B to C
  Disk 1 moved from A to C

Algorithm

```
Start 1: START
Start 2: Procedure Hanoi(disk, source, dest, aux)
        IF disk == 1, THEN
        move disk from source to dest
          ELSE
         Hanoi(disk - 1, source, aux, dest)
         move disk from source to dest
         Hanoi(disk - 1, aux, dest, source)
        END IF
Start 4:    END Procedure
Start 5:    STOP
```

C implementation: TOH

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int n;
   n=3;
   hanoi(n, 'A', 'C', 'B');
   return 0;
}

void hanoi(int n, char rodFrom, char rodTo, char rodMiddle)
{
   if(n==1)
      {
      printf("Disk 1 moved from %c to %c \n",rodFrom,rodTo);
      return 0;
      }
   hanoi(n-1,rodFrom,rodMiddle,rodTo);
   printf("Disk %d moved from %c to %c \n",n,rodFrom,rodTo);
   hanoi(n-1,rodMiddle,rodTo,rodFrom);
 }
```

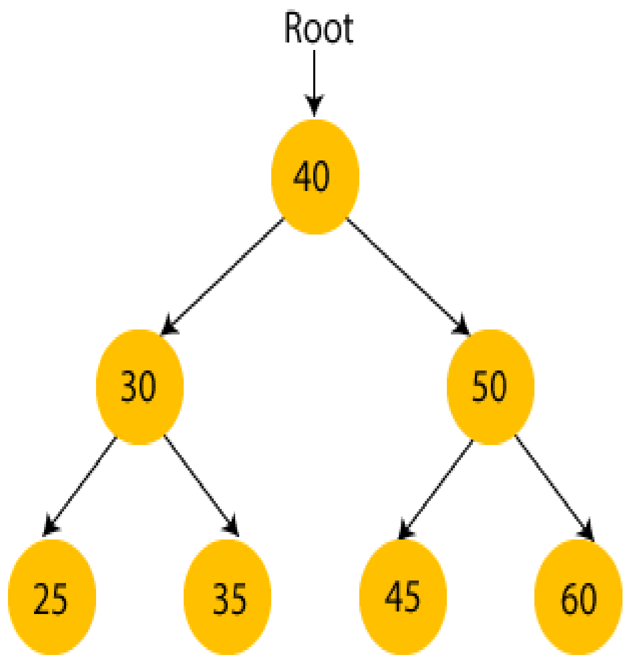
Application of Recursion

- This is the backbone of AI.
- The NP problem can’t be solved in general, but that can only be solved using recursion up to a certain extent(***not completely***) by limiting depth of recursion.
- The most important data structure ‘***Tree***’ doesn’t exist without recursion we can solve that in iterative way also but that will be a very tough task.
- Many of the well known sorting algorithms(Quick sort, Merge sort,etc) uses recursion.
- All the puzzle games(Chess, Candy crush, etc) broadly uses recursion.
- The uses of recursion is uncountable, nowadays because it is the backbone of searching, which is most important thing.

Search Tree

- A tree is a kind of data structure that is used to represent the data in hierarchical form. It can be defined as a collection of objects or entities called as nodes that are linked together to simulate a hierarchy.
- Tree is a non-linear data structure as the data in a tree is not stored linearly or sequentially.
- **A binary search tree** follows some order to arrange the elements.
  - In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node.
  - This rule is applied recursively to the left and right sub trees of the root.

Graphical Notation: Binary Search Tree



- In the above figure, we can observe that the root node is 40, and all the nodes of the left subtree are smaller than the root node, and all the nodes of the right subtree are greater than the root node.
- Similarly, we can see the left child of root node is greater than its left child and smaller than its right child. So, it also satisfies the property of binary search tree. Therefore, we can say that the tree in the above image is a binary search tree.

Advantages of Binary search tree

- Searching an element in the Binary search tree is easy as we always have a hint that which subtree has the desired element.
- As compared to array and linked lists, insertion and deletion operations are faster in BST.

Any Query?