Unit –V

TREES : Introduction to Binary Search Trees (BST), Properties of Binary Tree, Operation on BST, Traversals in Binary Trees, Heaps

Tree is a nonlinear non primitive data structure. Tree is a finite set of one or more nodes such that each node contains the address of other node. The node A is the root node.

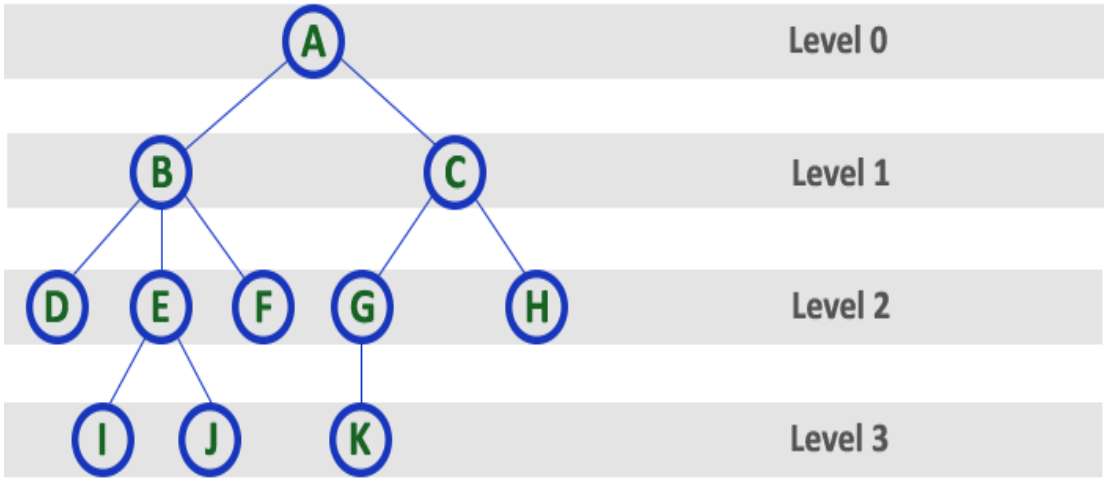


Figure 1 Tree

Terminologies:

Root: It is the unique node in the tree to which further subtree are attached, for above given tree, node A is the root node.

Degree of node: The total number of subtree connected to a node is called degree of a node. For example degree of node G is 1, degree of node C is 2 in figure 1 and degree of node B is 3 and degree of leaf node is always zero.

Level of a tree: The root node to be at level 0 and the children of the root node are at level 1 and the children of the node which are at level 1 will be at level 2 and so on.

Height of a tree: The total number of edges from leaf node to a particular node in **the longest path is** called height of that node. The height of a root node is said to be height of the tree. Figure 1 height of a tree is 3.

Leaf node: The node **which does not have a child is called as leaf node**. In figure 1 the leaf nodes are D, I, J, F, K , H.

Internal node: The node with atleast one node is called internal node. In figure 1 the internal nodes are A, B, C, E & G.

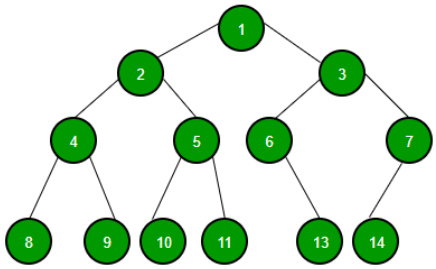
Depth: The total number of edges from root node to a particular node is called depth of that node.

Implement a Binary Tree

0/1/2 – Max Nodes

Binary Tree Data Structure

A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.



A Binary Tree node contains the following parts.

1. Data
2. Pointer to left child
3. Pointer to right child

Basic Operation On Binary Tree:

- Inserting an element.
- Removing an element.
- Searching for an element.
- Traversing an element. There are three types of traversals in a binary tree which will be discussed ahead.

Applications of Binary Tree:

- In compilers, Expression Trees are used which is an application of binary tree.
- Huffman coding trees are used in data compression algorithms.

the properties of a binary tree are discussed

1) The maximum number of nodes at level 'l' of a binary tree is 2^l.

Here level is the number of nodes on the path from the root to the node (including root and node). Level of the root is 0.

This can be proved by induction.

For root, l = 0, number of nodes = $2^0 = 1$

Assume that the maximum number of nodes on level 'l' is 2^l

Since in Binary tree every node has at most 2 children, next level would have twice nodes, i.e. $2 * 2^l$

2) The Maximum number of nodes in a binary tree of height 'h' is $2^h - 1$.

Here the height of a tree is the maximum number of nodes on the root to leaf path. Height of a tree with a single node is considered as 1.

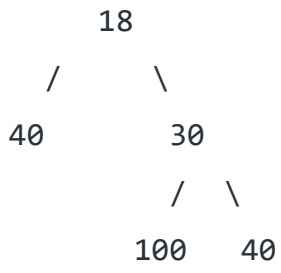
This result can be derived from point 2 above. A tree has maximum nodes if all

levels have maximum nodes. So maximum number of nodes in a binary tree of height h is $1 + 2 + 4 + .. + 2^{h-1}$. This is a simple geometric series with h terms and sum of this series is $2^h – 1$.

The following are common types of Binary Trees.

Full Binary Tree:-

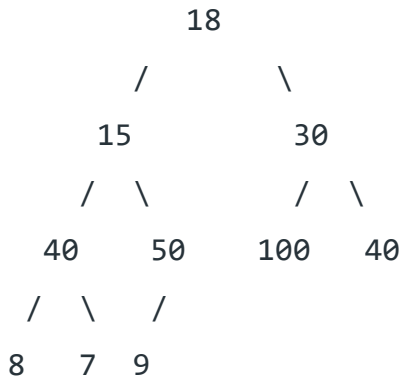
A Binary Tree is a full binary tree if every node has 0 or 2 children



Complete Binary Tree:-

A Binary Tree is a Complete Binary Tree if all the levels are completely filled except possibly the last level and the last level has all keys as left as possible. A complete binary tree is just like a full binary tree, but with two major differences:

- Every level must be completely filled
- All the leaf elements must lean towards the left.
- The last leaf element might not have a right sibling i.e. a complete binary tree doesn't have to be a full binary tree.
  The following are examples of Complete Binary Trees



Perfect Binary Tree:-

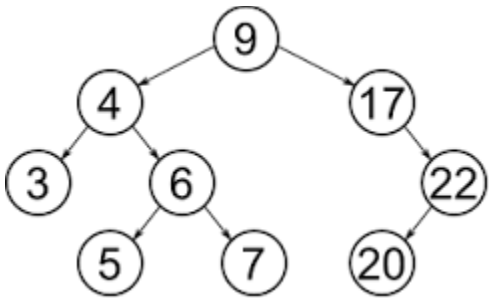
A Binary tree is a Perfect Binary Tree in which all the internal nodes have two children and all leaf nodes are at the same level.
The following are the examples of Perfect Binary Trees.

A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.

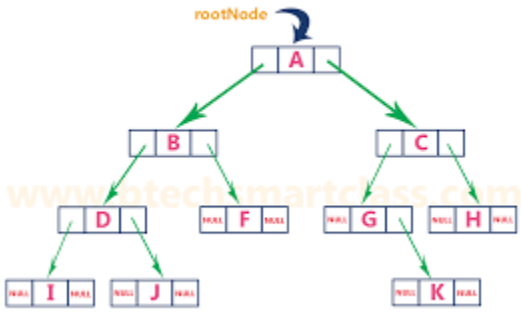
```
            18
          /    \
        15      30
       /  \     /  \
     40    50  100   40
```

Binary Search Tree (BST)

A binary tree is a finite set nodes which is either empty or consists of a root and two disjoint binary tree called left subtree and right subtree. Based on principle LST <= ROOT < RST.



Tree representation using linked list.



Complete binary tree: A binary tree T with n levels is complete if all levels except possibly the last are completely full, and the last level has all its nodes to the left side.

Full Binary Tree Theorem

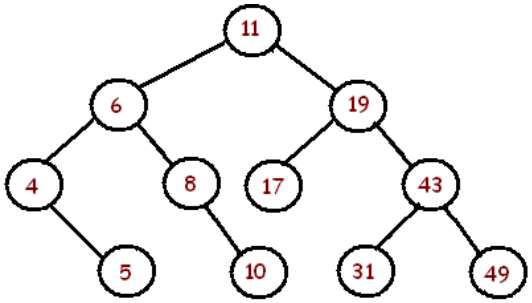
Theorem: Let T be a nonempty, full binary tree then:

1. If T has I internal nodes, the number of leaves is L = I + 1.

2. If T has I internal nodes, the total number of nodes is N = 2I + 1.
3. If T has a total of N nodes, the number of internal nodes is I = (N – 1)/2.
4. If T has a total of N nodes, the number of leaves is L = (N + 1)/2.
5. If T has L leaves, the total number of nodes is N = 2L – 1.
6. If T has L leaves, the number of internal nodes is I = L – 1.

Example 1: Given a sequence of numbers: 11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31

Draw a binary search tree by inserting the above numbers from left to right.



Function to insert a node:

1. Read the value of the node which is to be created and store it in a node called new
2. Initially if (root==NULL) then root =new
3. Again read the next value of node created in new
4. if (new->data < root_data) then connect the new node as a left child of the root otherwise the right child of a root.
5. Repeat step 3 and 4 for the construction of a complete binary tree.

```
NODE *CreateBST(NODE *root, int elem)
{
   if(root == NULL)
   {
      root=(NODE *)malloc(sizeof(NODE));
      root->left= root->right = NULL;
      root->data=elem;
      return root;
   }

   else
   {
      if( elem < root->data )
         root->left=CreateBST(root->left,elem);
      else
         if( elem > root->data )
            root->right=CreateBST(root->right,elem);
         else
            printf(" Duplicate Element !! Not Allowed !!!");
       return(root);
   }
}
```

Traversals

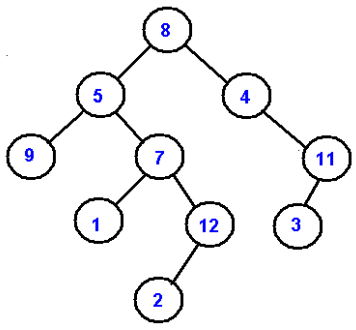
A traversal is a process that visits all the nodes in the tree. Since a tree is a nonlinear data structure, there is no unique traversal.

There are three different types of traversals

- PreOrder traversal - visit the parent first and then left and right children;
- InOrder traversal - visit the left child, then the parent and the right child;
- PostOrder traversal - visit left child, then the right child and then the parent;

As an example consider the following tree and its tree traversals:

PreOrder - 8, 5, 9, 7, 1, 12, 2, 4, 11, 3
InOrder - 9, 5, 1, 7, 2, 12, 8, 4, 3, 11
PostOrder - 9, 1, 2, 12, 7, 5, 3, 11, 4, 8



https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/?ref=lbp

Recursive inorder traversal C function

```
Void inorder(Node temp)
{
 if (temp!=NULL)
 {
        inorder(temp->lptr)
        printf("%d",temp->data);
        inorder(temp->rptr)
}
```

Recursive postorder traversal C function

```
Void postorder(Node temp)
{
 if (temp!=NULL)
 {
        postorder(temp->lptr)
        postorder(temp->rptr
        printf("%d",temp->data);
 }
}
```

Recursive preorder traversal C function

```
Void preorder(Node temp)
{
 if (temp!=NULL)
 {
        printf("%d",temp->data);
        preorder(temp->lptr)
        preorder(temp->rptr
 }
}
```

1. Design, develop and execute a program in C to create a max heap of integers by accepting one element at a time and by inserting it immediately in to the heap. Use the array representation for the heap. Finally display the heap list.

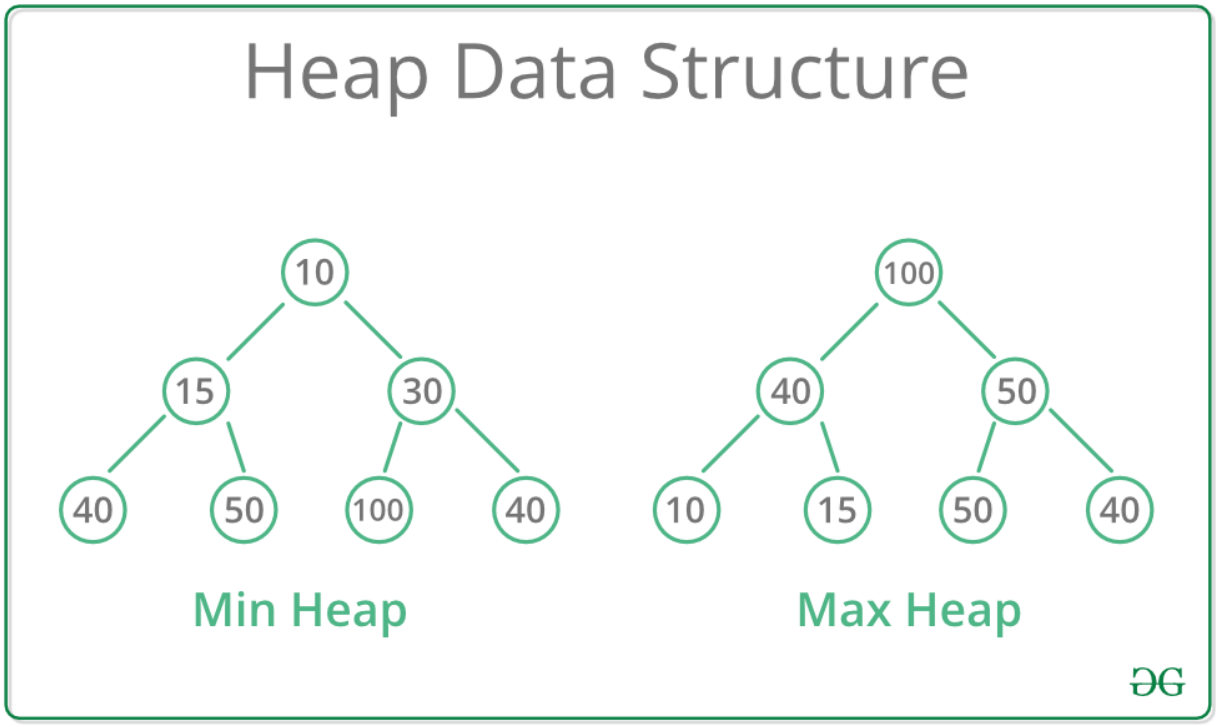
```
#include<stdio.h>
#include<time.h>
void heapify(int [],int );
void adjust(int [],int ,int);
void hsort(int [],int );
void main()
{
```

```
        int a[10], n, i;
        clrscr();
        printf("Enter the no of elements\n");
        scanf("%d",&n);
        printf("Enter the elements \n");
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        heapify(a,n);
        printf("The BST elements are \n");
        for(i=1;i<=n;i++)
        printf("%d\n",a[i]);
}

void heapify(int a[],int n)
 {
        int i;
        for(i=n/2;i>=1;i--)
        adjust(a,i,n);
 }

 void adjust(int a[], int i, int n)
 {
        int j, v, heap, k;
        k=i;
        v=a[k];
        heap=0;
        while(!heap && (2*k)<=n)
        {
                j=2*k;
                if(j<n)
                {
                        if(a[j]<a[j+1])
                        j=j+1;
                }
                if(v>=a[j])
                heap=1;
                else
                {
                        a[k]=a[j];
                        k=j;
                }
        }
        a[k]=v;
 }
```

Heap

Heap is a special Tree-based data structure in which the tree is a complete binary tree. Generally, Heaps can be of two types:

1. **Max-Heap**: In a Max-Heap the key present at the root node must be greatest among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.
2. **Min-Heap**: In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.



Min heap constructions
https://levelup.gitconnected.com/constructing-min-heap-from-an-array-1119347486c5

2. Design, develop and execute a program in C to create a Binary tree and perform the following operation inorder, preorder and postorder traversals.

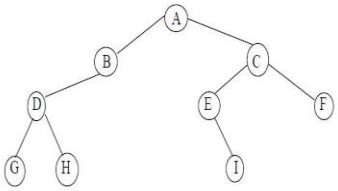
```
#include <stdlib.h>
typedef struct tnode
{
   int data;
   struct tnode *right,*left;
}TNODE;

TNODE *CreateBST(TNODE *, int);
void Inorder(TNODE *);
void Preorder(TNODE *);
void Postorder(TNODE *);
main()
{
   TNODE *root=NULL;                    /* Main Program */
   int opn,elem,n,i;
   do
   {
      clrscr();
      printf("\n Binary Search Tree Operations \n\n");
      printf("\n Press 1-Creation of BST");
      printf("\n      2-Traverse in Inorder");
      printf("\n      3-Traverse in Preorder");
      printf("\n      4-Traverse in Postorder");
      printf("\n      5-Exit\n");
      printf("\n      Your option ? ");
      scanf("%d",&opn);
      switch(opn)
      {
            case 1: root=NULL;
            printf("\n\nBST for How Many Nodes ?");
            scanf("%d",&n);
            for(i=1;i<=n;i++)
            {
                        printf("\nRead the Data for Node %d ?",i);
                        scanf("%d",&elem);
                        root=CreateBST(root,elem);
             }
             printf("\nBST with %d nodes is ready to Use!!\n",n); break;
            case 2: printf("\n BST Traversal in INORDER \n");
                     Inorder(root); break;
            case 3: printf("\n BST Traversal in PREORDER \n");
                      Preorder(root); break;
            case 4: printf("\n BST Traversal in POSTORDER \n");
                     Postorder(root); break;
            case 5: printf("\n\n Terminating \n\n"); break;
```

```
            default: printf("\n\nInvalid Option !!! Try Again !! \n\n");          break;
        }
        printf("\n\n\n\n  Press a Key to Continue . . . ");
        getch();
    }while(opn != 5);
}

TNODE *CreateBST(TNODE *root, int elem)
{
    if(root == NULL)
    {
        root=(TNODE *)malloc(sizeof(TNODE));
        root->left= root->right = NULL;
        root->data=elem;
        return root;
    }
    else
    {
        if( elem < root->data )
            root->left=CreateBST(root->left,elem);
        else
            if( elem > root->data )
                root->right=CreateBST(root->right,elem);
            else
                printf(" Duplicate Element !! Not Allowed !!!");

        return(root);
    }
}
void Inorder(TNODE *root)
{
    if( root != NULL)
    {
        Inorder(root->left);
        printf(" %d ",root->data);
        Inorder(root->right);
    }
}

void Preorder(TNODE *root)
{
    if( root != NULL)
    {
        printf(" %d ",root->data);
        Preorder(root->left);
        Preorder(root->right);
    }
}
```

```
void Postorder(TNODE *root)
{
   if( root != NULL)
   {
      Postorder(root->left);
      Postorder(root->right);
      printf(" %d ",root->data);
   }
}
```

Question Bank

1. Define the following with an example
    i. Binary tree
    ii. Complete binary tree
    iii. Almost complete binary tree
    iv. Binary search tree
    v. Depth of a tree
2. Given the following graph, write the inorder, preorder and postorder traversals.



3. In brief describe any 4 applications of trees.
4. Construct a binary tree from the traversal order given below:
    i. PREORDER = A B D E F C G H L J K
    ii. INORDER = D B F E A G C L J H K
5. Construct a binary tree for: ((6 + (3 - 2) * 5) ^2 + 3).
6. Write c function for the following tree traversals:
    i. Inorder ii) Preorder iii) Postorder
7. What is a tree? Explain
    i. Root tree
    ii. Degree

iii. Sibling

iv. Depth of a tree and give example

8. What is a binary tree? State the properties? How it is represented using array and linked list give example.
9. Define max heap? Write a C function to insert an item into max heap.
10. For any non empty binary tree T if no is the number of leaf nodes and n2 the number of nodes of degree 2. Then prove that n0=n2+1.
11. Explain the different tree traversal methods.
12. What is binary tree? Explain the different operations performed on binary tree.
13. Explain the different schemes of representation of binary tree.
14. Define the following with a suitable example.

    i) Binary tree

    ii) Degree of a binary tree

    iii) Level of a binary tree

    iv) Sibling.
15. Explain the following with an example:

    i. i) Forest ii) graph iii) winner tree.
16. Describe the binary search tree with an example. Write an iterative function to search for a key value in a binary search tree.
17. Explain the following with an example

    i. Selection trees

    ii. Forest tree and its traversals
18. Describe the binary search tree with an example. Write a recursive function to search for a key value in a binary search tree.
19. Construct an inorder tree for the following elements. 66, 56, 12, 34, 78, 98, 12,-56, 4, 18.
20. Explain selection trees, with suitable example
21. What is a forest? With a suitable example illustrate how you transform a forest into a binary tree.