-02-H

# Trees and Graphs

→ It is a non-linear data structure

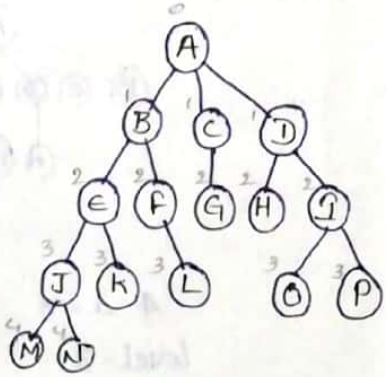
Tree - collection of nodes and edges.

* It is a finite non-empty set of elements.
* It is an abstract model of a hierarchial structure.

Applications:-

→ Organisation charts
→ File Systems
→ Programming environments

Tree terminology:- Root, siblings, internal node, external node, ancestors




root - node without a parent Eg: A

siblings - nodes shares the same parent

Eg: siblings for B is C, D

Internal node - a node with atleast one children

Eg: A, B, C, D, E, F, J, Q

external node - a node without children.

Eg: G, H, Q, J, K, L, O, P

ancestors - parent, grandparent, grand-grand parents ---

Eg: ancestors for F - B, A
K - E, B, A

Discendent - child, grand child, grand-grand child ---

Eg: discendent of a node C - G
B - E, F, J, K, L, M, N

Depth of a node - no. of ancestors

Eg: Depth of Q - 2
K - 3
C - 1
A - 0
J - 3

Height of a node - no. of nodes which are present in longest children path of a node.

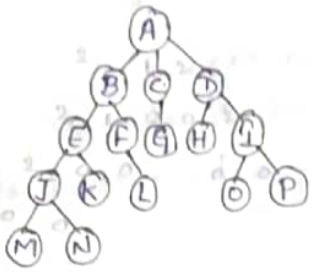
Height of a tree - maximum depth of any node.

| | M | N | K | L | G | H | O | P |
|---|---|---|---|---|---|---|---|---|
| Eg: | 4 | 4 | 3 | 3 | 2 | 2 | 3 | 3 |

Height of a tree = 4

Degree of a node - no. of it's children

A - 3, B - 2, C - 1, D - 2, E - 2, F - 1, G - 0, H - 0,
I - 2, J - 2, K, L, M, N, O, P - 0

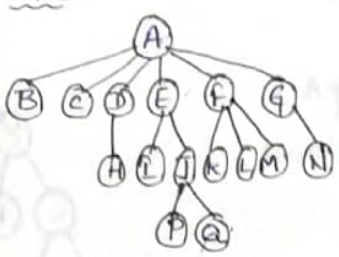



Degree of a tree - the max degree of any node in a tree

A = 3

degree of tree = 3

Level of a tree :-




level

0

1

2

3

4 - 1 = 3

level = 3

Root = A

Leaves = B, C, H, I, P, Q, K, L, M, N

Internal nodes = A, D, E, F, J, G

Children of A = B, C, D, E, F, G

" " D = H

" " E = I, J

" " F = K, L, M

" " G = N

Degree of E = 2

Depth of J = 2

P = 3

Depth of tree = 4

Height of tree = 3

Height of E = 2

" " A = 3

discendants of A = B, C, D, E, F, G, H, I, J, K, L, M, N, P, Q

" " E = I, J, P, Q

Siblings = {B, C, D, E, F, G} {I, J} {K, L, M} {P, Q}

ancestors of H = A, D

" " P = A, E, J

" " K = A, F

If root 'A' is removed then the tree become a forest

{{B}, {C}, {D, H}, {E, I, J, P, Q}, {F, K, L, M}, {G, N}}

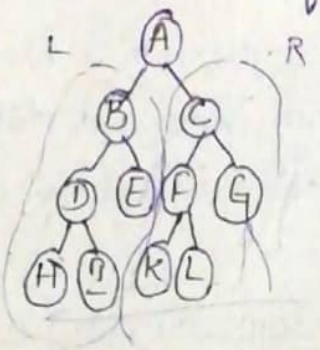
Degree of a tree = 6 [no of children of a node 'A']

## Binary tree / Dyadic tree :-

i) A binary tree is a tree with at most two children for each node. (or)

2) A binary tree either
   (i) is empty (no nodes) or
   (ii) has a root node, a left binary tree, and a right binary tree.

→ Each node in a binary tree has exactly two subtrees (one or both of these subtrees may be empty).




## Difference between tree and binary tree :-

→ A tree can never be empty but binary tree may be empty.

6-02-14 → A node in a binary tree may have at most two children, whereas a node in a tree may have any no. of children.

→ The sub trees of each node in a binary tree are ordered; i.e., we distinguish between the left and right sub-trees. The sub trees in a tree are unordered.

## Binary tree properties :-

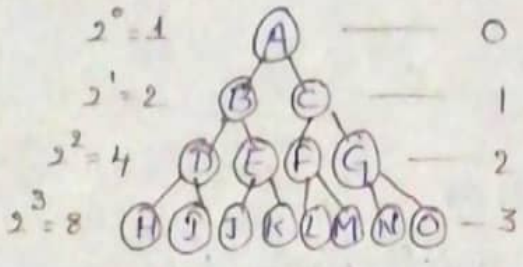
→ The maximum no. of nodes at level i is $2^i$.

→ A binary tree of height h ($h \geq 0$) has atleast $h+1$ and at most $2^{h+1} - 1$ nodes in it.

→ Every binary tree with n nodes ($n \geq 0$) has exactly

$n-1$ edges.

→ The height of the binary tree containing n ($n \geq 0$) nodes is at most $n-1$ and at least $\lceil \log(n+1) - 1 \rceil$.




Example of point 1

Height of a tree = no. of edges which are in longest path of a tree.

Representation of a binary tree :-

→ A binary tree must represent a hierarchical relationship between a parent node and child nodes.

→ There are 2 common methods used to represent a binary tree

* Linear / Sequential : representation using arrays
* Linked : representation using linked list.

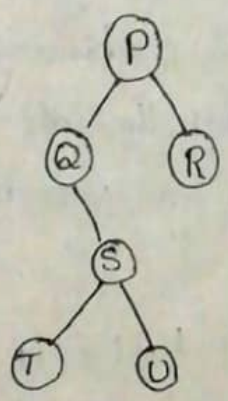
$= 2^{n+1} - 1$

$= 2^{3+1} - 1$

$= 2^4 - 1$

$= 16 - 1$

$= 15$




| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | Q | R | | S | | | | | T | U | | | | |

17.02.14

Advantages of array representation:

→ Any node can be accessed randomly by calculating the right index.

→ Programming languages such as BASIC, FORTRAN does not have dynamic memory allocation statistics.

→ In these languages, arrays are the only one method to store trees.

Disadvantages:-

→ Otherthan the binary tree the max. elements majority of the array of tree causing wastage of memory.

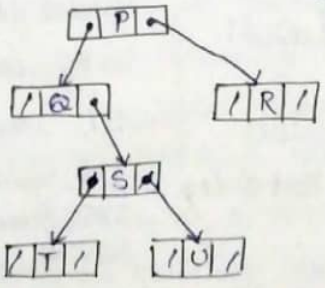
→ It is no way possible to enhance the tree structure if the array size is limited.

→ Inserting & deleting operations are inefficient in array representation

Linked Representation:-

→ THIS TYPE of representation is efficient to represent a binary tree over arrays

→ In this structure Every node consisting of 2 parts one for data and other to store address of left chain and another to store address of right chain

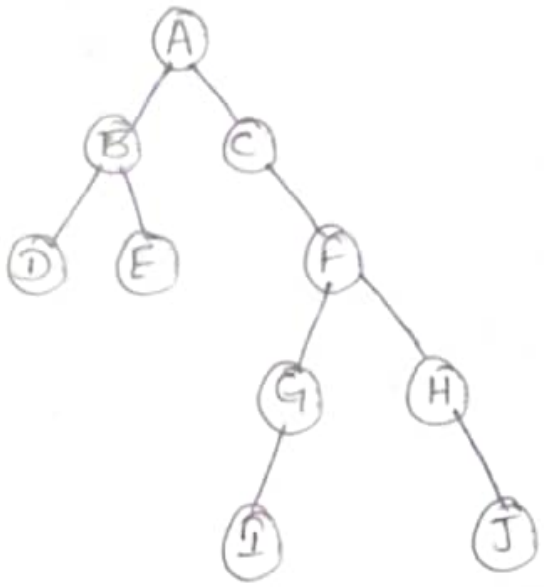



Operations on binary tree

→ Inserting a node into a tree

→ deleting a node from a tree

(A+B) * (C-D-E) / F - G




Pre-order

Inorder :- DBEACIGFHJ

Post Order :-

ALGORITHM FOR TRANSVERSAL OF BINARY TREE :-

BT_Recursive Inorder

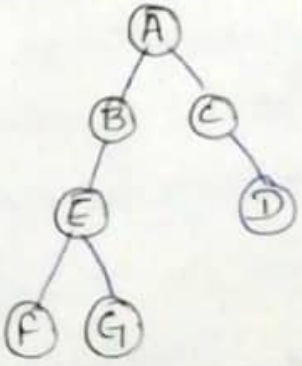
// input is root node of a binary
// t - a node in the binary tree
// lchild - left child of a node
// rchild - right child of a node

i) BT_recursive (t → lchild)
ii) visit (t) [: root]
iii) BT_recursive (t → Rchild)

BT_Recursive Preorder →

i) visit (t)
ii) BT_recursive (t→lchild)
iii) BT_recursive (t→Rchild)

Post Order :-
i) BT_recursive (t→lchild)
ii) BT_recursive (t→Rchild)
iii) visit (t)

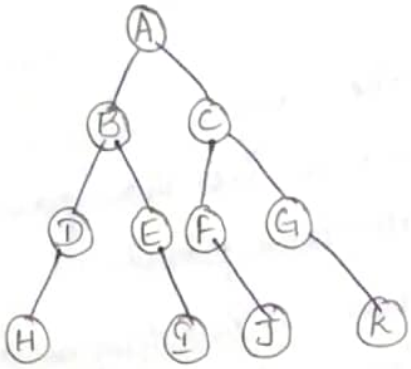


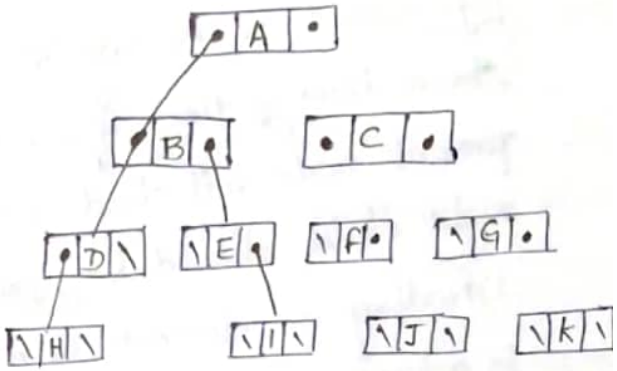

| L | Root | R |
|---|---|---|

FGEBDCA - Post order
FEGBACD - Inorder
ABEFGCD - Pre Order

EADED BINARY TREE :- If a binary tree is having n' of nodes then there must be a n+1 no of null nters.

re we are wasting the space to represent null inters.




Null pointers = n+1 = 11+1 = 12

Inorder: HDBEIAJFCGK

17

Expression Trees :-

) ((2*4)-3)+5

postfix

2

2 4

2 4 *

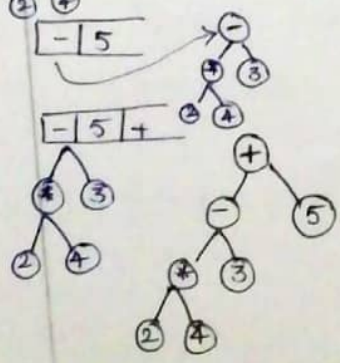
* 3

2 4

* 3 -

- 5

- 5 +



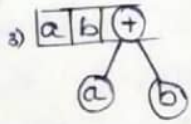
2) ((a+b)>=c) && (d>e)

ab+c>=de>&&

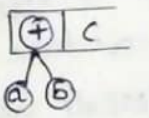
postfix

1) a 2) a b

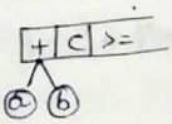
3) a b +



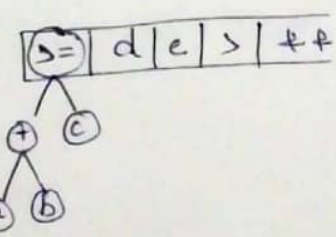
+ c

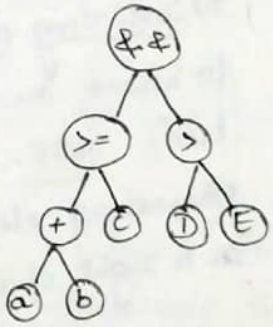


+ c >=

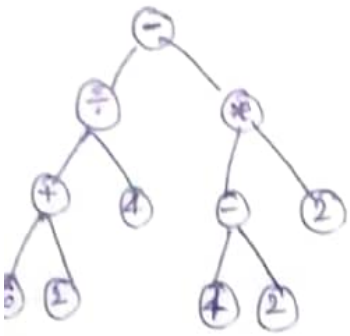


>= d e > &&





(6+2)÷4) - (4-2)*2



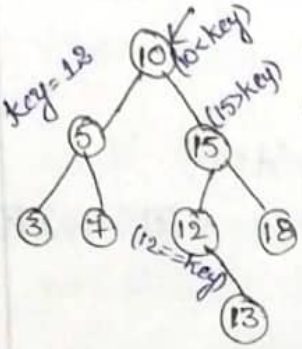
Operations on binary tree :-

In a binary tree if we take any parent node left child should be less than parent & right child should be greater than parent.

Root
Small Big

1) finding an element in a tree 2) Inserting an element in a tree 3) deleting element from a tree.

1) finding an element in a tree :- key element is compared with root node if(key == root)




else if (key > root) find in right child of root
else if (key < root) find in left child of root

2) Inserting an element in a tree :- There are 3 cases to insert an element into a tree i) root is null (empty tree) NOTE: INSERTING ELEMENT AS A ROOT NODE ii) inserting element as a left child iii) Inserting element as a right child.

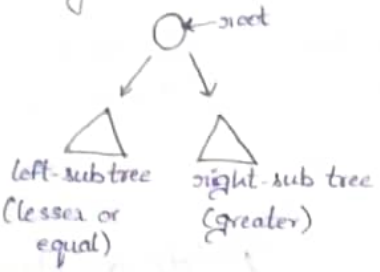
09-03-17



## Applications of Queues:-

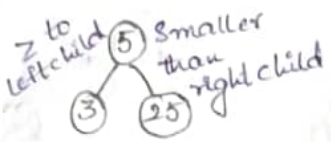
01-03-17

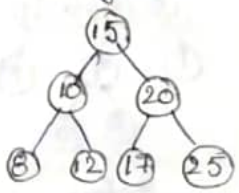
**Binary Search tree:-** A binary tree in which for each node, value of all the nodes in left sub-tree is lesser or equal and value of all the nodes in right sub-tree is greater.




Binary search tree is a binary tree which follows the below condition:

→ if you take any node in the binary tree it should be ≥ to it's left child and < right child




**Operations on binary search tree:-**

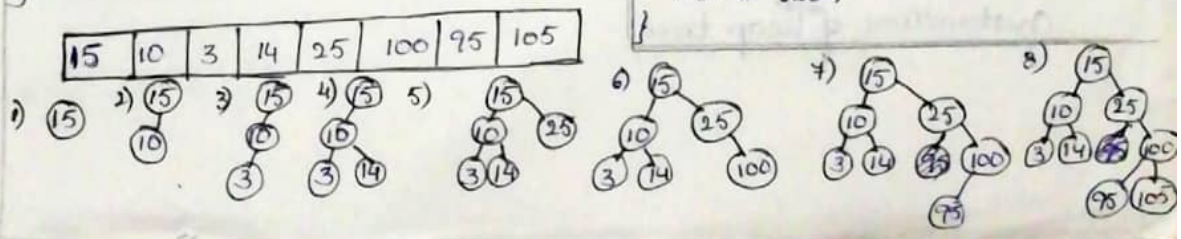
1) Searching an element 2) Inserting an element
3) Deleting an element

**Searching an element in binary Search tree:**

```
bool search(BST node * root, int data)
{
 if(root==NULL)
 {
  return false;
 }
  else if(root->data==data)
  {
   return true;
  }
   else if(data<=root->data)
   {
    return search(root->left, data);
   }
    else
     {
      return search(root->right, data);
     }
}
```

**Insertion Operation:-**

```
//To insert data in BST, returns address of root node
BstNode* Insert(BstNode *root, int data)
{
 if(root==NULL)// empty tree
 {
  root = GetNewNode(data);
 }
 //if data to be inserted is lesser,
                  insert in left sub-tree.
 else if(data<=root->data)
 {
  root->left = Insert(root->left, data);
 }
  //else, insert in right sub tree.
  else{
       root->right=Insert(root->right,
                          data);
      }
  return root;
}
```



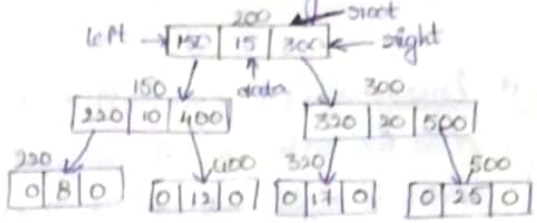
Delete an element:-

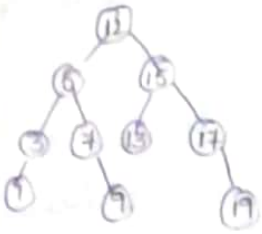
There are three situations where we are deleting an element

i) If a node is terminal node

ii) " " " having 1 child

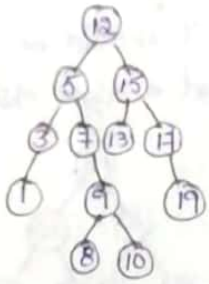
iii) " " " " 2 children



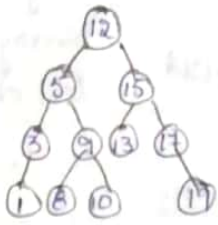



delete 19 In side tree if you make node 17 right link as null then 19 will be detached from 17

Delete: 7

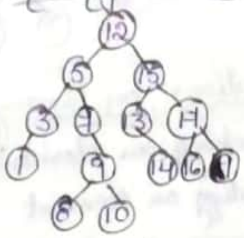


Before



After

Delete an element which is having two children:-



There are 2 methods

Method 1: Find min. node in right sub-tree

Method 2: Find max. node in left sub-tree

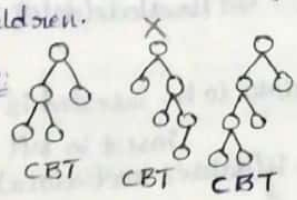
Method 1:- Find min in right subtree

i) identify the minimum elements in its right sub tree. ii) Copy the identified value in targeted node iii) Delete duplicate from right sub-tree

02-03-17 Complete BT:- A CBT is a BT which must follow below condition

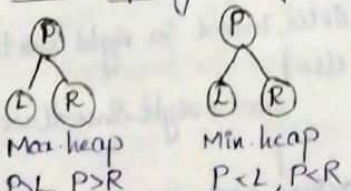
* Should → each & every node in the tree must have two children or zero children.

Eg:




Heap trees - A heap tree is a CBT. It should follow order property.

Order Property:- 2 types of order property → Maximum heap order, Minimum heap order

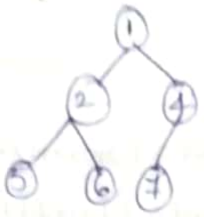



Construction of max heap trees:-

Day 3.1

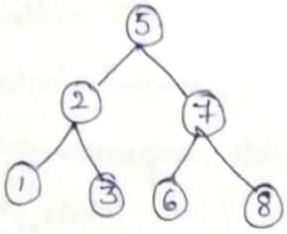
## Complete Binary Tree

A binary tree that is completely filled with the exception of the lowest level that is filled from left to right




## Binary Search Tree

- Nodes to the left are less than the root node while nodes to the right are greater than the root node.
- The BST is used in Searching & Sorting techniques

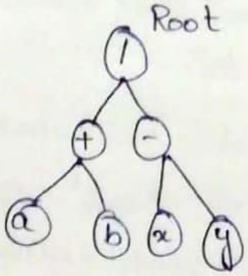



## Expression Tree

Used to evaluate simple arithmetic expression.

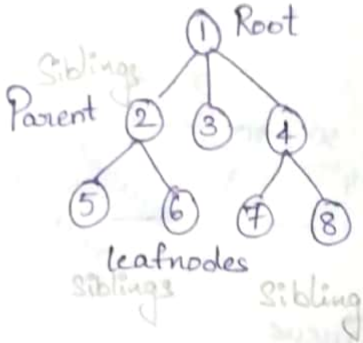
Expression trees are mainly used to solve algebraic expressions.

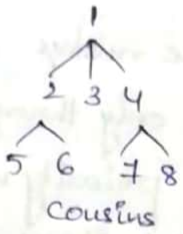
$(a+b)/(x-y)$



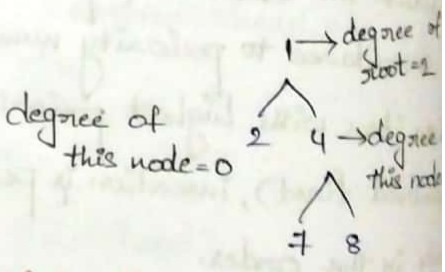

## Trees

- Non-linear Hierarchical Data
- Collection of nodes Connected by
- One of the node is designated root node & remaining — child leaf nodes of the root node.
- Used in many applications & also field.

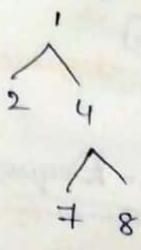



## Degree of the tree:-



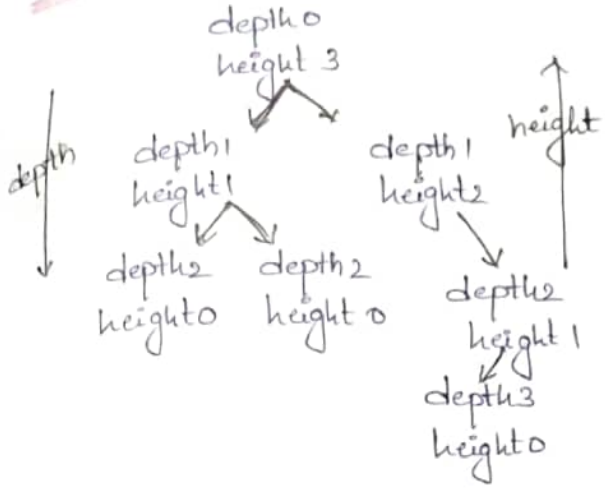

## Path of the tree:-



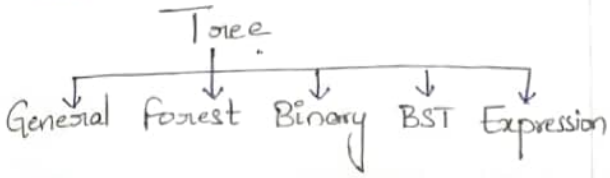

what will be the path to reach node with key value 8 &

1 → 4 → 8

② Height & depth of the tree

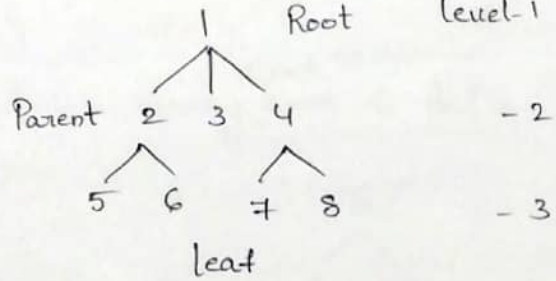



Tree - Types

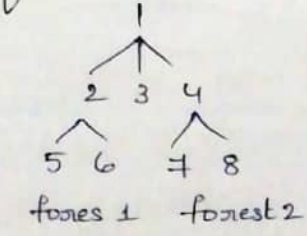



1) General Trees

- Basic representation of a tree
- has node & 1 or more childnodes.
- The root node is present at level 1 & all other nodes may be present at various levels.



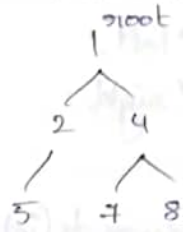

2) forest - a kind of tree obtained while deleting the root or parent node from the tree.

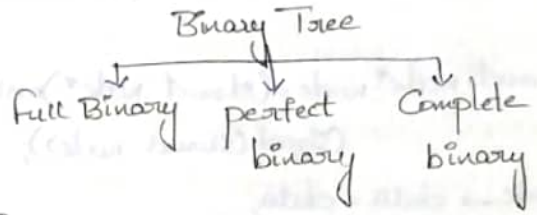



③ Dyadic tree

3) Binary Tree - in which each node has atmost 2 child nodes.

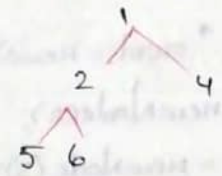
- Used in a range of applications like expression evaluation, databases ..



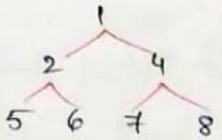

Full Binary Tree:-

* each node has exactly zero or 2 children.
* Every node other than leaf nodes has 2 child nodes.



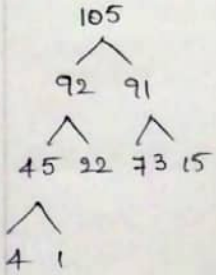

Perfect Binary Tree

All nodes have 2 children & all leaves are at same level.

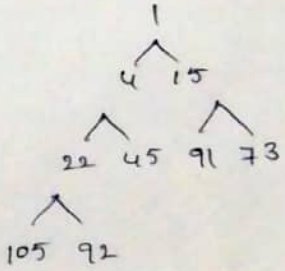



4, 92, 73, 45, 22, 91, 15, 105, 1

Maxheap



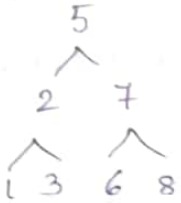

in Maxheap



# Day 3.2

## BST

- Nodes to the left are less than the root node while the nodes to the right are greater than the root node.
- Used in Searching & Sorting techniques.




## Binary Search Tree - Insertion

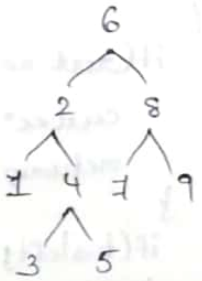
Traverse using in-order traversal.

Eg:

I/p:

6 2 8 7 1 4 5 3 9

o/p:

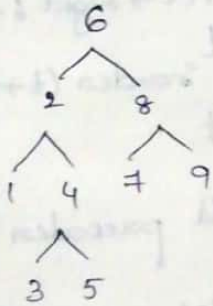



## Binary Search Tree - Searching

Given a binary search tree & find a value in it.

I/p: 4

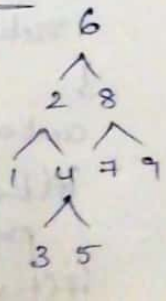
o/p: found




## Binary Search Tree - Deletion.

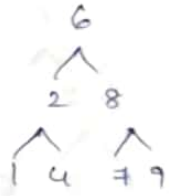
Given a BST, find & delete the given value from it.

Eg:




Case 1 : delete a leaf node

I/p: 3

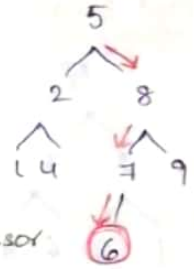



Case 2: Delete a node with 1 child

I/p: 4

Case 3: delete a node with 2 children

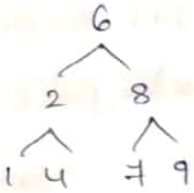
I/p: 5



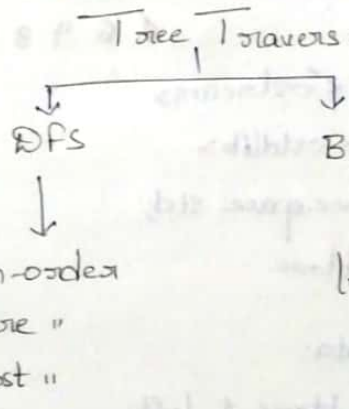

Step 1: find in-order successor of 5

Step 2: Replace 5 with in-order Successor

Step 3: Delete in-order Successor




## Tree - Traversal Techniques




In-order - Left Root Right

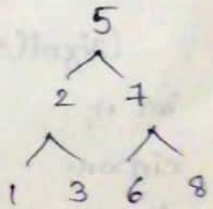
Pre order - Root left Right

Post " - left Right root

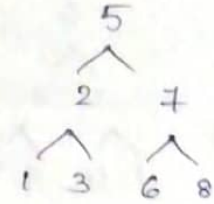
level " - level wise from left to right

Inorder

1 2 3 5 6 7 8

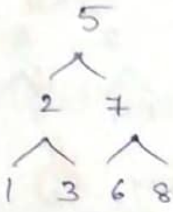


## Pre-order




5 2 1 3 7 6 8

## Post order




1 3 2 6 8 7 5

## Program to Create & implement a BST Using DFS to display the elements (in-order, pre & post-order)

**Sample i/p:**

5
5 8 9 4 6

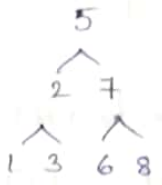
**Sample o/p:**

4 5 6 8 9
5 4 8 6 9
4 6 9 8 5

```
#include<iostream>
#include<cstdlib>
using namespace std;
struct btree
{
  int data;
  struct btree * left;
  struct btree * right;
};
struct btree * root * temp;
void Create()
{
  temp=(struct btree *) malloc
      (sizeof(struct btree));
  int n;
  cin>>n;
  temp->data=n;
  temp->right = temp->left
}
void insert(struct btree *t)
{
 if(t->data < temp->data &&
     t->right != NULL){insert(t->right
}
else if(t->data<temp->data && t->right
     NULL){t->right = temp;
}
else if(t->data>temp->data && t->left!=
     {insert(t->left);
}
else if(t->data > temp->data && t->left
     NULL){t->left=temp;
 }
}
 void inorder(struct btree *t)
{
  if(root == NULL){
    cout<<"No element";
    return;
  }
  if(t->left!=NULL)
  {
    inorder(t->left);
  }
  cout<<t->data<<" ";
  if(t->right != NULL)
  {
    inorder(t->right);
  }
}
Void preorder(struct btree *t)
{
  if(root == NULL)
  {
    cout<<"No element";
    return;
  }
  cout<<t->data<<" ";
  if(t->left!=NULL)
    preorder(t->left);
  if(t->right!=NULL)
```

```
    preorder(t->right);
}
void postorder(struct btree *t)
{
 if(root==NULL)
 {
  cout<<"No element";
  return;
 }
 if(t->left!=NULL)
    postorder(t->left);
 if(t->right!=NULL)
    postorder(t->right);
  cout<<t->data<<" ";
int main()
{
 int n;
 cin>>n;
 root=0;
 for(int i=0; i<n; i++)
 {
  create();
  if(root==NULL)
    root=temp;
  else
    insert(root);
  }
 inorder(root);
 cout<<endl;
 preorder(root);
 cout<<endl;
 postorder(root);
}
```

## Level Order Traversal



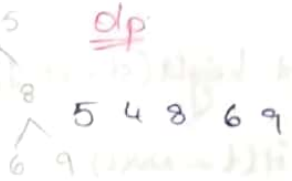

5 2 7 1 3 6 8

Write a program to create and implement a BST structure. Use BFS to display the elements (level order)

ilp

5

5 8 9 4 6




dp

5 4 8 6 9

```
#include<iostream>
#include<cstdlib>
using namespace std;
struct btree
{
 int data;
struct btree *left;
struct btree *right;
};
struct btree *root, *temp;
void create()
{
 temp=(struct btree*)malloc(sizeof
                (struct btree));
 int n;
 cin>>n;
 temp->data=n;
 temp->right=temp->left=NULL;
}
void insert(struct btree *t)
{
 if(t->data<temp->data &&
    t->right!=NULL){
        insert(t->right);
  }
else if(t->data<temp->data &&
```

```
        t->right == NULL){
            t->right = temp;
    }
    else if(t->data > temp->data && t->left!=
            NULL){insert(t->left);
    }
    else if(t->data>temp->data &&
            t->left == NULL){t->left=temp;
    }
}
int height(struct btree *t)
{
  if(t ==NULL)
  {
    return 0;
  }
  else
  {
    int lheight = height(t->left);
    int rheight = height(t->right);
    if(lheight>rheight)
    {
      return lheight+1
    }
    else
    {
      return rheight+1;
    }
  }
}
void printgivenlevel(struct btree *t,
                     int i)
{
  if (t== NULL)
  {
    return;
  }
  if(i==1)
  {
    cout<< t->data<<" ";
  }
  else if (i>1)
  {
    printgivenlevel(t->left, i-1);
    printgivenlevel(t->right, i-1);
  }
}
void levelorder(struct
{
  int h = height(t);
  int i;
  for(i=1; i<=h; i++)
  {
    printgivenlevel(t,i);
  }
}
int main()
{
  int n;
  cin>>n;
  for(int i=0; i<n; i++)
  {
    create();
    if(root == NULL)
    {
      root = temp;
    }
    else
    {
      insert(root);
    }
    levelorder(root);
}
```