**TREE**  
Tree is a nonlinear data structure and it is represented in hierarchical form.   
Tree is connected graph with n vertices and n-1 edges.  
depth-length of the path to that node.  
max depth in the tree is the height of the tree.  
A tree is a binary tree if and only it is having at most two children.  
Types of binary trees:  
A binary tree is strict tree if and only if one node is having 0 or 1 node.  
A binary tree is full if and only if it is strict and all its leaf should be at last level.  
A complete binary tree is a special type of binary tree where all the levels of the tree are filled completely except the lowest level nodes which are filled from as left as possible.

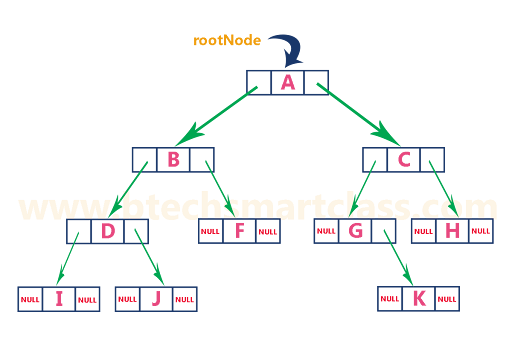
number of nodes in a full binary tree of height h is 2h+1-1  
number of nodes in complete binary tree is 2h to 2h+1 -1  
number of nodes in a full binary tree of height h is 2h are leaves and 2h-1 is non leaves.  
In any binary tree with n nodes will have n+1 null pointers  
no.of binary structures that we can create using n nodes is Catalan N  
   
 CN= **2NCN** N+1no of valid paranthesis sequences that we can draw with n pairs of open and closed paranthesis are Catalan N.  
  
**STRUCTURE OF A BINARY TREE**if a node is in ith  location   
 left =2\*i+1  
 right=2\*i+2  
 Size of the array =2h+1−1

Representation of trees  
\*Representation through 2d matrix

|  |  |
| --- | --- |
| 3 | 4 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

Node Left Right

\*Representation through double linked list  
  


BINARY TREE TRAVERSAL  
THERE ARE 3 TYPES OF TRAVERSALS   
 1)in order  
 2)pre order  
 3)post order

INORDER  
traverse the left subtree in inorder  
visit the root   
traverse the right subtree in inorder  
  
void inorder(struct node\*root)  
{  
 if(root)  
{  
 inorder(root->left)  
 printf(“%d”,root->data)  
 inorder(root->right)  
}  
}

Preorder  
void preorder(struct node\*root)  
{  
 if(root)  
{  
 printf(“%d”,root->data)  
 preorder(root->left)  
 preorder(root->right)  
}  
}

Post order  
void postorder(struct node\*root)  
{  
 if(root)  
{  
 postorder(root->left)  
 postorder(root->right)  
 printf(“%d”,root->data)  
}  
}

2d matrix code  
   
int tree[1005][2];  
 int main()  
{  
 int n;  
 cin>>n;  
for(int i=0;i<n-1;i++)  
 int p,c;  
 char x;  
 if(x==’L’) tree[p][0]=c;  
else tree[p][1]=c;  
}

INVERTED TREE   
if(root!=null)  
swap(root->left,root->right);  
invertTree(root->left);  
invertTree(root->right);  
}  
return root;  
}

SYMMETRIC TREE  
bool ismirror(treenode\*p,treenode\*q)  
{  
if(p==null && q==null)  
return 1;  
if(p==null or q==null)return 0;  
return p->val==q->val && ismirror(p->left,q->right) && ismirror(p->right,q->left);  
}

Bool issymmetric(treenode\*root)  
{  
if(root==null)return ntrue;  
return ismirror(root->left,root->right);

Int arrangecoins(int n)  
{  
long sum=0,int i=1,c=0;  
 while(sum<n)  
(  
 while(sum<n){  
 sum+=i;  
 i++;c++;  
}  
if(sum==n)  
return c;  
return c-1;  
}

Int islandperimeter(vector<vector<int>&grid)  
{ int n=grid.size(),m=grid[0].size;  
int c=0;  
for(int i=0;i<n;i++){  
for(int j=0;j<m;j++){  
if(grid[i][j]==1){  
if(i==0||grid[i-1][j]==0)c++;  
if(j==m-1||grid[i][j+1]==0)c++;  
if(i==n-1||grid[i+1][j]==0)c++;  
if(j==0||grid[i][j-1]==0)c++;  
}}}  
return c;  
}

Counting number of nodes  
int count(treenode\*root){  
if(root==null)return 0;  
else return 1+count(root->left)+count(root->right);  
}  
  
counting full nodes

Int count(treenode\*root){  
if(root==null)return 0;  
if(count->right==null && count->left==null)return 0;  
if(root->left==null)return count(root->right)  
if(root->right==null)return count(root->left);  
return1+count(root->left)+count(root->right);

Treenode\*insertintobst(treenode\*root,int val)  
{  
treenode \*nn=new treenode(val);  
i(root==null){  
root==nn;  
else{  
treenode\*t=root;  
while(1)  
{

If(t->val>=val)  
if(t->left==null){t->left=nn;break;}  
else t=t->left;  
}  
else{  
if(t->right==null){t->right=nn;break;}  
Else t=t->right;  
}}}  
return root;  
}  
};  
  
recursion  
Treenode\*insertintobst(treenode\*root,int val)  
{  
if(root==null)return new treenode(val);  
else if(root->val=val)root->left=insertintobst(root->left,val)  
else root->right=insertintobst(root->right,val)  
return root;  
}

Delete in bst;

Treenode\* findmax(tree node\*root)  
{  
treenode\*t=root;  
while(t->right)t=t->right;  
return t;  
}

Treenode\* deletenode(treenode\*root,int val)  
if(root==null)return root;  
if(root->val>key)root->left=deletenode(root->right,k)  
else if(root->val<key)root->right=deletenode(root->right,key);  
else{  
if(root->left&&root->right){  
tree node\*x=fondmax(root->left);

Swap(root->val,x->val)  
root->left=deletenode(root->left,key)  
}  
else  
{  
if(root->left!=null)root=root->left;

Else root=root->right;  
}}  
return root;  
}  
};

Treenode\*lowestcommon(treenode\* root,tree node\*p,tree node\*q)

If (root->val>p->val&&root->val>q->val)  
 return lowestcommon(root->left,p,q)  
If (root->val<p->val&&root->val<q->val)  
 return lowestcommon(root->right,p,q)  
return root;