EASY :

1. Symmetric Binary Tree

```
int is_sym(TreeNode *root1,TreeNode *root2)
 {
     if(root1==NULL && root2==NULL)
     return 1;
     if(root1==NULL && root2!=NULL)
     return 0;
     if(root1!=NULL && root2==NULL)
     return 0;
     if(root1->val!=root2->val)
     return 0;
     return is_sym(root1->left,root2->right&&
is_sym(root1->right,root2->left);
}
int Solution::isSymmetric(TreeNode* A)
{
    return is_sym(A,A);

}
```

2. Diameter of binary tree

```
class Solution {
public:
   int ans=0;
   int height(TreeNode *root)
   {
      if(root==NULL)
      return 0;
      return 1+max(height(root->left),height(root->right));
   }
   int diameterOfBinaryTree(TreeNode* root)
   {
      if(root==NULL)
      return 0;
      int left=height(root->left);
      int right=height(root->right);
      ans=max(ans,left+right);
      diameterOfBinaryTree(root->left);
      diameterOfBinaryTree(root->right);
      return ans;
   }
};
```

3. Invert Binary tree

```
class Solution {
public:
   TreeNode* invertTree(TreeNode* root)
   {
      if(!root)
      return NULL;
      swap(root->right,root->left);
      invertTree(root->left);
      invertTree(root->right);
      return root;
   }
};
```

4. Check Binary tree is balanced or not

```
int height(TreeNode* root){
    if(root==NULL)
    return 0;
    return max(height(root->left), height(root->right))+1;
}
int Solution::isBalanced(TreeNode *A)
{
    if(A == NULL)
    return 1;
    int l=height(A->left);
    int r=height(A->right);
    if(abs(l-r)<2 && isBalanced(A->left) &&
isBalanced(A->right))
    return 1;
    return 0;
}
```

5. Transform to sum tree

```
class Solution {
  public:
    int solve(Node *node){
      if(node==NULL)
      return 0;
      int a = solve(node->left);
      int b = solve(node->right);
      int x=node->data;
      node->data=a+b;
      return  a+b+x;
   }
    void toSumTree(Node *node)
    {
```

```
        solve(node);
    }
};
```

6. Leaf are at same level

```
class Solution{
  public:
    void solve(Node *root,int h,int &maxx,bool &f)
    {
       if(root==NULL)
       return;
       if(f==0)
       return;
       if(root->left==0 && root->right==0)
       {
           if(maxx==-1)
           maxx=h;
           else
           {
               if(h!=maxx)
               f=0;
           }
       }
       solve(root->left,h+1,maxx,f);
       solve(root->right,h+1,maxx,f);
    }
     bool check(Node *root)
    {
         int maxx=-1;
         bool f=1;
         solve(root,0,maxx,f);
         return f;
    }
};
```

7. Check if tree is isomorphic

```
class Solution{
  public:
   bool solve(Node* root,Node* root1)
    {
       if(root == NULL && root1 == NULL)
       return 1;
       else if(root == NULL || root1 == NULL)
       return 0;
       else if(root->data != root1->data)
```

```
        return 0;
        else
        return (solve(root->left,root1->right) && solve(root->right,root1->left)) ||
        (solve(root->left,root1->left) && solve(root->right,root1->right));
    }
    bool isIsomorphic(Node *root1,Node *root2)
    {
     if(solve(root1,root2))
     return 1;
     else
     return 0;
    }
};
```

8. Find successor and predecessor in BST

```
void findPreSuc(Node* root, Node*& pre, Node*& suc, int key)
{
    if(!root)
    return;
    findPreSuc(root->left , pre, suc, key);
    if(root->key<key)
    pre = root;
    if(root->key>key && !suc)
    suc = root;
    findPreSuc(root->right, pre, suc, key);
}
```

9. Construct BST using binary tree

```
class Solution{
  Public:
    int i=0;
    void inorder(Node* root, vector<int>& v)
    {
        if(root==0)
        return ;
        inorder(root->left, v);
        v.push_back(root->data);
        inorder(root->right, v);
    }
    void solve(Node *root, vector<int> &v)
```

```
    {
        if(root==0)
        return ;
        solve(root->left, v);
        root->data=v[i++];
        solve(root->right, v);
    }
    Node *binaryTreeToBST (Node *root)
    {
       vector<int>ans;
       inorder(root, ans);
       sort(ans.begin(), ans.end());
       solve(root, ans);
       return root;
    }
};
```

10. Count pairs from 2 BST whose sum is same as given sum

```
class Solution{
public:
  unordered_map<int,int>mp;
  int c=0;
  void maping(Node *root){
     if(root==NULL)
     return;
     mp[root->data]=1;
     maping(root->left);
     maping(root->right);
  }
  void count(Node *root,int x)
  {
     if(root==0)
     return;
     int n=x-root->data;
     if(mp.find(n)!=mp.end())
     c++;
     count(root->left,x);
     count(root->right,x);
  }
  int countPairs(Node* root1, Node* root2, int x){
    maping(root1);
    count(root2,x);
    return c;
   }};
```

11. Find a pair with given sum in a binary tree

```
class Solution {
public:
   bool solve(TreeNode* root, int k, unordered_set<int>& st){
      if(root==0)
      return false;
      if(st.count(k-root->val))
      return true;
      st.insert(root->val);
      return solve(root->left,k,st) || solve(root->right,k,st);
   }
   bool findTarget(TreeNode* root, int k) {
      unordered_set<int>st;
      return solve(root,k,st);
   }
};
```

MEDIUM :

1. Right View a Binary Tree.

```
vector<int> Solution::solve(TreeNode* A)
{
    vector<int>v;
    if(A==NULL)
    return v;

    queue<TreeNode*>q;
    q.push(A);
    v.push_back(A->val);
    while(!q.empty())
    {
        int n=q.size();
        vector<int>temp;
        while(n--)
        {
            TreeNode *curr=q.front();
            q.pop();
            if(curr->left!=NULL)
            {
                q.push(curr->left);
                temp.push_back(curr->left->val);
            }
            if(curr->right!=NULL)
            {
                q.push(curr->right);
                temp.push_back(curr->right->val);
            }
```

```
        }
        int sizee=temp.size();
        if(sizee!=0)
        v.push_back(temp[sizee-1]);
    }
    return v;
}
```

2. Lowest common ancestor in binary tree

```
class Solution {
public:
  TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q)
   {

      if(root==NULL || root==p || root==q)
      return root;
      TreeNode* l=lowestCommonAncestor(root->left, p, q);
      TreeNode* r=lowestCommonAncestor(root->right, p, q);
      if(l==NULL)
      return r;
      else if(r==NULL)
      return l;
      else
      return root;
   }
};
```

*** 3. All node distance k in binary tree**

```
class Solution {
public:
    void parents(map<TreeNode *,TreeNode *>&par,TreeNode *root)
   {
      queue<TreeNode *>q;
      q.push(root);
      while(!q.empty())
      {
         TreeNode *curr=q.front();
         q.pop();
         if(curr->left)
         {
            par[curr->left]=curr;
            q.push(curr->left);
```

```
        }
        if(curr->right)
        {
            par[curr->right]=curr;
            q.push(curr->right);
        }
    }

}
vector<int> distanceK(TreeNode* root, TreeNode* target, int k) {

    map<TreeNode *,TreeNode *>parent;
    parents(parent,root);
    vector<int>ans;
    map<TreeNode *,bool>vis;
    vis[target]=1;
    int i;
    queue<TreeNode *>qu;
    qu.push(target);
    int l=0;
    while(!qu.empty())
    {
        int n=qu.size();
        if(l==k)
        break;
        while(n--)
        {
            TreeNode *curr=qu.front();
            qu.pop();
             if(curr->left && !vis[curr->left])
            {
                vis[curr->left]=1;
                qu.push(curr->left);
            }
            if(curr->right && !vis[curr->right])
            {
                vis[curr->right]=1;
                qu.push(curr->right);
            }
            if(parent[curr] && !vis[parent[curr]])
            {
                vis[parent[curr]]=1;
                qu.push(parent[curr]);
            }

        }
        l++;
    }
    while(!qu.empty())
    {
```

```
            ans.push_back(qu.front()->val);
            qu.pop();
        }
        return ans;
    }
};
```

4. Validate binary search tree

```
class Solution {
public:
    bool isBST(TreeNode *root,long long int min,long long int max)
    {
        if(root==NULL)
        return 1;
        return (root->val>min && root->val<max && isBST(root->left,min,root->val)
        && isBST(root->right,root->val,max));
    }
    bool isValidBST(TreeNode* root)
    {
        return isBST(root,-1e18,1e18);
    }
};
```

5. Zigzag level order traversal

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root)
    {
        vector<vector<int>> v;
                if(root==NULL)
        return v;
                queue<TreeNode*> qu;
                qu.push(root);
        int h=0;
        while(!qu.empty())
        {
            int n=qu.size();
            vector<int> arr;
            while(n>0)
            {
                TreeNode* node = qu.front();
                qu.pop();
                arr.push_back(node->val);
                if(node->left)
```

```
                qu.push(node->left);
            if(node->right)
                qu.push(node->right);
            n--;
        }
        if(h%2==0)
        v.push_back(arr);
        else
        {
            reverse(arr.begin(),arr.end());
            v.push_back(arr);
        }
        h++;
      }
      return v;
   }
};
```

6. Height of Binary tree / Depth of binary tree

```
class Solution{
   public:
   int height(struct Node* node)
   {
          if(node==NULL)
            return 0;
            else
            return max(height(node->left)+1,height(node->right)+1);
   }
};
```

7. Bottom view of binary tree

```
class Solution {
  public:
    vector <int> bottomView(Node *root)
    {
       map<int,int>mp;
      queue<pair<Node*,int>>q;
      vector<int>ans;
      if(root==NULL)
      return ans;
      q.push({root,0});
      while(!q.empty())
      {
         Node* temp=q.front().first;
         int l=q.front().second;
         mp[l]=temp->data; // for top view  if(!mp[l]) mp[l]=temp->data;
         if(temp->left)
```

```
            q.push({temp->left,l-1});
            if(temp->right)
            q.push({temp->right,l+1});
            q.pop();
        }
        for(auto it:mp)
        ans.push_back(it.second);
    }
};
```

8. Construct binary tree using preorder and inorder

```
class Solution {
public:
   TreeNode* solve(vector<int>&preor,vector<int>&inor,int l,int r,int
&pre,unordered_map<int,int> &mp)
    {
        if(l>r)
        return 0;
        TreeNode* root=new TreeNode(preor[pre++]);
        if(l==r)
        return root;
        int k=mp[root->val];
        root->left=solve(preor,inor,l,k-1,pre,mp);
        root->right=solve(preor,inor,k+1,r,pre,mp);
        return root;
    }
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        int pre = 0;
        unordered_map<int,int>mp;
        for(int i=0;i<inorder.size();i++)
        mp[inorder[i]]=i;
        return solve(preorder,inorder,0,inorder.size()-1,pre,mp);
    }
};
```

9. Construct binary tree using inorder and postorder

```
class Solution {
public:
     TreeNode* solve(vector<int>&preor,vector<int>&inor,int l,int r,int
&post,unordered_map<int,int> &mp)
    {
        if(l>r)
        return 0;
        TreeNode* root=new TreeNode(preor[post--]);
         if(l==r)
        return root;
        int k=mp[root->val];
```

```
        root->right=solve(preor,inor,k+1,r,post,mp);
        root->left=solve(preor,inor,l,k-1,post,mp);
        return root;
    }
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        int post = postorder.size()-1;
        unordered_map<int,int>mp;
        for(int i=0;i<inorder.size();i++)
        mp[inorder[i]]=i;
        return solve(postorder,inorder,0,inorder.size()-1,post,mp);
    }
};
```

10. Vertical sum of binary tree

```
void solve(TreeNode *node, long int pos,map<long  int, long int> &mp)
{
    if (node == NULL)
    return;
    solve(node->left, pos-1, mp);
    mp[pos] += node->val;
    solve(node->right, pos+1, mp);
}
vector<int> Solution::verticalSum(TreeNode* A)
{   map<long int, long int>mp;
    solve(A, 0,mp);
    vector<int>ans;
    for(auto it:mp)
    ans.push_back(it.second);
    return ans;
}
```

11. Path sum

```
int solve(TreeNode *root, int b,int sum)
{
    if (!root)
    return 0;
    sum += root->val;
    if(!root->left && !root->right)
    return sum==b;
    int l = solve(root->left,b,sum);
    int r = solve(root->right,b,sum);
    return (l||r);
}
int Solution::hasPathSum(TreeNode* A, int B)
{
    int sum=0;
    int res=solve(A,B,sum);
    return res;
}
```

12. Sum tree

```
class Solution
{
   public:
   int solve(Node *root,bool &f)
   {
      if(root==0)
      return 0;
      if(root->left==0 && root->right==0)
      return root->data;
      int l=solve(root->left,f);
      int r=solve(root->right,f);
      int sum=l+r;
      if(root->data!=sum)
      f=0;
      return sum+(root->data);
   }
   bool isSumTree(Node* root)
   {
       bool f=1;
       solve(root,f);
       return f;
   }
};
```

13. Last node of complete binary tree

```
int Solution::lastNode(TreeNode* A)
{
    vector<int>res;
    queue<TreeNode*>q;
    if(A->left==0 && A->right==0)
     return  A->val;
     q.push(A);
     int prev=A->val;
     while(!q.empty())
     {
         TreeNode *temp=q.front();
         q.pop();
         if(temp->right!=0 && temp->left!=0)
         {
             q.push(temp->left);
             q.push(temp->right);
             prev=temp->right->val;
```

```
        }
        else
        {
            if(temp->left==0 && temp->right==0)
            return prev;
            else if(temp->right==0 && temp->left!=0)
            return temp->left->val;
            else if(temp->left==0)
            return prev;
        }
     }
    return prev;
}
```

14. Duplicate subtree of size 2 or more

```
class Solution {
  public:
   unordered_map<string,int>mp;
    string solve(Node *root)
    {
       if(!root)
       return "&";
       string s="";
       if(root->left==0 && root->right==0)
       {
          s=to_string(root->data);
          return s;
       }
       s += to_string(root->data);
       s += solve(root->left);
       s += solve(root->right);
       mp[s]++;
       return s;
    }
    int dupSub(Node *root) {
         solve(root);
         for(auto it:mp)
         if(it.second>=2)
         return true;
         return false;
    }
};
```

15. Min distance between two nodes in binary tree

```
class Solution{
    public:
    int ans=0;
    int solve(Node* root,int a,int b)
    {
        if(root==NULL|| ans>0)
        return 0;
        int l=solve(root->left,a,b);
        int r=solve(root->right,a,b);
        if((root->data==a||root->data==b))
        {
            if(l!=0) ans=l;
            else if(r!=0) ans=r;
            else return 1;
        }
        if(l!=0&&r!=0) ans=l+r;
        else if(l!=0) return l+1;
        else if(r!=0) return r+1;
        return 0;
    }
    int findDist(Node* root, int a, int b) {
        if(a==b) return ans;
        solve(root,a,b);
        return ans;
    }
};
```

15. Populate inorder successor of all nodes

```
class Solution{
public:
    Node *temp=0;
    void solve(Node *root)
    {
        if(root==0)
        return;
        solve(root->right);
        root->next=temp;
        temp=root;
        solve(root->left);
        return;
    }
    void populateNext(Node *root)
    {
        solve(root);
    }
};
```

16. Count BST nodes that lie in given range

```
int getCount(Node *root, int l, int h)
{
  if(root==0)
  return 0;
  if(root->data<=h && root->data>=l)
  return getCount(root->left,l,h)+getCount(root->right,l,h)+1;
  else if (root->data>h)
  return getCount(root->left,l,h);
  else
  return getCount (root->right, l, h);
 }
```

HARD :

1. Serialize and deserialize binary tree

```
class Codec {
public:
    TreeNode* solve(queue<string> &q)
    {
       string str=q.front();
       q.pop();
       if(str=="@")
       return 0;
        int n=stoi(str);
       TreeNode* root = new TreeNode(n);
       root->left=solve(q);
       root->right=solve(q);
       return root;
   }

   string serialize(TreeNode* root)
   {
       if(root==0)
       return "@,";
       return to_string(root->val)+","+ serialize(root->left) + serialize(root->right);
   }

   TreeNode* deserialize(string data)
   {
       string str="";
       queue<string>q;
```

```
        for(char s:data)
        {
            if(s==',')
            {
                q.push(str);
                str="";
                continue;
            }
            str+=s;
        }
        return solve(q);
    }
};
```

2. Vertical Traversal of Binary Tree

```
vector<vector<int> > Solution::verticalOrderTraversal(TreeNode* A)
{
    map<int, vector<int>> mp;
    int pos=0;
    vector<vector<int>>ans;
    queue<pair<TreeNode*, int>>q;
    if (A==0)
    return ans;
    q.push({A,0});
    while (!q.empty())
    {
        pair<TreeNode *,int>temp=q.front();
        q.pop();
        pos=temp.second;
        TreeNode* root=temp.first;
        mp[pos].push_back(root->val);
        if(root->left)
        q.push({root->left,pos-1});
        if(root->right)
        q.push({root->right,pos+1});
    }
    for(auto it : mp)
    ans.push_back(it.second);
    return ans;
}
```