Introduction to Trees

29 September 2024 12:32

```
class Solution {
  public:
    int countNodes(int i) {
        // your code here
        return pow(2,i-1);
    }
};
```

```
static int countNodes(int i)
    {
        int result=(int)Mathi.pow(2,i-1);
        return (result);
    }
```

p

Binary Tree Representation in C++

29 September 2024 12:33

My Code=>

```
void create_tree(node* root0, vector<int> &vec){

    int n =vec.size();
    if(n<=0)
    {
       return;
    }

    queue<node*>q;
    //root0=newNode(vec[0]);
    q.push(root0);


    int i=1;
    while(q.size()>0&&i<n)
    {

            node* temp=q.front();
           q.pop();
           if(i<n)
           {
              temp->left=newNode(vec[i]);
              q.push(temp->left);

              i++;
              }
           if(i<n)
           {
              temp->right=newNode(vec[i]);
              q.push(temp->right);
              i++;
           }

    }

  }
```

#2 Soln

```
node* solve(vector<int> &vec,int root_index){
    if(root_index>=vec.size()){
       return NULL;
    }
    struct node *root=newNode(vec[root_index]);
    root->left=solve(vec,(root_index*2)+1);
    root->right=solve(vec,(root_index*2)+2);
    return root;
  }
```

```
void create_tree(node* &root0, vector<int> &vec){
   //Your code goes here
   root0=solve(vec,0);
}
```

Binary Tree Representation in Java***

29 September 2024 12:34

```
public static void createTree(Node root0, ArrayList<Integer> v ){
    Queue<Node> q=new LinkedList<>();
    q.add(root0);
    for(int i=0;i<3;i++)
    {
       Node parent= q.poll();
       Node left=new Node(v.get(2*i+1));
        Node right=new Node(v.get(2*i+2));
        parent.left=left;
        parent.right=right;
        q.add(left);
        q.add(right);
    }
  }
```

Binary Tree Traversals in Binary Tree*

29 September 2024 12:34

Binary Tree Preorder Traversal

29 September 2024 12:49

C++

```cpp
void preorder(TreeNode* root, vector<int>&v)
    {
        if(!root)
        {
            return;
        }
        v.push_back(root->val);
        preorder(root->left,v);
        preorder(root->right,v);
    }
    vector<int> preorderTraversal(TreeNode* root)
    {
        vector<int>v;
        if(!root)
        return v;
        preorder(root,v);
        return v;

    }
```

JAVA

```java
void preorder(TreeNode root,LinkedList<Integer>v)
    {
        if(root==null)
        {
            return;
        }
        v.add(root.val);
        preorder(root.left,v);
        preorder(root.right,v);
    }
    public List<Integer> preorderTraversal(TreeNode root) {
        LinkedList<Integer>v=new LinkedList();
        if(root==null)
        {
            return v;
        }
        preorder(root,v);
        return v;
    }
```

Inorder Traversal of Binary Tree

29 September 2024 12:57

JAVA

```java
void inorder(TreeNode root,LinkedList<Integer>v)
    {
        if(root==null)
        {
            return;
        }

        inorder(root.left,v);
        v.add(root.val);
        inorder(root.right,v);

    }
    public List<Integer> inorderTraversal(TreeNode root) {
        LinkedList<Integer>v=new LinkedList();
        if(root==null)
        {
            return v;
        }
        inorder(root,v);
        return v;
    }
```

Post-Order Traversal Of Binary Tree

29 September 2024 12:57

JAVA

```
void postorder(TreeNode root,LinkedList<Integer>v)
    {
        if(root==null)
        {
            return;
        }

        postorder(root.left,v);
        postorder(root.right,v);
        v.add(root.val);
    }
    public List<Integer> postorderTraversal(TreeNode root) {
        LinkedList<Integer>v=new LinkedList();
        if(root==null)
        {
            return v;
        }
        postorder(root,v);
        return v;
    }
```

Level order Traversal / Level order traversal in spiral form

29 September 2024 13:05

```
public List<List<Integer>> levelOrder(TreeNode root)
    {
        List<List<Integer>>v=new ArrayList<List<Integer>>();
        if(root==null)
        {
            return v;
        }
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);
        while(q.size()>0)
        {
            int cnt=q.size();
            List<Integer>ans=new ArrayList<>();
            while(cnt>0)
            {
                TreeNode temp=q.peek();
                q.remove();

                ans.add(temp.val);
                if(temp.left!=null)
                {
                    q.add(temp.left);
                }
                if(temp.right!=null)
                {
                    q.add(temp.right);
                }
                cnt--;

            }
            v.add(ans);

        }
        return v;
    }
```

Iterative Preorder Traversal of Binary Tree

29 September 2024 14:29

```
public List<Integer> preorderTraversal(TreeNode root)
    {
        LinkedList<Integer>ans=new LinkedList();
        if(root==null)
        {
            return ans;
        }
        Stack<TreeNode>s=new Stack<>();
        TreeNode curr=root;

        while(curr!=null||s.size()>0)
        {
            if(curr!=null)
            {
                ans.add(curr.val);
                s.push(curr);
                curr=curr.left;
            }
            else
            {
                curr=s.peek();
                s.pop();
                curr=curr.right;
            }
        }
        return ans;

    }
```

Iterative Inorder Traversal of Binary Tree

29 September 2024 14:29

```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root)
    {
        LinkedList<Integer>ans=new LinkedList();
        if(root==null)
        {
            return ans;
        }
        Stack<TreeNode>s=new Stack<>();
        TreeNode curr=root;

        while(curr!=null||s.size()>0)
        {
            if(curr!=null)
            {

                s.push(curr);
                curr=curr.left;
            }
            else
            {
                curr=s.peek();
                s.pop();
                ans.add(curr.val);
                curr=curr.right;
            }
        }
        return ans;

    }
}
```

Post-order Traversal of Binary Tree using 2 stack***

29 September 2024 14:30

```
public List<Integer> postorderTraversal(TreeNode root)
    {
        LinkedList<Integer>ans=new LinkedList();
        if(root==null)
        {
            return ans;
        }
        Stack<TreeNode>s1=new Stack<>();
        Stack<TreeNode>s2=new Stack<>();
        TreeNode curr=root;
        s1.push(curr);

        while(s1.size()>0)  // (Why curr!=null work here)
        {

            curr=s1.peek();
            s1.pop();
            s2.push(curr);
            if(curr.left!=null)
            {
                s1.push(curr.left);
            }
            if(curr.right!=null)
            {
                s1.push(curr.right);
            }
        }
        while(s2.size()>0)
        {
            TreeNode temp=s2.peek();
            s2.pop();
            ans.add(temp.val);
        }
        return ans;

    }
```

Post-order Traversal of Binary Tree using 1 stack

29 September 2024 16:44

```
class Solution {
    public List<Integer> postorderTraversal(TreeNode root)
    {
        LinkedList<Integer>ans=new LinkedList();
        if(root==null)
        {
            return ans;
        }
        Stack<TreeNode>s1=new Stack<>();
        Stack<TreeNode>s2=new Stack<>();
        TreeNode curr=root;
        s1.push(curr);

        while(s1.size()>0)
        {

            curr=s1.peek();
            s1.pop();
            ans.add(curr.val);
            if(curr.left!=null)
            {
                s1.push(curr.left);
            }
            if(curr.right!=null)
            {
                s1.push(curr.right);
            }
        }
        LinkedList<Integer>v=new LinkedList<>();
        for(int i=ans.size()-1;i>=0;i--)
        {
            v.add(ans.get(i));
        }
        return v;

    }
}
```

Preorder, Inorder, and Postorder Traversal in one Traversal

29 September 2024 16:44

Height of a Binary Tree

29 September 2024 16:45

```
class Solution {
    public int maxDepth(TreeNode root)
    {
        if(root==null)
        {
            return 0;
        }
        return 1+Math.max(maxDepth(root.left),maxDepth(root.right));

    }
}
```

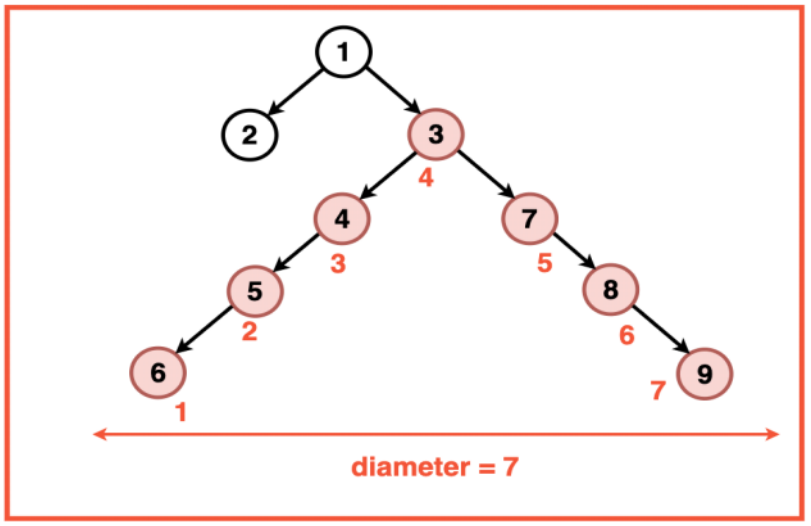
Check if the Binary tree is height-balanced or not***

29 September 2024 16:45

```
public static int isbalanced(TreeNode root,boolean f)
    {
        if(root==null)
        {
            return 0;
        }
        int l=isbalanced(root.left,f);
        int r=isbalanced(root.right,f);
        if(l==-1||r==-1)
        {
            return -1;
        }
        if(Math.abs(l-r)>1)
        {
            //f=false;   =====> how to use this;
            return -1;
        }
        return 1+Math.max(l,r);
    }
    public boolean isBalanced(TreeNode root) {
        if(root==null)
        {
            return true;
        }

        int h=isbalanced(root,f);
        return h==-1?false:true;

    }
```

Diameter of Binary Tree

29 September 2024 16:45



Diameter means max edges bet 2 nodes

```
class Solution {
    int d;
    public int find(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int l = find(root.left);
        int r = find(root.right);
        d = Math.max(d, l + r);
        return 1 + Math.max(l, r);
    }
    public int diameterOfBinaryTree(TreeNode root) {
        d = 0;
        int h = find(root);
        return d;
    }
}
```

Maximum path sum***

29 September 2024 16:45

```
int ans=0;
    int find(TreeNode root)
    {
        if(root==null)
        {
            return 0;
        }
        int lsum=find(root.left);
        int rsum=find(root.right);
        lsum=lsum>0?lsum:0;  //taking the negative path will never give you the max
of so we ignore that path and take 0
        rsum=rsum>0?rsum:0;
        ans=Math.max(ans,lsum+rsum+root.val);
        return Math.max(lsum,rsum)+root.val;
    }
    public int maxPathSum(TreeNode root)
    {
        ans=root.val;
        int k=find(root);
        return ans;

    }
```

How to use pass by reference in Java?

```
int ans=0;
    int find(TreeNode root,int ans[])
    {
        if(root==null)
        {
            return 0;
        }
        int lsum=find(root.left,ans);
        int rsum=find(root.right,ans);
        lsum=lsum>0?lsum:0;
        rsum=rsum>0?rsum:0;
        ans[0]=Math.max(ans[0],lsum+rsum+root.val);
        return Math.max(lsum,rsum)+root.val;
    }
    public int maxPathSum(TreeNode root)
    {
        int ans[]=new int[1];
        ans[0]=Integer.MIN_VALUE;
        int k=find(root,ans);
        return ans[0];

    }
```

Check if two trees are identical or not

29 September 2024 16:45

```
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q)
    {
        if(p==null&&q==null)
        {
            return true;
        }
        if(p==null||q==null)
        {
            return false;
        }
        if(p.val!=q.val)
        {
            return false;
        }
        return isSameTree(p.left,q.left)&&isSameTree(p.right,q.right);

    }
}
```

Zig Zag Traversal of Binary Tree

29 September 2024 16:45

WE CAN DO WITH QUEUE ALSO WITH FLAG:

```
class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root)
    {
        List<List<Integer>>ans=new ArrayList<List<Integer>>();
        //List<List<Integer>>ans=new ArrayList<List<Integer>>();
        Stack<TreeNode>s1=new Stack<>();
        Stack<TreeNode>s2=new Stack<>();
        if(root==null)
        {
            return ans;
        }
        s1.push(root);
        while(s1.size()>0||s2.size()>0)
        {
            if(s1.size()>0)
            {
                int cnt=s1.size();
                List<Integer>v=new ArrayList<>();
                while(cnt>0)
                {
                    TreeNode temp=s1.peek();
                    s1.pop();
                    v.add(temp.val);
                    if(temp.left!=null)
                    {
                        s2.push(temp.left);
                    }
                    if(temp.right!=null)
                    {
                        s2.push(temp.right);
                    }
                    cnt--;
                }
                ans.add(v);
            }

            if(s2.size()>0)
            {
                int cnt=s2.size();
                List<Integer>u=new ArrayList<>();
                while(cnt>0)
                {
                    TreeNode temp=s2.peek();
                    s2.pop();
                    u.add(temp.val);
                    if(temp.right!=null)
                    {
                        s1.push(temp.right);
                    }
                    if(temp.left!=null)
                    {
                        s1.push(temp.left);
                    }
                    cnt--;
                }
                ans.add(u);
            }
```

```
            }
            return ans;

        }
    }
```

Boundary Traversal of Binary Tree***

30 September 2024 20:02

```
ArrayList<Integer>ans=new ArrayList<>();
   void leftB(Node node)
   {
      if(node==null)
      {
         return;
      }
      if(node.left==null&&node.right==null)
      {
         return;
      }
      if(node.left!=null)
      {
         ans.add(node.data);
         leftB(node.left);
      }
      else if(node.right!=null)
      {
         ans.add(node.data);
         leftB(node.right);
      }

   }
   void rightB(Node node)
   {
      if(node==null)
      {
         return;
      }
      if(node.left==null&&node.right==null)
      {
         return;
      }
      if(node.right!=null)
      {
         rightB(node.right);
         ans.add(node.data);
      }
      else if(node.left!=null)
      {
         rightB(node.left);
         ans.add(node.data);
      }


   }

   void leaveB(Node node)
   {
      if(node==null)
      {
```

```
        return;
      }
      leaveB(node.left);
      if(node.left==null&&node.right==null)
      {
        ans.add(node.data);
      }
      leaveB(node.right);
    }

    ArrayList<Integer> boundaryTraversal(Node node)
    {

      if(node==null)
      {
        return ans;
      }
      ans.add(node.data);

      if(node.left!=null)
      {
        leftB(node.left);
      }
      //left leave
      if(node.left!=null)
      {
        leaveB(node.left);
      }
      if(node.right!=null)
      {
        leaveB(node.right);
      }

      if(node.right!=null)
      {
        rightB(node.right);
      }

      return ans;
    }
```

1 2 2 5 5 4 8 9 6 7 3 3

Vertical Order Traversal of Binary Tree***

30 September 2024 20:02

```
class Pair{
        TreeNode node;
        int id;
        Pair(TreeNode node,int id)
        {
            this.node=node;
            this.id=id;
        }
    }
class Solution {
    public List<List<Integer>> verticalTraversal(TreeNode root)
    {
        List<List<Integer>> ans = new ArrayList<>();
        TreeMap<Integer, List<Integer>> mp = new TreeMap<>();
        Queue<Pair>q=new LinkedList<>();
        q.add(new Pair(root,0));
        while(q.size()>0)
        {
            Pair a=q.poll();

            TreeNode x=a.node;
            int id=a.id;
            //mp.get(id).add(x.val);
             // Add the node value to the map at the corresponding vertical level
            //mp.computeIfAbsent(id, k -> new ArrayList<>()).add(x.val);
            //other way
            if (!mp.containsKey(id))
            {
                mp.put(id, new ArrayList<>());
            }
            mp.get(id).add(x.val);
            if(x.right!=null)
            {
                q.add(new Pair(x.right,id+1));
            }
            if(x.left!=null)
            {
                q.add(new Pair(x.left,id-1));
            }

        }
        // Add the vertical levels to the answer
        for (Map.Entry<Integer, List<Integer>> entry : mp.entrySet()) {
            ans.add(entry.getValue());
        }
        return ans;
    }
}
```

Top View of Binary Tree

30 September 2024 20:03

```
class Pair{
    Node node;
    int id;
    Pair(Node node,int id)
    {
        this.node=node;
        this.id=id;
    }
};

class Solution {
    // Function to return a list of nodes visible from the top view
    // from left to right in Binary Tree.
    static ArrayList<Integer> topView(Node root)
    {
        ArrayList<Integer>ans=new ArrayList<>();
        Queue<Pair>q=new LinkedList<>();
        Map<Integer,Integer>mp=new TreeMap<>(); // TreeMap to maintain order of keys

        q.add(new Pair(root,0));


        while(q.size()>0)
        {
            Pair temp=q.poll();
            Node node=temp.node;
            int id=temp.id;
            // if(mp.find(id)==mp.end())
            // {
            //     mp.get(id).add(node.val);
            // }


            // If the horizontal distance (id) is not yet in the map,
            // this is the first node at this horizontal distance (top view).
            if (!mp.containsKey(id)) {
                mp.put(id, node.data);
            }

            if(node.left!=null)
            {
                q.add(new Pair(node.left,id-1));
            }

            if(node.right!=null)
            {
                q.add(new Pair(node.right,id+1));
            }
        }
        for (Map.Entry<Integer,Integer> entry : mp.entrySet()) {
            ans.add(entry.getValue());
```

```
        }
        return ans;

        // code here
    }
}
```

Bottom View of Binary Tree

30 September 2024 20:03

Easy to Understand

```
class Solution
{
  public class Pair {
      Node node;
      int hd;
      Pair(Node node, int hd) {
         this.node = node;
         this.hd = hd;
      }
  }
   public ArrayList <Integer> bottomView(Node root)
   {
      // Code here
      //jaise top view mein kia tha
      //level order kro
      Node ptr = root;
      TreeMap<Integer, Integer> map = new TreeMap<>();
      Queue<Pair> q = new LinkedList<>();
      q.add(new Pair(ptr, 0));
      while(!q.isEmpty()){
         Pair p = q.poll();  /return and removes the element at the front end of the container
         Node nd = p.node;
         int h = p.hd;

         map.put(h, nd.data);

         if(nd.left != null){
            q.add(new Pair(nd.left, h-1));
         }
         if(nd.right != null){
            q.add(new Pair(nd.right, h+1));
         }

      }
      ArrayList<Integer> ans = new ArrayList<>();
      for(Map.Entry<Integer, Integer> hm : map.entrySet()){
         int a = hm.getValue();
         ans.add(a);
      }

      return ans;

   }
}
```

Right/Left View of Binary Tree Recursive sol of all present

30 September 2024 20:03

```
class Solution {
    public List<Integer> rightSideView(TreeNode root)
    {
        List<Integer>ans=new ArrayList<>();
        if(root==null)
        {
            return ans;
        }
        Queue<TreeNode>q=new LinkedList<>();
        q.add(root);
        while(q.size()>0)
        {
            int cnt=0;
            int n=q.size();
            while(cnt<n)
            {
                TreeNode temp=q.peek();
                q.remove();
                if(cnt==n-1)
                {
                    ans.add(temp.val);
                }
                if(temp.left!=null)
                {
                    q.add(temp.left);
                }
                if(temp.right!=null)
                {
                    q.add(temp.right);
                }
                cnt++;
            }
        }
        return ans;

    }
}
```

```
Void find(TreeNode curr,List<integer>res,int currDepth)//curr-> res-> 0;
If(curr==null)
Return;
If(currdepth==res.size()
{
Res.add(curr.val);
}
Find(curr.roight,res,currDepth+1);
Find(curr.left,res,currDepth+1);
```

RECURSIVE SOLN:

```
class Node {
  int data;
  Node left, right;
  Node(int x) {
    data = x;
    left = right = null;
```

```
    }
}


static void RecursiveRightView(Node root, int level, int[] maxLevel,ArrayList<Integer> result) {
        if (root == null) return;
        // If current level is more than max level,
        // this is the first node of that level
        if (level > maxLevel[0]) {
            result.add(root.data);
            maxLevel[0] = level;
        }
        // Traverse right subtree first, then left subtree
        RecursiveRightView(root.right, level + 1,
                           maxLevel, result);
        RecursiveRightView(root.left, level + 1,
                           maxLevel, result);
    }
    // Function to return the right view of the binary tree
    static ArrayList<Integer> rightView(Node root) {
        ArrayList<Integer> result = new ArrayList<>();
        int[] maxLevel = new int[] {-1};

        // Start recursion with root at level 0
        RecursiveRightView(root, 0, maxLevel, result);

        return result;
    }
```

****

Bottom View:

```
void dfs(Node* root, int dist, int level, auto &map){
    if (root == nullptr) return;
    if (map.find(dist) == map.end() || level >= map[dist].second) map[dist] = { root->key, level }; //
only change is level condition reverses.
    dfs(root->left, dist - 1, level + 1, map);
    dfs(root->right, dist + 1, level + 1, map);
}
```

++

Top View:

```
#include <iostream>
#include <map>
using namespace std;
struct Node
{
    int key;
    Node *left, *right;
    Node(int key){
        this->key = key;
        this->left = this->right = nullptr;
    }
};
void dfs(Node* root, int dist, int level, auto &map){
    if (root == nullptr) return;
```

```
    if (map.find(dist) == map.end() || level < map[dist].second) map[dist] = { root->key, level };
    dfs(root->left, dist - 1, level + 1, map);
    dfs(root->right, dist + 1, level + 1, map);
}

int main()
{
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->right = new Node(4);
    root->right->left = new Node(5);
    root->right->right = new Node(6);
    root->right->left->left = new Node(7);
    root->right->left->right = new Node(8);
    map<int, pair<int, int>> map;
    dfs(root,0,0,map);
for (auto it: map) cout << it.second.first << " ";
    return 0;
}
```

Symmetric Binary Tree

30 September 2024 20:03

```
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q)
    {
        if(p==null&&q==null)
        {
            return true;
        }
        if(p==null||q==null)
        {
            return false;
        }
        if(p.val!=q.val)
        {
            return false;
        }
        return isSameTree(p.left,q.right)&&isSameTree(p.right,q.left);

    }
    public lboolean isSymmetric(TreeNode root)
    {
        if(root==null)
        {
            return true;
        }
        if(root.left==null&&root.right==null)
        {
            return true;
        }
        if(root.left==null||root.right==null)
        {
            return false;
        }

        return isSameTree(root.left,root.right);

    }
}
```

Root to Node Path in Binary Tree

01 October 2024 00:18

```
class Solution {
   public static void find(Node root,ArrayList<ArrayList<Integer>>ans,ArrayList<Integer>curr)
   {

      if(root==null)
      {
         return;
      }
      curr.add(root.data);
      if(root.left==null&&root.right==null)
      {
         ans.add(new ArrayList<>(curr));
         curr.remove(curr.size()-1);    //Why this code Because it is pass by ref
         return;
      }
      if(root.left!=null) //if is using taki ek side se print ho cheeze
      {
        //curr.add(root.left.data) => why this code is not working we are already adding val above
         find(root.left,ans,curr);
      }
      if(root.right!=null)
      {
         find(root.right,ans,curr);
      }
      curr.remove(curr.size()-1);
   }
   public static ArrayList<ArrayList<Integer>> Paths(Node root)
   {
      ArrayList<ArrayList<Integer>>ans=new ArrayList<>();
      ArrayList<Integer>curr=new ArrayList<>();

      find(root,ans,curr);
      return ans;

   }
}
```

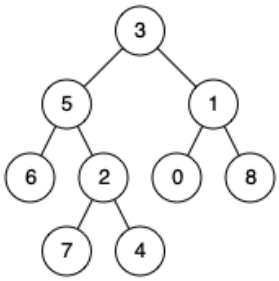
```
import java.util.ArrayList;
import java.util.List;

// TreeNode structure
```

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int x) {
        val = x;
        left = null;
        right = null;
    }
}

public class Solution {
    // Function to find the path from the
    // root to a given node with value 'x'
    public boolean getPath(TreeNode root, List<Integer> arr, int x) {
        // Base case: If the current
        // node is null, return false
        if (root == null) {
            return false;
        }

        // Add the current node's
        // value to the path list
        arr.add(root.val);

        // If the current node's value is equal
        // to the target value 'x', return true
        if (root.val == x) {
            return true;
        }

        // Recursively search for the target value
        // 'x' in the left and right subtrees
        if (getPath(root.left, arr, x) || getPath(root.right, arr, x)) {
            return true;
        }

        // If the target value 'x' is not found
        // in the current path, backtrack
        arr.remove(arr.size() - 1);
        return false;
    }

    // Function to find and return the path from
    // the root to a given node with value 'B'
    public List<Integer> solve(TreeNode A, int B) {
        // Initialize an empty
        // list to store the path
        List<Integer> arr = new ArrayList<>();

        // If the root node is null,
        // return the empty path list
        if (A == null) {
            return arr;
        }
```

```
        // Call the getPath function to find
        // the path to the node with value 'B'
        getPath(A, arr, B);

        // Return the path list
        return arr;
    }

    public static void main(String[] args) {
        TreeNode root = new TreeNode(3);
        root.left = new TreeNode(5);
        root.right = new TreeNode(1);
        root.left.left = new TreeNode(6);
        root.left.right = new TreeNode(2);
        root.right.left = new TreeNode(0);
        root.right.right = new TreeNode(8);
        root.left.right.left = new TreeNode(7);
        root.left.right.right = new TreeNode(4);

        Solution sol = new Solution();

        int targetLeafValue = 7;

        List<Integer> path = sol.solve(root, targetLeafValue);

        System.out.print("Path from root to leaf with value " +
              targetLeafValue + ": ");
        for (int i = 0; i < path.size(); ++i) {
            System.out.print(path.get(i));
            if (i < path.size() - 1) {
                System.out.print(" -> ");
            }
        }
    }
}
```

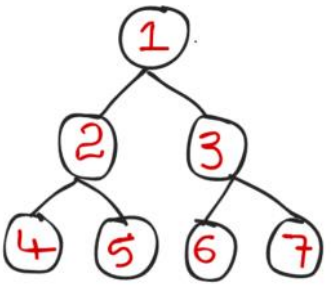
LCA in Binary Tree

01 October 2024 00:18



4 & 7 => 2
5 & 8 => 3
5 & 4 => 5

```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p,
TreeNode q)
    {
        if(root==null)
        {
            return root;
        }
        if(root==p||root==q)
        {
            return root;
        }
        TreeNode LH=lowestCommonAncestor(root.left,p,q);
        TreeNode RH=lowestCommonAncestor(root.right,p,q);
        if(LH!=null&&RH!=null)
        {
            return root; //Code Samjh bhai chaap mat//if LH return 6
ans RH returns 4 to 6 aur 4 kyu return karna return 2 instead of 6&4
        }
        if(LH!=null)
        {
            return LH;
        }
        if(RH!=null)
        {
            return RH;
        }
        return null;

    }
}
```



Easy code

```
 public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p,
TreeNode q)
    {
        if(root==null)
        {
            return root;
```

```
        }
        if(root==p||root==q)
        {
            return root;
        }
        TreeNode LH=lowestCommonAncestor(root.left,p,q);
        TreeNode RH=lowestCommonAncestor(root.right,p,q);
        if(LH==null)
        {
            return RH;
        }
        else if(RH==null)
        {
            return LH;
        }
        else
        {
            return root;
        }

    }
```

Maximum width of a Binary Tree***

01 October 2024 00:18

```
class Solution {
    public int widthOfBinaryTree(TreeNode root) {
        if (root == null) return 0;

        int maxWidth = 0;
        Queue<Pair<TreeNode, Integer>> queue = new LinkedList<>();
        queue.offer(new Pair<>(root, 0));

        while (!queue.isEmpty()) {
            int size = queue.size();
            int minIndex = queue.peek().getValue(); // to prevent integer overflow
            int first = 0, last = 0;

            for (int i = 0; i < size; i++) {
                Pair<TreeNode, Integer> current = queue.poll();
                TreeNode node = current.getKey();
                int index = current.getValue() - minIndex;

                if (i == 0) first = index;
                if (i == size - 1) last = index;

                if (node.left != null)
                    queue.offer(new Pair<>(node.left, 2 * index));
                if (node.right != null)
                    queue.offer(new Pair<>(node.right, 2 * index + 1));
            }

            maxWidth = Math.max(maxWidth, last - first + 1);
        }

        return maxWidth;
    }
}
```

Check for Children Sum Property

01 October 2024 00:19

```
class Solution {
    int v;
    Boolean flag;
    int find(TreeNode root)
    {
        if(root==null)
        {
            return 0;
        }
        if(root.left==null&&root.right==null)
        {
            return root.val;
        }
        int l=find(root.left);
        int r=find(root.right);
        if(root.val!=(l+r))
        {
            flag=false;
            return 0;
        }
        return root.val;
    }

    public boolean checkTree(TreeNode root)
    {
        if(root==null)
        {
            return true;
        }
        flag=true;
        v=find(root);
        return flag;

    }
}
```

```
##GFG SLUTIN
class Solution
{
  //Function to check whether all nodes of a tree have the value
  //equal to the sum of their child nodes.
  static  int v;
  static int flag;
  static int find(Node root)
  {
    if(root==null)
    {
      return 0;
    }
    if(root.left==null&&root.right==null)
    {
      return root.data;
    }
    int l=find(root.left);
```

```
        int r=find(root.right);
        if(l==-1||r==-1)
        {
            return -1;
        }
        if(root.data!=(l+r))
        {

            return -1;
        }
        return root.data;
    }

    public static int isSumProperty(Node root)
    {
        // add your code here
        if(root==null)
        {
            return 1;
        }
        //flag=0;
        v=find(root);
        return v==-1?0:1;


    }
}
```

Print all the Nodes at a distance of K in a Binary Tree

01 October 2024 00:19

```
class Solution
{
    public List<Integer> distanceK(TreeNode root, TreeNode target, int k)
    {
        Map<TreeNode,TreeNode>parent=new HashMap<>();
        //Store the child corresponding to par
        markParents(root,null,parent);


        //#step 2
        TreeNode tgt=findNode(target.val,root); //No need to find tgt ans target node already given

        //#step 3
        Queue<TreeNode>q=new LinkedList<>();
        Set<TreeNode>visited=new HashSet<>();
        q.offer(tgt);
        visited.add(tgt);

        //q.offer(target);
        //visited.add(target);


        int level=0;
        int n=q.size();
        while(q.size()>0)
        {
            if(level==k)
            {
                break;
            }
            int size=q.size();
            level++;
            while(size>0)
            {
                TreeNode curr=q.poll();
                if(curr.left!=null && !visited.contains(curr.left))
                {
                    q.offer(curr.left);
                    visited.add(curr.left);
                }
                if(curr.right!=null && !visited.contains(curr.right))
                {
                    q.offer(curr.right);
                    visited.add(curr.right);
                }
                TreeNode parentNode=parent.get(curr);
                if(parentNode!=null&&!visited.contains(parentNode))
                {
                    q.offer(parentNode);
                    visited.add(parentNode);
                }
                size--;
            }

        }

        List<Integer> result = new ArrayList<>();
        while (!q.isEmpty()) {
            result.add(q.poll().val);
        }
        Collections.sort(result);
```

```
        return result;
    }


    public static void markParents(TreeNode root,TreeNode par,Map<TreeNode,TreeNode>parent)
    {
        if(root==null)
        {
            return;
        }
        parent.put(root,par);
        markParents(root.left,root,parent);
        markParents(root.right,root,parent);
    }


    public static TreeNode findNode(int tgt,TreeNode root)
    {
        if(root==null)
        {
            return null;
        }
        if(root.val==tgt)
        {
            return root;
        }
        TreeNode l=findNode(tgt,root.left);
        TreeNode r=findNode(tgt,root.right);
        if(l==null)
        {
            return r;
        }
        if(r==null)
        {
            return l;
        }
        return root;
    }
}
```

Minimum time taken to BURN the Binary Tree from a Node

01 October 2024 00:19

```
class Solution
{
   /*class Node {
         int data;
         Node left;
         Node right;

         Node(int data) {
               this.data = data;
               left = null;
               right = null;
         }
   }*/

   public static int minTime(Node root, int target)
   {
      Map<Node,Node> parent=new HashMap<>();
      markParent(root,null,parent);
      Node tgt=findTarget(root,target);

      Queue<Node>q=new LinkedList<>();
      Set<Node>visited=new HashSet<>();
      q.offer(tgt);
      visited.add(tgt);
      int level=0;

      while(q.size()>0)
      {
         int size=q.size();
         while(size>0)
         {
            Node curr=q.poll();
            if(curr.left!=null && !visited.contains(curr.left))
            {
               q.offer(curr.left);
               visited.add(curr.left);
            }
            if(curr.right!=null && !visited.contains(curr.right))
            {
               q.offer(curr.right);
               visited.add(curr.right);
            }
            Node parentNode=parent.get(curr);
            if(parentNode!=null&&!visited.contains(parentNode))
            {
               q.offer(parentNode);
               visited.add(parentNode);
            }
            size--;
```

```
        }
        level++;
      }
      return level-1;

    }

    static Node findTarget(Node root,int tgt)
    {
      if(root==null)
      {
        return null;
      }
      if(root.data==tgt)
      {
        return root;
      }
      Node l=findTarget(root.left,tgt);
      Node r=findTarget(root.right,tgt);
      if(l==null)
      {
        return r;
      }
      if(r==null)
      {
        return l;
      }
      return root;
    }
    static void markParent(Node root,Node par,Map<Node,Node> parent)
    {
      if(root==null)
      {
        return;
      }
      parent.put(root,par);
      markParent(root.left,root,parent);
      markParent(root.right,root,parent);
    }

}
```

Count total Nodes in a COMPLETE Binary Tree

01 October 2024 00:20

```
class Solution {
    public int countNodes(TreeNode root)
    {
        Queue<TreeNode>q=new LinkedList<>();
        if(root==null)
        {
            return 0;
        }
        q.add(root);
        int cnt=0;
        while(q.size()>0)
        {
            TreeNode curr=q.poll();
            cnt++;
            if(curr.left!=null)
            {
                q.add(curr.left);
            }
            else
            {
                break;
            }
            if(curr.right!=null)
            {
                q.add(curr.right);
            }
            else
            {
                break;
            }
        }
        return cnt+q.size();
    }
}
```

O(N) O(N) time and space

```
public int countNodes(TreeNode root) {
      // Check if the tree is empty
      if (root == null) {
          return 0;
      }
// Find the height of the left subtree
      int lh = findHeightLeft(root);
      // Find the height of the right subtree
      int rh = findHeightRight(root);
// If the heights are equal, the tree
      // is a full binary tree, and we can
      // calculate the total nodes
```

```
        if (lh == rh) {
            return (1 << lh) - 1;
        }
// If the heights are not equal,
        // recursively count nodes in the
        // left and right subtrees
        return 1 + countNodes(root.left) + countNodes(root.right);0
    }
// Function to find the
    // height of the left subtree
    private int findHeightLeft(TreeNode node) {
        int height = 0;
        while (node != null) {
            height++;
            node = node.left;
        }
        return height;
    }
// Function to find the
    // height of the right subtree
    private int findHeightRight(TreeNode node) {
        int height = 0;
        while (node != null) {
            height++;
            node = node.right;
        }
        return height;
    }
```

space complexity is O(log N).
Time Complexity: O(log N * log N)

Requirements needed to construct a Unique Binary Tree | Theory

01 October 2024 00:20

```
public static boolean isPossible(int a, int b)
  {
     // if(a==2||b==2&&a!=b)
     // {
     //     return true;
     // }
     // else
     // {
     //     return false;
     // }
      return(a!=b&&a+b!=4);
     // Code here
  }
```

Construct Binary Tree from inorder and preorder

01 October 2024 00:20

```
class Solution {

    public TreeNode buildTree(int[] preorder, int[] inorder) {
        return solve(preorder,0,preorder.length-1,inorder,0,inorder.length-1);

    }
    int search(int[] inorder,int s,int e, int val)
    {
        for(int i=s;i<=e;i++)
        {
            if(inorder[i]==val)
            {
                return i;

            }
        }
        return -1;
    }
    TreeNode solve(int[] preorder,int pres,int pree,int[] inorder,int ins,int ine)
    {
        if(pres>pree||ins>ine)
        {
            return null;
        }
        TreeNode root=new TreeNode(preorder[pres]);
        int mid=search(inorder,ins,ine,preorder[pres]);
        int leftcnt=mid-ins;
        root.left=solve(preorder,pres+1,pres+leftcnt,inorder,ins,mid-1);
        root.right=solve(preorder,pres+leftcnt+1,pree,inorder,mid+1,ine);
        return root;
    }
}
```

Construct the Binary Tree from Postorder and Inorder Traversal

01 October 2024 00:20

```
class Solution {
    public TreeNode buildTree(int[] inorder, int[] postorder) {
        return solve(postorder,0,postorder.length-1,inorder,0,inorder.length-1);
    }
    TreeNode solve(int[] postorder,int posts,int poste,int[] inorder,int ins,int
ine)
    {
        if(posts>poste||ins>ine)
        {
            return null;
        }
        TreeNode root=new TreeNode(postorder[poste]);
        int mid=search(inorder,ins,ine,postorder[poste]);
        //int leftcnt=mid-ins;
        root.left=solve(postorder,posts,posts-ins+mid-1,inorder,ins,mid-1);
        root.right=solve(postorder,poste-ine+mid,poste-1,inorder,mid+1,ine);
        return root;
    }
    int search(int[] inorder,int s,int e, int val)
    {
        for(int i=s;i<=e;i++)
        {
            if(inorder[i]==val)
            {
                return i;

            }
        }
        return -1;
    }
}
```

Serialize and deserialize Binary Tree

01 October 2024 00:20

Morris Preorder Traversal of a Binary Tree

01 October 2024 00:20

Morris Inorder Traversal of a Binary Tree

01 October 2024 00:21

Flatten Binary Tree to LinkedList

01 October 2024 00:21

Summary

24 July 2025 17:13

Morris Traversal and Flatten Binary Tree is pending