DOMAIN WINTER WINNING CAMP

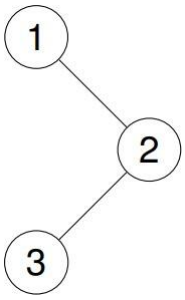
Student Name: Divya Sharma **UID:**22BCS14192
Branch: CSE **Section/Group:**22BCS_FL_IOT-603/B
Semester:5th

VERYEASY

1. Binary Tree Inorder Traversal

Given the root of a binary tree, return the inorder traversal of its nodes' values.

Example1:
Input:root=[1,null,2,3]
Output: [1,3,2]
Explanation:



CODE:

```
#include <iostream>
#include <vector>
using namespace std;

struct TreeNode {
    int val;TreeNode
    *left;
    TreeNode*right;
    TreeNode(intx):val(x),left(nullptr),right(nullptr){}
};

voidinorder(TreeNode*root,vector<int>&result){ if
    (!root) return;
    inorder(root->left, result);
    result.push_back(root->val);
    inorder(root->right, result);
}
```

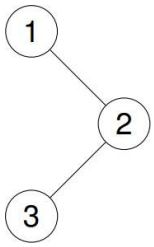
```
vector<int>inorderTraversal(TreeNode*root){
    vector<int> result;
    inorder(root,result);
    return result;
}

intmain() {
    // Create tree: [1, null, 2, 3]
    TreeNode*root=newTreeNode(1);
    root->right = new TreeNode(2);
    root->right->left=newTreeNode(3);

    vector<int>result=inorderTraversal(root);
    for (int val : result) {
        cout<<val<<"";
    }
    cout<<endl;

    return0;
}
```

```
1 3 2
```

2. Binary Tree Preorder Traversal

Given the root of a binary tree,return the preorder traversal of its nodes' values.

Example1:
Input:root=[1,null,2,3]
Output: [1,2,3]
Explanation:



CODE:

```
#include <iostream>
#include <vector>
usingnamespacestd;

struct TreeNode{
    int val;
    TreeNode*left;
```

```
    TreeNode*right;
    TreeNode(intx):val(x),left(nullptr),right(nullptr){}
};

voidpreorder(TreeNode*root,vector<int>&result){ if
    (!root) return;
    result.push_back(root->val);
    preorder(root->left, result);
    preorder(root->right,result);
}

vector<int>preorderTraversal(TreeNode*root){
    vector<int> result;
    preorder(root,result);
    return result;
}

intmain() {
    // Create tree: [1, null, 2, 3]
    TreeNode*root=newTreeNode(1);
    root->right = new TreeNode(2);
    root->right->left=newTreeNode(3);

    vector<int>result=preorderTraversal(root);
    for (int val : result) {
        cout<<val<<"";
    }
    cout<<endl;

    return0;
}
```

```
1 2 3
```

3. Binary Tree-Sum of All Nodes

Given the root of a binary tree, you need to find the sumo f all the node values in the binary tree.

Example1:

Input:root = [5,2, 6, 1,3, 4, 7]

Output:28

Explanation:The sum of all nodesis5+2+6+1+3+4+7=28.

CODE:

```
#include <iostream>
usingnamespacestd;
```

```
struct TreeNode {
   int val;TreeNode
   *left;
   TreeNode*right;
   TreeNode(intx):val(x),left(nullptr),right(nullptr){}
};

intsumOfAllNodes(TreeNode*root){ if
   (!root) return 0;
   returnroot->val+sumOfAllNodes(root->left)+sumOfAllNodes(root->right);
}

intmain() {
   // Create tree: [5, 2, 6, 1, 3, 4, 7]
   TreeNode*root=newTreeNode(5);
   root->left = new TreeNode(2);
   root->right = new TreeNode(6);
   root->left->left=newTreeNode(1);
   root->left->right = new TreeNode(3);
   root->right->left = new TreeNode(4);
   root->right->right=new TreeNode(7);

   cout<<"Sumof allnodes:"<<sumOfAllNodes(root)<<endl;

   return0;
}
```

```
Sum of all nodes: 28
```

Easy:

1. Same Tree

Two binary trees are considered the same if they are structurally identical,and the nodes have the same value.

Example1:

Input:p = [1,2,3],q = [1,2,3]

Output:true

CODE:

```
#include <iostream>
usingnamespacestd;

struct TreeNode {
   int val;
   TreeNode* left;
   TreeNode*right;
```

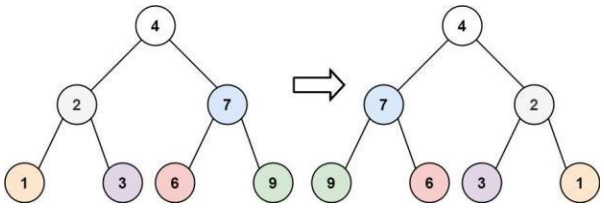
```
    TreeNode(intx):val(x),left(nullptr),right(nullptr){}
};
boolisSameTree(TreeNode*p,TreeNode*q){
    if(!p &&!q)returntrue; //Both treesare empty
    if(!p||!q||p->val!=q->val)returnfalse;//Structureorvaluemismatch return
    isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
}

intmain() {
    // Create tree p: [1, 2, 3]
    TreeNode*p=newTreeNode(1);
    p->left = new TreeNode(2);
    p->right=new TreeNode(3);

    // Create tree q: [1, 2, 3]
    TreeNode*q=newTreeNode(1);
    q->left = new TreeNode(2);
    q->right=new TreeNode(3);

    cout<<(isSameTree(p,q)?"true" :"false")<<endl;

    return0;
}.
```

```
true
```

2. Invert Binary Tree

Given the root of a binary tree, invert the tree,and return its root

Example1:



Input:root =[4,2,7,1,3,6,9]
Output:[4,7,2,9,6,3,1]
CODE:

```
#include <iostream>
usingnamespacestd;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode*right;
```

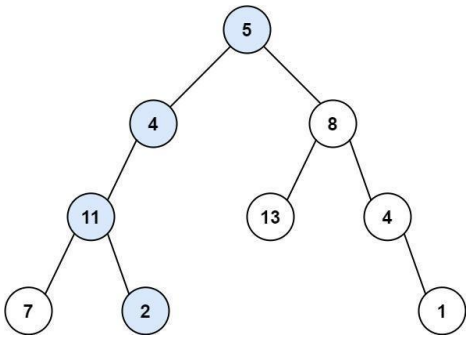
```
    TreeNode(intx):val(x),left(nullptr),right(nullptr){}
};

TreeNode*invertTree(TreeNode*root){ if
    (!root) return nullptr;
    swap(root->left,root->right);//Swapleftandrightchild
    invertTree(root->left);
    invertTree(root->right);
    return root;
}

voidpreorderTraversal(TreeNode*root){
    if (!root) return;
    cout << root->val <<"";
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}

intmain() {
    // Create tree: [4, 2, 7, 1, 3, 6, 9]
    TreeNode*root=newTreeNode(4);
    root->left = new TreeNode(2);
    root->right = new TreeNode(7);
    root->left->left=newTreeNode(1);
    root->left->right = new TreeNode(3);
    root->right->left = new TreeNode(6);
    root->right->right=new TreeNode(9);

    root = invertTree(root);

    preorderTraversal(root);//Expectedoutput:4796231 cout <<
    endl;

    return0;
}
```

```
4 7 9 6 2 3 1
```

3. PathSum

Givenabinarytreeandasum,returntrueifthetreehasaroot-to-leafpath suchthataddingupallthe values along the path equals the given sum. Return false if no such path can be found.

Example1:



Input:root=[5,4,8,11,null,13,4,7,2,null,null,null,1],targetSum=22
Output: true
Explanation:Theroot-to-leafpathwiththetargetsumisshown. CODE:

```
#include <iostream>
usingnamespacestd;

struct TreeNode {
   int val;
   TreeNode* left;
   TreeNode*right;
   TreeNode(intx):val(x),left(nullptr),right(nullptr){}
};

boolhasPathSum(TreeNode*root,inttargetSum){ if
   (!root) return false;
   if(!root->left&& !root->right)return root->val==targetSum;//Checkifleaf
   returnhasPathSum(root->left,targetSum-root->val)||hasPathSum(root->right,targetSum-
root->val);
}

intmain() {
   //Createtree:[5,4,8,11,null,13,4,7,2,null,null,null,1]
   TreeNode* root = new TreeNode(5);
   root->left = new TreeNode(4);
   root->right=newTreeNode(8);
   root->left->left = new TreeNode(11);
   root->left->left->left=newTreeNode(7);
   root->left->left->right=newTreeNode(2);
   root->right->left = new TreeNode(13);
   root->right->right = new TreeNode(4);
   root->right->right->right=newTreeNode(1);

   inttargetSum= 22;
   cout<<(hasPathSum(root,targetSum)?"true":"false")<<endl; return

   0;
```

}

```
true
```

Medium:

1. ConstructBinaryTreefromPreorderandInorderTraversal

Giventwointegerarrayspreorderandinorderwherepreorderisthepreordertraversalofabinarytree and inorder is the inorder traversal of the same tree, construct and return the binary tree.

Example1:

Input:preorder=[3,9,20,15,7],inorder=[9,3,15,20,7]

Output: [3,9,20,null,null,15,7]

CODE:

```
#include <iostream>
#include<unordered_map>
#include <vector>
usingnamespacestd;

struct TreeNode {
   int val;
   TreeNode* left;
   TreeNode*right;
   TreeNode(intx):val(x),left(nullptr),right(nullptr){}
};

TreeNode*buildTreeHelper(vector<int>&preorder,intpreStart,intpreEnd,
                    vector<int>& inorder, int inStart, int inEnd,
                    unordered_map<int, int>& inMap) {
   if(preStart>preEnd||inStart>inEnd)returnnullptr;

   TreeNode*root=newTreeNode(preorder[preStart]); int
   inRoot = inMap[root->val];
   intnumsLeft =inRoot-inStart;

   root->left=buildTreeHelper(preorder,preStart+1,preStart+numsLeft,inorder,inStart,inRoot - 1,
inMap);
   root->right=buildTreeHelper(preorder,preStart+numsLeft+1,preEnd,inorder,inRoot+1, inEnd,
inMap);

   return root;
}

TreeNode*buildTree(vector<int>&preorder,vector<int>&inorder){
   unordered_map<int, int> inMap;
   for(int i=0; i<inorder.size(); i++)inMap[inorder[i]] =i;
   returnbuildTreeHelper(preorder,0,preorder.size()-1,inorder,0,inorder.size()-1,inMap);
```

```
}

voidprintInorder(TreeNode*root){
    if (!root) return;
    printInorder(root->left);
    cout<<root->val<<"";
    printInorder(root->right);
}

intmain() {
    vector<int>preorder= {3,9, 20,15, 7};
    vector<int>inorder= {9,3, 15,20, 7};

    TreeNode*root=buildTree(preorder,inorder);

    printInorder(root); // Output: 9 3 15 20 7
    cout<<endl;

    return0;
}
```

```
9 3 15 20 7
```

2. ConstructBinaryTreefromInorderandPostorderTraversal

Giventwointegerarraysinorderandpostorderwhereinorderistheinordertraversalofabinarytree and postorder is the postorder traversal of the same tree, construct and return the binary tree.

Example1:
Input:inorder=[9,3,15,20,7],postorder=[9,15,7,20,3]
Output: [3,9,20,null,null,15,7]
CODE:

```
#include <iostream>
#include<unordered_map>
#include <vector>
usingnamespacestd;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode*right;
    TreeNode(intx):val(x),left(nullptr),right(nullptr){}};
TreeNode*buildTreeHelper(vector<int>&inorder,intinStart,intinEnd,
                         vector<int>& postorder, int postStart, int postEnd,
                         unordered_map<int, int>& inMap) {
```

```
    if(inStart > inEnd||postStart >postEnd)returnnullptr;

    TreeNode*root=newTreeNode(postorder[postEnd]); int
    inRoot = inMap[root->val];
    intnumsLeft =inRoot-inStart;

    root->left=buildTreeHelper(inorder,inStart,inRoot-1,postorder,postStart,postStart+numsLeft
-1, inMap);
    root->right=buildTreeHelper(inorder,inRoot+1,inEnd,postorder,postStart+numsLeft, postEnd - 1,
inMap);

    return root;
}

TreeNode*buildTree(vector<int>&inorder,vector<int>&postorder){
    unordered_map<int, int> inMap;
    for(int i=0; i<inorder.size(); i++)inMap[inorder[i]] =i;

    returnbuildTreeHelper(inorder,0,inorder.size() -1,postorder,0,postorder.size()-1,inMap);
}

voidprintInorder(TreeNode*root){
    if (!root) return;
    printInorder(root->left);
    cout<<root->val<<"";
    printInorder(root->right);
}

intmain() {
    vector<int>inorder= {9,3, 15,20, 7};
    vector<int> postorder = {9, 15, 7, 20, 3};

    TreeNode*root=buildTree(inorder,postorder);

    printInorder(root);//Output:9315207 cout
    << endl;

    return0;
}
```

```
9 3 15 20 7
```

3. SumRoottoLeafNumbers

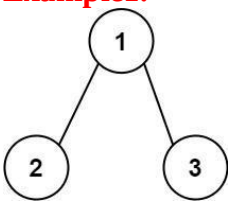
Youaregiventheroot of abinarytreecontainingdigits from0 to9 only.

Eachroot-to-leafpathinthetreerepresentsanumber.

Forexample,theroot-to-leafpath1->2->3representsthenumber123.
Returnthetotalsumofallroot-to-leafnumbers.Testcasesaregeneratedsothattheanswerwillfitin a 32-bit integer.

Aleaf nodeisanodewith no children.

Example1:



Input:root=[1,2,3]Output:25
Explanation:
Theroot-to-leafpath1->2representsthenumber12.
Theroot-to-leafpath1->3representsthenumber13.
Therefore, sum = 12 + 13 = 25.
CODE:

```
#include <iostream>
usingnamespacestd;

struct TreeNode {
   int val;
   TreeNode* left;
   TreeNode*right;
   TreeNode(intx):val(x),left(nullptr),right(nullptr){}
};

intsumNumbersHelper(TreeNode*root,intcurrentSum){ if
   (!root) return 0;
   currentSum=currentSum *10+ root->val;
   if(!root->left&&!root->right)returncurrentSum;//Leafnode
   returnsumNumbersHelper(root->left,currentSum)+sumNumbersHelper(root->right,
currentSum);
}

int sumNumbers(TreeNode* root) {
   returnsumNumbersHelper(root,0);
}

intmain() {
   TreeNode*root=newTreeNode(1);
   root->left = new TreeNode(2);
   root->right=new TreeNode(3);
```

```
    cout<<sumNumbers(root)<< endl; // Output:25

    return0;
}
```

```
25
```

Hard:

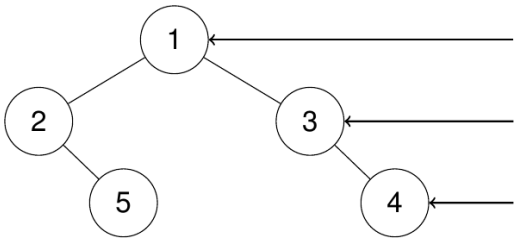
1. BinaryTreeRightSide View

Giventherootofabinarytree,imagineyourselfstandingontherightsideofit,returnthevaluesof the nodes you can see ordered from top to bottom.

Example1:

Input:root=[1,2,3,null,5,null,4]

Output: [1,3,4]



Explanation:

CODE:

```
#include <iostream>
#include <vector>
#include <queue>
usingnamespacestd;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode*right;
    TreeNode(intx):val(x),left(nullptr),right(nullptr){}
};

vector<int>rightSideView(TreeNode*root){
    vector<int> result;
    if(!root)returnresult;
```

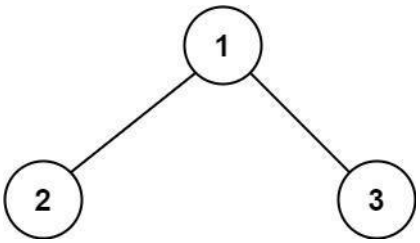
```
    queue<TreeNode*>q;
    q.push(root);

    while(!q.empty()){
        intlevelSize=q.size();
        for(inti=0;i<levelSize;i++){
            TreeNode*current=q.front();
            q.pop();

            // Add the last node of the current level to the
            resultif(i==levelSize-1)result.push_back(current-
            >val);

            if(current->left)q.push(current->left);
            if(current->right)q.push(current->right);
        }
    }
    returnresult;
}

intmain() {
    TreeNode*root=newTreeNode(1);
    root->left = new TreeNode(2);
    root->right=new TreeNode(3);
    root->left->right = new TreeNode(5);
    root->right->right=newTreeNode(4);

    vector<int>result=rightSideView(root);
    for (int val : result) {
        cout<<val<<"";
    }
    // Output: 1 3 4
    return0;
}
```

```
1 3 4
```

2. **BinaryTreeMaximumPathSum**

Apathinabinarytreeis asequenceofnodeswhereeachpairofadjacentnodesinthesequencehas an edge connecting them. A node can only appear in the sequence at most once. Note that the path does not need to pass through the root.
Thepath sum of apathisthe sum of thenode's values in thepath.
Giventhe root of abinarytree, return the maximum pathsum of any non-empty path.

Example1:



Input:root=[1,2,3]
Output: 6
Explanation:Theoptimalpathis2 ->1->3withapathsumof2+1+3=6. CODE:

```
#include <iostream>
#include <climits>
usingnamespacestd;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode*right;
    TreeNode(intx):val(x),left(nullptr),right(nullptr){}
};

intmaxPathSumHelper(TreeNode*root,int&maxSum){ if
    (!root) return 0;

    //Computemaximumpathsums forleftand right subtrees
    int leftMax = max(0, maxPathSumHelper(root->left,
    maxSum));intrightMax=max(0,maxPathSumHelper(root-
    >right,maxSum));

    //Updatetheoverallmaximumpath sum
    maxSum=max(maxSum, leftMax+rightMax +root->val);

    //Returnthemaximumpathsumincludingthecurrentnode
    return max(leftMax, rightMax) + root->val;
}

int maxPathSum(TreeNode* root) {int
    maxSum = INT_MIN;
    maxPathSumHelper(root,maxSum);
    return maxSum;
}

intmain() {
    TreeNode*root=newTreeNode(1);
    root->left = new TreeNode(2);
    root->right=new TreeNode(3);

    cout<<maxPathSum(root)<< endl; // Output:6
```

```
    return0;
}
```

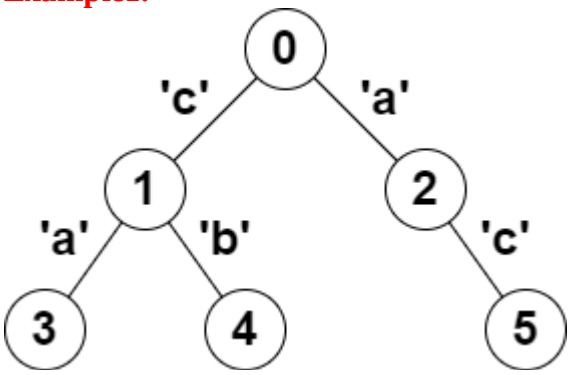
6

Very Hard:

1. **CountPathsThatCanFormaPalindromeinaTree**

Youaregivenatree(i.e. aconnected,undirectedgraphthathasnocycles)rootedatnode0consisting of n nodes numbered from 0 to n - 1. The tree is represented by a 0-indexed array parent of size n, where parent[i] is the parent of node i. Since node 0 is the root, parent[0] == -1.
Youarealsogivenastringsoflengthn,wheres[i]isthecharacterassignedtotheedge between i and parent[i]. s[0] can be ignored.
Returnthenumberofpairsofnodes(u,v)suchthatu<vandthecharactersassignedtoedgesonthe path from u to v can be rearranged to form a palindrome.
Astringis apalindromewhenit readsthesamebackwardsas forwards.

Example1:



Input:parent=[-1,0,0,1,1,2],s="acaabc" Output: 8
Explanation:Thevalidpairs are:
- Allthepairs(0,1),(0,2),(1,3),(1,4)and(2,5)resultinonecharacterwhichisalwaysa palindrome.
- Thepair(2,3)resultinthe string"aca"whichisa palindrome.
- Thepair(1,5)resultinthestring"cac"whichisa palindrome.
- Thepair(3,5)resultinthestring"acac"whichcanberearrangedintothepalindrome"acca". CODE:

```
#include<iostream>
#include <vector>
#include<unordered_map>
#include <unordered_set>
using namespace std;

//Helperfunctiontoperform DFSandcountpalindromic paths
voiddfs(intnode,intmask,constvector<vector<int>>&graph,conststring&s,unordered_map<int, int>&
count, int& result) {
```

```
    //Updateresultbasedonthecurrentmask result
    += count[mask];
    for(int i =0; i <26; ++i) {
        result+=count[mask^ (1<<i)];
    }

    //Incrementthecountforthecurrentmask
    count[mask]++;

    //Recurforchildnodes
    for(intchild :graph[node]){
        dfs(child,mask^(1<<(s[child]-'a')), graph,s,count, result);
    }

    //Decrementthecounttobacktrack
    count[mask]--;
}

intcountPalindromePaths(vector<int>&parent,strings){ int
    n = parent.size();
    vector<vector<int>>graph(n);
    for (int i = 1; i < n; ++i) {
        graph[parent[i]].push_back(i);
    }

    unordered_map<int,int> count;
    count[0]=1;//Initialmask(nocharacters) int
    result = 0;
    dfs(0,0,graph,s,count,result); return
    result;
}

intmain() {
    vector<int>parent={-1,0,0,1,1,2};
    string s = "acaabc";
    cout<<countPalindromePaths(parent,s)<<endl;//Output:8 return 0;
}
```

8

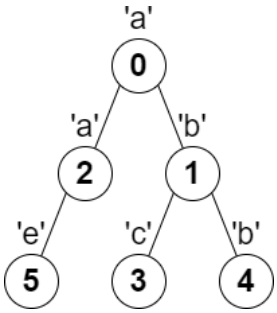
2. LongestPathWithDifferentAdjacentCharacters

Youaregivenatree(i.e. aconnected,undirected graph that has no cycles) rootedat node0consisting ofnnodes numbered from0ton - 1. The tree is represented by a0-indexedarrayparentof sizen, where parent[i] is the parent of node i. Since node 0 is the root, parent[0] == -1.

Youare alsogiven astring s oflength n, wheres[i] isthecharacter assignedto nodei.

Returnthelengthofthelongestpathinthetreesuchthatnopairofadjacentnodesonthepathhavethe same character assigned to them.

Example1:



Input:parent=[-1,0,0,1,1,2],s="abacbe"
Output: 3
Explanation: The longest path where each two adjacent nodes have different characters in the tree is the path: 0 -> 1 -> 3. The length of this path is 3, so 3 is returned. It can be proven that there is no longer path that satisfies the conditions.

CODE:

```
fromcollectionsimportdefaultdict

deflongestPath(parent,s): n
   = len(parent)
   adj= defaultdict(list)

   #Buildtheadjacencylistofthetree for i
   in range(1, n):
      adj[parent[i]].append(i)

   #Thiswillstorethelongestpathstartingfromeachnode longest =
   [0] * n

   defdfs(node):
      first_max,second_max =0, 0

      #Exploreallthechildrenofthecurrentnode for
      child in adj[node]:
         child_path=dfs(child)

         #Onlyconsiderthechild'spathifthecharactersaredifferent if
         s[child] != s[node]:
            if child_path > first_max:
               second_max=first_max
               first_max = child_path
            elifchild_path>second_max:
               second_max = child_path

      #Thelongestpaththatpassesthroughthisnodeisthesumoffirst_maxandsecond_max+1 (for the
current node itself)
      longest[node]=first_max+1
```

```
        #Returnthelengthofthelongestpathforthesubtreerootedatthisnode return
        first_max + 1

    #StartDFSfromtheroot(node0) dfs(0)

    #Theansweristhelongestpathinthetree return
    max(longest)

#Example usage
parent= [-1,0, 0,1,1, 2]
s ="abacbe"
print(longestPath(parent, s))#Output:3
```

```
3
```