DAY-6

NAME-Daksh

UID-22BCS15372

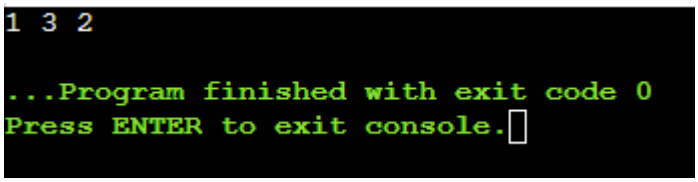
DATE-26/12/2024

Question1

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

struct TreeNode { int val;
    TreeNode* left;
    TreeNode*right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {} TreeNode(int x) : val(x),
    left(nullptr), right(nullptr) {}
    TreeNode(intx,TreeNode*left,TreeNode*right):val(x),
left(left),right(right){}
};


voidinorderTraversalHelper(TreeNode*root,vector<int>& result) {
    if(root==nullptr){ return;
    }
    inorderTraversalHelper(root->left,result);
```

```
    result.push_back(root->val); inorderTraversalHelper(root-
    >right,result);
}


vector<int>inorderTraversal(TreeNode*root){ vector<int>
    result; inorderTraversalHelper(root, result);
    returnresult;
}
intmain(){
     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<<"";
    }
deleteroot->right->left; delete
    root->right; delete root;

    return0;
}
```

Output:

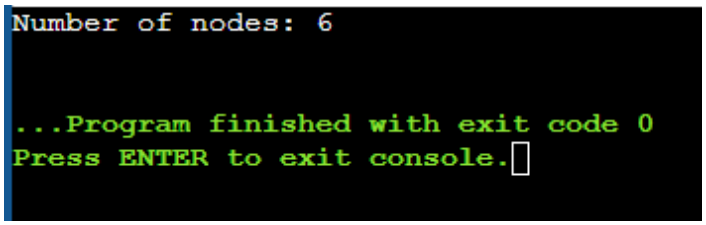


Question2

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

struct TreeNode { int val;
    TreeNode* left;
    TreeNode*right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {} TreeNode(int x) : val(x),
    left(nullptr), right(nullptr) {}
    TreeNode(intx,TreeNode*left,TreeNode*right):val(x),
left(left),right(right){}
};

intcomputeDepth(TreeNode*node){ int depth = 0;
    while(node){ depth++;
        node=node->left;
    }
    returndepth;
}

intcountNodes(TreeNode*root){ if (!root)
    return 0;

    int leftDepth = computeDepth(root->left);
    intrightDepth=computeDepth(root->right);

    if(leftDepth==rightDepth) {

        return(1<<leftDepth)+countNodes(root->right);
```

```
    }else{
        return(1<<rightDepth)+countNodes(root->left);
    }
}

intmain(){
    //Exampleusage:
    TreeNode*root=newTreeNode(1); root->left =
    new TreeNode(2);
    root->right = new TreeNode(3); root->left-
    >left=newTreeNode(4);
    root->left->right=newTreeNode(5); root->right-
    >left=newTreeNode(6);

    cout<<"Numberofnodes:"<<countNodes(root)<<endl; delete root->left->left;
    deleteroot->left->right;
    deleteroot->right->left; delete
    root->left;delete root->right;
    delete root;

    return0;
}
```

Output:

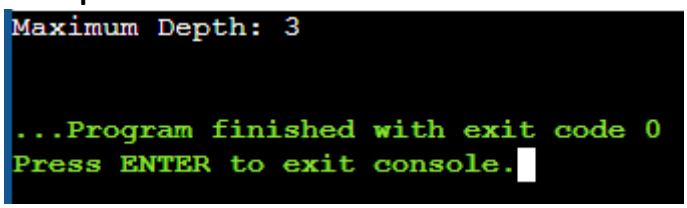


Question3

```
#include <iostream>
usingnamespacestd;

struct TreeNode { int val;
    TreeNode* left;
    TreeNode*right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {} TreeNode(int x) : val(x),
    left(nullptr), right(nullptr) {}
    TreeNode(intx,TreeNode*left,TreeNode*right):val(x),
left(left),right(right){}
};

intmaxDepth(TreeNode*root){ if (!root)
    return 0;
    int leftDepth = maxDepth(root->left);
    intrightDepth=maxDepth(root->right); return
    max(leftDepth, rightDepth) + 1;
}

intmain(){

    TreeNode*root=newTreeNode(1); root->left =
    new TreeNode(2);
    root->right = new TreeNode(3); root->left-
    >left=newTreeNode(4);
    root->left->right = new TreeNode(5); root->right-
    >right=newTreeNode(6);

    cout<<"MaximumDepth:"<<maxDepth(root)<<endl; delete root->left->left;
```

```
    delete root->left->right;
    deleteroot->right->right; delete
    root->left;
    deleteroot->right; delete
    root;

    return0;
}
```

Output:



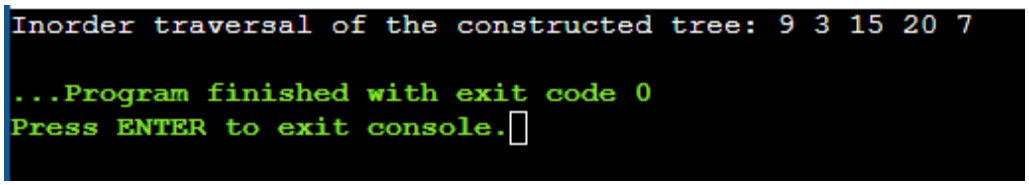
Question4

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

struct TreeNode { int val;
    TreeNode* left;
    TreeNode*right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {} TreeNode(int x) : val(x),
    left(nullptr), right(nullptr) {}
    TreeNode(intx,TreeNode*left,TreeNode*right):val(x),
left(left),right(right){}
};

TreeNode*buildTreeHelper(vector<int>&preorder,int preStart, int preEnd,
```

```
                        vector<int>& inorder, int inStart, int inEnd,
                        unordered_map<int,int>&inorderIndexMap){
    if(preStart>preEnd||inStart>inEnd){ return nullptr;
    }

    int rootVal = preorder[preStart];
    TreeNode*root=newTreeNode(rootVal);

    introotIndex=inorderIndexMap[rootVal]; int

    leftSubtreeSize = rootIndex - inStart;

    root->left=buildTreeHelper(preorder,preStart+1,preStart
+leftSubtreeSize,
                               inorder,inStart,rootIndex-1, inorderIndexMap);
    root->right=buildTreeHelper(preorder,preStart+ leftSubtreeSize + 1, preEnd,
                                inorder,rootIndex+1,inEnd,
inorderIndexMap);

    returnroot;
}


TreeNode*buildTree(vector<int>&preorder,vector<int>& inorder) {

    unordered_map<int,int>inorderIndexMap; for (int i = 0; i <
    inorder.size(); i++) {
        inorderIndexMap[inorder[i]]=i;
    }

    returnbuildTreeHelper(preorder,0,preorder.size()-1,
```

```
                        inorder,0,inorder.size()-1,
                        inorderIndexMap);
}

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);

    cout<<"Inordertraversaloftheconstructedtree:"; printInorder(root);

    return0;
}
```

Output:

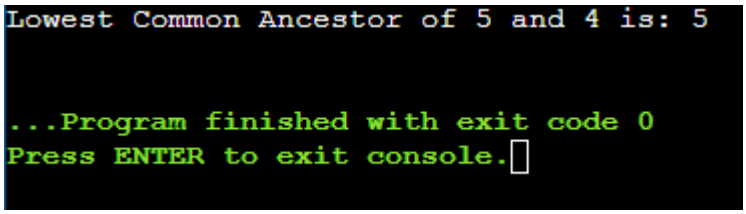


Question5

```
#include <iostream>
usingnamespacestd; struct
TreeNode {
    int val; TreeNode* left;
    TreeNode*right;
    TreeNode(intx):val(x),left(nullptr),right(nullptr){}
};
TreeNode*lowestCommonAncestor(TreeNode*root, TreeNode* p, TreeNode* q) {
    if(!root)returnnullptr;
    if(root==p||root==q)return root;
    TreeNode* left = lowestCommonAncestor(root->left, p, q);
    TreeNode*right=lowestCommonAncestor(root->right,p,q);

    if(left&&right){ return root;
    }

    returnleft?left:right;
}
intmain(){
    TreeNode*root=newTreeNode(3);  root->left  =
    new TreeNode(5);
    root->right  =  new  TreeNode(1);  root->left-
    >left=newTreeNode(6);
    root->left->right  =  new  TreeNode(2);  root->right-
    >left  =  new  TreeNode(0);  root->right-
    >right=newTreeNode(8);
    root->left->right->left  =  new  TreeNode(7);  root->left-
    >right->right=newTreeNode(4);
```

```
    TreeNode*p=root->left;
    TreeNode*q=root->left->right->right;

    TreeNode*lca=lowestCommonAncestor(root,p,q); if (lca) {
        cout<<"LowestCommonAncestorof"<<p->val<<"and "<< q->val <<" is: "<< lca-
>val << endl;
    }else{
        cout<<"Nocommonancestorfound."<<endl;
    }

    return0;
}
```

Output:



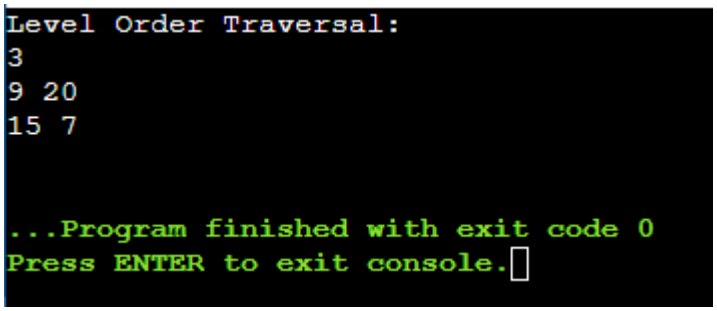
Question6

```
#include <iostream> #include
<vector> #include <queue>
usingnamespacestd; struct
TreeNode {
    int val; TreeNode* left;
    TreeNode*right;
    TreeNode(intx):val(x),left(nullptr),right(nullptr){}
};
vector<vector<int>>levelOrder(TreeNode*root){
```

```
    vector<vector<int>>result; if (!root)
    return result; queue<TreeNode*> q;
    q.push(root);
    while(!q.empty()){
        int levelSize = q.size();
        vector<int>currentLevel;

        for (int i = 0; i < levelSize; i++) { TreeNode* node =
            q.front(); q.pop(); currentLevel.push_back(node-
            >val); if (node->left) q.push(node->left);
            if(node->right)q.push(node->right);
        }

        result.push_back(currentLevel);
    }

    returnresult;
}

intmain(){
    TreeNode*root=newTreeNode(3); root->left =
    new TreeNode(9);
    root->right=newTreeNode(20);
    root->right->left=newTreeNode(15); root->right-
    >right=newTreeNode(7);


    vector<vector<int>>traversal=levelOrder(root); cout <<"Level


    Order Traversal:"<< endl;
    for(constauto&level:traversal){ for (int val :
        level) {
            cout<<val<<"";
```

```
        }
        cout<<endl;
    }

    return0;
}
```

Output:

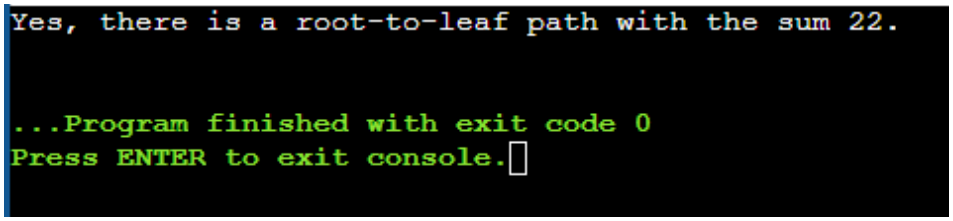


Question7

```
#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) { returnroot-
        >val==targetSum;
    }
    int remainingSum = targetSum - root-
    >val;returnhasPathSum(root->left,remainingSum)||
hasPathSum(root->right,remainingSum);
```

```
}

intmain(){
    TreeNode*root=newTreeNode(5); root->left =
    new TreeNode(4);
    root->right=newTreeNode(8);
    root->left->left = new TreeNode(11); root->right->left
    = new TreeNode(13); root->right->right = new
    TreeNode(4); root->left->left->left=newTreeNode(7);
    root->left->left->right = new TreeNode(2); root->right-
    >right->right=newTreeNode(1);

    inttargetSum=22;
    if(hasPathSum(root,targetSum)){
        cout<<"Yes,thereisaroot-to-leafpathwiththesum"<< targetSum <<"."<< endl;
    }else{
        cout<<"No,thereisnoroot-to-leafpathwiththesum"<< targetSum <<"."<< endl;
    }
    return0;
}
```

Output:



Question8

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

classUnionFind{ public:
    vector<int>parent,rank;

    UnionFind(intn):parent(n),rank(n,0){ for (int i = 0; i <
        n; ++i) {
            parent[i]=i;
        }
    }

    intfind(intx){
        if(x!=parent[x]){
            parent[x]=find(parent[x]);
        }
        returnparent[x];
    }

    voidunite(intx,inty){ int rootX
        = find(x); int rootY = find(y);
        if(rootX!=rootY){
            if(rank[rootX]>rank[rootY]){ parent[rootY] = rootX;
            }elseif(rank[rootX]<rank[rootY]){ parent[rootX] =
                rootY;
            }else{
```

```
            parent[rootY]=rootX; rank[rootX]++;
        }
    }
}
};

intnumberOfGoodPaths(vector<int>&vals,
vector<vector<int>>& edges) {
    int n = vals.size(); vector<vector<int>>adj(n);
    for (const auto& edge : edges) {
        adj[edge[0]].push_back(edge[1]);
        adj[edge[1]].push_back(edge[0]);
    }
    vector<int> sortedNodes(n);
    iota(sortedNodes.begin(),sortedNodes.end(),0);
    sort(sortedNodes.begin(),sortedNodes.end(),[&](inta,intb)
{
        returnvals[a]<vals[b];
    });

    UnionFind uf(n);
    unordered_map<int,int>count; int
    goodPaths = 0;

    for (int node : sortedNodes) {
        intnodeValue=vals[node];

        count[nodeValue]++; goodPaths++;

        for(intneighbor:adj[node]){
            if(vals[neighbor]<=nodeValue){ uf.unite(node,
                neighbor);
```

```
            }
        }
        unordered_map<int,int>componentCount; for (int neighbor :
        adj[node]) {
            if(vals[neighbor]<=nodeValue){ int root =
                uf.find(neighbor);
                componentCount[root]++;
            }
        }

        for(auto&[_,size]:componentCount){ goodPaths += size *
            (size - 1) / 2;
        }
    }

    returngoodPaths;
}

intmain(){
    vector<int>vals={1,3,2,1, 3};
    vector<vector<int>>edges={{0,1},{0,2},{2,3},{2,4}};

    cout <<"Number of good paths: "<<
numberOfGoodPaths(vals,edges)<<endl;

    return0;
}
```

Output:

Input:

```plaintext
vals = [1, 3, 2, 1, 3]
edges = [[0, 1], [0, 2], [2, 3], [2, 4]]
```

Output:

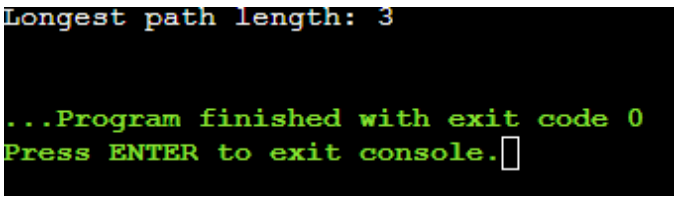
```plaintext
Number of good paths: 6
```

Question9

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

intdfs(intnode,constvector<vector<int>>&adj,conststring& s, int& maxPathLength)
{
    intlongest=0,secondLongest= 0;

    for(intneighbor:adj[node]){
        intchildPath=dfs(neighbor,adj,s,maxPathLength); if (s[node] !=
        s[neighbor]) {
            if (childPath > longest) {
                secondLongest=longest; longest =
                childPath;
            }elseif(childPath>secondLongest){ secondLongest =
                childPath;
            }
        }
```

```
    }
    maxPathLength=max(maxPathLength,longest+ secondLongest + 1);

    returnlongest+1;
}

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

    intmaxPathLength=0;
    dfs(0,adj,s,maxPathLength);

    returnmaxPathLength;
}

intmain(){
    vector<int>parent={-1,0,0,1,1,2}; string s =
    "abacbe";

    cout<<"Longestpathlength:"<<longestPath(parent,s)<<endl;
    return0;
}
```

Output:



Question10

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

classSolution{ public:
    intmaxComponents=0;
    intdfs(intnode,constvector<vector<int>>&adj,const vector<int>& values,
vector<bool>& visited, int k) {
        visited[node]=true;
        intsubtreeSum=values[node]; for (int
        neighbor : adj[node]) {
            if(!visited[neighbor]){
                subtreeSum+=dfs(neighbor,adj,values,visited,k);
            }
        }
        if(subtreeSum%k==0){
            maxComponents++; return 0;
        }

        returnsubtreeSum;
    }

    intcomponentValue(intn,vector<vector<int>>&edges, vector<int>& values, int k) {
        vector<vector<int>>adj(n);
        for (const auto& edge : edges) {
            adj[edge[0]].push_back(edge[1]);
            adj[edge[1]].push_back(edge[0]);
        }
```

```
        vector<bool>visited(n,false);
        inttotalSum=dfs(0,adj,values,visited,k); if (totalSum % k
        == 0) {
            maxComponents++;
        }

        returnmaxComponents;
    }
};

int main() {
    Solutionsol; int n =
    5;
    vector<vector<int>>edges={{0,1},{1,2},{1,3},{3,4}};
    vector<int>values={1,2,3,4,5}; int k = 3;

    intresult=sol.componentValue(n,edges,values,k);
    cout<<"Maximumnumberofcomponents:"<<result<<endl;

    return0;
}
```

Output: