DAY-6

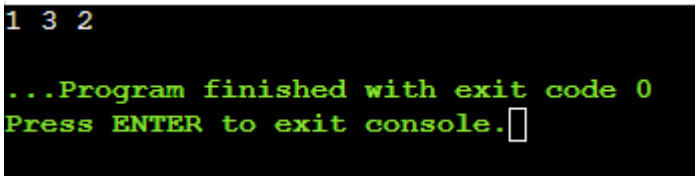
NAME_Darshan_Maurya
UID_22BCS15280
DATE_26/12/2024

Question 1

```
#include <iostream>
#include <vector>
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(int x, TreeNode* left, TreeNode* right) : val(x),
left(left), right(right) {}
};


void inorderTraversalHelper(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);
    return result;
}
int main() {
     TreeNode* root = new TreeNode(1);
    root->right = new TreeNode(2);
    root->right->left = new TreeNode(3);

    vector<int> result = inorderTraversal(root);
 for (int val : result) {
        cout << val << " ";
    }
delete root->right->left;
    delete root->right;
    delete root;

    return 0;
}
```

Output:

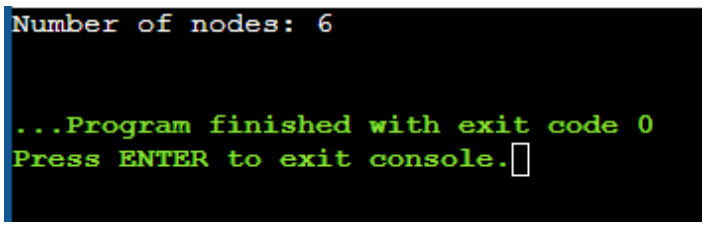


Question 2

```
#include <iostream>
#include <cmath>
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(int x, TreeNode* left, TreeNode* right) : val(x),
left(left), right(right) {}
};

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

int countNodes(TreeNode* root) {
    if (!root) return 0;

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

    if (leftDepth == rightDepth) {

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

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

int main() {
    // Example usage:
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);

    cout << "Number of nodes: " << countNodes(root) << endl;
    delete root->left->left;
    delete root->left->right;
    delete root->right->left;
    delete root->left;
    delete root->right;
    delete root;

    return 0;
}
```

Output:

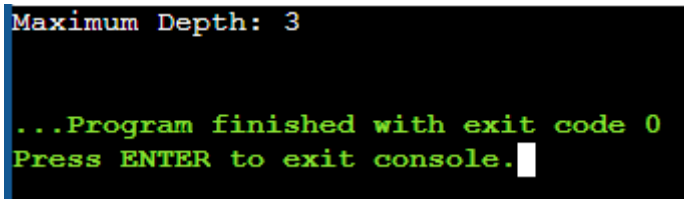


Question 3

```
#include <iostream>
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(int x, TreeNode* left, TreeNode* right) : val(x),
left(left), right(right) {}
};

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

int main() {

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

    cout << "Maximum Depth: " << maxDepth(root) << endl;

    delete root->left->left;
```

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

    return 0;
}
```

Output:



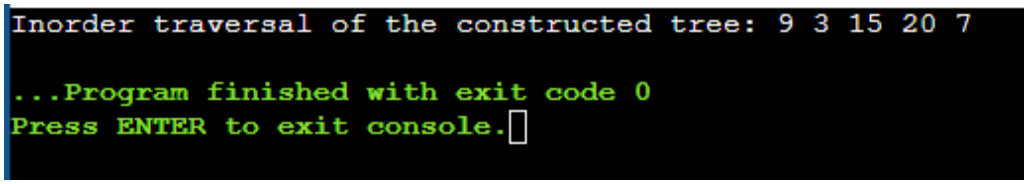
Question 4

```
#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(int x, 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 = new TreeNode(rootVal);

    int rootIndex = 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);

    return root;
}


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;
    }

    return buildTreeHelper(preorder, 0, preorder.size() - 1,
```

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

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

int main() {

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

    TreeNode* root = buildTree(preorder, inorder);

    cout << "Inorder traversal of the constructed tree: ";
    printInorder(root);

    return 0;
}
```

Output:

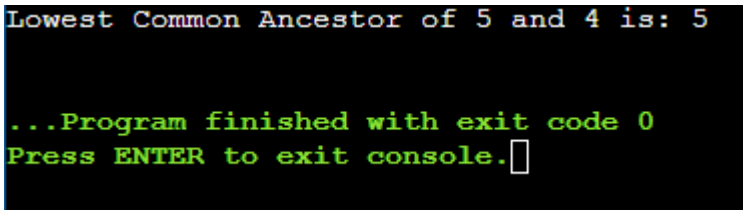


Question 5

```
#include <iostream>
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
TreeNode* lowestCommonAncestor(TreeNode* root,
TreeNode* p, TreeNode* q) {
    if (!root) return nullptr;
    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;
    }

    return left ? left : right;
}
int main() {
    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);
```

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

    TreeNode* lca = lowestCommonAncestor(root, p, q);
    if (lca) {
        cout << "Lowest Common Ancestor of " << p->val << " and
" << q->val << " is: " << lca->val << endl;
    } else {
        cout << "No common ancestor found." << endl;
    }

    return 0;
}
```

Output:



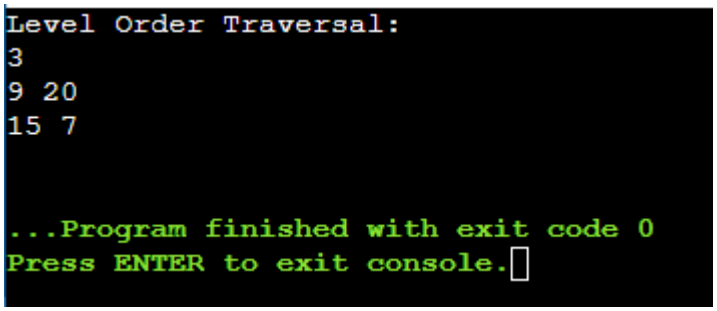
Question 6

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : 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);
    }

    return result;
}

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

    vector<vector<int>> traversal = levelOrder(root);

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

```
        }
        cout << endl;
    }

    return 0;
}
```

Output:

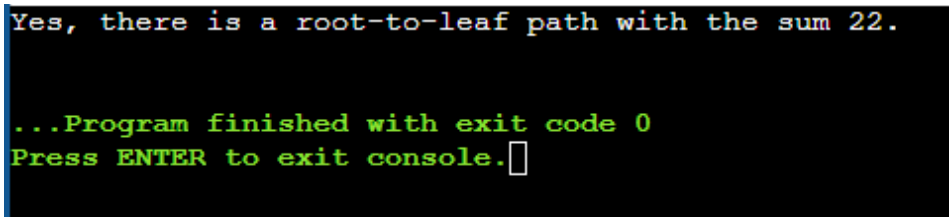


Question 7

```
#include <iostream>
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
bool hasPathSum(TreeNode* root, int targetSum) {
    if (!root) return false;
    if (!root->left && !root->right) {
        return root->val == targetSum;
    }
    int remainingSum = targetSum - root->val;
    return hasPathSum(root->left, remainingSum) ||
hasPathSum(root->right, remainingSum);
```

```
}

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

    int targetSum = 22;
    if (hasPathSum(root, targetSum)) {
        cout << "Yes, there is a root-to-leaf path with the sum " <<
targetSum << "." << endl;
    } else {
        cout << "No, there is no root-to-leaf path with the sum " <<
targetSum << "." << endl;
    }
    return 0;
}
```

Output:



Question 8

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

class UnionFind {
public:
    vector<int> parent, rank;

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

    int find(int x) {
        if (x != parent[x]) {
            parent[x] = find(parent[x]);
        }
        return parent[x];
    }

    void unite(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);
        if (rootX != rootY) {
            if (rank[rootX] > rank[rootY]) {
                parent[rootY] = rootX;
            } else if (rank[rootX] < rank[rootY]) {
                parent[rootX] = rootY;
            } else {
```

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

int numberOfGoodPaths(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(), [&](int a, int b)
{
    return vals[a] < vals[b];
  });

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

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

    count[nodeValue]++;
    goodPaths++;

    for (int neighbor : 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;
        }
    }

    return goodPaths;
}

int main() {
    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;

    return 0;
}
```

Output:

Input:

plaintext

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

Output:

plaintext

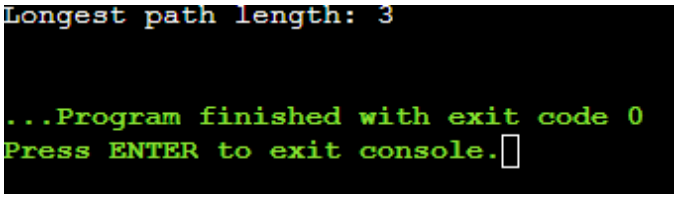
```
Number of good paths: 6
```

Question 9

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

int dfs(int node, const vector<vector<int>>& adj, const string&
s, int& maxPathLength) {
    int longest = 0, secondLongest = 0;

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

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

    return longest + 1;
}

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

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

    return maxPathLength;
}

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

    cout << "Longest path length: " << longestPath(parent, s) <<
endl;
    return 0;
}
```

Output:



Question 10

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

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

        return subtreeSum;
    }

    int componentValue(int n, 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);
        int totalSum = dfs(0, adj, values, visited, k);
        if (totalSum % k == 0) {
            maxComponents++;
        }

        return maxComponents;
    }
};

int main() {
    Solution sol;
    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;

    int result = sol.componentValue(n, edges, values, k);
    cout << "Maximum number of components: " << result <<
endl;

    return 0;
}
```

Output: