Day 6

NAME: -Shivam Yadav

UID: -22BCS15259

DATE: -27/12/24

Section: - 620-A

Question 1: -

Binary Tree Inorder Traversal

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

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

class Solution {
public:
    void inorderHelper(TreeNode* root, vector<int>& result) {
```

```
        if (root == NULL) return;
        inorderHelper(root->left, result);
        result.push_back(root->val);
        inorderHelper(root->right, result);
    }
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorderHelper(root, result);
        return result;
    }
};

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

    Solution solution;
    vector<int> result = solution.inorderTraversal(root);

    // Output the result
    for (int val : result) {
```

```
        cout << val << " ";
    }
    cout << endl;

    return 0;
}
```

OUTPUT: -

Question 2: -

Binary Tree - Sum of All Nodes

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

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

class Solution {
```

```
public
    int sumOfTree(TreeNode* root) {
        if (root == NULL) return 0;
        return root->val + sumOfTree(root->left) +
sumOfTree(root->right);
    }
};

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);

    Solution solution;
    int sum = solution.sumOfTree(root);

    cout << "Sum of all node values: " << sum << endl;

    return 0;
}
```

OUTPUT: -

```
Sum of all node values: 15
```

Question 3: -

Same Tree

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

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

class Solution {
public:
    bool 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);
    }
};

int main() {
    TreeNode* tree1 = new TreeNode(1);
    tree1->left = new TreeNode(2);
    tree1->right = new TreeNode(3);

    TreeNode* tree2 = new TreeNode(1);
    tree2->left = new TreeNode(2);
    tree2->right = new TreeNode(3);

    Solution solution;
    bool result = solution.isSameTree(tree1, tree2);

    if (result) {
        cout << "The two trees are the same." << endl;
    } else {
        cout << "The two trees are not the same." << endl;
    }
```

```
    return 0;
}
```

OUTPUT: -

```
The two trees are the same.
```

Question 4: -

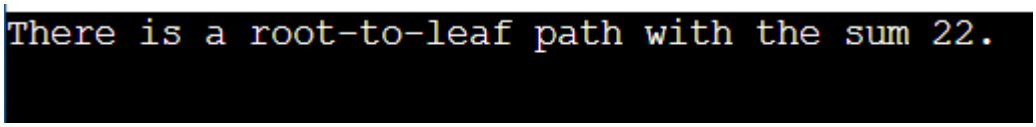
Path Sum

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

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

class Solution {
public:
    bool hasPathSum(TreeNode* root, int sum) {
        if (root == NULL) return false;
```

```
        if (root->left == NULL && root->right == NULL) {
            return root->val == sum;
        }
        int newSum = sum - root->val;
        return hasPathSum(root->left, newSum) ||
hasPathSum(root->right, newSum);
    }
};

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

    Solution solution;
    int sum = 22;
```

```
    bool result = solution.hasPathSum(root, sum);


    if (result) {
        cout << "There is a root-to-leaf path with the sum " << sum << "." << endl;
    } else {
        cout << "No root-to-leaf path with the sum " << sum << " exists." << endl;
    }


    return 0;
}
```

OUTPUT: -



Question 5: -

Construct Binary Tree from Preorder and Inorder Traversal

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

struct TreeNode {
```

```
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
    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;
        }

        int preorderIndex = 0;
        return buildTreeHelper(preorder, inorderIndexMap,
preorderIndex, 0, inorder.size() - 1);
    }

private:
    TreeNode* buildTreeHelper(vector<int>& preorder,
unordered_map<int, int>& inorderIndexMap,
                              int& preorderIndex, int inorderStart, int inorderEnd)
{
```

```
        if (inorderStart > inorderEnd) return NULL;

        int rootVal = preorder[preorderIndex++];

        TreeNode* root = new TreeNode(rootVal);

        int rootIndexInInorder = inorderIndexMap[rootVal];

        root->left = buildTreeHelper(preorder, inorderIndexMap,
preorderIndex, inorderStart, rootIndexInInorder - 1);
        root->right = buildTreeHelper(preorder, inorderIndexMap,
preorderIndex, rootIndexInInorder + 1, inorderEnd);

        return root;
    }
};

void printTree(TreeNode* root) {
    if (root == NULL) return;
    cout << root->val << " ";
    printTree(root->left);
    printTree(root->right);
}
```

```
int main() {
    Solution solution;

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

    TreeNode* root = solution.buildTree(preorder, inorder);
    cout << "Preorder of constructed tree: ";
    printTree(root);
    cout << endl;

    return 0;
}
```

OUTPUT: -

```
Preorder of constructed tree: 3 9 20 15 7
```

Question 6

Binary Tree Level Order Traversal

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

struct TreeNode {
    int val;
```

```
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> result;
        if (root == NULL) return result;

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

        while (!q.empty()) {
            int levelSize = q.size();
            vector<int> level;
            for (int i = 0; i < levelSize; ++i) {
                TreeNode* node = q.front();
                q.pop();
                level.push_back(node->val);
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
```

```
            result.push_back(level);
        }

        return result;
    }
};

void printLevelOrder(vector<vector<int>>& result) {
    for (const auto& level : result) {
        for (int val : level) {
            cout << val << " ";
        }
        cout << endl;
    }
}

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);

    Solution solution;
```

```
    vector<vector<int>> result = solution.levelOrder(root);

    cout << "Level order traversal:" << endl;
    printLevelOrder(result);

    return 0;
}
```

OUTPUT: -

```
Level order traversal:
3
9 20
15 7
```

Question 7: -

Populating Next Right Pointers in Each Node

```
#include <iostream>
#include <queue>
using namespace std;
struct Node {
    int val;
    Node *left;
    Node *right;
    Node *next;
    Node(int x) : val(x), left(NULL), right(NULL), next(NULL) {}
};
```

```
class Solution {
public:
    void connect(Node* root) {
        if (!root) return;
        Node* leftmost = root;
        while (leftmost) {
            Node* current = leftmost;
            Node* prev = NULL;
            Node* nextLeftmost = NULL;
            while (current) {
                if (current->left) {
                    if (!nextLeftmost) nextLeftmost = current->left;
                    if (prev) prev->next = current->left;
                    prev = current->left;
                }

                if (current->right) {
                    if (!nextLeftmost) nextLeftmost = current->right;
                    if (prev) prev->next = current->right;
                    prev = current->right;
                }

                current = current->next;
            }
```

```
            leftmost = nextLeftmost;
        }
    }
};
void printNextPointers(Node* root) {
    while (root) {
        Node* current = root;
        while (current) {
            cout << current->val;
            if (current->next) {
                cout << " -> ";
            } else {
                cout << " -> NULL";
            }
            current = current->next;
        }
        cout << endl;
        root = root->left;
    }
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
```

```
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->left = new Node(6);
    root->right->right = new Node(7);

    Solution solution;
    solution.connect(root);
    cout << "Next pointers after population:" << endl;
    printNextPointers(root);

    return 0;
}
```

OUTPUT: -

```
Next pointers after population:
1 -> NULL
2 -> 3 -> NULL
4 -> 5 -> 6 -> 7 -> NULL
```

Question8: -

Kth Smallest Element in a BST (Binary Search Tree)

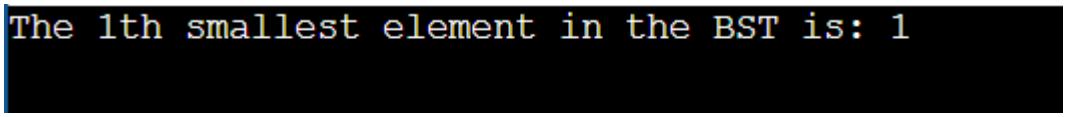
```
#include <iostream>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left;
```

```
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        int count = 0;
        int result = -1;
        inOrderTraversal(root, k, count, result);
        return result;
    }

private:
    void inOrderTraversal(TreeNode* node, int k, int& count, int&
result) {

        if (!node) return;
        inOrderTraversal(node->left, k, count, result);
        count++;
        if (count == k) {
            result = node->val;
            return;
        }
```

```
        inOrderTraversal(node->right, k, count, result);
    }
};
TreeNode* insert(TreeNode* root, int val) {
    if (root == NULL) {
        return new TreeNode(val);
    }
    if (val < root->val) {
        root->left = insert(root->left, val);
    } else {
        root->right = insert(root->right, val);
    }
    return root;
}

int main() {
    Solution solution;
    TreeNode* root = NULL;
    root = insert(root, 3);
    root = insert(root, 1);
    root = insert(root, 4);
    root = insert(root, 2);

    int k = 1;
```

```
    int result = solution.kthSmallest(root, k);

    cout << "The " << k << "th smallest element in the BST is: " <<
result << endl;

    return 0;
}
```

OUTPUT: -



Question 9: -

Count Paths That Can Form a Palindrome in a Tree

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

class Solution {
public:
    int countPalindromePairs(vector<int>& parent, string& s) {
        int n = parent.size();
        vector<int> adj[n];
        for (int i = 1; i < n; ++i) {
```

```
            adj[parent[i]].push_back(i);
        }

        unordered_map<int, int> freqMap;
        freqMap[0] = 1;
        int count = 0;
        dfs(0, adj, s, freqMap, 0, count);

        return count;
    }

private:
    void dfs(int node, vector<int> adj[], string& s, unordered_map<int,
int>& freqMap, int mask, int& count) {

        mask ^= (1 << (s[node] - 'a'));
        count += freqMap[mask];
        for (int i = 0; i < 26; ++i) {
            count += freqMap[mask ^ (1 << i)];
        }
        freqMap[mask]++;
        for (int child : adj[node]) {
            dfs(child, adj, s, freqMap, mask, count);
        }
```

```
        freqMap[mask]--;
    }
};

int main() {
    Solution solution;
    vector<int> parent = {-1, 0, 0, 1, 1, 2};
    string s = "abca";
    int result = solution.countPalindromePairs(parent, s);
    cout << "The number of pairs is: " << result << endl;

    return 0;
}
```

OUTPUT: -

```
The number of pairs is: 5
```

QUESTION 10: -

Longest Path With Different Adjacent Characters

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

class Solution {
```

```
public:
    int longestPath(vector<int>& parent, string& s) {
        int n = parent.size();
        vector<vector<int>> tree(n);
        for (int i = 1; i < n; ++i) {
            tree[parent[i]].push_back(i);
        }
        int result = 0;
        dfs(0, tree, s, result);

        return result;
    }

private
    int dfs(int node, vector<vector<int>>& tree, string& s, int& result) {
        int max1 = 0, max2 = 0;
        for (int child : tree[node]) {

            int childPath = dfs(child, tree, s, result);

            if (s[node] != s[child]) {
                if (childPath > max1) {
                    max2 = max1;
                    max1 = childPath;
```

```
            } else if (childPath > max2) {
                max2 = childPath;
            }
        }

        result = max(result, max1 + max2 + 1);
        return max1 + 1;
    }
};

int main() {
    Solution solution;
    vector<int> parent = {-1, 0, 0, 1, 1, 2};
    string s = "abacda";
    int result = solution.longestPath(parent, s);

    // Output the result
    cout << "The length of the longest path is: " << result << endl;

    return 0;
}
```

OUTPUT: -