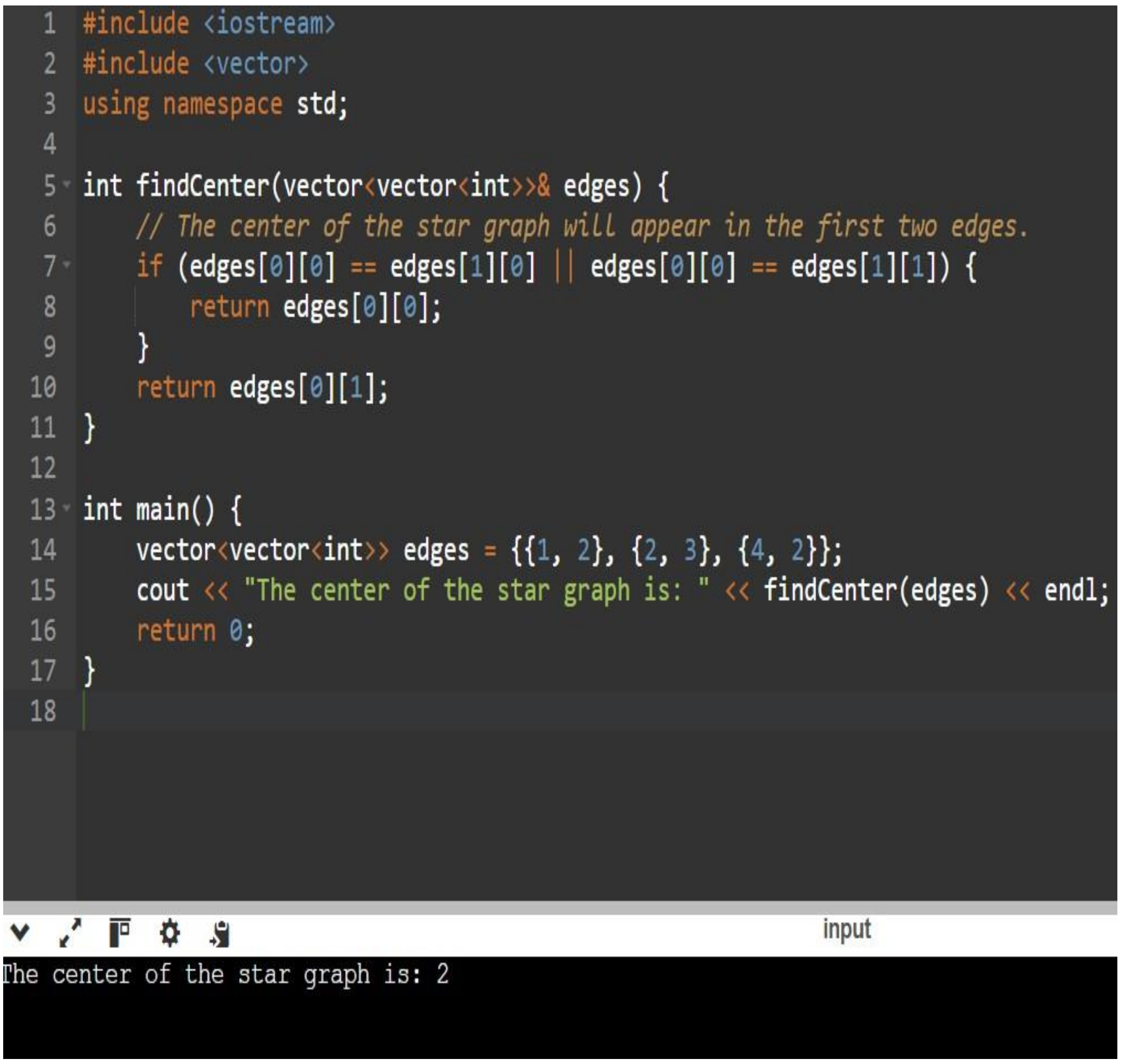
Domain Winter Camp DAY-7

Name: - LAKSHAY KAMBOJ

Section/Group: -901-Kpit-B

UID: - 22BCS11185

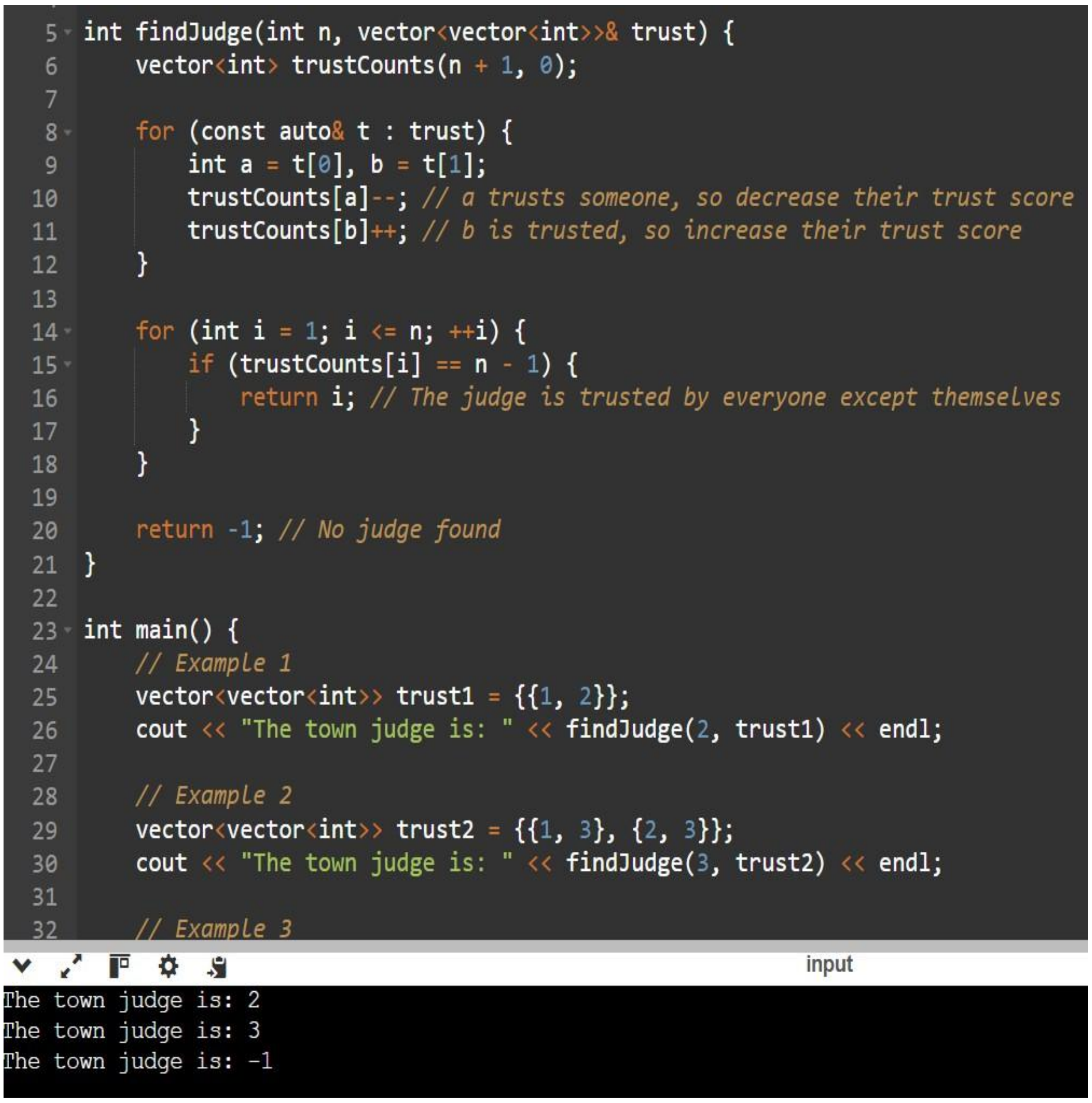
Problem 1



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

int findCenter(vector<vector<int>>& edges) {
    // The center of the star graph will appear in the first two edges.
    if (edges[0][0] == edges[1][0] || edges[0][0] == edges[1][1]) {
        return edges[0][0];
    }
    return edges[0][1];
}

int main() {
    vector<vector<int>> edges = {{1, 2}, {2, 3}, {4, 2}};
    cout << "The center of the star graph is: " << findCenter(edges) << endl;
    return 0;
}
```

```
The center of the star graph is: 2
```

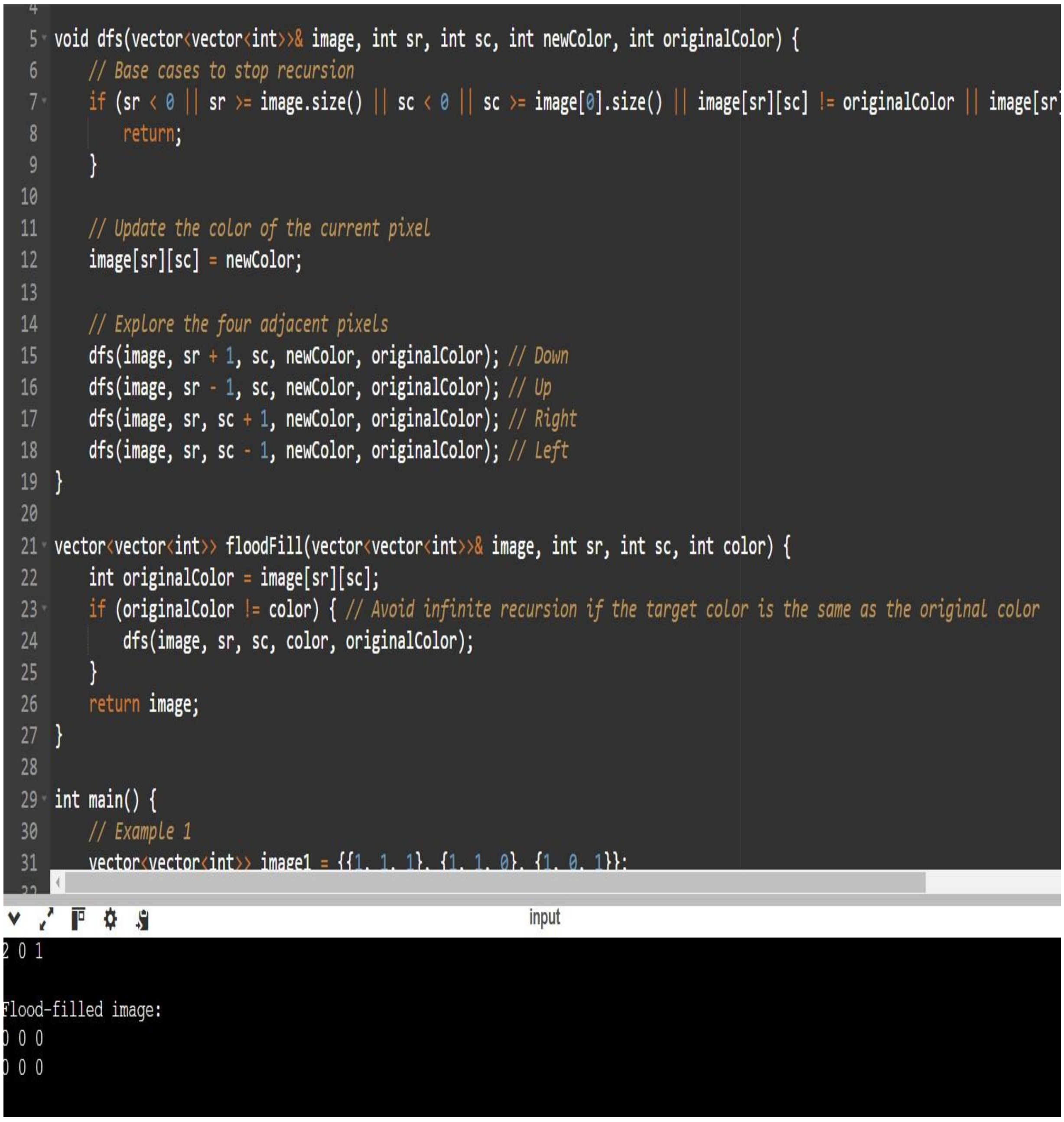
Problem 2



```
int findJudge(int n, vector<vector<int>>& trust) {
    vector<int> trustCounts(n + 1, 0);

    for (const auto& t : trust) {
        int a = t[0], b = t[1];
        trustCounts[a]--; // a trusts someone, so decrease their trust score
        trustCounts[b]++; // b is trusted, so increase their trust score
    }

    for (int i = 1; i <= n; ++i) {
        if (trustCounts[i] == n - 1) {
            return i; // The judge is trusted by everyone except themselves
        }
    }

    return -1; // No judge found
}

int main() {
    // Example 1
    vector<vector<int>> trust1 = {{1, 2}};
    cout << "The town judge is: " << findJudge(2, trust1) << endl;

    // Example 2
    vector<vector<int>> trust2 = {{1, 3}, {2, 3}};
    cout << "The town judge is: " << findJudge(3, trust2) << endl;

    // Example 3
```

```
The town judge is: 2
The town judge is: 3
The town judge is: -1
```

Problem 3

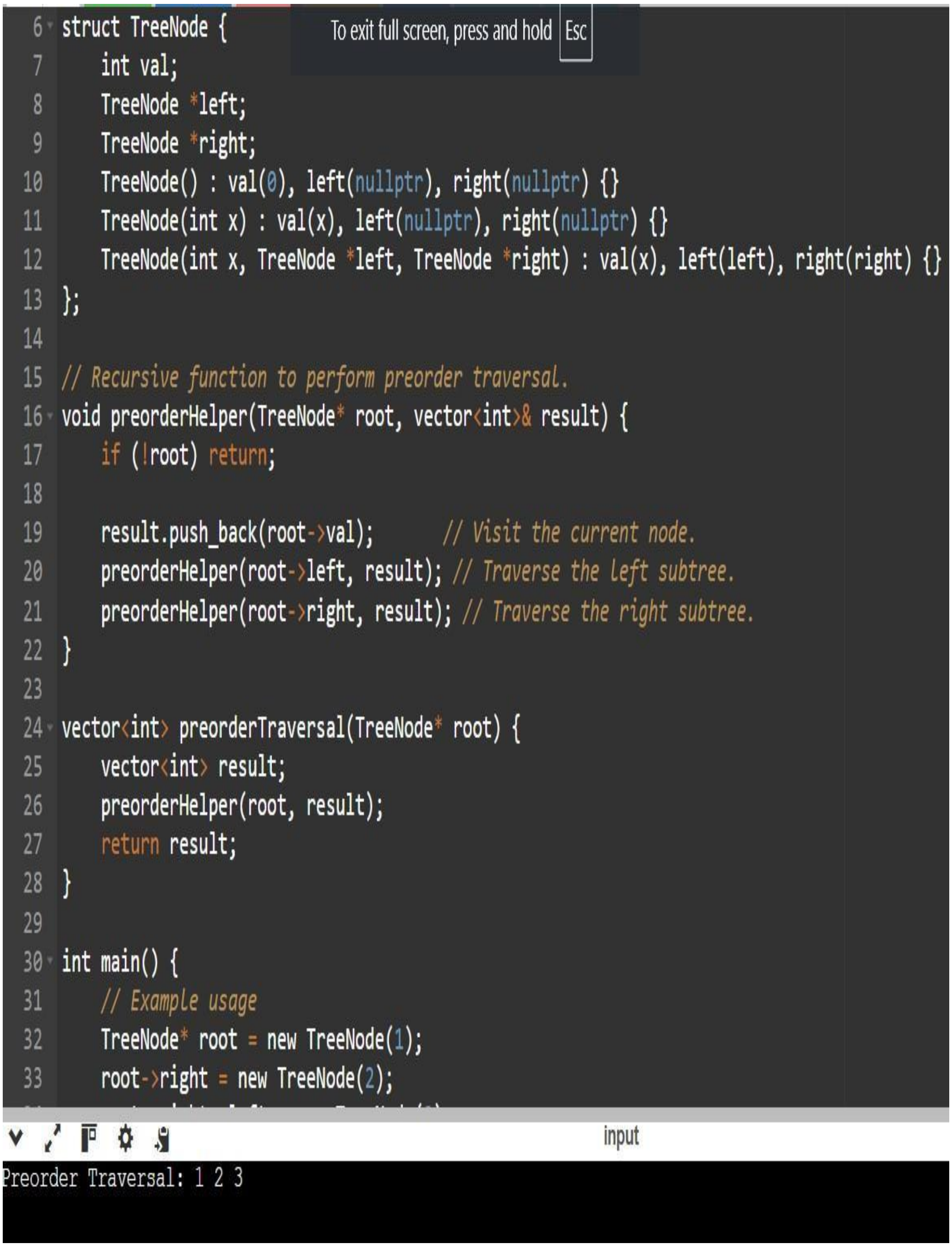


```cpp
void dfs(vector<vector<int>>& image, int sr, int sc, int newColor, int originalColor) {
    // Base cases to stop recursion
    if (sr < 0 || sr >= image.size() || sc < 0 || sc >= image[0].size() || image[sr][sc] != originalColor || image[sr
        return;
    }

    // Update the color of the current pixel
    image[sr][sc] = newColor;

    // Explore the four adjacent pixels
    dfs(image, sr + 1, sc, newColor, originalColor); // Down
    dfs(image, sr - 1, sc, newColor, originalColor); // Up
    dfs(image, sr, sc + 1, newColor, originalColor); // Right
    dfs(image, sr, sc - 1, newColor, originalColor); // Left
}

vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int color) {
    int originalColor = image[sr][sc];
    if (originalColor != color) { // Avoid infinite recursion if the target color is the same as the original color
        dfs(image, sr, sc, color, originalColor);
    }
    return image;
}

int main() {
    // Example 1
    vector<vector<int>> image1 = {{1, 1, 1}, {1, 1, 0}, {1, 0, 1}};
```

input

```
2 0 1

Flood-filled image:
0 0 0
0 0 0
```

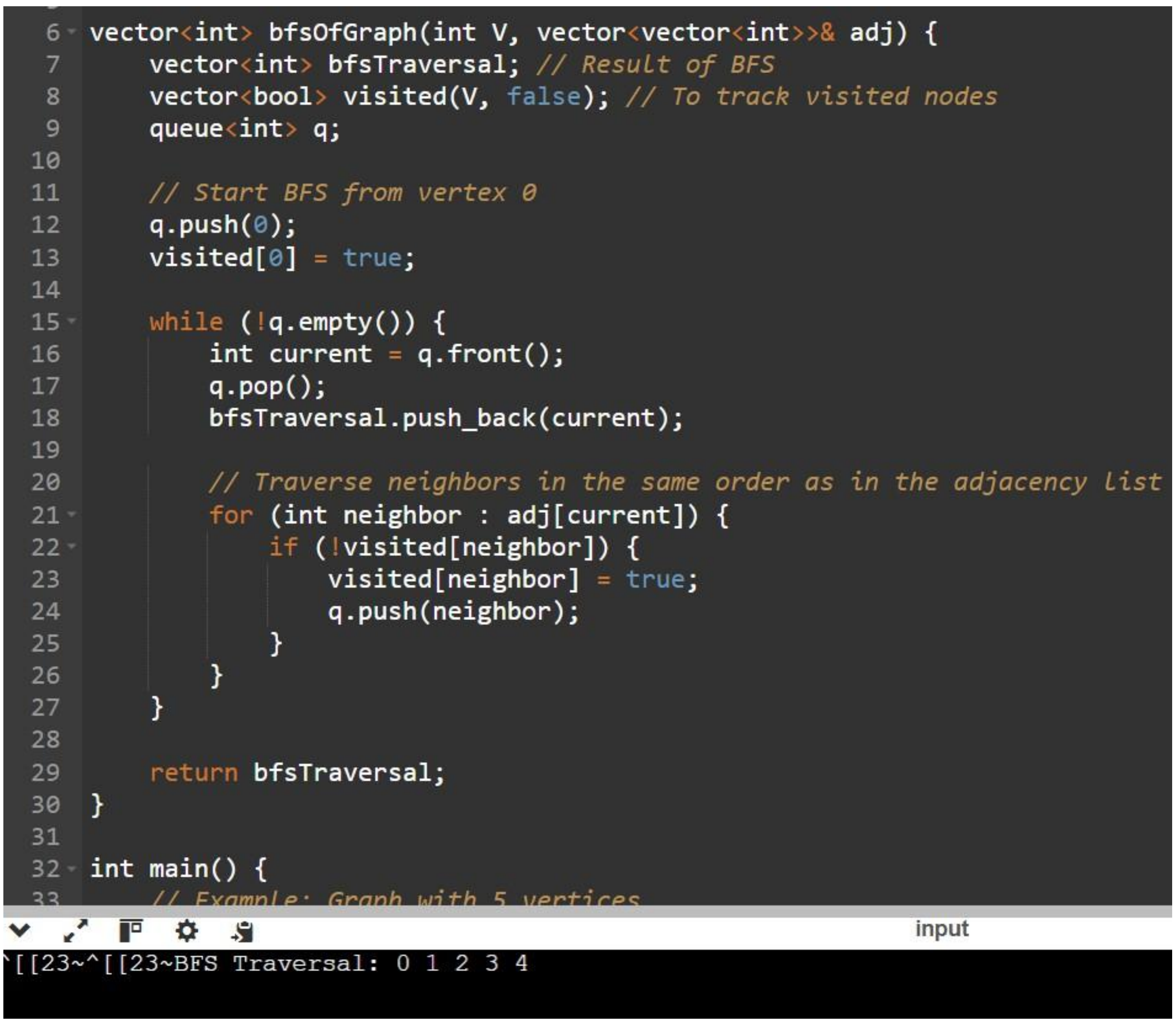
Problem 4



```cpp
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) {}
};

// Recursive function to perform preorder traversal.
void preorderHelper(TreeNode* root, vector<int>& result) {
    if (!root) return;

    result.push_back(root->val);        // Visit the current node.
    preorderHelper(root->left, result); // Traverse the left subtree.
    preorderHelper(root->right, result); // Traverse the right subtree.
}

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

int main() {
    // Example usage
    TreeNode* root = new TreeNode(1);
    root->right = new TreeNode(2);
```

```
Preorder Traversal: 1 2 3
```

Problem 5



```cpp
vector<int> bfsOfGraph(int V, vector<vector<int>>& adj) {
    vector<int> bfsTraversal; // Result of BFS
    vector<bool> visited(V, false); // To track visited nodes
    queue<int> q;

    // Start BFS from vertex 0
    q.push(0);
    visited[0] = true;

    while (!q.empty()) {
        int current = q.front();
        q.pop();
        bfsTraversal.push_back(current);

        // Traverse neighbors in the same order as in the adjacency list
        for (int neighbor : adj[current]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }

    return bfsTraversal;
}

int main() {
    // Example: Graph with 5 vertices
```

```
`[[23~^[[23~BFS Traversal: 0 1 2 3 4
```