

Domain Winter Camp DAY-7

Name :- Ishan Sharma

Section/Group:-901-Kpit-B

UID :- 22BCS11144

Problem 1

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int findCenter(vector<vector<int>>& edges) {
6      // The center of the star graph will appear in the first two edges.
7      if (edges[0][0] == edges[1][0] || edges[0][0] == edges[1][1]) {
8          return edges[0][0];
9      }
10     return edges[0][1];
11 }
12
13 int main() {
14     vector<vector<int>> edges = {{1, 2}, {2, 3}, {4, 2}};
15     cout << "The center of the star graph is: " << findCenter(edges) << endl;
16     return 0;
17 }
18
```



input

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

Problem 2

```

5 int findJudge(int n, vector<vector<int>>& trust) {
6     vector<int> trustCounts(n + 1, 0);
7
8     for (const auto& t : trust) {
9         int a = t[0], b = t[1];
10        trustCounts[a]--; // a trusts someone, so decrease their trust score
11        trustCounts[b]++; // b is trusted, so increase their trust score
12    }
13
14    for (int i = 1; i <= n; ++i) {
15        if (trustCounts[i] == n - 1) {
16            return i; // The judge is trusted by everyone except themselves
17        }
18    }
19
20    return -1; // No judge found
21 }
22
23 int main() {
24     // Example 1
25     vector<vector<int>> trust1 = {{1, 2}};
26     cout << "The town judge is: " << findJudge(2, trust1) << endl;
27
28     // Example 2
29     vector<vector<int>> trust2 = {{1, 3}, {2, 3}};
30     cout << "The town judge is: " << findJudge(3, trust2) << endl;
31
32     // Example 3

```



input

```

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

```

Problem 3

```

4
5 void dfs(vector<vector<int>>& image, int sr, int sc, int newColor, int originalColor) {
6     // Base cases to stop recursion
7     if (sr < 0 || sr >= image.size() || sc < 0 || sc >= image[0].size() || image[sr][sc] != originalColor || image[sr][sc] == newColor)
8         return;
9 }
10
11 // Update the color of the current pixel
12 image[sr][sc] = newColor;
13
14 // Explore the four adjacent pixels
15 dfs(image, sr + 1, sc, newColor, originalColor); // Down
16 dfs(image, sr - 1, sc, newColor, originalColor); // Up
17 dfs(image, sr, sc + 1, newColor, originalColor); // Right
18 dfs(image, sr, sc - 1, newColor, originalColor); // Left
19 }
20
21 vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int color) {
22     int originalColor = image[sr][sc];
23     if (originalColor != color) { // Avoid infinite recursion if the target color is the same as the original color
24         dfs(image, sr, sc, color, originalColor);
25     }
26     return image;
27 }
28
29 int main() {
30     // Example 1
31     vector<vector<int>> image1 = {{1, 1, 1}, {1, 1, 0}, {1, 0, 1}};
32 }

```



input

2 0 1

Flood-filled image:

0 0 0

0 0 0

Problem 4

```
6 struct TreeNode {
7     int val;
8     TreeNode *left;
9     TreeNode *right;
10    TreeNode() : val(0), left(nullptr), right(nullptr) {}
11    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
12    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
13 };
14
15 // Recursive function to perform preorder traversal.
16 void preorderHelper(TreeNode* root, vector<int>& result) {
17     if (!root) return;
18
19     result.push_back(root->val);    // Visit the current node.
20     preorderHelper(root->left, result); // Traverse the left subtree.
21     preorderHelper(root->right, result); // Traverse the right subtree.
22 }
23
24 vector<int> preorderTraversal(TreeNode* root) {
25     vector<int> result;
26     preorderHelper(root, result);
27     return result;
28 }
29
30 int main() {
31     // Example usage
32     TreeNode* root = new TreeNode(1);
33     root->right = new TreeNode(2);
```

▼ 🔍 ⚙️ 📁 input

Preorder Traversal: 1 2 3

Problem 5

```
6 vector<int> bfsOfGraph(int V, vector<vector<int>>& adj) {
7     vector<int> bfsTraversal; // Result of BFS
8     vector<bool> visited(V, false); // To track visited nodes
9     queue<int> q;
10
11     // Start BFS from vertex 0
12     q.push(0);
13     visited[0] = true;
14
15     while (!q.empty()) {
16         int current = q.front();
17         q.pop();
18         bfsTraversal.push_back(current);
19
20         // Traverse neighbors in the same order as in the adjacency list
21         for (int neighbor : adj[current]) {
22             if (!visited[neighbor]) {
23                 visited[neighbor] = true;
24                 q.push(neighbor);
25             }
26         }
27     }
28
29     return bfsTraversal;
30 }
31
32 int main() {
33     // Example: Graph with 5 vertices
```

input

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