

DOMAIN WINTER WINNING CAMP

Student Name: Unnati Srivastava

UID: 22BCS13475

Branch: BE-CSE

Section/Group: TPP_FL_603-A

DAY 7:

QUES 1: Find Center of Star Graph

There is an undirected star graph consisting of n nodes labeled from 1 to n . A star graph is a graph where there is one center node and exactly $n - 1$ edges that connect the center node with every other node.

You are given a 2D integer array `edges` where each `edges[i] = [ui, vi]` indicates that there is an edge between the nodes `ui` and `vi`. Return the center of the given star graph.

Solution:

```
#include <iostream>

#include <vector>

using namespace std;

class Solution {
public:
    int findCenter(vector<vector<int>>& edges) {
        if (edges[0][0] == edges[1][0] || edges[0][0] == edges[1][1])
            return edges[0][0];
        return edges[0][1];
    }
};

int main() {
    Solution solution;

    vector<vector<int>> edges = {{1, 2}, {2, 3}, {4, 2}};

    cout << "Center of the star graph: " << solution.findCenter(edges) << endl;

    return 0;
}
```

}

Center of the star graph: 2

QUES 2: Find the Town Judge

In a town, there are n people labeled from 1 to n . There is a rumor that one of these people is secretly the town judge.

If the town judge exists, then:

1. The town judge trusts nobody.
2. Everybody (except for the town judge) trusts the town judge.
3. There is exactly one person that satisfies properties 1 and 2.

You are given an array `trust` where `trust[i] = [ai, bi]` representing that the person labeled `ai` trusts the person labeled `bi`. If a trust relationship does not exist in trust array, then such a trust relationship does not exist.

Solution:

```
#include <iostream>

#include <vector>

using namespace std;

class Solution {
public:
    int findJudge(int n, vector<vector<int>>& trust) {
        vector<int> trustScore(n + 1, 0);
        for (auto& t : trust) {
            trustScore[t[0]]--;
            trustScore[t[1]]++;
        }
        for (int i = 1; i <= n; i++) {
            if (trustScore[i] == n - 1) {
                return i; // Judge is trusted by everyone except themselves
            }
        }
    }
};
```

```
    }  
    }  
    return -1;  
}  
};  
  
int main() {  
    Solution solution;  
    int n = 2;  
    vector<vector<int>> trust = {{1, 2}};  
    cout << "Town Judge: " << solution.findJudge(n, trust) << endl;  
    return 0;  
}
```

```
Town Judge: 2
```

QUES 3: BFS of graph link

Given a connected undirected graph represented by an adjacency list adj, which is a vector of vectors where each adj[i] represents the list of vertices connected to vertex i. Perform a Breadth First Traversal (BFS) starting from vertex 0, visiting vertices from left to right according to the adjacency list, and return a list containing the BFS traversal of the graph.

Note: Do traverse in the same order as they are in the adjacency list.

Solution:

```
#include <iostream>  
  
#include <vector>  
  
#include <queue>  
  
using namespace std;  
  
class Solution {  
public:  
    vector<int> bfsOfGraph(int V, vector<vector<int>>& adj) {
```

```
vector<int> result;

vector<bool> visited(V, false);

queue<int> q;

q.push(0);

visited[0] = true;

while (!q.empty()) {
    int node = q.front();

    q.pop();

    result.push_back(node);

    for (int neighbor : adj[node]) {
        if (!visited[neighbor]) {
            q.push(neighbor);
            visited[neighbor] = true;
        }
    }
}

return result;
}

};

int main() {
    Solution solution;

    vector<vector<int>>> adj = {{2, 3, 1}, {0}, {0, 4}, {0}, {2}};

    int V = adj.size();

    vector<int> bfsTraversal = solution.bfsOfGraph(V, adj);

    for (int node : bfsTraversal) {
        cout << node << " ";
    }

    cout << endl;
```

```
    return 0;  
}
```

```
0 2 3 1 4
```

QUES 4: DFS of Graph

Given a connected undirected graph represented by an adjacency list `adj`, which is a vector of vectors where each `adj[i]` represents the list of vertices connected to vertex `i`. Perform a Depth First Traversal (DFS) starting from vertex 0, visiting vertices from left to right as per the adjacency list, and return a list containing the DFS traversal of the graph.

Note: Do traverse in the same order as they are in the adjacency list.

Solution:

```
#include <iostream>  
  
#include <vector>  
  
using namespace std;  
  
class Solution {  
public:  
  
    void dfs(int node, vector<vector<int>>& adj, vector<bool>& visited, vector<int>& result)  
    {  
        visited[node] = true;  
        result.push_back(node);  
        for (int neighbor : adj[node]) {  
            if (!visited[neighbor]) {  
                dfs(neighbor, adj, visited, result);  
            }  
        }  
    }  
  
    vector<int> dfsOfGraph(int V, vector<vector<int>>& adj) {  
        vector<int> result;
```

```
        vector<bool> visited(V, false);  
        dfs(0, adj, visited, result);  
        return result;  
    }  
};  
  
int main() {  
    Solution solution;  
    vector<vector<int>>> adj = {{2, 3, 1}, {0}, {0, 4}, {0}, {2}};  
    int V = adj.size();  
    vector<int> dfsTraversal = solution.dfsOfGraph(V, adj);  
    for (int node : dfsTraversal) {  
        cout << node << " ";  
    }  
    cout << endl;  
    return 0;  
}
```

```
0 2 4 3 1
```

QUES 5: Matrix

Given an m x n binary matrix mat, return the distance of the nearest 0 for each cell.

The distance between two adjacent cells is 1.

Solution:

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

public:

```
vector<vector<int>> updateMatrix(vector<vector<int>>& mat) {  
    int m = mat.size(), n = mat[0].size();  
    vector<vector<int>> dist(m, vector<int>(n, -1));  
    queue<pair<int, int>> q;  
    for (int i = 0; i < m; ++i)  
        for (int j = 0; j < n; ++j)  
            if (mat[i][j] == 0) {  
                dist[i][j] = 0;  
                q.push({i, j});  
            }  
    vector<int> directions = {-1, 0, 1, 0, -1};  
    while (!q.empty()) {  
        auto [x, y] = q.front();  
        q.pop();  
        for (int i = 0; i < 4; ++i) {  
            int newX = x + directions[i], newY = y + directions[i + 1];  
            if (newX >= 0 && newX < m && newY >= 0 && newY < n &&  
dist[newX][newY] == -1) {  
                dist[newX][newY] = dist[x][y] + 1;  
                q.push({newX, newY});  
            }  
        }  
    }  
    return dist;  
}  
};  
  
int main() {  
    Solution solution;  
    vector<vector<int>> mat = {{0, 0, 0}, {0, 1, 0}, {0, 0, 0}};
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
vector<vector<int>> result = solution.updateMatrix(mat);  
for (auto& row : result) {  
    for (int cell : row) cout << cell << " ";  
    cout << endl;  
}  
return 0;  
}
```

```
0 0 0  
0 1 0  
0 0 0
```