Domain Winter Camp DAY-6

Name :- Aditya Sharma

Section/Group:-901-Kpit-B

UID :- 22BCS10722

Problem 1 Minimum Height Trees

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

vector<int> findMinHeightTrees(int n, vector<vector<int>> &edges)
{
    // Edge case: Single node tree
    if (n == 1)
        return {0};

    // Step 1: Build the graph (adjacency list) and degree array
    vector<vector<int>> graph(n);
    vector<int> degree(n, 0);
    for (const auto &edge : edges)
    {
        graph[edge[0]].push_back(edge[1]);
        graph[edge[1]].push_back(edge[0]);
        degree[edge[0]]++;
        degree[edge[1]]++;
    }

    // Step 2: Initialize the queue with leaf nodes (degree == 1)
    queue<int> leaves;
    for (int i = 0; i < n; i++)
    {
        if (degree[i] == 1)
        {
            leaves.push(i);
        }
    }

    // Step 3: Trim the leaves iteratively
    while (n > 2)
    {
        int leavesCount = leaves.size();
        n -= leavesCount; // Remove current level of leaves

        for (int i = 0; i < leavesCount; i++)
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS C:\Users\adity\OneDrive\Desktop\WINTER DOMAIN CAMP\DAY 7> cd '
Roots of Minimum Height Trees: 3 4
PS C:\Users\adity\OneDrive\Desktop\WINTER DOMAIN CAMP\DAY 7>
```

Problem 2 Accounts Merge

```cpp
C++ Q2.cpp > ...
1    #include <iostream>
2    #include <vector>
3    #include <queue>
4    using namespace std;
5
6    vector<int> findMinHeightTrees(int n, vector<vector<int>> &edges)
7    {
8        // Edge case: Single node tree
9        if (n == 1)
10           return {0};
11
12       // Step 1: Build the graph (adjacency list) and degree array
13       vector<vector<int>> graph(n);
14       vector<int> degree(n, 0);
15       for (const auto &edge : edges)
16       {
17           graph[edge[0]].push_back(edge[1]);
18           graph[edge[1]].push_back(edge[0]);
19           degree[edge[0]]++;
20           degree[edge[1]]++;
21       }
22
23       // Step 2: Initialize the queue with leaf nodes (degree == 1)
24       queue<int> leaves;
25       for (int i = 0; i < n; i++)
26       {
27           if (degree[i] == 1)
28           {
29               leaves.push(i);
30           }
31       }
32
33       // Step 3: Trim the leaves iteratively
34       while (n > 2)
35       {
36           int leavesCount = leaves.size();
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

Roots of Minimum Height Trees: 3 4
PS C:\Users\adity\OneDrive\Desktop\WINTER DOMAIN CAMP\DAY 7> c
Roots of Minimum Height Trees: 3 4

Problem 3 Rotting Oranges

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

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

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

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

    void unionSet(int x, int y)
```

Problem 4 Pacific Atlantic water Flow

```cpp
C++ Q4.cpp > ...
1    #include <iostream>
2    #include <vector>
3    using namespace std;
4
5    void dfs(vector<vector<int>> &heights, vector<vector<bool>> &visited, int x, int y, int prevHeight)
6    {
7        int m = heights.size(), n = heights[0].size();
8
9        // Base conditions
10       if (x < 0 || x >= m || y < 0 || y >= n || visited[x][y] || heights[x][y] < prevHeight)
11       {
12           return;
13       }
14
15       visited[x][y] = true;
16
17       // Explore in all 4 directions
18       vector<pair<int, int>> directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
19       for (auto &dir : directions)
20       {
21           dfs(heights, visited, x + dir.first, y + dir.second, heights[x][y]);
22       }
23   }
24
25   vector<vector<int>> pacificAtlantic(vector<vector<int>> &heights)
26   {
27       int m = heights.size(), n = heights[0].size();
28
29       // Create visited matrices for both oceans
30       vector<vector<bool>> pacific(m, vector<bool>(n, false));
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

cd "c:\Users\adity\OneDrive\De

```
Cells that can flow to both oceans:
[0, 4]
[1, 3]
[1, 4]
[2, 2]
[3, 0]
[3, 1]
[4, 0]
```

Problem 5 Network Delay Time

```cpp
C++ Q5.cpp > ...
1    #include <iostream>
2    #include <vector>
3    #include <queue>
4    #include <climits>
5    using namespace std;
6
7    int networkDelayTime(vector<vector<int>> &times, int n, int k)
8    {
9        // Step 1: Build the adjacency list
10       vector<vector<pair<int, int>>> graph(n + 1);
11       for (const auto &time : times)
12       {
13           int u = time[0], v = time[1], w = time[2];
14           graph[u].emplace_back(v, w);
15       }
16
17       // Step 2: Min-heap to perform Dijkstra's algorithm
18       priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
19       vector<int> dist(n + 1, INT_MAX);
20       dist[k] = 0;
21       pq.emplace(0, k); // {distance, node}
22
23       while (!pq.empty())
24       {
25           int curDist = pq.top().first;
26           int node = pq.top().second;
27           pq.pop();
28
29           // If the current distance is greater than the stored distance, skip
30           if (curDist > dist[node])
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\adity\OneDrive\Desktop\WINTER DOMAIN CAMP\DAY 7> cd "c:\Users\adity\
Minimum time for all nodes to receive the signal: 2