WINTER DOMAIN CAMP

NAME: Vikram Kumar **UID:22BCS12220**

SECTION: KPIT-901-B **DATE: 28/12/2024**

Q1. Find centre of star graph(Easy) .

Sol.

```cpp
//find center of star graph(easy).
#include <iostream>
#include <vector>
using namespace std;

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() {
    vector<vector<int>> edges1 = {{1, 2}, {2, 3}, {4, 2}};
    vector<vector<int>> edges2 = {{1, 2}, {5, 1}, {1, 3}, {1, 4}};

    cout << "Center of first graph: " << findCenter(edges1) << endl;
    cout << "Center of second graph: " << findCenter(edges2) << endl;

    return 0;
}
```

OUTPUT:

```
Center of first graph: 2
Center of second graph: 1
```

Q2. BFS of graph link(medium) Sol.

```cpp
//BFS of graph link(medium).
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

vector<int> bfsTraversal(int V, vector<vector<int>>& adj) {
    vector<int> bfs;
    vector<bool> visited(V, false);
    queue<int> q;
    q.push(0);
    visited[0] = true;
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        bfs.push_back(node);
        for (int neighbor : adj[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
    return bfs;
}

int main() {
    // Example 1
    vector<vector<int>> adj1 = {{2, 3, 1}, {0}, {0, 4}, {0}, {2}};
    vector<int> result1 = bfsTraversal(5, adj1);
    cout << "BFS Traversal (Example 1): ";
    for (int node : result1) {
        cout << node << " ";
    }
    cout << endl;

    // Example 2
    vector<vector<int>> adj2 = {{1, 2}, {0, 2}, {0, 1, 3, 4}, {2}, {2}};
```

```
39      vector<int> result2 = bfsTraversal(5, adj2);
40      cout << "BFS Traversal (Example 2): ";
41      for (int node : result2) {
42          cout << node << " ";
43      }
44      cout << endl;
45
46      // Example 3
47      vector<vector<int>> adj3 = {{1}, {0, 2, 3}, {1}, {1, 4}, {3}};
48      vector<int> result3 = bfsTraversal(5, adj3);
49      cout << "BFS Traversal (Example 3): ";
50      for (int node : result3) {
51          cout << node << " ";
52      }
53      cout << endl;
54
55      return 0;
56  }
```

OUTPUT:

```
BFS Traversal (Example 1): 0 2 3 1 4
BFS Traversal (Example 2): 0 1 2 3 4
BFS Traversal (Example 3): 0 1 2 3 4


...Program finished with exit code 0
Press ENTER to exit console.
```

Q3. Minimum height tree (medium).

Sol.

```cpp
//Minimum height tree(medium).
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_set>
using namespace std;

vector<int> findMinHeightTrees(int n, vector<vector<int>>& edges) {
    if (n == 1) return {0};
    vector<unordered_set<int>> adj(n);
    for (const auto& edge : edges) {
        adj[edge[0]].insert(edge[1]);
        adj[edge[1]].insert(edge[0]);
    }
    queue<int> leaves;
    for (int i = 0; i < n; ++i) {
        if (adj[i].size() == 1) {
            leaves.push(i);
        }
    }
    while (n > 2) {
        int leafCount = leaves.size();
        n -= leafCount;

        for (int i = 0; i < leafCount; ++i) {
            int leaf = leaves.front();
            leaves.pop();
            for (int neighbor : adj[leaf]) {
                adj[neighbor].erase(leaf);
                if (adj[neighbor].size() == 1) {
                    leaves.push(neighbor);
                }
            }
        }
    }
    vector<int> result;
    while (!leaves.empty()) {
        result.push_back(leaves.front());
```

```
            leaves.pop();
        }

        return result;
    }

int main() {
    // Example 1
    int n1 = 4;
    vector<vector<int>> edges1 = {{1, 0}, {1, 2}, {1, 3}};
    vector<int> result1 = findMinHeightTrees(n1, edges1);
    cout << "MHT Roots (Example 1): ";
    for (int node : result1) {
        cout << node << " ";
    }
    cout << endl;

    // Example 2
    int n2 = 6;
    vector<vector<int>> edges2 = {{3, 0}, {3, 1}, {3, 2}, {3, 4}, {5, 4}};
    vector<int> result2 = findMinHeightTrees(n2, edges2);
    cout << "MHT Roots (Example 2): ";
    for (int node : result2) {
        cout << node << " ";
    }
    cout << endl;

    return 0;
}
```

OUTPUT:

```
MHT Roots (Example 1): 1
MHT Roots (Example 2): 3 4


...Program finished with exit cod
Press ENTER to exit console.
```

Q4.Account merge (hard) Sol.

```cpp
//Account merge(Hard).
#include <iostream>
#include <vector>
#include <unordered_map>
#include <unordered_set>
#include <algorithm>
#include <functional>
using namespace std;

class Solution {
public:
    vector<vector<string>> accountsMerge(vector<vector<string>>& accounts) {
        int n = accounts.size();
        vector<int> parent(n);
                for (int i = 0; i < n; ++i) {
            parent[i] = i;
        }
        function<int(int)> find = [&](int x) {
            if (parent[x] != x) {
                parent[x] = find(parent[x]);
            }
            return parent[x];
        };
        auto unionSets = [&](int x, int y) {
            int rootX = find(x);
            int rootY = find(y);
            if (rootX != rootY) {
                parent[rootX] = rootY;
            }
        };

        unordered_map<string, int> emailToIndex;
        for (int i = 0; i < n; ++i) {
            for (int j = 1; j < accounts[i].size(); ++j) {
                const string& email = accounts[i][j];
                if (emailToIndex.count(email)) {
                    unionSets(i, emailToIndex[email]);
                } else {
```

```
                    emailToIndex[email] = i;
                }
            }
        }
        unordered_map<int, unordered_set<string>> groups;
        for (const auto& [email, index] : emailToIndex) {
            int root = find(index);
            groups[root].insert(email);
        }
        vector<vector<string>> result;
        for (const auto& [index, emails] : groups) {
            vector<string> mergedAccount;
            mergedAccount.push_back(accounts[index][0]); // Add name
            mergedAccount.insert(mergedAccount.end(), emails.begin(), emails.end());
            sort(mergedAccount.begin() + 1, mergedAccount.end()); // Sort emails
            result.push_back(mergedAccount);
        }

        return result;
    }
};

int main() {
    Solution solution;

    // Example 1
    vector<vector<string>> accounts1 = {
        {"John", "johnsmith@mail.com", "john_newyork@mail.com"},
        {"John", "johnsmith@mail.com", "john00@mail.com"},
        {"Mary", "mary@mail.com"},
        {"John", "johnnybravo@mail.com"}
    };
    vector<vector<string>> result1 = solution.accountsMerge(accounts1);
    cout << "Example 1 Output:\n";
    for (const auto& account : result1) {
        for (const auto& entry : account) {
            cout << entry << " ";
        }
```

```
77          cout << endl;
78      }
79
80      // Example 2
81      vector<vector<string>> accounts2 = {
82          {"Gabe", "Gabe0@m.co", "Gabe3@m.co", "Gabe1@m.co"},
83          {"Kevin", "Kevin3@m.co", "Kevin5@m.co", "Kevin0@m.co"},
84          {"Ethan", "Ethan5@m.co", "Ethan4@m.co", "Ethan0@m.co"},
85          {"Hanzo", "Hanzo3@m.co", "Hanzo1@m.co", "Hanzo0@m.co"},
86          {"Fern", "Fern5@m.co", "Fern1@m.co", "Fern0@m.co"}
87      };
88      vector<vector<string>> result2 = solution.accountsMerge(accounts2);
89      cout << "\nExample 2 Output:\n";
90      for (const auto& account : result2) {
91          for (const auto& entry : account) {
92              cout << entry << " ";
93          }
94          cout << endl;
95      }
96
97      return 0;
98  }
```

Output:

```
Example 1 Output:
John john00@mail.com john_newyork@mail.com johnsmith@mail.com
Mary mary@mail.com
John johnnybravo@mail.com

Example 2 Output:
Ethan Ethan0@m.co Ethan4@m.co Ethan5@m.co
Kevin Kevin0@m.co Kevin3@m.co Kevin5@m.co
Hanzo Hanzo0@m.co Hanzo1@m.co Hanzo3@m.co
Gabe Gabe0@m.co Gabe1@m.co Gabe3@m.co
Fern Fern0@m.co Fern1@m.co Fern5@m.co


Program finished with exit code 0
```

Q5.Network delay time.(hard)

Sol.

```cpp
//Network delay time(Hard).
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>
#include <climits>
using namespace std;

class Solution {
public:
    int networkDelayTime(vector<vector<int>>& times, int n, int k) {
        vector<vector<pair<int, int>>> graph(n + 1);
        for (const auto& time : times) {
            int u = time[0], v = time[1], w = time[2];
            graph[u].emplace_back(v, w);
        }
        vector<int> dist(n + 1, INT_MAX);
        dist[k] = 0;
        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;
        pq.emplace(0, k);

        while (!pq.empty()) {
            auto [time, node] = pq.top();
            pq.pop();
            if (time > dist[node]) continue;
            for (const auto& [neighbor, weight] : graph[node]) {
                int newDist = time + weight;
                if (newDist < dist[neighbor]) {
                    dist[neighbor] = newDist;
                    pq.emplace(newDist, neighbor);
                }
            }
        }
        int maxTime = 0;
        for (int i = 1; i <= n; ++i) {
            if (dist[i] == INT_MAX) return -1;
            maxTime = max(maxTime, dist[i]);
        }
```

```
        return maxTime;
    }
};

int main() {
    Solution solution;

    // Example 1
    vector<vector<int>> times1 = {{2, 1, 1}, {2, 3, 1}, {3, 4, 1}};
    int n1 = 4, k1 = 2;
    cout << "Example 1 Output: " << solution.networkDelayTime(times1, n1, k1) << endl;

    // Example 2
    vector<vector<int>> times2 = {{1, 2, 1}};
    int n2 = 2, k2 = 1;
    cout << "Example 2 Output: " << solution.networkDelayTime(times2, n2, k2) << endl;

    // Example 3
    vector<vector<int>> times3 = {{1, 2, 1}};
    int n3 = 2, k3 = 2;
    cout << "Example 3 Output: " << solution.networkDelayTime(times3, n3, k3) << endl;

    return 0;
}
```

OUTPUT:

```
Example 1 Output: 2
Example 2 Output: 1
Example 3 Output: -1


...Program finished with e
Press ENTER to exit consol
```