Rohit Sahu
22BCS13677
IOT 615-B
Day 7 Questions

Q1. 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.

Q2. Find if Path Exists in Graph
There is a bi-directional graph with n vertices, where each vertex is labeled from 0 to n - 1 (inclusive). The edges in the graph are represented as a 2D integer array edges, where each edges[i] = [ui, vi] denotes a bi-directional edge between vertex ui and vertex vi. Every vertex pair is connected by at most one edge, and no vertex has an edge to itself.

Q3. Minimum Height Trees
A tree is an undirected graph in which any two vertices are connected by exactly one path. In other words, any connected graph without simple cycles is a tree.

Q4. Evaluate Division
You are given an array of variable pairs equations and an array of real numbers values, where equations[i] = [Ai, Bi] and values[i] represent the equation Ai / Bi = values[i]. Each Ai or Bi is a string that represents a single variable.

Q5. All Paths From Source to Target
Given a directed acyclic graph (DAG) of n nodes labeled from 0 to n - 1, find all possible paths from node 0 to node n - 1 and return them in any order.

Code:
```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>
#include <map>
#include <algorithm>
#include <queue>
#include <stack>
using namespace std;

int findCenterOfStar(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];
}

bool validPath(int n, vector<vector<int>>& edges, int source, int destination) {
   vector<vector<int>> adj(n);
   for (auto &e: edges) {
```

```
        adj[e[0]].push_back(e[1]);
        adj[e[1]].push_back(e[0]);
    }
    vector<bool> visited(n,false);
    queue<int>q;
    q.push(source);
    visited[source] = true;
    while (!q.empty()) {
        int u=q.front();
        q.pop();
        if (u==destination) return true;
        for (auto &v: adj[u]) {
            if (!visited[v]) {
                visited[v]=true;
                q.push(v);
            }
        }
    }
    return false;
}

vector<int> findMinHeightTrees(int n, vector<vector<int>>& edges) {
    if (n==1) return {0};
    vector<int> deg(n,0);
    vector<vector<int>> adj(n);
    for (auto &e: edges) {
        adj[e[0]].push_back(e[1]);
        adj[e[1]].push_back(e[0]);
        deg[e[0]]++;
        deg[e[1]]++;
    }
    queue<int> leaves;
    for (int i=0;i<n;i++) {
        if (deg[i]==1) leaves.push(i);
    }
    int count=n;
    while (count>2) {
        int sz=leaves.size();
        count-=sz;
        for (int i=0;i<sz;i++) {
            int leaf=leaves.front();
            leaves.pop();
            deg[leaf]=0;
            for (auto &v: adj[leaf]) {
                if (deg[v]>0) {
                    deg[v]--;
                    if (deg[v]==1) leaves.push(v);
                }
```

```
        }
      }
    }
    vector<int> ans;
    while (!leaves.empty()) {
      ans.push_back(leaves.front());
      leaves.pop();
    }
    return ans;
}

vector<double> calcEquation(vector<vector<string>>& equations, vector<double>& values,
vector<vector<string>>& queries) {
    unordered_map<string,vector<pair<string,double>>> graph;
    for (int i=0;i<equations.size();i++) {
      graph[equations[i][0]].push_back({equations[i][1], values[i]});
      graph[equations[i][1]].push_back({equations[i][0], 1.0/values[i]});
    }
    vector<double> ans;
    for (auto &q: queries) {
      if (!graph.count(q[0]) || !graph.count(q[1])) {
        ans.push_back(-1.0);
        continue;
      }
      if (q[0]==q[1]) {
        ans.push_back(1.0);
        continue;
      }
      unordered_map<string,bool> visited;
      queue<pair<string,double>>Q;
      Q.push({q[0],1.0});
      visited[q[0]]=true;
      double res=-1.0;
      while (!Q.empty()) {
        auto front=Q.front();
        Q.pop();
        string cur=front.first;
        double valCur=front.second;
        if (cur==q[1]) {
          res=valCur;
          break;
        }
        for (auto &nx: graph[cur]) {
          if (!visited[nx.first]) {
            visited[nx.first]=true;
            Q.push({nx.first,valCur*nx.second});
          }
        }
```

```
        }
        ans.push_back(res);
    }
    return ans;
}

void dfsPaths(vector<vector<int>>& graph, int node, vector<int>& path, vector<vector<int>>& ans) {
    if (node == graph.size()-1) {
        ans.push_back(path);
        return;
    }
    for (auto &nx : graph[node]) {
        path.push_back(nx);
        dfsPaths(graph, nx, path, ans);
        path.pop_back();
    }
}

vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {
    vector<vector<int>> ans;
    vector<int> path;
    path.push_back(0);
    dfsPaths(graph,0,path,ans);
    return ans;
}

int main() {
    vector<vector<int>> edges1 = {{1,2},{2,3},{4,2}};
    cout << findCenterOfStar(edges1) << endl;

    vector<vector<int>> edges2 = {{0,1},{1,2},{2,0}};
    cout << (validPath(3, edges2, 0, 2) ? "true" : "false") << endl;

    vector<vector<int>> edges3 = {{1,0},{1,2},{1,3}};
    vector<int> res3 = findMinHeightTrees(4, edges3);
    for (auto &x: res3) cout << x << " ";
    cout << endl;

    vector<vector<string>> eq4 = {{"a","b"},{"b","c"}};
    vector<double> val4 = {2.0,3.0};
    vector<vector<string>> queries4 = {{"a","c"},{"b","a"},{"a","e"},{"a","a"},{"x","x"}};
    vector<double> res4 = calcEquation(eq4, val4, queries4);
    for (auto &x: res4) cout << x << " ";
    cout << endl;

    vector<vector<int>> graph5 = {{1,2},{3},{3},{}};
    vector<vector<int>> res5 = allPathsSourceTarget(graph5);
    for (auto &path : res5) {
```

```
        for (auto &num : path) cout << num << " ";
        cout << endl;
    }

    return 0;
}
```

Output:

```
nnnvqq.xux' '--dbgExe=C:\
2
true
1
6 0.5 -1 1 -1
0 1 3
0 2 3
```