Assignment 7

Mayur Zope SE Comp A 75

There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight take to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the scity. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Check whether the graph is connected or not. Justify the storage representation used.

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

// Graph class using adjacency list representation
class Graph {
private:
  unordered_map<string, list<pair<string, int>>> adjList;

public:
  // Add a flight (edge) between two cities with cost (time/fuel)
  void addFlight(const string& cityA, const string& cityB, int cost) {
    adjList[cityA].push_back({cityB, cost});
    adjList[cityB].push_back({cityA, cost});
  }

  // Perform BFS to check if all nodes are reachable from a starting node
  bool isConnected() {
    unordered_map<string, bool> visited;
    if (adjList.empty()) return true; // empty graph is trivially connected
```

```
    // Get the first node to start BFS
    string startCity = adjList.begin()->first;

    // BFS
    queue<string> q;
    q.push(startCity);
    visited[startCity] = true;

    while (!q.empty()) {
        string city = q.front();
        q.pop();

        // Explore neighbors
        for (auto neighbor : adjList[city]) {
            if (!visited[neighbor.first]) {
                visited[neighbor.first] = true;
                q.push(neighbor.first);
            }
        }
    }

    // If all cities are visited, the graph is connected
    return visited.size() == adjList.size();
}

// Print the graph (adjacency list representation)
void printGraph() {
    for (auto& pair : adjList) {
        cout << pair.first << " -> ";
```

```
            for (auto& neighbor : pair.second) {
                cout << "(" << neighbor.first << ", " << neighbor.second << ") ";
            }
            cout << endl;
        }
    }
};

int main() {
    Graph g;
    int n;

    cout << "Enter the number of flights: ";
    cin >> n;

    // Taking input for flights between cities
    for (int i = 0; i < n; ++i) {
        string cityA, cityB;
        int cost;
        cout << "Enter flight details (CityA CityB Cost): ";
        cin >> cityA >> cityB >> cost;
        g.addFlight(cityA, cityB, cost);
    }

    // Check if the graph is connected
    if (g.isConnected()) {
        cout << "The graph is connected." << endl;
    } else {
        cout << "The graph is not connected." << endl;
```

```
    }

    // Optionally, print the graph
    cout << "\nGraph (Adjacency List Representation):\n";
    g.printGraph();

    return 0;
}
```

// OUTPUT