CS 202: DATA STRUCTURES

ASSIGNMENT 4

GRAPH THEORY

Due Date: April 23, 2021, 11:55 pm

You have been hired by the airline “Sisyphus Airways”. The company runs flights to and from several locations spread throughout the country. But recently, another competitor company has come up. Hence, Sisyphus Airways wishes to reduce its running costs and improve its client experience in order to remain competitive in the market. You are required to help them achieve this.

Thanks to all the Data Structures you’ve been studying, you know how to design a system that takes in the cities and the flights between them, and outputs details like which cities are reachable and what are the shortest paths to them. Let’s get to work!

Task 1: Dataset Parsing 20 Marks

You first need to determine how exactly will you be representing your graph in terms of the underlying data structure.

You can opt for either of the two given data structures to store your **Graph**:

1. Adjacency Matrix
2. Adjacency List

For loading the graph from the file (inside the constructor), you **must use** the **addEdge** function (will make things easier for you). It will be graded.

Note: In order to cater to representations of flights which are either all one-way or two-way, your graph implementation should be able to represent both directed and undirected graphs. You will be passed a flag in the constructor, and depending on its value, the graph should either be a directed graph (if the flag is set to true) or an undirected graph (if the flag is set to false).

In the addEdge function, if the flag indicates an undirected graph, then e.g. for A B 60, you will store both (A,B,60) and (B,A,60). If the flag indicates an directed graph, then e.g. for A B 60, you will only store (A,B,60).

Please note that your code for parsing should be generalized enough to work on all datasets based on the same format as that provided to you. Corresponding to this task you are required to build a **display** function which returns a **string**, meaning that the entire graph will be displayed as a single string. For reference check the example below:

For a graph having 4 cities and 5 flights with format:

[City1] [City2] [Weight]

INPUT:

```
n 4
c 5
A B 60
A C 10
B D 25
C B 5
D A 12
```

OUTPUT OF DISPLAY FUNCTION (undirected graph):

```
(A,B,60) (A,C,10) (A,D,12)
(B,A,60) (B,C,5) (B,D,25)
(C,A,10) (C,B,5)
(D,A,12) (D,B,25)
```

OUTPUT OF DISPLAY FUNCTION (directed graph):

```
(A,B,60) (A,C,10)
(B,D,25)
(C,B,5)
(D,A,12)
```

A tricky part will be to decide how to map the location name ‘A’ to the required entry in your graph data structure.

Note: You are encouraged to make a helper function that can help you with that. You can add any other helper functions in Graph.h for future parts as well.

Deliverables: Graph constructor (loads file), addEdge and display function.

Task 2: Connectivity Check 20 Marks

Sisyphus Airways requires you to check if certain cities can be reached from specific cities through the flights that they offer.
Once your graph is ready, it needs to check whether two cities are connected via a flight or not i.e. if a flight path exists from one city to another city through a series of flights.

You are, therefore, required to fill a **Reachable Function,** which checks if, in a given graph, one city can be reached from another. The function should return a **Boolean** meaning that it only shows if such a path exists or not.
There are several ways to check reachability. You need to perform a simple traversal of the graph. As a hint, there is a traversal that checks depth and another for breadth. You need to implement either one. There you go, we just did half of the work for you.

Note: Do NOT use the Dijkstra's algorithm for reachability. You will score 0, if you do so. Choose another way – we gave you a hint above.

Deliverables: Reachable function.

Task 3: Shortest Path Between Cities 30 Marks

Passengers use Sisyphus Airways to travel to multiple cities across the country. Sisyphus Airways wants to improve their customers' overall experience by devising routes which minimise their flight times. Hence, Sisyphus Airways has assigned you the task of implementing an algorithm to calculate the shortest possible routes between cities.

From your data structures class, you remember your instructor talking about the **Dijkstra's shortest path algorithm** and decide to use that.
Dijkstra's shortest path algorithm is one of the most famous algorithms for finding the shortest path between two nodes, and in your case, two cities. If your supervisor tries to find a path between two nodes which are not connected, the system should cater to it. The algorithm only needs to return **the weight** of the shortest path, and not the path itself. If no path exists, **return -1.**

Note: You can assume that the graph provided to you in the test file for this part will be an undirected graph and that all weights are non-negative.

Task 4: Ordering Cities In Flight Routes 30 Marks

Certain flight routes require a specific order of cities to stop at temporarily during the journey. You are required to arrange the cities in a given directed graph while preserving their order relative to the overall flight routes. Topological sorting seems the appropriate algorithm to use here, and so you decide to solve the given task by implementing it.

You need to implement the topoSort function, which needs to sort the cities topologically and return the sorted cities. Since there are multiple possible correct orderings, the test files will check your output against all possible orderings for that graph. Your output only needs to match one possible ordering in order to pass the tests.

Note: You can assume that the graphs provided to you in the test file for this part will be directed graphs and have no directed cycles.

Test files

Use g++ test*.cpp -std=c++11, where * is task number, to compile the test case.
And then "./a.out" to run the file.

Caution

Sisyphus Airways does not appreciate instances of malpractice such as hardcoding your way through tasks. Ensure that you spend more time on completing the assignment than exploring a risky way out of it.

Submission Guidelines

You will only be required to submit the **Graph.h** and **Graph.cpp** files in the form of a **zip** archive with the following naming convention: "PA4_ <your_roll_number>.zip".

Good Luck!