CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf, Trivedi

Due Wednesday, April 15, 11:59PM

Assignment 9 - Graph II

OBJECTIVES

1. **Applications of Depth First Traversal**
2. **Dijkstra's Shortest Path Algorithm**

Overview

In this assignment, you will perform Depth First traversal for finding connected cities in a graph and implement Dijkstra's algorithm to find the shortest path between cities.

Graph Class

Your code should implement graph traversal for cities. A header file that lays out this graph can be found in **Graph.hpp** on Moodle. *As usual, do not modify the header file. You may implement helper functions in your .cpp file if you want as long as you don't add those functions to the Graph class.*

Your graph will utilize the following struct:

```
struct vertex;

struct adjVertex{
    vertex *v;
    int weight;
};

struct vertex{
    string name;
    bool visited;
    int distance;
    vertex *pred;
    vector<adjVertex> adj;
};
```

CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf, Trivedi

Due Wednesday, April 15, 11:59PM

NOTE (1): Coderunner will set **`v->visited = false; v->distance = 0; v->pred = nullptr;`** for all vertices **`v`** before calling depthFirstTraversal, dijkstraTraverse and shortestPath methods.

NOTE (2): In order to avoid issues with Coderunner for valid traversal sequences, process adjacent nodes in the order that they appear in the adjacency list, i.e., process nodes by going through the adjacency list per node from index 0 to the size of the adjacency list.

You need to implement the following methods:

void addEdge(string v1, string v2, int num);

- ➔ Make a connection between v1 and v2 (same as last assignment). **Important** point here is to **add weights to the edges**. Each edge will have a corresponding weight attached to it. *e.g.* addEdge("Aurora", "Bloomington", 5);

void depthFirstTraversal(string sourceVertex):

- ➔ Use Depth first traversal from sourceVertex to traverse the graph.
- ➔ Print the city name and corresponding distance from the sourceVertex. Use the following format for printing:

```
// for printing the path found through DF Traversal (each node)
cout << n->name << " -> ";
// print “DONE” at the end when all the cities have been visited.
cout << "DONE";
```

void dijkstraTraverse(string start):

- ➔ Use Dijkstra's algorithm to compute the single source shortest path in the graph from the start city to all other nodes in its component.
- ➔ **NOTE:** You need to update the **`distance`** and **`pred`** attributes for each **`v`**.

void shortestPath(string s1, string s2):

- ➔ Print the shortest path found between city (s1) and city (s2) by invoking dijkstraTraverse.

CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf, Trivedi

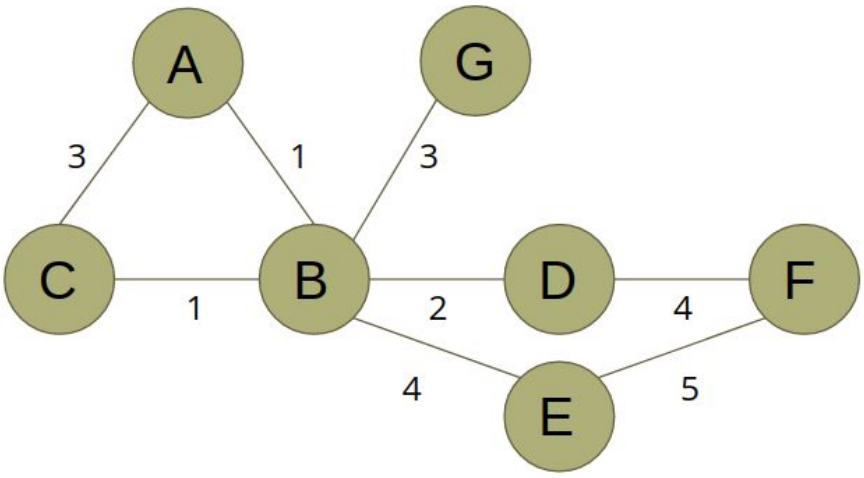
Due Wednesday, April 15, 11:59PM

➔ Print the city names on the path from city (s1) to city (s2), inclusive. Use the following format for printing:

```
// for printing the path found through dijkstraTraverse (each node)
cout << n->name << " -> ";
// print “DONE” and path distance at the end
cout << "DONE " << "[" << distance << "]" << endl;
```

NOTE: Once you are done with your assignment on code runner you need to click on 'Finish attempt' and the 'Submit all and finish'. If you don't do this, you will not get graded.

Example Run: For the given the following Graph and the corresponding adjacency list



A	→ B (1) → C (3)
B	→ A (1) → C (1) → D (2) → E (4) → G (3)
C	→ A (3) → B (1)
D	→ B (2) → F (4)
E	→ B (4) → F (5)
F	→ D (4) → E (5)
G	→ B (3)

CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf, Trivedi

Due Wednesday, April 15, 11:59PM

```
>> depthFirstTraversal("A")

[Output]
A -> B -> C -> D -> F -> E -> G -> DONE
```

>> Before calling `dijkstraTraverse("A")`

Vertex	A	B	C	D	E	F	G
Distance	0	0	0	0	0	0	0
Pred	NULL	NULL	NULL	NULL	NULL	NULL	NULL

>> After calling `dijkstraTraverse("A")`

Vertex	A	B	C	D	E	F	G
Distance	0	1	2	3	5	7	4
Pred	NULL	A	B	B	B	D	B

```
>> shortestPath("A", "F");

[Output]
A -> B -> D -> F -> DONE [7]

>> shortestPath("A", "C");

[Output]
A -> B -> C -> DONE [2]
```