CSCI 2270 – Data Structures

Instructors: Ashraf, Godley

Due Tuesday, July 14, 11:59PM

Assignment 5 - Graph

OBJECTIVES

1. **Dijkstra's Shortest Path Algorithm**

Overview

In this assignment, you will 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: Ashraf, Godley

Due Tuesday, July 14, 11:59PM

NOTE (1): Coderunner will set `v->visited = false; v->distance = 0; v->pred = nullptr;` for all vertices `v` before calling the dijkstraTraverse 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.

We have implemented the following methods for you:

`void addVertex(string name);`

➔ Add new vertex 'name' to the graph.

`void displayEdges();`

➔ Display all the edges in the graph.

Format for printing:

If we create a graph with the following structure

```
graph.addVertex("Boulder");
graph.addVertex("Denver");
graph.addVertex("Las Vegas");

graph.addEdge("Boulder", "Denver");
graph.addEdge("Las Vegas", "Denver");
```

We print the edges in the following manner.

```
Boulder --> Denver
Denver --> Boulder Las Vegas
Las Vegas --> Denver
```

The order of vertices printed is the same as the order in which they were added to the graph. Similarly, the order of vertices to the right of "-->" sign is the same as the order in which the corresponding edge was added to the graph.

`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 _grader_setAllVerticesUnvisited();`

CSCI 2270 – Data Structures

Instructors: Ashraf, Godley

Due Tuesday, July 14, 11:59PM

```
void _grader_print_shortest_distance(string destination);

vertex* searchVertex(string start, vector<vertex*> &vertices)
```

You will need to implement the following method:

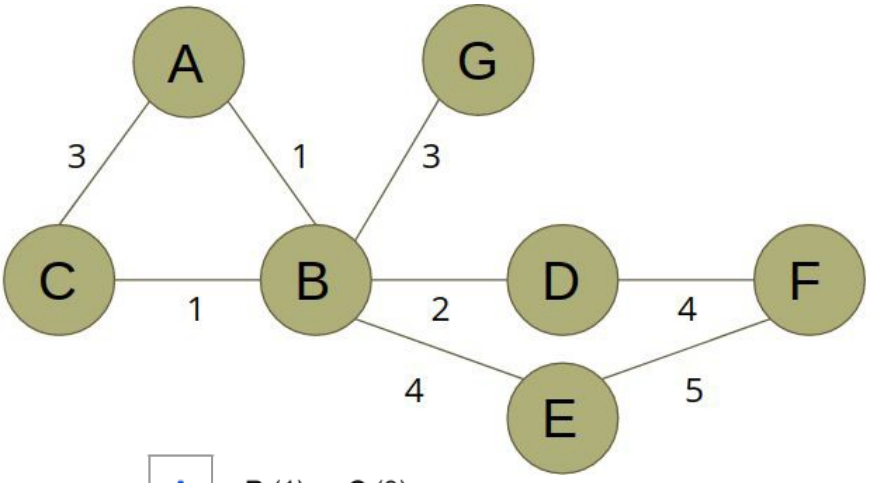
```
void dijkstraTraverse(string start);
```

- ➔ 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.
- ➔ You must use searchVertex() from the source, which has been provided to you within Graph.cpp.
- ➔ **NOTE:** You need to update the **`distance`** and **`pred`** attributes for each **`v`**.

NOTE 1: Do not modify any code provided to you other than dijkstraTraverse(). So, do not change any of the code within Graph.cpp or Graph.hpp.

NOTE 2: Once you are done with your assignment in Coderunner, you need to click on 'Finish attempt' and the 'Submit all and finish'. If you don't do this, your solution will not be 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: Ashraf, Godley

Due Tuesday, July 14, 11:59PM

>> 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