CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf

Due on Nov 11, 11:59PM

Assignment 9 - Graph II

OBJECTIVES

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

Overview

For this assignment, we would like you to

1. Find the connected cities of a city network using Depth First Traversal i.e., find the connected components of a graph using Depth First Traversal.
2. Implement Dijkstra's algorithm to find the shortest-path from any city (Source) to every other city.

Graph Class

A header file named **Graph.hpp** that lays out the city network representation as a Graph ADT is in Moodle. Similar to the recitation/assignments, we use the following structs to represent **vertex**, **adjVertex**

```
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,

The following methods must be implemented

`void createEdge(string v1, string v2, int num);`

- ➔ Make a connection between `v1` and `v2` (same as last assignment).
- ➔ The 3rd variable is the weight of the edge. If it takes 5 units to travel from `Aurora` to `Bloomington`, you call `addEdge("Aurora", "Bloomington", 5);`

`void depthFirstTraversal(string sourceVertex);`

- ➔ Use Depth First Traversal with the city named in the variable `sourceVertex` as the starting point to traverse the city network (graph).
- ➔ For every city encountered using DF Traversal, starting from the `sourceVertex` your code must print the name of the city in the exact format as shown below.

```
// 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 traverseWithDijkstra(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 don't print anything in this function. Storing the `distance` associated with every node relative to the `start` node is enough.

`void minDistPath(string s1, string s2);`

- ➔ Print the shortest path found between city (`s1`) and city (`s2`) by invoking `traverseWithDijkstra`.
- ➔ Print the city names on the path from city (`s1`) to city (`s2`), both inclusive in the following format

```
// for every node in the path found through traverseWithDijkstra
cout << n->name << " -> ";
// print "DONE" and path distance at the end
cout << "DONE " << "[" << distance << "]" << endl;
```

CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf,

Precautions

1. Do not modify the header file.
2. Coderunner will set `v->visited = false; v->distance = 0; v->pred = nullptr;` for all vertices v before calling `depthFirstTraversal`, `traverseWithDijkstra` and `minDistPath` methods.
3. 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.
4. 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.

Suggestions

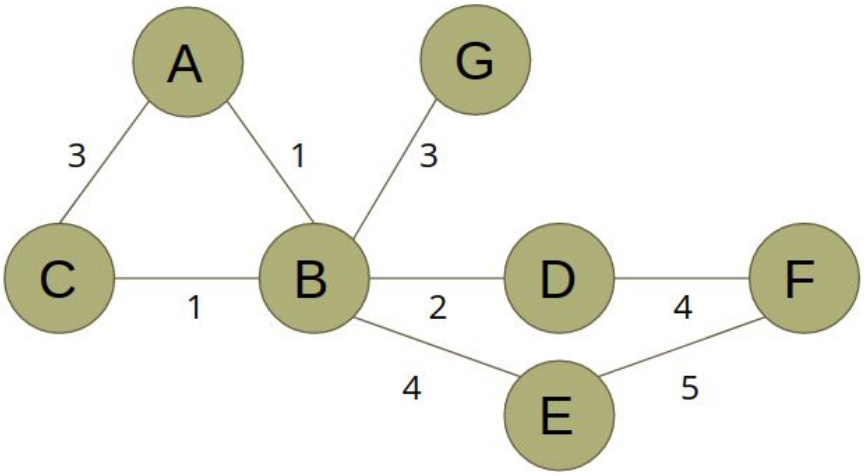
1. You may add helper functions in the Graph.cpp file to make your code module.

P.T.O

CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf,

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)

>> depthFirstTraversal("A")

[Output]

A -> B -> C -> G -> D -> F -> E -> DONE

>> Before calling traverseWithDijkstra("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 `traverseWithDijkstra("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

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

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

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

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