Computer Science 245

Final Project

Dijkstra's Algorithm

Due Date: Friday, December 17th, 12:00 PM

50 Points

Objective

The purpose of this assignment is to have students implement a client file using the operations of the `Graph` class to produce a summary table of the shortest paths from a vertex using Dijkstra's Algorithm, and check for any cycles that may exist in the graph.

Assignment Summary

Build a directed graph of vertices from the data stored in an input file, print out a listing of vertices contained in the graph in sorted order (three per line), produce a summary table of the shortest paths from a starting vertex input by the user to all other vertices.

Create a preliminary analysis & design with problem description, specification, client algorithm, and outline of the `Graph` class. Discuss the data structures used to build the graph. Also include a short, sample input and output file different from the one shown with this assignment, and be sure you explain the logic of Dijkstra's algorithm very clearly. Name this file `README` and store in your project directory.

Program Files and Input

You may copy over several files for use with this program by typing :

```
cp /pub/digh/CSC245/Prog4/* .
```

The input file should be provided on the command line. Flag an incorrect command line input or an invalid filename, and exit.

The input file contains two or more different route possibilities. Each line of input will be in the following format :

`Origin;Destination;Trip-length`

You should assume that the `Origin` and `Destination` cities are strings of maximum length 30 which are delimited by a semicolon. These strings may or may not have blanks in them. `Trip-length` is an integer representing the weight of the edge between the two cities.

The user will be inputting the starting vertex at the keyboard. If they enter the name of a vertex which is not available, inform them so, and allow them to re-enter the value.

You should name your client file `dijkstra.cpp` and use the operations of the `Graph` class as seen in your class handouts and in the file `graph.h` which is provided. You do not need to modify the `Graph` class in any way. The only operations you will really be using from this class are `AddVertex` and `AddEdge` when building the graph and `GetToVertices` and `WeightIs` with your Dijkstra's Algorithm.

`VertexType` is our template data type in the Graph class. Use `string` in place of it in your client file. For example,

```
Graph<string>  myGraph(50);
Queue<string> myQ(50);
```

Summary of Dijkstra Algorithm Logic

After inputting your data, building your graph, and storing your vertices into some container like a vector or list, use the following general logic to implement Dijkstra's Algorithm.

1. Determine how many vertices have been input. Assign this to some integer.
2. Declare and initialize a structure that can support each of the following components. You could use a series of four parallel arrays or a single array of structs.
   - The strings representing the different vertices.
   - The booleans to "mark" a vertex once the weight of a vertex has been chosen as the smallest among the current unmarked ones. All values in this array should start out as false.
   - The integers representing the total distance used in reaching this vertex. All values in this array should probably start out with a value like `INT_MAX` so they will not be considered among the current smallest of those already initialized.
   - The strings representing the previous vertex visited before reaching a vertex.
3. Determine the index within your local container of vertices where the starting vertex is located. Mark it as visited and flush its distance to 0. Print out the data corresponding to this index with "N/A" used as the previous vertex.

4. Determine the vertices which are adjacent to your current vertex using the `GetToVertices` operation.

5. Now, for each vertex that is adjacent to your current vertex, determine the weight value that exists between each one and your current vertex. If an adjacent vertex is currently unmarked and its distance value in the table is greater than the sum of the weight value plus the distance of the last marked vertex (your current vertex), reset its distance value to this smaller sum. Store the name of the current vertex as the previous vertex of each adjacent vertex.

6. Find the minimum distance value among all unmarked vertices, and set the vertex that corresponds to this minimum distance as your new current vertex. Mark this vertex.

7. Print out the data corresponding to your current marked vertex.

8. Repeat the previous four steps above until all of your vertices have been marked.

This should not be coded as one long function, but rather a collection of several functions. I strongly recommend writing a function for each of the following tasks :

- Determining the index value a given string is within your array of strings.
- Initializing your array(s).
- Printing a given row in your summary table.
- Finding the minimum value among your current unmarked vertices.
- Determining whether a string entered by the user is a valid vertex.

You have three data files provided as sample input. The data in `ginfile1.dat` corresponds exactly to the graph of different cities shown in your class handout.

Hand In

Create a typescript file and perform each of the following steps :

1. provide a sample compilation
2. provide a sample run with input

Finishing Up

Move your excecutable file into a file named `RELEASE` using the `mv` command.

You now need to move your code into your `scratch` directory where you replace `last_fm` with your cobra username.

```
cp RELEASE /scratch/csc245/last_fm/Prog4
cp README /scratch/csc245/last_fm/Prog4
cp typescript /scratch/csc245/last_fm/Prog4
cp *.h /scratch/csc245/last_fm/Prog4
cp *.cpp /scratch/csc245/last_fm/Prog4
```

Extra Credit. You will receive three extra credit points if you complete this program by Friday, December 10th at 11:59 PM. You will also receive extra points for printing a cycle path. This may be added through the due date.

Sample Run

```
        ^^^^^^^^^^^^^^^^    DIJKSTRA'S ALGORITHM    ^^^^^^^^^^^^^^^^

        A Weighted Graph Has Been Built For These 7 Cities :

                   Atlanta             Austin           Chicago
                    Dallas             Denver           Houston
                Washington

        Please input your starting vertex: Washington
        ------------------------------------------------------------------
                   Vertex                Distance                 Previous

               Washington                       0                      N/A
                  Atlanta                     600               Washington
                   Dallas                    1300               Washington
                  Houston                    1400                  Atlanta
                   Austin                    1500                   Dallas
                   Denver                    2080                   Dallas
                  Chicago                    2200                   Dallas

        ------------------------------------------------------------------
                         The graph contains a cycle

          The Cycle is :
                    Vertex                        To
                  Atlanta          ->             Washington
               Washington          ->                 Dallas
                   Dallas          ->                 Denver
                   Denver          ->                Atlanta
```