Dijkstra's Algorithm

COP 4530 Programming Project 4

Instructions

For Programming Project 4, you will be implementing an undirected weighted Graph ADT and performing Dijkstra's Algorithm to find the shortest path between two vertices. Your graph can be implemented using either an adjacency list, adjacency matrix, or an incidence matrix. Your graph will implement methods that add and remove vertices, add and remove edges, and calculate the shortest path. The priority queue data structures must be implemented by you. You cannot use the standard library priority queue data structures.

A large portion of this assignment is researching and implementing Dijkstra's Algorithm. There is information about this algorithm in your textbook and widely available on the web.

Working in teams of four, your project must consist of codes, report (2-3 pages), presentation, zipped together.

Abstract Class Methods

`void addVertex(std::string label)`
Creates and adds a vertex to the graph with label. No two vertices should have the same label.

`void removeVertex(std::string label)`
Removes the vertex with label from the graph. Also removes the edges between that vertex and the other vertices of the graph.

`void addEdge(std::string label1, std::string label2, unsigned long weight)`
Adds an edge of value weight to the graph between the vertex with label1 and the vertex with label2. A vertex with label1 and a vertex with label2 must both exist, there must not already be an edge between those vertices, and a vertex cannot have an edge to itself.

`void removeEdge(std::string label1, std::string label2)`
Removes the edge from the graph between the vertex with label1 and the vertex with label2. A vertex with label1 and a vertex with label2 must both exist and there must be an edge between those vertices

`unsigned long shortestPath(std::string startLabel, std::string endLabel, std::vector<std::string> &path)`
Calculates the shortest path between the vertex with startLabel and the vertex with endLabel using Dijkstra's Algorithm. A vector is passed into the method that stores the shortest path between the vertices. The return value is the sum of the edges between the start and end vertices on the shortest path.

Examples

Below in an example of the functionality of the implemented graph:

```
std::vector<std::string> vertices1 = { "1", "2", "3", "4", "5", "6" };
std::vector<std::tuple<std::string, std::string, unsigned long>>
edges1 = { {"1", "2", 7}, {"1", "3", 9}, {"1", "6", 14}, {"2", "3",
10}, {"2", "4", 15}, {"3", "4", 11}, {"3", "6", 2}, {"4", "5", 6},
{"5", "6", 9} };

g.shortestPath("1", "5", path); // == 20
g.shortestPath("1", "5", path); // = { "1", "3", "6", "5" }
```

Deliverables

Please submit complete projects as zipped folders. The zipped folder should contain:

- Graph.cpp (Your written Graph class)
- Graph.hpp (Your written Graph class)
- GraphBase.hpp (The provided base class)

```
#ifndef GRAPHBASE_H
#define GRAPHBASE_H

#include <vector>
#include <string>

class GraphBase {
  virtual void addVertex(std::string label) = 0;
  virtual void removeVertex(std::string label) = 0;
  virtual void addEdge(std::string label1, std::string label2, unsigned long weight) =
0;
  virtual void removeEdge(std::string label1, std::string label2) = 0;
  virtual unsigned long shortestPath(std::string startLabel, std::string endLabel,
std::vector<std::string> &path) = 0;
};

#endif
```

- main.cpp (your written file)

  Report

  Slides

And any additional source and header files needed for your project.

Hints

Though it may be appealing to use an adjacency matrix, it might be simpler to complete this algorithm using an adjacency list for each vertex.

I suggest using a separate class for your edge and vertex.

Remember to write a destructor for your graphs!

Rubric

Any code that does not compile will receive a zero for this project.

Criteria	Points
Codes (Code implements Dijkstra's Algorithm, Code uses object-oriented **design principles (Separate headers and sources, where applicable), Code is well documented). Outputs are the vector of shortest path and total cost.**	50
Your implementation of ADT priority queue data structures.	**10**
Report (well formatted)	20
Presentation	20
Total Points	**100**