Dijkstra's Algorithm

Instructions

For Programming Project 5, 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. Each vertex in your graph will have a string label that will help identify that vertex to you and the test file.

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.

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} };

for (const auto label : vertices1) g.addVertex(label);  for
(const auto &tuple : edges1) g.addEdge(std::get<0>(tuple),
std::get<1>(tuple), std::get<2>(tuple));

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