# Find the shortest distance using Dijkstra's Algorithm.

## Objectives

* Extend your weighted graph into a undirected graph.
* Find the shortest path between two vertices.

## Overview

This assignment focuses on Undirected Graphs. In this assignment, a railroad company company called "Ticket to Ride" wants to build new tracks across the United States. Your task is to figure out the shortest distance between any two cities on their map in order to help build their advertising for the train service.

Note: Parts of this assignment are going to be difficult.

## Part 1: Update WeightedGraph

Update your WeightedGraph class.

We have to make a few changes to WeightedGraph. Set both set methods to be virtual. That's it. Make that small change and you are good to go for Part 2.

virtual void set(int, int, int); // Sets index i,j to weight  
virtual void set(int, int); // Calls "set(i,j,weight)" with weight=1

## Part 2: Write UndirectedGraph

Build an UndirectedGraph class which extends your WeightedGraph class for an directed graph. Here's my header file undirectedgraph.h. There are no private data members and the constructor is empty. This looks shorter, but will be significantly harder than anything you've been asked to write this semester.

#pragma once  
#include <limits.h>  
#include "weightedgraph.h"  
#include "linkedlist.hpp"  
  
class UndirectedGraph :  
 public WeightedGraph  
{  
public:  
 UndirectedGraph(int mVertices);  
  
 void set(int, int, int); // Sets a vertex at both (i,j) and (j,i) with a weight.  
 void set(int, int); // Sets a vertex at both (i,j) and (j,i) with 1.  
 // This method should call the 3-parameter set method  
 // as set(i,j,1);  
  
 LinkedList<int> dijkstra(int, int) const;  
 // Returns the path from "start" to "finish".  
 // The first value in the returned list should be start.  
 // The last value in the returned list should be finish.  
};

Here's my constructor. That's it. It calls the WeightedGraph parent.

UndirectedGraph::UndirectedGraph(int mVertices) : WeightedGraph(mVertices)  
{   
}

## Part 3: Write Dijkstra's Algorithm

This is a difficult algorithm to write. Hopefully this pseudocode will make things easier.

* Create an integer array of length Number of Vertices named distances.
* Create an integer array of length Number of Vertices named previous.
* Create a Boolean array of length Number of Vertices named unreached.
* Set all of distances to be INT MAX.
* Set all of previous to be -1.
* Set all of unreached to be true.
* Set distance[start] to be 0.
* Create an integer unreachedCount set to Number of Vertices.
* while unreachedCount greater than 0
  + Find the index position of the smallest element in distances where the matching index of unreached is also true. This index position is named vertex.
  + Set unreached[vertex] to be false.
  + Decrement unreachedCount by 1.
  + For each vertex v which is a neighbor to to vertex.
    - If the weight at (vertex,v) is greater than 0 and weight(vertex,v) + distance[vertex] is less than distances[v]...
      * Update distances[v] to be the weight at (vertex,v) + distance[vertex].
      * Update previous[v] to be vertex.
* Create a linked list of ints.
* set a int vertex to finish.
* while vertex does not equal -1:
  + push vertex onto the front of the linked list
  + set vertex as previous[vertex]
* Delete all of your dynamically created arrays.
* Return the linked list

## Part 4. Write the Program.cpp file.

Ask the user for the following:

* The number of vertices in the graph.
* The names of each vertex (you can store this in a vector, linked list, or array).
* The number of edges in the graph.
* The start index, end index, and weight of each edge. Because this is an undirected graph, start and end can be entered in either order.
* The start index of a route (as an integer).
* The end index of a route (as an integer).

The program should print back the name of each city and its index, the original graph (which will always be symmetric), then the shortest path from start to end using the names of the index positions, and the sum of the edges from start to end.

Two sample input files will be supplied which you can pipe to your program.

### Example 1.

3  
a  
b  
c  
  
3  
0 1 2  
0 2 3  
1 2 4  
  
1  
2

Here's the output of the program based on the above input file. I've piped everything directly to the program via the command line.

Number of vertices in our matrix: Enter the name of place 0: Enter the name of place 1: Enter the name of place 2: Number of edges in our matrix: Enter a starting vertex number: Enter an ending vertex number: Enter a weight: Enter a starting vertex number: Enter an ending vertex number: Enter a weight: Enter a starting vertex number: Enter an ending vertex number: Enter a weight: Enter the starting location vertex number: Enter the finishing location vertex number:   
  
Vertices:  
0: a  
1: b  
2: c  
  
Original Graph.  
 0 2 3  
 2 0 4  
 3 4 0  
  
Path from b to c.  
1: b  
2: c  
  
Distance: 4  
Done.

This is identical to the run above, but I've typed the values at the prompts.

Number of vertices in our matrix: 3  
Enter the name of place 0: a  
Enter the name of place 1: b  
Enter the name of place 2: c  
Number of edges in our matrix: 3  
Enter a starting vertex number: 0  
Enter an ending vertex number: 1  
Enter a weight: 2  
Enter a starting vertex number: 0  
Enter an ending vertex number: 2  
Enter a weight: 3  
Enter a starting vertex number: 1  
Enter an ending vertex number: 2  
Enter a weight: 4  
Enter the starting location vertex number: 1  
Enter the finishing location vertex number: 2  
  
0: a  
1: b  
2: c  
Original Graph.  
 0 2 3  
 2 0 4  
 3 4 0  
Path from b to c.  
1: b  
2: c  
Distance: 4  
Done.

### Example 2.

The only difference between the first and second example is a slight change to the graph.

3  
a  
b  
c  
  
3  
0 1 2  
0 2 1  
1 2 4  
  
1  
2

Here's the output.

Number of vertices in our matrix: Enter the name of place 0: Enter the name of place 1: Enter the name of place 2: Number of edges in our matrix: Enter a starting vertex number: Enter an ending vertex number: Enter a weight: Enter a starting vertex number: Enter an ending vertex number: Enter a weight: Enter a starting vertex number: Enter an ending vertex number: Enter a weight: Enter the starting location vertex number: Enter the finishing location vertex number:   
  
Vertices:  
0: a  
1: b  
2: c  
  
Original Graph.  
 0 2 1  
 2 0 4  
 1 4 0  
  
Path from b to c.  
1: b  
2: a  
3: c  
  
Distance: 3  
Done.

## Part 5.

Final Test. Our company "Ticket to Ride" wants the following cities in the country connected: Atlanta, Chicago, Dallas, Detroit, Los Angeles, Miami, New Orleans, New York, San Francisco, Seattle, Nashville, and Tampa.

Plot the route from San Francisco to New York. If your program generates the expected route and overall shortest distance, I'll replace any previous homework score with a 100.

## Files

Here are the files that should go into your final submission.

Main.cpp  
WeightedGraph.cpp  
WeightedGraph.h  
UndirectedGraph.cpp  
UndirectedGraph.h  
LinkedList.hpp  
Node.hpp