CSE 331 - Project 6
Weighted Digraphs

TA: James Daly
dalyjame@msu.edu
Due Date: 11:59 pm Dec 5, 2014

1 Project Description

In this project, you will complete an implementation of a weighted directed graph (digraph). You are given skeleton code (i.e., Main.cpp) that performs reading inputs and printing outputs as well as a class declaration for a weighted digraph (WeightedDigraph.h) and stub implmentation (WeightedDigraph.cpp). Your job is to complete the implementation of the following methods in the `WeightedDigraph` class:

1. `InsertArc`: Adds an arc of the given weight between the two vertices.
2. `GetOutDegree`: The number of arcs leaving the given vertex.
3. `GetArcWeight`: Finds the weight of an arc between two vertices.
4. `GetPathWeight`: Finds the sum weight of a directed path.
5. `AreConnected`: Checks whether an arc exists between two vertices.
6. `DoesPathExist`: Checks whether a directed path exists between two vertices.
7. `IsPathValid`: Determines whether an arc exists along each step of the path.
8. `FindMinimumWeightedPath`: Finds a path of minimum sum weight between two vertices. This can be done with Dijkstra's algorithm.

You may use any classes from the STL that you need to store your digraph. You may modify the constructor and destructor to initialize and cleanup these structures. Note that the constructor uses the `InsertArc` method to set itself up. You should probably ensure that your structure is properly initialized before this happens. Your memory requirements must be $O(|V| + |A|)$.

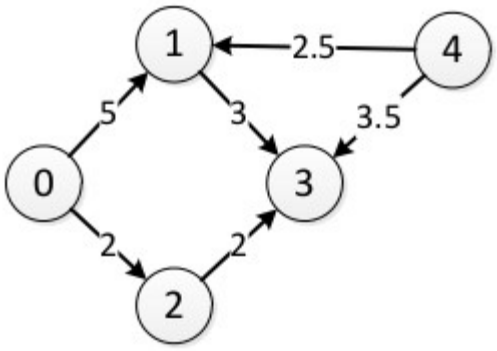
The `InsertArc` is the only method that is not marked `const` and it is a private method. As such, once constructed, your digraph will be immutable. You will lose points if you modify the digraph declaration such that it is not immutable.

Your `InsertArc`, `GetOutDegree`, `GetArcWeight`, and `AreConnected` methods must all run in constant time. `GetPathWeight` and `IsPathValid` should run in time linear in the length of the input `list`. Each of them should use a fixed amount of extra memory. The `DoesPathExist` method should use at most $O(|A|)$ time and $O(|V|)$ extra memory. The `FindMinimumWeightedPath` method should use $O(|A| + |V| \log |V|)$ time and $O(|V| + |A|)$ extra memory.

You may assume that the paths input into `GetPathWeight` and `IsPathValid` are non-empty (they contain at least one vertex). You may not assume that all of the arcs in the path exist. Both `GetArcWeight` and `GetPathWeight` must return infinity if an arc is missing. You may assume that `FindMinimumWeightedPath` will only be called if a path exists. In that function, you may handle this case in whatever manner you choose, including crashing.

```
5
0 1 5.0
0 2 2.0
2 3 2.0
1 3 3.0
4 1 2.5
4 3 3.5
```

(a)
g1.txt



(b) Visual representation of g1

```
>./ugraph input/g1.txt 0 3
The graph has 5 vertices and 6 edges
Path from 0 to 3:
0 -> 2 -> 3
Path weight: 4

>./ugraph input/g1.txt 3 0
The graph has 5 vertices and 6 edges
No path from 3 to 0 exists!
```

(c) Example output

Figure 1: Sample Input

You must provide your own code; you may not copy another's code. Remember to read the project guidelines before you start coding. Remember to update the numArcs variable when adding arcs. Leaking memory or other related issues will negatively impact your score.

Note that not all of the above methods are directly tested by the supplied test harness in Main.cpp, although you will probably want to use the untested methods to implement the tested ones. It is your responsibility to ensure that all of the required methods work properly. **If any of the required methods do not compile, your project will not be graded.**

2 Input Test Cases

The program has three inputs. The first is a file containing the digraph. The first line of the file contains the number of vertices in the graph. Each remaining line contains three values separated by spaces which represent an arc. The first is the source vertex for the arc. The second is the destination vertex of the arc. The third is the arc weight. Your program will read the strings from the file and store them inside of the set (the constructor will do this for you if you implement `InsertArc`). The other two inputs are vertex numbers.

Your program will then construct a weighted digraph from the file. It will then determine if a path from the first vertex to the second exists. If so, it will find such a path of minimum weight and compute its weight. The included main function will do this for you.

3 Project Deliverables

The following files must be submitted via Handin no later than 11:59 pm Dec 5, 2014.

1. `WeightedDigraph.h` - contains the declaration for the `WeightedDigraph` class.
2. `WeightedDigraph.cpp` - contains your implementation of the `WeightedDigraph` class.

4 Written Questions

There are no written questions for this project.