Programming Assignment 11

Due 4 Dec @ 11:59pm

Write a program that implements Dijkstra's single source, shortest paths algorithm. The WeightedDigraph is provided that represents a weighted, directed graph (the edges, also provided, are WeightedDiedges). When adding edges, you do not need to worry about self-loops, parallel (duplicate) edges, or cycles.

The DijkstraSP.java template is provided. You will need to ensure that the passed in graph does not have any edges with negative weight. You must properly implement the API's indicated in the javadoc. NOTE: You should only modify the DijkstraSP.java file.

The main method reads in a file (formatted below) and the source vertex to conduct the search from. It will create the graph, call your search method, and print out the results to standard output.

Grading Notes

You must:

- Use the template provided for you
- Have a style (indentation, good variable names, etc.)
- Comment your code well (no need to over do it, just do it well)

You may not:

- Make your program part of a package.
- Use *code* from anywhere except your own brain.

Submission Instructions:

- Name a folder with your gmu username
- Put your java files in the folder (but not your .class)
- Zip the folder (not just the files) and name the zip “username-pa2.zip”
- Submit to blackboard

Grading Rubric

No Credit:

- Non-submitted assignments
- Late assignments
- Non-compiling assignments
- Non-independent work

1pt	Submission Format
1pt	Style and Comments
6pts	DijkstraSP algorithm
1pt	distTo, hasPathTo
1pt	pathTo

Example input file "graph1.txt":

First line is number of vertices, subsequent lines are the edges. The edges are integers from and to followed by a double that is the weight.

```
7
0 1 2.0
0 3 1.0
1 4 10.0
1 3 3.0
2 0 4.0
2 5 5.0
3 4 2.0
3 2 2.0
3 5 8.0
3 6 4.0
4 6 6.0
6 5 1.0
```

Example DijkstraSP Run

```
>  java DijkstraSP graph1.txt 0
Graph: V=7 E=12
[0] neighbors=2: (0->3=1.00), (0->1=2.00)
[1] neighbors=2: (1->3=3.00), (1->4=10.00)
[2] neighbors=2: (2->5=5.00), (2->0=4.00)
[3] neighbors=4: (3->6=4.00), (3->5=8.00), (3->2=2.00), (3->4=2.00)
[4] neighbors=1: (4->6=6.00)
[5] neighbors=0:
[6] neighbors=1: (6->5=1.00)
Paths to source: 0
  path for 0 :
  path for 1 : (0->1=2.00)
  path for 2 : (0->3=1.00)(3->2=2.00)
  path for 3 : (0->3=1.00)
  path for 4 : (0->3=1.00)(3->4=2.00)
  path for 5 : (0->3=1.00)(3->6=4.00)(6->5=1.00)
  path for 6 : (0->3=1.00)(3->6=4.00)

>  java DijkstraSP graph2.txt 0
Graph: V=8 E=15
[0] neighbors=2: (0->2=0.26), (0->4=0.38)
[1] neighbors=1: (1->3=0.29)
[2] neighbors=1: (2->7=0.34)
[3] neighbors=1: (3->6=0.52)
[4] neighbors=2: (4->7=0.37), (4->5=0.35)
[5] neighbors=3: (5->1=0.32), (5->7=0.28), (5->4=0.35)
[6] neighbors=3: (6->4=0.93), (6->0=0.58), (6->2=0.40)
[7] neighbors=2: (7->3=0.39), (7->5=0.28)
Paths to source: 0
  path for 0 :
  path for 1 : (0->4=0.38)(4->5=0.35)(5->1=0.32)
  path for 2 : (0->2=0.26)
  path for 3 : (0->2=0.26)(2->7=0.34)(7->3=0.39)
  path for 4 : (0->4=0.38)
  path for 5 : (0->4=0.38)(4->5=0.35)
  path for 6 : (0->2=0.26)(2->7=0.34)(7->3=0.39)(3->6=0.52)
  path for 7 : (0->2=0.26)(2->7=0.34)
```