05. Dijkstra's Shortest Path Implementation

Setup

It is recommended you complete the associated **activity (https://canvas.wisc.edu/courses/254888/pages/05-dijkstra-shortest-path-activity?module_item_id=3939006)** prior to this.

Work Individually

You must develop the code for this assignment entirely on your own. If you encounter difficulties while working, you are encouraged to discuss general algorithms and debugging strategies with anyone you would like. If you feel that getting assistance will require someone else viewing your code, the course staff are the only people that you are allowed to share or show any part of your code with prior to the hard deadline for this assignment. You may not store or share your code in any way that other students can access. And you are not allowed to view or make use of any Dijkstra's Shortest Path Algorithm code that is not written entirely by you.

Provided Graph Data Structure

0. Start by downloading this **CS400Graph.java (http://pages.cs.wisc.edu/~cs400/CS400Graph.java)** starter implementation, along with the **GraphADT.java (http://pages.cs.wisc.edu/~cs400/GraphADT.java)**.
1. Review the provided Vertex and Edge inner classes to see how they can be used together to represent a weighted directed graph.
2. All vertices are stored in a Hashtable, using the data stored within a vertex as its key. For this you can use your own implementation or the one available via java.util. Review the insertVertex, removeVertex, insertEdge, and removeEdge method implementations to confirm how this graph data structure is represented. For further clarity, you can also review the provided definitions of containsVertex, containsEdge, getWeight, getEdgeCount, getVertexCount, and isEmpty.
3. None of these members (inner classes, fields, and methods) should be modified while implementing this assignment.

Path Inner Class

4. Instead of a predecessor table, we will keep track of all discovered paths by representing them as path objects (instances of the Path inner class). We'll create and extend these path objects while expanding our search out from the start node. This means that no additional path reconstruction step will be needed. Review the Path class's header and four fields as well as how they are initialized by the Path(Vertex) constructor.
5. Implement the extend constructor for this Path inner class. This constructor should copy the path passed to it, and extend it by a single edge passed as the second parameter. The final node of the new path will be the target vertex of this edge. This extension will require to update the path's overall costs too. Make sure that the dataSequence field in the newly constructed instance references a different list object than the source path did. This will allow you to change the list within any copy, without effecting the list within the source/original.
6. Notice that the compareTo method allows paths to be ordered by distance. And in the case of ties, lexicographically by their toString() representation.

dijkstrasShortestPath method

7. Implement the method named dijkstrasShortestPath. This method takes references to the data in any start vertex and any end vertex as input. From this input, the method generates and returns a Path object representing the shortest path from this start vertex to this end vertex. The next few steps contain some additional tips and suggestions regarding the implementation of this method.

- We have included an import statement for the java.util.PriorityQueue class. You should create an instance of this class within the dijkstrasShortestPath method to hold the discovered paths (including total cost distance, and previous vertex data) that have not yet been determined to be the shortest possible paths to their given end vertex. Sometimes the vertices within these paths are called the frontier.
- Make sure that you keep track of which vertices you have already found shortest paths to, so that you do not continue to explore longer paths to those nodes. You might want to consider using the java.util implementation of a Hashtable.
- If your search fails to find any path from the start to end vertices, then this method should throw a NoSuchElementException to indicate that this has happened.

Assignment Submission

Please submit your progress on this assignment early and often to gradescope.com (https://gradescope.com/) as a form of back up, and to get feedback from the automated grading tests. The one file that you should include with your submissions is: CS400Graph.java. Your most recent submission in gradescope will be marked "active" by default, but you can change this to an earlier on-time or less buggy submission any time prior to the hard deadline. Ensure that your final "active" submission for this assignment is clearly organized, consistently styled, well documented with comments, and includes the following file header information at the top of each source file:

```
// --== CS400 File Header Information ==--
// Name: <your full name>
// Email: <your @wisc.edu email address>
// Notes to Grader: <optional extra notes>
```

Grading Rubric

Criteria	Assignment	Points
Formative Gradescope Tests	Individual	18
Summative Gradescope Tests	Individual	12
Code Clarity (Manual)	Individual	5
Total Points Possible		**35**