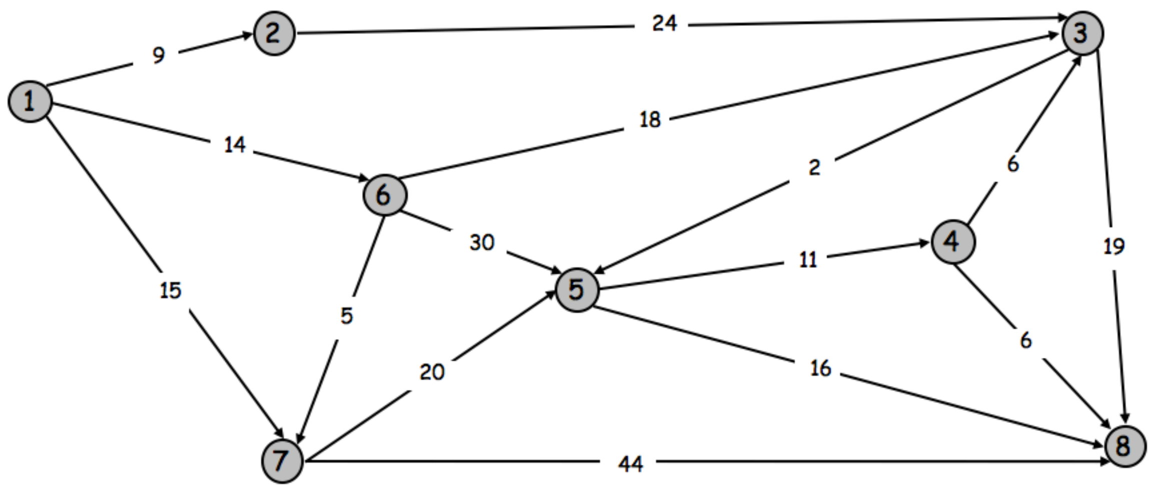
HW8 CSCD320

To turn in: please wrap up all your java source files into a zip file, then turn in the single zip file on the **EWU Canvas** by going to CSCD320-01 course page on Canvas, then clicking Assignments→HW8->submit. Please name your zip file with your last name, followed by the first initial of your first name, followed by hw8. For example, if you are John Smith, name you file as smithjhw8.zip

Specifics
Give an input weighted directed graph as shown in the diagram below, with the weight being labeled on each edge in the diagram.



1) Please implement **the Adjacency List** representation to construct the graph in the main memory of your computer.
2) Please implement the **Dijkstra's algorithm** to output all shortest paths from the vertex **1** to all others vertices, as we learned in class. You are allowed to use any Java **built-in** data structures. Please **do not use** third-party external java libraries.
3) Please invoke your method of the Dijkstra's algorithm implementation on the input graph above in a separate java file **Tester.java**.
4) Please print out explicitly the shortest paths that your program finds in the input graph on the **standard output** when running your program. Please follow the output format shown in the below. (The order of the different paths being displayed will NOT be graded. Namely, it is OK that your output could first **display** my Path3, then **display** my Path 2.)
Path 1): vertex 1 to vertex1, 1 → 1, length 0
Path 2) vertex 1 to vertex 2, 1 → 2, length 9
Path 3) vertex 1 to vertex 6, 1→ 6, length 14
Path 4) vertex 1 to vertex 7, 1→ 7, length 15
Path 5) vertex 1 to vertex 3, 1→6→3, length 32
……….
………
………

5) Please organize your source code so that I can compile **all** your source files located in **one** folder using command, **javac *.java**, and run your program using command on command line, **java Tester.**
6) **You can have your own design for any details that have NOT been specified in this document.**
7) **Please complete this homework independently because all logical ideas have been conveyed and confirmed in the classroom.**

Bonus (up to 10%)

You can use a **Priority Queue (or a structure equivalent to a Priority Queue, such as a TreeMap)** in your implementation as the Q container that initially stores all vertices, because a Priority Queue returns and removes the smallest object in it with a time-complexity of O(log n). You are allowed to use the java build-in PriorityQueue class to fulfill this requirement. If you attempted the Bonus points, please **print out a message explicitly** on the standard output to inform the grader of your efforts. Otherwise, you can lose the bonus points if you forget to display the message.