Dijkstra's Shortest Paths Algorithm (SSAD)

This assignment involves implementing an adjacency list representation of a weighted graph, and using it to apply Dijkstra's shortest paths algorithm (single-source, all destinations) to a weighted graph.

Design and implementation requirements

The program will read the specification of the problem from an input file whose name will be given on the command line. The input file will begin with two lines specifying the number of vertices in the graph, *V*, and the source vertex. Each of those values will be preceded by a label that ends with a colon character (`':'`). The next line will be blank. The remainder of the file will be *V* lines, each containing *V* integer values for the weights of the edges in the graph. Weights will be integers in the range 1 to 99. Missing edges will be designated by the value 0. Here's a sample:

```
Number of vertices: 11
Start vertex:       4

    0    0    0   10    0   94    0    0   73    0    0
    0    0    0   13    0    0    0    0    2    0    0
    0    0    0    0    0   43    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0
    0    0    0   87    0    0    0    0    0   68    0
    0    0    0    0    0    0    0    0    0    0    0
    0   97    0    0    0    0    0    0    0    0    0
    0    0    0    0    0   91    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0
    0   17    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0   55    0    0    0    0
```

Output will be written to a file whose name will be given on the command line. The output file consists of two sections, one displaying information about the given graph and the other displaying the results of running the SSAD algorithm.

The graph information display consists of two lines of labels (see sample), followed by *V* lines of output displaying the graph information in a format similar to the list of neighbors structure. For each vertex, there will be one row of data listing the vertex number, followed by whitespace and a pipe symbol, followed by a list of neighbor records. A neighbor record consists of a neighbor (vertex) number, followed by a colon character, followed by the weight of the corresponding edge. Each integer in a neighbor record is written in a field of 3 columns. The graph information display is followed by a blank line.

The SSAD results display begins with two lines of labels (see sample), followed by *V* lines showing the results for each possible destination vertex (including the source vertex itself). Each of these lines consists of three sections, the destination vertex number, the length of the shortest path found, and the actual path (a sequence of vertex numbers separated by colon characters). If a vertex is not reachable from the source vertex, display `"inf"` instead of the length.

Here's an output sample that corresponds to the input sample above:

```
Node  | Out-neighbors
-----------------------------------------------------------------------
    0      3: 10    5: 94    8: 73
    1      3: 13    8:  2
    2      5: 43
    3
    4      3: 87    9: 68
    5
    6      1: 97
    7      5: 91
    8
    9      1: 17
   10      6: 55

Start vertex is: 4

       Total
Dest | Weight | Path
-----------------------------------------------------------------------
   0      inf
   1       85     9   1
   2      inf
   3       87     3
   4        0
   5      inf
   6      inf
   7      inf
   8       87     9   1   8
   9       68     9
  10      inf
```

There are some explicit requirements, in addition to those on the *Programming Standards* page of the course website:

- You must implement a Java class for the weighted adjacency list representation; whether it is a formal generic is entirely up to you. Use of any other structure to represent the graph will result in a deduction of 50%.
- Your main class must be called `SSAD`, and that must implement a `main()` method.

We will invoke your program from the command line as follows:

```
java SSAD <input file name> <output file name>
```

Dijkstra's SSAD algorithm must be implemented as a separate function, not as a member of the adjacency list class. That function should take an initialized graph object as one of its parameters, and compute the path lengths and shortest paths, but it should not write any of the display to the output file. Writing output should be done by a different function that creates the necessary adjacency list object and calls the SSAD method.

Since you'll be implementing the complete program, the interfaces of the adjacency list generic and the SSAD function are up to you.

Testing

We will post a number of files specifying weighted graphs, and the corresponding SSAD solutions. In addition, we will post the following two tools:

`SSADGen.jar`	This is an executable jar file that will generate test data (an input file, and an annotated reference solution file). Run the file with the `-help` switch for instructions.
`LogComparator.jar`	This is an executable jar file that will compare an annotated reference solution file to the file generated by your solution, and write score information. Run this with no command-line parameters for instructions.

Since the `LogComparator` tool grades by comparing output, it is important to format your output exactly as specified above.

Evaluation

Your submission will be graded on CentOS 7, using a collection of several different graphs (created by the `SSADGen` tool) and the `LogComparator` tool. In addition, a TA will examine your graded submission to verify that you have implemented and used an adjacency list representation of the graph.

You should document your implementation in accordance with the *Programming Standards* page on the course website. It is possible that your implementation will be evaluated for documentation.

If you make more than one submission, the latest one will be graded. Submissions after the due date will be penalized 10% per day, with a maximum late penalty of 50%.

What to turn in and how

For this assignment, you must submit a zip file containing all the Java source code files for your implementation (i.e., `.java` files). Submit only the Java source files. Do not submit Java bytecode (class) files. If you use packages in your implementation (and that's good practice), your zip file must include the correct directory structure for those packages.

That's easy to verify by copying and unzipping the file you are planning to submit. After unzipping the file you plan to submit, it should be possible to compile your solution with the command

```
javac SSAD.java
```

Instructions, and the appropriate link, for submitting to the Curator are given in the *Student Guide* at the Curator website:

http://www.cs.vt.edu/curator/.

You will be allowed to submit your solution multiple times, in order to make corrections. The Curator will not be used to grade your submissions in real time, but you will have already done the grading using the posted code.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the pledge statement provided on the Programming Standards page in one of your submitted files.