

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 a file named "Graph.txt". 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 named "SSADResults.txt". 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.
- That class must include a member function that writes the display of the graph in the proper format.
- Your main class must be called `P5`, and that must implement a `main()` method.

Dijkstra's SSAD algorithm must be implemented as a separate function, not as a member of the adjacency list generic. 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. That 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. You may want to develop additional test cases yourself; it is perfectly acceptable to share such test cases via the Forum board. Since the Curator grades by comparing your output, it is important to format your output as specified above.

Evaluation

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 and design, as well as for correctness of results. If so, your submission that achieved the highest score will be evaluated by one of the TAs, who will assess a deduction (ideally zero) against your score from the Curator.

What to turn in and how

This assignment will be auto-graded using a test harness on the Curator system. The testing will be done under Windows (which should not matter at all) using Java version 1.8u25 or later.

Submit a single `.jar` file (not a zip archive) containing the Java source code for all of the classes for your solution to the Curator System. Submit nothing else. Your solution should not write anything to standard output.

Your submitted `.jar` file will be expanded in the test directory and compiled with the command `javac *.java`.

Your program will then be executed with the command `java P5`.

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.