2 Programming Project

This project involves implementing an adjacency list representation of a weighted graph, and using it to apply Dijkstra's shortest paths algorithm (single-source, all destinations(SSAD)) to a weighted directed graph. The program will read the specification of the problem from a given file named for instance "File.txt ". The input file will begin with two lines specifying the number of vertices in the graph, $G = (V, E)$, 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 a matrix (similar to the adjacency-matrix representation) of $|V|$ rows, 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 an example:

```
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 for instance "Result.txt ". The output file should consist of two sections, one displaying information about the given graph and the other displaying the results of running the Dijkstra's shortest paths SSAD algorithm.

The graph information display consists of a row of labels (see example), followed by $|V|$ rows of output displaying the graph information in a format similar to the adjacency-list representation of the graph. For each vertex, there will be one row of data listing the vertex number, 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. The graph information display is followed by a blank line.

The SSAD results display begins with two lines of labels (see example), one showing the source vertex and a blank line, followed by $|V|$ rows 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 (total weight) of the shortest path found, and the actual path (a sequence of vertex numbers separated by spaces). If a vertex is not reachable from the source vertex, you should display "`inf`" instead of the length.

Here's an output example that corresponds to the input example 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

Dest │ Total 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      │
```

2.1 Notes

There are some explicit requirements for the programming project:

- Code has to be written in C++, no other platform is accepted.

- You must implement a C++ class for the weighted adjacency list representation of a directed graph; whether it is a template or not is entirely up to you.

- Your main class must be called **digraph**.

- The data read from a file and specifying the graph can be stored in a dynamic one dimensional or two dimensional array as a private member of the class together with the source vertex.

- That class must include a member function that writes the display of the graph in the proper format indicated above.

- You have to write an extensive REPORT containing the answers to the 14 theoretical questions and explaining the programming project, and how to operate your program. For each input file specification, you have to draw the corresponding digraph in your report and HIGHLIGHT the shorted path found by the Dijkstra's SSAD algorithm for each destination vertex (when the path exists obviously).

- Dijkstra's SSAD algorithm must be implemented as a separate function, not as a member of the digraph class. That function should take an initialized graph object as one of its parameters (and the source vertex as the second), 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 digraph object and calls the SSAD method.

- Since you'll be implementing the complete program, the interfaces of the digraph class and the SSAD function are up to you.

2.2 Testing

We will include a number of files specifying weighted graphs, and the corresponding SSAD solutions. You may want to develop additional test cases yourself (there will be some bonus points for doing that). For marking purposes, it is important to format your output as specified above.