CS 330: Algorithms: Design & Practice
Lab#10: Dijkstra's Shortest Path Algorithm

This week we will implement Dijkstra's Shortest Path Algorithm (read, Cormen, Chapter 6). Here is a synopsis from your text:

Given: A graph G = <V, E> where V is a set of *n* vertices and E is an ordered pair of edges of the form $< X, Y, W_{xy} >$ such that *X*, *Y* are vertices in V and W_{xy} is the weight on the edge <X, Y> and $W_{xy} \geq 0$.

In addition to storing the graph itself, the algorithm requires the following structures:

shortest[v]: an array of length *n*, with one entry for each vertex. The value in *shortest[v]* is the weight of the shortest path known from a given starting vertex, *s* to *v*.

pred[v]: an array of length *n* such that *pred[v]* contains the vertex that is the known predecessor of *v* on the shortest path from *s* to *v*.

FRONTIER: A priority queue or an array.

Here is the algorithm:

```
Procedure DIJKSTRA(G, s)

Inputs: G as described above, and a starting vertex, s.

Results: For each non-source vertex v in V, shortest[v] is
the weight of the shortest path from s to v and pred[v] is
the vertex preceding v on some shortest path.

for each vertex v in V:
   shortest[v] = ∞
   pred[v] = null

shortest[s] = 0, pred[s] = null

for each vertex v in V:
   insert v in FRONTIER

while FRONTIER is not empty:
   u = remove from FRONTIER the vertex with shortest value
   for each vertex v adjacent to u:
      RELAX(u, v)
```

Where RELAX() is defined below:

```
Procedure RELAX(u, v)
Inputs: u, v are vertices in V such that there is an edge
between them
Result: Updates the values in shortest[v] and pred[v] if
possible

if shortest[u] + weight(u, v) < shortest[v]
   shortest[v] = shortest[u] + weight(u, v)
   pred[v] = u
```

Task#1: As you did in Lab#9, first implement the Graph ADT (instructions and design will be provided on Piazza). Implement and test similarly: for a given input vertex (for the file tinyEWD.txt – see below, output its neighbors) and corresponding weight. For example:

```
Enter a vertex: 7
7 has the following neighbors:
<7, 3, 0.39>, <7, 5, 0.28>

Enter a vertex: 5
5 has the following neighbors:
<5, 4, 0.35>, <5, 7, 0.28>, <5, 1, 0.32>
```

Task#2: Implement Dijkstra's algorithm (in Python, C/C++, or Java), exactly as shown above. Instructions and design will be provided on Piazza. Your program should input a graph from a data file. Two data files are provided:

Data File 1: Contains 8 vertices and 15 edges.
https://cs.brynmawr.edu/Courses/cs330/spring2020/tinyEWD.txt

Data File 2: Contains 250 vertices and over 2500 edges.
https://cs.brynmawr.edu/Courses/cs330/spring2020/mediumEWD.txt

Side Question: What is EWD??

The data files are in the following format:

```
8
15
4 5 0.35
5 4 0.35
4 7 0.37
5 7 0.28
7 5 0.28
…plus 10 more lines…
```

The first line of input file contains the number of vertices (8). The second line contains the number of edges (15). These are followed by edges, one per line. Each field separated by one or more spaces. Thus, the first edge listed in the data file above is an edge <4, 5, 0.35>. That is, a directed edge from vertex 4 to vertex 5 with a weight of 0.35. Vertices are numbered 0..n-1. When done, it should printout the shortest path from vertex 0 to a given destination vertex. The output of the program should be as shown below:

Enter the source vertex: 0
Enter the destination vertex: 6
There is a path from 0 to 6.
The shortest path has a cost 1.51. Here it is:

0 -> 2 0.26
2 -> 7 0.34
7 -> 3 0.39
3 -> 6 0.52

First, study the algorithm and decide on the overall design of the program. During development, test using Data File 1 (see Figure 1). Once done, try Data File 2.

What to hand in

1. A PDF file containing a printout of your programs along with the output from the two graphs. For the first dataset, use vertex 0 and 6 as source and destination. For the second data file, show the results for 0 to 4, and 0 to 1.

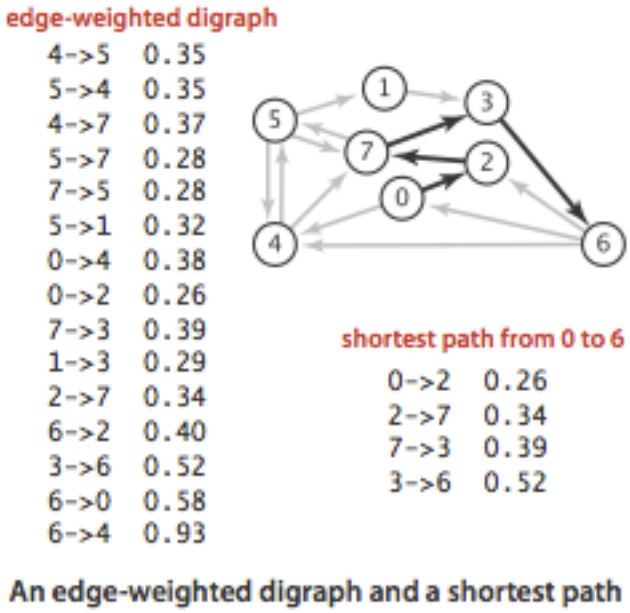

Figure 1: The graph of Data File 1 (tinyEWD.txt)