CS 337: Algorithms: Design & Practice

Lab#8: Dijkstra's Shortest Path Algorithm

In this lab we will implement Dijkstra's Shortest Path Algorithm (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 $W_{xy} \geq 0$. Assume vertices are numbered 0..n-1. The data files, described below, all have graphs of this form.

```
Procedure DIJKSTRA(G, s)

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


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.


Let:
    shortest[v]: an array of length n, with one entry for
each vertex.
    pred[v]: an array of length n. pred[v] contains the
vertex that is the predecessor of v on the shortest path
from s to v.
    Q: A min priority queue of nodes. The key is shortest


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 Q

while Q is not empty:
   u = remove from Q the vertex with smallest value of
shortest
  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
   update Q as needed for the new value of shortest[v]
```

If infinity or null is inconvenient in your implementation, feel free to use -1.   Doing so requires an some extra "if" statements but is otherwise equivalent.

Data

Three data files are provided:

https://cs.brynmawr.edu/cs337/Labs/Lab08Data/tinyEWD.txt
https://cs.brynmawr.edu/cs337/Labs/Lab08Data/mediumEWD.txt

https://cs.brynmawr.edu/cs337/Labs/Lab08Data/hugeEWD.txt

These data files are also available on UNIX in the directory /home/gtowell/Public/337/Lab08Data. In addition, there is a compressed tar file at /home/gtowell/Public/337/lab08data.tgz for ease of scp.

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 is 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. I strongly encourage you to do all of your development and debugging using the "tiny".

Task #0:

Implement Dijkstra's algorithm (in Python, C, Go, or Java). Your program should input a graph from a data file of the format described above. You may use any implementation of PriorityQueue; or write one from scratch. (You need not follow Cormen.) A poor priority queue implementation will make things slow on task#2.

Task #1

Write a program to read in a data file of the form described above into a graph ADT. After reading, the program should enter an interactive loop in which the user is asked for an origin and destination vertex. The program should, in response, printout the shortest path between these vertices. The shortest path should be computed using Dijkstra's algorithm as it is given above. For instance, the following is a sample interactive session:

```
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
Enter the source vertex: 0
Enter the destination vertex: 7
…
```

The program should NOT die as a result of user interaction. It should be robust to out of bounds node numbers (e.g. -1 or 3.1415) and input that is not a number at all (e.g., "This is a test"). In addition, you program should have a graceful way of exiting (ie tell the user what to enter to quit).

Here are some more results from a slightly different program incarnation and using the medium graph (printing the path backwards — as below — is OK).

```
Source Vertex (-1 to exit): 240
Destination Vertex: 23
The shortest path from 240 to 23 has cost 0.46
Path -- backwards
to 23 from 176
to 176 from 160
to 160 from 231
to 231 from 226
to 226 from 138
to 138 from 240
```

```
Source Vertex: 188
Destination Vertex: 4
The shortest path from 188 to 4 has cost 0.21
Path -- backwards
to 4 from 240
to 240 from 188
Source Vertex (-1 to exit): 4
Destination Vertex: 190
The shortest path from 4 to 190 has cost 0.45
Path -- backwards
to 190 from 220
to 220 from 1
to 1 from 107
to 107 from 69
to 69 from 128
to 128 from 4
Source Vertex (-1 to exit): -1
Goodbye!
```

Task#2:

Write a program to determine the following information using the huge graph

1. The cost of highest cost, shortest path to another node when starting from node 3310. That is, if you travel the most efficient route, what node that is reachable has the highest cost to get to.
2. The destination of the highest cost path.
3. The number of nodes that are reachable from node 3310

For example, asking the same question of the tiny graph, and starting from node 7, I get the following result

```
FROM 7: CAN GET TO 8 nodes
    MOST EXPENSIVE to get to is node 0 with a cost of 1.490000
```

What to hand in

1. A report describing the design of your program. This should describe all data structures and abstractions that you created. Where needed, use illustrations. Include in your report a description of your solution for task 2 and the time complexity (big O) of your task#2 solution.

2. Appendices containing:
   a. A printout of your program
   b. Output from three sample runs. For `tinyEWD.txt`, use vertex 0 and 6 as source and destination. For `mediumEWD.txt`, show results for 0 to 4, and 0 to 1.
   c. The output of your program for task 2 showing the answers you obtained.

Approximate Grading Rubric

Introduction	10
Dijkstra description	16
Description of all data structures and abstractions. Why?	26
Illustration of data structures, as appropriate	8
Task 2: Answers to questions	6
Task 2: Description of algorithm and algorithmic analysis	14
Appendix	
code (for all tasks)	10
sample runs to task 1	5
output from task 2	5
Total	100

Identify yourself using the names of two species of north american birds