CS29003 Algorithms Laboratory

Assignment No: 9

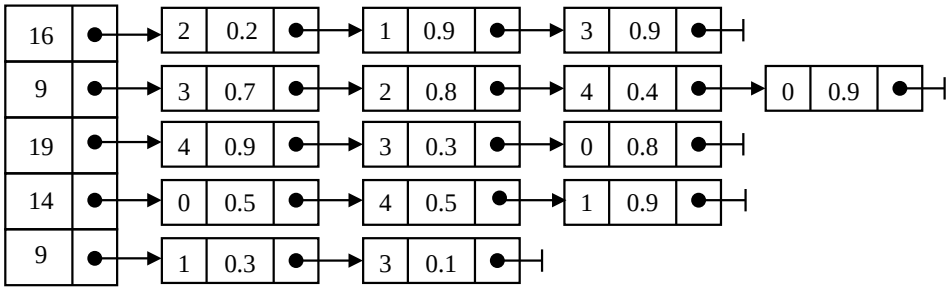
Last date of submission: 03–April–2018

Let $G = (V, E)$ be a directed graph with each edge e carrying a real-valued weight $ewt(e)$ in the interval $(0, 1]$, and with each vertex v carrying a positive integer weight $vwt(v)$. Let s and t be two distinct vertices in V. In this assignment, you compute best s,t-paths. As usual, we take $V = \{0, 1, 2, \ldots, n-1\}$. Renumbering the vertices, if necessary, we can always assume that $s = 0$ and $t = n-1$. We will keep this assumption throughout the assignment.

Part 1: Digraph construction

As in the previous assignment, you build a graph from user inputs. The user first enters $n = |V|$ and $m = |E|$. Subsequently, the endpoints of the m edges are entered by the user one by one. Assume that the user does not make an attempt to insert the same edge multiple times. Write a function **`readgraph(n,m)`** to this effect. You may use your constructor from Assignment 8 with relevant changes, keeping in mind that this time you are dealing with *directed* graphs. Use your own adjacency-list representation.

Now, the user enters the edge weights against your listing of the edges. Notice that your listing may be different from the listing given in the sample output. Finally, for $v = 0, 1, 2, \ldots, n-1$, the user enters the vertex weights. The digraph of the sample output may be represented as shown in the following figure.



Part 2: Dijkstra's single-source-shortest-path algorithm

Dijkstra's SSSP algorithm is meant for computing shortest distances from a designated source to all vertices in a graph. When the destination is specified too, the same algorithm seems to be the best algorithm. We only make an optimization (without improving the worst-case complexity though). The moment when the destination t is processed, that is, dequeued from the min-priority queue of unprocessed vertices, we return the current shortest s,t distance, and stop. Implement this optimized variant of Dijkstra's SSSP algorithm in a function **`Dijkstra(G,n,s,t)`**. The algorithm should also print a shortest s,t-path.

An efficient implementation of Dijkstra's algorithm requires a min-priority queue supporting **`deletemin`** and **`changepriority`**. Moreover, you need to locate arbitrary vertices in the priority queue, so you need a vertex-locator array. Inside all the heap and priority-queue functions, you need to update the locator array whenever an element in the queue changes position. Write your own set of functions for this priority queue.

This generic point-to-point shortest-path function can be used in a variety of applications. For example, you can run this function on the weighted graph constructed in Part 1 (ignoring the vertex weights).

Part 3: Application on most reliable network paths

Think of G as a communication network. The edge weights stand for the probabilities that the links are functional (have not failed). Let $v_0, v_1, v_2, \ldots, v_l$ be a (directed) path in G. The reliability of this path is defined as the product of all the edge weights appearing on the path, that is, by the formula $\prod_{i=0}^{l-1} ewt(v_i, v_{i+1})$. The task is to find a most reliable s,t path. The logarithm function (to any base > 1) is an increasing function. So maximizing $\prod_{i=0}^{l-1} ewt(v_i, v_{i+1})$ is equivalent to minimizing $\sum_{i=0}^{l-1} \big[-\log(ewt(v_i, v_{i+1}))\big]$. Fix a base

of the logarithm (like e or 10), and replace each edge weight in G by the negative of its logarithm to that base. Then, run **`Dijkstra`** on it.

Part 4: Applications with vertex weights

In this part, assume that the edges do not carry any weight. The cost of a (directed) path $v_0, v_1, v_2, \ldots, v_l$ is the sum of the weights of the vertices appearing on the path, as given by the formula $\sum_{i=0}^{l} vwt(v_i)$. In this part, we want to compute a shortest s,t-path in G with path costs (distances) defined in this way.

Create a new digraph H from G. H consists of $2n$ vertices (where $n = |V(G)|$). Each vertex v in G maps to two vertices v_{in} and v_{out} in H. Add the directed edge (v_{in}, v_{out}) in H with weight $vwt(v)$. Add the edges of G appropriately to H. Then, run **`Dijkstra`** on H with appropriate source and destination vertices. This call should print a shortest s,t-path in the original graph G (not a shortest path in H).

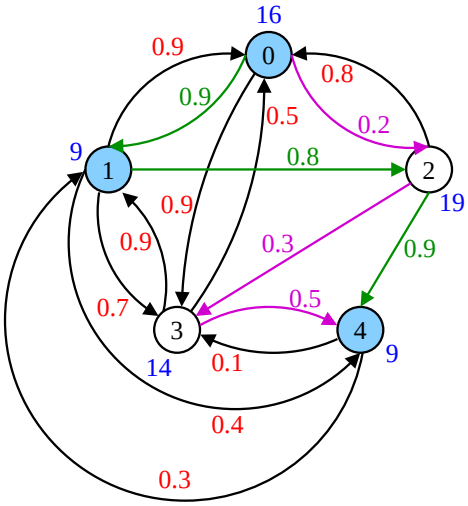
The *main*() function

- Call **`readgraph`** to construct G interactively from the user. Take $s = 0$ and $t = n - 1$.
- Run **`Dijkstra`** on G to compute the shortest s,t-distance and a shortest s,t-path.
- Now, let the edge costs be functioning probabilities of links in a communication network. Change the edge weights (and print the changes). Call **`Dijkstra`** to compute a most reliable s,t-path.
- Finally, create H from G as explained in Part 4. Run **`Dijkstra`** on H, and print a shortest s,t-path in G (and its cost).

Submit a single C/C++ source file. Do not use global/static variables.

Sample output

The sample output below corresponds to the weighted digraph of the following figure. The shortest path of Part 2 is shown by magenta links. The shortest path of Part 3 is shown by green links (the natural logarithm is used). Finally, the shortest path of Part 4 is shown by the highlighted vertices.




```
n = 5
m = 15
s = 0
t = 4

+++ Reading edges
 1 0   3 1   1 4   2 0   1 2   2 3   0 3   4 3   0 1   1 3
 0 2   4 1   3 4   3 0   2 4

+++ The generated graph
    0  ->  2,  1,  3
    1  ->  3,  2,  4,  0
    2  ->  4,  3,  0
    3  ->  0,  4,  1
    4  ->  1,  3

+++ Reading edge weights
    ewt( 0,2 ) = 0.2
    ewt( 0,1 ) = 0.9
    ewt( 0,3 ) = 0.9
    ewt( 1,3 ) = 0.7
    ewt( 1,2 ) = 0.8
    ewt( 1,4 ) = 0.4
    ewt( 1,0 ) = 0.9
    ewt( 2,4 ) = 0.9
    ewt( 2,3 ) = 0.3
    ewt( 2,0 ) = 0.8
    ewt( 3,0 ) = 0.5
    ewt( 3,4 ) = 0.5
    ewt( 3,1 ) = 0.9
    ewt( 4,1 ) = 0.3
    ewt( 4,3 ) = 0.1

+++ Reading vertex weights
    vwt(0) = 16
    vwt(1) = 9
    vwt(2) = 19
    vwt(3) = 14
    vwt(4) = 9

+++ Running Dijkstra on the original graph
--- Shortest (0,4) distance is 1.000000
--- Shortest (0,4) path: 0 - 2 - 3 - 4
```

```
+++ Changing the edge weights
    Edge weight (0,2) changes from 0.2 to 1.609438
    Edge weight (0,1) changes from 0.9 to 0.105361
    Edge weight (0,3) changes from 0.9 to 0.105361
    Edge weight (1,3) changes from 0.7 to 0.356675
    Edge weight (1,2) changes from 0.8 to 0.223144
    Edge weight (1,4) changes from 0.4 to 0.916291
    Edge weight (1,0) changes from 0.9 to 0.105361
    Edge weight (2,4) changes from 0.9 to 0.105361
    Edge weight (2,3) changes from 0.3 to 1.203973
    Edge weight (2,0) changes from 0.8 to 0.223144
    Edge weight (3,0) changes from 0.5 to 0.693147
    Edge weight (3,4) changes from 0.5 to 0.693147
    Edge weight (3,1) changes from 0.9 to 0.105361
    Edge weight (4,1) changes from 0.3 to 1.203973
    Edge weight (4,3) changes from 0.1 to 2.302585

+++ Running Dijkstra on the log-converted graph
--- Shortest (0,4) distance is 0.433865
--- Shortest (0,4) path: 0 - 1 - 2 - 4

+++ Converting vertex weights to edge weights
    0  ->  1
    1  ->  6,  2,  4
    2  ->  3
    3  ->  0,  8,  4,  6
    4  ->  5
    5  ->  0,  6,  8
    6  ->  7
    7  ->  2,  8,  0
    8  ->  9
    9  ->  6,  2

+++ Running Dijkstra on the vertex-weight graph
--- Shortest (0,4) distance is 34.000000
--- Shortest (0,4) path: 0 - 1 - 4
```