CMPE/CISC-365
Fall 2019
Week 1 Lab Assignment
Not to be handed in for grading

This week's lab assignment is very straightforward: create a functioning implementation of Dijkstra's Algorithm.

You may write your program in Java, Python, C or C++.

Your program must be able to read the definition of a graph from a text file, apply Dijkstra's Algorithm using Vertex 0 as the starting point, and (for the purpose of verification) report the number of the vertex that is furthest away from Vertex 0

I have provided you with several data files of varying sizes. The name of each file indicates the number of vertices in the graph defined by the contents of the file. To make your task easier, I have ensured that each of the graphs is connected.

The format of each file is as follows:

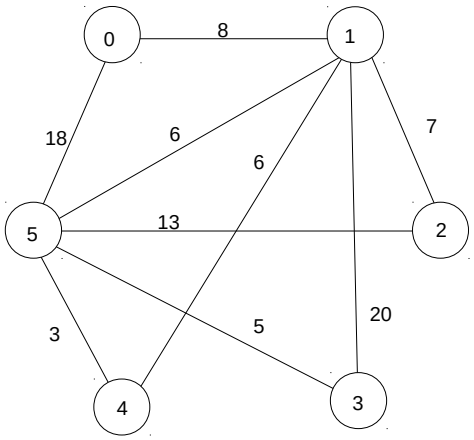
- the first line contains a single integer, identifying the number of vertices in the graph
- each following line contains a sequence of integers defining the next row of the adjacency matrix of the graph (with weights).

The smallest given graph has 6 vertices. The file looks like this:

```
6
0 8 0 0 0 18
8 0 7 20 6 6
0 7 0 0 0 13
0 20 0 0 0 5
0 6 0 0 0 3
18 6 13 5 3 0
```

A zero entry in the matrix indicates that there is no edge joining those two vertices.

Thus this graph looks like this:



The vertex that is furthest from Vertex 0 is Vertex 3, at a distance of 19 (The shortest route from 0 to 3 is 0 – 1 – 5 – 3. Your program is not required to show the route.)

Here is the pseudo-code for Dijkstra's Algorithm as presented in class. You will need to translate to a language of your choice and add code to read the contents of the graph file. You can prompt the user for the name of the graph file, or you can hard-code the file name into your program.

You can probably find full implementations of Dijkstra's Algorithm online. I strongly encourage you to write it yourself. Next week's lab (which you **will** be required to submit) will ask you to modify Dijkstra's Algorithm to solve a different problem. The modification will require a good understanding of the algorithm.

```
# Goal: Find least-weight paths from A to all other vertices
# Weight of edge (v,w) is given by W[v][w]

Cost[A] = 0
Reached[A] = True
for each other vertex x:
        Reached[x] = False
for each neighbour x of A:
        Estimate[x] = W[A][x]
        Candidate[x] = True
for all other vertices z:
        Estimate[z] = infinity
        Candidate[z] = False
while not finished:
        # find the best candidate
        best_candidate_estimate = infinity
        for each vertex x:
                if Candidate[x] == True and Estimate[x] < best_candidate_estimate:
                        v = x
                        best_candidate_estimate = Estimate[x]
        Cost[v] = Estimate[v]
        Reached[v] = True
        Candidate[v] = False
        for each vertex y:                  # update the neighbours of v
                if W[v][y] > 0  and  Reached[y] == False:
                        if Cost[v] + W[v][y] < Estimate[y]:
                                Estimate[y] = Cost[v] + W[v][y]
                                Candidate[y] = True
                                Predecessor[y] = v
```

You will probably have noticed that with the exception of the small data file that is offered for you to use when testing the correctness of your implementation, the sizes of the graphs increase by a factor of 2 (i.e each one has twice as many vertices as the one before.) Time permitting, conduct some empirical experiments to see if you can identify a pattern in the average running time of your implementation on these graphs. (This is optional!)