CSCM12 – Software concepts and efficiency April 2023
Tonicha Crook & Cécilia Pradic

Coursework 2

1. By submitting coursework, you state that you fully understand and are complying with the University's policy on Academic Integrity and Academic Misconduct. The policy can be found at `https://myuni.swansea.ac.uk/academic-life/academic-misconduct`. The consequences of committing academic misconduct can be extremely serious and may have a profound effect on your results and/or further progression. The penalties range from a written reprimand to cancellation of all of your marks and withdrawal from the University.

2. This coursework is coding and rather easier to account for the shorter turn-around – if you have engaged with the lab thus far, you should be able to make significant progress in the span of a single lab session.

3. There are 50 marks available. 5 bonus marks will be awarded for lab attendance.

4. Please follow **strictly** the submission instructions; marking will be done in part by an automated system, so please adhere to the format and the instructions.

Submission instructions (5 marks)

You will be using `https://csautograder.swansea.ac.uk` to submit your coursework. You should have access to that[1] and to a CSCM12 section with a "Coursework 2" project. There, you will be expected to upload two files, `Djikstra.java` and `MyPriorityQueue.java`. Then your submission will undergo a series of tests, the output of which you can check to an extent via the web interface. Your submission will be partially graded automatically (typically, the correctness of your methods will be checked); then we will also handgrade your submission after the deadline, and come up with your final grade for the coursework.

On canvas, you will be provided with a number of java files. There will be two types:

- auxiliary files `Pair.java`, `WeightedEdge.java`, `WeightedGraph.java`, `PriorityQueue.java` and `ALPriorityQueue.java` that you will use for your functions; you should not need to modify or submit them, but you will need them to compile your project on your machine. The submission platform will use them in addition to further files doing testing.

- template files `Dijkstra.java` and `MyPriorityQueue.java`; these are the files you should modify and submit to the automated platform. You should not remove methods from those files (even if you have not filled in with code) or change their types.

The only import satements you will be allowed in your submissions will be the following:

- `import java.util.ArrayList;`

- `import java.util.LinkedList;`

- `import java.util.function.Function;`

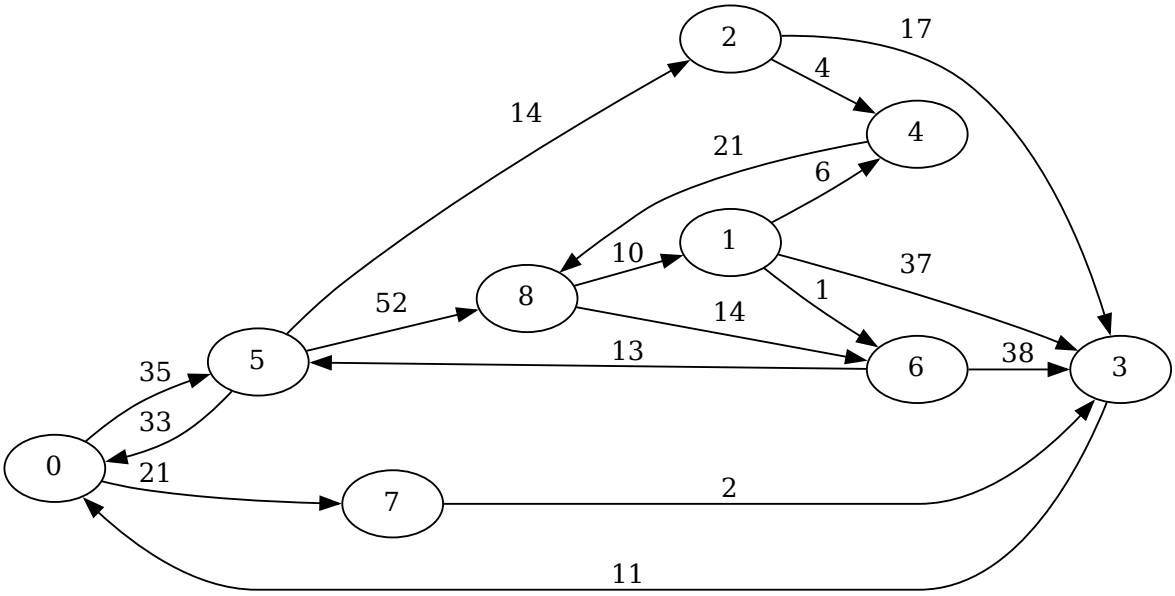
[1] If you have a 503 error when trying to load the website, try to clear your cookie/go to private mode navigation and log back in; it is a recurrent issue.

Furthermore, there should only be one import statement per line.

It is possible you get different counterexamples for the same submission, as the tests are randomly generated. The virtual machines on which the tests will be run are running Ubuntu 18.04 with Java 17 – so in particular it means that your code should compile with Java 17. For the most part, I don't think this should matter very much, although if you have a compilation error on the server that you don't manage to replicate on your machine, it might be worth invesigating whether this is the cause[2].

Main task (20 marks)

Recall that a weighted graph is a graph whose edge are labelled by numbers called *weights*. For the purpose of this exercise, we shall consider weights that are strictly positive integers only. Below you may find a representation of such a graph:



For now we shall interpret weights as distances; accordingly, define the weight of a path is the sum of its labels. For instance, the weight of the path $(0, 5, 8, 6)$ in the graph above is $35 + 52 + 14 = 101$. The distance from a vertex u to another vertex v will be the minimal weight of a path that goes from u to v (if there is no path, the distance is ∞). The goal will to code an algorithm answering the following question:

"Given a starting vertex v, compute the distances from v to every other vertices in the graph"

To do so, you will write the body of the method

```
static int[] distances(WeightedGraph g, int source)
```

[2]While I am saying this is not a huge deal, I have been bitten by this – one thing you can do with java 18 but not 17 apparently is using the `var` keyword to let java do some type-inference for you...

```
Djikstra(G, source)
    Q ← an empty priority queue
    Enqueue source with priority 0 in Q
    while Q is not empty do
        Dequeue the element v with minimal priority d from Q
        if v was not visited before then
            Set the distance between source and v to be d
            for all edges v --w--> v' do
                Enqueue v' with priority d + w in Q
            end
        end
    end

    return the computed distances
```

Figure 1: Pseudo-code for Dijkstra's algorithm

in a class `Dijkstra`. If the distance between `source` and `i` in the weighted graph `g` is $d < \infty$, the output array should contain the value `d` at cell `i`; otherwise, if $d = \infty$, the cell `i` should contain -1.

In particular, with an input `g` corresponding to the above graph and `source = 0`, the output should be

```
[0, 84, 49, 23, 53, 35, 85, 21, 74]
```

To implement this, you will use *Dijkstra's algorithm* that will allow to efficiently compute this value. It is presented briefly in Figure 1. Note that it uses a priority queue as an auxiliary datastructure; its efficiency therefore depends on the complexity of the operations for the specific implementation of priority queues that is used. The benefit of using a priority queue is that the output may be computed in a single traversal, so the queue only contains every edge of the input graph at most once during the whole runtime.

You will write your code in a provided `Dijkstra.java` file that should compile with the provided `Pair.java`, `WeightedEdge.java`, `WeightedGraph.java`, `PriorityQueue.java` and `ALPriorityQueue.java` java files. You should be able to compile them on the command-line by issuing the command

```
$ javac Pair.java WeightedEdge.java WeightedGraph.java PriorityQueue.java ALPriorityQueue.java Dijkstra.java
```

Optimizing (15 marks)

The priority queue implementation works, but is rather naive. We have seen in the lectures and labs how to do better using heaps. Implement your own class `MyPriorityQueue` in `MyPriorityQueue.java` based on a min-heap and make use of it `Dijkstra.distances` for extra marks.

Complexity analysis (5 marks)

What is the complexity of your implementation in terms of the number n of vertices and the number m of edges of the input graph? Include a brief argument (note that this depends on which implementation for priority queues you are using, take that into account! You should discuss *your* implementation).

A further Dijkstra-like problem (5 marks)

Now, let us try to solve a similar exercise to test your understanding of Dijkstra's algorithm. We will still be interested in computing distances from a node to all the others in a graph, but we will take a different interpretation of the weights now.

We now think temporally; every edge will take one minute to traverse, but they will not always be available. We take a label of k over an edge to mean that it will be available for traversal only after k minutes spent trying to traverse the graph.

In the graph above, it will mean that should be able to reach 2 from 0 in 37 minutes: first we can wait for 35 minutes for the edge $0 \to 5$ to be available, and arrive at 5 at the 36th minutes. Then the edge $0 \to 2$ is immediately available (it was already open at $t = 14$ minutes), so we may take it to arrive at two after 37 minutes.

Now your task will be to implement in the class `Dijkstra` a method

```
static int[] times(WeightedGraph g, int source)
```

that will compute the minimal amount of time needed to go from `source` to every other vertex v, and store that information in the cell indexed by v of the output (if v is unreachable, the cell should contain -1).

On the example graph above and with `source = 0`, the expected output is:

```
[0, 40, 37, 23, 38, 36, 40, 22, 39]
```