CSCM12 – Software concepts and efficiency March 31st 2025
Manjiri Joshi, Cécilia Pradic & Deshan Sumanathilaka

# Lab 7: graphs

For the lab this week, your task will be to complete the file `GraphsExercises.java`. Use the main function to test your functions and show your exampes if you want to sign off.

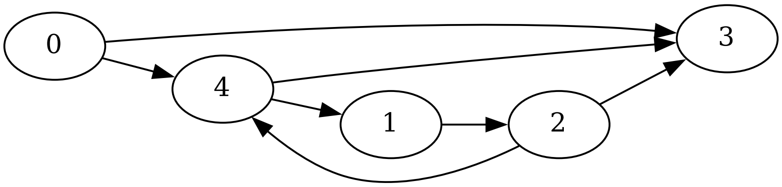
1. **The two graph representations** For this exercise, we shall consider unlabelled directed graphs whose vertices are identified with integers. Recall that there are two graph representations that are interesting:

   - Using arrays of adjacency lists
   - Using an adjacency matrix

   For the latter, we will simply use elements of type `int[][]`. For the former, we define a class `Graph`. It contains two attributes:

   - `adjSucc`, whose $i$th cell should contain all the vertices $j$ such that $(i, j)$ is an edge of the graph
   - `adjPred`, whose $i$th cell should contain all the vertices $j$ such that $(j, i)$ is an edge of the graph.

   (a) Define an adjacency matrix that correspond to the following graph



   (b) Write the code for the constructor

   `public Graph(boolean[][] adjM)`

   that converts from the adjacency matrix representation to an adjacency list representation.

   (c) Double-check that your function works using the method to print out graphs provided to you. For instance, if you call `g.toDotFile("ex")` in your `main`, you should obtain a file `ex.dot` in your folder. Then you can

```
FloydWarshall($M$)
    $D \leftarrow$ a copy of $M$
    $n \leftarrow$ dimension of $M$
    for $k$ from $0$ to $n-1$ do
        for $i$ from $0$ to $n-1$ do
            for $j$ from $0$ to $n-1$ do
                $D[i][j] \leftarrow \min(D[i][j], D[i][k] + D[k][j])$
            end
        end
    end
    return $D$
```

Figure 1: The Floyd-Warshall algorithm.

copy-paste the content to `https://viz-js.com`, you will have a picture of your graph; allternatively if you have graphviz[1] installed, issuing `dot -Tpng ex.dot > ex.png` in the terminel should generate a graphical representation in `ex.png`.

(d) Write the code for the method

```
public boolean[][] toMatrix()
```

that converts a `Graph` to its adjacency matrix representation

2. **Computing distances in weighted graphs** Given the adjacency matrix of a weighted graph, where the cell $(i, j)$ contains a positive integer or $\infty$ (that morally corresponds to having no edges), the *Floyd-Warshall* algorithm depicted in Figure 1 outputs another matrix that gives all distances. The key insight that allows to check that it does its job correctly is that, after the outer loop has been iterated $k$ times, then the cell $D[i][j]$ contains the minimal length of a path that goes from $i$ to $j$ and may use intermediate vertices in the set $\{0, \ldots, k-1\}$.

(a) What is the time complexity of the Floyd-Warshall algorithm (in function of the number of vertices in the input graph)?

(b) Implement the function

```
static public int[][] allDistances(int[][] graph)
```

in the class `GraphsExercises` using the Floyd-Warshall algorithm. Since there is no $\infty$ values in `int` in java, you may use for instance $-1$ to represent that instead.

(c) **Challenge** Implement a function

[1] `https://graphviz.org/`; hopefully it is installed by default on the linux machines

```
static public LinkedList<Integer>[][] allShortestPaths(int[][] graph)
```

that, instead f just giving the distances, outputs in each cell $(i, j)$ a path of minimal length that goes from $i$ to $j$.

3. **Graph traversals**

   (a) Implement the method

   ```
   public LinkedList<Integer> toListDFS(int i)
   ```

   of `Graph` that performs a depth-first search and enumerate all the vertices encountered in order. You may use either recursion or an imperative implementation using a stack, up to you.

   (b) Implement the method

   ```
   public int[] allDistancesFrom(int source)
   ```

   that outputs an array `A` such that `A[target]` contains the distance from `source` to `target` in the graph, assuming that the distance between two vertices is 1 if there is an edge, and $\infty$ otherwise (as in the previous question, use a dummy value like `-1` if there is no path at all). Hint: use a variation of a BFS - if that helps, a method performing a BFS is provided to you. Looking up the documentation of `LinkedList`, and in particular the `pollLast` method, might be useful.

   (c) **Challenge:** Adapt the previous method to obtain minimal paths.

   (d) **Challenge:** Write a class `BFSIterator` extending `Iterator<Integer>` and contains a constructor

   ```
   public BFSIterator(Graph g, int start)
   ```

   that allows to enumerate all the vertices of the graph in a breadth-first manner.