German University in Cairo
Department of Computer Science
Dr. Youmna Hassan

Concepts of Programming Languages, Spring Term 2025
Project 2: Railway Network

Due: 19th May 2025

1. Project Description

In this project, you are going to implement a railway network system. The network consists of stations connected by railroads, and traversing any railroad amounts to incurring some cost associated with this railroad. The task is to represent the network as a graph, where stations are represented as nodes, and the edges between any two nodes are **directed and weighted** with the traversal costs. The graph will be modelled as an adjacency matrix, whose row and column indices act as station identifiers.

Key Concepts:

- Graph Representation: The graph will be represented as a 2D adjacency matrix of integer numbers or `[[Int]]` in Haskell. **Rows and columns are indexed starting from 1**. Each cell in the adjacency matrix holds the weight of the edge between the stations denoted by the row and column indices; in particular, from the node denoted by the row index to the node denoted by the column index. Edge weights are integers in the closed range $[0, 9]$. If a cell holds the number 0, this means that there is no edge between the two stations referenced by the row and column indices.
- Shortest Path Algorithm: You will implement Dijkstra's algorithm to calculate the shortest path between two stations. This algorithm is well-known for finding the shortest path in a directed, weighted graph with positive weights. **You do not need to have any prior knowledge of the algorithm as its operation will be explained in this document**[1]**.**

2. Implementation Description.

Your implementation must contain the following type definitions.

```
data PriorityQueue = PQ PQNode PriorityQueue | Empty deriving (Show, Eq)
data PQNode = Node Int Int deriving (Show, Eq)
```

These will be used to compute the cost of the shortest path between any two stations.

a) `PriorityQueue` is a recursive data structure representing a priority queue or a sequence of elements given in ascending order according to the values of their keys.

b) `PQNode` is a representation of a graph node using the station id and a key, as enqueued in an instance of the `PriorityQueue`. The key of a priority queue node represents the cost of reaching it from some source node.

[1] You can, however, if you would like to, find more information on this algorithm in section 24.3 in this link.

Your implementation must also contain the following functions. You are not allowed to change the given type definitions of the functions, but you can add any other helper functions you need and you can use built-in functions. For the functions `railsNetwork` and `pullLever`, you should use the two functions provided in the accompanying starter file `project_2_starter_file.hs` on the cms for generating random numbers and pretty-printing the adjacency matrix. This file also has the public test cases and instructions for running them.

a) `railsNetwork :: Int -> Int -> [[Int]]`

- Description: This function generates a random railway network, represented as an adjacency matrix (a list of lists of integers).
- Input:
  1. The number of stations.
  2. The seed for the random number generator.
- Output: A 2D list (adjacency matrix) representing the graph. The matrix will have n rows and n columns (where n is the number of stations).
  Note: The main diagonal should always contain zeroes as no station should have an edge with a positive weight to itself. In other words, there are no self-loops in the graph.

Example:

```
> railsNetwork 5 12345

    [[0,1,0,9,5],
     [3,0,7,9,8],
     [9,6,0,0,7],
     [6,7,1,0,6],
     [9,5,3,2,0]]
```

Here, the number 3 in the cell at row 2 and column 1 means that there is a cost of 3 to travel from station 2 to station 1. The output of this example is visualized as the graph structure in the file `railsNetwork 5 12345.svg` on the cms.

b) `pullLever :: Int -> Int -> Int -> [[Int]] -> [[Int]]`

- Description: This function modifies the weight of an edge between two specific stations while keeping all other edges in the railway network unchanged.
- Input:
  1. The seed for the random number generator.
  2. The first station identifier.
  3. The second station identifier.
  4. The current adjacency matrix representing the railway network.
- Output: A new adjacency matrix with the modified weight for the edge between the two specified stations.

Examples:

```
> pullLever 123 3 4 (railsNetwork 5 12345)

    [[0,1,0,9,5],
     [3,0,7,9,8],
     [9,6,0,5,7],
     [6,7,1,0,6],
     [9,5,3,2,0]]

> pullLever 123 3 3 (railsNetwork 5 12345)

    [[0,1,0,9,5],
     [3,0,7,9,8],
     [9,6,0,0,7],
     [6,7,1,0,6],
     [9,5,3,2,0]]
```

`pullLever` should **not** be invoked while `computeShortestPathCost` is running.

c) `initializeSource :: Int -> [[Int]] -> PriorityQueue`

- Description: The function outputs a priority queue where each station is represented as an instance of `PQNode` with a key value initialized to a large number (9999), which represents the infinite cost to reach those stations initially. Only the source station's key is updated to 0, as the cost of reaching the source from itself is zero.
  To ensure that the priority queue maintains the correct order, the nodes should be sorted by their keys after updating the key of the source node.
- Input:
  1. The source station identifier.
  2. The adjacency matrix representing the railway network.
- Output: The function outputs a priority queue where each station is represented as an instance of PQNode with a key value initialized to a large number (9999), which represents the infinite cost to reach those stations initially. The source station's key is updated to 0, as the cost of reaching the source from itself is zero.

Example:

```
> initializeSource 2 (railsNetwork 5 12345)

    PQ (Node 2 0) (PQ (Node 1 9999) (PQ (Node 3 9999) (PQ (Node 4 9999)
    (PQ (Node 5 9999) Empty))))
```

d) `computeShortestPathCost :: Int -> [[Int]] -> PriorityQueue -> Int`

- Description: This function calculates the cost of the shortest path from the source station to a specified destination station using Dijkstra's algorithm. It takes in the following parameters:

- Input:
  1. The destination station identifier.
  2. The adjacency matrix representing the railway network.
  3. The priority queue returned by initializeSource which contains the starting station's updated key value of 0 and all other stations with an initial key value of 9999
- Output: The function then processes the priority queue to explore the shortest paths. It repeatedly dequeues the station associated with the smallest key, which should always be the first node in the priority queue, from the priority queue and updates the keys of its neighbors based on the edge weights retrieved from the adjacency matrix. A station neighbors another if it is adjacent to it or, in other words, if the weight of their connecting edge is in the range $[1, 9]$. The current key of a neighboring node should be updated only if the sum of **the key of the node which has just been dequeued** and **the weight of the connecting edge from the node which has just been dequeued to this same neighboring node** is strictly less than its current key. Once the destination station is dequeued from the priority queue, the function returns its key.

  Note that this function will modify the priority queue during the computation process by adjusting the keys of the nodes as shorter paths are discovered. The result returned by this function is the cost of traversing the shortest path from the source station to the destination station.

Example:

```
> computeShortestPathCost 1 (railsNetwork 5 12345)
(PQ (Node 2 0) (PQ (Node 1 9999) (PQ (Node 3 9999) (PQ (Node 4 9999)
(PQ (Node 5 9999) Empty)))))
    3
> computeShortestPathCost 1 (railsNetwork 5 12345)
(initializeSource 5 (railsNetwork 5 12345))
    8
```

3. **Teams.** You are allowed to work in teams of four members. You must stick to the same team you worked with in Project 1. IDs for the submitted teams are posted on the CMS.

4. **Deliverables.** You should submit a single `.hs` file named with your team ID containing your implementation. The submission link will be posted on the CMS prior to the submission.