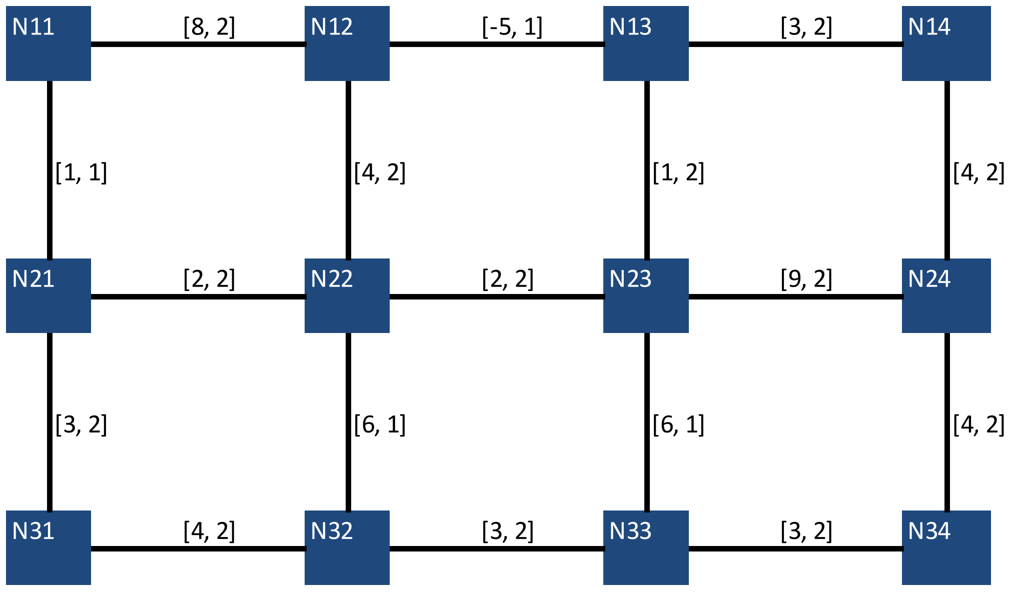
Exercise: Object Oriented model for optimized routing

Input

Consider the following "graph", consisting of 12 nodes and 17 edges.



[cost, capacity]

Each node has a name, indicated in white. Each edge has a cost and a capacity associated with it. The edges are bidirectional, that means they can be used in both directions. Each time an edge occurs on a path, we assume that 1 unit of the capacity of that edge is consumed. When the remaining capacity for an edge becomes 0, it can no longer be used to extend that same path. For example:

- The path passing following nodes in order, "N11, N12, N13, N14", has a total cost of (8-5+3) = 6 and consumes 1 capacity unit on each of these edges. This means that the edges between (N11;N12) and (N13;N14) have 1 capacity remaining, but the capacity of (N12;N13) is fully used.
- The path passing the nodes, "N11, N21, N22" has a total cost of (1+2) = 3. We could extend the path by looping back to node N21, because the edge (N21;N22) has 2 capacity available. The extended path would now pass the nodes, "N11, N21, N22, N21", for a total cost of (1+2+2) = 5. Extending that path once more towards node N11 is not allowed, because it would also require 2 capacity units on the edge between (N11;N21), which only has 1 capacity available. Note that this implies looping paths are possible, as long as the capacity constraint allows this.

Tasks

Task 1: Design and implement an object oriented data model in which the “graph” above can be represented. Consider the different responsibilities you want to assign to each class. (timing ca. 15-30 minutes)

(Tip 1: As a starting point, consider the following classes: Graph, Node and Edge.)

Task 2: Design and implement a shortest path algorithm that can find “the shortest route” (i.e. the path through the graph with the lowest total cost) between any given two nodes. There are several well-known algorithms that solve the shortest path problem (Dijkstra, Bellman-Ford, Floyd-Warshall and many others). You are allowed to use these algorithms as inspiration for your own implementation. We recommend that you take some time to read up on the different options and their limitations. Making the right choice of algorithm, ensures the amount of changes you’ll need to make it work on our example are minimal. (timing ca. 1-2 hours)

(Tip 2.1: Consider extending your data model with classes: ShortestPathAlgorithm and/or Path.)
(Tip 2.2: Don’t overlook the negative weighted edge between N12 and N13!)
(Tip 2.3: Make some adaptations to the standard algorithm to support the capacities.)

Task 3: Implement a test class that contains the above example graph and then determines the “shortest path” between node N11 and N34 on that graph. Write out the shortest path as a sequence of nodes visited along the path. (timing ca. 15-30 minutes)

Requirements

All code should be written in Java. You can use any IDE of your choice. Please write sufficient Javadoc to clarify your code. You will be assessed on your programming style, so try to write your code as close as possible to “production quality”. You may use the internet to look up information, but don’t just copy/paste any code or packages that you find online, we will notice.

Submitting your code

After receiving these instructions, you have **three hours** of time to complete this exercise. When you are finished, you can zip and e-mail your code to the e-mail address below. If you are using IntelliJ, please zip the entire IntelliJ folder. If you are using another IDE, zipping only the source code is sufficient.

Contact information

jonas.vannieuwenberg@comsof.com

09 275 31 00