# Programming assignment 4

## Getting started

All your submissions should be implemented in Java. If you do not have a Java development environment, we recommend that you either use the Visual Studio Code Java extensions or Jetbrains IntelliJ. If you use IntelliJ, do not forget to register as a student, so you get access to the full versions. If you run Linux, macOS, or WSL, and want more control over your JDK, we recommend SDKMAN!

Note that you must solve all of the programming problems in Java. You are allowed to use Python (or any other suitable tool) to analyse the results, plot graphs, and so on. If you have discussed an alternative language with Morgan, it's fine to ignore the Java requirement and use, e.g., C++ instead.

## Problems

### Problem 1

Implement the undirected and directed graph ADT using an adjacency list. You should use two different classes, one for either (but it might make sense to use a common base class). Your graph should support at least the following operations: #edges, #vertices, add edge, remove edge, and degree. You should provide iterators for vertices, edges, and the adjacency of a provided vertex.

Provide two methods to add edges, one that allows you to set the edge weight and one that sets it to 1.0 as default.

### Problem 2

Implement depth- and breadth-first search for the directed and undirected graph classes. You should at least support methods to check whether the source vertex is connected to a specified vertex in the graph and to iterate over the path between the source and a specified vertex.

You should use the same implementation for directed and undirected graphs. You can, e.g., use a base class for the two graph/edge types or define interfaces.

## Problem 3

Implement Kruskal's algorithm to compute minimal spanning trees. If the graph is not connected, you should compute a forest of minimal spanning trees. You can assume that there are no negative edge weights in the graph. Reuse your implementations of union-find from PA1 and your heap from PA3. Note that you need to change it to a minheap. You might also need to make additional changes to your implementations.

## Problem 4

Implement Dijkstra's and Bellman-Ford's algorithms to compute the single-source shortest path in a graph. Reuse your heap from PA3/the previous assignment. Note that you might need to change it/add additional methods. You are allowed to assume the required conditions for each algortihm. Compare the two for a few random graphs. Which performs best?

## Problem 5

Given a set of courses and prerequisites, implement an algorithm that gives an order to take the courses in such that all prerequisites are fulfilled. You can assume that the courses are given in the format `1DV002;1DV001`, where `1DV001` is the prerequisite for `1DV002`. Each row contains a pair of courses. You can assume that the resulting graph is acyclic.

Use the provided data.txt when you are testing your algorithm

## Problem 6 (optional, for higher grade)

Implement a method that, given a connected, undirected graph and an edge, determines whether the edge is a "bridge" or not. A "bridge" edge is an edge that disconnects the graph, i.e., if it is removed, the graph is no longer connected.

## Submission guidelines

Submit your solutions as a single zip-file via Moodle no later than 17:00 on November 10, 2023 (cutoff 08:00 November 13). This is a group assignment that can be done in groups of one or two students. Your submission should contain well-structured and organized Java code for the problems with a README.txt (or .md) file that describes how to compile and run the Java programs and a report in PDF format that describes your experiment and findings from Problem 4. It is sufficient if one person in the group submits the work to Moodle, but make sure that the submissions contains all your names.