ALGORITHMICS - Bachelor of Software Engineering, 2025

LAB GUIDE. SESSION 6

GOALS:

- **Backtracking algorithms**

1. Working with graphs

Many real problems can be modeled through a graph; in this case, we will focus on a graph with the following characteristics:

1) It is a **complete directed graph**, that is, there is an edge between any pair of nodes.
2) The existence of loops (edges with the same origin and destination) is irrelevant in this case, given the subsequent assumptions.
3) Between any two different nodes, there is a probability **p1** that the edge has a positive weight and a probability **p2 (p2=1-p1)** that it has a negative weight. In the following: **p1 = p2 = 0.5**.
4) If the weight of an edge is positive, it has an integer value in the interval **[minWeight ... maxWeight]** and if it has a negative weight, it is an integer value in the interval **[-maxWeight ... -minWeight]**. In the following: **minWeight = 10**; **maxWeight = 99** that is, we will work with **[10 ... 99]** or **[-99 ... -10]**
5) The problem we are asked to solve is to find in such a graph a single path that has a predetermined **objective cost**. For us, the objective cost will be 0, so we can call that path sought **NullPath**.
6) A tolerance is allowed for the path searched. In the following, it will be less than or equal (in absolute value) to the maximum weight of an edge (tolerance<=99), that is, **any path with a cost in the range [-99 ... 99] will be accepted**.
7) That **NullPath** that must be calculated must have as origin and destination nodes two predetermined nodes of the graph, it must be **simple** (no node of the graph can appear more than once) and **all the nodes** of the graph must participate in it. In what follows, we will assume that there was previously a reorganization of the nodes of the graph such that we will always be interested in the origin being the first node of the node vector (**origin=0**) and the destination being the last node of the node vector (**target=n-1**).

YOU ARE REQUESTED TO:

PART A:

A class **NullPath.java** (expecting as a parameter **n**). Initially, this class will randomly generate (for **n** nodes) a weight matrix according to the conditions stated above, then it will calculate and write the first **NullPath** found.

PART B:

A class **NullPathTimes.java**. It will grow in graph size like this: (**n = 20, 25, 30, 35**, …, until you lose patience).

For each **n**, 100 random weight matrices are generated and for each of them the time will be calculated and accumulated, so that at the end the average time will be calculated and written (**t_accumulated/100**).

The reason for doing this number of cases (100) is because the time varies significantly depending on the random weight matrix generated. Obviously, if we took a larger value instead of 100, the average time obtained would be more reliable; but having said that, 100 is considered sufficient.

In this session, the time will be taken without the Java optimizer (with **-Xint**).

PART C:

Finally, fill in a table showing the average times obtained (in milliseconds) for the sizes indicated in the previous step.

What time complexity do you assign to the algorithm that calculates the **NullPath** between the origin and the destination?

2. Work to be done

- An `algstudent.s6` **package** in your course project. The content of the package should be the Java files used and created during this session.
- A `session6.pdf` **document** using the course template (the document should be included in the same package as the code files). You should create one activity each time you find a "YOU ARE REQUESTED TO" instruction.

Deadline: The delivery of this lab will be carried out, in time and form, according to the instructions given by the lab teacher.