Project 2

The subject of this project is to create an implementation of the Nagamochi-Ibaraki algorithm (see in the lecture notes, and also below) for computing the connectivity of an undirected graph, and experiment with it.

Just in case you may have missed the class, here is a brief description of the algorithm. We compute the function $\lambda(G)$, the edge-connectivity of the graph G, recursively.

Function $\lambda(G)$

1. *Base case:* if $n = 2$ (number of nodes), then count how many edges connect the two nodes, and return this number as the value of $\lambda(G)$.
2. If $n > 2$, then do:
   - Run the Maximum Adjacency (MA) ordering algorithm on the current graph (see the lecture note); let the obtained ordering be $v_1, v_2, \ldots, v_n$.
   - Set $x = v_{n-1}$, $y = v_n$.
   - Set $a = d(y)$ (the degree of y)
   - Create the graph G_{xy}, by merging nodes x and y.
     (In this operation the possibly arising self-loops are removed, but parallel edges are kept.)
   - Compute $\lambda(G_{xy})$, by recursively calling the function for G_{xy}, which is a smaller graph (has $n - 1$ nodes).
   - Return $\lambda(G) = \min\{a, b\}$.

Tasks of the Project:

1. Explain how your implementation of the algorithm works. Provide pseudo code for the description, with sufficient comments to make it readable and understandable by a human.

2. Write a computer program that implements the algorithm. You may use any programming language under any operating system, this is entirely of your choice.

3. Run the program on randomly generated examples. Let the number of nodes be fixed at $n = 20$, while the number m of edges will vary between 19 and 190, in steps of 3. For any such value of m, the program creates 5 graphs with $n = 20$ nodes and m edges. The actual edges are selected randomly. Parallel edges and self-loops are *not* allowed in the graph generation. Note, however, that the Nagamochi-Ibaraki algorithm allows parallel edges in its internal working, as they may arise due to the merging of nodes.

4. Experiment with your random graph examples to find an experimental connection between the number of edges in the graph and its connectivity $\lambda(G)$. (If the graph happens to be disconnected, then take $\lambda(G) = 0$.) Show the found relationship graphically in a diagram, exhibiting $\lambda(G)$ as a function of m, while keeping $n = 20$ fixed. Since the edges are selected randomly, therefore, we reduce the effect of randomness in the following way: run 5 independent experiments for every value of m, one with each generated example for this m, and average out the results. (This is why 5 graphs were created for every m.)

5. For every connectivity value $\lambda = \lambda(G)$ that occurred in the experiments, record the largest and smallest number of edges with which this λ value occurred. Let us call their difference the *spread* of λ, and let us denote it by $s(\lambda)$. Show in a diagram how $s(\lambda)$ depends on λ, based on your experiments.

6. Give a short explanation why the functions found in items 4 and 5 look the way they look. In other words, try to argue that they indeed show the behavior that can be intuitively expected, so your program likely works correctly. Note that it is part of your task to find out what behavior can be expected.

7. Also include a section in the project document that is often referred to in a software package as "ReadMe file." The ReadMe file (or section) provides instructions on how to run the program. Even though in the

default case we do not plan to actually run the program, we should be able to do it, if needed. For example, if something does not seem right, and we want to double check whether the program indeed works correctly.

Important note: If there is anything that is not specified in this project description, that automatically means it is left to your choice.

Submission guidelines

Describe everything, including algorithms, program, sources, results and figures neatly and clearly in a study. Include everything in a *single document* that can be read as a report. It should have a *professional appearance,* scanned handwriting, hand-drawn figures etc. are not acceptable! Note that points can be lost even if the content is completely correct, but the appearance is not professional.

The preferred file type is pdf. Do not submit executable code, but include the source code as an Appendix in the document. The project report will be read as a document and not run as a program (but there are two exceptions, see them under Evaluation).

Submit the document through eLearning. Do not send it via e-mail!

Notes:

- The work should be fully individual and original. Any form of cheating is a serious violation of University policies and can lead to serious consequences.
- It may be helpful to think about the whole project presentation that your task is not only to solve a technical problem, but you also have to "sell" the results. Try to look at your work from the viewpoint of a potential customer, to whom you want to sell such a software product. How convincing would your presentation look for a customer?

Evaluation

The evaluation will focus on how well each of the specific tasks have been carried out, and how professional the whole work looks. Even though the

submission typically will not be run, only read as a document, there are two exceptions. You will be asked to demonstrate on a computer how your program actually works, if any of the following cases occur:

1. You do not agree with the grade and want to improve it. In this case the demonstration should show that your work is actually better than the received grade.

2. There is suspicion that the work is not original or not individually done or the results were not produced by your own correctly running program. In this case a demonstration is required to clarify the situation and to receive any score.