Assignments

The assignments give the credits consisting of the `LAB1`-instalment in LADOK.

The course has two programming assignments, each consisting of two parts, *a* and *b*. These will be referred to as 1a, 1b, 2a, and 2b. Only 1a and 2a are mandatory, 1b and 2b are for higher marks. Since 1b and 2b are for higher marks, no instructions in how to solve them are given but it is fairly easy to understand how to complete them if 1a and 2a have been solved.

The assignments 1a and 1b are done in Java and deal with cryptography. The assignments 2a and 2b are done in *Octave*, a *MATLAB*-lookalike, in which we will work with matrix representations of graphs. The assignment 2a entails implementation of an algorithm for finding a minimal spanning tree in a weighted graph and in 2b we will concern ourselves with Dijkstra's algorithm for finding the shortest path between two vertices in a weighted graph. You need to work in pairs and show your work together.

Grading. To form a grade we will introduce special *points*. Each assignment gives 1 point. If all assignments are solved, this gives in total 4 points. A further point is given if *all* the solutions are presented within the time frame of the course, resulting in the highest number of points to be 5 which will give an A, the highest mark. However, 1 point will be deducted if *any* of the solutions are presented outside the time frame of the course. (This is in accordance with the exam goals of the training since one skill of an engineer is to be able to present solutions to given problems within a given time frame.) Hence the lowest grade E would be given if only assignments 1a and 2a are solved and if *any* of them are presented outside the time frame of the course. (Corresponding to 1 point: 1 for 1a, 1 for 1b, but -1 point for presentation outside the time frame of the course.) Variations of this of course can be had. I am also ready to discuss extra assignments if you wish to reach a high grade but have not had the possibility of presenting *all* your solutions within the time frame of the course. (Unforseen things may happen.) We can decide on this individually *and* according to your interests.

Assignment 1a - Cryptography Using The RSA Algorithm [1 point]. This assignment will be solved in Java utilizing the `BigInteger` class and communication through stream sockets. Before beginning to work on the assignment, read through the Cryptography section in 4.5 Applications of Congruences and/or the Swedish material on cryptography presented at the course webb (kap6.pdf and ovnkap6.pdf). Also get the Java programs `P2PTCP.java` and `StringSender.java`. Rewrite the `P2PTCP` program so that when it runs as a server, it publishes a public RSA key with a size specified by a command line argument. Any time a client connects to a server, the server sends this key to the client which in turn receives it and encrypts a random integer between 1 and 100 and sends to the server. The server then decrypts the integer and writes it on the screen. Work with sending objects of the class `BigInteger` in text format (`String`). (If you wish you can also serialize the objects which would be a more object-oriented approach, choose what you feel most comfortable with.) Write the program so that it runs in a loop where the user has an opportunity to send several integers. (This requirement also holds for 1b.)

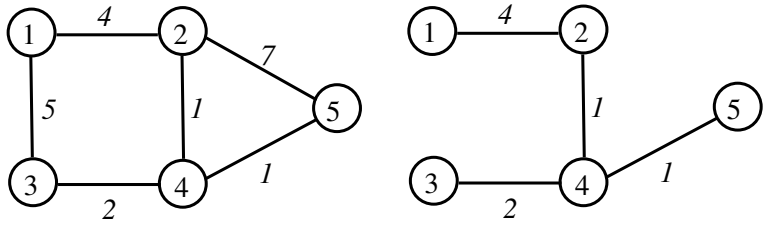
Assignment 1b - Completely Safe Communication Using The RSA Algorithm [1 point]. (Note: That communication is completely safe is a *theoretical* standpoint. In later courses you will learn that *no* communication can be completely safe for pragmatical reasons - as a preliminary exercise, see if you can identify the weakness of the following scenario, *why* is this not safe?)

Based on the preceding assignment, consider the following cryptographic scenario: Two parties, Alice and Bob, hence called A and B, wish to establish a completely safe communication that no third party, Eve, hence called E, can break into. To do this, A and B each create one instance of RSA so that both A and B have a their own public and secret keys. The protocol now to establish safe communication is:

1. A sends her public key to B.
2. For B to certify that B is really talking to A, B generates a secret number, `s`, encrypts it with the A's public key forming the number `s'` and sends to A.
3. Using her secret key, A decrypts the secret `s'` number and sends back `s` to B which can then be certain that it is A that he is communicating with. Now B has a safe channel to A: B can talk to A.
4. To establish safe communication in the other direction, the whole procedure is done with roles reversed and with the other set of keys that B has. Another option is that B generates a symmetric key and send that in encrypted form to A and then, henceforth, the secret symmetric key is used for safe communication. (This is the way it is done, RSA is often used as a platform to exchange symmetric keys rather than encrypting the whole message.)

Write a program that enables this scenario. Can you describe why this is not safe in reality? Note that E can hear all the communication and send things too to either A or B, how can E break this? (This is the theoretical foundation of *certificates*.)

Assignment 2a - Finding a Minimal Spanning Tree [1 point]. Fire up Octave in the virtual machine *Groucho* obtained from the course webb. Here, represent a weighted graph by so-called *weighted adjacency matrices* and implement an algorithm to obtain the edges in a minimal spanning tree for the graph at hand. Consider the following graph with a minimal spanning tree:



The so-called *weighted adjacency matrix* would here be given by the octave code

```
>> M=[0 4 5 0 0;4 0 0 1 7;5 0 0 2 0;0 1 2 0 1;0 7 0 1 0]
M =

   0   4   5   0   0
   4   0   0   1   7
   5   0   0   2   0
   0   1   2   0   1
   0   7   0   1   0
```

That is, row one describes all edges incident on vertex 1, there are two of these edges, one with weight 4 and one with weight 5, leading to node 2 and 3 respectively, this being indicated by the presence of a 4 (for the weight) at column 2 (for the node the edge leads to) and a 5 (for the weight) at column 3 (for the node the edge leads to). The other rows are of course formed analogously. The task in this assigment is to create an Octave program (possibly composed of several functions) to compute and output a minimal spanning tree. An invocation of a function part of such a program could have the following appearance:

```
>> minimalspanningtree(M)
ans =

   2   4
   4   5
   3   4
   1   2
```

And this would then mean that the minimal spanning tree is specified by four edges, the first incident on nodes 2 and 4 (which has weight 1), the second incident on nodes 4 and 5 (which also has weight 1), the third incident on nodes 3 and 4 (which has weight 2), and the last incident on nodes 1 and 2 (which has weight 4). This minimal spanning tree is depicted above beside the graph.

Guidance. To solve this problem, it is necessary to gain understanding in how matrices, functions, conditionals and loop constructs work in Octave. The solution will probably consist of multiple functions calling each other and you need to use quite a bit of understanding concerning what a tree really is, it is especially useful to know that a minimal spanning tree of a graph is a subtree of the graph that contains all vertices, as such it contains no loops and the number of edges is exactly one less than the number of vertices. (A tree with v vertices always have $v - 1$ edges.) For starters, write simple functions that work with matrix representations of graphs, find out how to determine whether there is a path from one node to another based on the matrix representation etc. (Here the so-called *Path Matrix* is useful.)

Assignment 2b - Dijkstra's Algorithm [1 point]. Use your acquired Octave-skills to implement Dijkstra's algorithm for shortest path between two given vertices in a weighted graph. The function invocation in Octave should look somthing like

```
shortestpath(M,v1,v2)
```

where `M` is a weighted adjacency matrix and `v1` and `v2` are two vertices in the graph. You can output the path as a sequence of vertices.