Data Structures 2018
Exercise 12 (Week 49)

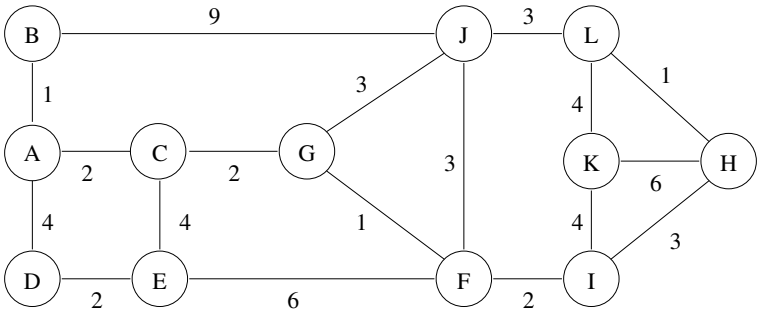
- **Notice that based on university's new regulations on degrees, you can have a degree fail from the course which is put to the register.**

  If a student does not participate in the course and does not cancel his/her enrollment, or if he/she discontinues the course, he/she will be assigned a fail grade for the course in question.

- **Students who participate in exercise group must be in place before the exercise group begins (12.15/14.15/16.15). Students who come late do not get the exercise points.**
- **Check the numbers of exercises made before you come to the exercise group. By this means we can save a lot of time when filling the exercise point list.**
- **Notice that pseudocode does not mean the same this as Java code. Pseudocode is not a programming language dependent presentation for an algorithm.**

1.-2. In the file Graph.java is a simple implementation of a undirected graph as an adjacency matrix and some of the operations (e.g. depth-first search). The marking of vertices and edges has been implemented by implementing classes for the vertices and edges containing a member variable called marker for this purpose.
Breadth-first search can be implemented with the help of a queue: implement this variation of breadth-first search. The queue previously implemented on the course is given in Queue.java. Hint: The implementation is almost identical to that of depth-first search, except in this case the algorithm is implemented in a while-loop instead of the recursion and uses the queue.
Here is how you can use the queue: First, add the start vertex to the queue. Then repeat the following until the queue is empty: dequeue the first element, check the edges attached to it and add the unvisited neighbour elements to the end of the queue.
Test the algorithm with some simple graphs by using the test program GraphTest.java.

3. Find the shortest paths from H in the following graph using Dijkstra's algorithm.



4. Form a minimum spanning tree for the graph in problem 3 using Kruskal's algorithm.
5. Form a minimum spanning tree for the graph in problem 3 using the algorithm of Prim and Jarnik.
6. Describe how brute-force match matches *ababc* to *abcbabcababcab*.
7. Form the failure function for the string *abcababcd* and match it to *abcabcbabcababcdacd* using the Knuth-Morris-Pratt -algorithm.