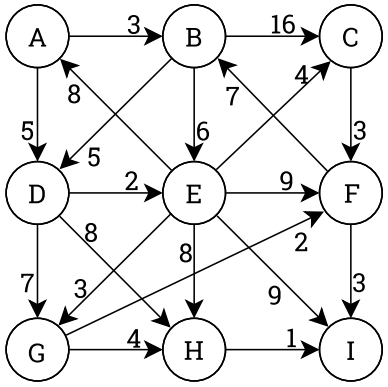
Data Structures and Algorithms II

20./21.4.2017

Exercise 3

13. Simulate manually the functionality of Dijkstra algorithm for attached weighted directed graph with A as the starting vertex. Write the modified contents of distance vector D after each stage (selection of vertex). As you maintain the distance vector D, maintain also the predecessor vector P. Whenever you find a new shorter path to vertex w via vertex v, update $P[w]=v$. Using this predecessor vector, we get the actual paths after the main algorithm has terminated.



14. Write an algorithm that finds the set of root vertices of a directed graph. A root vertex is such vertex that has no incoming edges. Time complexity?

15. Write an algorithm that finds and returns the "near" vertices of a given vertex v in graph G. To a near vertex, there is a path of cost at most k (as the sum of weights of the edges of the path) from vertex v. Parameters are thus a graph, starting vertex, and maximum path cost. Return value is a set of vertices. Use depth or breadth first search and limit the depth of search by the cost of the path.

The following task X1 is obligatory for all students. X-tasks must be done **oneself** by each student. Copies/versions of the same answer won't be accepted. Answers must be sent by Wednesday 19.4. 21:00 using the instructions below. You'll receive an automatic reply by email soon after successful submission. If you won't get the email reply, something went wrong. If the reply contains compiler errors, there is something wrong in the file. Then **resend a fixed version**. The answer must contain a short **self evaluation** where you evaluate the functionality, correctness, time complexity, and possible points of improvement of your solution. A correct self evaluation (for a full answer) is worth one point. The points of these tasks form a part of course evaluation.

Send your solution using a www-form, address and credentials of which you got by email. The solution should be a compilable Java source code file of name *userid*`.java` where *userid* is the first part of your email address. Also the **class name** of your solution must be *userid*. As the submission is Java source code, the self evaluation must be in comments of the program.

Take a skeleton from course www-page. Do not change the header (name, parameters) of the X-task method(s). Please make sure that the program/class is compilable as such, i.e., have the whole answer in the same class and do not use a package.

X1. Write an algorithm that finds the heaviest tree component of a directed graph. The weight of a tree is the sum of the weights of its edges. A legitimate tree is a separate full component of a graph that has no back, forward, or cross edges (within the tree, or from outside). Algorithm returns the weight of the heaviest tree, or *Float.NaN* if there is not a single legitimate tree in the graph. The heaviest tree is also placed to the list of vertices given as a parameter, root as the first element of the list. What it the time complexity of your algorithm?

Do not start coding (or touch the keyboard) before you have a proper plan and a working algorithm on paper. Make your solution modular, solve the problem in stages.

Hints: You need to first find the potential root vertices in the graph, then check that the components of the roots really are distinct trees and calculate the weights of the trees. Take a skeleton and test program from course www-page. A partial solution is better than no solution. If you make a partial solution, write the self evaluation correspondingly.