Algorithms and Complexity
2015
Mästarprov1: Algorithms

Mästarprov 1 should be solved individually in written form and presented orally. No collaboration is allowed.

Written solutions should be handed in latest on **Monday, February 23rd 17.00,** to Johan in written or printed form (personally or his mailbox). Be sure to save a copy of your solutions. Mästarprov 1 is a mandatory and rated part of the course. The test consists of four tasks. The test is roughly graded as follows: Two task correctly solved give an E. Three tasks correctly solved give a C and all tasks correctly solved give an A. You can read more about the grading criteria and the final grade on the course web page. The report should be written in English.

In all problems you should give an analysis of the time complexity of your algorithm and you should be able to argue for its correctness.

1. You are given an undirected graph G with n nodes. The graph is represented with adjacency lists. Is the graph a tree or not? You want to decide it algorithmically in time $O(n)$.

Observe that this can easily be done in time $O(|E|)$, but you are asked to do better than this. Also observe that you know n from start but you don't know $|E|$ *explicitely.* (Can be computed though.)

Describe an $O(n)$-algorithm that solves the problem.

2. Directed graphs can be used to give abstract representations of projects divided into interdependent sub-projects. We get a graph G with nodes $p_1, p_2, ..., p_n$ representing sub-projects. A directed edge (p_i, p_j) means that p_i must be done before p_j. We can assume that G is *acyclic*, i.e. there are no directed cycles in G and that the ordering is *topological* so that (p_i, p_j) is an edge implies $i < j$. Let us furthermore assume that there is a starting node p_0 with no incoming edges and that it is possible to start at p_o and perform all p_i :s using paths from p_0. To each node p_i we have assigned a time t_i it will take to perform (just) p_i. But if we want to complete p_i there is a set of other task we must perform first so there will be a total time T_i it takes.

Use dynamic programming to find all values T_i.

3. Let us study a flow network N. We know that the Ford-Fulkerson algorithm gives us a maximal flow from s to t. Let us assume that the flow has the value A. We are not satisfied with this. In fact, we need a flow $A+k$ where $k > 0$ and we therefore have to increase some capacities for the edges. The FF algorithm gives us a minimal cut (X, Y) with s in X and t in Y. (See textbook or lecture notes.) If e is an edge with capacity $c(e)$ going from X to Y, a simple idea is to increase the capacity of e to $c(e) + k$. Will this work? Obviously, the capacity of the cut will be increased to $A + k$. But there could be another minimal cut without e and then the idea won't work.

We now study this problem: Given a network N, is there a *unique* minimal cut in N or not? Design an efficient algorithm that solves this problem.

4. We have n circles $C_1, C_2, ..., C_n$ in the plane. The circles C_i are given with with midpoint (x_i, y_i) and radius r_i. We want to find the maximal size of an overlap between the circles. To be more precise, we say that a point $p = (x, y)$ is inside the circle C_i if $(x - x_i)^2 + (y - y_i)^2 \leq r_i^2$. If p is inside exactly k circles we say that p represents a k-overlap between circles. The problem is then to find the largest value for k. (We don't have to find a particular p representing the overlap.)

Give an algorithm that solves this problem. It should have time-complexity $O(n \log n)$. You can assume that the following problems can be solved in constant time: To find the intersection points between two circles and to find the intersection points between a line and a circle. (It can happen that the two points degenerate to just one.)