Algorithms and Complexity
2016
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 **Friday, February 26th 17.00,** to Mladen (course assistent) in written or printed form, personally or his mailbox. (Yes, we would like to have it in paper form.) 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. A student has a number of subjects to study. She has N days to do it. Let us number the days $1, 2, ..., N$. Every day she has K ours left for her studies. For each subject i there is an number t_i of hours needed to master the subject. For each subject there is a deadline d_i such that $1 \leq d_i \leq N$. To be precise, the task should be completed at the end of the day. Is it possible for her to plan her studies so that no deadline is exceeded? How should the studies be planned? We are allowed to split the studies on several different days. Find a greedy algorithm that tells her which subjects to study each day.

2. We will now look at a variant of the max-flow problem. Assume that we have a network of n computers connected in i graph G. One computer s is a *sender* and a set R of computers are *receivers*. Then, of course, there are other computers. To state it formally, $R \subset V(G),\ s \notin R$. We assume that all computers in R are connected to s in the network. We are now going to send information from s to all nodes in R. We will think of this information sent as a flow and the value of the flow as a transmission rate. For each node $v \in R$ we have a demand that the node should get a flow d_v. We can assume that s can send an unlimited amount of information. The communication protocol we use works so that each computer in the network processes the information a certain time before sending it forward. This means that for each node in $v \in V(G) - R - \{s\}$ there is a number α_v such that the flow trough v cannot be greater that α_v. (So observe that the *capacities* are on the nodes instead of the edges). Your task is now to construct an efficient algorithm that decides if it is possible to get a flow of this type, i.e., a flow from s to the nodes in R such that each $v \in R$ gets at least d_v. Analyse the complexity of your algorithm carefully.

3. In this problem we assume that we have a set $F_1, F_2, ...F_n$ of n two-dimensional figures. The figures are pieces of paper and they are connected (that is in one piece). If we can take figure F_i, rotate it, translate it and place it all inside F_j,we say that F_i is *contained* in F_j. Another equivalent way of defining this relation is to say that we can cover F_i with F_j. We are now interested in finding a the largest k such that it is possible to find a sequence of figures $F_{i_1}, F_{i_2}, ..., F_{i_k}$ such that F_{i_t} is contained in $F_{i_{t+1}}$ for $t = 1, 2, ..., k-1$. You can not use a figure more than once. Your task is to describe an algorithm that solves this problem. We will assume that we do not know much about the figures. The only information we get is a matris C with dimension $n \times n$ with

$$c_{ij} = \begin{cases} 1 & \text{if } F_i \text{ is contained in } F_j \\ 0 & \text{otherwise} \end{cases}$$

Your algorithm should be polynomial in n.

4. We will study a very abstract deductive system. We have a set of n sentences $s_1, s_2, ..., s_n$. We can deduce certain sentences from other with the help of a set of k *basic deduction rules.* A basic deduction rule will be written in this form: Let U be a set of sentences and s a sentence. Then $[U; s]$ means that s can be deduced from the set U. Informally, we can think of a proof of a sentence s from a set A as a sequence of a finite number of applications of basic deduction rules. To put it more formally, we will use the notation $D(A; s)$ to mean that s can be *derived* from A . We can define D recursively by:

a. If $s \in A$ then $D(A; s)$.

b. If there is a set C such that all sentences in C can be derived from A in a finite number of steps and there is a basic deduction rule $[C; s]$, then $D(A; s)$.

We should note some things that follow from this definition:

If $D(A; s)$ and $A \subset A'$, then $D(A'; s)$.

If $D(A; s)$, then it is possible that there is a set $A_0 \subset A$ such that $D(A_0; s)$.

Since $D(\{s\}; s)$ it is possible to have arbitrarily long proofs.

So given a set B of sentences and a sentence s we want to know if $D(B; s)$ is true or not. Construct an algorithm that decides this. Assume that the input is $M = \{s_1, s_2, ..., s_n\}$, a specified subset $B \subset M$, a specified sentence $s \in M$ and a set of k basic deduction rules given on the format described above. Your algorithm should be polynomial in n and k.

(This is a very simplified form of deduction. In normal cases we do not assume that the set M is limited. It is also common to have the "basic deduction rules" in a more general form. For instance, we could have $[\{P, P \Rightarrow Q\}; Q]$ for *all* sentences and not just for some sentences. This means we do not have a limited number k of rules.)