Algorithms and Complexity
2014
Mästarprov2: Complexity

Mästarprov 2 should be solved individually in written form and presented orally. No collaboration is allowed.

Written solutions should be handed in latest on **Tuesday, May 6th 17.00,** to Johan (personally or in mailbox). Be sure to save a copy of your solutions. Mästarprov 2 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. After you have submitted your work you should sign up for an oral exam.

When you solve the problems you can use the fact that some problems are known to be NP-Complete. These are the NP-Complete problems mentioned in the textbook and in the lecture notes. You can also use the assumption $P \neq NP$ (which, of course, could turn out to be false).

1. A variant of subset sum

The normal subset sum problem can be stated in this form: Given a set $a_1, a_2, ..., a_n$ of positive integers and an integer M, is it possible to find a set $e_1, e_2, ..., e_n$ where $e_i \in \{0, 1\}$ such that $\sum_{i=1}^{n} e_i a_i = M$?

Now we change the problem so that instead of $e_i \in \{0, 1\}$ we demand that $e_i \in \{0, 2\}$. We can call this problem DOUBLE SUBSET SUM.

Show how we can reduce SUBSET SUM to DOUBLE SUBSET SUM.

Then show how to reduce DOUBLE SUBSET SUM to SUBSET SUM.

Which one of the reductions show that DOUBLE SUBSET SUM is NP-Hard?

2. Partitioning of a network

Let us assume that we have a network consisting of persons and that we have a relation *friend* where $friend$(Jonas,Anna) = 1 tells us that Jonas and Anna are friends and $friend$(Anna,Linda) = 0 tells us that Anna and Linda are not friends. The relation is assumed to be symmetric and reflexive but not necessarily transitive. If P is the set of all persons in the network and $A \subseteq P$, we then say that the set A has friendship density $FD(A) = k$ if the probability that two different persons chosen randomly from A are friends is k.
Another way of stating it is that $\frac{\#[\text{pairs of friends in } A]}{\binom{|A|}{2}} = k$.
Let us then assume that we are interested in finding sets A with a high value for FD. We can call them *high density sets*. More precisely, we can take a number L and try to find large sets with FD greater than L. We can define this as an optimization problem:

Input: A network in the form of a graph G with $V(G) = P$ and with edges $E(G)$ representing friendships and a real number L such that $0 < L < 1$.

Goal: Find the largest size of an $A \subseteq P$ such that $FD(A) > L$.

Formulate the corresponding *decision problem*. Show that this problem is in NP and then show that this problem is NP-Complete by reducing the problem CLIQUE to our problem.

3. Finding service providers

Let us say that we run a company that, in some general sense, processes information. Some of these processes are too difficult for us to handle and we want to find some other resources that can handle them for us. Let us say that the processes are named $Pr_1, Pr_2, ..., Pr_n$. When we run the processes there are amounts of information that need to be handled for each process and we assume that these different amounts are known. Let us call them $L_1, L_2, ..., L_n$. (We don't have to care about what units we use.) These numbers can be seen as *weights* of the processes.

Now we have W service providers which we call $SP_1, SP_2, ..., SP_W$. They can all handle the processes but they have different ways of setting the prices for the services they provide. The prices are set in the following way:

Let us look at SP_j and let us assume that we want the provider to handle the processes $Pr_{i_1}, Pr_{i_2}, ..., Pr_{i_j}$. The price of this will consist of two parts, the first being a *fix* price $f_{i_1,j} + f_{i_2,j} + ... + f_{i_j,j}$ (SEK). The second part is based on the weights. Set $S_j = \sum_{k=i_1}^{i_j} L_k$. The provider has lists $R_{1,j}, R_{2,j}, ..., R_{M_j,j}$, $g_{1,j}, g_{2,j}, ..., g_{M_j,j}$ where $0 < R_{1,j} < R_{2,j} < ... < R_{M_j,j}$ and $0 < g_{1,j} < g_{2,j} < ... < g_{M_j,j}$. If $R_{p-1,j} < S_j \leq R_{p,j}$, the *load dependent* price is g_{p_j}. At the endpoints of the range we have $0 < S_j \leq R_{1,j}$ giving price g_1, $S_j = 0$ giving price 0 and $R_{M_j,j} < S_j$ giving infinite price (impossible to perform).

So if $R_{p-1,j} < S_j \leq R_{p,j}$ we get the total price $f_{i_1,j} + f_{i_2,j} + ... + f_{i_j,j} + g_{p,j}$. We call this cost C_j. We now try to decide from which providers we should order which services. Each service should be ordered from exactly one service provider. This means that the set of the n processes should be split on the W providers. The total cost is then $C = \sum_{j=1}^{W} C_j$. We want to minimize this C. We formulate this as a decision problem:

Input: An integer n. A list $L_1, L_2, ..., L_n$ of real numbers. An integer W. A list $M_1, M_2, ..., M_W$ of integers. Sets of lists of real numbers $f_{i,j}$ where $i = 1, 2, ..., n$ $j = 1, 2, ..., W$. Sets of list of real numbers $g_{1,j}, g_{2,j}, ..., g_{M_j,j}$ and $R_{1,j}, R_{2,j}, ..., R_{M_j,j}$ where $j = 1, 2, ..., W$. A real number T.

Goal: Is it possible to assign each process i to a unique service provider j such that the total cost C computed as described earlier is $\leq T$?

Show that this problem is NP-complete.

(Hint: Even if the problem seems complicated with all indices, it is possible to find a reduction that simplifies everything.)

4. Paths in a network

In this problem we have a set M of web-pages with links between the pages. We can represent the set and the links with a directed graph G where the directed edges indicate (directed) links. By a path in G we mean a sequence $P_1, P_2, ..., P_k$ of nodes such that $(P_1, P_2) \in E(G), (P_2, P_3) \in E(G), ...$ and $P_i \neq P_j$ if $i \neq j$. (Which of course is the normal definition of a path).

For some reason, we are interested in finding long paths in G. We can not expect this problem to be efficiently solvable. Let us nevertheless assume that we have an algorithm $F(G, K)$ that tells us if G contains a path of length K ($F =$ TRUE) or not ($F =$ FALSE). The length is defined as the number of nodes in the path, so we require that K shall be a positive integer. We assume that the running time of F is $T(|V(G)|, |E(G)|)$. This algorithm doesn't give us the actual paths.

Construct an algorithm $F_1(G)$, such that it gives us a path of longest possible length in G. The algorithm must use F and it must have a worst case time-complexity $O(P(|V(G)|+|E(G)|)T(|V(G)|, |E(G)|))$ where $P(x)$ is a polynomial in x.