Algorithms and Complexity
2019
Mästarprov 2: 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 **Thursday, April 25th 23.00,** on Canvas.

Mästarprov 2 is a mandatory and graded part of the course. The test consists of four tasks. The test is roughly graded as follows: Two tasks 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. Experimental Cooking

We want to try some experimental cooking. We have a list of n possible ingredients and we want to mix them. But they cannot be mixed in any crazy way, some don't go well with others. We write down an $n \times n$ matrix D called the *discord matrix*. The element D_{ij} is a real number between 0 and 1 where 1 means "i and j don't go together at all" and 0 means "i and j go together perfectly". When we now try to experiment we want to choose as many ingredients as possible, but at the same time have as little *total discord* as possible. We define total discord as the sum all discords between the chosen ingredients. We define the following problem:

EXPERIMENTAL COOKING
Input: An $n \times n$ matrix D with real numbers between 0 and 1. An integer k. A real number t.
Goal: Is there a subset S of $\{1, , ..., n\}$ of size k such that if $dsum$ is the sum of all D_{ij} such that $i < j, i \in S, j \in S$ then $dsum \leq t$?

So for instance, if

$$D = \begin{pmatrix} 0.0 & 0.4 & 0.2 & 0.9 & 1.0 \\ 0.4 & 0.0 & 0.1 & 1.0 & 0.2 \\ 0.2 & 0.1 & 0.0 & 0.8 & 0.5 \\ 0.9 & 1.0 & 0.8 & 0.0 & 0.2 \\ 1.0 & 0.2 & 0.5 & 0.2 & 0.0 \end{pmatrix}$$

and $S = \{1, 3, 5\}$ then $dsum = 0.2 + 1.0 + 0.5 = 1.7$.

Show that this problem is NP-Complete. (Hint: You can use INDEPENDENT SET.)

2. A Variant of SUBSET SUM

We know that the problem SUBSET SUM, i.e., given positive integers $a_1, a_2, ..., a_n, M$, decide if there is a subset sum equal to M is NP-Complete. But maybe we could get a simpler problem if we look at a ”softer” variant:

Input: Positive integers $a_1, a_2, ..., a_n$. A positive integer M.
Goal: Is there a subset of $a_1, a_2, ..., a_n$ with sum S such that $|M - S| < d$ where $d = \lfloor \log M \rfloor$?

We might now ask if this problem, we can call it ALMOST SUBSET SUM, can be solved in polynomial time. But we can prove that it is NP-Complete by reducing the ordinary SUBSET SUM problem to ALMOST SUBSET SUM. Your task is to show how such a reduction can be done. We give some hints: Assume that we have an instance $a_1, a_2, ..., a_n, M$ of SUBSET SUM. We make a special instance $a'_1, a'_2, ..., a'_n, M'$ of ALMOST SUBSET SUM.
How should $a'_1, a'_2, ..., a'_n, M'$ be chosen so that the instance of SUBSET SUM has a solution if and only if the instance of ALMOST SUBSET SUM has a solution?

3. Party Planning

In this problem we imagine that you are planning a party for n persons. They are to be placed at different tables. You want everyone to be comfortable and therefore think it would be a good idea if everyone seated at a table knows everyone else at the table. So imagine that you have a list that looks like: $\{(p_1, p_3), (p_4, p_7), \cdots (p_i, p_j) \cdots\}$ where a pair (p_i, p_j) means that persons p_i and p_j know each other. We assume that this relation is symmetrical but not necessarily transitive. You have k tables available. We assume that the tables are large so that each table has space for n persons, if you want. Your problem is to decide if, given the list, it is possible to find a seating arrangement as described above. Since you are a computer scientist you naturally want to solve the general problem for any lists and any number k of tables. But then you realize that this problem seems hard. Decide if this problem is NP-Complete or not by either giving a proof of NP-Completeness or giving an *efficient* algorithm that solves the problem

4. A variant of CNF-SAT

Let $\phi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ be a CNF formula. The CNF-SAT problem is to decide if all m clauses can be satisfied at the same time. A perhaps more complicated problem is to decide if there is a set of values for the variables so that exactly k clauses are satisfied where k is an integer such that $0 < k \leq m$. We can see that if we set $k = m$ we get the standard CNF-SAT problem.

This new problem is NP-Complete. Let us nevertheless assume that we have an algorithm $F(\phi, k)$ that decides the problem. Such an algorithm can probably not be efficient, but could still be useful. But let us assume that we are not just interested in knowing if it is possible to satisfy exactly k clauses but we want to know how we can do it. Let us assume that there are n variables in the formula. Your task is to design an algorithm $G(\phi, k)$ that gives us a valuation (values for all variables) such that exactly k clauses are satisfied (if such a valuation exists). The algorithm G can use calls $F(\phi', i)$ repeated times. (ϕ' might differ from ϕ and i might differ from k.) But the algorithm should use F no more than $q(n, m)$ times where $q(n, m)$ is *polynomial* in n and m. Furthermore, if we assume that the time complexity of $F(\phi, k)$ is $T(n, m)$, the time complexity for $G(\phi, k)$ should be $p(n, m) + q(n, m)T(n, m)$ where $p(n, m)$ is polynomial in n and m. Design such an algorithm G and find p and q.