Algorithms and complexity
Hösten 2009
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, November 16th 17.00,** in my postbox or in the student reception in Osquars backe 2 plane 2. 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 in course-PM, or on the course web page.

1. Currency exchange

The country Brutopia has currency brutos. In the country there are five types of notes. They have denominations of 50, 40, 10, 5, 1 brutos. The notes are old and the government of Brutopia therefore wants its citizens to use as few notes as possible. Therefore, it wants the citizens to pay exactly the right amount and do it with as few notes as possible. We have thus the following problem:

Given an integer A, we want to write

$$A = k_{50}50 + k_{40}40 + k_{10}10 + k_5 5 + k_1$$

where $k_{50} + ... + k_1$ should be as small as possible.

Help the brutopians by designing an algorithm that solves this problem. It must be a greedy algorithm which has time complexity $O(\log A)$.

2. Shortest paths

We have seen examples of several algorithms which calculate the shortest distance between nodes in weighted graphs. We have also seen that a shortest path not always is the path with the least number of edges. But now, let us deal with two distance concepts simultaneously. We set

$l(w)$ = length of the path v, measured as the sum of edge weights.

$l_{\#}(v)$ = length of the path v as measured by the number of edges.

We assume that we have a graph where there might be paths of equal length in $l(v)$-sense between two nodes. We then want to choose a path which is minimal

in the $l_{\#}v$-sense among them. More precisely:

We are looking for a path v^*, $i \to j$ such that

1. $l(v^*) \leq l(v)$ for all paths iv: $\to j$.

2nd $l_{\#}(v^*) \leq l_{\#}(v)$ for all paths iv: $\to j$ such that $l(v^*) = l(v)$.

Construct an algorithm that finds such a path. For full score it is required that the algorithm works in graphs which might have negative edge weights but no negative cycles.

3. A Packing Problem

The transport company Godbud transports goods in trucks. The goods are packed in boxes $p_1, p_2, ..., p_n$. These boxes have weights $w_1, w_2, ..., W_n$ kg. They will be loaded in trucks that are able to take just M kg. The boxes must be loaded into the trucks in the right order. This means that boxes $p_1, p_2, ..., P_{s_1}$ packed into truck 1 for some s_1. Truck 2 is packed with the boxes $p_{s+1}, p_{s+2}, ..., p_{s_2}$ for some s_2 and so on. Then $w_{s_i+1} + ... + w_{s_{i+1}} \leq M$ for all i. The miljötrafikövervakningsverket (a fictive organization) now puts an additional requirement on the company. Godbud must load the trucks so that $d_i = M - p_{w_i+1} + ... + w_{s_{i+1}}$ is as small as possible for all i. More precisely, it is required that $D = \sum_{i=1}^{n} d_i$ should be as small as possible.

Construct an efficient algorithm that solves this problem, i.e. decides what the optimal D is and how the $s_1, s_2, ...$ are to be chosen.

4. To search a matrix

We have seen that if we want to determine whether a given element a is in a list $x_1, x_2, ..., x_n$, it takes $O(n)$ operations if the list is not sorted. If the list is sorted it is possible to determine whether a is in the list with $O(\log n)$ operations.

Suppose now that we have an $n \times n$ matrix with n^2 numbers x_{ij} sorted in increasing order along the rows and columns. We can also assume that all numbers are different. This means that $j < k \Rightarrow x_{ij} < x_{ik}$ and $x_j < x_k$ for all i.

We now want to determine if a number a is an element x_{ij}, that is, if a is in the matrix. It is easy to see that we can do this with $O(n \log n)$ operations. But we can do better? Use decomposition to construct an algorithm. Is it possible to find a with $O(n)$ operations?