Algorithms and Complexity
2013
Extra Mästarprov1: Algorithms

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

This test is given to students who failed to get E on the ordinary Mästarprov 1. It consists of four problems. If at least two problems are solved correctly the test gives grade E. Your solutions should be handed in latest May 27.

1. Fixpoints in sequences

Given a sorted array of distinct integers $A[1, ..., n]$, we say that a fix point is an index i such that $A[i] = i$. Of course, there doesn't have to be any fix points in a sequence. Design a divide-and-conquer algorithm that runs in time $O(\log n)$ and decides if there are any fix points in the seqeunce. The algorithm doesn't have to find all fix points, just decide if there are any fix points. Prove that the time complexity is correct.

2. Identifying corrupted strings

In this problem we are given a string of characters $s[1, 2, ..., n]$, which we believe to be a corrupted text document in which all punctuation has vanished. The text could look something like 'itwasasmallroomlookingoutonthebackgarden'. We have an electronic dictionary in the form of a Boolean function $dict()$ such that

$$dict(w) = \begin{cases} 1 \text{ if } w \text{ is a valid word} \\ 0 \text{ otherwise} \end{cases}$$

We assume that this test can be done in time $O(1)$.

We now want to know if the string s is a valid string of words or not. (We donÂ´t try to decide if it is a valid sentence.) We try to use Dynamic programming.

We define $T(k) = \begin{cases} 1 \text{ if s[1,...,k] is a valid string of words} \\ 0 \text{ otherwise} \end{cases}$

Show that this is a correct recursion formula for $T(k)$:

$$\begin{cases} T(0) = 1 \\ T(k) = max\,\{T(i) \cdot dict(s[i+1, ..., k]) : 0 \leq i \leq k-1\} \text{ for } 1 \leq k \leq n \end{cases}$$

Use the formula to write a program in pseudo-code that decides if $s[1, ..., n]$ is a valid string of words or not. Estimate the time complexity of your algorithm.

3. Reliable connections in a network

You are working in a company which has a set of n computers connected in a network. Not all computers are connected directly to each other, but for each pair of computers we know that there is at least one path in the network that connects them. For each connection between two computers, there is a probability p that the connection might be corrupted. If we have a path, then the probability that the path is corrupted is $1-(1-p_1)(1-p_2)\cdot\ldots\cdot(1-p_k)$ where $p_1, p_2, \ldots, p_k$ are the probabilities for corruption of the connections on the path. Your boss wants to know if, given a small number ϵ, for each pair of computers there is a path between them with a chance of corruption smaller than ϵ.

Your boss wants you develop an algorithm that solves this problem. You start to think about it and realizes that you perhaps can use a famous algorithm you know already. But in order to do that you have to simplify the problem a bit: You want to replace $(1-p_i)(1-p_j)$ with $1-p_i-p_j$. That means that we cancel all products p_ip_j. This means that the probability of corruption of the path will be approximated $p_1+p_2+\ldots+p_k$. Your boss says it is OK to use this simplification. Design an effective algorithm that solves the problem, that is, finds if there for each pair of computers is at least one path with chance of corruption smaller than ϵ. Estimate and prove the time complexity of your algorithm. It should be as efficient as possible.

We assume that the information about the network is given by an array $f[i,j]$ such that

$$f[i,j] = \begin{cases} p \text{ if there is a connection with chance } p \text{ of corruption} \\ \infty \text{ otherwise} \end{cases}$$

4. Winning a game

You and an friend play a game which has the following form: At each step the game consists of two piles of chips. (One of them could be empty). On each chip there is a positive number. You and your friend take turns and choose one pile at each turn and take the top chip from the pile. So for instance, if the piles look like:

2	
4	1
1	7
3	2

and it is your turn you can choose between the top chips 2 or 1. If one of the piles is empty you only have one choice. And if both piles are empty the game ends. The winner is the player with the largest sum on the chips chosen by the player. In this simple type of game it is possible to construct an optimal strategy for each player. By a strategy we mean a rule for how you should chose your pile in every possible situation. By an optimal strategy we mean a strategy that works at least as well as any other strategy when your friend play as well as possible. Design an algorithm that finds such an optimal strategy. We assume that we know the contents of the piles at the start of the game. The algorithm should *precompute* the strategy in time at most $O(n^2)$ where n is the number of chips. Then in every move you should be able to consult your strategy and find the best move in time $O(1)$.