Technische Universität München
Department of Computer Science
Chair for Efficient Algorithms
Prof. Dr. Sami Khuri

Summer semester 2000
Problem Set 9
June 27, 2000

Efficient Algorithms and Data Structures II

Problem 1 (6 credits)

Compute the prefix function π for the pattern `ababbabbababbababbabb` when the alphabet is $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

Problem 2 (6 credits)

Compute the λ and γ functions for the pattern $P =$ `0101101201` when the alphabet is $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$.

Problem 3 (8 credits)

Give examples to show that by combining the bad-character and good-suffix heuristics, the Boyer-Moore algorithm can perform much better than if it used just the good-suffix heuristic.

Turn around.

Problem 4 (20 credits)

Define an *upsequence* of length k in an array $L = L_1 L_2 \cdots L_n$ of numbers to be a sequence $i_1, \ldots, i_k$ with $1 \le i_1 < i_2 < \cdots < i_k \le n$, for which $L_{i_1} \le L_{i_2} \le \cdots \le L_{i_k}$, i.e., to be a not necessarily contiguous subsequence of L which is non-decreasing from left to right.

The problem, given an array L, is to find the greatest length of any upsequence which occurs in L. For example, if L is sorted in decreasing order, the answer is 1.

L may contain astronomically many upsequences, so it would be a bad idea to examine them all (and it is not immediate obvious how to proceed). However, the following idea leads to a reasonably efficient solution. For any prefix $B = L_1 \cdots L_m$ of L $(0 \le m \le n)$, and for any length k of upsequence which occurs at all in B, define the *record economical top element* (RETE) with respect to k and B to be the smallest number which is the last (i.e., biggest) member of an upsequence of length k in B.

Proceed from left to right, letting B grow from the empty sequence to all of L, and at all times keep track of all the RETEs which exist. At the end, the biggest k for which a RETE exists will be the answer.

A surprising fact is that as we consider each of the numbers $L_1, \ldots, L_n$ in turn, we will find that it is a new RETE in exactly one way. For example, suppose that $L_1 = 3$, $L_2 = 5$, $L_3 = 2$, and $L_4 = 7$. The RETEs for this prefix are:

2 for 1-upsequences,
5 for 2-upsequences,
7 for 3-upsequences.

The fifth number, L_5, will be a new RETE:

for 1-upsequences if and only if $L_5 < 2$,
for 2-upsequences if and only if $2 \le L_5 < 5$,
for 3-upsequences if and only if $5 \le L_5 < 7$,
for 4-upsequences if and only if $7 \le L_5$.

a. Devise an efficient (in the worst case) algorithm to find the length of a longest upsequence of L by carrying out the above idea.

b. Find the running time function of your algorithm.