TEL AVIV UNIVERSITY
Department of Computer Science
0368.4281 – Advanced topics in data structures
Spring Semester, 2007/2008

Homework 3, June 29, 2008

Due on Wed. July 16. Don't forget to keep a copy of the homework.

1. We are given a set of m strings $s_1\#_1$, $s_2\#_2$, $\cdots$, $s_m\#_m$, where the symbols $\#_j$ are distinct sentinel symbols not appearing anywhere else and smaller than all other characters. We want to find the longest substring which appears in at least 10 of the input strings. To that end we have a suffix array, SA, of $s_1, \cdots, s_m$, each entry of SA contains a pair (i, j) if it represents suffix j of s_i. We also have an lcp array containing the longest common prefix of pairs of consecutive suffixes in SA.

Give an algorithm to find the required string in linear time. The algorithm should use as small extra storage (on top of the two arrays above) as you can.

Hint: Note that if SA contains suffixes of at least 10 different strings between positions k_1 and k_2 then the lcp of all the suffixes between positions k_1 and k_2 is a common substring appearing in at least 10 strings.

2. The recurrence for the running time of the algorithm for computing a suffix array presented in class is T(n) = T(2n/3) + O(n). Show how to modify the algorithm to give one whose recurrence is T(n) = T(3n/7) + O(n).

3. We denote the ith character of string s by $s[i]$. Given a string s we define a partition of s (Sometimes called Lempel-Ziv parsing) into blocks $s_1, \ldots, s_k$ as follows. The first block s_1 is the first character of s. Say we defined $s_1, \ldots, s_i$ and the last character of s_i is $s[\ell]$. The ith block s_{i+1} is the shortest substring of s starting with $s[\ell + 1]$ which does not equal to one of the blocks $s_1, \ldots, s_i$. If there is no such substring starting with $s[\ell+1]$ then s_{i+1} is the suffix of s from $s[\ell+1]$ until the last character of s. For example if $s = AABABBBABAABABBB$ then its partition into blocks is $|A|AB|ABB|B|ABA|ABAB|BB$.

a) Describe an efficient algorithm to find the partition of a string s of length n. Analyze your algorithm.

Notice that each block s_j consists of a previous block plus one additional character. For example the block s_5 in the partition above is equal to s_2 followed by "B". We can encode s using a sequence of pairs, one for each block. The pair encoding s_j consists of an index of a previous block s_i, $i < j$, and a character σ such that $s_j = s_i\sigma$. (The index is 0 in case s_i does not exist.)

b) What is the length (asymptotically) as a function of n of the encoding of a periodic string s consisting of n repetitions of AB ($s = (AB)^n$)?

c) What is the Burrows-Wheeler Transform of the string $s = (AB)^n$? We denote this string by $BWT(s)$.

d) Use the Move-To-Front (MTF) encoding to convert $BWT(s)$ to a string of integers, what is the sequence of integers that you obtain ?

e) If you use Huffman code to encode the integer string $MTF(BWT(s))$ into a bit sequence, what would be the length of the sequence ?

f) Can you suggest an improvement to the compression algorithm that uses the BWT followed by MTF, followed by Huffman encoding, so it performs better on periodic strings as s above.

4. Show that we can color a full binary tree with two colors red and black such that it becomes a red black tree if and only if for every node v the length of the longest path from v to a leaf descendant of v is at most twice the length of the shortest path from v to a leaf descendant of v.

5. We perform a sequence of m operations on an (initially empty) heterogenous 2-4 finger search trees (i.e. a finger search tree without level links). Out of these m operations k are concatenations where the ith concatenation concatenates two trees such that the smaller among them contains n_i elements. The other $m-k$ operations are find, insert, delete, and split. The j-th operation among these operations is performed on an element at distance d_j from the beginning of a list containing n_j elements.
a) **Prove** that using the implementations of the operations as described in class one can perform the sequence in

$$O(\sum_{i=1}^{k} \log(n_i) + \sum_{j=1}^{m-k} \log(\min\{d_j, n_j - d_j\}))$$

time. (Hint, first show that the amortized cost of rebalancing during insertion/deletion and catenation is constant. Then analyze the cost of a split carefully.)

b) Improve the bound in the previous question to

$$O(k + \sum_{j=1}^{m-k} \log(\min\{d_j, n_j - d_j\}))$$

time.