TEL AVIV UNIVERSITY
Department of Computer Science
0368.4281 – Advanced topics in data structures
Fall Semester, 2005/2006

Homework 1, Nov 12, 2005

Due on Monday Nov 28. Don't forget to keep a copy of the homework.

1. A string s of length n is *periodic* if there is a string u of length $\leq n/2$ such that

$$s = u^k u' \ , \tag{1}$$

where k is an integer ≥ 2, u^k is the concatenation of k copies of u, and u' is a prefix of u. The smallest period of s is the shortest u for which Equation (1) holds. Suppose you are given a suffix tree of s together with an LCA data structure show how to use it to find the smallest period of S or declare that S is not periodic.

2. A *maximal repeat* of a string s is a substring α of s such that there exist $i < j$ where $s_i, \ldots, s_{i+|\alpha|-1} = s_j, \ldots, s_{j+|\alpha|-1} = \alpha$. Furthermore, if $i > 1$, then $s_{i-1} \neq s_{j-1}$, and if $j+\alpha-1 < n$, then $s_{i+\alpha} \neq s_{j+\alpha}$.
(a) Prove that s has at most n maximal repeats. (Try to find 2 proofs: one based on suffix trees, and a direct argument.)
(b) Give an algorithm to find all maximal repeats in s. Prove its correctness and analyze its running time.
(c) A *nonoverlapping maximal repeat* is one where there exist $i < j$ as before, and in addition $i + |\alpha| - 1 < j$. Give an algorithm that finds the longest nonoverlapping maximal repeat.

3. Given a string s, $|s| = n$, the suffix array, SA, of s, is a permutation of $\{1, 2, \ldots, n\}$ such that $SA[j] = i$ if and only if the suffix of s starting with the character i, $(i = 1, \ldots, n)$ is the j^{th} when we order the suffixes lexicographically. We add a special character \$ to each suffix which is smaller than any other character so that the lexicographic order of the suffixes is well defined.
(a) Given a permutation π of $1, 2, \ldots, n$, is there always a string s of length n such that π is the suffix array of s ? Prove your answer.
(b) Below are three suffix arrays. For each of these suffix arrays find a string s of length n, over the smallest possible alphabet Σ, such that the correponding array is a suffix array of s. Prove that there is no string s' over a smaller alphabet such that the suffix array is a suffix array of s'.

1)

n	n-1	$\cdots$	2	1

2)

1	2	$\cdots$	n-1	n

3) Assume n is even:

n	1	n-1	2	n-3	3	$\cdots$	$n/2+1$	$n/2$

4. The LCP array for a string s is defined such that $LCP[i]$ is the longest common prefix of $SA[i]$ and $SA[i-1]$ (LCP[1] is undefined).

Recall the algorithm of Kasai, Lee, Arimura, Arikawa, and Park to compute the LCP array given the suffix array. This algorithm works on the suffixes in order of decreasing lengths. For suffix i, $s_i, \ldots, s_n$ we have to find its position in SA. If it is in position j, that is $SA[j] = i$, then we calculate $LCP[j]$ which is the longest common prefix of $SA[j]$ and $SA[j-1]$. To find, in iteration i, for which j, $SA[j] = i$ a naive implementation would use the *rank array* which is the "inverse" of the suffix array. That is $RANK[i] = j$ if and only if $SA[j] = i$. Here is a code for this algorithm:

```
for(i=1;i<=n;i++) RANK[SA[i]]=i;
h=0;
for(i=1;i<=n;i++) {
  k = RANK[i];
  if (k==1) LCP[k]=-1 ;
  else {
   j=SA[k-1];
   while(i+h <=n && j+h<=n && s_{i+h}==s_{j+h}) h++ ;
   LCP[k] = h;
  }
  if (h>0) h--;
}
```

This implementation uses in addition to the string s, 3 arrays, SA, R, and LCP. Lets define a mapping, $RANKNEXT$, as follows

$$RANKNEXT[i] = RANK[SA[i] + 1] .$$

This mapping is defined for every $i = 1, \ldots, n$ except for $i = RANK[n]$ and it maps i to the rank of the suffix obtained by removing the first character from the suffix of rank i. Note that

$$RANKNEXT[RANK[j]] = RANK[SA[RANK[j]] + 1] = RANK[j + 1] .$$

Use the $RANKNEXT$ mapping to find an implementation of the algorithm of Kasai *et. al.* that uses only 2 arrays in addition to s rather than 3 arrays.

5. Given a pattern $P = P_1, \ldots, P_m$ of length m, and a text $T = T_1, \ldots, T_n$ of length n ($n >> m$), design an algorithm that for every position $i \leq n - m$ of the text determines whether P matches $T_i, \ldots, T_{i+m}$ with at most $\lceil m/2 \rceil$ errors. Furthermore for every i such that P does not match $T_i, \ldots, T_{i+m}$ with at most $m/2$ errors the algorithm has to calculate the index j of the $\lceil m/2 \rceil^{th}$ error. That is the algorithm has to find $j = j(i)$ such that $P_j \neq T_{i+j}$ and $P_1, \ldots, P_{j-1}$ matches $T_i, \ldots, T_{i+j-1}$ with exactly $\lceil m/2 \rceil - 1$ errors.

Try to get an algorithm with as best performance as you can. Analyze your algorithm.