CS21003 Algorithms I, Autumn 2013–14

Class test 2

Maximum marks: 20 | Time: 14-Nov-2013 | Duration: 1 hour

Roll no: ____________ **Name:** ______________________________

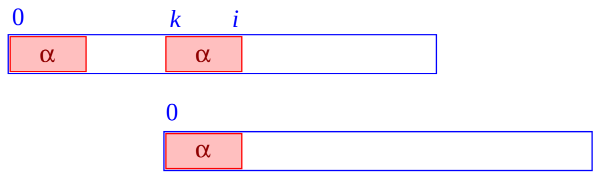
[*Write your answers in the question paper itself. Be brief and precise. Answer* _all_ *questions.*]

1. You are given an array A of n positive integers, each having bit-length $\leqslant l$. Propose an $O(nl/\log n)$-time algorithm to sort A. **(10)**

Solution Let $t = \lceil \log_2 n \rceil$. We perform radix sort with respect to the radix $R = 2^t$. We have $n = 2^{\log_2 n} \leqslant R = 2^{\lceil \log_2 n \rceil} < 2^{\log_2 n + 1} = 2n$, that is, $R = \Theta(n)$. Counting sort with respect to each digit takes $O(n+R)$, that is, $O(n)$ time. The total number of R-ary digits to be considered is $\lceil l/t \rceil = \Theta(l/\log n)$. Therefore, the running time of this radix sort on A is $O(nl/\log n)$. Extracting the R-ary digits of all the elements of A can also be done in the same time.

2. Let T be a string of length m. The *prefix table* of T is an array $P[0\ldots m-1]$ such that $P[k]$ stores the length of the longest common prefix of $T[k\ldots m-1]$ and T (for each k in the range $0 \leqslant k \leqslant m-1$). Propose an algorithm to compute the prefix table P of T, given only the failure function table $F[0\ldots m-1]$ for T. Notice that T itself is not provided as an input to your algorithm—only F and m are supplied. What is the running time of your algorithm? **(10)**

Solution We clearly have $P[0] = m$. So suppose that we want to compute $P[k]$ for $1 \leqslant k \leqslant m-1$. Let α be the longest common prefix of T and $T[k\ldots m-1]$. The following figure demonstrates that α must be a proper border of $T[0\ldots i]$. The problem is that α need not be the longest proper border of $T[0\ldots i]$. Nevertheless, any proper border (like α) can be obtained from the longest proper border by iterating the failure function F. In the code that follows, j stands for the length of α.




```
int *calcpfxtbl ( int *F, int m )
{
   int *P;
   int i, j;

   /* Allocate memory and initialize the prefix table */
   P = (int *)malloc(m * sizeof(int));
   P[0] = m; for (i=1; i<m; ++i) P[i] = 0;

   /* Look at the failure function table. In order that we discover longer borders
      earlier, we look at F[i] values in the decreasing sequence of i. */
   for (i = m-1; i > 0; --i) {
      /* Look at all non-empty proper borders of T[0...i]. Let k = i-j+1. If P[k]
         is non-zero, it is assigned this value in an earlier iteration. Since
         earlier iterations handle larger i, P[k] (if set) is not overwritten. */
      j = F[i];
      while (j > 0) {
         if (P[i-j+1] == 0) P[i-j+1] = j;
         j = F[j-1];
      }
   }
   return P;
}
```

The running time of this algorithm is dominated by the inner while loop. For any given i, the number of iterations in this loop is the number b_i of non-empty proper borders of $T[0\ldots i]$. The running time of the algorithm is $O\left(\sum_{i=1}^{m-1} b_i\right)$. In the worst case (think about strings like a^m or $a^t b a^{2t}$), this can be $O(m^2)$. For random strings, each b_i is expected to be small, provided that the string alphabet Σ has at least two symbols. More precisely, if $s = |\Sigma|$, then $T[0\ldots i]$ has a proper border of length j with probability $1/s^j$ (for $j \leqslant i/2$). In the random case, we expect close to $O(m)$-time performance of this algorithm. A worst-case $O(m)$-time algorithm may exist, but I do not know. The following string demonstrates that we cannot prematurely break the inner while loop whenever some $P[i-j+1]$ is found to be non-zero. We cannot break even when we see an arbitrarily long sequence of non-zero $P[i-j+1]$ values in consecutive iterations of the loop.

$$abacabadeabacabad\,f\,abacabadeabacabad\,g\,abacabadeabacabad\,f\,abacabadeabacabad$$

There exist worst-case $O(m)$-time algorithms to compute P from T, but our current problem is different.

For rough work and leftover answers