Data Structures and Algorithms
FR 6.2 Informatik

Sanders, Telikepalli WS 03/04

http://www.mpi-sb.mpg.de/~sanders/courses/algdat03/index.html

---

Solutions to assignment 6

Exercise 1
The multikey quicksort algorithm for string sorting runs in time $O(D + n \log n)$ where D is the total length of the distinguishing prefixes and n is the number of strings. We ask you to show that this makes it a good suffix sorting algorithm for a string $S[0 : n)$ in the best case and *on the average*, i.e., characters are chosen uniformly and independently at random from an alphabet $\{1, \ldots, \sigma\}$.

1. A naive use of the multikey quicksort algorithm would make copies of all suffixes and hence needs $\Omega(n^2)$ space and time. Outline how this can be avoided.
2. Show that $D = O(n)$ in the *best case* if $\sigma \geq n$, i.e., if you can choose a particular input for given n.
3. Consider two nonoverlapping substrings $s_i = S[i : i + k)$ and $s_j = S[j : j + k)$. Show that the probability that $s_i = s_j$ is σ^{-k}.
4. Consider two substrings $s_i = S[i : i + k)$ and $s_j = S[j : j + k)$ with $i \neq j$. Show that the probability that $s_i = s_j$ is σ^{-k} even if s_i and s_j overlap. Hint: $P[A \cap B] = P[A] \cdot P[B|A]$
5. Show that the probability that there are two identical substrings of length k is bounded by $n^2\sigma^{-k}$. Hint: $P[A \cup B] \leq P[A] + P[B]$
6. Show that there is at most a $1/n$ probability that there are any two suffixes with a common prefix of length $3 \log_\sigma n$ or longer.
7. Now show that the average case running time of the multikey quicksort algorithm for suffix sorting is bounded by $O(n \log n)$.[1] Hint: Assume the runtime of an algorithm is bounded by $f(n)$ with probability $p(n)$ and otherwise it is bounded by $g(n)$, then its expected runtime is bounded by $(1 - p(n))g(n) + p(n)f(n)$.

Solution

1. As usual, we can store the string once ($O(n)$) and then represent each suffix by the index of its first character ($n \cdot O(1)$) so that we need only $O(n)$ space. The multikey quicksort algorithm can be easily adapted to work on this data structure. Basically, we only have to compute $i + \ell + 1$ to access the appropriate character, where i is the respective index of the first character and ℓ the parameter of the procedure (cp. slides).
2. If the string consists of n different characters (which is possible due to $\sigma \geq n$), the first character of each suffix differs from the first character of all other suffixes. Hence, the total length of the distinguishing prefixes is n.
3. The probability that two characters (which are chosen uniformly and independently at random from an alphabet $\{1, \ldots, \sigma\}$) are equal is $1/\sigma$. Since the probability that $S[i+\ell_2] = S[j + \ell_2]$ does not depend on the event "$S[i + \ell_1] = S[j + \ell_1]$", $0 \leq \ell_1 < \ell_2 < k$, we can just multiply the probability $1/\sigma$ k times (according to $P[A \cap B] = P[A] \cdot P[B]$ if A and B are independent) so that we obtain $1/\sigma^k$.

[1] Note that this is optimal for ordered alphabets.

4. We want to show that $P(\ S[i:i+k) = S[j:j+k)\ ) = \sigma^{-k}$, w.l.o.g. for $i < j$. We know that $P(\ S[i] = S[j]\ ) = 1/\sigma$. Furthermore, we can use the formula for conditional probabilities: $P(\ S[i:i+\ell] = S[j:j+\ell]\ ) = P(\ S[i:i+\ell) = S[j:j+\ell)\ ) \cdot P(\ S[i+\ell] = S[j+\ell] \mid S[i:i+\ell) = S[j:j+\ell)\ )$, for $0 < \ell < k$. Under the condition that $S[i:i+\ell) = S[j:j+\ell)$, we have to consider the case that $S[i+\ell]$ is fixed when we compare $S[i+\ell]$ with $S[j+\ell]$. However, $S[j+\ell]$ is still randomly chosen and does not depend on the assumed condition. Hence, we simply get $P(\ S[i+\ell] = S[j+\ell] \mid S[i:i+\ell) = S[j:j+\ell)\ ) = P(\ S[i+\ell] = S[j+\ell]\ ) = 1/\sigma$. Now, we can easily conclude the claimed probability.

5. There are $(n-k+1)(n-k)/2 \leq n^2$ pairs of substrings of length k that does not start at the same position. According to $P[A \cup B] \leq P[A] + P[B]$, we can sum up the probability $1/\sigma^k$ n^2 times to obtain the upper bound $n^2\sigma^{-k}$ for the probability that there are at least two identical substrings.

6. The upper bound of $n^2\sigma^{-k}$ that we obtained in 5 applies to substrings of length k *or longer* because the cases of longer identical substrings are included in the cases of identical substrings of length k. Hence, we can just set $k := 3\log_\sigma n$ and obtain the probability $n^2\sigma^{-3\log_\sigma n} = 1/n$.

7. We distinguish between two cases:

   - There are at least two suffixes with a common prefix of at least length $3\log_\sigma n$. The probability of this case is bounded by $p(n) = 1/n$ due to 6. D is trivially bounded by n^2 so that we get an upper bound for the runtime of $f(n) = O(n^2 + n\log n)$.
   - Otherwise, D is bounded by $n \cdot 3\log_\sigma n$ so that we get an upper bound for the runtime of $g(n) = O(n \cdot 3\log_\sigma n + n\log n) = O(n\log n)$.

   The expected runtime is bounded by $(1-p(n))g(n)+p(n)f(n) \leq O(n\log n)+O(n+\log n) = O(n\log n)$.

Exercise 2

The *Burrows-Wheeler* transform of a string T of length n is defined as follows: Assume that T is terminated by a special sentinel character \$ that is considered smaller than any other character. Now consider the $n \times n$ matrix of characters where each row contains a different cyclic rotation of T. Sort the rows lexicographically. $BW(T)$ is the rightmost column of the sorted matrix.[2] For example, $BW($`mississippi$`$) =$ `ipssm$pissii`.

What is $BW($`abracadabra`$)$? Assume you are given a suffix array of T. Explain how to derive $BW(T)$ in linear time using the suffix array.

Solution

$BW($`abracadabra$`$) =$ `ard$rcaaaabb`

The fact that \$ is the smallest character means that after the Burrows-Wheeler transform the sorted rows will have the same order as in the suffix array. The last character in each row will be exactly the (cyclic) predecessor of the starting character of the corresponding suffix. Using this observation we can derive the $BW(T)$ from the suffix array of T by just following the pointers to the suffixes and outputing the predecessors of the addressed characters. The procedure will take $O(n)$ time , where n is the size of the given string.

Exercise 3

Construct a finite automaton that enters an accepting state at every occurrence of the strings `mama` or `papa`.

[2]The most important application of the BW tranform is text compression. It turns out that T can be reconstructed from $BW(T)$. Moreover, $BW(T)$ contains a lot of repeated characters and is easy to compress using simple compression methods. For details refer to P. Ferragina and G. Manzini, An experimental study of a compressed index, Information Sciences, 135 (2001) 13–28.

Solution
We construct an automaton with two main "chains": One to recognize "papa", the other to recognize "mama". We take care that, whenever Mum appeared partially, we are still ready to receive Dad. We choose

- the set of states $Q = \{S, M1, P1, M2, P2, M3, P3, M4, P4\}$
- the start state $q_0 = S$
- the set of final states $F = \{M4, P4\}$
- the input alphabet $\Sigma = \{\text{a,b,c,}\ldots\text{,z}\}$

The transition function $d : Q \times \Sigma \to Q$ is the following:

$d(S, m) = M1$;	$d(M1, a) = M2$;	$d(M2, m) = M3$;	$d(M3, a) = M4$;
$d(M1, m) = M1$;	$d(M3, m) = M1$;		
$d(S, p) = P1$;	$d(P1, a) = P2$;	$d(P2, p) = P3$;	$d(P3, a) = P4$;
$d(P1, p) = P1$;	$d(P3, p) = P1$;		
$d(M1, p) = P1$;	$d(M2, p) = P1$;	$d(M3, p) = P1$;	
$d(P1, m) = M1$;	$d(P2, m) = M1$;	$d(P3, m) = M1$;	
$d(M4, m) = M3$;	$d(P4, p) = P3$;		
$d(q, c) = S$	in all other cases		

Exercise 4
Give the Morris-Pratt failure function for the pattern `ananas`.

Solution
If F is the Morris-Pratt failure function of a pattern, then $F[i] = j$ means: The j characters preceding the position i match the pattern start.

i	0	1	2	3	4	5	6
P	a	n	a	n	a	s	
F	0	0	0	1	2	3	0