—— SOLUTIONS ——

King's College London

This paper is part of an examination of the College counting towards the award of a degree. Examinations are governed by the College Regulations under the authority of the Academic Board.

Enter your candidate number in the box provided above and on the answer book(s) provided. Do this now.

MSc / MSci EXAMINATION

7CCSMTSP/CSMTSP – TEXT SEARCHING AND PROCESSING

MAY 2008

TIME ALLOWED: TWO HOURS.

ANSWER **THREE** OF THE **FIVE** QUESTIONS.

NO CREDIT WILL BE GIVEN FOR ATTEMPTING ANY FURTHER QUESTIONS.

ALL QUESTIONS CARRY EQUAL MARKS.

THE USE OF ELECTRONIC CALCULATORS IS **NOT** PERMITTED.

BOOKS, NOTES OR OTHER WRITTEN MATERIAL MAY **NOT** BE BROUGHT INTO THIS EXAMINATION.

NOT TO BE REMOVED FROM THE EXAMINATION HALL

TURN OVER WHEN INSTRUCTED

1. String overlaps

The overlap between two strings y and x, denoted by $\text{ov}(y, x)$, is the longest suffix of y that is also a prefix of x.

a. Choose a string-matching algorithm and describe how it can be used to compute $\text{ov}(y, x)$. What is the running time of the computation corresponding to your choice?

[15 marks]

Answer

Any sequential string-matching algorithm searching y for x can be used.

Use of MP or KMP algorithm: stop the search when the algorithm has just read the last letter of y. Then, if i is the index on x at that time, $\text{ov}(y, x) = x[0\,..\,i-1]$.

Running time: $O(|x| + |y|)$.

Use of the string-matching automaton of x: parse y with the automaton up to the last symbol. The corresponding state of the automaton is associated with a prefix of x that is precisely $\text{ov}(y, x)$.

Running time: $O(|x|\times \text{card}A + |y|)$ if the complete automaton is used; $O(|x|+|y|)$ if an implementation by failure link or default state of the automaton is used.

[unseen]

b. Describe how to compute $\text{ov}(y, x)$ in time $O(\min\{|y|, |x|\})$ using only $O(\min\{|y|, |x|\})$ extra memory space (space in addition to the space used for the input).

[10 marks]

Answer

If $|x| \leq |y|$, the processing (or parsing) of y can start at position $|y| - |x|$ on y because the overlap, as a suffix of y, does not start before this position. Doing so, the running time to preprocess x is $O(|x|)$ as is the time to process the suffix of y; giving $O(|x|)$, that is, $O(\min\{|y|, |x|\})$ running time. Extra space (for the border table or the automaton implemented with failure links or default state) is $O(|x|)$, that is, $O(\min\{|y|, |x|\})$ memory space.

If $|x| > |y|$, only the prefix of x of length $|y|$ needs to be preprocessed because the overlap cannot be longer. Doing so the running time to preprocess the prefix of x is $O(|y|)$ as is the time to process y; giving $O(|y|)$, that is, $O(\min\{|y|, |x|\})$ running time again.

Extra space (for the border table or the automaton implemented with failure links or default state of the prefix of x) is $O(|y|)$, that is, $O(\min\{|y|, |x|\})$ memory space again.

[unseen]

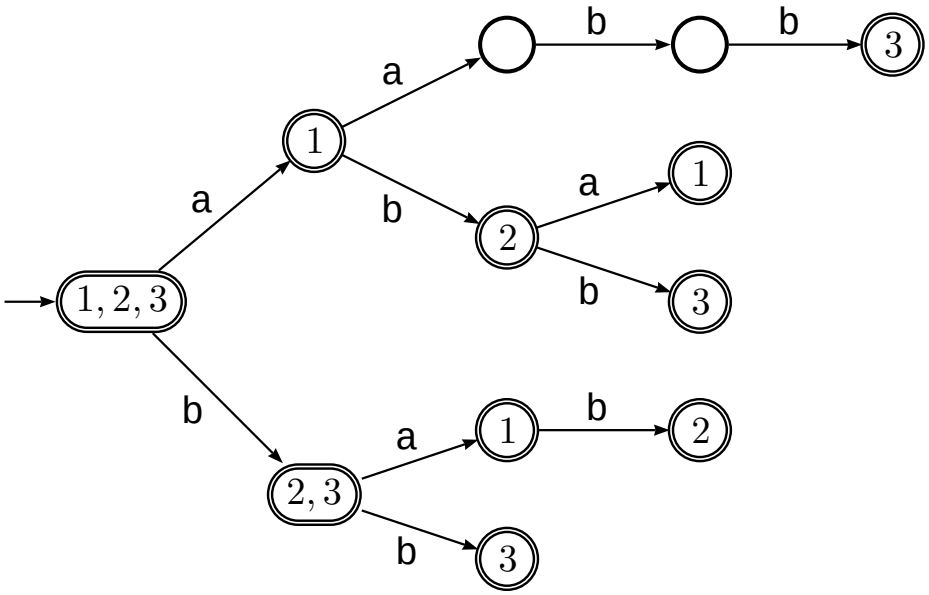
SEE NEXT PAGE

c. Design the common suffix tree of the strings $x_1 = \mathsf{aba}$, $x_2 = \mathsf{bab}$, and $x_3 = \mathsf{aabb}$.

Sketch how to build the common suffix tree of a finite set of nonempty strings $X = \{x_1, x_2, \ldots, x_k\}$, $k > 0$, in which each terminal state is labelled by the set of indices of strings having a suffix associated with the state.

[15 marks]

Answer



To build the common suffix tree, we first build the suffix tree of x_1. Then, each string is inserted in the current tree in a similar process as for the initial construction of the tree.

When a state p is met having a suffix link to a terminal state q, the set of string indices of q is added to the one of p.

[unseen]

d. Given a finite set of nonempty strings $X = \{x_1, x_2, \ldots, x_k\}$, $k > 0$, describe how to compute the overlaps between all pairs (x_i, x_j), $i \neq j$, using the common suffix tree of the strings in X.

Let $N = \Sigma_{i=1}^{k} |x_i|$. What is the running time of your procedure?

[10 marks]

Answer

Build the common suffix tree of the strings in X; running time: $O(N \times \log a)$ where a is the size of the alphabet.

For each string x_j, follow the branch of the tree labelled by x_j (from leaf to root to avoid branching). If a terminal state is met during the traversal, for each index i of its set, $i \neq j$, update $ov(x_i, x_j)$ to the length of the prefix of x_i corresponding to the state. The latest update of $ov(x_i, x_j)$ gives the correct value. Update time is $O(k)$. The running time of this step is $O(k \times N)$.

The total running time is then $O(N \times (k + \log a))$.

[unseen]

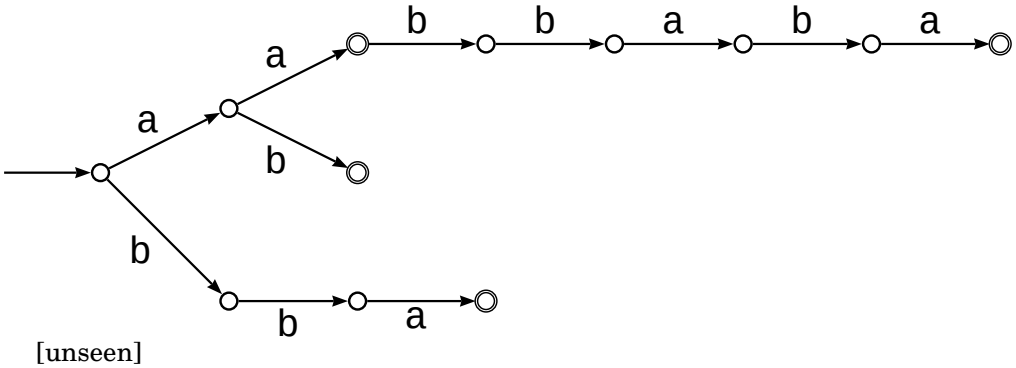
2. Dictionary-Matching Automaton

Let Σ be the alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and X be a finite set of strings of Σ^*. The dictionary-matching automaton of X over Σ is denoted by $\mathcal{D}(X)$.

a. Draw the trie of the set $(\{\mathtt{aabbaba}, \mathtt{aa}, \mathtt{ab}, \mathtt{bba}\})$. Mark its terminal states.

[5 marks]

Answer



[unseen]

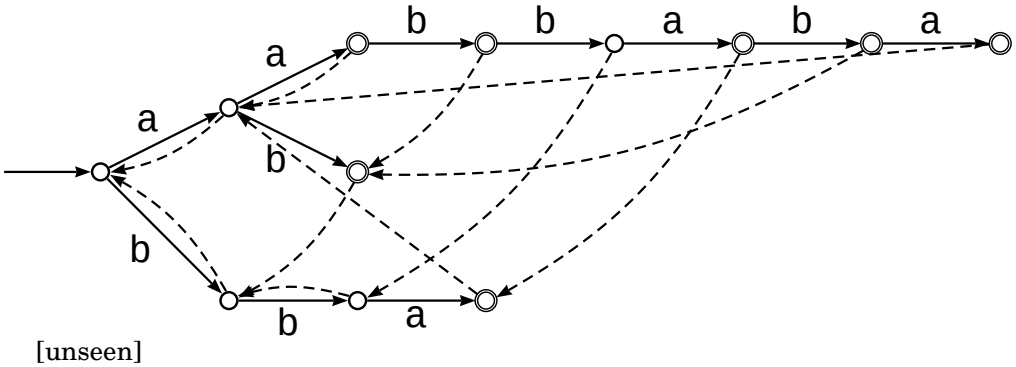
b. Define the notion of a failure link on a state (node) of the trie of X. Draw the implementation with failure links of $\mathcal{D}(\{\mathtt{aabbaba}, \mathtt{aa}, \mathtt{ab}, \mathtt{bba}\})$. Mark its terminal states.

[10 marks]

Answer

Let p be a state of the trie of X distinct from the root. Let $u \in \Sigma^+$, be the label of the path from the root to state p. Then the failure state $f(p)$ of state p is the state of the trie whose path from the root is labelled by the longest possible proper suffix of u.

[bookwork]



[unseen]

c. Describe in pseudo-code the next-state function for the implementation with failure links of $\mathcal{D}(X)$.

[10 marks]

Answer

NextState(p, a)
    if there is an edge (p, a, q) in the trie
        return q
    else if $f(p)$ is defined
        return NextState$(f(p), a)$
    else return the root of the trie

[bookwork]

d. What data structure would you use to implement a state of the dictionary-matching automaton?

[10 marks]

Answer

For each node in the automaton, one can use a structure comprising two pointers, one for the failure link and one for the list of next nodes defined by the transition function, a boolean field to mark terminal states, and possibly a field for storing some data associated with the state (for example, the letter labelling the incoming edge).

[unseen]

e. Describe in your own words or in pseudo-code how the failure links of the implementation of a dictionary-matching automaton can be computed. What are the terminal states of the automaton?

[15 marks]

Answer

The computation is done top-down in a width-first traversal of the trie. The root has no failure link. The failure target of the children of the root is the root.

Assume that we want to compute the failure link of node q (the failure links of states at a lower level have already been computed). Also assume that the parent of q is the node p and that the arc connecting the nodes p and q is labelled by the letter a. Then, the failure link $f(q)$ of the node q is given by $f(q) =$ NextState$(f(p), a)$ if $f(p)$ is defined (where NextState is the function described in 2.c), otherwise $f(q)$ is the root.

The set of terminal states comprises the terminal states of the trie and states q for which $f(q)$ is a terminal state.

[bookwork]

3. Lexicographically maximal suffix

Given a fixed word $x = x[0\,..\,m-1]$, for each integer i, $0 \le i < m$, we denote by s_i the position of the lexicographically maximal suffix of $x[0\,..\,i]$ (prefix of length $i+1$ of x), by p_i the smallest period of $x[s_i\,..\,i]$, and by r_i the remainder $(i-s_i+1)\,\%\,p_i$. (Note that $(i-s_i+1)$ is the length of $x[s_i\,..\,i]$, $i-s_i+1 = q\times p_i+r_i$ for some positive integer q, and $0 \le r_i < p_i$.)

a. Assuming that `a` $<$ `b`, give the values of s_i, p_i, and r_i corresponding to the word `ababaababaababbb` for all integer $i \in [0, 15]$

[10 marks]

Answer

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x	a	b	a	b	a	a	b	a	b	a	a	b	a	b	b	b
s_i	0	1	1	1	1	1	1	1	1	1	1	1	1	1	13	13
p_i	1	1	2	2	2	5	5	5	5	5	5	5	5	5	1	1
r_i	0	0	0	1	0	0	1	2	3	4	0	1	2	3	0	0

[unseen]

b. Let $i \in [1, m-1]$.

When $x[i] = x[s_{i-1} + r_{i-1}]$, what are the values of s_i, p_i, and r_i as expressions of s_{i-1}, p_{i-1}, and r_{i-1}?

[5 marks]

When $x[i] < x[s_{i-1} + r_{i-1}]$, show that $s_i = s_{i-1}$, $p_i = i - s_{i-1} + 1$, and $r_i = 0$.

[5 marks]

When $x[i] > x[s_{i-1} + r_{i-1}]$, show that $s_i \ge i - r_{i-1}$.

[5 marks]

Answer

By definition of p_{i-1} and r_{i-1}, the current maximal suffix $x[s_{i-1}\,..\,i-1]$ is of the form $u^q v$ for two words u and v satisfying: $|u| = p_{i-1}$, $|v| = r_{i-1}$, and v is a prefix of u. The positions s_{i-1} stops to be the position of the current maximal suffix only if the suffix starting at position $x[i-r_{i-1}]$ becomes lexicographically larger or smaller than $x[s_{i-1}\,..\,i-1]$. Comparing the letters $x[i]$ and $x[s_{i-1} + r_{i-1}]$ gives the answer.

When $x[i] = x[s_{i-1} + r_{i-1}]$, the periodicity continues; then, $s_i = s_{i-1}$, $p_i = p_{i-1}$, and $r_i = r_{i-1} + 1$ if $r_{i-1} + 1 < p_{i-1}$ or $r_i = 0$ otherwise.

When $x[i] < x[s_{i-1} + r_{i-1}]$, all the suffixes of $x[0\,..\,i]$ different from $x[s_{i-1}\,..\,i]$ are smaller than it without being a prefix of it. In particular, $x[s_{i-1}\,..\,i]$ has no nonempty border, which is equivalent to say that that its length is its smallest period: $p_i = i - s_{i-1} + 1$. We also have $s_i = s_{i-1}$ and $r_i = 0$.

When $x[i] > x[s_{i-1} + r_{i-1}]$, the suffix $x[i - r_{i-1}\,..\,i]$ is larger than $x[s_{i-1}\,..\,i]$ and than all suffixes of $x[0\,..\,i]$ starting at any position j, $j < i - r_{i-1}$. Therefore $s_i \ge i - r_{i-1}$.

[unseen]

SEE NEXT PAGE

c. Based on the three properties stated in Question 3.b, describe in pseudo-code an algorithm for computing the maximal suffix of the word x, using only a constant memory space in addition to the array used to store x.

[15 marks]

Answer

MaximalSuffix(x, m)
$s \leftarrow 0$; $p \leftarrow 1$; $r \leftarrow 0$; $i \leftarrow 1$
while $i < m$ do
    if $x[i] = x[s + r]$ then
        $r \leftarrow$ if $r = p$ then 0 else $r + 1$; $i \leftarrow i + 1$
    elseif $x[i] < x[s + r]$ then
        $p \leftarrow (i - s + 1)$; $r \leftarrow 0$; $i \leftarrow i + 1$
    else $x[i] > x[s + r]$ then
        $s \leftarrow i - r$; $p \leftarrow 1$; $r \leftarrow 0$; $i \leftarrow s + 1$
return s

[unseen]

d. Analyse the running time of the algorithm you described in Question 3.c. Justify your answer.

[10 marks]

Answer

The algorithm is in the answer of Question 3.c.

The value of the expression $s + i$ strictly increases during each step of the while loop: this is obvious if one of the first two conditions holds because the value of s is unchanged while i is incremented by 1; this is also true when the third condition holds because s is incremented by at least p, i is decremented by $r - 1$, and $r < p$.

Therefore, since the initial value of $s+i$ is 1 and the final value is smaller than $2m$, the algorithm runs in time $O(m)$.

[unseen]

4. Doubling

Let y be a fixed text of length n.

For a word u and a positive integer k, $First_k(u)$ is u if $|u| \leq k$ and is $u[0\,..\,k-1]$ otherwise. The integer $R_k[i]$ is the rank of $First_k(y[i\,..\,n-1])$ inside the sorted list of all $First_k(u)$ where u is a nonempty suffix of y (ranks are numbered from 0).

a. Give R_1, R_2, R_3, R_4, R_8 for the word `aababbabba`, assuming `a` $<$ `b`.

[10 marks]

Answer

i	0	1	2	3	4	5	6	7	8	9
$y[i]$	a	a	b	a	b	b	a	b	b	a
$R_1[i]$	0	0	1	0	1	1	0	1	1	0
$R_2[i]$	1	2	3	2	4	3	2	4	3	0
$R_3[i]$	1	2	5	3	6	5	3	6	4	0
$R_4[i]$	1	2	5	3	7	5	3	6	4	0
$R_8[i]$	1	2	7	4	9	6	3	8	5	0

[unseen]

b. State the doubling lemma and prove it.

[15 marks]

Answer

Lemma 1 *$Rank_{2k}[i]$ is the rank of the pair $(Rank_k[i], Rank_k[i+k])$ in the sorted list of these pairs.*

Proof. Let i be a position on y and let $u = First_{2k}(y[i\,..\,n-1])$. Let j be a position on y and let $v = First_{2k}(y[j\,..\,n-1])$. We show that $u \leq v$, which is equivalent to $Rank_{2k}[i] \leq Rank_{2k}[j]$, iff $(Rank_k[i], Rank_k[i+k]) \leq (Rank_k[j], Rank_k[j+k])$.

First case: $First_k(u) < First_k(v)$. This is equivalent to $Rank_k[i] < Rank_k[j]$ so the result holds in this case.

Second case: $First_k(u) = First_k(v)$. This is equivalent to $Rank_k[i] = Rank_k[j]$. Then the comparison between u and v depends only on the second halves of these words; in other terms, $Rank_{2k}[i] \leq Rank_{2k}[j]$ is equivalent to $Rank_k[i+k] \leq Rank_k[j+k]$.

[bookwork]

c. Describe an efficient algorithm to compute R_{2k} from R_k. What is its running time?

[15 marks]

Answer

Two steps: first sort positions i according to the pairs $(R_k[i], R_k[i+k])$; then assign the same R_{2k} rank to positions associated with the same pair.

First step can be implemented by bucket sort (count sort) in linear time; second step is obvious and runs also in linear time.

[bookwork]

d. Analyse the running time of the algorithm based on Question 4.c to compute R_1, R_2, R_4, ..., R_{2k}, where k is the smallest integer satisfying the inequality $n \leq 2^k$. Justify your answer.

[10 marks]

Answer

It is $O(n \times \log n)$ because there are $\lceil \log n \rceil$ steps and each step can be implemented to run in $O(n)$ from answer to Question 4.c.

[bookwork]

5. Linear-time suffix sorting

Let y be a string of length n.

a. List the nonempty suffixes of the string abaabba in lexicographic order assuming a $<$ b.

[5 marks]

Answer

a,aabba,abaabba,abba, ba,baabba,bba.

[unseen]

b. Let P_{01} be the positions on y of the form $3q$ or $3q+1$. Let P_2 be the positions on y of the form $3q+2$. Describe the four steps of the Skew algorithm to sort the suffixes of y.

[20 marks]

Answer

1. Sort the position in P_{01} according to their associated 3-grams. Let $t[i]$ be the rank of i in the sorted list.
2. Recursively sort the suffixes of $t[0]t[3]\ldots t[1]t[4]\ldots$. For a position i in P_{01}, let $s[i]$ be the rank of its associated suffix in the sorted list of them, L_{01}.
3. Sort the positions j in P_2. Let L_2 be the sorted list.
4. Merge lists L_{01} and L_2.

[in lectures, 5 marks for each step]

c. Let L_{01} be the list of positions of P_{01} sorted according to their associated suffixes; let $s[i]$ be the rank of i in L_{01}. Describe how to sort P_2 in time $O(|P_2|)$.

[10 marks]

Answer

Sorting elements j of P_2 remains to sort their associated pairs $(y[j], s[j+1])$. This can be done in linear time using radix sort.

[in lectures]

d. In addition to L_{01} and the table s in Question 5.c, let L_2 be the list of positions of P_2 sorted according to their associated suffixes. Describe how to compare i in L_{01} with j in L_2 in constant time.

[15 marks]

Answer

If i is of the form $3q$, $i+1$ and $j+1$ are both in L_{01}, and then $s[i+1]$ and $s[j+1]$ are both defined. Therefore, comparing i and j amounts to compare $(y[i], s[i+1])$ and $(y[j], s[j+1])$.

If i is of the form $3q+1$, $i+2$ and $j+2$ are both in L_{01}, and then $s[i+2]$ and $s[j+2]$ are both defined. Therefore, comparing i and j amounts to compare $(y[i]y[i+1], s[i+2])$ and $(y[j]y[j+1], s[j+2])$.

[in lectures, 10 marks]

In both cases the comparison is done in constant time.

[in lectures, 5 marks]