SOLUTIONS

King's College London

UNIVERSITY OF 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 EXAMINATION

CSMTSP – TEXT SEARCHING AND PROCESSING

MAY 2004

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

—— SOLUTIONS ——

1. Borders and prefixes

Given a word $x = x[0\,..\,m-1]$, the table $Border$ is defined by: $Border[0] = -1$, and $Border[j] =$ the maximal length of (proper) borders of $x[0\,..\,j-1]$ for $0 < j \leq m$.
For the same word, the table $Prefix$ is defined by: $Prefix[i] = lcp(x, x[i\,..\,m-1]) =$ maximal length of prefixes common to x and $x[i\,..\,m-1]$, for $0 \leq i < m$.

a. Give the tables $Border$ and $Prefix$ associated with the word abaababaabaab.

[10 marks]

Answer

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$x[i]$	a	b	a	a	b	a	b	a	a	b	a	a	b	$-$
$Border[i]$	-1	0	0	1	1	2	3	2	3	4	5	6	4	5
$Prefix[i]$	12	0	1	3	0	6	0	1	5	0	1	2	0	$-$

b. Design a string-matching algorithm using the table $Border$. State and prove its running time.

[20 marks]

Answer

```
MP(string x, y; integer m, n)
    i ⟵ 0; j ⟵ 0
    while j < n do
        while (i = m) or (i ≥ 0 and x[i] ≠ y[j]) do
            i ⟵ Border[i]
        i ⟵ i + 1; j ⟵ j + 1
        if i = m then output('x occurs in y at position', j − i)
```

The algorithm runs in time $O(n)$ because each positive letter comparison strictly increases the value of j that goes from 0 to n without ever decreasing, and each negative letter comparison strictly increases the value of $j - i$ that goes from 0 to at most n without ever decreasing.

c. Let j be a position on the word x, $0 < j \leq m$. Let $i = j - Border[j]$. What can you say about $Prefix[i]$? Does the equality $Prefix[i] = Border[j]$ always hold?

[5 marks]

Answer

The word $x[i\,..\,j-1]$ is a prefix of x, then $Prefix[i] \geq j - i$. The equality $Prefix[i] = Border[j]$ does not always hold.

Counterexample: Take the word $abaababaabaab$, for which tables $Border$ and $Prefix$ are computed in question 1.a. We have

$$j = 5 \Rightarrow Border[j] = 2 \Rightarrow i = j - Border[j] = 3$$

But $Prefix[i] = 3 \neq 2 = Border[j]$.

SEE NEXT PAGE

d. Let i be a position on the word x, $0 \leq i < m$. Let $j = i + \mathit{Prefix}[i]$. What can you say about $\mathit{Border}[j]$? Does the equality $\mathit{Border}[j] = \mathit{Prefix}[i]$ always hold?

[5 marks]

Answer

The word $x[i \,..\, j-1]$ is a prefix of x, then a border of $x[0 \,..\, j-1]$, thus $\mathit{Border}[j] \geq j - i$. The equality $\mathit{Border}[j] = \mathit{Prefix}[i]$ does not always hold.

Counterexample: Take the word $abaababaabaab$, for which tables $Border$ and $Prefix$ are computed in question 1.a. We have

$$i = 4 \Rightarrow Prefix[i] = 0 \Rightarrow j = i + Prefix[i] = 4$$

But $Prefix[i] = 0 \neq 1 = Border[j]$.

e. A square is a word of the form uu where u is a non-empty word. Indicate how all squares that are prefixes of x can be found using the table Border.

[5 marks]

Answer

$x[0 \,..\, j-1]$ is a square if and only if $j/(j - \mathit{Border}[j])$ is an even integer, which implies that j should be even.

f. Indicate how all squares that are prefixes of x can be found using the table Prefix.

[5 marks]

Answer

$x[0 \,..\, 2i-1]$ is a square if and only if $\mathit{Prefix}[i] \geq i$.

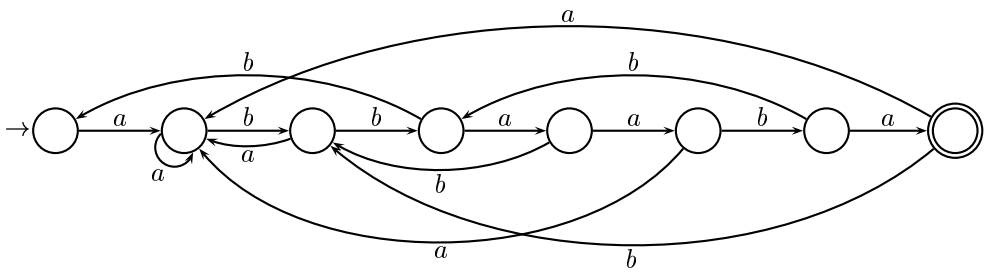
2. String-Maching Automaton

Let Σ be the alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. For $x \in \Sigma^*$, the string-matching automaton of x, *SMA*(x), is the minimal deterministic automaton accepting the language $\Sigma^* x$.

a. Design the automaton $SMA(\mathtt{abbaaba})$.

[10 marks]

Answer



b. Describe in pseudo-language how to transform *SMA*(x) to get *SMA*(xa) for a word x and a letter a.

[20 marks]

Answer

In the algorithm below assume that the SMA x is described by the 5-tuple

$$(Q, \Sigma, initial, \{terminal\}, \delta)$$

UPDATESMA$(SMA\ x, char\ a)$
1 $r \leftarrow \delta(terminal, a)$
2 add new state s to Q
3 $\delta(terminal, a) \leftarrow s$
4 **for** all $\sigma \in \Sigma$
5 **do** $\delta(s, \sigma) \leftarrow \delta(r, \sigma)$
6 $terminal \leftarrow s$
7 **return** $(Q, \Sigma, initial, \{terminal\}, \delta)$

c. Define the notion of a backward arc in *SMA*(x) and state the bounds on the number of them.

[10 marks]

Answer

States being prefixes of x, a backward arc is of the form (u, a, va) where u and va are prefixes of x, ua is not a prefix of x, and v is the longest suffix of u having this property. The number of backward arcs is between 0 and $|x|$.

d. State the time and space complexities of searching y for x with $SMA(x)$ both when the complete automaton is used with a transition table, and when only significant arcs are implemented.

[10 marks]

Answer

		Complete	Only Significant
Preprocessing on pattern	time	$O(m \times card\Sigma)$	$O(m)$
	space	$O(m \times card\Sigma)$	$O(m)$
Search on text	time	$O(n)$	$O(n)$
	space	$O(m \times card\Sigma)$	$O(m)$

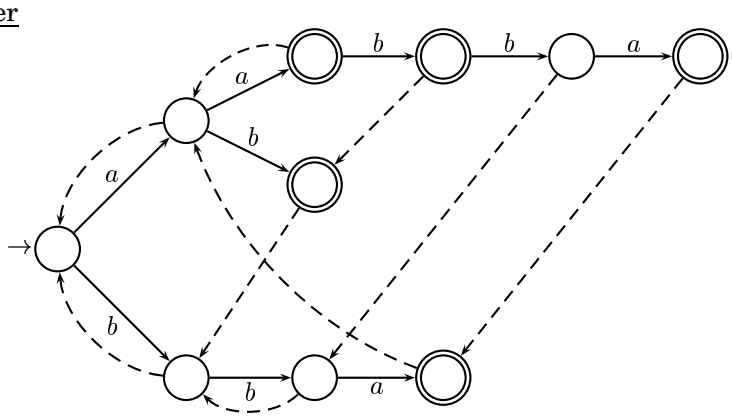
3. Dictionary-Maching Automaton

Let Σ be the alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. The dictionary-matching automaton of a finite set X of words over Σ is denoted $DMA(X)$.

a. Design $DMA(\{\mathtt{aabba}, \mathtt{aa}, \mathtt{ab}, \mathtt{bba}\})$, including the failure links defined on states of the automaton.

[15 marks]

Answer



b. Design in pseudo-code the next-state function based on failure links and used during the search of a text for a word of the set X.

[15 marks]

Answer

```
NEXTSTATE(p, δ)
1  if g(p, δ) is defined
2     then return g(p, δ)
3     else  if f(p) is defined
4              then return NEXTSTATE(f(p), δ)
5              else  return q_0
```

c. How would you implement a node of the DMA automaton?

[10 marks]

Answer

For each node in the automaton, one needs two pointers (one for the failure link and one for the list of next nodes defined by the transition function) and possibly one variable for storing the data (a symbol of the alphabet for example).

d. Describe in your own words or in pseudo-code how the failure link of a node can be computed.

[10 marks]

Answer

Assume that we want to compute the failure link of node q. Also assume that parent of q is the node p and that the arc connecting the nodes p and q is labelled by the character τ. Then the failure link $f(q)$ of the node q is given by

$$f(q) = \text{NEXTSTATE}(f(p), \tau)$$

if $f(p)$ is defined, otherwise $f(q) = q_0$, where q_0 is the initial state and $\text{NEXTSTATE}(p, \delta)$ is the function described in 3.b. The initial state has no failure link.

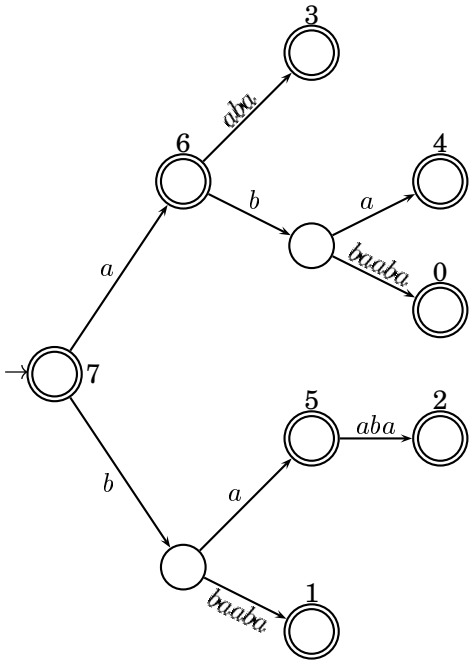
Also, note that if $f(q)$ is final, q has to be set to terminal, too.

4. Suffix Trie and Suffix Tree

a. Design the trie of suffixes of the word aabbabbabab.

[15 marks]

Answer



b. Give an example of a word of length m on the alphabet $\{\mathtt{a}, \mathtt{b}\}$ having a trie of suffixes of size $\Omega(m^2)$.

[10 marks]

Answer

$\mathtt{a}^{m/4}\mathtt{b}^{m/4}\mathtt{a}^{m/4}\mathtt{b}^{m/4}$.

c. Design an algorithm to compact the suffix trie of a word into its suffix tree.

[15 marks]

Answer

```
COMPACT(trie T)
 1  r ← root of T
 2  for each arc (r, a, p)
 3  do COMPACT(subtree of T rooted at p)
 4     if p has exactly one child
 5        then q ← that child
 6             u ← label of (p, q)
 7             replace p by q as child of r
 8             set a · u as label of (r, q)
```

d. Describe possible data structures required to implement the suffix tree of a word.

[10 marks]

Answer

For each node of the suffix tree, one needs two pointers (one for the failure link and one for the list of next nodes defined by the transition function) and possibly one variable for storing the data (a symbol of the alphabet for example).

5. Doubling

Let y be a fixed text of length n. Recall that, for a word u, $First_k(u)$ is u if $|u| \leq k$ and is $u[0..k-1]$ otherwise. Recall also that $R_k[i]$ is the rank of $First_k(y[i..n-1])$ inside the ordered list of all $First_k(u)$, u non-empty 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	10
$y[i]$	a	b	b	a	a	b	a	b	b	a	b
$R_1[i]$	0	1	1	0	0	1	0	1	1	0	1
$R_2[i]$	1	4	3	0	1	3	1	4	3	1	2
$R_3[i]$	3	7	5	0	2	6	3	7	6	1	4
$R_4[i]$	3	8	5	0	2	7	3	9	6	1	4
$R_8[i]$	3	9	6	0	2	8	4	10	7	1	5

b. State the doubling lemma and prove it.

[15 marks]

Answer

Lemma 1 *$Rank_{2k}[i]$ is the rank of $(Rank_k[i], Rank_k[i+k])$ in the ordered 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 $v = First_{2k}(y[i\,..\,n-1])$. 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: $First_k(u) < First_k(v)$ is equivalent to $Rank_k[i] < Rank_k[j]$ so the result holds in this case. Second case: $First_k(u) = First_k(v)$ 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 $Rank_k[i+k] \leq Rank_k[j+k]$.

c. Design an efficient algorithm to compute R_{2k} from R_k.

[15 marks]

Answer

Two steps: first sort positions i according to the pairs $(R_k[i], R_k[i+k])$; then the same R_{2k} assign rank to positions associated with the same pair. First step can be implemented by bucket sort in linear time, second step is obvious and run also in linear time.

d. State the complexity of the algorithm based on Question 5.c to compute R_1, R_2, R_4, ..., R_{2k}, where k is the smallest integer satisfying the inequality $n \leq 2^k$. Explain 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 $O(n)$ time using bin sorting.