—— SOLUTIONS ——

King's College London

UNIVERSITY OF LONDON

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 2006

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. Borders and overlaps

Given a word $u = u[0 \,..\, m-1]$, its table *Border* is defined by: $Border[0] = -1$, and $Border[j] =$ the maximal length of (proper) borders of $u[0 \,..\, j-1]$ for $0 < j \leq m$.

a. Give the table *Border* associated with the word abaabaaabaa.

[10 marks]

Answer

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$x[i]$	a	b	a	a	b	a	a	a	b	a	a				
$Border[i]$	-1	0	0	1	1	2	3	4	1	2	3	4			

b. Let x and y be two strings on the alphabet $\{\mathtt{a}, \mathtt{b}\}$. Show how to find the positions of the occurrences of x in y using the table *Border* associated with the string $x\mathtt{c}y$ (c is a letter different from a and b).

[10 marks]

Answer

Note that $Border[i] \leq |x|$ because the letter c does not appear in y. If a positions i on y is such that $Border[i] = |x|$ then x is a suffix of $x\mathtt{c}y[0 \,..\, i]$ and then of $y[0 \,..\, i]$. So, x occurs in y at position $i - |x| + 1$. [5 marks]
Conversely, if x occurs in y at position j, then x is a border of $x\mathtt{c}y[0 \,..\, i]$ for $i = j + |x| - 1$ because the border cannot be longer than x. Thus $Border[i] = |x|$. [5 marks]
Conclusion: condition $Border[i] = |x|$ can be used to signal occurrences of x in y.

c. The overlap between x and y, $ov(x, y)$, is the longest word that is both a prefix of x and a suffix of y. How would you find $ov(x, y)$ using the table *Border* associated with the string $x\mathtt{c}y$? How would you do it using the table *Border* associated with the string x?

[15 marks]

Answer

Let $k = Border[|x| + |y| + 2]$, then $ov(x, y) = x[0 \,..\, k-1]$. [5 marks]
With the table *Border* associated with the string x, apply MP algorithm until $j = |y| + 1$; then $ov(x, y) = x[0 \,..\, i-1]$. [10 marks]

d. Design in pseudo-code an algorithm that computes the table $Border$ of the word x. State and prove its running time.

[15 marks]

Answer

```
COMPUTE_BORDERS(string x; integer m)
1  Border[0] ← −1
2  for i ← 0 to m − 1
3  do j ← Border[i]
4     while j ≥ 0 and x[i] ≠ x[j]
5     do j ← Border[j]
6     Border[i + 1] ← j + 1
7  return Border
```

[8 marks]

The algorithm runs in time $O(m)$. [3 marks]

Indeed, the running time is proportional to the number of symbol comparisons done at Line 4. Each positive comparison leads to an increment of variable i which values are in increasing order. Each negative comparison leads to an increment of expression $i - j$ which values are in increasing order. So, there are at most $2m$ comparisons, which proves the result. [4 marks]

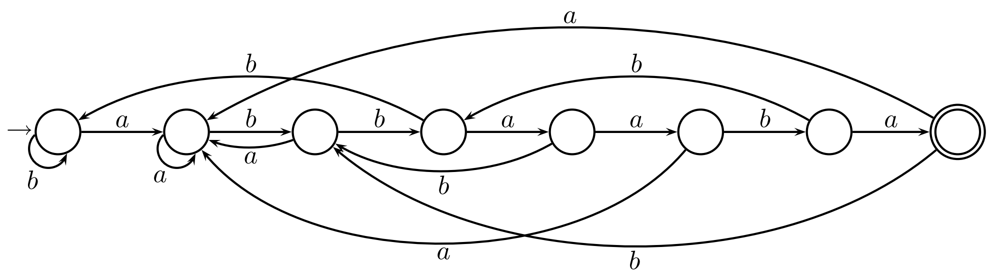
2. String-Matching Automaton

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



All arcs labelled with c lead to the initial state.

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)
r ← δ(terminal, a)
add new state s to Q
δ(terminal, a) ← s
for all σ ∈ Σ
do δ(s, σ) ← δ(r, σ)
terminal ← s
return (Q, Σ, initial, {terminal}, δ)
```

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. Searching a list of strings Consider a list of strings $L = (y_1, y_2, \dots, y_k)$ in lexicographic order: $y_1 \leq y_2 \leq \dots \leq y_k$. Let x be another string that is to be found in the list L. All strings x and y's have the same length.

a. What is the asymptotic running time of a binary search for x in L if no extra information on the strings y's is known? Give a "worst-case" example to your answer.

[15 marks]

Answer

The asymptotic cost of a binary search for the string x of length n in the list L of k lexicographically sorted strings y_i is $O(n \log k)$ time. A "worst-case" example could be the search for $x = bbb \dots b$ in the list $L = (aaa \dots a, aaa \dots b, aaa \dots bb, aaa \dots bbb, \dots, bbb \dots b)$

b. For two strings u and v, $lcp(u, v)$ denotes the maximum length of their common prefixes. Let $\ell = lcp(x, y_1)$, $r = lcp(x, y_k)$, and $i = \lfloor (k+1)/2 \rfloor$. Assume that $y_1 \leq x \leq y_k$ and $\ell > r$. How does x compare with y_i when $\ell < lcp(y_1, y_i)$ and $\ell > lcp(y_1, y_i)$ respectively?

[15 marks]

Answer

Assume that $l > r$ and that $l < lcp(y_1, y_i)$.
Then let $u = y_1[1 \dots l]$, $\sigma = y_1[l+1]$, $\tau = y_i[l+1]$. Then $u\tau$ is a prefix of x and $\sigma < \tau$. This implies that $y_i < x < y_k$. Now assume that $l > r$ and $l > lcp(y_1, y_i)$.
In this case,we have that $\sigma \neq \tau$ and $\sigma < \tau$ which implies that $y_1 < x < y_i$.

c. State the cost of the binary search algorithm based on the use of longest common prefixes of (non-empty) suffixes of y's.

[10 marks]

Answer

Running a binary search for a string x of length m by using the longest common prefixes of the n sorted suffixes of y takes $O(m + \log n)$ time.

d. How many longest common prefixes of suffixes of y need to be preprocessed to run the binary search of 3.c?

[10 marks]

Answer

The binary tree has $n + 1$ leaves and n internal nodes. Since we need an lcp value for each node, $2n + 1$ values need to be preporcessed before running the binary search algorithm.

4. Doubling

Let y be a fixed text of length n. Recall that, for a word u, $First_k(u)$ is u if $|u| \le k$ 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
$y[i]$	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

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 $u \le v$, which is equivalent to $Rank_{2k}[i] \le Rank_{2k}[j]$, iff $(Rank_k[i], Rank_k[i+k]) \le (Rank_k[j], Rank_k[j+k])$. 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, $Rank_{2k}[i] \le Rank_{2k}[j]$ is equivalent to $Rank_k[i+k] \le Rank_k[j+k]$.

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 $n \le 2^k$. 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.

5. Word transformation

Let $x = x[0]x[1] \cdots x[m-1]$ be a word of length m. For an integer i, $0 \le i < m$, the i-rotation of x is the word $x[i]x[i+1] \cdots x[m-1]x[0]x[1] \cdots x[i-1]$. We assume in this question that the m rotations of x are pairwise distinct and that x is the smallest of them according to the lexicographic order.

The BW matrix of x, denoted by $BW(x)$, is the $m \times m$ matrix whose lines are the rotations of x in lexicographic order.

The BW transform of x, denoted by $L(x)$, is the last column of the BW matrix. (It is a word of length m.)

a. Give the BW matrix of the word $x = \texttt{aabbab}$. Give $T(\texttt{aabbab})$.

[5 marks]

Answer

$$BW(\texttt{aabbab}) = \begin{pmatrix} \texttt{a} & \texttt{a} & \texttt{b} & \texttt{b} & \texttt{a} & \texttt{b} \\ \texttt{a} & \texttt{b} & \texttt{a} & \texttt{a} & \texttt{b} & \texttt{b} \\ \texttt{a} & \texttt{b} & \texttt{b} & \texttt{a} & \texttt{b} & \texttt{a} \\ \texttt{b} & \texttt{a} & \texttt{a} & \texttt{b} & \texttt{b} & \texttt{a} \\ \texttt{b} & \texttt{a} & \texttt{b} & \texttt{a} & \texttt{a} & \texttt{b} \\ \texttt{b} & \texttt{b} & \texttt{a} & \texttt{b} & \texttt{a} & \texttt{a} \end{pmatrix}$$

$T(\texttt{aabbab}) = \texttt{bbaaba}$

b. How would you compute the BW matrix of x?
What would be the running time of the algorithm both if the alphabet is bounded and if it is unbounded?

[15 marks]

Answer

Rotations of x are segments of length m of the word $x' = x[1]x[2] \cdots x[m-1]x[0]x[1] \cdots x[m-1]$. Sorting the suffixes of this word gives the answer. [5 marks]
On an unbounded alphabet, suffixes can be sorted either by using the suffix tree or the suffix automaton of x', which is done in $O(m \times \log a)$ time, where a is the size of the alphabet of x. [10 marks]
On a bounded alphabet, suffixes can be sorted with the suffix array of x' which requires $O(m)$ time. [5 marks]

c. Let a be a letter and u, v be two different strings of the same length. Prove that $au < av$ if and only $ua < va$.

[10 marks]

Answer

Let w be the longest common prefix of u and v. Since $u \neq v$, w is a proper prefix of u and of v. Then, $u = wbu'$ and $v = wcv'$ for some letters b, c and some words u', v'. The condition $au < av$ is equivalent to $b < c$, because aw is the longest prefix of au and av, which is equivalent to $u < v$ and to $ua < va$, because none of u and v is a prefix of the other.

d. Let $F(x)$ be the first column of $BW(x)$. Let a be any letter occurring in x. Show that the rank of this letter among occurrences of a's in $F(x)$ is equal to its rank among occurrences of a's in $L(x)$. [Hint: use Question 5.c.]

[10 marks]

Answer

The occurrence of a in $F(x)$ is associated with a rotation au of x. Consider an occurrence of a in $F(x)$ associated with the rotation av and of higher rank among occurrences of a's in $F(x)$. By definition of $BW(x)$ we have $au < av$, and then $ua < va$ by Question 5.c. Therefore the rank of the second occurrence of a among occurrences of a's in $L(x)$ is also higher in $L(x)$. Which gives the conclusion.

e. Note that letters in $F(x)$ are in non-decreasing order and that if $L(x)[k] = x[i]$, $0 \leq i < m - 2$, then $x[i+1] = F(x)[k]$. Describe how to compute x from $L(x)$.

[10 marks]

Answer

First compute $F(x)$ by sorting the letter of $L(x)$.
The first letter of x is $L(x)[0]$.
Assume that the letter $x[i]$ has been computed. The next letter $x[i+1]$ can be computed as follows: let $L(x)[k]$ be the letter having the same rank among occurrences of $x[i]$'s in $L(x)$ as its rank among occurrences of $x[i]$'s in $F(x)$; then $x[i+1] = F(x)[k]$.