King's College London

UNIVERSITY OF LONDON

DRAFT VERSION

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 2002

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. **Aho-Corasick automaton**

a. Design the Aho-Corasick (AC) automaton over the alphabet $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ for the dictionary of strings: $\{\mathsf{abababa}, \mathsf{aaba}, \mathsf{baba}, \mathsf{bab}\}$.

[15 marks]

See last page.

b. Define and explain the data structures used to implement an AC automaton for dictionary matching.

[15 marks]

The Aho-Corasick automaton is a deterministic finite automaton together with a failure function (link) f. Data structures for nodes and arcs of an AC automaton can be defined as:

```
typedef struct ARC {
  int letter;
  struct ARC * next;
} arc;
typedef struct NODE {
  int final;
  struct NODE * f;
  arc * list;
} node;
```

An AC automaton is accessible from its initial state; its type can be declared as:

```
typedef node * automaton;
```

c. Describe in pseudo-code the search procedure for finding all the occurrences of patterns of the dictionary in a text using the implementation of 1.b.

[10 marks]

The search procedure using the AC automaton is defined by the following procedure for moving from one state to another according to the current symbol being read from the text. Let $p \in Q$ be a state of the automaton, f be the failure link and $\delta \in \Sigma$ be a symbol of the alphabet.

$Next\_State(p, \delta)$
  if (δ, q) in the list $p \rightarrow list$ **then**
    return q;
  else if $p \rightarrow f$ non NULL **then**
    return $Next\_State(p \rightarrow f, \delta)$
  else return initial state;

SEE NEXT PAGE

d. State the time and space complexity for building the AC automaton of a set of strings X, as well as the complexity of the search procedure applied to a text of length n. State the maximum time spent by the search procedure on a single symbol of the text

[10 marks]

Preprocessing the AC automaton for a set X of strings has a time and space complexity $O(|X| \log |\Sigma|)$. The time complexity for the search procedure on a text of length n is $O(n \log |\Sigma|)$. The maximum time spent on a single symbol of the text during the search procedure is $O(max\{|x| : x \in X\})$.

2. **Borders of strings**

a. Report all periods and borders of the string aabbaaabbaaabbaaa.

[10 marks]

The periods and the borders of the string aabbaaabbaaabbaaa are:
$p_0 = 5, b_0 =$ aabbaaabbaaa
$p_1 = 11, b_1 =$ aabbaaa
$p_2 = 16, b_2 =$ aa
$p_3 = 17, b_3 =$ a
$p_4 = 16, b_4 = \epsilon$

b. Prove that the border of a border of a string x is also a border of x. Let $border(x)$ be the longest (proper) border of x. Prove that a (proper) border of x is either $border(x)$ or a (proper) border of $border(x)$.

[15 marks]

Recall that a border of x is both a prefix and a suffix of x. Let b_1 be a border of x and b_2 be a border of b_1. Then, by definition, b_2 is a prefix of b_1. By transitivity of the notion of prefix, b_2 then also a prefix of x. Similarly, b_2 is a suffix of b_1,, hence a suffix of x. Thus b_2 is both a prefix and a suffix of x, which implies that b_2 is also border of x.

Let *border(x)* be the longest proper border of x. Let b be a (proper) border of x. If it is the longest border, it is *border(x)* by definition. Otherwise, b is a prefix of x shorter than its prefix *border(x)*. Thus b a (proper) prefix of *border(x)*. Similarly, it is a (proper) suffix of *border(x)*, therefore a border of it.

c. Design an algorithm that computes the lengths of borders of all non-empty prefixes of a string x.

[20 marks]

```
procedure COMPUTE_BORDERS(x : string ; m : integer);
begin
    Border(0) := -1;
    for i:= 1 to m do begin
        j:= Border(i-1);
        while j ≥ 0 and x[j] ≠ x[j + 1] do j := Border[j];
        Border(i) := j +1;
    end;
end;
```

d. Give the output of your algorithm of 2.c for aabbaaabbaaa.

[5 marks]

Output of the above algorithm for aabbaaabbaaa
Border(0 : 12) = (-1, 0, 1, 0, 0, 1, 2, 2, 3, 4, 5, 6, 7)

3. **Searching a list of strings** Consider a list of strings $L = (y_1, y_2, \ldots, y_k)$ in lexicographic order: $y_1 \leq y_2 \leq \cdots \leq y_k$. 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]

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 $x = aaa \ldots a$ in the list $L = (aaa \ldots a, aaa \ldots b, aaa \ldots bb, aaa \ldots bbb, \ldots, bbb \ldots b)$

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]

Assume that $l > r$ and that $l < lcp(y_1, y_i)$. Let $u = y_1[1 \ldots l]$, $\sigma = y_1[l + 1]$, $\tau = x[l + 1]$. 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 $l > lcp(y_1, y_i)$. Let $k = lcp(y_1, y_i)$, $u = y_1[1 \ldots k]$, $\sigma = y_1[k+1]$,i and $\tau = y_i[k + 1]$. Then $u\sigma$ is a prefix of x and $\sigma < \tau$ which implies that $y_1 < x < y_i$.

SEE NEXT PAGE

c. State the cost of the binary search algorithm based on the use of longest common prefixes of y's.

[10 marks]

Running a binary search for a string x of length n by using the longest common prefixes of the k sorted strings y's would take $O(n + \log k)$ time.

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

[10 marks]

At each step in the algorithm of the previous question, one uses three longest common prefixes, namely $lcp(x, y_1)$, $lcp(x, y_k)$ and $lcp(y_1, y_i)$. Since $i = \lfloor (k+1) \rfloor$, we will need to process $\log k$ longest common prefixes among the $y's$ and two ones on x.

4. **Approximate matching**

a. Define the k-differences approximate pattern matching problem. How would you initialize the dynamic programming matrix for this problem?

[10 marks]

The k-difference approximate pattern matching problem consists in finding all the substrings of a given text x which are at a distance less than or equal to a given value k of a pattern y. To solve this problem via dynamic programming one has to initialize all values in the first row of the DP matrix to zero and all values in the first column to 1,2,3 ... That is $DP[i, 0] \leftarrow i$ for $i = 0$ to m and $DP[0, j] \leftarrow 0$ for $j = 0$ to n.

b. Let DP be the k-differences dynamic programming matrix of two strings x and y of respective lengths m and n. Give the recurrence relation to compute $DP[i, j]$ for $0 \leq i < m$, $0 \leq j < n$ when unit costs are applied for each edit operation. Give the DP matrix for $x =$ aaababaabbab, $y =$ ababb and $k = 1$.

[15 marks]

If unit cost operations (insert, delete, substitute) are assumed then the following relation produces the desired value for $DP[i, j]$.
$DP[i, j] = DP[i-1, j-1]$ if $x_i = y_i$,
$DP[i, j] = min\{DP[i-1, j-1] + 1, DP[i-1, j] + 1, DP[i, j-1] + 1\}$ otherwise.

		0	1	2	3	4	5	6	7	8	9	10	11	12
			a	a	a	b	a	b	a	a	b	b	a	b
0		0	0	0	0	0	0	0	0	0	0	0	0	0
1	a	1	0	0	0	1	0	1	0	0	1	1	0	1
2	b	2	1	1	1	0	1	0	1	1	0	1	1	0
3	a	3	2	1	1	1	0	1	0	1	1	1	1	1
4	b	4	3	2	2	1	1	0	1	1	1	1	2	1
5	b	5	4	3	3	2	2	1	1	2	1	1	2	2

The string y occurs with k = 1 error in the string x ending at position 6,7,9 and 10.

SEE NEXT PAGE

c. Outline the trace-back strategy for locating the starting positions of the approximate occurrences of a pattern in a text.

[15 marks]

Recall that DP table output all locations in a text x where a k-difference occurrence of a pattern y ends. To locate the possible starting position of a k-difference occurrence one has to perform a trace-back operation through the matrix. We start at any location of the DP matrix for which $DP[i, m] \leq k$ and follow the inverse operations that where performed to achieve the values in the matrix. Obviously the optimal "return path" is not unique so one chooses any minimum value from the three adjacent cells until we hit the top of the matrix which will give us the value of the possible starting position of the k-difference occurrence.

d. Give the relation to compute $DP[i, j]$ as in 4.b when weighted costs are assumed.

[10 marks]

If weighted costs are used then one has to change the formula for computing the value of $DP[i, j]$. During the main part of the dynamic programming algorithm, we assign $DP[i, j] \leftarrow min\{DP[i-1, j-1] + Sub(x_i, y_j), DP[i-1, j] + Del(x_i), DP[i, j-1] + Ins(y_j)\}$.
Where $Sub(x_i, y_j)$ is the cost for substitution x_i with y_j,$Del(x_i)$ is the cost for deleting x_i, $Ins(y_j)$ is the cost for inserting y_j.

5. **Suffix tree**

a. Design the suffix tree of the string aaaabaaab.

[15 marks]

See last page.

b. Define and explain the data structures used to implement a suffix tree.

[15 marks]

Each node or state p of the automaton can be implemented as a structure containing two pointers: the first pointer is to implement the suffix link; the second pointer gives access to the list of arcs outgoing from state p. The list can contain 4-tuples in the form (a, i, l, q) where a is a letter, i and l are integers, and q is a pointer to a state. they are such that (p, u, q) is an arc of the automaton with $a = y[i]$ and $u = y[i \ldots i + l - 1]$.

c. Let s_y be the suffix link function of the suffix tree of y. Prove that $s_y(p)$ is a fork if the node p is a fork in the tree. Give the lower and upper tight bounds on the number of nodes in the tree.

[10 marks]

Recall that in the construction of a suffix tree we insert suffixes in order of decreasing lengths. By definition, if p is a fork then p has outdegree two at least or is a terminal node with outdegree one and represents a suffix. Denote av as the head of at least two suffixes ending at node p, and let $S_y(p) = q$, where q is the node identified with v. Then we know that in some previous iteration the suffix starting with v was inserted in the tree and since at least two tails split at p then similarly they must be a split at q or else it is terminal state with outdegree one.
let T_y be the suffix tree associated with the string y of length $n+1$. Suppose that T_y is a complete binary tree, thus branching by two at each level. this will maximize the number of internal nodes in T_y. By proceeding by induction we can see that the two subtrees of T_y of size $(n+1)/2$ also have $n/2$ internal nodes. i.e. the sequence: 1,1+2=3,3+4=7,7+8=15,15+16=31 ... Hence for a string of length n, there are at least n and at most $2n$ nodes in the tree.

d. Describe an outline in your own words of the computation of $s_y(p)$ if it is the only one value $s_y(q)$, q node of the tree, not yet defined in the structure.

[10 marks]

Main steps:

1. Goto parent r of p.
2. Use the suffix link of r.
3. Go down the tree by the label of (r,p).

The running time of step 3 above is proportional to the number of arcs along the path and not to the length of the label.

FINAL PAGE