CSMTSP Text Searching and Processing - Solutions

1. (a) The periods and borders of the string `abaababaabaababaababa` are:

$p_0 = 5, b_0 =$ `abaababaabaababa`
$p_1 = 13, b_1 =$ `abaababa`
$p_2 = 18, b_2 =$ `aba`
$p_3 = 20, b_3 =$ `a`
$p_4 = 21, b_5 = \epsilon$

(b) The table $Border$ of the string `abaababaabaababaababa`:
$Border[0:21] = [-1, 0, 0, 1, 1, 2, 3, 2, 3, 4, 5, 6, 4, 5, 6, 7, 8, 9, 10, 11, 7, 8]$

(c)
```
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[i] ≠ x[j + 1] do j := Border[j] ;
        Border[i] := j + 1 ;
    end ;
end ;
```

Since we know already the borders of the prefixes of the word u and that a border of a border is also a border, we try to extend the border of u with the symbol a. If this is successful then we just increment the border by one, if not we fail and try and increment the border of the border of u and so on until we have a match or we find an empty border.

(d) To find if some prefix of x is a square one needs to scan the table $Border$ for values such that $Border[i] = i/2$.

2. (a) See last pages.

(b) The search procedure of 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, δ be the transition function, f be the failure link and $\alpha \in \Sigma$ be a symbol of the alphabet.

Next_State(p, α)
if $\delta(p, \alpha)$ is defined **then** return $\delta(p, \alpha)$
else if $f(p)$ is defined **then** return Next_State($f(p), \alpha$)
else return q_0 (initial state)

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

(a), (b), (c) See last pages.

(d) To find the number of occurrences of a given substring u of a text x in a suffix tree of x we need to walk down the tree by following the symbols of u. Then by doing a traversal of the subtree starting under the last symbol of u we count the number of terminal nodes. This number is the number of times u occurs in x. In the case where the last symbol does not end on a node but within a label of an edge of the tree, then we will consider the node where this edge starts.

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

		0	1	2	3	4	5	6	7	8	9	10
			A	B	A	A	B	A	A	B	C	A
0		0	0	0	0	0	0	0	0	0	0	0
1	B	1	1	0	1	1	1	1	1	0	1	1
2	A	2	1	1	0	1	1	0	1	1	1	1
3	A	3	2	2	1	0	1	1	0	1	2	1
4	C	4	3	3	2	1	1	2	1	1	1	2
5	A	5	4	4	3	2	2	1	2	2	2	1

The string y occurs with $k = 1$ error in the string x ending at positions 6 and 10.

(c) Recall that the 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 in 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) 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 $DP[i, j] \leftarrow \min\{DP[i-1, j-1] + Sub(x_i, y_j)$,
$DP[i, j-1] + Ins(y_j)$,
$DP[i-1, j] + Del(x_i)\}$
. Where $Sub(x_i, y_j)$ is the cost for substituing x_i with y_j, $Ins(y_j)$ is the cost for inserting y_j and $Del(x_i)$ is the cost for deleting x_i.

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

(b) If information concerning the LCP's is known then the time complexity can be reduced to $O(n + \log k)$

(c) 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 $i = \lfloor (k+1)/2 \rfloor$, we will need to preprocess $\log k$ longest common prefixes among the y's and two ones on x.

(d) The suffix array of the string `ababba` is:

1. `a`
2. `ababba`
3. `abba`
4. `ba`
5. `babba`
6. `bba`

(e) The time complexity for searching for a pattern x in a text y is $O(|x| + \log |y|)$

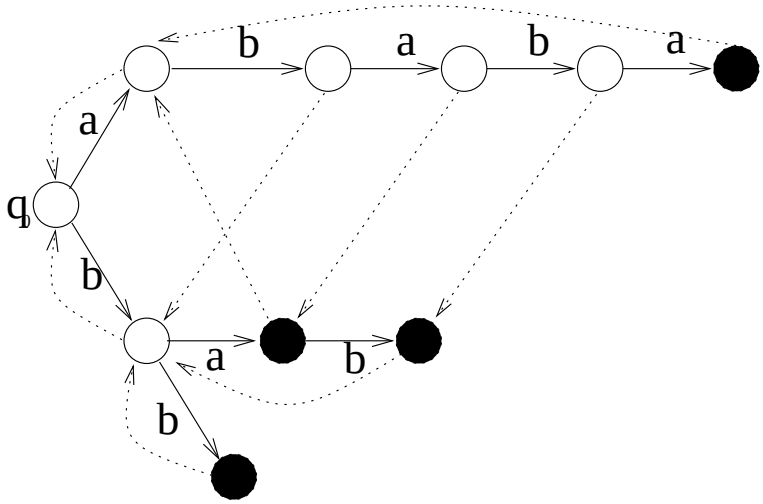

Figure 1: **Question 2. (a)** The Aho-Corasick automaton for the strings: `ababa`, `bab`, `bb`. Black nodes are terminal states and failure links are dotted lines.

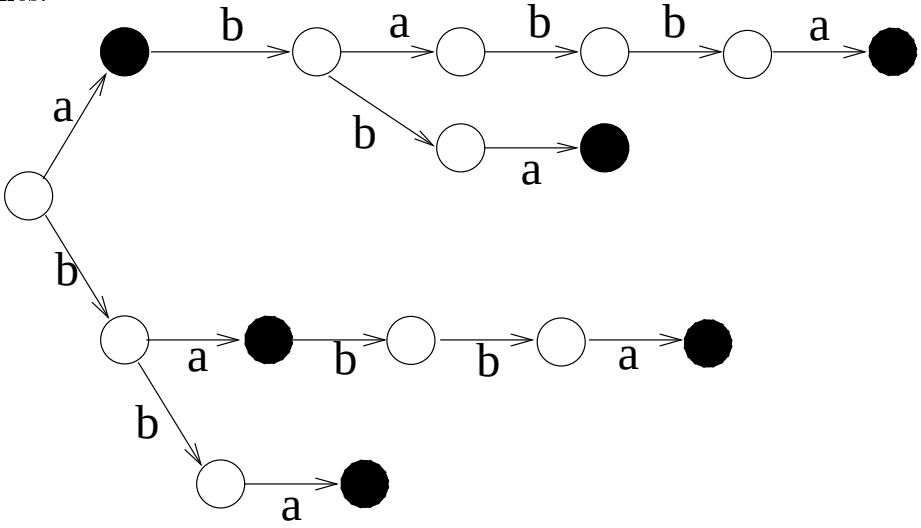

Figure 2: **Question 3. (a)** Suffix trie without suffix links of `ababba`

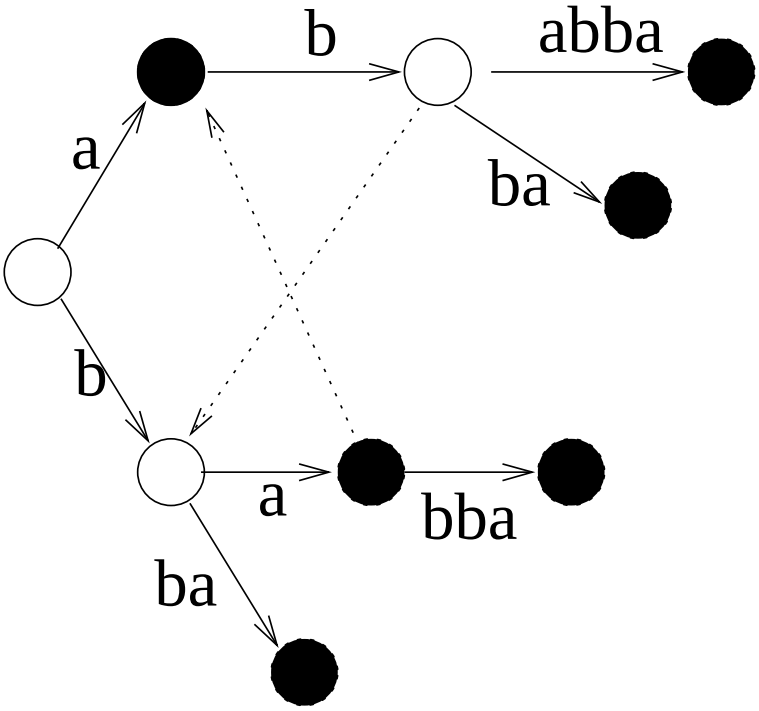

Figure 3: **Question 3. (b)** Suffix tree with suffix links of `ababba`

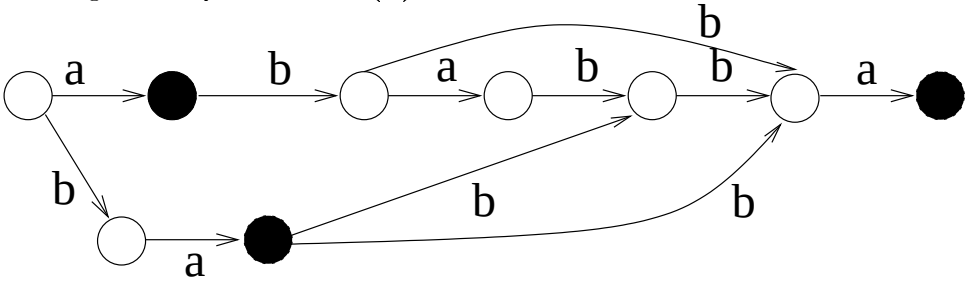

Figure 4: **Question 3. (c)** Suffix automaton of `ababba`