CSMTSP Text Searching and Processing - Solutions

1. In the following we consider the binary alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$.

(a) Describe in pseudo-code a linear-time construction of the *KMP_next* array used in the Knuth-Morris-Pratt (KMP) string-matching algorithm. *[10 marks]*

[Ans]

procedure ComputeKMPNext(p) $\{m = |p|\}$
begin
    $j \leftarrow KMPnext[1] \leftarrow 0$
    for $i \leftarrow 1$ **to** m **do**
        while $j > 0$ **and** $p[i] \neq p[j]$ **do** $j \leftarrow KMPnext[j]$
        $j \leftarrow j + 1$
        if $i = m$ **or** $p[i+1] \neq p[j]$ **then** $KMPnext[i+1] \leftarrow j$
        else $KMPnext[i+1] \leftarrow KMPnext[j]$
end

(b) Give the *KMP_next* array for the pattern $x =$"bbabbabaaa". *[10 marks]*

[Ans]

0	1	2	3	4	5	6	7	8	9	10	11	i
ϵ	b	b	a	b	b	a	b	a	a	a		$p[i]$
-1	0	1	0	1	2	3	4	0	0	0		$BorderArray[i]$
	0	1	2	1	2	3	4	5	1	1	1	$MPnext[i]$
	0	0	2	0	0	2	0	5	1	1	1	$KMPnext[i]$

(c) Describe in pseudo-code the search procedure of the KMP string-matching algorithm. *[10 marks]*

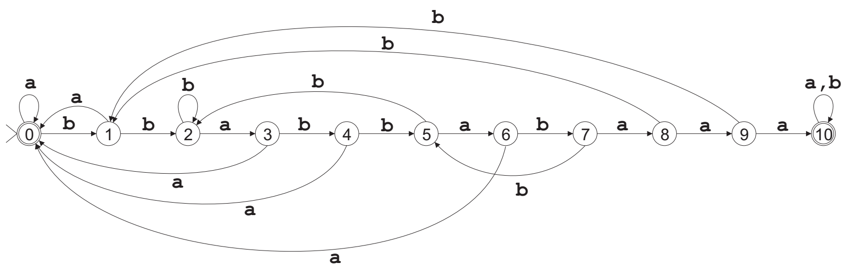
[Ans]

procedure KMP(t, p) $\{ n = |t|, m = |p|\}$
begin

$i \leftarrow j \leftarrow 1$
while $j \leq n$ **do**
    while $(i = m + 1)$ **or** $(i > 0$ **and** $p[i] \neq t[j])$ **do** $i \leftarrow KMPnext[i]$
    $i \leftarrow i + 1$; $j \leftarrow j + 1$
    if $i = m + 1$ **then** Output$(j - i + 1)$
end

(d) For $x \in \Sigma^*$, the string-matching automaton of x, $SMA(x)$, is the minimal deterministic automaton accepting the language $\Sigma^* x$. Design the following string-matching automaton: $SMA($"bbabbabaaa"$)$. *[10 marks]*

[Ans]



(e) State the maximal time spent by the KMP search procedure on a single letter of a text for a pattern of length m (over Σ), when the text is over Σ, and when the text is over the alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. *[10 marks]*

[Ans] It is constant in the first case because each mismatch is immediately followed by a match, and in the second case it is $O(\log m)$ because the delay is $\leq \log_\Phi(m + 1)$, where $\Phi = (1 + \sqrt{5})/2$.

2. It is recalled that $Suf[j]$ is the longest suffix of x ending at position j on x.

(a) Compute the table Suf for $x =$ "aaabbabbababaabbab". *[10 marks]*

[Ans]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	i
ϵ	a	a	a	b	b	a	b	b	a	b	a	b	a	a	b	b	a	b	$p[i]$
	0	0	0	2	1	0	6	1	0	5	0	3	0	0	2	1	0	18	$Suf[i]$

(b) Let k, j, k', ℓ be positions on the word x such that $1 \leq k < j < k' \leq \ell < m$. Assume that $x[k..j]$ and $x[k'..\ell]$ are the longest suffixes of x ending at respective positions j and ℓ on x, and that $k' > k + (m - j)$. Give the value $Suf[\ell - (m - j)]$. *[15 marks]*

[Ans]

$Suf[\ell - (m - j)] = Suf[\ell]$

(c) Write a procedure that computes table Suf. *[15 marks]*

[Ans]

```
procedure ComputeSuf(p)    { m = |p| }
begin
    Suf[m] ← m; j ← m − 1; k ← m
    for i ← m − 1 downto 1 do
        if i − Suf[i + m − j] > k then Suf[i] ← Suf[i + m − j]
        else
            j ← i; k ← min(i, k); kk ← k + m − i
            while k ≥ 0 and x[k] = x[kk] do
                k ← k − 1; kk ← kk − 1
            Suf[i] ← m − kk
end
```

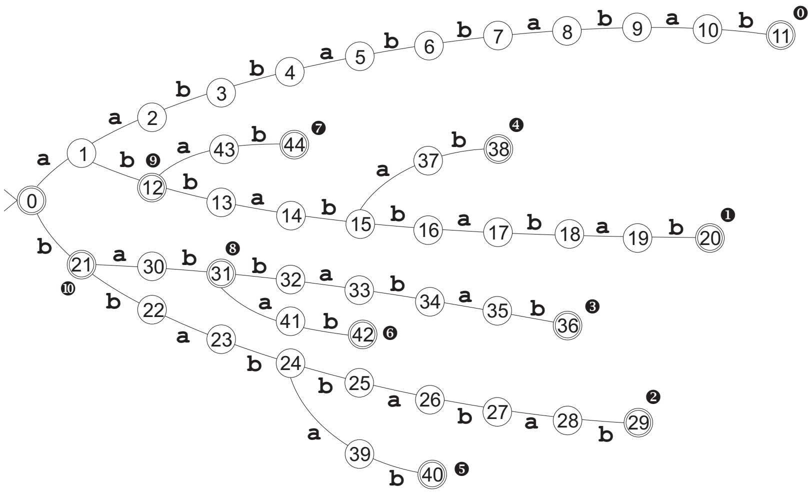
(d) What is the time complexity of your procedure in **2(c)**. Is it optimal? Justify the answer. *[10 marks]*

[Ans]

$O(m)$ because i or k are decremented at each step, and these variables take $O(m)$ values.

3. (a) Give the trie of suffixes of the word "aabbabbabab". *[15 marks]*

[Ans]



(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]*

[Ans]

$a^{m/4}b^{m/4}a^{m/4}b^{m/4}$ for two distinct letters a and b.

(c) Design an algorithm to compact the trie of suffixes of a word into its suffix tree. *[15 marks]*

[Ans]

The following procedure compacts a trie T, even if suffix links are defined on states.

```
procedure Compact(Trie T)
begin
    r ← root of T
    for each arc (r, a, p) do
        Compact(subtree of T rooted at p)
        if p has exactly one child then
            q ← that child
            u ← label of (p, q)
            replace p by q as child of r
            set a · u as label of (r, q)
```

(d) Describe possible data structures required to implement the suffix tree of a word y. *[10 marks]*

[Ans]

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 state p. The list can contain 4-tuples in the form (a, i, ℓ, q) where a is a letter, i and ℓ rae 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 $u = y[i..i + \ell - 1]$.

4. (a) Define the Longest Common Subsequence (LCS) problem. What is the Longest Common Subsequence of "`longest`" and "`large`"? *[10 marks]*

[Ans]

The Longest Common Subsequence (LCS) of two strings, x and y, is a subsequence of both x and of y of maximum possible length. For x="`longest`" and y="`large`", $LCS(x, y) =$ "`lge`".

(b) Give the recursive relation to compute the length of $LCS[x, y]$, using notation $m = |x|$, $n = |y|$ and condition $m \leq n$. *[15 marks]*

[Ans]

$$L[i,j] = \begin{cases} 0, & \text{if either } i = 0 \text{ or } j = 0 \\ L[i-1, j-1] + 1, & \text{if } x[i] = y[j] \\ \max\{L[i-1, j], L[i, j-1]\}, & \text{if } x[i] \neq y[j] \end{cases}$$

(c) Design an algorithm using Dynamic Programming to compute $|LCS[x, y]|$ in space $O(m)$. *[15 marks]*

[Ans]

```
procedure LCS(x, y)    {m = |x|, n = |y|}
begin
   for i ← 0 to m do L[i] ← L'[i] ← 0
   for j ← 1 to n do
      for i ← 1 to m do
         if x_i = y_j then L[i] ← L'[i − 1] + 1
         else
            if L'[i] > L[i − 1] then L[i] ← L'[i]
            else L[i] ← L[i − 1]
      L' ← L
   return L[m]
```

12 **end**

(d) Explain how to recover an LCS using $O(m \times n)$ space.*[10 marks]*

[Ans]

To recover an LCS one has to perform a trace-back operation through the matrix. We start in location $[m, n]$ and follow recursively to the location $[m', n']$ which value $L[m', n']$ has been used to get $L[m, n]$. The process ends on the first line or first column of the matrix. Letters collected at places on the path corresponding to equalities form an LCS.

5. 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) Compute R_1, R_2, R_3, R_4, R_8 for the word "aabbabbabab". *[15 marks]*

[Ans]

0	1	2	3	4	5	6	7	8	9	10	i
a	a	b	b	a	b	b	a	b	a	b	$y[i]$
0	0	1	1	0	1	1	0	1	0	1	$R_1[i]$
0	1	4	3	1	4	3	1	3	1	2	$R_2[i]$
0	3	8	7	3	8	6	2	5	1	4	$R_4[i]$
0	4	10	8	3	9	7	2	6	1	5	$R_8[i]$

(b) State the doubling lemma and prove it. *[10 marks]*

[Ans] Let d be a symbol smaller than all symbols of the alphabet of y. For each position i on y and each $k > 0$, $R_{2k}[i]$ is the rank of $(R_k[i], R_k[i+k])$ in the ordered list of these pairs after padding y by enough d's.

Proof: it is sufficient to prove that, for two positions i and j, $R_{2k}[i] \leq R_{2k}[j]$ if and only if $(R_k[i], R_k[i+k]) \leq (R_k[j], R_k[j+k])$ in the lexicographic order. This is a simple verification.

(c) Design an algorithm to compute R_{2k} from R_k. *[15 marks]*

[Ans]

Using R_k, create the array L such that $L[i] = (R_k[i], R_k[i+k])$ for $0 \leq i \leq m$. Then $R_{2k}[i] =$ rank of $L[i]$ into the ordered list of elements of L, according to the doubling lemma.

(d) State the complexity of the algorithm based on question **5(c)** to compute $R_1, R_2, \ldots, R_{2^k}$, where k is the smallest integer satisfying the inequality $m \leq 2^k$. Explain your answer. *[10 marks]*

[Ans]

$O(m \log m)$ because there are $O(logm)$ steps, and each step can be implemented in linear time using bucket sorting in the procedure described in answer **5(c)**.

Final Page