**CSMTSP Text Searching and Processing - Solutions**

**1.** (a) The string-matching automata $SMA$(a), $SMA$(aa), $SMA$(aab), $SMA$(aaba):

(b) The process of adding a new character $\tau$ to the computed automaton $SMA(x)$ ($x \in \Sigma^*, \tau \in \Sigma$) in the on-line algorithm is referred to unwinding the arc $\delta(\tau)$ from the final state computed in the previous step.

1) Let $r := \delta(final\_state, \tau)$, this is the state to which the arc we a re about to unwind is pointing. 2) Add a new final state $s$ such $s := \delta(final\_state, \tau)$. 3) Now the new final state in $SMA(x\sigma)$ will behave for each symbol in the alphabet as the state $r$.

More formerly,

**while not** end of $x$ **do begin**
  $\tau :=$ next symbol of $x$ ; $r := \delta(terminal, \tau)$ ;
  add new state $s$ to $Q$ ; $\delta(terminal, \tau) := s$ ;
  **for** all $\sigma in \Sigma$ **do** $\delta(s, \sigma) := \delta(r, \sigma)$ ;
  $terminal := s$ ;
**end**;

(c) Backward arcs are arcs in the automaton that do not point to the initial state. There are two backward arcs in $SMA$(aaba): node 2 to node 2 with symbol a on the arc and node 4 to 2 with symbol a on the arc.

(d) Recall that each state of the automaton $SMA(x)$ is identified with a prefix of $x$. A backward edge is of the form $(u, \tau, v\tau)$ where $u$ and $v$ are prefixes of $x$ and $\tau \in \Sigma$, a symbol such that $u \neq v\tau$ and $v\tau$ is the longest suffix of $u\tau$ that is a prefix of $x$. Note that $u\tau$ is not a prefix of $x$.

Let $p(u, v, \tau) = |u| - |v|$ (a period of $u$). We now prove that each period of the prefixes $p, 1 \leq p \leq |x|$ corresponds to at most one backward arc. Thus there are at most $|x|$ such edges.

Suppose that two backward edges $(u, \tau, v\tau)$ and $(u', \tau', v'\tau')$ have the same period $p = p(u, v, \tau) = p(u', v', \tau')$ then $v\tau = v'\tau'$. Otherwise if $v\tau$ is say a proper prefix of $v'\tau'$ then $v\tau$ would also occur at position $p$ like $v'$ so $u\tau$ would also be a prefix of $x$. This is a contradiction, hence $v = v', \tau = \tau'$ and $u = u'$. Hence $SMA(x)$ has no more than $|x|$ backward arcs.

**2.** (a) The periods and borders of the string aaabaaaabaaaabaaaa are:
$p_0 = 5, b_0 =$ aaabaaaabaaaa
$p_1 = 10, b_1 =$ aaabaaaa
$p_2 = 15, b_2 =$ aaa
$p_3 = 16, b_3 =$ aa
$p_4 = 17, b_4 =$ a
$p_5 = 18, b_5 = \epsilon$

(b) Recall that a border of $x$ is a both 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 a prefix, $b_2$ is 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 $x$. The string $x$ of length $n$ has at most $n$ borders starting from $\epsilon$ to $border(x)$. By using the induction principle on the above remark, we know that $b_0 = border(x)$ is the largest proper border of $x$. Let $b_1$ be the next largest proper border of $x$. We now from the above that $b_1$ is in turn a border of $b_0$. Hence the recurrence $b_i = border(b_{i-1})$ for all $i \in \{0, \ldots, k\}$, where $b_k = \epsilon, 1 \leq k \leq n$, implies a mapping of the borders such that $b_0 = \epsilon \subset border(x[1]) \subset \ldots \subset border(x[n]) = border(x[n-1])$. The inclusion operator is defined as "border of" which implies that any border of $x$ is either $b_0 = border(x)$ or a border of $b_0$.

(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 ;
```

(d) Output of above algorithm for `aaabaaabaaaa`:
$Border[0:12] = [-1, 0, 1, 2, 0, 1, 2, 3, 4, 5, 6, 7, 3]$

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

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

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

(d) At each step in the algorithm of part c) 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 $x$.

**4.** (a) The expanded suffix tree associated with the string `abaababa`\$:

The expanded suffix tree associated with the string `abaababa`\$:

(b) Given a string $x$ of length $n$, the expanded suffix tree of $x$ may contain in the "worst case" $\theta(n^2)$ internal nodes. If the symbols in $x$ are all different e.g.: $x =$ `abcdef`...\$, then we have $n$ arcs each having $n, n-1, \ldots, 1$ nodes. In the other extreme case, an expanded suffix tree of $x$ may contain exactly $n-1$ internal nodes. For example, the expanded suffix tree of $x =$ `aaa`...\$ consists of one arc with $n-1$ outgoing leaves.

(c) A compact suffix tree is an expanded suffix tree which has had every sequence of arcs formed by nodes with only one child into a single arc and label that arc with a substring. Let $T_x$ be the compact suffix tree associated to the string $x$ of length $n+1$. Suppose that $T_x$ is a complete binary tree, thus branching by two at each level. This will maximize the number of internal nodes in $T_x$. By proceeding by induction we can see that the two subtrees of $T_x$ of size $(n+1)/2$ also have $n/2$ internal nodes. i.e. the sequence: $1, 1+2=3, 3+4=7, 7+8=15, 15+16=31, \ldots$

(d) (i) Suppose that $w$ is not a primitive, then $w = v^k$ for some $k \neq 1$ then $ww = v^{2k}$ which is clearly not a square. Hence $w$ must a primitive string. We can see this by sliding two copies of the string $w$ from right to left until we have a match. A match will only occur when $w$ and $w$ are aligned. If $w$ was not a primitive then a match would occur earlier.

(ii) If a square $ww$ starts at position $i$ in $x$ then clearly by the above result, the suffix tree $T_x$ will have a leaf $i$ followed by a leaf $i+|w|$ since $w$ is a primitive i.e. the arc going to the leaf $i+|w|$ will be labelled with $w$. For example in the diagram, we can see that a square $ww = abab$ starts at position 4 with consecutive leaves 4 and 6 defining this square.

**5.** (a) The Hamming distance of a two strings $x$ and $y$ is the number of mismatches allowed between the two. The Levenstein distance is an edit distance that allows insertion, deletion and substitutions of one character at a time each having a unit cost.

(b) Build a $(n+1) \times (m+1)$ table $C$ where we place the string $y$ at the top and the string $x$ on the left. Initialize all entries $C[i, 0] = C[0, j] = 0$. Now proceed to compute $C[i, j]$ by taking the minimum value of:
$C[i, 1, j-1] + substitute(x_i, y_j)$
$C[i-1, j] + delete(x_i)$
$C[i, j-1] + insert(y_j)$
according to the costs of the operations: substitute, delete and insert.

(c) By using dynamic programing the above procedure can be extended to that of computing a shortest source-to-sink path in an edge-weighted grid of a directed acyclic graph once the table $C$ has been computed. This will result in an optimum edit script for transforming $x$ into $y$ with a minimum total cost.

(d) Let $cost(del) = cost(ins) = 1$, and $cost(subst) \leq cost(del) + cost(ins)$. Then no edit script will use the substitution operation. The pairs of matching symbols preserved in an optimal edit script constitute a longest common subsequence of $x$ and $y$. If $s$ is the length of a longest common subsequence between $x$ and $y$, then on a minimum edit distance path in the grid $C$, because no substitution is made then we either go down of up. This results in the minimum edit distance $e$ being the sum of the lengths of $x$ and $y$ minus twice the length of the longest common subsequence between $x$ and $y$.