# COMP9601 Assignment 3

## Problem 1

We use $k$ to denote the number of strings in $X$. Define $\Lambda = \{\$_1, ...\$_k\}$ as $k$ distinct characters which do not appear in the strings in $X$. We first concatenate all strings in $X$ and each string is followed by a character in $\Lambda$ as follows

$$S = s_1\$_1 s_2\$_2, ..., s_{k-1}\$_{k-1} s_k\$_k.$$

Let $K$ denote the length of this concatenated string. We define $\alpha[i] = LCP(SA[i], SA[i-1])(i = 2, ..., n)$ where $LCP(a, b)$ is the longest prefix of two strings. Given $\alpha$, we can find the longest common string (LCS) of all $|X|$ strings by binary searching on the length of the LCS. If we want to find the LCS with length $l$, we can greedily divide the interval $i = [2, K]$ into maximal sub-intervals while the value of $\alpha[i]$ in each sub-interval is at least $l$ or the size of the interval is 1 but $\alpha[i] < l$. If there is a sub-interval that contains at least one suffix from each string in $X$. Then the LCS with length $l$ exists.

It suffices to show that we can construct $\alpha$ in $O(n)$ time. Then the algorithm stated above can be finished in $O(n \log n)$ time. We define $h[i] = \alpha(SA^{-1}[i])$ which is the $LCP(T[i...K], T[SA^{-1}[i] - 1...K])$. Therefore, we can construct $\alpha[i]$ if we have $h$ with $\alpha[i] = h[SA[i]]$. In fact, $h[i]$ has the following property (proof is omitted):

$$h[i] \geq h[i-1] - 1.$$

Equipped with this property, we don't need to check the first $h[i-1] - 1$'s characters in $T[i...K]$ and $T[SA^{-1}[i] - 1...K]$ for computing $h[i]$. Hence, we can calculate the $h$ array in $O(n)$ time.

## Problem 2

The algorithm is illustrated as in Algorithm 1. We briefly explain the algorithm in the rest of the solution.

Because the suffixes of $T$ has been sorted with respect to their first $k$ characters, if $Start[i] = 0$, the first $k$ characters of $T[A[i]...n]$ and $T[A[i-1]...n]$ must be the same (it means the rank can not be decided with only the first $k$ characters). We define $A^{-1}[i] = j$ if $A[j] = i$.

Find the intervals $[s_i, t_i]$ (where $s_i$ is the start index and $t_i$ is the end index of $i$-th interval) such that $Start[j] = 0, \forall j \in [s_i, t_i]$, $Start[s_i - 1] = 1$ (if $s_i > 1$) and $Start[t_i + 1] = 1$ (if $t_i < n$). In other words, each interval is the region with maximal length such that $Start[j]$ is 0.

For each region $[s_i, t_i]$, we need to sort the suffixes (start from the positions of $A[s_i-1], A[s_i], ..., A[t_i]$) with respect to the first $2k$ characters. As stated above, the first $k$ characters of these suffixes are the same. The rank can be decided by checking the rank of the suffix $T[A[j]+k...n]$ ($\forall j \in [s_i-1, t_i]$) in $A^{-1}$. In other words, because we have the rank of suffixes with respect to the first $k$ characters. The rank of $T[A[j] + k...n]$ with respect to $T[A[j] + k...A[j] + 2k - 1]$ can be found by directly checking $A^{-1}[A[j] + k]$. Define $\delta(i) = r_j$ (if $A^{-1}[A[j] + k] = i$) where $r_j$ is the index of the region.

**Algorithm 1:** Calculate $A', Start'$ from $A, Start$

**Input**: $A, Start, k$
**Output**: $A', Start'$

Initial $A'[i] = A[i], \forall i = 1, ..., n$;
Initial $\alpha(i) = 0, \delta(i) = 0, \beta(i) = 0, \forall i = 1, ..., n$;
Initial $beg(i) = 0, \forall i = 1, ..., n$; // the start position of regions of consecutive 0s ;
$i = 1, r = 0$;// r is the number of regions of consecutive 0s ;
**while** $i \neq n$ **do**
    **if** $Start[i] = 0$ **then**
        $r = r + 1, beg(r) = i - 1$;
        $j = i$; **while** $j \leq n$ ***and*** $Start[j] = 0$ **do**
            $\alpha(i) = r$;
            $j = j + 1$;
        **for** $p = \{i - 1, i, ...j - 1\}$ **do**
            $\delta(A^{-1}[A[p] + k]) = r$;
            $\beta(A^{-1}[A[p] + k]) = p$;
        $i = j$;
    $i = i + 1$;
Let $b = beg$;
**for** $i = 1...n$ **do**
    **if** $\delta(i) \neq 0$ **then**
        $x = b(\delta(i))$; //the rank in $A'$;
        $A'[x] = A[\beta(i)]$;
        **if** $\alpha(A^{-1}[A'[x - 1]]) = \alpha(A^{-1}[A'[x]])$ ***and*** $\alpha(A^{-1}[A'[x - 1] + k]) = \alpha(A^{-1}[A'[x] + k])$ **then**
            $Start'[x] = 0$;
        $b(\delta(i)) = b(\delta(i)) + 1$;

Define $\beta(i) = j$ (if $A^{-1}[A[j] + k] = i$). After calculating $\delta(i)$ and $\beta(i)$, we can find the rank of suffixes with respect to the first $2k$ characters by scanning them once.

## Problem 3

Denote $[l, r]$ as the suffix range of $P$ and $[l', r']$ as the suffix range of $P'$. Denote $|P|$ is the length of $P$ Denote $[l_c, r_c]$ as the suffix range of $PP'$ which is our goal to find with given $[l, r]$ and $[l', r']$.

We have the following two facts:

- Because $P$ is the prefix of $PP'$, we have $l \leq l_c \leq r_c \leq r$.
- Because the ranks of the suffixes with same prefix $P$ (i.e. $T[SA[i]...n]$ $i \in [l, r]$) are decided by $T[SA[i] + |P|...n]$), we have $SA^{-1}[SA[i] + |P|] \leq SA^{-1}[SA[j] + |P|]$ ($\forall i, j \in [l, r], i \leq j$).

With these two facts, $l_c$ can be found by binary searching on the smallest position such that $SA^{-1}[SA[l_c] + |P|] \geq l'$. Similarly $r_c$ can be found by binary searching on the largest position such that $SA^{-1}[SA[r_c] + |P|] \leq r'$.

## Problem 4

1. $SA^{-1}[k] = \Psi^k[1]$.
2. In order to get $SA[j]$, we calculate $\Psi^0[j] = j, \Psi^1[j], \Psi^2[j], ...$ in sequence, then

$$SA[j] = SA[\Psi^k[j]] - k$$

holds for all $k$. We check whether $B[\Psi^k[j]] = 1$ every step, and stop if it is true. Since $SA[\Psi^k[j]]$ increases a unit one time, this algorithm must stop in $O(\log n)$ time. Then we have $SA[j] = SA[\Psi^t[j]] - t$, where $t \geq 0$ is the smallest $t$, $B[\Psi^t[j]] = 1$ holds.

## Problem 5

Define $Appear[i, x] = \#$ of x in $BWT[0...i-1]$. We start with $T[n] = 1$ which is the smallest suffix ($SA[1] = n$). By checking $BWT[1]$, it can be known that $T[n-1] = BWT[1]$. Then we find the rank of $T[n-1...n]$ by find the range of suffixes which start with the character $T[n-1]$. With the help of $Appear[1, BWT[1]]$, we know the rank of the suffix $T[n-1...n]$ accurately (say $k$). Then, $T[n-2] = BWT[k]$. Following the steps above, the string $T$ can be constructed from the last character to the first. The algorithm is illustrated in Algorithm 2. With the help of

**Algorithm 2:** The algorithm

$i = 1, p = n$;
$T[n] = \$$;
**while** $p > 1$ **do**
  $p = p - 1$;
  $x = BWT[i]$;
  $T[p] = x$;
  $i = count[x] + Appear[i, x] + 1$;

$Appear[i, x] = \#$ of x in $BWT[0...i-1]$. The $\Psi$ can be constructed with the similar approach with Algorithm 2. The algorithm for construct $\Psi$ is illustrated in Algorithm 3. It need $O(n|\Sigma|)$ ($|\Sigma|$ is the size of character size) to calculate $Appear$ and $O(n)$ time to calculate $\Psi$. Hence, the time complexity is $O(n|\Sigma| + n) = O(n|\Sigma|)$.

**Algorithm 3:** The algorithm

$i = 1, p = n$;
**while** $p > 1$ **do**
    $p = p - 1$;
    $x = BWT[i]$;
    $i' = count[x] + Appear[i, x] + 1$;
    $\Psi[i'] = i$;
    $i = i'$;
$\Psi[1] = i$;