**CSE5095: Research Topics in Big Data Analytics**
Professor Sanguthevar Rajasekaran
Note Taker: Nhan Nguyen
Date: $13^{th}$ March 2014

---

**Lecture Recap:**
In the last lecture, we learnt about the suffix-prefix problem. We have shown that:
- We can solve the suffix-prefix problem in $O(M + n^2)$ time where $M = \sum_{i=1}^{n} |S_i|$.
- We can construct a suffix array for a string of size $m$ in $O(m)$ time.
- The exact string matching problem can be solved using a suffix array and binary search in $O(n \log m)$ time.

In this lecture, we'll show that string matching can be done in $O(n + \log m)$ time.

**Lemma:**
We can search for a pattern $P$ in a text $T$ in $O(n + \log m)$ time where $n = |P|$ and $m = |T|$, given the suffix array for $T$.

**Proof:**
Let $S_i$ stand for the suffix of $T$ starting at position $i$, for $1 \leq i \leq m$. Let $SA[1 : m]$ be the suffix array for $T$. Specifically, $SA[j]$ is the starting position in $T$ of the $j$th smallest suffix of $T$.

For any two suffixes $S_i$ and $S_j$, let $LCP(i, j)$ be the length of their longest common prefix.

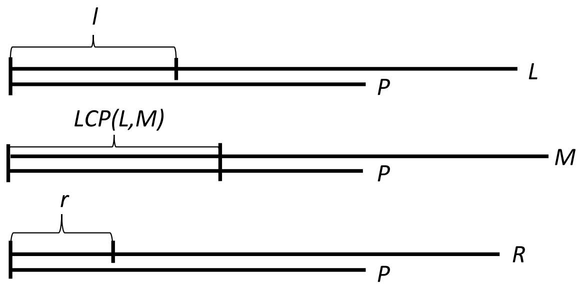
**Example:**
If $T = gaagcctgat$, then
$LCP(1, 8) = 2$.
Assume that we can get $LCP(i, j)$ in $O(1)$ time for any $i$ and $j$. From hereon, we let $LCP(i, j)$ denote the length of the longest common prefix between the $i$th smallest and the $j$th smallest suffixes of $T$. To search for $P$ in $T$, we will use binary search (on the suffix array) again but with some crucial modifications. Note that in any iteration of the binary search we have three integers $L, M$, and $R$. Here $L$ is the left boundary and $R$ is the right boundary. $M$ is nothing but $(L+R)/2$. Note that in any iteration of the binary search we compare $P$ and suffix $M$ to see whether there is a match, $P$ will be to the left of suffix $M$, or $P$ will be the right of suffix $M$. (Recall that suffix $k$ refers to the $k$th smallest suffix of $T$.) We keep track of the length of the longest common prefix between $P$ and suffix $L$. Let this be $l$. We also keep track of the length of the longest common prefix between $P$ and suffix $R$. Let this be $r$.

Call a comparison of a character of $P$ with any character (in $T$) as redundant if this character of $P$ has already been compared with a character (in $T$). We want to minimize the number of redundant comparisons. The following algorithm ensures that there will be at most one redundant comparison in any iteration of the binary search. Let $MLR = \max\{l, r\}$.

- **Case 1:** $l = r$
  We start the comparison between $P$ and suffix $M$ starting from position $(l + 1)$.
  Note that in this case $l = r = MLR$.

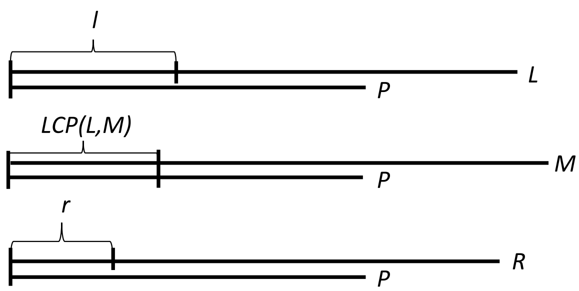
- **Case 2:** $l > r$.

  · **Case 2a:** $LCP(L, M) > l$



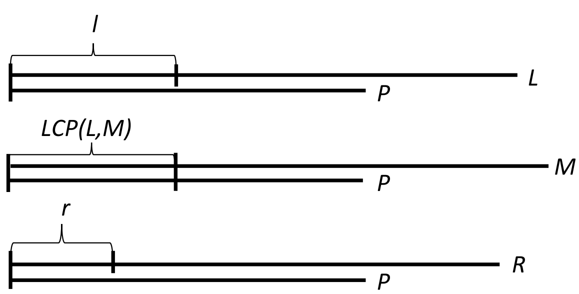

In this case $P >$ suffix $M$
Set $L = M$, move on to the next iteration.

· **Case 2b:** $LCP(L, M) < l$




In this case $P <$ suffix $M$
Set $R = M; r = LCP(L, M)$;

· **Case 2c:** $LCP(L, M) = l$



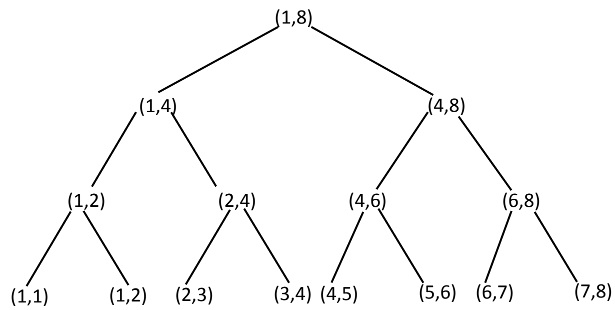

In this case, we start the comparison of $P$ starting from position $MLR+1$ in suffix $M$. Depending on how $P$ and suffix $M$ compare, the binary search will proceed.

- **Case 3:** $l < r$. This case is analogous to Case 2.

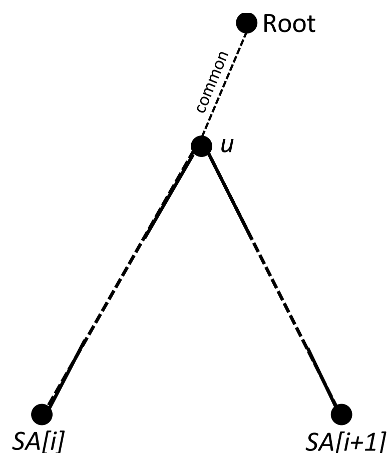
**Anlysis:**
Call a comparison of a character in $P$ as redundant if this character has already been compared.
In any iteration of the binary search, either we terminate the search, or do not do any character comparison of $P$, or start comparing from position $MLR + 1$. When we start comparison of $P$ from position $MLR + 1$, this character might have already been compared (in a previous iteration). Characters to the right of this character in $P$ would not have been compared before.
Note: in any iteration, we only do at most one redundant comparison.
$\rightarrow$ Total number of comparisons is $O(n + \log m)$.

## Construction of the $LCP$ array

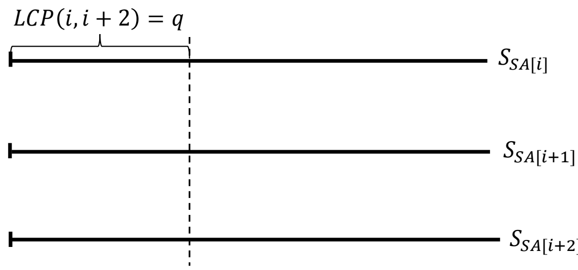
Let $LCP(i, j)$ stand for the length of the longest common prefix between the $i$th smallest and the $j$th smallest suffixes of $M$. Think of a tree for binary search, as follows:




We have a complete binary tree with $(1, m)$ as the root. Any internal node $(i, j)$ will have two children $(i, \lfloor \frac{i+j}{2} \rfloor)\&(\lfloor \frac{i+j}{2} \rfloor, j)$
There are $m$ leaves $(1, 1), (1, 2), (2, 3), (3, 4), \ldots, (m - 1, m)$.
To compute $LCP(i, i + 1)$ for any $i$ we do a lexicographic DFS on the suffix tree for $T$.
Let $u$ be the internal node that is closest to the root among the nodes visited between leaf $SA[i]$ and leaf $SA[i + 1]$.
Then, we can see that $LCP(i, i + 1) =$ the string depth of node $u$.
$\rightarrow$ we can compute all the leaf $LCP$ values in $O(m)$ time.




Then compute $LCP(i, j)$ for any $j \geq (i + 2)$ using the following
**FACT**: $LCP(i, j) = \min_{k=i}^{j-1} LCP(k, k + 1)$
**Proof:**
$LCP(i, j) \leq LCP(k, k + 1) \forall k = i \cdots j - 1$
$\rightarrow LCP(i, j) \leq \min_{k=i}^{j-1} LCP(k, k + 1)$.




In this case, $LCP(i, i+2) \geq min\{LCP(i, i+1), LCP(i+1, i+2)\}$.

$\rightarrow$ We can extended this to see that $LCP(i, j) \geq \min_{k=i}^{j-1} LCP(k, k+1)$. $\square$

**Question**: How do we construct the suffix array without going through the the suffix tree construction?

Three teams have proposed linear time algorithms:
- KÄRKÄINNEN, SANDERS 2003
- KO, ALURU 2003
- KIM, SIM, PARK, PARK 2003

Now we will see the skew algorithm of (KÄRKÄINNEN, SANDERS 2003).
Let T = $t_0 t_1 t_2 t_3 t_4 t_5 \dots t_{m-1}$. W.l.o.g. assume that $m = 3q$ for some integer $q$.
The idea: Recursively sort the suffixes that start at positions $i$ such that $i \bmod 3 \neq 0$.
Then use this ordering to find the ordering of the remaining one third suffixes.

**Notations:**
Let $B_k = \{i \in [0, m] : i \bmod 3 = k\}$ with $k = 0, 1, 2$.
Let $S_i$ be the suffix of $T$ starting from position $i$.
Let $S_C$ be the set of suffixes starting from positions in $C$, where $C$ is a set of integers.
Let $B = B_1 \bigcup B_2$.
① Sort the suffixes $S_B$; Let $Q$ be the sorted list.
② Using the above, sort the suffixes $S_{B_0}$; Let $Q'$ be the sorted list.
③ Merge $Q$ and $Q'$.

**Note:** If sufficies to assume that $\Sigma = \{1, 2, 3, \dots, m\}$. (TBC)