We use some definitions for our proposed model. Consider a string S with a length of $|S| = m$, comprised of symbols drawn from an ordered alphabet. We represent the i-th symbol of S as $S[i]$, where $1 \leq i \leq n$. $S[i, j]$ refers to a substring of S from position i to j, inclusive. Specifically, $S[1, j]$ denotes the prefix of S that concludes at position j, and $S[i, n]$ signifies the suffix of S commencing at position i, denoted as S_i. We employ the symbol to express the lexicographic order relationship between strings.

The Suffix Array of a string S, denoted as SA, is an array of integers ranging from 1 to m. This array ranks all the suffixes of S in lexicographic order. We denote the position of the suffix S_i in the Suffix Array as $pos(S_i)$.

The LCP-array, another crucial component, consists of integers that store the length of the longest common prefix (lcp) between consecutive suffixes in SA. $LCP[1]$ is set to 0, and for $1 < i \leq m$, $LCP[i]$ equals $lcp(S_{SA}[i], S_{SA}[i - 1])$. Here, $lcp(u, v)$ represents the longest common prefix of strings u and v. Constructing both SA and the LCP-array can be achieved in linear time[**?**].

The range minimum query (RMQ) concerning the LCP array entails finding the smallest lcp value within a given interval of SA. We define $RMQ(i, j)$ as the minimum value of $LCP[k]$ for $i < k \leq j$. Remarkably, for a string S with length n and its corresponding LCP-array, the $lcp(S_{SA}[i], S_{SA}[j])$ is equivalent to $RMQ(i, j)$.

Let $S = \{S_1, S_2, \ldots, S_m\}$ be a collection of m strings. The generalized Suffix Array of S is the Suffix Array SA of the concatenated string $S = S^1\$_1\, S^2\$_2 \ldots S^m\$_m$, where each symbol $\$i$ is a distinct separator that does not occur in S and precedes every symbol in S, and $\$_i < \$_j$ if $i < j$. For a suffix $S_{SA}[i]$ of S, we denote the prefix of $S_{SA}[i]$ that ends at the first separator $\$_j$ by $S_{SA}[i]$. The total length of the generalized Suffix Array is $N = m + \sum_{l=1}^{m} |S_l|$.

To simplify the notation, we introduce the arrays STR and SA''. STR indicates the string in S from which a suffix came, formally $STR[i] = j$ if the suffix $S_{SA}[i]$ ends with symbol $\$_j$. SA'' holds the position of a suffix with respect to the string it came from (up to the separator), defined as $SA''[i] = k$ if $S_{SA}[i] = S_j^k\, \$j$. Taken together, STR and SA'' specify the order of all suffixes in S.

2 Methodology

Our model solution can solve the all-pair suffix-prefix matching problem when adding a new string. Let's assume that we already have m strings: $[S_1, S_2, S_3, \ldots, S_m]$, and we possess a grid named OV with a size of $m \times m$. In this grid, each cell (i, j) represents the maximum length of a suffix of string S_j that is also a prefix of string S_i. When a new string Sm' is introduced, we only need to update the m'-th row and m'-th column of the OV grid. Utilizing our existing model, we can avoid recalculating the entire grid, thereby significantly reducing complexity.

In contrast, using the conventional model would require a complete recalculation, leading to higher complexity. Specifically, the approximate complexity of adding a new string using our model is $O(len \cdot \log(len)) + O(m \cdot \log(tot\_len))$, where len represents the length of the new string, m is the number of strings, and tot_len is the sum of the lengths of all strings plus m. On the other hand, the complexity of the existing model would be $O(tot\_len + m^2)$, which is considerably greater than that of our model.

Our model solution primarily relies on an online Suffix Array approach. To effectively implement the online Suffix Array, we employ data structures such as Treap, Suffix Array, LCP array, Segment Tree, BIT (Binary Indexed Tree), stack, and priority queue.

In our solution, we consider an existing grid OV with dimensions $m \times m$. Our objective is to add a new row and column to this grid. To accomplish this, we'll utilize the notations and definitions mentioned earlier.

We possess an existing Suffix Array that encompasses all the strings added thus far. Our approach involves continually appending new suffixes to this Suffix Array. To simplify our implementation of the online Suffix Array, we adopt a strategy where we add suffixes in reverse order, starting from the last character and working our way to the first character in the new string. For this purpose, we prepend each of these characters to the front of the string S, which is formed by concatenating all the previous strings while introducing a unique separator between them.

Proposed model **1 Notation and defination**