Big Data Lecture Notes on March 4

1 Suffix Tree Overview

Consider any string: $T = x_1, x_2, ..., x_m \in \sum^m$, $\sum \rightarrow$ Alphabet.
A suffix tree on T is a rooted directed tree where:
1. It has m leaves, one for each possible suffix of T.
2. Every internal node other than the root has a degree of ≥ 2.
3. Each edge is labelled with a substring of T.
4. Each leaf has a label in the range $[1, m]$.
5. Labels of no two edges out of a node can start with the same character.
6. The concatenation of the edge labels along the path from the root to leaf i, spells the suffix $T[i : m]$.

Example 1: $T = abacd$

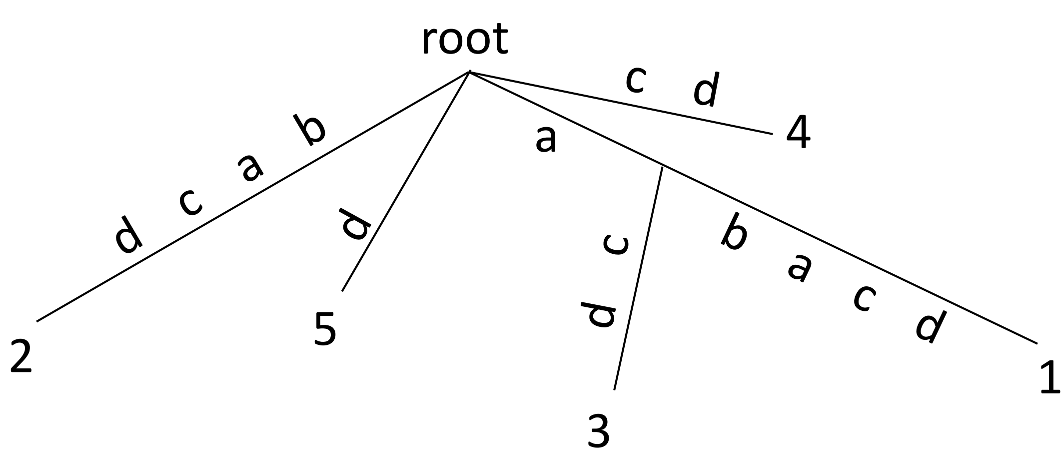

Figure 1: Example 1

Theorem: We can construct a suffix tree on any string of length m in $O(m)$ time.

Note: Consider a string $T = acabac$. Here the suffix ac is a prefix of another suffix $acabac$. We won't be able to construct a suffix tree satisfying all of the above characteristics for this string since we may not be able to have a leaf corresponding to the suffix ac. So we put a character that is not in the alphabet to the end of every string to handle this problem. This character is denoted as \$.

Definition:
1. The label of any path is the ordered concatenation of labels on the edges in the path.
2. The path label of any node is the label of the path from the root to this node.
3. The string depth of any node is the number of characters in its label.

Fact: We can construct a suffix tree on any string of length m in $O(m^2)$ time.
Proof: We will insert one suffix at a time starting from a path for T. Let R_i be the subtree for suffixes 1 to i. R_1 is the path corresponding to T. To insert suffix $i+1$ (i.e., $x_{i+1}x_{i+2}\cdots x_m$) into R_i start matching the characters of this suffix with a unique path in R_i. We will come to a stage where no more matches are possible:
1. If this happens at a node of R_i, create a new child for this node with the remaining substring (of suffix $i+1$) as the label of the resultant edge.
2. If this happens in the middle of an edge label, split the edge by inserting a new node and proceed as in case 1.

Example 2: $T = cdaabc\$$

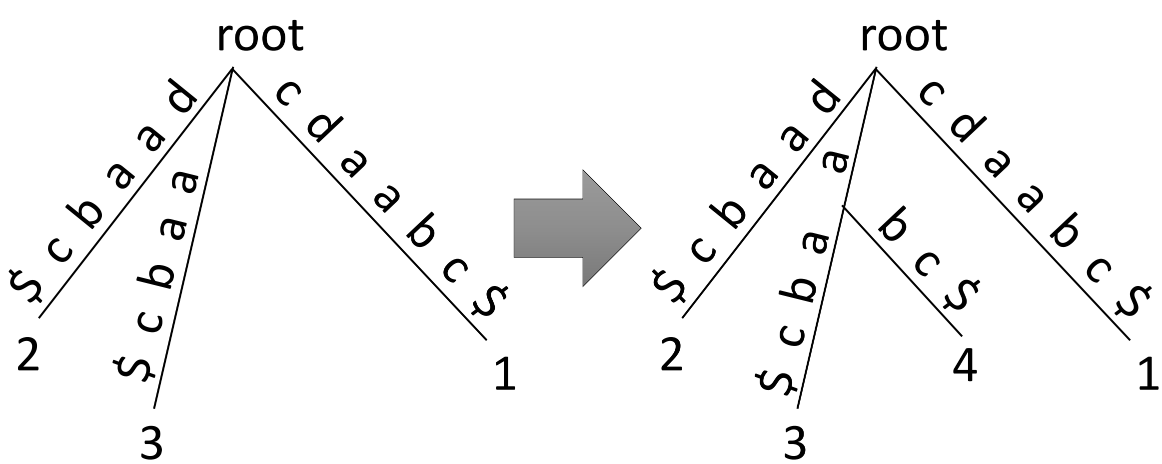

Figure 2: Example 2. This figure shows the process of inserting suffix 4 into R_3.

2 Generalized Suffix Tree

Input: Strings $S_1, S_2, ..., S_n$
We can construct a single suffix tree for all of these strings where there is a leaf for every suffix of every string. Each leaf is labelled as a pair (i, j), where i refers to the string number and j refers to the suffix number within string i.

Example 3: $S_1 = abac$, $S_2 = bbac$

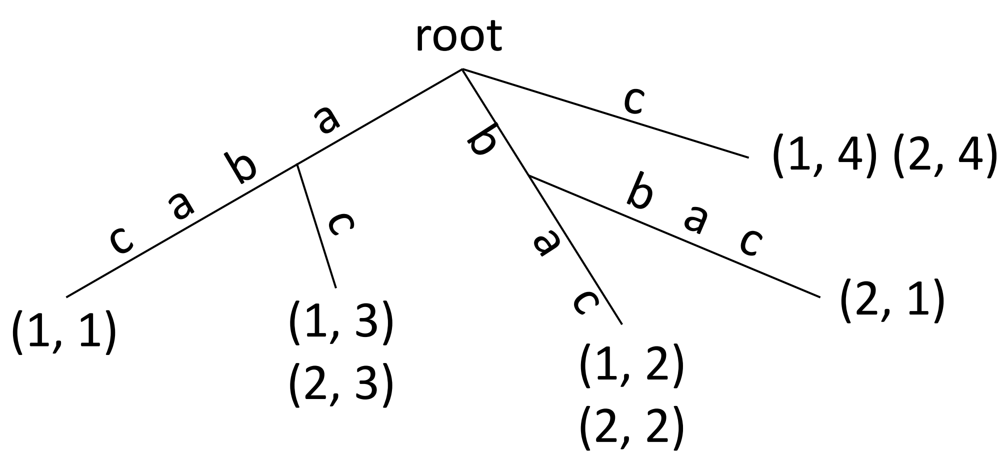

Figure 3: Example 3

Fact: If $\sum_{i=1}^{n} |S_i| = M$, then we can construct a generalized suffix tree in $O(M)$ time. One idea: Construct the string $S_1\$_1 S_2\$_2 \cdots S_n\$_n$, and build a generalized suffix tree for this string in $O(M)$ time. Trim the unnecessary suffixes.

3 Problem 1: Exact String Matching

Input: a text $T = t_1 t_2 \cdots t_m$ and a pattern $P = p_1 p_2 \cdots p_n$
Output: All the occurrences of P in T
Algorithm: Build a suffix tree for T in $O(m)$ time. Start matching the characters in P with the labels along a unique path from the root.

Example 4: $T = aabcab$, $P = ab$

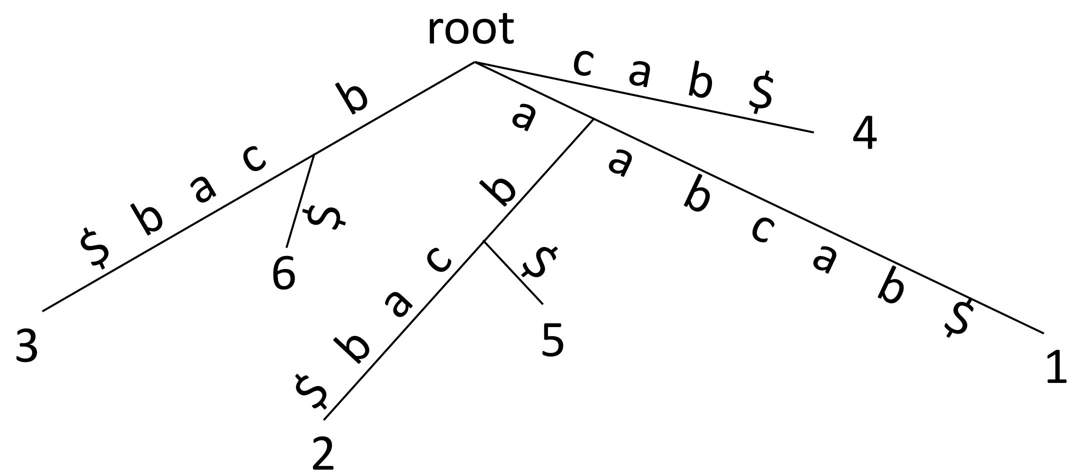

Figure 4: Example 4

If we exhaust all the characters of P and come to a node then all the leaves rooted at that node correspond to matches of P. If we have not exhausted all the characters of P and at some point we cannot match any more characters of P, then P is not a substring of T.

In the above example, the subtree rooted at the node whose path label is ab has suffixes 2 and 5 in its leaves. Each of these suffixes has ab as a prefix and hence corresponds to a match of the pattern P.

Time to traverse the subtree is $O(k)$, where k is the number of matches of P in T.
So the total time $= O(m + n + k)$.

4 Problem 2: Exact set matching

Input: a text $T = x_1 x_2 \cdots x_m$ and a set $P = \{P_1, P_2, ..., P_q\}$ of patterns
Output: All the occurrence of all the patterns in T
Algorithm: Build a suffix tree for T in $O(m)$ time. Use the algorithm for Problem 1 for each pattern separately. Let $|P_i| = n_i$ and the number of occurrences of P_i in T be k_i, $1 \leq i \leq q$.
Then the time for search $= O(\sum_{i=1}^{q} n_i + \sum_{i=1}^{q} k_i)$.
So, the total run time $= O(m + N + K)$, where $N = \sum_{i=1}^{q} n_i$ and $K = \sum_{i=1}^{q} k_i$.