Big Data Lecture Notes on March 6

Problem: 3

Input:

(1) A database DB of texts = $\{T_1, T_2, \ldots, T_k\}$

(2) A set of patterns = $\{P_1, P_2, \ldots, P_q\}$ where

$$\sum_{i=1}^{k} |T_i| = M \text{ and } \sum_{i=1}^{q} |P_i| = N$$

Output:

For each pattern output its occurrences in DB.

Algorithm:

(1) Build a generalized suffix tree Q on the strings in DB.

(2) for $1 \le i \le q$ do

Match the characters of P_i with a unique path in Q starting from root.

(3) If we match all the characters of P_i and came to a node u, the leaves in the sub-tree rooted at u have all the occurrences of P_i; if we cannot match all the characters of P_i then P_i does not occur in any text of DB.

Analysis:

(1) Time to build Q = O(M)

(2) Time to search for P_i = O($|P_i| + k_i$) where k_i = # of occurrences of P_i in DB.

So, the total runtime = O($M + N + K$) where $K = \sum_{i=1}^{q} k_i$ □

Problem: 4

Input:

Two strings S_1 and S_2.

Output:

The longest common substring between S_1 and S_2.

Example:

S_1 = "desperate"

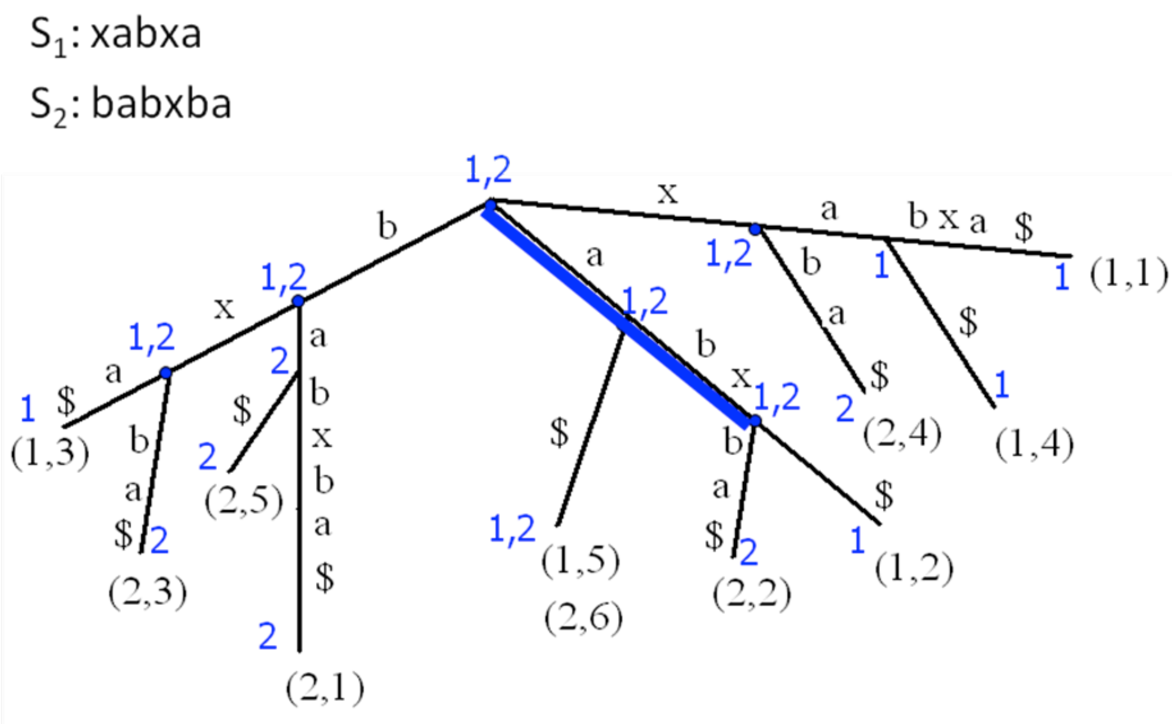
S_2 = "lifespan"

Longest common substring = "esp"

$S_1 = a_1a_2\ldots a_i\ldots a_{n1}$.

$S_2 = b_1b_2\ldots b_j\ldots b_{n2}$.

A trivial algorithm: $\forall_{i,j}$ compute the longest common substring starting at i of S_1 and j of S_2.



Fact:

We can solve this problem in O(M) time where $M = n_1 + n_2$

Algorithm:

(1) Build a generalized suffix tree Q of S_1 and S_2.

(2) Mark nodes in Q as follows:

A node u will be marked with 1 if the sub-tree rooted at u has at least one leaf corresponding to a suffix from S_1. A node u will be marked with 2 if the sub-tree rooted at u has at least one leaf corresponding to a suffix from S_2. This marking can be done through a traversal of the tree in O(M) time. If u is marked with both 1 and 2, then the path label of u is common to S_1 and S_2.

(3) Do another traversal in the tree Q to identify the node with the largest string depth that is marked with 1 and 2. This takes O(M) time. So, the total runtime = O(M).

Problem: 5

Input:

Two strings S_1 and S_2 and an integer l.

Output:

Occurrences of substrings of S_2 of length $\geq l$ in S_1.

Algorithm:

The same as before except that we look for all the nodes marked with 1 and 2 whose string depth is $\geq l$.

Problem: 5(a)

Input:

String S_1 and a collection of strings $C_1, C_2, \ldots, C_q$.

Output:

Occurrences of substrings of C_i of length $\geq l$ in S_1, for $1 \leq i \leq q$.

Fact:

This can be solved in O(M) time where $M = |S_1| + \sum_{i=1}^{q} |C_i|$

Idea:

(1) Build a generalized suffix tree Q for $C_1, C_2, \ldots, C_q$.

(2) Mark a node u if the sub-tree rooted at u has a leaf from S_1 and a leaf from at least one of $C_1, C_2, \ldots, C_q$. Report all the marked nodes of depth $\geq l$.

Problem: 6

Input:

A set of strings $S_1, S_2, \ldots, S_n$

Output:

$l[2:n]$ such that $l(i)$ = length of the longest common substring that occurs in at least i strings $2 \leq i \leq n$.

Example:

"length", "strength", "english", "substring", "link"

Here: $l(2) = 5$

Fact:

We can solve this problem in O(Mn) time.

Proof:

(1) Build a generalized suffix tree Q for the given set of strings.

(2) For any node u in Q, let $c[u]$ = # of distinct strings represented in the leaves of the subtree rooted at u. Keep an array $v[2:n]$. Traverse tree Q. At the end of each traversal, $v[i]$ is the largest string depth of any node whose $c[.]$ value is i.

(3) Note that $v[i]$ is the length of the longest common substring that occurs in exactly i strings.

(4) To compute the $l(i)$ values do a prefix maxima operation on $v[n], v[n-1], \ldots, v[2]$. □

Computation of *c[.]* array:

Keep a bit array $u[1:n]$ for each node u in Q.

$u[i] = 1$ if one of the leaves of the sub-tree rooted at u corresponds to a suffix of S_i (where $1 \le i \le n$).

Do an inorder traversal of the tree. The bit array for any node u is the OR of the bit arrays of its children.

$c[u]$ = # of 1's in $u[1:n]$.

Total runtime = $O(Mn)$ □

Problem: 7

All pairs suffix-prefix computation.

Problem*

Input:

$S_1, S_2, \ldots, S_n$; $\sum_{i=1}^{k} |S_i| = M$

Output:

For every ordered pair (i, j), the length of the largest suffix of S_i which is a prefix of S_j.