CSE 4502/5717: Big Data Analytics

Notes by Ashily Varghese

Lecture 11 (2/28/2018)

SUFFIX TREE

FACT: For any given string of length m, we can construct a suffix tree in O (m) time (UKKONEN, 1992).

FACT: There is an easy algorithm that takes O (m^2) time.

PROOF:

Let T = t_1 t_2 t_3 ..... t_m

For i = 1 to n do

Let R_{i-1} be the tree containing the suffixes S_1, S_2, …., S_{i-1};

Insert S_i into R_{i-1} to get R_i as follows:

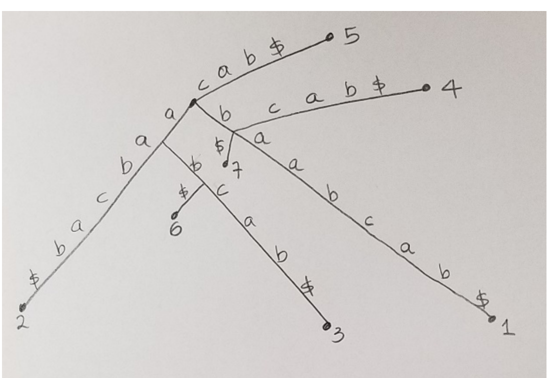
Start matching the characters of S_i with labels of edges starting from the root.

We will come to a point where no more characters can be matched.

If this happens at a node u in R_{i-1}, then create a new child for u with an edge whose label will be the remaining characters of S_i.

If this happens in the middle of an edge, split the edge and create a new node as before.

Example: T = baabcab$



Definition:

1. The label of a path is the ordered concatenation of the edge labels in the path.

2. The path label of a node is the label of the path from the root to that node.

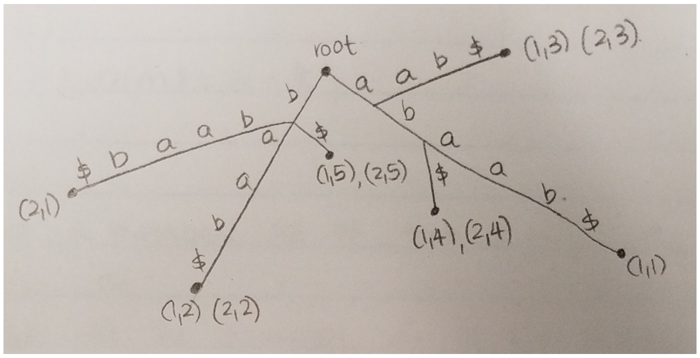
3. The string depth of a node is the number of characters in its path label.

A GENERALIZED SUFFIX TREE

Let $S_1, S_2, \ldots, S_k$ be strings from an alphabet Σ

A generalized suffix tree on $S_1, S_2, \ldots, S_k$ is a tree Q in which there is a leaf for every suffix of every string. A leaf is labelled with (i,j) where i is the string ID and j is the suffix number within this string.

Example: S_1 = abaab\$, S_2 = bbaab\$



FACT: We can construct a generalized suffix tree on $S_1, S_2, \ldots, S_k$ in $O(\sum_{i=1}^{k} |S_i|)$ time.

One Idea: Construct a suffix tree on $S_1 \$_1 S_2 \$_2 \ldots S_k, \$_k$ and eliminate unwanted paths.

Problem1

INPUT:

$T = t_1 t_2 \ldots t_m$ <-- TEXT

$P = p_1 p_2 \ldots p_n$ <-- PATTERN

OUTPUT: All occurrences of P in T.

Example:

T = b a a b a b a a b b

P = a a b

There are two occurrences of P in T (starting from positions 2 and 7).

A simple algorithm takes O (mn) time.

Claim: We can solve this problem in O (m+n+ k) time using a suffix tree, where k is the number of matches.

Algorithm:

1. Construct a suffix tree Q for T;

2. Start matching the characters of P starting from the root. If we are able to match all the characters and end up at a mode u, then all the leaves in the subtree rooted at u correspond to matches.

On the other hand, if we are not able to match all the characters of P, then P does not occur in T.

Problem 2

INPUT: T; P_1, P_2, …., P_q

OUTPUT: All the occurrences of all the patterns in T.

Clam: We can solve this in O (m + N + k) time, where

m = |T|;

N = $\sum_{(i=1 \text{ to } q)} |P_i|$ and K = $\sum_{(i=1 \text{ to } q)} k_i$, where k_i is the number of occurrences of P_i in T, 1<= i <= q.

Idea: Construct a suffix tree Q for T,

for 1<= i <= q do

Use the previous algorithm to find the occurrences of P_i in T.

Run time = O (m) + O ($\sum_{(i=1 \text{ to } q)}$ ($|P_i|$ + $|k_i|$)) = O (m + N + K).

Problem 3

INPUT: A database DB of texts T_1, T_2, …, T_q and patterns P_1, P_2, …., P_n

OUTPUT: All the occurrences of all the patterns in the DB.

FACT: We can solve this in O (M + N + K) time, where M = $\sum_{(i=1 \text{ to } q)} |T_i|$, N = $\sum_{(i=1 \text{ to } n)} |P_i|$ and

K = $\sum_{(i=1 \text{ to } n)} k_i$, where k_i is the number of occurrences of P_i in the DB.

IDEA: Construct a generalize suffix tree Q for T_1, T_2, …, T_q

For 1 <= I <= n do

Find the occurrences of P_i in Q;

Problem 4

The longest common substring problem

INPUT: Two strings S_1 and S_2

OUTPUT: The longest common substring between S_1 and S_2

Example:

S_1 = identical

S_2 = dentist

Longest common substring = denti

$S_1 = a_1\ a_2 \ldots . a_i \ldots a_n$

$S_2 = b_1\ b_2 \ldots . b_j \ldots . b_m$

One simple Algorithm:

For all i, j do: Identify the longest substrings starting from a_i in S_1 and b_j in S_2 that match.

Total runtime = O (n^3).

FACT: We can solve this problem in O (m + n) time using a suffix tree.

Proof: An Algorithm:

1. Construct a generalized suffix tree on S_1 and S_2

2. With a tree traversal label any node u of Q with 1 if the subtree rooted at u has a leaf corresponding to a suffix of S_1;

Label any node u of Q with 2 if the subtree rooted at u has a leaf corresponding to a suffix of S_2;

If a node u has labels 1 and 2 then its path label is common to S_1 and S_2.

Thus a node with labels 1 and 2 and whose string depth is the largest will give us the answer.

More details will be provided in the next lecture.