566-HW1

Ömer Faruk Özdemir – ofo5108

February 2023

1

We don't need to define spm_p, because in the matching algorithm it is never used, as there could not be a mismatch at p+1 th character since the length of the string is p.

2

There are $|S|$ many leaf nodes, as each leaf node represents a suffix and there is only one leaf node for every suffix.
According to the creation of suffix tree every node has at most 1 incoming edge. If a node is an inner node, it means it was branched out at some point. Hence it has at least 2 outgoing edges and each branching creates a leaf node.
$N = |Leaf| > |IN| - - - > |Leaf| + |IN| + 1(root) = O(N)$
There can be constant edges out of each node. Hence the number of edges is O(N) as well.

3

A leaf node represents a suffix starting at index i, and each suffix has a suffix link to the suffix starting at index i+1. (Empty string suffix will point to the root)
A node in the suffix tree represents a prefix of a suffix. If you remove the first character, it is still a prefix of the suffix starting at i+1, hence it will have a suffix link pointing to there.

The same proof is valid for suffix trie and tree.

More info:
In the suffix tree, the landing position will be a node as well.

If the source is a leaf node, the landing node is a leaf node too(or root). xP is a suffix hence P is a suffix too.

If the source is an intermediate node, the landing position will be an intermediate node as well(or root). Source node branched and became an intermediate node at some point, and this is also valid for the destination due to their relationship(xS - S).

4

Create a suffix tree using Ukkonen algo in linear time. Traverse the tree and Count number of prefixes of each suffix on the way. Each path on the tree creates distinct substrings. There are O(n) number of nodes and edges hence traverse takes linear time as well.
ex:
$|S| = 8$
[1,2] edge contributes 1
[3,*] edge contributes 6 to the count.

5

We can use a generalized suffix tree. With y many strings we would have a length of M+y gen-string. M+y is at most 2.M hence O(M)

Create a suffix tree in linear time.
for each s in S:
search substring s in the tree, (we will always find one match, s, hence ignore 1 of the matches)
a. if you ever find a # without scanning the whole substring, you fail
b. if you find a substring, increase the count

6

Create the suffix tree in linear time. While creating the suffix tree, save character in incoming suffix link on nodes(x in xS).

1. Traverse the tree,
2. If all descendant leaves under a node has the same character in their incoming suffix link(x in xS), then include s(u) or u into left-aligned nodes/substrings.

The complexity is O(n) because there are O(n) many nodes.

The algorithm is correct because all descendant leaves under a node represent all suffixes starting with s(u), which represents all s(u) occurences in the string.

7

a. Create the string w = t$s. Create Z values in w. $O(|t| + |s|)$

```
for i in (0,|s|-1):
  if Z[|t|+1+i] == |s| - i:
    return s[Z[|t|+1+i]:]
return ""
```

Z_i is the longest prefix which starts at ith position. When the Z value equals to suffix length, we've found the the longest suffix prefix intersection, because all further suffixes are shorter.
$Complexity : O(|t| + |s|)$

b. Create a suffix tree from s in linear time.
Start a search for t from root, find all reachable suffixes on the way and update the string(LCPS). Each found string is a suffix of s and a prefix of t.
$Complexity : O(|s| + |t|)$