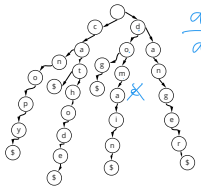
Problem 1. Draw a prefix tre containing the strings `cat`, `cathode`, `canopy`, `dog`, `danger`, `domain`. Terminate each string with a $ character to indicate the ends of words.



done
dog

Problem 2. Draw a suffix tree for the string `ABAABA$`.

$
A$
BA$
ABA$
AABA$
BAABA$
ABAABA$

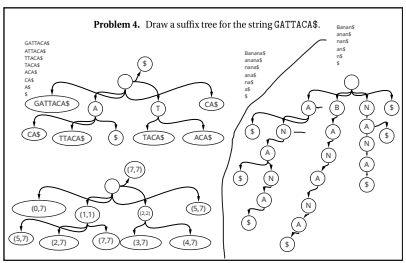
Number of characters bounded by n^2

(index,length)
or
(start_index, end_index)

Problem 3. Describe an algorithm that given a sequence of strings over a constant-size alphabet, counts how many different ones there are (i.e. how many there are ignoring duplicates). Your algorithm should run in $O(T)$ time where T is the total length of all of the strings.

Counting all the "$" in the trie will give the number of non-duplicate strings

Problem 4. Draw a suffix tree for the string `GATTACA$`.



Problem 5. Describe an algorithm that given a string S of length n, counts the number of distinct substrings of S in $O(n)$ time.

Construct suffix-tree for the entire word, using the O(n) algorithm.

Next count all the lengths contained in each tree node

Eg BANANA

Problem 6. Earlier, in Tutorial 4, we learned about the longest common **subsequence** problem. Now we consider a similar, related problem, the *longest common substring* problem. Given two strings s_1 and s_2, your goal is to find a string that is a substring of both s_1 and s_2 and is as long as possible. Describe how a suffix tree can be used to solve this problem. Your algorithm should run in $O(n+m)$ time, where n, m are the lengths of s_1, s_2 respectively.

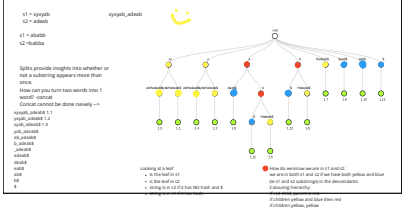
s1 = xysqab
s2 = adbab

s1 = abab$b
s2 =babba

Splits provide insights into whether or not a substring appears more than once.

How can you turn two words into 1 word? -concat
Concat cannot be done naively —>

xysqab_adbab$ 1.1
ysqab_adbab$ 1.2
sqab_adbab$ 1.3
qab_adbab$
ab_adbab$
b_adbab$
_adbab$
adbab$
dbab$
bab$
ab$
b$
$


xysqab
adbab
root
$
b
a
xysqab
adbab
xysqab
adbab
xysqab
adbab
xysqab
adbab
xysqab
adbab
xysqab
adbab
xysqab
adbab
xysqab
adbab
b
a
b
a
b
a