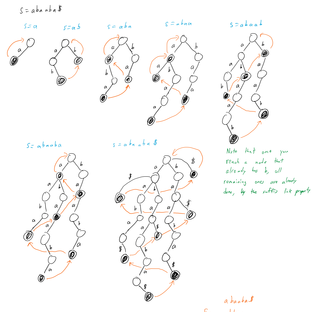
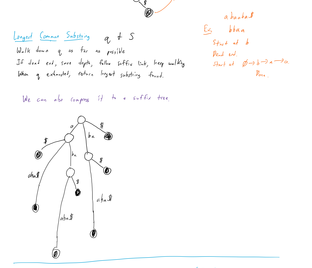
Lec17-more-suffix-trees-arrays

Sunday, October 20, 2024 10:14 PM

Last time we learned about suffix tries & trees
Lately learned concepts of suffix tree construction
We build it up while walking the string from left to right

S = abababa$





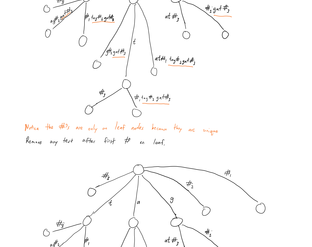
Note that once you reach a node that already has $, all remaining ones are already done, by the suffix link property

Longest Common Substring q, t ∈ S

While done q in for as possible
If dest exit, save depth, follow suffix link, keep walking
When q exhausted, return longest substring found.

Ex: abaaba$
bbaa
Start at b
Find no
Start at g→b→a→ba
Ans.

We can also compress it to a suffix tree



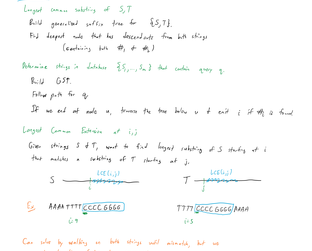
Generalized suffix trees to multiple strings (GST)

Goal: represent set of strings $P = \{S_1, S_2, \ldots, S_n\}$

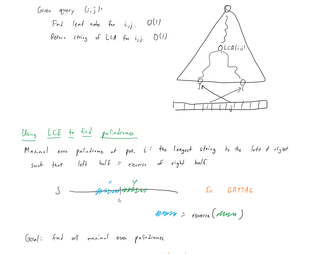
Ex: aabf, dbg, gaf

unique string terminator

Sol: Build suffix trees for $aabf\$_1, dbg\$_2, gaf\$_3$



Notice the edges are only on first nodes because they are unique
Having any third within first $ is fixed.



When building suffix tree, can just stop on any end-of-string character $

Applications

Longest common substring of S,T

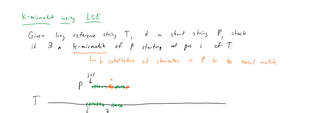
Build generalized suffix tree for $\{S, T\}$.
Find deepest node that has descendants from both strings
(ancestors with $\#_1$ & $\#_2$)

Pattern matching in database $\{S_1, \ldots, S_n\}$ that enables query q.

Build GST.
Follow path for q.
If we exit at node v, traverse the tree below v if exit ∈ if $\#_i$ is found

Longest Common Substrings of i,j

Given strings S & T, want to find longest substring of S starting at i that matches a substring of T starting at j.



Ex: AAAATTTTCCCCGGGG TTTTCCCCGGGGAAAA
i=5 j=5

Can solve by walking on both strings until mismatch, but we want to do this faster by preprocessing.

Sol: Build GST for S, T. O(|S| + |T|) time

Preprocess tree so that lowest common ancestor can be found in constant time.

∃ $O(|S|+|T|)$ time algorithm that does $O(1)$ LCA queries

We are not covering LCA in DSA, but could implement this structure that does $O(|S|+|T|)$ time to build initially.

Create array mapping suffix numbers to leaf nodes. $O(|S| + |T|)$ time

Given query (i,j):
Find leaf node for $S_i j_j$ O(1)
Return string depth of LCA for i,j O(1)

Using LCE to find palindromes

Maximal even palindrome at pos i: the longest string to the left & right such that left half = reverse of right half.

S = GATTAC

S_{rev} = reverse(S)

Goal: find all maximal even palindromes

Construct $S' = S + S_{rev}$ O(|S|)

Preprocess S & S' in LCE query table O(1) time O(|S|)

LCE(i, n+1) = length of longest palindrome centered at i

For i: O(|S|)
Compute LCE(i, n+1) O(1)
O(|S|)

k-mismatch using LCE

Given long reference string T, & a short string P, check if P is a k-mismatch of P starting at pos i of T.

= substitutions of characters in P to be exact match

j=1 pos in P
c=0 # mismatches so far

O(k)

Repeat until c>k:
  j = j + LCE(i, j) + 1
  i = i + LCE(i, j) + 1
  If j > |P|+1, return True
  c = c+1
return False

O(1) to get LCE(i,j)

matched all of P

Really all k-mismatches takes $O(k|T|)$ time

Suffix arrays

Suffix trees are $O(n)$ space, but the constant is large
(≥ 20 bytes per char in good implementation)

10 Gb genome → 200 GB to store

Suffix arrays are more efficient even than suffix trees & be most of the same ops.

Idea: Sort all suffixes of string lexicographically & keep the indices.

S = cattcat$

1	cattcat$
2	attcat$
3	ttcat$
4	tcat$
5	cat$
6	at$
7	t$
8	$

sorted

8	$
6	at$
2	attcat$
5	cat$
1	cattcat$
7	t$
4	tcat$
3	ttcat$

Search substring in binary search of suffixes.

Could: a map between suffixes starting with "ca" are next to each other.

Hyp: binary search for all those.

Only keep index & weight not S

Construction: $O(n^2 \log n)$ by sorting suffixes where each comparison takes $O(n)$ time.

But better $O(n)$ algorithms such as building suffix tree.

$\sum = \{\$, a, c, t\}$

S = cattcat$
1234567 8

To copy sorted lexicographically leaf labels are exactly suffix array