Algorithms in Bioinformatics

More suffix trees

Inexact Matching

Recap

- Office hours today start at 4 (plus I'll leave a little time at the end of today for questions)
- Next assignment: essay
  - Officially due Thursday (after break), but feel free to submit the next Tuesday if you need more time

- No class in 1 week

Last time(s): Exact matching

We focused on:

- hashing : import something
- suffix trees

There are many variants, but key tradeoff:

fixed P vs fixed T

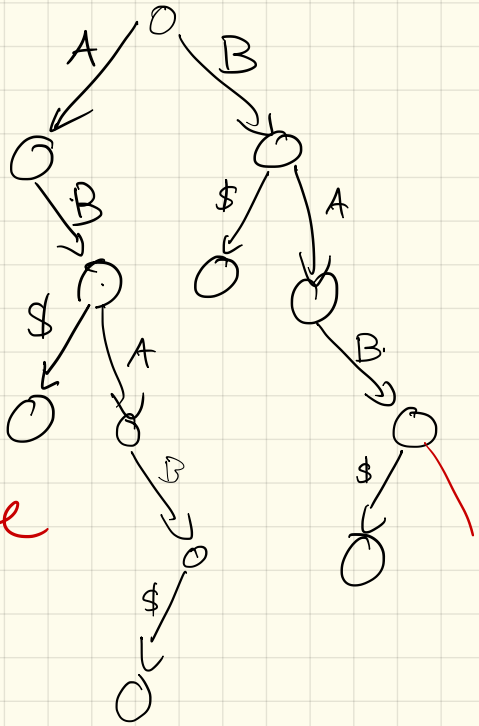
More suffix trees:

- Generalized suffix trees: a tree that recognizes suffixes for > 1 string

Idea:

- Build suffix tree for S_1

Eg: $S_1 = ABABA\$$



Then parse for S_2.

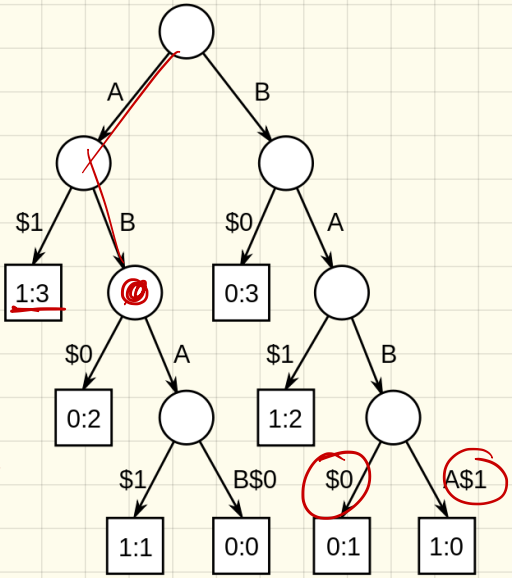
$S_2 = BABA*$

$O(|S_2|)$ time to add to tree

Final tree:

S_1 = BABA ($0)

S_2 = ABAB ($1)



3 is index of suffix in string 1

(This can be done efficiently, but we won't unpack those details yet...)

Can actually do this for even more strings:

- Make each terminate with a different end character. ($1, $2, $3)
- Annotate tree as you go
  - ↳ encodes all suffixes of all ~~trees~~ strings in a single tree.

More applications:

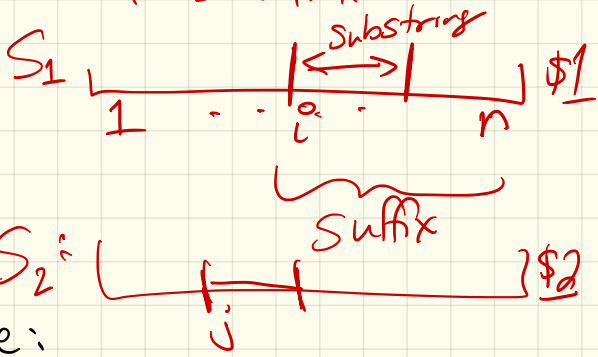
- Longest common substring
  ↳ contiguous set of characters (not sequence)

How?

Use generalized suffix tree.

Mark internal nodes:

a substring in S_1 is a prefix of a suffix.



Runtime:

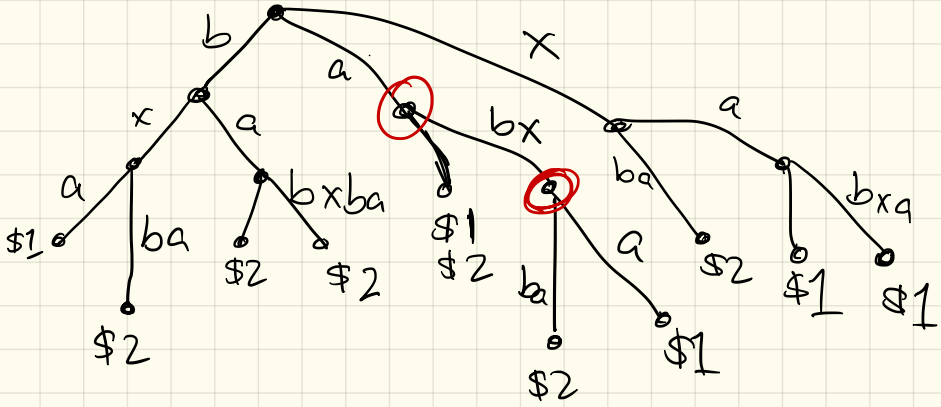
Linear time
(to build gen. suffix tree)

Example:

$S_1 = xabxa\ (\$1)$

$S_2 = babxba\ (\$2)$

Tree:



Any internal node with both $1 + $2 at leaves in its subtree is a common substring. Return the deepest.

- DNA contamination problem:

(Fun dinosaur story)

Problem: Given string S_1 & a known string S_2, find all substrings of S_2 that occur in S_1 and are longer than some length ℓ.

(These are candidates for unwanted pieces of S_2 that have contaminated S_1.)

How? (Similar to last one!)

Build gen. suffix tree.

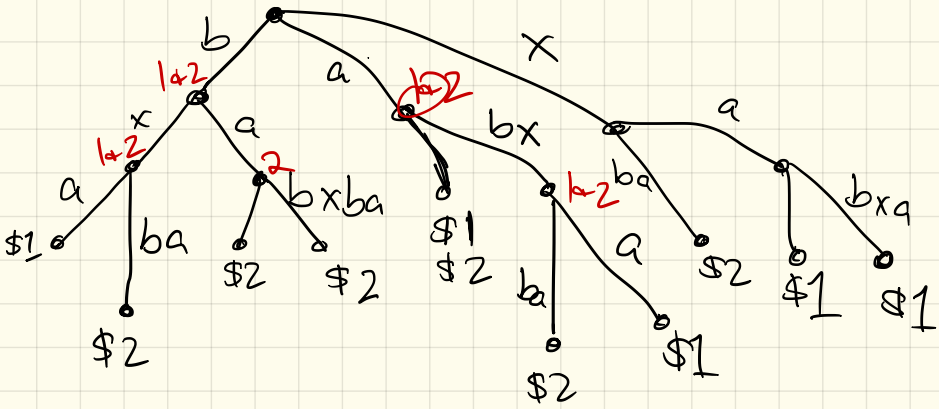
$O(n)$

Same Example:

$S_1 = xabxa\ (\$1)$ unknown DNA

$S_2 = babxba\ (\$2)$ my DNA

Tree:



Set $\ell = 2$:

Look for any nodes of depth ≥ 2.

Longest Common Extensions:

Input: Two strings S_1 + S_2

$|S_1|, |S_2| = n$

and long sequence of index pairs (i, j)

Output:

For each (i, j), return length of longest common prefix of $S_1(i \ldots n)$ matching a prefix of $S_2(j \ldots n)$

How?

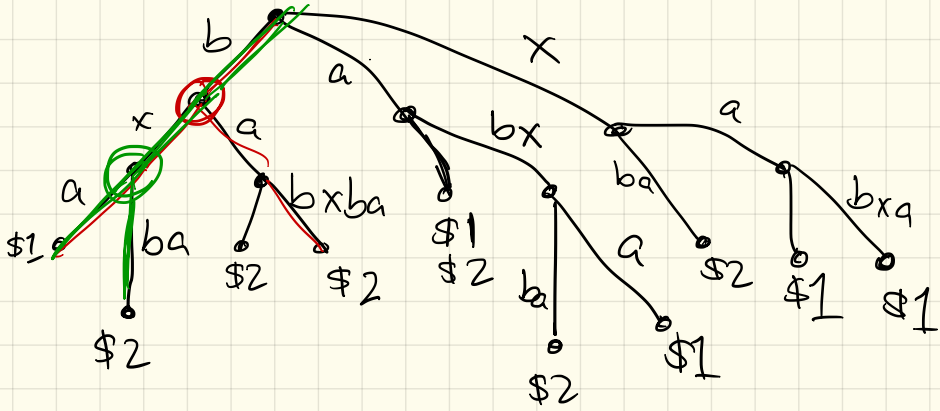
- Use lowest common ancestor in gen. suffix tree:

Some Example:

$S_1 = xabxa$ ($1)

$S_2 = babxba$ ($2)

Tree:



Let $i = 3$ & $j = 1$

1 : b

or $i = 3$ & $j = 3$

2 : bx

Inexact Matching:

the k-mismatch problem

Idea: For some simpler variations on matching, don't need full-on dynamic programming.

Why bother?

Not going to allow insertions or deletions

~O(n) instead of O(mn)

Problem:

K-mismatch Problem:

~~**Approximate Pattern Matching Problem**~~:
Find all approximate occurrences of a pattern in a text.

Input: A pattern $\mathbf{p} = p_1 p_2 \dots p_n$, text $\mathbf{t} = t_1 t_2 \dots t_m$, and parameter k, the maximum number of mismatches.

Output: All positions $1 \leq i \leq m - n + 1$ such that $t_i t_{i+1} \dots t_{i+n-1}$ and $p_1 p_2 \dots p_n$ have at most k mismatches (i.e., $d_H(\mathbf{t}_i, \mathbf{p}) \leq k$).

Ex: P = bend

T = abentbananaend

k = 2

Then 3 2-mismatches:

- at position 2,
  bent has 1 mismatch

at pos 6,
bana w/ 2 mismatches

at pos 11,
aend
↕
bend
1 mismatch

First approach: brute force:

```
APPROXIMATEPATTERNMATCHING(p, t, k)
1  n ← length of pattern p
2  m ← length of text t
3  for i ← 1 to m − n + 1
4       dist ← 0
5       for j ← 1 to n
6            if t_{i+j−1} ≠ p_j
7                 dist ← dist + 1
8       if dist ≤ k
9            output i
```

Runtime: $O(mn)$

$O((m-n+1)n)$

Goal: $O(km)$

where $|P| = n$

$|T| = m$ (& $m > n$)

→ instead of $O(mn)$

Method: use longest common extension idea

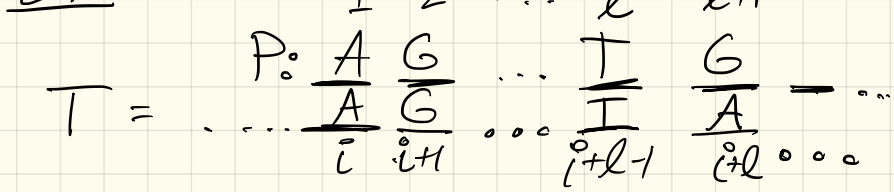
For any position i in T (the longer string), do a longest common extention query.

Say it has length ℓ.

if $\ell = |P|$, done.

otherwise, position $i+\ell$ is a mismath:

Ex:



Then use a mismatch (so $k-1$ left), reset to $T[i+\ell+1]$ & $P[\ell+2]$ & continue, looking for $k-1$ match

Runtime: $O(km)$