Algorithms in Bioinformatics

Exact pattern matching (cont)

Recap

- Email me if you're coming by office hours tomorrow
- HW due Tuesday
- Longer "Midterm" assignment is posted (comments welcome!)

Today: Pattern Matching

The Boyer-Moore Algorithm:

Align P against T, & start comparing at end of T:

(1) Character mismatch: get to index i with $T[i] \neq P[j]$ even though aligned

Say $T[i] = c$:

(a) if c is in P, shift until a copy of c is aligned

(b) if c is not in P, shift past to $i+1$

Preprocessing: Bad character rule

We can precalculate the skips for mismatches to speed things up:

Say $\Sigma = \{A, C, T, G\}$

and $P = TCGC$

space: $O(|P||\Sigma|)$ but can improve...

P ↓ (Bug!)

	T	C	G	C
A	0	1	2	3
C	0	-	~~0~~ 2	-
G	0	1	-	0
T	-	0	1	2

Σ

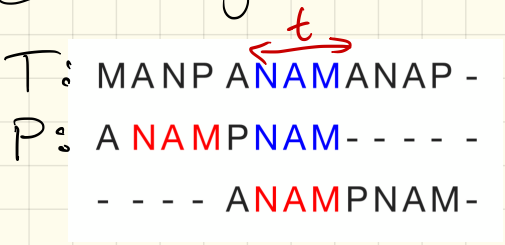
P: TCGC
T: AATCAATAGC
P: TCGC

T: AATCCA..
P: TCGC

Store: for each position i in P and $x \in \Sigma$, the closest occurance of x in P to the left of i.

(2) Good suffix rule:

Given alignment of T+P:



If t is the longest suffix of P that matches T in current position, we can shift P so that it matches t earlier in P.

(In fact, character before prev. occurance of t in P should be different from character before the suffix of P=t.)

If none:

- Find smallest shift that matches prefix of P to suffix of P & shift.
- Or else shift down by |P|.

Preprocessing for good suffix:

- For each i, $L(i)$ is largest position $< i$ such that $P[i..n]$ matches a suffix of $P[1..L(i)]$.

Ex: $P = CAGTAGTAG$

$L(8) = 6$

$L'(8) = 3$

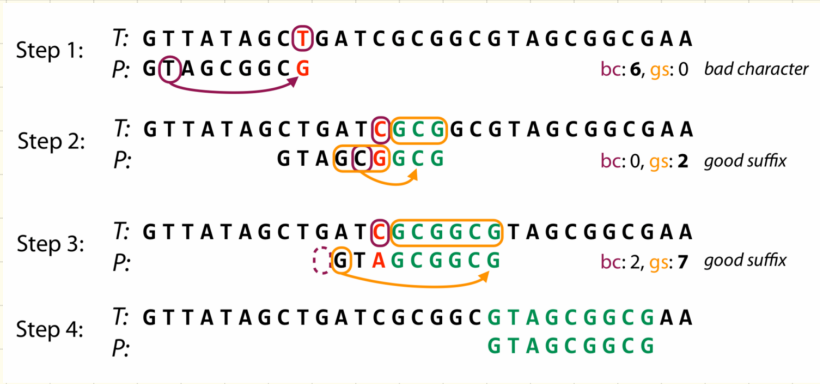
- $L'(i)$ is largest position $< |P|$ such that $P[i..n]$ matches a suffix of $P[1..L'(i)]$ and preceding character of suffix is $\neq P(i-1)$

Use these to shift!

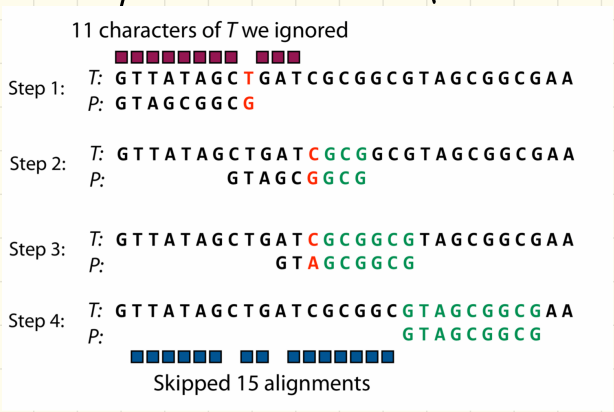
The algorithm:

Use bad character or good suffix (whichever gives bigger shift)

Example:



Takeaway: We skipped a lot!



Runtime: $O(m+n)$

Why?

- Preprocessing for BC:

  Scan t for each x, find all positions where x occurs.

Ex: P: a b a c b a b c (O(|P|) to scan)

$x = a$: 6, 3, 1
b : 7, 5, 2
c : 8, 4

} $O(n)$

Then: at mismatch, scan list until get # < i

- Good suffix rule:

  Similar trick.

Then: trade-off is the key!

Next: KMP

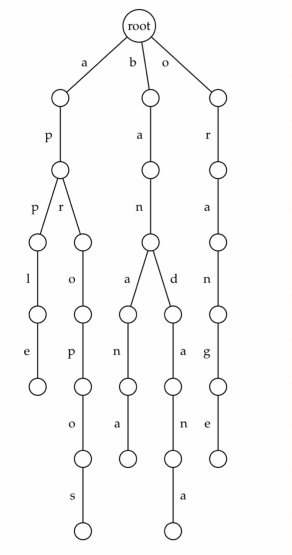
Prefix-free trees: (tries)

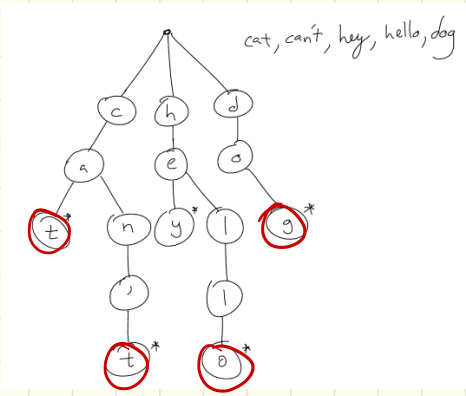
Consider strings S from an alphabet Σ.

Build a tree:

- Every node (except root) gets a label from Σ.
- order children of a node in alphabetical order
- There are $|S|$ leaves, and that each root-to-leaf path to a leaf gives a (unique) string from S.

Ex:





Then:

Multiple Pattern Matching Problem:
Given a set of patterns and a text, find all occurrences of any of the patterns in the text.

Input: A set of k patterns $\mathbf{p^1}, \mathbf{p^2}, \ldots, \mathbf{p^k}$ and text $\mathbf{t} = t_1 \ldots t_m$.

Output: All positions $1 \leq i \leq m$ such that a substring of $\mathbf{t}$ starting at position i coincides with a pattern $\mathbf{p^j}$ for $1 \leq j \leq k$.

How? Use trie

For each position i:

Scan T[i] + ...

Move in tree

at a leaf, output yes!

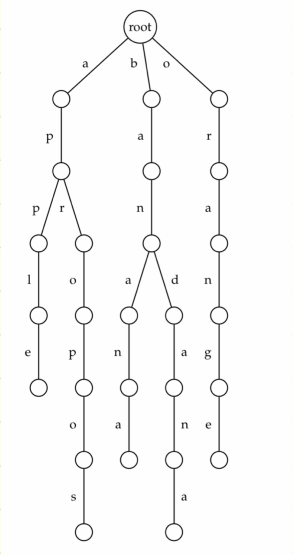
increment i

Runtime: build trie

search: $O(m \cdot \max\{p_i\})$

Compressed tries:

Tries are not space-efficient:

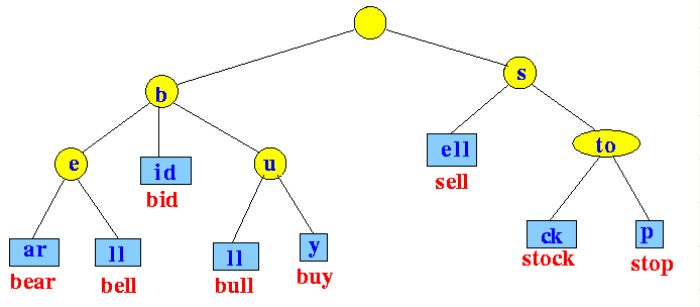


Problem:

redundant nodes!

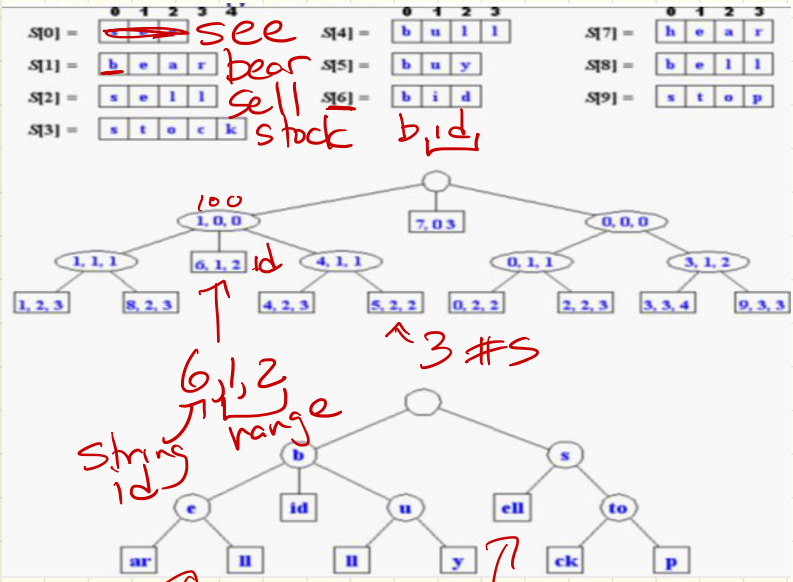
degree 2 nodes contain nothing new

Goal: Make each node have ≥ 2 children



Why? Seems like more space in single node.

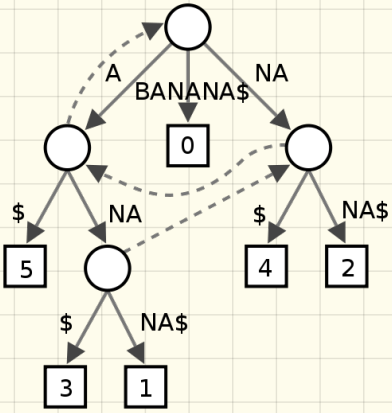
Strength is really when you store the strings separately & use the trie to index.



Suffix tries:

Suppose all strings in S are suffixes of some X.

Can get better representation:



BANANA:

1 2 3 4 5 6

6 suffixes

Why? Space!

How many suffixes are there?

$1+2+3+\cdots+(n-1) = O(n^2)$

How big is this tree?

$O(n)$

Suffix trie vs keyword!

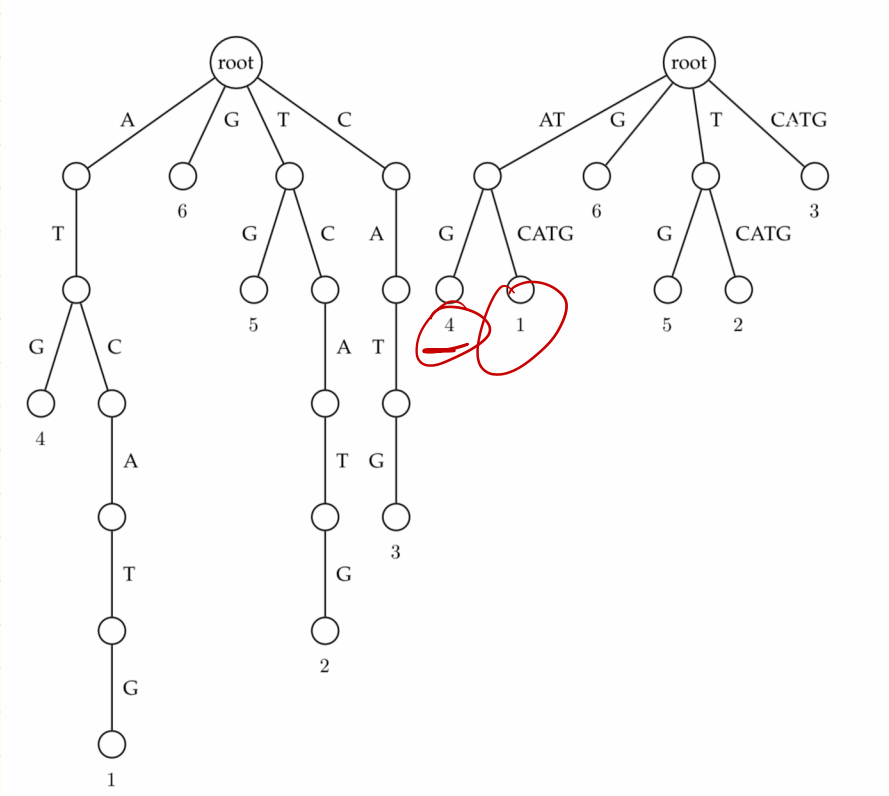
String: ATCATG

4

1

Keyword trie:

Suffix trie:



How to use for exact pattern matching:

```
SUFFIXTREEPATTERNMATCHING(p, t)
1  Build the suffix tree for text t
2  Thread pattern p through the suffix tree.
3  if threading is complete
4      output positions of every p-matching leaf in the tree
5  else
6      output "pattern does not appear anywhere in the text"
```

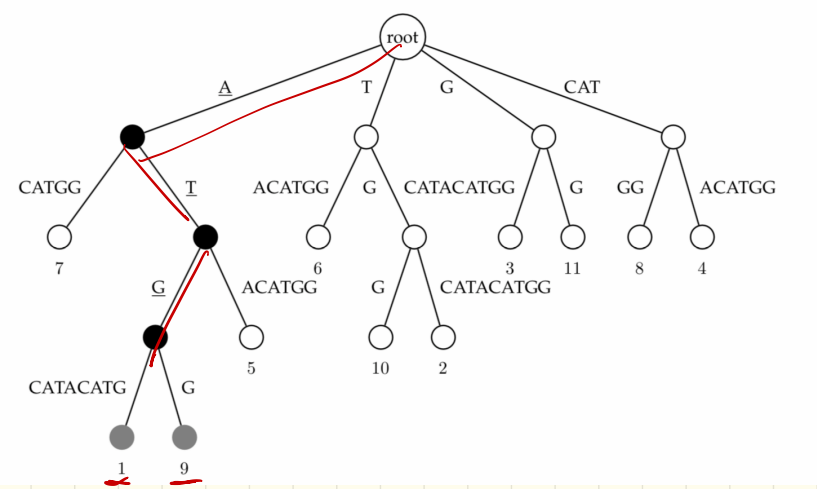
black box

Example: P = ATG
T = ATGCATACATGG

threading: start at root & trace P

p-matching: leaves below

Suffix tree:



Difference: trade-off

- Earlier BM alg:
  vary T, but for one pattern P

- Suffix trie:
  preprocess T (fixed)
  search for any patterns quickly

Next time:
inexact matching