Strings Matching:

- Tries
- Compressed tries
- suffix trees & arrays.

String matching: given

Text T & Patten P
↳ strings over Σ (Alphabet)

Find some or all occurrences of P in T as substrings

SICOMP 1997

- one shot: $O(T)$ time [Knuth-Morris-Pratt, Boyer-Moore CACM 1977, KARP-Rabin IBM JRD 1987]
- Static DS: preprocess T, query P
  - goal: $O(P)$ query, $O(T)$ space
- other DSs consider when P has wildcards or when P need not match as an exact substring (Hamming / edit distance)

STOC 2004

~ see eg: [Cole, Gottlieb, Levenstein, Maaß & Nowak - CPM 2005]

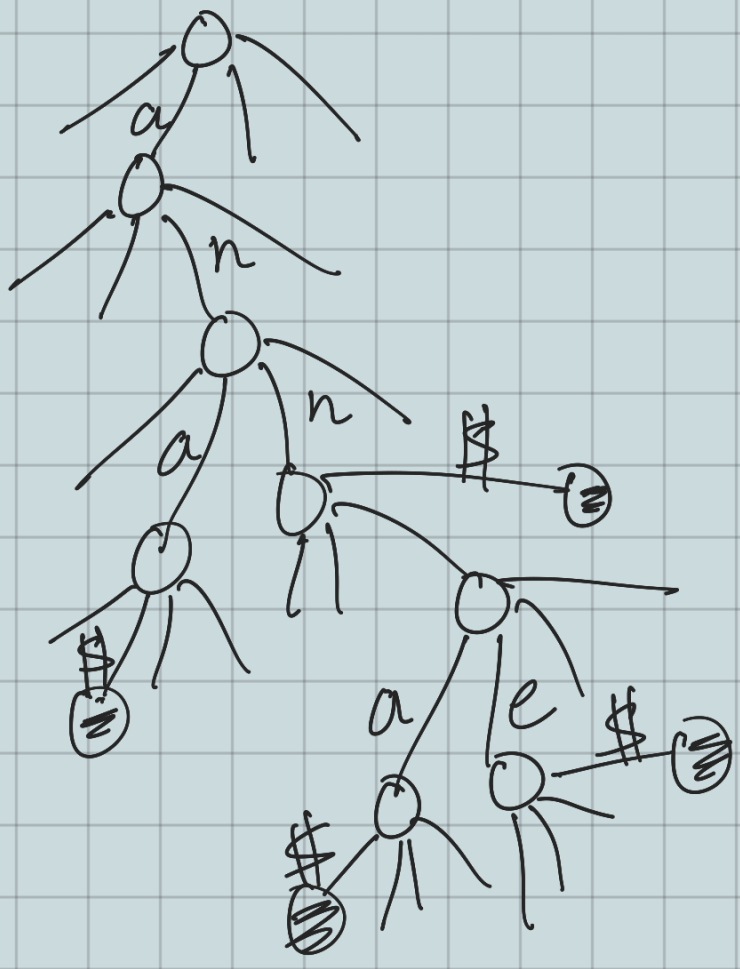
Warm up: Prefix among strings.
$(T_1 \ldots T_k)$ e.g. (library search)

Trie: rooted tree with child branches labeled with letters in Σ.

- To represent strings as root-to-leaf paths in a trie.
- Add a new letter \$ to end of each string.

(otherwise can't distinguish prefixes as absent or present)

e.g. {Ana, ann, anna, anne}



Trie representation:

T = # nodes in trie $\leq \sum_{i=1}^{K} |T_i|$

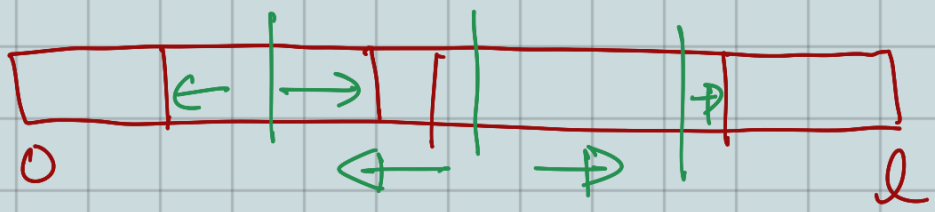
[node store children]

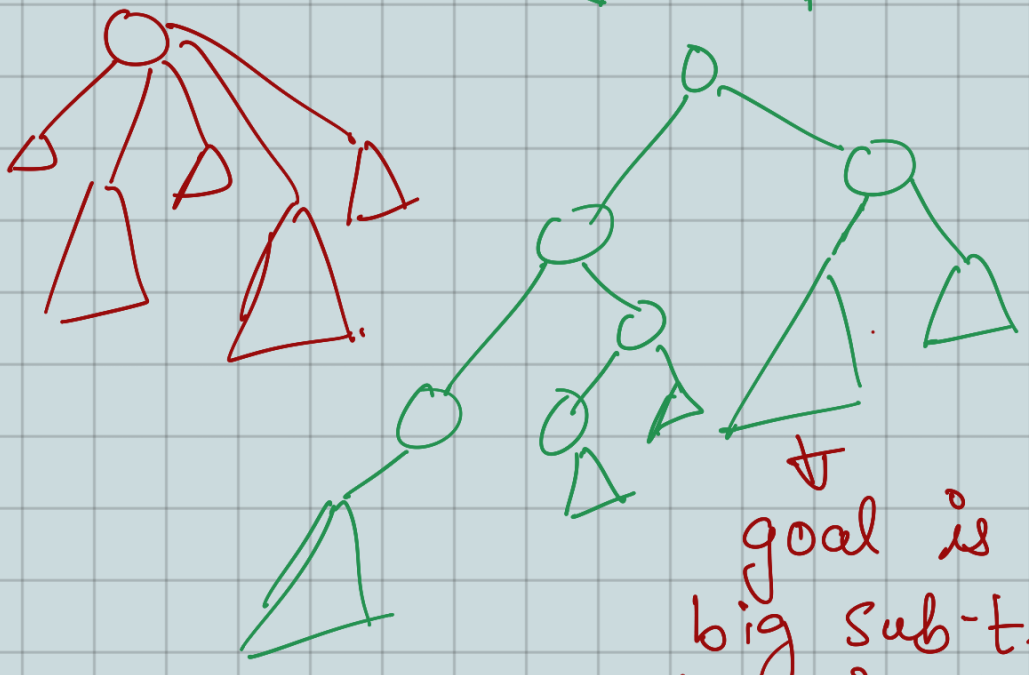
	query	Space
(1) as array	$O(P)$	$O(T \cdot \Sigma)$
(2) as balanced BST	$O(P \cdot \lg \Sigma)$	$O(T)$
(3) as hash table ↳ No pred.	$O(P)$	$O(T)$
(3.5) vEB / y-fast	$O(P \cdot \lg\lg \Sigma)$	$O(T)$
(3.75) Tries [Kopelowitz & Lewenstein 2006]	$O(P + \lg \Sigma)$	$O(T)$
(4) weight balanced BST	$O(P + \lg K)$	$O(T)$
(5) leaf trimming	$O(P + \lg \Sigma)$	$O(T)$
(*) vEB only when you fall off	$O(P + \lg\lg \Sigma)$	$O(T)$

④ Weight-balanced BST

$(P + \lg K)$ query, $O(T)$ space.

→ # descendant leaves





goal is to get the big sub-tree closer to the root

- Every 2 edges follow either
  - advance 1 letter in P

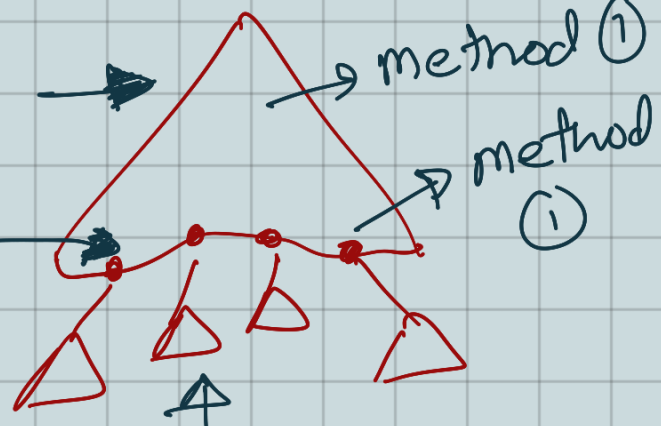
or - reduce # candidate T_i to 2/3

(5) leaf trimming:

- cut below maximally deep nodes with $\geq |\Sigma|$ descendant nodes

$$\#\text{ leaves} \leq \frac{|T|}{|\Sigma|}$$

$$\#\text{ branching nodes} < \frac{|T|}{|\Sigma|}$$



non branching top nodes

method (2)

method (4)

$K < |\Sigma|$

$O(P + \lg \Sigma)$ query time.

(*) String sorting:

$O(T + K \cdot \lg \Sigma)$

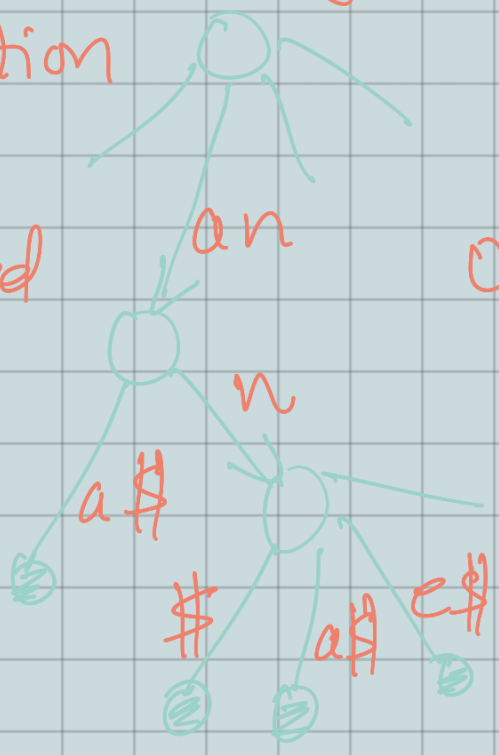
$O(T \cdot K \lg K)$

via comparison.

Compresed tries: contract non-branching paths into single edge.

- Same representation apply..,

with T = # compresed nodes



O(K) nodes.

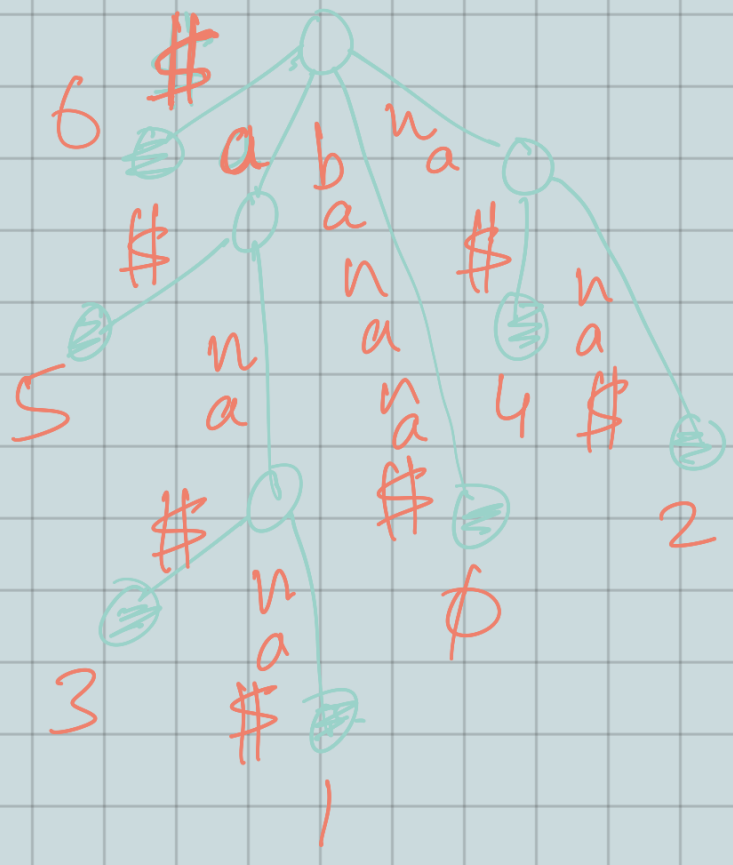
Suffix trees: Compressed trie of $|T|$ suffixes $T[i:]$ of T (with \$ appended)

- e.g: b a n a n a $
  0 1 2 3 4 5 6
- $|T|+1$ leaves
- edge label = substring $T[i:j]$
- store as two indicies (i, j)

$\Rightarrow$ $O(T)$ space.

Suffixes:

0	b	a	n	a	n	a	$
1		a	n	a	n	a	$
2			n	a	n	a	$
3				a	n	a	$
4					n	a	$
5						a	$
6							$



Applications:

- Search for P gives subtree whose leaves correspond to all occurrences of P
  - $O(P)$ time via hash ($\rightarrow \nu EB$)
  - $O(P + \lg \Sigma)$ via trays $\Rightarrow$ leaves stored in T
- list first K occurrences in $O(K)$ more time
  - every node points to leftmost descend leaf
  - leaves connected via linked list
- # of occurrences in $O(1)$ more time (subtrees sizes)
- Longest repeated substring in T
  - $O(T)$ time
  - = branching node of maximum "letter depth".
- longest substring match of $T[i:]$ & $T[j:]$ $O(1)$ via LCA query

Suffix array:

Sort the suffixes of T
just store the indices $\Rightarrow O(|T|)$ space.

→ e.g. b a n a n a $
0 1 2 3 4 5 6

$\$ < a < b \ldots$

SA ←

6	$
5	a $
3	a n a $
1	a n a n a $
0	b a n a n a $
4	n a $
2	n a n a $

LCP:

$O(|T| + \text{sort}(\Sigma))$ construction.

→ Searchable in $O(|P| \lg |T|)$ via binarysearch
→ LCP[i] = length of longest common prefix of i^{th} & $(i+1)^{st}$ suffix in order

→ when binary Searching in interval SA[i:j],
only need to compare from letter
$RMQ_{LCP}(i, j-1)$

Cartesian tree of LCP