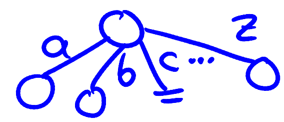
Lecture 8

<u>String matching</u>: given <u>text</u> T & pattern P,
say both strings over alphabet Σ,
find some/all <u>occurrences</u> of P in T
as substring

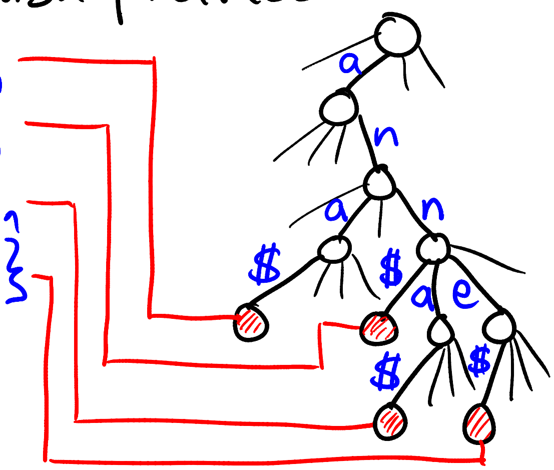
- one-shot: $O(|T|)$ time [Knuth, Morris, Pratt — SICOMP 1977; Boyer, Moore — CACM 1977; Karp, Rabin — IBM JRD 1987]
- static data structure: <u>suffix tree</u>
  - $O(|T|)$ space (words — later, $O(|T|)$ <u>bits</u>)
  - $O(|T| + \text{sort}(\Sigma))$ preprocessing
  - $O(|P|)$ query

Trie = tree with child branches (not necc. all present) labeled by letters from Σ;

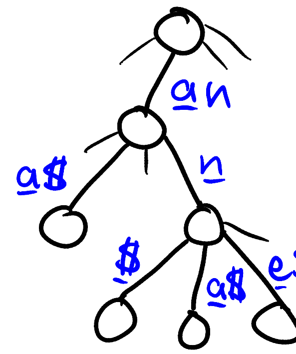


$\equiv$ tree where root-to-leaf paths correspond to strings over Σ;

- terminate strings with new letter \$ to distinguish prefixes
- e.g.: {ana, ann, anna, anne}



trie

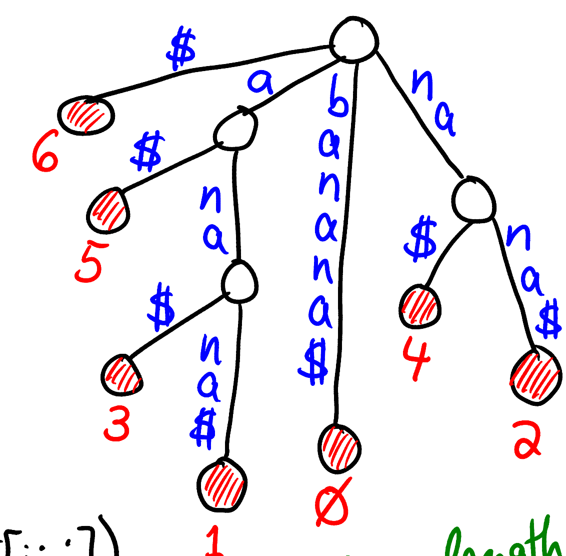


compressed trie

Compressed trie: contract nonbranching paths to single edge, keyed by first letter in path

Suffix tree (trie): Compressed trie of all $|T|+1$ suffixes of $T\$$

- e.g.: b a n a n a \$
  Ø 1 2 3 4 5 6



- $|T|+1$ leaves
- store edge labels as two indices into T ($T[i:j]$) ← or just store length & check at end
  $\Rightarrow O(|T|)$ space [words]

Suffix arrays: sort suffixes of T
just store their indices

- e.g.: b a n a n a \$
  ∅ 1 2 3 4 5 6

6	\$
5	a \$
3	a n a \$
1	a n a n a \$
∅	b a n a n a \$
4	n a \$
2	n a n a \$

∅ 1 3 ∅ ∅ 2

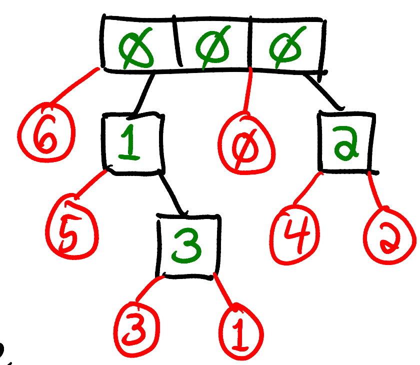
- searchable in $O(|P| \cdot \lg |T|)$ via simple binary search
- lcp array: lcp[i] = length of longest common prefix between ith & (i+1)st suffixes in suffix array
- suffix + lcp arrays (+ RMQ DS) ⇒ searchable in $O(|P| + \lg |T|)$ [PS4]

Equivalence to suffix trees:

- in-order traversal of leaves in suffix tree → suffix array
- Cartesian tree of lcp array
  - put all mins. at root
  - recurse in resulting array pieces to form subtrees

  → internal nodes of suffix tree

  - suffixes fit in between as leaves (ordered by SA)



- lcp's give letter depth ⇒ (length of) edge labels
- lcp array computable in $O(|T|)$ time [Kasai et al. – CPM 2001] or directly in suffix-array construction (below)

Constructing suffix tree in $O(|T| + \text{sort}(\Sigma))$

[Kärkäinen & Sanders — ICALP 2003], inspired by [Farach — FOCS 1997; Farach-Colton, Ferragina, Muthukrishnan — JACM 2000]

1. sort Σ ~ initially in $\text{sort}(\Sigma)$ time $\rightarrow O(n \lg n)$ say
   — later, radix sort will be $O(|T|)$
2. replace each letter in T by its rank in Σ
   — preserves order & lcp's of suffixes
3. form $T_0 = \langle (T[3i], T[3i+1], T[3i+2]) \text{ for } i = 0, 1, 2, \dots \rangle$
   $T_1 = \langle (T[3i+1], T[3i+2], T[3i+3]) \text{ for } i = 0, 1, 2, \dots \rangle$
   $T_2 = \langle (T[3i+2], T[3i+3], T[3i+4]) \text{ for } i = 0, 1, 2, \dots \rangle$
   $\Rightarrow \text{suffixes}(T) \approx \bigcup_{i=0,1,2} \text{suffixes}(T_i)$
4. recurse on $\langle T_0, T_1 \rangle \Rightarrow \frac{2}{3} n$ "letters"
   $\rightarrow$ sorted order & lcps of $\bigcup_{i=0,1} \text{suffixes}(T_i)$
5. radix sort $\text{suffixes}(T_2)$ by writing
   $T_2[i:] = T[3i+2:] = \langle T[3i+2], T[3i+3:] \rangle = \langle T[3i+2], T_0[i+1:] \rangle$
   — also get lcps in $\text{suffixes}(T_2)$ by trying to extend +1
6. merge $\bigcup_{i=0,1} \text{suffixes}(T_i)$ with $\text{suffixes}(T_2)$ via:
   — $T_0[i:]$ vs. $T_2[j:] = T[3i:]$ vs. $T[3j+2:]$
   $= \langle T[3i], \underbrace{T[3i+1:]}_{T_1[i:]} \rangle$ vs. $\langle T[3j+2], \underbrace{T[3j+3:]}_{T_0[j+1:]} \rangle$
   — $T_1[i:]$ vs. $T_2[j:] = T[3i+1:]$ vs. $T[3j+2:]$
   $= \langle T[3i+1], T[3i+2], \underbrace{T[3i+3:]}_{T_0[i+1:]} \rangle$ vs. $\langle T[3j+2], T[3j+3], \underbrace{T[3j+4:]}_{T_1[j+1]} \rangle$
   — also get lcps by trying to extend +1 or +2

$\Rightarrow T(n) = T(\frac{2}{3} \cdot n) + O(n)$ $\quad n = |T|$

Applications of suffix trees

- count # occurrences of P: augment subtree sizes
- list first k occurrences: $O(|P|+k)$
- longest repeated substring: $O(|T|)$
  - branching node of maximum letter depth
- multiple documents via multiple \$'s: $T = T_0 \$_0 T_1 \$_1 \cdots$
- longest common substring: $O(|T|)$
  - max.-letter-depth node with ≥ 2 distinct \$'s below

Document retrieval [Muthukrishnan — SODA 2002]

- extract k distinct documents containing P in $O(|P|+k)$
- P's subtree maps to interval $[i, j]$ of suffix array
- each $\$_k$ stores index in suffix array of previous $\$_k$
- want \$'s in interval $[i, j]$ with stored index $< i$
  (first occurrence of that $\$_k$ in suffix array)
- store range-minimum query DS on stored index
- $O(|P|+k)$ query: L16
  - find position k of min. stored index in $[i, j]$ in $O(1)$
  - if stored index is $< i$:
    - output it
    - recurse in $[i, k-1]$ & $[k+1, j]$

$\Rightarrow O(1)$ time per output

$\underline{LCA}(v, w)$ = lowest common ancestor of v & w
= longest common prefix match of substrings

- can preprocess a tree for $O(1)$-time LCA queries
  (actually $\equiv$ RMQ) [Lecture 16 again]
- longest palindrome centered at $T[i]$
  = longest common prefix of $T[i:]$ & $rev(T)[-i:]$

$\Rightarrow$ longest palindrome substring in $O(|T|)$ time

- map folding (all-layers simple folds)
  $\approx lcp(T[i:], rev(comp(T))[-i:])$
  [Arkin, Bender, Demaine, Demaine, Mitchell, Sethia, Skiena
  — CGTA 2004]