6.851	Lecture 16	Apr. 19, 2012

TODAY: Strings
- tries & trays
- compressed tries
- suffix trees & arrays
- document retrieval
- linear-time construction

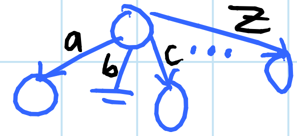
String matching: given text T & pattern P, here both strings over alphabet Σ, find some/all occurrences of P in T as substrings
- one-shot: $O(T)$ time [Knuth, Morris, Pratt – SICOMP 1977; Boyer & Moore – CACM 1977; Karp & Rabin – IBM JRD 1987]
- static DS: preprocess T, query $= P$
  - goal: $O(P)$ query
    $O(T)$ space

- other data structures consider when P has wildcards, or when P need not match as an exact substring (Hamming/edit distance) ~ see e.g. [Cole, Gottlieb, Lewenstein – STOC 2004] [Maaß & Novak – CPM 2005]

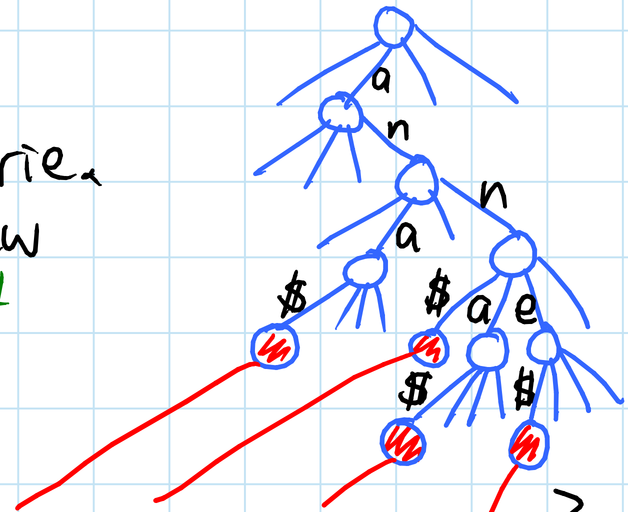
Warmup: predecessor among strings $T_1, \dots, T_k$

(e.g. library search)

Trie = rooted tree with child branches labeled with letters in Σ



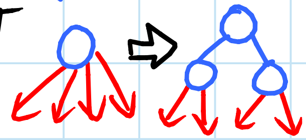
- to represent strings as root-to-leaf paths in a trie, terminate them with a new letter \$ (otherwise can't distinguish prefixes as absent or present)
  - e.g.: {ana, ann, anna, anne}



- in-order traversal of leaves = sorted strings

Trie representation: $T = \#$ nodes in trie $\le \sum_{i=1}^{k} |T_i|$

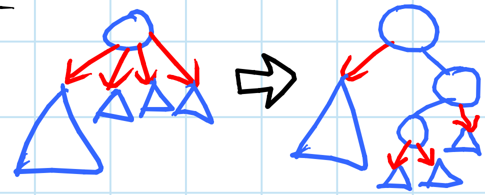
node stores children:	query	space
① as array	$O(P)$	$O(T\Sigma)$
↳ blank cells store predecessor/successor		
② as balanced BST	$O(P \lg \Sigma)$	$O(T)$
③ as hash table	$O(P)$	$O(T)$
↳ doesn't support predecessor queries/sorting		
3.5 as van Emde Boas/y-fast	$O(P \lg\lg \Sigma)$	$O(T)$
3.75 = 3 + 3.5 (only need vEB when fall off)	$O(P + \lg\lg \Sigma)$	$O(T)$



[Farach-Colton – personal communication, 2012]:

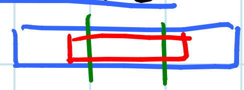
node stores children:	query	space
(4) as weight-balanced BST	$O(P + \lg k)$	$O(T)$
↳ # descendant leaves in T		↳ # leaves

– split children in left & right halves to optimally balance sum of weights



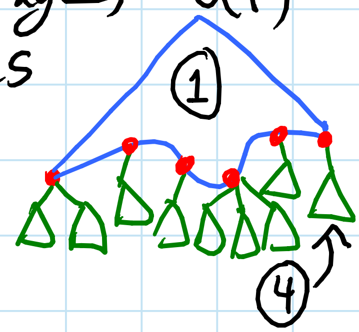
⇒ every 2 edges followed either advances P letter or reduces # candidate T strings to 2/3

⇒ charge to $O(P)$ or $O(\lg k)$



	query	space
(5) leaf trimming (indirection)	$O(P + \lg \Sigma)$	$O(T)$

– cut below maximally deep nodes with $\geq |\Sigma|$ descendant (leave)s



⇒ # leaves in top trie $\leq |T| / |\Sigma|$

⇒ # branching top nodes $\leq |T| / |\Sigma|$

– use (1) on branching top nodes
& (1) on top leaves (to find right bottom trie)
& (2) on rest of top (⇒ nonbranching in T)

⇒ $O(T)$ space on top

– bottom trees have $< |\Sigma|$ descendant (leave)s

⇒ (4) achieves $O(P + \lg \Sigma)$ query time

↳ simplification by Farach-Colton of:

	query	space
(6) suffix trays	$O(P + \lg \Sigma)$	$O(T)$

[Cole, Kopelowitz, Lewenstein – ICALP 2006]

Application: sorting strings $T_1, \ldots, T_k$

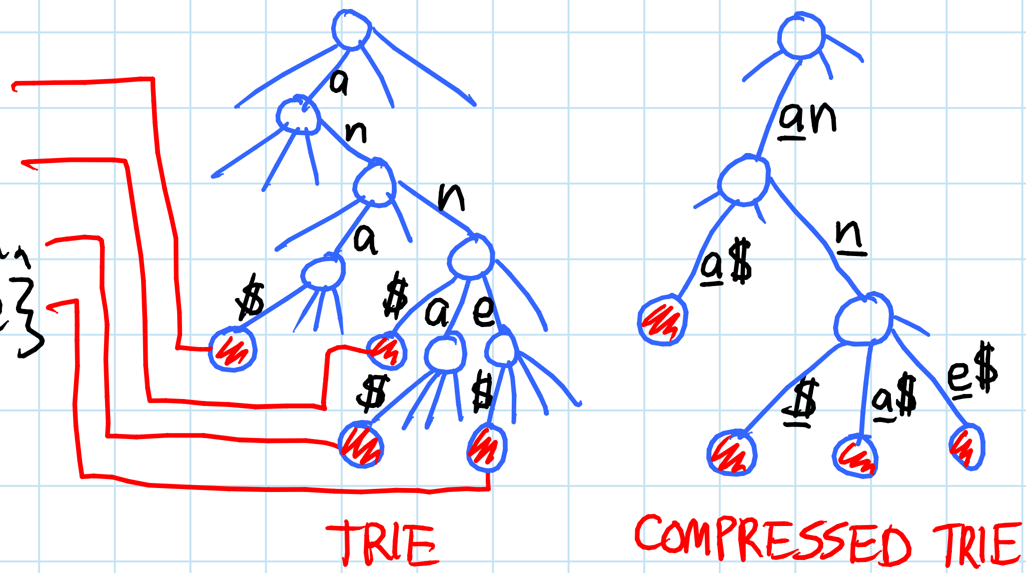
– repeatedly insert into trie/tray

$\Rightarrow O(T + k \lg \Sigma)$

– typically $O(T)$ & $\ll O(T k \lg k)$ via comparison

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

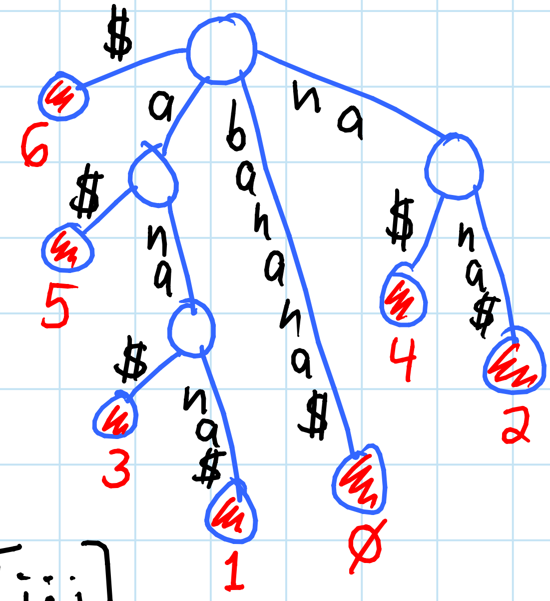
e.g. {ana, ann, anna, anne}



– same representations apply, with T = # compressed nodes

<u>Suffix tree (trie):</u>

compressed trie of all $|T|$ 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 $T[i:j]$

$\Rightarrow$ store as two indices (i,j)

$\Rightarrow O(T)$ space

<u>Applications:</u>

- search for P gives subtree whose leaves correspond to <u>all</u> occurrences of P
  - $O(P)$ time via hashing
  - $O(P + \lg \Sigma)$ via trays $\Rightarrow$ leaves sorted in T
  - $O(P + \lg\lg \Sigma)$ via hash + vEB
- list first k occurrences in $O(k)$ more time
  - every node points to leftmost descend. leaf
  - leaves connected via linked list
- # occurrences in $O(1)$ more time (subtree sizes)

- longest repeated substring in T: $O(T)$ time
  = branching node of maximum "letter depth"
- longest substring match of $T[i:]$ vs. $T[j:]$: $O(1)$ via LCA query

- all occurrences of $T[i:j]$ = $(j-i)$th "weighted" level ancestor of leaf for $T[i:]$ for compression
  - store nodes in long path/ladder of L15 in van Emde Boas predecessor DS ⇒ $O(\lg \lg T)$
  - can't afford lookup tables at the bottom...
  - use ladder decomposition on bottom trees ⇒ jump to top of $O(\lg \lg n)$ ladders (to reach height $O(\lg n)$)
  - only need predecessor query on last ladder ⇒ $O(\lg \lg T)$ query & $O(T)$ space
    [Abbott, Baran, Demaine, ... – 6.897, Spr. 2005, L19.5]
- multiple documents via mult. \$s: $T = T_1 \$_1 \cdots T_k \$_k$
  make these leaves (trim below)
- count # distinct documents containing P
  - store # distinct \$s below each node
- longest common substring in $O(T)$
  = branching node with ≥2 distinct \$'s below
- find d distinct documents containing P in $O(d)$ more
  "document retrieval problem" [Muthukrishnan – SODA 2002]
  - each $\$_i$ stores leaf # of previous $\$_i$
  - in interval $[l, n]$ of leaves below a node, want first $\$_i$, i.e. $\$_i$ storing $< l$, for each occ. i

  - so find $m = \text{RMQ}(l, n)$ on array of stored values [L15]
  - if stored value at leaf m is $< i$:
    - found desired $\$_i$ ~ output it
    - recurse in intervals $[l, m-1]$ & $[m+1, n]$
  ⇒ $O(1)$ time per output (& can stop anytime)

Suffix arrays: 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

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: 0, 1, 3, 0, 0, 2

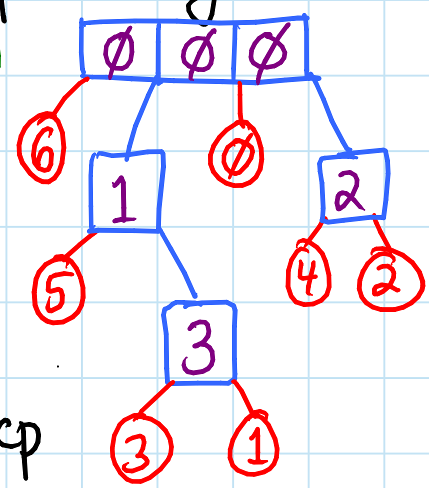
- searchable in $O(P \lg T)$ via binary search
- lcp[i] = length of longest common prefix of ith & (i+1)st suffix in order
- when binary searching in interval $SA[i:j]$, only need to compare from letter $RMQ_{lcp}(i, j-1)$
- via RMQ of L15, $O(P + \lg T)$ search [2007, PS4]

Suffix trees $\Leftrightarrow$ suffix arrays:

- ($\Rightarrow$) via in-order traversal of leaves
- ($\Leftarrow$) via Cartesian tree of lcp array
  - put all mins at root (unlike L15)
  - nonleaf child subtrees: recurse
  - suffixes fit in between as leaves
  - lcp value forming a node = letter depth of that node

  $\Rightarrow$ edge length = child lcp − parent lcp

  $\Rightarrow$ can reconstruct labels
  - all doable in linear time [L15]



- lcps computable in $O(T)$ from SA [Kasai et al. — CPM 2001]

or directly in suffix-array construction below

Constructing suffix array ($\Rightarrow$ tree) in $O(T + \text{sort}(\Sigma_i))$

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

① sort Σ_i – initially in sort(Σ_i) time (or, if don't need children sorted, just number Σ_i arbitrarily)
– later, radix sort in $O(T)$ time

② replace each letter by its rank in Σ_i $\Rightarrow \leq |\Sigma_i|$

③ form $T_0 = \langle (T[3i], T[3i+1], T[3i+2]) \text{ for } i = 0, 1, 2, \ldots \rangle$

$T_1 = \langle (T[3i+1], T[3i+2], T[3i+3]) \text{ for } i = 0, 1, 2, \ldots \rangle$

$T_2 = \langle \underbrace{(T[3i+2], T[3i+3], T[3i+4])}_{\text{single "letter"}} \text{ for } i = 0, 1, 2, \ldots \rangle$

$\Rightarrow \text{suffixes}(T) \simeq \bigcup_{i=0,1,2} \text{suffixes}(T_i)$

④ recurse on $\langle T_0, T_1 \rangle$ $\Rightarrow \frac{2}{3}|T|$ "letters"

$\rightarrow$ sorted order & lcps of $\bigcup_{i=0,1} \text{suffixes}(T_i)$

⑤ radix sort suffixes(T_2) by writing

$T_2[i:] \simeq T[3i+2:] = \langle T[3i+2], T[3i+3:] \rangle \simeq \langle T[3i+2], T_0[i+1:] \rangle$

– also get lcps in suffixes(T_2): try to extend by 1

⑥ merge $\bigcup_{i=0,1} \text{suffixes}(T_i)$ with 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. $T[3j+2:]$

$= \langle T[3i+1], T[3i+2], T[3i+3:] \rangle \rightarrow T_0[i+1:]$

vs. $\langle T[3j+2], T[3j+3], T[3j+4:] \rangle \rightarrow T_1[i+1:]$

– also get lcps: try to extend by 1 or 2

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

6.851 Advanced Data Structures
Spring 2012