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

String matching: given text $T$ & pattern $P$, find some/all occurrences of $P$ in $T$ as substrings

- one-shot: $O(T)$ time (Knuth, Morris, Pratt - 1977; Boyer & Moore - 1977; Karp & Rabin - IBM 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
  ~ Maas & Novak - CPM 2005
Warmup: predecessor among strings $T_1, \ldots, T_k$ (e.g. library search)

**Trie** = rooted tree with child branches labeled with letters in $\Sigma$

- 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.:
    - $\{\text{ana}, \text{ann}, \text{anna}, \text{anne}\}$

- in-order traversal of leaves = sorted strings

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

- node stores children:
  - as array
    - blank cells store predecessor/successor
  - as balanced BST
    - query space $O(P)$, $O(T\Sigma)$
  - as hash table
    - query space $O(P)$, $O(T)$
  - as van Emde Boas/y-fast
    - query space $O(P\lg\Sigma)$, $O(T)$
  
$(3.75) = (3) + (3.5)$ (only need VEB when fall off) $O(P + P\lg\Sigma)$, $O(T)$
[Farach-Colton — personal communication, 2012]:

4. Node stores children:
   - as weight-balanced BST
   - # descendant leaves in T
   - 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 $O(lg k)$

5. Leaf trimming (indirection)
   - cut below maximally deep nodes with $\geq |\Sigma|$ descendant leaves
   - # 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 (not branching in T)
   - $O(T)$ space on top
   - bottom trees have $< |\Sigma|$ descendant leaves
   - (4) achieves $O(P + lg |\Sigma|)$ query time

6. Suffix trays

[Cole, Kopelowitz, Lewenstein — ICALP 2006]
Application: sorting strings $T_1, \ldots, T_k$
- repeatedly insert into trie/tray
  $\Rightarrow O(T + k \log \Sigma)$
  - typically $O(T)$ $&$ $\ll O(T k \log k)$ via comparison

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

e.g. $\{ \text{ana}, \text{ann}, \text{anna}, \text{anne} \}$

TRIE

COMPRESSED TRIE

- same representations apply, with $T =$ # compressed nodes
Suffix tree (trie):
- compressed trie of all $|T|$ suffixes $T[i:]$ of $T$ (with $\$\$ appended)
  - e.g.: b a n a n a $\$$
  - $|T|+1$ leaves
  - edge label = substring $T[i:j]$
  - store as two indices $(i, j)$
  - $O(|T|)$ space

Applications:
- search for $P$ gives subtree whose leaves correspond to all occurrences of $P$
  - $O(|P|)$ time via hash (+UEB) $\Rightarrow$ leaves can still be sorted in $T$
  - $O(|P| \cdot \lg S_i)$ via trays $\Rightarrow$ leaves sorted in $T$
- 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 $T[i:j] = (1T1-j)\text{th level ancestor of leaf for } T[i:]$ for compression?
- store nodes in long path/ladder of $L_{15}$ in van Emde Boas predecessor DS $\Rightarrow O(lg lg T)$
- can't afford lookup tables at the bottom...
- use ladder decomposition on bottom trees $\Rightarrow$ jump to top of $O(lg lg n)$ ladders (to reach height $O(lg n)$)
- only need predecessor query on last ladder $\Rightarrow O(lg lg T)$ query & $O(T)$ space $[Abbott, Baran, Demaine,... - 6.897, Spr. 2005, L19.5]$

- multiple documents via multi. $T_1 \cdots T_k$ $\Rightarrow$ count # distinct documents containing $P$
- store # distinct $S_i$'s below each node
- longest common substring in $O(T)$ $\Rightarrow$ branching node with $\geq 2$ distinct $S_i$'s below
- find d distinct documents containing $P$ in $O(d)$ more "document retrieval problem" $[Muthukrishnan - SODA 2002]$
- each $S_i$ stores leaf # of previous $S_i$
- in interval $[l,n]$ of leaves below a node, want first $S_i$, i.e. $S_i$ storing $< l$, for each occ. $i$
- $\Rightarrow$ find $m = \text{RMQ}(l,n)$ on array of stored values $[L15]$
- if stored value at leaf $m$ is $< i$:
  - found desired $S_i$ $\Rightarrow$ output it
- recurse in intervals $[l,m-1]$ & $[m+1,n]$ $\Rightarrow O(1)$ time per output $\Rightarrow$ (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\$ | 6 $\$
    $\emptyset\ 1\ 2\ 3\ 4\ 5\ 6$

- searchable in $O(P\lg T)$ via binary search
- $lcp[i] =$ length of longest common prefix of $i$th & $(i+1)$th 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 array:

- ($\Rightarrow$) via in-order traversal of leaves
- ($\Leftarrow$) via Cartesian tree of $lcp$ array
  - put all mins at root (unlike $L15$)
  - non-leaf 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 $lcp$
  - can reconstruct labels
  - all doable in linear time [L15]
  - $lcp$s computable in $O(T)$ from $SA$ [Kasai et al. - cpm 2001] or directly in suffix-array construction below
Constructing suffix array (⇒ tree) in $O(T + \text{sort}(\Sigma))$  
[Kärkkäinen & Sanders - ICALP 2003], inspired by  
[Farach - FoCS 1997; Farach-Colton, Ferragina, Muthukrishnan - JACM]

1. Sort $\Sigma$ - initially in $\text{sort}(\Sigma)$ time (or, if don't need children sorted, just number $\Sigma$ arbitrarily)  
   - later, radix sort in $O(T)$ time

2. replace each letter by its rank in $\Sigma \Rightarrow |\Sigma| \leq |T|$  
3. form $T_0 = \langle (T[3i], T[3i+1], T[3i+2]) \rangle$ for $i = 0, 1, 2, \ldots$  
   $T_1 = \langle (T[3i+1], T[3i+2], T[3i+3]) \rangle$ for $i = 0, 1, 2, \ldots$  
   $T_2 = \langle (T[3i+2], T[3i+3], T[3i+4]) \rangle$ for $i = 0, 1, 2, \ldots$  
   
   - single "letter"

⇒ suffixes(t) = $\bigcup_{i=0,1,2} \text{suffixes}(T_i)$  
4. recurse on $\langle T_0, T_1 \rangle$ ⇒ $\frac{2}{3}|T|$ "letters"  
   → sorted order & lcp of $\bigcup_{i=0,1} \text{suffixes}(T_i)$

5. radix sort suffixes ($T_2$) by writing  
   $T_2[i:] = T[3i+2:] = \langle T[3i+2], T[3i+3:] \rangle \approx \langle T[3i+2], T_0[i+1:] \rangle$  
   - also get lcp 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. $T[3j+2:]$  
     = $\langle T[3i], T[3i+1:] \rangle$ vs. $\langle T[3j+2], T[3j+3:] \rangle$  
   - $T_1[i:]$ vs. $T_2[j:] = T[3i+1:]$ vs. $T[3j+2:]$  
     = $\langle T[3i+1], T[3i+2], T[3i+3:] \rangle$ vs. $\langle T[3j+2], T[3j+3], T[3j+4:] \rangle$

⇒ also get lcp: try to extend by 1 or 2  
$\Rightarrow T(n) = T(\frac{2}{3}n) + O(n) = O(n)$ (n = |T|)