Lecture 8

String matching: given text $T$ & pattern $P$, say both strings over alphabet $\Sigma$, find some/all occurrences of $P$ in $T$ as substring


- static data structure: suffix tree
  $O(|T|)$ space (words—later, $O(|T|)$ bits)
  $O(|T| + \text{sort}(\Sigma'))$ preprocessing
  $O(|P|)$ query
Trie = tree with child branches labeled by letters from $\Sigma'$

- tree where root-to-leaf paths correspond to strings over $\Sigma'$
- terminate strings with new letter $\$ to distinguish prefixes

- e.g.: \{ana, ann, anna, anna$\}

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.: $banana$$\$
  \$ 1 2 3 4 5 6
- $|T|+1$ leaves
- store edge labels as two indices into $T$ ($T[i:j]$)
  $\Rightarrow O(|T|)$ space [words]

Or, just store length & check at end
Suffix arrays: sort suffixes of $T$
just store their indices

- e.g.: $\text{banana} \, \text{$}$
  $\emptyset \, 1 \, 2 \, 3 \, 4 \, 5 \, 6$

- searchable in $O(|P| \cdot \log |T|)$ via simple binary search

- $\text{lcp array: } \text{lcp}[i] =$ length of longest common prefix between $i$th & $(i+1)$th suffixes in suffix array

- suffix + lcp arrays $\Rightarrow$ searchable in $O(|P| + \log |T|)$ $\Rightarrow$ RMA DS

Equivalence to suffix trees:
- in-order traversal of leaves in suffix tree
  $\Rightarrow$ suffix array

- Cartesian tree of lcp array
  - put all mins. at root
  - recurse in resulting array pieces to form subtrees
  $\Rightarrow$ internal nodes of suffix tree

- suffixes fit in between as leaves (ordered by SA)
- lcp's give letter depth $\Rightarrow$ (length of) edge labels
- lcp array computable in $O(|T|)$ time $[\text{Kasai et al.-CPM}]$
  or directly in suffix-array construction (below)
Constructing suffix tree in \(O(11\ell + \text{sort}(\Sigma))\)

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

1. sort \(\Sigma\)' initially in \(\text{sort}(\Sigma)\) time \(\sim O(n \log n)\) say
- later, radix sort will be \(O(11\ell)\)

2. replace each letter in \(T\) by its rank in \(\Sigma\)
- preserves order & lcp's of suffixes

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 \(i = 0, 1, 2, \ldots\)
\(T_2 = \langle (T[3i+3], T[3i+3], T[3i+4]) \rangle\) for \(i = 0, 1, 2, \ldots\)

\(\Rightarrow\) 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"
   -> sorted order & lcp's of \(\bigcup_{i=0,1} \text{suffixes}(T_i)\)

5. radix sort suffixes \((T_2)\) by writing
   \(T_2[i:] = T[3i+2:] = \langle T[3i+2], T[3i+3] \rangle \sim \langle T[3i+2], T_0[i+1] \rangle\)
   - also get lcp's in suffixes \((T_2)\) by trying to extend +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. \(T[3j+2:]\)
     \(\langle T[3i, T[3i+1]], \rangle \sim \langle T[3j+2], T[3j+3] \rangle\)
     \(\overbrace{T_1[i:] \sim \overbrace{T_0[i+1]}}^{T_0[i+1]}\)
   - \(T_1[i:]\) vs. \(T_2[j:] = T[3i+1:]\) vs. \(T[3j+2:]\)
     \(\langle T[3i+1, T[3i+2], T[3i+3]] \rangle \sim \langle T[3j+2], T[3j+3], T[3j+4] \rangle\)
     \(\overbrace{T_0[i+1]} \sim \overbrace{T_1[i+1]} \sim \overbrace{T_0[i+1]} \sim \overbrace{T_1[i+1]}\)
   - also get lcp's by trying to extend +1 or +2

\(\Rightarrow T(n) = T(\frac{2}{3}n) + O(n)\) \quad \(n=11\ell\)
Applications of suffix trees
- count # occurrences of P: augment subtree sizes
- list first k occurrences: O(|Pl+k)
- longest repeated substring: O(|T|)
  - branching node of maximum letter depth
- multiple documents via multiple $'s: T=T_0$$_1T_1$$_1...$
- 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(|Pl+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(|Pl+k) query:
  - 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]
⇒ O(1) time per output
\( \text{LCA}(v,w) = \text{lowest common ancestor of } v \text{ and } w \)

- longest common prefix match of substrings
  - can preprocess a tree for \( O(1) \)-time LCA queries
    (actually \( \equiv \text{RMQ} \))
    [Lecture 16 again]

- longest palindrome centered at \( T[i] \)
  - longest common prefix of \( T[i:] \) & \( \text{rev}(T)[-i:] \)

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

- map folding (all-layers simple folds)
  \( \approx \text{lcp}(T[i:], \text{rev} \circ \text{comp}(T))[-i:] \)

[Ark"{i}n, Bender, Demaine, Demaine, Mitchell, Sethia, Skiena

--- CGTA 2004]