6.851	Lecture 18	Apr. 26, 2012

TODAY: Succinct data structures II (of 2)
- compact & succinct suffix arrays/trees
- $O(T \lg\lg T)$-bit suffix array
- $O(T)$-bit suffix array
- succinct suffix array → tree

Compact suffix arrays/trees:

TODAY * Grossi & Vitter: [STOC 2000; SICOMP 2005] first
$(1+1/\varepsilon)\,|T| \lg|\Sigma| + 2|T| + O(\frac{T}{\lg\lg T})$ bits
($|T| \lg|\Sigma|$ = OPT*; $2|T|$ if $|\Sigma|>2$; *without compression)
$O(\frac{P}{\log_\Sigma T} + |\text{output}| \cdot \log_\Sigma^\varepsilon T)$ query

- Ferragina & Manzini: [FOCS 2000; JACM 2005]
$5 \cdot H_k(T) \cdot |T| + O(T \frac{\sigma + \lg T}{\lg\lg T} + T^\varepsilon \sigma^{\sigma+1})$ Burrows–Wheeler Trans.
($H_k(T)$: kth order empirical entropy) for any fixed k
$= \sum_{|w|=k} \Pr\{w \text{ occurring}\} \cdot H_0(\text{string of successor characters of } w)$
($\Pr\{w \text{ occurring}\} = \#\text{occurrences}/|T|$)
$= \sum_x P_x \lg \frac{1}{P_x}$, $P_x = \frac{\#\text{occ.}(wx)}{\#\text{occ.}(w)}$
$=$ # bits/char. in OPT code depending on last k chars
$O(|P| + |\text{output}| \cdot \lg^\varepsilon T)$ query

- Sadakane: [J. Alg. 2003] large alphabets
$\frac{1+\varepsilon'}{\varepsilon} \cdot H_0(T) \cdot |T| + O(T \lg\lg\sigma + \sigma \lg\sigma)$ bits
$O(P \lg T + |\text{output}| \cdot \lg^\varepsilon T)$ query for $0<\varepsilon,\varepsilon'<1$

Succinct:

- Grossi, Gupta, Vitter: [SODA 2003]
  $H_k(T) \cdot |T| + O(T \lg \Sigma \cdot \frac{\lg \lg T}{\lg T})$ bits
  $O(P \lg \Sigma + \frac{\lg^2 T}{\lg \lg T} \lg \Sigma)$ query
- Ferragina, Manzini, Mäkinen, Navarro: [TALG 2007]
  $H_k(T) \cdot |T| + O(T / \lg^{\varepsilon} n)$ bits
  $O(P + |\text{output}| \cdot \lg^{1+\varepsilon} T)$ query + $O(P)$ counting query

Extras:

- low-space construction [Hon, Sadakane, Sung - FOCS 2003; Hon, Lam, Sadakane, Sung, Yu - Alg. 2007]
  $O(|T| \lg |\Sigma|)$ working
- suffix-tree ops. like maximal common substrings
  [Han & Sadakane - CPM 2002; Sadakane - TCS 2007]
- document retrieval [Sadakane - J. Disc. Alg. 2007]
- dynamic [Chan, Hon, Lam, Sadakane - T. Alg. 2007]
- implementations & experiments [...]

Compressed suffix arrays: [Grossi & Vitter - STOC 2000]

follow divide & conquer of linear-time suffix tree/array alg. [L16], but 2-way instead of 3-way

- start: text $T_0 = T$
  size $n_0 = n$
  suffix array $SA_0 = SA$ (in sorted order of suffixes)
  $SA[i]$ = index in T of ith suffix of T
- step: $T_{k+1} = \langle (T_k[2i], T_k[2i+1]) \text{ for } i = 0, 1, \ldots, n/2 \rangle$
  $n_{k+1} = n_k/2 = n/2^k$
  $SA_{k+1} = \frac{1}{2} \cdot$ extract even entries of SA_k
- represent SA_k using SA_{k+1}:
  ① $\underline{\text{even-succ}}_k(i) = \begin{cases} i & \text{if } SA_k[i] \text{ is even} \\ j & \text{if } SA_k[i] = SA_k[j] - 1 \text{ is odd} \end{cases}$
  (jth suffix starts right after ith suffix)
  ② $\underline{\text{even-rank}}_k(i) = \#\text{even suffixes preceding } i\text{th suffix}$
  $= \#\text{even values in } SA_k[:i]$
  ③ $SA_k[i] = 2 \cdot SA_{k+1}[\text{even-rank}_k(\text{even-succ}_k(i))] - (1 - \text{is-even}_k(i))$
  (even-succ$_k$(i): round to even suffix $\Rightarrow$ in SA_{k+1}; even-rank$_k$(…): name of even suffix in SA_{k+1}; SA_{k+1}[…]: index of even suffix in T_{k+1}; $2 \cdot$ …: index of even suffix in T_k; $-(1 - \text{is-even}_k(i))$: unround)
- stop recursion at level $\ell = \lg\lg n \Rightarrow n_\ell = n/\lg n$
  $\Rightarrow$ can afford naive ptr. encoding: $n_\ell \lg n_\ell \le n$ bits

$\Rightarrow O(\lg\lg n)$ query to $SA[i]$, given $O(1)$-time is-even-suffix & even-succ & even-rank

$\underline{\text{is-even-suffix}_k}(i) = \begin{cases} 1 & \text{if } SA_k[i] \text{ is even} \\ 0 & \text{else} \end{cases}$ → n_k bits

$\underline{\text{even-rank}_k}$ = rank_1 in is-even-suffix → $O\left(n_k \frac{\lg\lg n_k}{\lg n_k}\right)$ bits [L17]

$\underline{\text{even-succ}_k}$:

- trivial for $SA_k[i]$ even ($n_k/2$ such i's)
- store answers for $SA_k[i]$ odd ($n_k/2$ such i's)
- order by i ⇒ $\text{even-succ}_k(i) = \text{odd-answers}[\underbrace{\text{odd-rank}_k(i)}_{= i\text{-even-rank}_k(i)}]$

$$= \text{order by odd suffix } T_k[SA_k[i]:]$$
$$= \text{order by } (T_k[SA_k[i]], T_k[\underbrace{SA_k[i]+1}_{\text{even}}:])$$
$$= \text{order by } (T_k[SA_k[i]], T_k[SA_k[\text{even-succ}_k(i)]:]) \quad \text{(def. of even-succ)}$$
$$= \text{order by } (T_k[SA_k[i]], \underbrace{\text{even-succ}_k(i)}_{\text{answer!}}) \quad (SA_k \text{ is a suffix array})$$

- actually store these pairs, in order by value
- assume $|\Sigma| = 2$ here

⇒ storing sorted array of $n_k/2$ values, $2^k + \lg n_k$ bits each

- store leading $\lg n_k$ bits of each value v_i via unary differential encoding: $0^{\text{lead}(v_1)}10^{\text{lead}(v_2)-\text{lead}(v_1)}1\ldots$
  ⇒ $n_k/2$ 1's & $\le 2^{\lg n_k} = n_k$ 0's ⇒ $\le \frac{3}{2} n_k$ bits [L17]
  - $\text{lead}(v_i) = \text{rank}_0(\text{select}_1(i)) - \text{rank}_0(\text{select}_1(i-1))$
- store trailing 2^k bits of each v_i explicitly in array
  ⇒ $2^k \cdot n_k/2 = n/2$ bits

⇒ $\frac{1}{2}n + \frac{3}{2}n_k + \underbrace{O\left(\frac{n_k}{\lg\lg n_k}\right)}_{\text{rank \& select}}$ bits

Total: $\sum_{k=0}^{\lg\lg n} \left(n_k + \frac{1}{2}n + \frac{3}{2}n_k + O\left(\frac{n_k}{\lg\lg n_k}\right)\right)$

$= \frac{1}{2} n \lg\lg n + 5n + O\left(\frac{n}{\lg\lg n}\right)$ bits — PROGRESS ~ BUT TOO BIG

Compact suffix array: $O(n)$ bits

- store only $1/\varepsilon + 1$ levels of recursion:
  $0, \varepsilon\ell, 2\varepsilon\ell, \ldots, \ell = \lg\lg n$
- represent $SA_{k\varepsilon\ell}$ using $SA_{(k+1)\varepsilon\ell}$ similarly:
  - "even" → "divisible by $2^{\varepsilon\ell}$" = "in $SA_{(k+1)\varepsilon\ell}$"
  - is-even$_k$ = $n_{k\varepsilon\ell}$-bit vector as before + rank$_1$ struct.
  - succ$_k(i) = j$ where $SA_{k\varepsilon\ell}[i] = SA_{k\varepsilon\ell}[j] - 1$
    stored in $n + 2n_{k\varepsilon\ell} + O\left(\frac{n_{k\varepsilon\ell}}{\lg\lg n_{k\varepsilon\ell}}\right)$ bits like even-succ
    ~ except can't skip ½ the values
- to compute $SA_{k\varepsilon\ell}[i]$:
  ① follow succ$_k$ repeatedly until j in $SA_{(k+1)\varepsilon\ell}$ (is-even$_k$ bit vector)
  ② recurse: $SA_{(k+1)\varepsilon\ell}[\text{rank}_1(j)]$
  ③ multiply by $2^{\varepsilon\ell}$, subtract by # steps in ①
- $\le 2^{\varepsilon\ell} = \lg^{\varepsilon} n$ succ$_k$ calls per level

$\Rightarrow O(\lg^{\varepsilon} n \lg\lg n) = O(\lg^{\varepsilon'} n)$ query to $SA[i]$

- space: $\sum_{k=0}^{1/\varepsilon} \left(n_{k\varepsilon\ell} + n + 2n_{k\varepsilon\ell} + O\left(\frac{n_{k\varepsilon\ell}}{\lg\lg n_{k\varepsilon\ell}}\right)\right)$
  ($n_{k\varepsilon\ell}$: is-even$_k$; $n + 2n_{k\varepsilon\ell}$: succ$_k$)

$$= \left(\tfrac{1}{\varepsilon} + 6\right)n + O\left(\frac{n}{\lg\lg n}\right) \text{ bits}$$

- optimizations:
  - succ$_k$ free at level $k = 0 \Rightarrow \Sigma \le 4n_{\varepsilon\ell} = O\left(\frac{n}{\lg^{\varepsilon} n}\right)$
  - store is-even$_k$ as succinct dictionary (with rank)
    $\Rightarrow \lg\binom{n_{k\varepsilon\ell}}{n_{(k+1)\varepsilon\ell}} + o \sim n_{(k+1)\varepsilon\ell} \lg \frac{n_{k\varepsilon\ell}}{n_{(k+1)\varepsilon\ell}} + o = O\left(n \frac{\lg\lg n}{\lg^{\varepsilon} n}\right)$
    ($\frac{n_{k\varepsilon\ell}}{n_{(k+1)\varepsilon\ell}} = 2^{\varepsilon\ell}$)
    → [Brodnik & Munro] of L17

$\Rightarrow \left(\tfrac{1}{\varepsilon} + 1\right) \cdot n + O\left(\frac{n}{\lg\lg n}\right)$ bits

OPEN: $O(n)$ bits & $o(\lg^{\varepsilon} n)$ query

Compact suffix tree given suffix array:

[Munro, Raman, Rao – J. Alg. 2001]

- store compressed binary trie on $2n+1$ nodes in $4n + o(n)$ bits using balanced parens. [L17]
  - ↪ "skipping the skips": can't store edge lengths
- during search for P, maintain letter depth of node:
  - to descend $x \to y$:
    - compute length of edge by finding longest match between leftmost-leaf(y) & rightmost-leaf(y), starting at letter depth of x (leaf rank + SA)
    - check that P matches these letters

$\Rightarrow O((|P| + |\text{output}|) \cdot \text{cost for SA lookup})$

↪ # matches via # leaves in subtree

↪ enumerate k matches via leaf select+rank

Grossi & Vitter achieve $O(\frac{P}{\log_\Sigma T} + |\text{output}| \cdot \log_\Sigma^\varepsilon T)$

↖ word RAM

Leaf ops:

leaf = ( )

leaf-rank = $\text{rank}_{(}$ (here)

leaf-select = $\text{select}_{(}(i)$

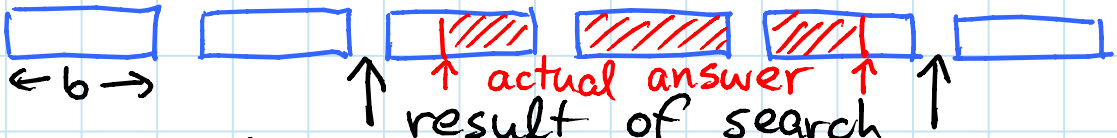
leaves in subtree = $\text{rank}_{(}$ (matching) of parent) − $\text{rank}_{(}$ (here)

leftmost-leaf = leaf-select(leaf-rank(here) + 1)

rightmost-leaf = leaf-select(leaf-rank(matching) of parent − 1))

"Succinct" suffix tree, given suffix array [MRR'01]

- store answer to every query with $|P| < b$
  $\Rightarrow O(2^b \lg n)$ bits $= O(\sqrt{n})$ if $b \le \frac{1}{3}\lg n$
- now consider $|P| \ge b$
- use suffix tree above on every bth suffix in suffix order (keep every bth leaf in tree)
- search in tree narrows to interval of size-b blocks in SA:



$\Rightarrow$ need to find first & last match in a block

- lookup table: (not really needed)
  for any b b-bit strings in sorted order & any $\le b$-bit query string, find first & last prefix match
  $\Rightarrow O(2^{b^2+b} \lg b)$ bits $= O(\sqrt{n})$ if $b \le \frac{1}{5}\sqrt{\lg n}$
  (2^{b^2}, 2^b, $O(\lg b)$)
- to search in a block:
  ① locate the b suffixes in T using SA
  ② read next b bits from P & from all b suffixes (in T) & use lookup table to narrow range
  ③ repeat $\lceil |P|/b \rceil$ times

  ① } $O(b)$ · SA query + ② } $O(b)$ · $O(P/b)$

$\Rightarrow O(P)$ time + $O(b)$ · cost of SA query
$\Rightarrow O(P + \lg^{\varepsilon} n)$ time for $b = O(\lg^{\varepsilon} n)$
$\Rightarrow$ dominated by $O(P \cdot$ SA query$)$ from previous struct.

- $O(\frac{n}{b})$ bits plus size of suffix array