Lecture 22

Leaf operations:

leaf	rightmost child of parent	( ))
leaf rank		$\text{rank}_{())}(\text{here})$
leaf count in subtree	leaf count(node) + leaf count(right sibs)	$\text{rank}_{())}[\text{matching) of parent}] - \text{rank}_{())}(\text{here})$
leaf select		$\text{select}_{())}(i)$
leftmost leaf in subtree		leaf select(1 + leaf rank(here))
rightmost leaf in subtree		leaf select(leaf rank(char. before matching) of parent))

Compact suffix arrays/trees

* TODAY: Grossi & Vitter [STOC 2000; SICOMP 2005]:
  $$\left(1/\varepsilon + O(1)\right) \underbrace{|T| \lg |\Sigma|}_{\text{OPT}} \text{ bits}$$
  $$O\left(\frac{|P|}{\log_{|\Sigma|} |T|} + |\text{output}| \cdot \log_{|\Sigma|}^{\varepsilon} |T|\right) \text{ query}$$

- Ferragina & Manzini [FOCS 2000]:
  $$5 \cdot \underbrace{H_k(T) \cdot |T|}_{k\text{th-order ``compressed OPT''}} + O\left(\frac{|T|}{\lg |T|}\left(|\Sigma| + \lg\lg |T|\right) + |T|^{\varepsilon} |\Sigma| 2^{|\Sigma| \lg |\Sigma|}\right)$$
  bits, for any fixed k
  - $H_k(T)$ = kth order empirical entropy
    $$= \sum_{|w|=k} \underbrace{\Pr\{w \text{ occurring}\}}_{= \#\text{occurrences}/|T|} \cdot H_0(\text{string of successor characters of } w)$$
    = # bits/char. in OPT code depending on last k chars.
  $$O(|P| + |\text{output}| \cdot \lg^{\varepsilon} |T|)$$
- Sadakane [J. Alg. 2003]
  $$1/\varepsilon \cdot H_0(T) \cdot |T| + O(|T| \lg\lg |\Sigma| + |\Sigma| \lg |\Sigma|) \text{ bits}$$
  $$O(|P| \lg |T| + |\text{output}| \cdot \lg^{\varepsilon} |T|) \text{ query}$$
- Grossi, Gupta, Vitter [SODA 2003]: succinct
  $$H_k(T) \cdot |T| + O\left(|T| \lg |\Sigma| \cdot \frac{\lg\lg |T|}{\lg |T|}\right) \text{ bits}$$
  $$O(|P| \lg |\Sigma| + \lg^{O(1)} |T|) \text{ query}$$
- low-space construction: [Han, Sadakane, Sung – FOCS 2003]
  $O(|T| \lg |\Sigma|)$ bits working space
  $O(|T|)$ time for suffix array; $O(|T| \lg^{\varepsilon} |T|)$ for suffix tree
- more "suffix-tree like" [Han & Sadakane – CPM 2002]
  e.g. find maximal common substrings

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

- follow divide & conquer of linear-time suffix tree alg. but 2-way instead of 3-way
- start: text $T_0 = T$
  - size $n_0 = n$
  - suffix array $SA_0 = SA$:
    $SA[i]$ = index in T of $\underbrace{\text{ith suffix}}_{\text{in sorted order of suffixes of } T}$ of T
- step: text $T_{k+1} = \langle (T_k[2i], T_k(2i+1)) \text{ for } i = 0, 1, \dots, n/2 \rangle$
  - size $n_{k+1} = n_k/2 = n/2^k$
  - suffix array SA_{k+1}
    $= \frac{1}{2} \cdot$ extract even entries of SA_k
- represent SA_k using SA_{k+1}:
  1. $\text{even-succ}_k(i) = \begin{cases} i & \text{if } SA_k[i] \text{ is even} \\ j & \text{if } SA_k[i] = SA_k[j] - 1 \text{ odd} \end{cases}$
  2. $\text{even-rank}_k(i)$ = # even suffixes preceding ith suffix
     = # even values in $SA_k[:i]$
  3. $$SA_k[i] = \underbrace{2 \cdot \underbrace{SA_{k+1}[\underbrace{\text{even-rank}_k(\underbrace{\text{even-succ}_k(i)}_{\text{round to even suffix} \Rightarrow \text{in } SA_{k+1}})}_{\text{name of even suffix in } SA_{k+1}}]}_{\text{index of even suffix in } T_{k+1}}}_{\text{index of even suffix in } T_k} - \underbrace{\{1 \text{ if } SA_k[i] \text{ odd}\}}_{\text{round back}}$$
- stop recursion at level $\ell = \lg \lg n \Rightarrow n_\ell = n / \lg n$
  $\Rightarrow$ can afford naïve $n_\ell \lg n_\ell \le n$-bit encoding

$\Rightarrow O(\lg \lg n)$ query to $SA[i]$ … if even-succ & even-rank & is-even-suffix are $O(1)$.

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

$\underline{\text{even-rank}_k}$ = rank_1 in is-even-suffix ~ $O(n_k \frac{\lg \lg n_k}{\lg n_k})$ bits

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

- 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 $\Rightarrow \text{even-succ}_k(i) = \text{odd-answers}[\underbrace{\text{odd-rank}_k(i)}_{i - \text{even-rank}_k(i)}]$

= order by odd suffix $T_k[SA_k[i]:]$

= order by $(T_k[SA_k[i]], T_k[\underbrace{SA_k[i]+1}_{\text{even}}:])$

= order by $(T_k[SA_k[i]], T_k[SA_k[\text{even-succ}_k(i)]:])$

= order by $(T_k[SA_k[i]], \underbrace{\text{even-succ}_k(i)}_{\text{answer}})$ (SA_k is suffix array)

- actually store these pairs ~ in order by value (assuming $|\Sigma| = 2$)

$\Rightarrow$ 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 by unary differential encoding: $0^{\text{lead}(v_1)} 1 \, 0^{\text{lead}(v_2) - \text{lead}(v_1)} 1 \dots$
  - $\Rightarrow n_k/2$ 1's & $\le 2^{\lg n_k} = n_k$ 0's $\Rightarrow \le \frac{3}{2} n_k$ bits
  - random access: $\text{leading}(v_i) = \text{rank}_0(\text{select}_1(i))$
- store trailing 2^k bits of each v_i explicitly in array
  - $\Rightarrow 2^k \cdot n_k/2 = n/2$ bits

$\Rightarrow \frac{1}{2} n + \frac{3}{2} n_k + O(\frac{n_k}{\lg \lg n_k})$ bits

$$\underline{\text{Total space}}: \sum_{k=1}^{\lg \lg n} \left[n_k + O\left(n_k \frac{\lg \lg n_k}{\lg n_k}\right) + \frac{1}{2} n + \frac{3}{2} n_k + O\left(\frac{n_k}{\lg \lg n_k}\right) \right]$$
$$= \frac{1}{2} n \lg \lg n + 3n + O\left(\frac{n}{\lg \lg n}\right) \text{ bits}$$

Compact suffix array: $O(n)$ bits [Grossi & Vitter]

- 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" now means "even $\varepsilon\ell$ times" (in $SA_{(k+1)\varepsilon\ell}$)
  - is-even$_k$: $n_{k\varepsilon\ell}$-bit vector as before + rank$_1$ structure
  - $\text{succ}_k(i) = j$ 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
  - to compute $SA_{k\varepsilon\ell}[i]$:
    1. follow succ$_k$ repeatedly until j is in $SA_{(k+1)\varepsilon\ell}$
    2. recurse: $SA_{(k+1)\varepsilon\ell}[\text{rank}_1(j)]$ is-even$_k$ bit vect.
    3. multiply by $2^{\varepsilon\ell}$, subtract by # calls to succ$_k$

$\Rightarrow$ search cost $\le 2^{\varepsilon\ell} = \lg^\varepsilon n$ succ$_k$ calls per level
$= O(\lg^\varepsilon n \lg\lg n) = O(\lg^{\varepsilon'} n)$ time

- space: $\sum_{k=0}^{1/\varepsilon} \left[n_{k\varepsilon\ell} + O\left(n_{k\varepsilon\ell} \frac{\lg\lg n_{k\varepsilon\ell}}{\lg n_{k\varepsilon\ell}}\right) + n + 2n_{k\varepsilon\ell} + O\left(\frac{n_{k\varepsilon\ell}}{\lg\lg n_{k\varepsilon\ell}}\right)\right]$
  $= 7(1/\varepsilon + 1)n + O\left(\frac{n}{\lg\lg n}\right)$ bits
- optimizations:
  - succ$_k$ free at level $k=0$ $\Rightarrow$ $\le 4n_{\varepsilon\ell} = O\left(\frac{n}{\lg^\varepsilon n}\right)$ bits
  - store is-even bit vector as succinct dictionary (with rank)
    $\Rightarrow \underbrace{\lg \binom{n_{k\varepsilon\ell}}{n_{(k+1)\varepsilon\ell}}}_{\approx\, n_{(k+1)\varepsilon\ell} \lg \frac{n_{k\varepsilon\ell}}{n_{(k+1)\varepsilon\ell}}} + O\left(n_{k\varepsilon\ell} \frac{(\lg\lg n_{k\varepsilon\ell})^2}{\lg n_{k\varepsilon\ell}}\right)$ bits
    $\frac{n_{k\varepsilon\ell}}{n_{(k+1)\varepsilon\ell}} = 2^{\varepsilon\ell}$
    $= O\left(n \frac{\lg\lg n}{\lg^\varepsilon n}\right)$ bits

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

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

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.
  "skipping the skips" — don't know edge lengths
- as search for P proceeds, 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

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

↳ # matches via leaf size; enumerate k via leaf select & rank

"Succinct" suffix tree, given suffix array [Munro, Raman, Rao]

- use structure above on every $b = \frac{1}{2}\sqrt{\lg n}$ suffixes in suffix order (i.e. keep every bth item/leaf of suffix array/tree)
- search narrows to an interval of size-b blocks in SA:

- need to find first & last match in a block
- lookup table: for any b b-bit strings in sorted order, for any $\le b$-bit query string, find first & last prefix match
  $\Rightarrow O(2^{b^2} 2^b \lg b) = O(\sqrt[4]{n}\, 2^{\frac{1}{2}\sqrt{\lg n}} \lg\lg n) = o(n^{\frac{1}{4}+\varepsilon})$ bits
- to search in a block:
  1. read next b bits from all b suffixes in block & from P
     via T
     $\Rightarrow O(b)$ time
  2. repeat $\lceil \frac{|P|}{b} \rceil$ times
     $\Rightarrow O(|P| + b)$ time at bottom
     ↳ reducible to $\lg^\varepsilon n$ by reducing b
- $O(\frac{n}{b})$ bits plus size of suffix array

Grossi & Vitter achieve $O\left(\frac{|P|}{\log_{|\Sigma|} |T|} + |\text{output}| \cdot \log^\varepsilon_{|\Sigma|} |T|\right)$ query & $O(|T|)$ bits