6.851	Class 10	Apr. 16, 2014

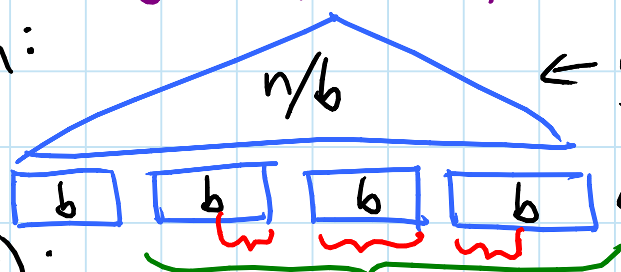
<u>TODAY</u>: succinct DS review

- succinct/compact suffix trees
  (succinct: $+o(T)$; compact: $\hookrightarrow O(T)$)
- compact/compressed suffix arrays
  (compact: $O(T)$; compressed: $O(T \lg\lg T)$)
- succinct rank/select

<u>Succinct suffix tree</u>, given suffix array (SA):

[Munro, Raman, Rao 2001]

- <u>indirection</u>:



← compact suffix tree ⇒ $O(T/b)$ bits

← SA (sorted suffixes)

- <u>search</u>(P):
  - top search narrows to interval of blocks
  - refine within first & last block:
    - follow all b SA pointers in block
    - read first b bits from each T suffix
    - compare to first b bits of P — $O(b)$
    - repeat $O(\lceil P/b \rceil)$ times

    $\Rightarrow O(b \cdot \text{SA query} + P)$ time if $|P| \geq b$

    $= O(P + \lg^{\varepsilon} n)$ for b & SA query $= O(\lg^{\varepsilon'} n)$

    (will be dominated by compact suffix tree)
  - for $|P| \leq b$: store first & last match $\forall P$

  $\Rightarrow O(2^b \lg n)$ bits $= O(n^{\varepsilon})$ for $b \leq \varepsilon' \lg n$

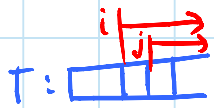
Compact suffix tree: [MRR'01]

- store structure of suffix tree as augmented balanced parens $\Rightarrow O(n)$ bits
- length of edge $x \to y$
  $= $ length of prefix match between
  $T[SA[\text{leftmost-leaf}(x)] + \text{letter-depth}(x):]$
  $T[SA[\underbrace{\text{rightmost-leaf}(x)}_{\text{rank}_{()}(\text{matching}_{(}(\text{parent}(x)) - 1))}] + \text{letter-depth}(x):]$
- search(P):
  - maintain letter-depth(x) (initially $\emptyset$)
  - compare P with each edge traversed
  - stop if mismatch

  $\Rightarrow O(P \cdot \text{SA query})$
  - additional SA query per output

Compact suffix array: [Grossi & Vitter 2000]

- $T_0 = T$, $SA_0 = SA$
- $T_k = T_{k-1}$ with every 2 letters combined
- $SA_k = \frac{1}{2}$(even entries of SA_{k-1})
- only store $1/\varepsilon + 1$ levels: $k \cdot \varepsilon \cdot l$, $l = \lg\lg n$
- represent $SA_{k\varepsilon l}$ using $SA_{(k+1)\varepsilon l}$ plus:
  - which suffixes divisible by $2^{\varepsilon l} = \lg^{\varepsilon} n$ $\}\ n_k$ bits
  - $rank_1$ DS on this bit vector
  - $succ_k(i) = j$ if $SA_k[j] = SA_k[i] + 1$



$\Rightarrow SA_k[i] = 2^{\varepsilon l} \cdot SA_{k+1}[rank(\underbrace{succ_k^{(x)}(i)}_{\text{iterate } x \text{ times until divisible by } 2^{\varepsilon l}})] - x$

$succ_k$ representation:

- store $(\underbrace{T_k[SA_k[i]]}_{2^k \text{ bits}}, \underbrace{succ_k(i)}_{\lg n_k \text{ bits}})$ for all i
- these bit strings are sorted (by i)
- store leading $\lg n_k$ bits in unary differential:
  $0^{lead(v_1)}\, 1\, 0^{lead(v_2) - lead(v_1)}\, 1 \ldots$

$\Rightarrow O(n_k)$ bits & $lead(v_i) = rank_0(select_1(i))$

- store trailing 2^k bits explicitly

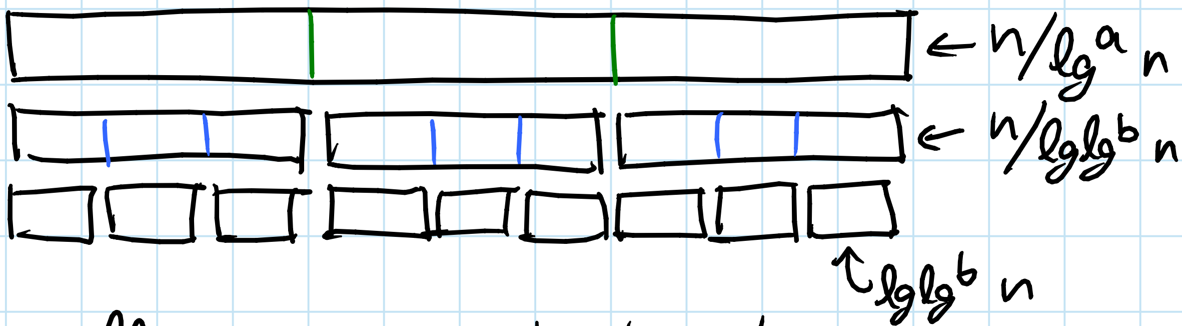
$\Rightarrow n$ bits

total: $\left(\frac{1}{\varepsilon} + \underbrace{O(1)}_{\text{can be } 1+o(1)}\right) n$ bits

query: $O(\underbrace{\lg^{\varepsilon} n}_{\#\text{ steps/level}} \cdot \underbrace{\lg\lg n}_{\#\text{ levels}})$ time $= O(\lg^{\varepsilon'} n)$

Succinct rank/select:

- divide n-bit string into blocks of $\lg^a n$ bits
- divide blocks into subblocks of $\lg\lg^b n$ bits



- store all answers at the top:
  $o(n)$ if answer $= o(\lg^a n)$ bits
- use lookup table on bottom:
  $2^{\lg\lg^b n} \ll n^\varepsilon$ possible subblocks
- in middle: deal with local ranks $\le \lg^a n$
  (ranks within block) $\Rightarrow O(\lg\lg n)$ bits
  $\Rightarrow$ can store all answers in $o(n)$ bits
  if $< b$ local ranks
  - select splits by 1 bits $\Rightarrow$ 2 cases:
    - sparse ($> \lg^{2a} n$ bits) $\Rightarrow$ afford global indices of 1s
    - dense ($\le \lg^{2a} n$ bits) $\Rightarrow$ store local indices of 1s

Problem: given a balanced n-parenthesis string, design a static DS with $o(n)$ extra bits of space supporting in $O(1)$ time:

- **match**(i): find the matching parenthesis of paren at position i

Hint: divide the string into
- blocks of size polylg n
- subblocks of size polylglg n