Today: succinct DS review
- succinct/compact suffix trees
  \( o(T) \mapsto O(T) \)
- compact/compressed suffix arrays
  \( O(T) \mapsto O(T \log^2 T) \)
- succinct rank/select

**Succinct suffix tree**, given suffix array (SA):

[Munro, Raman, Rao 2001]

- **indirection**:
  ![Diagram of indirection]

- **search** \( (P) \):
  - 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 \( P \)
    - repeat \( O(\lceil P/b \rceil) \) times
    \( \Rightarrow O(b \cdot \text{SA query} + P) \) time if \( |P| > b \)
    \( = O(P + \log^2 n) \) for \( b \) & SA query = \( O(\log^2 n) \)
    (will be dominated by compact suffix tree)
  - for \( |P| \leq b \): store first & last match \( A \) \( P \)
    \( \Rightarrow O(2^b \log n) \) bits = \( O(n^\varepsilon) \) for \( b \leq \varepsilon \log n \)
Compact suffix tree: \[MRR'01\]

- Store structure of suffix tree as augmented balanced parens \(\Rightarrow O(n)\) bits
- Length of edge \(\text{match}(x)\) = length of prefix match between \(T[SA[\text{leftmost-leaf}(x)] + \text{letter-depth}(x);]\) and \(T[SA[\text{rightmost-leaf}(x)] + \text{letter-depth}(x);]\) rank\(_{\text{match}}\) \((\text{parent}(x) - 1)\)

- \(\text{search}(p)\):
  - 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: \cite{GrossiVitter2000}

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

succ_k representation:
- store \( (T_k[SA_k[i]], succ_k(i)) \) for all \( i \)
  \( 2^k \) bits \( \log n_k \) bits
- these bit strings are sorted (by \( i \))
- store leading \( \log n_k \) bits in unary differential:
  \( \text{lead}(v_1) 1 \text{lead}(v_2) - \text{lead}(v_1) 1 \ldots \)
  \( \Rightarrow O(n_k) \) bits & \( \text{lead}(v_i) = \text{rank}_0(\text{select}_1(i)) \)
- store trailing \( 2^k \) bits explicitly
  \( \Rightarrow n \text{ bits} \)

total: \( \left( \frac{1}{2^k} + O(1) \right) n \text{ bits} \)

query: \( O(lg^2 n \cdot lg lg n) \) time = \( O(lg^\varepsilon n) \)

\# steps/level/\# levels
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: $2^{\lg \lg^b n} \ll n^2$ possible subblocks
- in middle: deal with local ranks $\leq \lg^a n$ (ranks within block) ⇒ $O(\lg \lg n)$ bits
  ⇒ can store all answers in $o(n)$ bits if $b$ local ranks
- select splits by 1 bits ⇒ 2 cases:
  - sparse ($\lg^2 n$ bits) ⇒ afford global indices of 1s
  - dense ($\leq \lg^2 n$ bits) ⇒ 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 polylog n
- subblocks of size polylog log n