Today: succinct data structures I (of 2)
- Survey
- succinct binary tries
  - level-order
  - via balanced parentheses
- succinct rank & select

Goal: small space, often static

**Implicit DS:** space = \( \text{OPT} + O(1) \) bits
  - information theoretic for rounding
  - typically, DS is "just the data",
    permuted in some order
  - e.g. sorted array, heap

**Succinct DS:** space = \( \text{OPT} + o(\text{OPT}) \)
  - lead constant of 1

**Compact DS:** space = \( O(\text{OPT}) \)
  - often a factor of \( w \) smaller than
    "linear-space" data structures
  - e.g. suffix trees use \( O(n) \) words
    for \( n \)-bit string
Minisurvey:
- implicit dynamic search tree: [Franceschini & Grossi - ICALP 2003/WADS 2003]
  $O(lg n)$ worst-case time/insert, delete, predecessor
  also $O(lg_B N)$ cache oblivious
- succinct dictionary: [Brodnik & Munro - SICOMP 1999; Pagh - SICOMP 2001]
  $n lg_{1/n} n = lg(n) + O(n \frac{lg lg n}{lg n})$ bits
- $O(1)$ membership query (static)
- succinct binary trie: [Munro & Raman - SICOMP 2001]
  $C_n = (\frac{2^n}{n})/\binom{n+1}{2} \sim 4^n$ such tries (Catalan)
  $lg C_n + o(lg C_n) = 2n + o(n)$ bits
- $O(1)$ left child, right child, parent, subtree size
- $O(1)$ ins/del. leaf, subdivide edge [Farzan & Munro - TCS 2011]
- succinct k-ary trie: (e.g. suffix tree) [Farzan & Munro - SWAT 2008]
  $C_k^n = (\frac{k^{n+1}}{n})/\binom{n+1}{2}$ tries, $lg C_k^n + o$ bits
- $O(1)$ child with label $i$, parent, subtree size, ...

improving [Benoit, Demaine, Munro, Raman, Raman, Rao - Algorithmica 2005]
- succinct permutations: [Munro, Raman, Raman, Rao - ICALP 2003]
  $lg n! + o(n)$ bits, $O(\frac{lg n}{lg lg n})$ time to compute $\pi^k(x) \forall k (1+\epsilon) n lg n$ bits, $O(1)$ time $\pi^k$ (including $k<0$)

- generalizes to functions [Munro & Rao - ICALP 2004]
- compact Abelian groups: [Farzan & Munro - ISSAC 2006]
  $O(lg n)$ bits for group of order $n$ (!) or elt. in group
  $O(1)$ multiply, inverse, equality testing
- graphs [Farzan & Munro - ESA 2008; Barbay, Aleardi, He, Munro - Alg. 2012]
- implicit n-bit ints: inc./dec. in $O(lg n)$ bit reads
  & $O(1)$ bit writes [Rahman & Munro - Alg. 2010]

OPEN: $O(1)$ word RAM?
Level-order representation of binary tries: [Munro]

For each node in level order:
- Write 0/1 for whether have left child
- Write 0/1 for whether have right child

⇒ 2n bits

E.g:

A
B C
D E
G

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(1) 1 1 0 1 1 1 0 1 0 0 0 0 0 0 0
A B C • D E F • G • • • • • • •

Equivalently:
- Append external node (•) for each missing child
- For each node in level order:
  - Write 0 if external, 1 if internal

⇒ Extra leading 1 (2n+1 bits)
**Navigation:** (in external-node view)
left & right children of $i$th internal node are at positions $2i$ & $2i+1$

**Proof:** by induction on $i$:
- just after $(i-1)$st internal node’s children (as external nodes have no children)
- either same level $i-1$ or new $i$

- $i$th internal children
- $i-1$ internals
- $j$ externals
- pos. $i+j$
- remaining children of $i-1$
  - $= 2(i-1) - (i-1) - j$
  - all internal seen
  - root external seen

  - $= i - j - 1$

**Rank & Select** in bit string:
- $\text{rank}_1(i) = \# 1’s$ at or before position $i$
- $\text{select}_1(j) = \text{position of } j\text{th } 1 \text{ bit}$

$\Rightarrow$ left-child$(i) = 2 \cdot \text{rank}_1(i)$
right-child$(i) = 2 \cdot \text{rank}_1(i) + 1$
parent$(i) = \text{select}(\lfloor i/2 \rfloor)$

*(but subtree-size impossible in level-order rep)*
1. Use lookup table for bitstrings of length $\frac{1}{3} \lg n$ 

   $\Rightarrow O(\sqrt{n \lg n \lg \lg n})$ bits of space 

   bitstring, query, answer

2. Split into $(\lg^2 n)$-bit chunks:

   \[ \frac{n}{\lg^2 n} \]

   Store cumulative rank: $\lg n$ bits

   $\Rightarrow O(\frac{n}{\lg^2 n} \lg n) = O(\frac{n}{\lg n})$ bits 

   (couldn't afford $\lg n$-bit chunks)

3. Split each chunk into $(\frac{1}{2} \lg n)$-bit subchunks:

   \[ \frac{n}{\frac{1}{2} \lg n} \]

   Store cumulative rank within chunk: $\lg \lg n$ bits

   $\Rightarrow O(\frac{n}{\frac{1}{2} \lg n} \lg \lg n) = o(n)$ bits

4. Rank = rank of chunk 

   + relative rank of subchunk within chunk 

   + relative rank of element within subchunk (via lookup table)

   $\Rightarrow O(1)$ time, $O(n \frac{\lg \lg n}{\lg n})$ bits

- $O(\frac{n}{\lg^k n})$ bits possible for any $k = O(1)$ 
  [Pătraşcu - FOCS 2008]

- $O(\frac{\lg n}{\lg \lg n})$ insert/delete/rank/select 
  [He & Munro - SPIRE 2010]
Select: [Clark & Munro - Clark's PhD 1996]

1. Store array of indices of every \((\log n \log \log n)\)th 1 bit
   \[\Rightarrow O\left(\frac{n}{\log n \log \log n} \log n\right) = O\left(\frac{n}{\log n} \log n\right)\text{ bits}\]

2. Within group of \(\log n \log \log n\) 1 bits, say \(r\) bits:
   if \(r \geq (\log n \log \log n)^2\)
   then store array of indices of 1 bits in group
   \[\Rightarrow O\left(\frac{n}{\log n \log \log n} \cdot (\log n \log \log n) \log n\right) = O\left(\frac{n}{\log n} \log n\right)\text{ bits}\]
   
   else reduced to bitstring of length \(r \leq (\log n \log \log n)^2\)

3. Repeat 1&2 on all reduced bitstrings to reduce to bitstrings of length \((\log \log n)^9(1)\)

4. Store relative index \((\log \log n)\) bits) of every \((\log \log n)^2\)th 1 bit \((\log n \log \log n \approx \log \log n\) also OK but bigger)
   \[\Rightarrow O\left(\frac{n}{(\log \log n)^2} \cdot (\log \log n) \log n\right) = O\left(\frac{n}{\log \log n} \log n\right)\text{ bits}\]

5. Within group of \((\log \log n)^2\) 1 bits, say \(r\) bits:
   if \(r \geq (\log \log n)^4\)
   then store relative indices of 1 bits
   \[\Rightarrow O\left(\frac{n}{(\log \log n)^4} \cdot (\log \log n)^3 \log \log n\right) = O\left(\frac{n}{\log \log n} \log n\right)\text{ bits}\]
   
   else reduced to bitstring of length \(r \leq (\log \log n)^4\)

4. Use lookup table for bitstrings of length \(\leq \frac{1}{10} \log n\)
   \[\Rightarrow O\left(\sqrt{n} \log n \log \log n\right)\]
   
   \# bitstrings query \(j\) answer

\[\Rightarrow O(1) \text{ query, } O\left(\frac{n}{\log \log n}\right) \text{ bits}\]
\[- O\left(n / \log ^k n\right) \text{ bits } \forall k = O(1) \quad [Pătraşcu - FOCS 2008]\]
Binary tries as balanced parentheses: [Munro & Raman - STOC'01]

Binary trie: ordered tree

node - first child
left child - next sibling
right child - prev. sibling
parent - node or parent

Balanced paren: (= bitstring)

(( () ( )( ) )( ) )
*ABBCCDDAEEFFE GG*

Left paren: [ & matching right]
Next char: [if else none]
Char after matching: [if ]
Prev. char: ) => its matching:
Prev. char: ( => that:
\frac{1}{2} distance to enclosing

Rank( ) of enclosing
- rank( ) of here

- Similar to (& using) rank & select, can find matching & enclosing parens. in O(1) time, O(n) space
- All operations above in O(1) time
- From subtree size can accumulate index of node for auxiliary data (e.g. pointer to text)