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 = OPT + O(1) bits
- typically, DS is “just the data”, permuted in some order
- e.g. sorted array, heap

Succinct DS: space = OPT + o(OPT)
- lead constant of 1

Compact DS: space = O(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]
  \( O(\log n) \) worst-case time/insert/delete/predecessor
  also \( O(\log_B n) \) cache oblivious
- succinct dictionary:  
  [Brodinik & Munro - SICOMP 1999; Pagh - SICOMP 2001]
  \( \frac{n\lg n}{n} = \lg(n) + O(n \frac{\log \log n^2}{\log n}) \) bits
  \( O(1) \) membership query (static)
- succinct binary trie:  
  [Munro & Raman - SICOMP 2001]
  \( C_n = \binom{2^n}{n}/(n+1) \sim 4^n \) 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:  
  [Farzan & Munro - SWAT 2008]
  \( C^k_n = \binom{kn+1}{n}/(kn+1) \) tries, \( \lg C^k_n + o \) bits
  \( O(1) \) child with label \( i \), parent, subtree size...

- improving [Benoît, Demaine, Munro, Raman, Raman, Rao - Algorithmica 2005]
- succinct permutations:  
  [Munro, Raman, Raman, Rao - ICALP 2003]
  \( \lg n! + o(n) \) bits, \( O(\log \log n) \) time to compute \( \pi^k(x) \forall k \)
  \( (1+\varepsilon) n\lg n \) bits, \( O(1) \) time \( \pi^k \) (including \( k<0 \))
- generalizes to functions [Munro & Rao - ICALP 2004]
- compact Abelian groups:  
  [Farzan & Munro - ISSAC 2006]
  \( O(\log n) \) 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 \( O(\log n) \) 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   F
     /   /\   /\   /
    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
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 $(i-1)\text{st}$ internal node's children
  (as external nodes have no children)
- either same level $i^{-1}$ or new $i$

\[ \text{i-th internal children} \]

\[ \text{i-1 internals} \quad j \text{ externals} \]

\[ \text{pos. } i+j \]

\[ \text{remaining children of } i-1 \]
\[ = 2(i-1) - (i-1) - j \]
\[ = i-j-1 \]

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

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

(but subtree-size impossible in level-order rep)
Rank: [Jacobsen - FOCS 1989]

1. Use lookup table for bitstrings of length $\frac{1}{2} \lg n$  
   $\Rightarrow O(\sqrt{n} \lg n \lg \lg n)$ bits of space

2. Split into $O(\frac{n}{\lg^2 n})$-bit chunks:
   $\Rightarrow O(\frac{n}{\lg^2 n} \lg n) = O(\frac{n}{\lg n})$ bits
   (couldn't afford $\lg n$-bit chunks)

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

- $O(\frac{n}{\lg^2 n})$ 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
   \[ O\left(\frac{n}{\log n \log \log n}\log n \right) = O\left(\frac{n}{\log n} \right) \text{ bits} \]

2. Within group of \( \log n \log \log n \log n \) 1 bits, say \( r \) bits:
   - If \( r \geq (\log n \log \log n)^4 \)
     - Then store array of indices of 1 bits in group
       \[ O\left(\frac{n}{\log n \log \log n} \log n \right) = O\left(\frac{n}{\log n} \right) \text{ bits} \]
     - # such groups  #1 bits index
   - Else reduced to bitstring of length \( r \leq (\log n \log \log n)^3 \)

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

4. Store relative index \( (\log n \log \log n \text{ bits}) \) of every \( (\log \log n)^3 \)th 1 bit (\( (\log n \log \log n \log \log n \) also OK but bigger)
   \[ O\left(\frac{n}{(\log \log n)^3 \log \log n}\right) = O\left(\frac{n}{\log n} \right) \text{ bits} \]

5. Within group of \( (\log \log n)^3 \) 1 bits, say \( r \) bits:
   - If \( r \geq (\log \log n)^4 \)
     - Then store relative indices of 1 bits
       \[ O\left(\frac{n}{(\log \log n)^4 (\log n \log \log n)} \log \log n \right) = O\left(\frac{n}{\log n} \right) \text{ bits} \]
     - # such groups  #1 bits rel. index
   - Else reduced to bitstring of length \( r \leq (\log \log n)^4 \)

4. Use lookup table for bitstrings of length \( \leq \frac{1}{2} \log n \)
   \[ O\left(\sqrt{n \log n \log \log n} \right) \]
   # bitstrings query answer

\[ O(1) \text{ query, } O\left(\frac{n}{\log n} \right) \text{ bits} \]
\[ O(n/\log k n) \text{ bits } \forall \ k=O(1) \] [Pătraşcu - FOCS 2008]
Binary tries as balanced parentheses: [Munro & Raman - SICOMP 2001]

```
<table>
<thead>
<tr>
<th>binary trie</th>
<th>ordered tree</th>
<th>balanced paren (=bitstring)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A E G</td>
<td>*</td>
<td>( ( ( ( ( ( ( ( ( ) ) ) ) ) ) ) )</td>
</tr>
<tr>
<td>B CD F</td>
<td></td>
<td><em>ABBCCDĐAEEFEGG</em></td>
</tr>
</tbody>
</table>
```

- node
- left child
- right child
- parent
- subtree size
- # leaves in subtree
- size(node)
- sizes(right siblings)
- size(ancestor)

left paren. [& matching right]
next char. [if (; else none]
char. after matching ) [if (]
pren. char. ) ⇒ its matching (prev. char. ( ⇒ that (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, $O(h)$ space
- All operations above in $O(1)$ time
- From subtree size can accumulate index of node for auxiliary data (e.g. pointer to text)