**Succinct data structures**: small space, usually static

*Implicit DS*: space = information-theoretic OPT + \(O(1)\) bits (for rounding)
- typically, data structure is “just the data”, permuted in some order

*Succinct DS*: space = information-theoretic OPT + \(o(OPT)\)
- lead constant of 1

*Compact DS*: space = \(O(\text{information-theoretic OPT})\)
- still better than “linear-space” data structures
- on e.g. n-bit strings by factor of \(O(w)\)
Mini-survey:
- **implicit** dynamic search tree:
  \[ \text{Franceschini & Grossi – ICALP 2003 & WADS 2003} \]
  \( O(\log n) \) worst case/insert, delete, search
  (cf.: heap & sorted array)
  comparison model on elements
  word RAM for pointers

- **succinct** dictionary:
  \[ \text{Brodwnik & Munro – SICOMP 1999; Pagh – SICOMP 2001} \]
  \[ \lg (\binom{u}{n}) + O(n \frac{(\log \log n)^2}{\log n}) \] bits
  \( O(1) \) membership query

* **succinct** binary trie:
  \[ \text{Munro & Raman – SICOMP 2001} \]
  \[ C_n = \binom{2^n}{n+1} \sim 4^n \] such tries
  \[ \lg C_n + o(\lg C_n) = 2n + o(n) \] bits
  \( O(1) \) left child, right child, parent, subtree size

- **compact/succinct** k-ary trie:
  \[ \text{Benoit, Demaine, Munro, Raman, Raman, Rao – Algorithmica 2005} \]
  \[ C_k^n = \binom{kn+1}{n+1} \sim 2^{(\lg k + \lg e)n} \] such tries
  \( (\lg k + \lg e) n + o(n) \) bits
  \( O(1) \) child with label \( i \), parent, subtree size

- **succinct** rooted ordered tree \[ \text{Benoit et al. 2005} \]
  only \( C_n \) such trees
  \( 2n + o(n) \) bits
  \( O(1) \) \( i \)th child, parent, subtree size
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

\[ \rightarrow 2n \text{ bits} \]

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

More useful view:
1. append external node (•) for each missing child
2. visit all nodes in level order
3. write 1 for internal node, 0 for external

\[ \rightarrow \text{extra leading 1} \quad (2n+1 \text{ bits}) \]
Navigating level-order representation: (with external nodes)
left & right children of ith internal node are at positions i & 2i

Proof:
- suppose ith internal node at position i+j
- i.e. j external nodes up to ith external node
- i-1 previous internal nodes have 2(i-1) children
- i-1 already seen as internal nodes (all but root)
- j already seen as external nodes
⇒ left child at position (i+j) + \frac{2(i-1)-(i-1)-j+1}{2} = 2i.

Rank & Select in bit string:
rank_1(i) = # 1's at or before position i
select_1(j) = position of jth 1 bit

⇒ left-child(i) = 2 \cdot rank_1(i)
right-child(i) = 2 \cdot rank_1(i) + 1
parent(i) = select(Li/2)

(but no subtree size in level-order rep.)
Rank: [Jacobsen - FOCS 1989]

1. Use lookup table for bitstrings of length $\frac{1}{2} \lg n$
   $\Rightarrow O(\sqrt{n} \lg n \lg \lg n)$ space
   
   bitstring/query i/answer

2. Split into $n/\lg^2 n$ chunks:

   \[ \begin{array}{c}
   \hline
   \hline
   \hline
   \hline
   \hline
   \hline
   \hline
   \hline
   \hline
   \hline
   \end{array} \]

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

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

   \[ \begin{array}{c}
   \hline
   \hline
   \hline
   \hline
   \hline
   \hline
   \hline
   \hline
   \hline
   \end{array} \]

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

4. Rank query sums
   - rank of chunk
   - relative rank of subchunk
   - relative rank of element within subchunk (lookup table)

   $\Rightarrow O(1)$ time, $o(n)$ bits of space

Note: can't afford $\Theta(\frac{n}{\lg n})$ chunks
   with $(\lg n)$-bit cumulative ranks
Select: \[ (\text{Clark} \& \text{Munro} - \text{Clark's Ph.D. 1996}) \]

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

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
     \[ \Rightarrow O\left(\frac{n^{1/(\log \log n)^2}}{(\log n \log \log n) \log n}\right) = O\left(\frac{n^{1/(\log \log n)}}{(\log \log n)}\right) \text{ bits} \]
     #such groups/#1 bits/index size
   - else reduced to bitstring of length \(r \leq (\log n)^4\)

3. Repeat steps 1 & 2 on all reduced bitstrings to reduce to bitstrings of length \(O(\log \log n \log \log n)\):
   - store relative index \((\log \log n)\) bits of every \((\log \log n)^2\) th 1 bit
     \[ \Rightarrow O\left(\frac{n^{1/(\log \log n)^2}}{(\log \log n)^2 \log \log n}\right) = O\left(\frac{n^{1/(\log \log n)^2}}{(\log \log n)^2}\right) \text{ bits} \]
   - Within group of \((\log \log n)^2\) 1 bits, say \(r\) bits:
     - if \(r \geq (\log \log n)^4\) then store array of indices of 1 bits in group
       \[ \Rightarrow O\left(\frac{n^{1/(\log \log n)^4}}{(\log \log n)^2 \log \log n}\right) = O\left(\frac{n^{1/(\log \log n)^4}}{(\log \log n)^2}\right) \text{ bits} \]
       #such groups/#1 bits/index size
     - else reduced to bitstring length \(r \leq (\log n)^4\)

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

\[ \Rightarrow O(1) \text{ query, } o(n) \text{ bits of space} \]
Binary tries as balanced parentheses

\[ \text{balanced parenns} (=\text{bitstring}) \]
\[
((()()()()(())())
\]
\[ *A\ B\ B\ C\ C\ D\ D\ A\ E\ F\ F\ E\ G\ G* \]

- like (& using) rank & select, can find matching/enclosing parens in \( O(1) \) time with \( o(n) \) space
- all operations above in \( O(1) \) time