Today: Integers & van Emde Boas
- models: word RAM & cell probe
- predecessor problem
- van Emde Boas DS
- y-fast trees

Models for integer data structures:
- word = w-bit integer \( \in \{0, 1, \ldots, w-1\} \)
  \( \leq 2^w \)
- all elements: inputs, outputs, ...

- transdichotomous RAM (Random Access Machine):
  - memory = array of \( S \) words
  - operations read/write \( O(1) \) words
  - words serve as pointers
  \( \Rightarrow w \geq \lg S \)
  - in particular \( w \geq \lg n \)

- word RAM: transdichotomous RAM
  with C-style operations: 
  \([\), \(\), \(+\), \(-\), \(*\), \(/\), \(%\), \(<\), \(\), \&, \(\|\), \(^\)\), \^\)
  - standard model

- cell probe: count \# memory reads & writes
  - computation is free
  - unrealistic
  - useful for lower bounds

\[ \text{BST} \text{ WEAK} \]
\[ \text{transdichotomous RAM} \]
\[ \text{word RAM} \]
\[ \text{cell probe} \]
\[ \text{pointer machine} \]
\[ \text{bridging two worlds: machine/problem} \]
\[ \text{STRONG} \]
Predecessor problem: maintain set $S$ of $n$ words subject to:
- Insert ($x \in U$)
- Delete ($x \in S$)
- Predecessor ($x \in U$): max $\{y \in S | y < x\}$
- Successor ($x \in U$): min $\{y \in S | y > x\}$

- Harder than dictionaries/hashing
- Comparison model $\Rightarrow$ BST: $\Theta(lg n)$/op. optimal

- Word RAM:
  - Van Emde Boas: $O(lg w)$/op. $\Theta(n)$ space
  - $y$-fast trees: $O(lg w)$ w.h.p. $\Theta(n)$ space
    [Willard-IPL 1983]
  - Fusion trees: $O(lg w, n)$ w.h.p. $\Theta(n)$ space
    [Fredman & Willard-JCSS 1993; Raman-ESA 1996]

- $L_{12}$:
  - $\min: O(\sqrt{\frac{lg n}{lg \frac{lg w}{lg lg n}}})$ w.h.p. $\Theta(n)$ space
  - Cell probe lower bound: $\Omega(\sqrt{\frac{lg w}{lg \frac{lg w}{lg lg n}}})$
    $O(n \ poly(lg n))$ space $\Rightarrow \Omega(\min\{lg w, \frac{lg w}{lg \frac{lg w}{lg lg n}}\})$
    [Pătrașcu & Thorup - STOC 2006 & SODA 2007]
  - VEB optimal for $w = O(poly(lg n))$
  - Fusion trees optimal for $w = 2^{\Omega(lg n)}$

- Pointer machine, word specified by pointer:
  - Van Emde Boas: $O(lg lg w)$/op. $\Theta(n)$ space
  - Lower bound: $\Omega(lg lg u)$/op. $\Omega(lg w)$ space
    [Mehlhorn, Näher, Alt - SICOMP 1988]
van Emde Boas: (Peter) (reinterpreted by Bender & Farach-Colton)

- idea: \( T(u) = T(\sqrt{u}) + O(1) \)
  \( = O(\log \log u) \)

- split universe \( U \) into \( \sqrt{u} \) clusters, each size \( \sqrt{u} \)

- hierarchical coordinates: word \( x = <c, i> \)
  - \( c = x // \sqrt{u} \) = cluster containing \( x \)
  - \( i = x \% \sqrt{u} \) = \( x \)'s index within cluster

  integer division \( \uparrow \) mod

  - \( x = c \sqrt{u} + i \quad \Rightarrow O(1) \)-time conversion

- binary perspective:
  - split bits in half
  - \( c = \) high order = \( x \gg w/2 \)
  - \( i = \) low order = \( x \& (1 << w/2) - 1 \)
  - \( x = (c << w/2) | i \)

- recursive \( \text{vEB} \) \( V \) of size \( u \):
  - \( V.\text{cluster}[i] = \text{vEB} \) of size \( \sqrt{u} \) for \( 0 \leq i < \sqrt{u} \)
  - \( V.\text{summary} = \text{vEB} \) of size \( \sqrt{u} \) \& \( w' = w/2 \)
  - stores which clusters \( c \) are nonempty
  - \( V.\text{min} = \) minimum element in \( V \), not stored recursively
    or None if \( V \) is empty
  - \( V.\text{max} = \) (copy of) max. element in \( V \)
\textbf{Successor} \((V, x = \langle c, i \rangle)\):
- if \(x < V.\text{min}\): return \(V.\text{min}\) (special: not stored recursively)
- if \(i < V.\text{cluster}[c].\text{max}\):
  return \(\langle c, \text{Successor}(V.\text{cluster}[c], i) \rangle\)
- else: \(c' = \text{Successor}(V.\text{summary}, c)\)
  return \(\langle c', V.\text{cluster}[c'].\text{min} \rangle\)

\textbf{Insert} \((V, x = \langle c, i \rangle)\):
- if \(V.\text{min} = \text{None}\): \(V.\text{min} = V.\text{max} = x\); return \(O(1)\)
- if \(x < V.\text{min}\): swap \(x \leftrightarrow V.\text{min}\)
- if \(x > V.\text{max}\): \(V.\text{max} = x\)
- if \(V.\text{cluster}[c].\text{min} = \text{None}\): Insert \((V.\text{summary}, c)\) \(\Rightarrow\) next call is \(O(1)\)
- Insert \((V.\text{cluster}[c], i)\)

\textbf{Delete} \((V, x = \langle c, i \rangle)\):
- if \(x = V.\text{min}\):
  \(c = V.\text{summary}.\text{min}\)
- if \(c = \text{None}\): \(V.\text{min} = \text{None}\); return \(O(1)\) (now empty)
  \(x = V.\text{min} = \langle c, i = V.\text{cluster}[c].\text{min} \rangle\)
- Delete \((V.\text{cluster}[c], i)\)
- if \(V.\text{cluster}[c].\text{min} = \text{None}\): (empty now)
  Delete \((V.\text{summary}, c)\) \(\Rightarrow\) previous call \(O(1)\)
- if \(V.\text{summary}.\text{min} = \text{None}\): \(V.\text{max} = V.\text{min}\)
- else: \(c' = V.\text{summary}.\text{max}\)
  \(V.\text{max} = \langle c', V.\text{cluster}[c'].\text{max} \rangle\)
- node = OR of children
- path from leaf \( x \) to root is monotone
- could binary search for 0\( \rightarrow \)1 transition
- max/min of last 0's left/right sibling is predecessor/successor of \( x \) (if \( \mathcal{E} \) is)
- store sorted linked list on elements to find successor/predecessor
- query in \( O(lg lg u) \) ~ roughly same as above

- even in pointer machine \& \( O(u lg w) \) space: node stores linked list of pointers to ancestor of height \( 2^i \) for \( i = 0, 1, \ldots, lg w \)

- but updating these bits costs \( \Theta(lg u)/\text{op.} \)
- vEB's not-storing-min reduces to \( \Theta(lg w) \)

- again possible on pointer machine with \( O(u lg w) \) space [VEBKZ77]
Indirection: (trick from [Willard - IPL 1983])
- take $O(lg w)$ query, $O(w)$ update DS
  such as "simple" tree above
- reduce update to $O(lg w)$:
  split $n$ elements into chunks of size $O(w)$

\[
\begin{array}{ccc}
\Theta(n/w) & \Rightarrow & \Theta(lg w) \text{ query} \\
\Theta(w) & \Rightarrow & \Theta(lg w) \text{ update} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{one representative} & \Rightarrow & \Theta(w) \text{ via BSTs}
\end{array}
\]

- query: query top $\rightarrow O(lg w)$
  query bottom $\rightarrow O(lg w)$
- update: update bottom $\rightarrow O(lg w)$
  split & possibly merge with neighbor
to keep chunks $O(w)$ size
  $\Rightarrow$ update top $\rightarrow O(w)$, charged
to $O(w)$ updates in chunk

$\Rightarrow O(lg w)$ query & amortized update

- top structure can actually use $u' = \frac{u}{\Theta(w)}$:
bottoms can guarantee separation $\Omega(w)$
between representatives
  $\Rightarrow \Theta(w)$ space $\sim$ on pointer machine!
- similar trick, splitting $u$ directly instead of $n$,
  applied to stratified trees in [VEB-IPL1977]
Saving space:
- don't store empty clusters in vEB
  ⇒ V.clusters = hash table
  - Θ(1) w.h.p. e.g. via dynamic perfect hashing
  - space = O(# nonempty "child" clusters + 1)
  - charge each table entry to min in child
  - Insert cuts up element into O(lgw) min fields
  ⇒ O(n lg w) space
  - tight in worst case (⇒ not O(n)!
  - O(n) space via indirection as above

x-fast trees: [Willard - IPL 1983]
- don't store Øs in simple tree view
- store hash table of root-to-1 paths per length viewed in binary; 0 = left, 1 = right
- i.e. prefixes of elements in S
- O(lgw) query via binary search as before
- Θ(w) update as before
- Θ(nw) space

y-fast trees: [Willard - IPL 1983]
= x-fast trees
+ indirection as above
- O(lgw) query still
- O(lgw) amortized w.h.p. update
- O(n) space