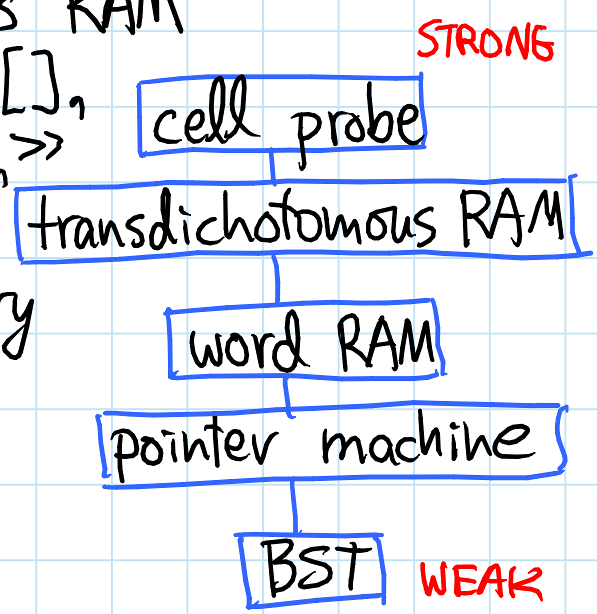
6.851	Lecture 11	Mar. 22, 2012

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 $\in \mathcal{U} = \{0, 1, \dots, u-1\}$ ($u = 2^w$)
  - ↑ 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 $w \geq \lg n$ — bridging two worlds: machine/problem
- word RAM: transdichotomous RAM with C-style operations: [], +, -, *, /, %, <, >, &, |, ^, ~, <<, >>
  - standard model
- cell probe: count # memory reads & writes
  - computation is free
  - unrealistic
  - useful for lower bounds



Predecessor problem: maintain set S of n words subject to
- insert($x \in \mathcal{U}$)
- delete($x \in S$)
- predecessor($x \in \mathcal{U}$): $\max\{y \in S \mid y < x\}$
- successor($x \in \mathcal{U}$): $\min\{y \in S \mid y > x\}$

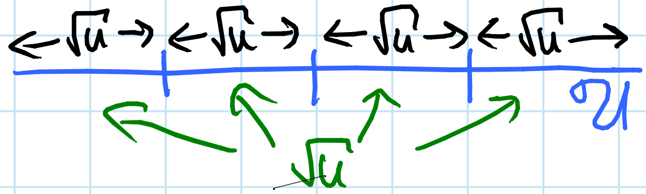
- harder than dictionaries/hashing
- comparison model $\Rightarrow$ BST: $\Theta(\lg n)$/op. optimal

- word RAM:
  - TODAY
    - van Emde Boas: $O(\lg w)$/op. $= O(\lg \lg u)$ $\Theta(u)$ space
      [FOCS 1975; IPL 1977] $O(\lg w)$ w.h.p. $\Theta(n)$ space
    - y-fast trees: $O(\lg w)$ w.h.p. $\Theta(n)$ space
      [Willard – IPL 1983]
  - L12
    - fusion trees: $O(\log_w n)$ w.h.p. $\Theta(n)$ space
      [Fredman & Willard – JCSS 1993; Raman – ESA 1996]
  - L13
    - min: $\le O(\sqrt{\lg n})$ w.h.p. $\Theta(n)$ space
  - cell probe lower bound: $O(n \operatorname{polylg} n)$ space $\Rightarrow \Omega\left(\min\left\{\log_w n, \frac{\lg w}{\lg \frac{\lg w}{\lg \lg n}}\right\}\right)$
    [Pătrașcu & Thorup – STOC 2006 & SODA 2007]
    $\Rightarrow$ vEB optimal for $w = O(\operatorname{polylg} n)$
    & fusion trees optimal for $w = 2^{\Omega(\sqrt{\lg n} \lg \lg n)}$

- pointer machine, word specified by node pointer:
  - van Emde Boas: $O(\lg \lg u)$/op. $\Theta(u)$ space
  - lower bound: $\Omega(\lg \lg u)$/op. $\Omega(u)$ space
    [Mehlhorn, Näher, Alt – SICOMP 1988]

van Emde Boas: (Peter) (reinterpreted by Bender & Farach-Colton)

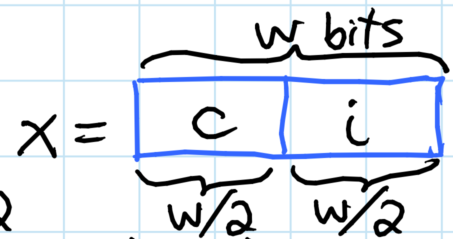
- *idea*: $T(u) = T(\sqrt{u}) + O(1)$
  $= O(\lg \lg u)$
- split universe $\mathcal{U}$ into $\sqrt{u}$ clusters, each size $\sqrt{u}$



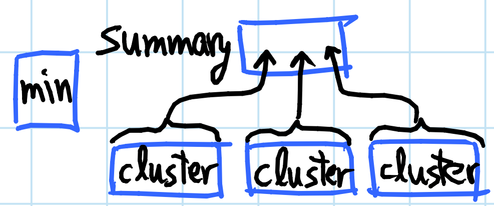
- hierarchical coordinates: word $x = \langle c, i \rangle$
  - $c = x \,//\, \sqrt{u}$ = cluster containing x
  - $i = x \,\%\, \sqrt{u}$ = x's index within cluster

  integer division — mod
  - $x = c\sqrt{u} + i \Rightarrow O(1)$-time conversion

- binary perspective:
  - split bits in half



  - c = high order = $x \gg w/2$
  - i = low order = $x \,\&\, ((1 \ll w/2) - 1)$
  - $x = (c \ll w/2) \mid i$



- recursive vEB V of size u:
  - $V.\text{cluster}[i]$ = vEB of size $\sqrt{u}$ for $0 \le i < \sqrt{u}$
  - $V.\text{summary}$ = vEB of size $\sqrt{u}$ & $w' = w/2$
    - stores which clusters c are nonempty
  - $V.\text{min}$ = minimum element in V, <u>not</u> stored recursively
    <u>OR</u> None if V is empty
  - $V.\text{max}$ = (copy of) max. element in V

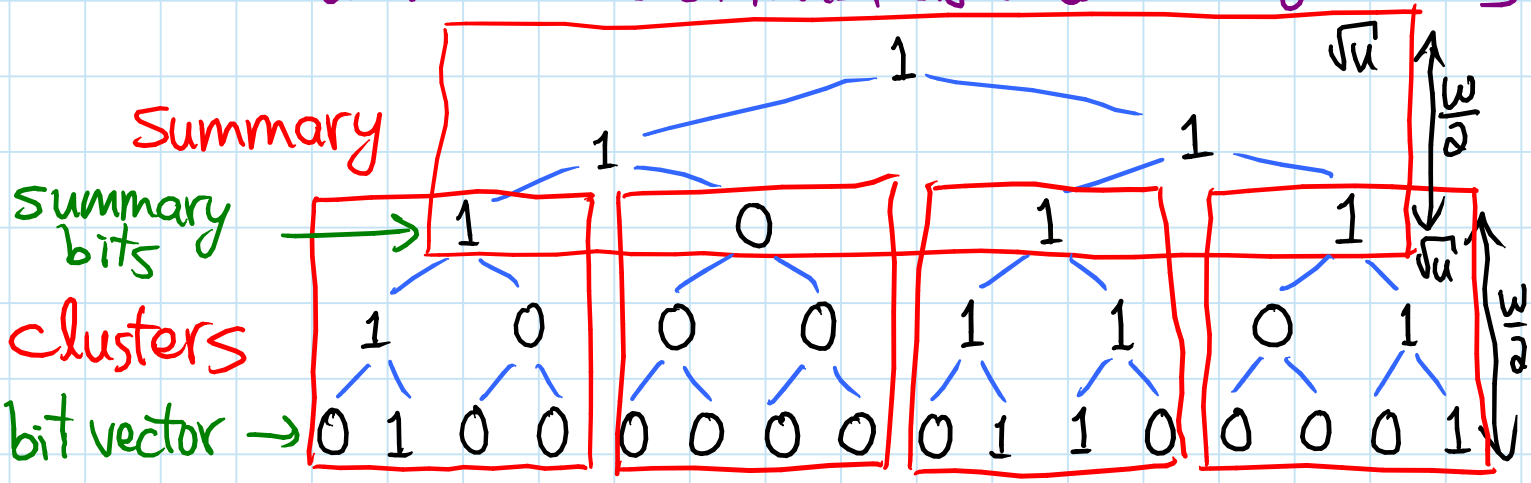
```
Successor(V, x=<c, i>):
 - if x < V.min: return V.min          (special: not stored recursively)
 - if i < V.cluster[c].max:
        return <c, Successor(V.cluster[c], i)>
 - else: c' = Successor(V.summary, c)
        return <c', V.cluster[c'].min>
```

```
Insert(V, x=<c, i>):
 - if V.min = None: V.min = V.max = x ; return      O(1)
 - if x < V.min: swap x <-> V.min
 - if x > V.max: V.max = x
 - if V.cluster[c].min = None:
      Insert(V.summary, c) => next call is O(1)
 - Insert(V.cluster[c], i)                because min not stored recursively
```

```
Delete(V, x=<c, i>):
 - if x = V.min:
    - c = V.summary.min
    - if c = None: V.min = None ; return       O(1) (now empty)
    - x = V.min = <c, i=V.cluster[c].min>      (unstore new min)
 - Delete(V.cluster[c], i)
 - if V.cluster[c].min = None:   (empty now)
      Delete(V.summary, c) => previous call O(1)
 - if V.summary.min = None: V.max = V.min
 - else: c' = V.summary.max
      V.max = <c', V.cluster[c'].max>
```

Tree view: expand recursion [vEB-FOCS 1975]

[van Emde Boas, Kaas, Zijlstra – Math. Sys. Th. 1977]



- node = OR of children

$\Rightarrow$ path from leaf x to root is monotone

$\Rightarrow$ could binary search for $0 \to 1$ transition

- max/min of last 0's left/right sibling is predecessor/successor of x (if $\notin S$)
- store sorted linked list on elements to find successor/predecessor

$\Rightarrow$ query in $O(\lg \lg u)$ ~ roughly same as above

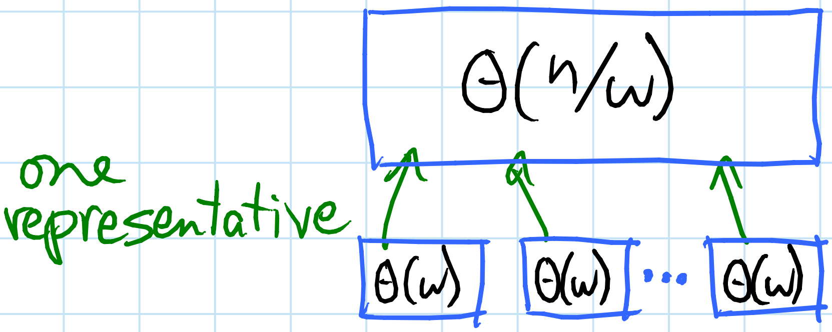
- even in pointer machine & $O(u \lg w)$ space: node stores linked list of pointers to ancestor of height 2^i for $i = 0, 1, \ldots, \lg w$

- but updating these bits costs $\Theta(\lg u)$/op.
- vEB's not-storing-min reduces to $\Theta(\lg w)$
- again possible on pointer machine with $O(u \lg w)$ space [vEBKZ77]

"stratified tree"

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 $\Theta(w)$



} $O(\lg w)$ query, $O(w)$ update

} $O(\lg w)$ via BSTs

- 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 $\Theta(w)$ size
  $\Rightarrow$ update top $\rightarrow O(w)$, charged to $\Theta(w)$ updates in chunk

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

- top structure can actually use $u' = u/\Theta(w)$: bottoms can guarantee separation $\Omega(w)$ between representatives
  $\Rightarrow \Theta(u)$ space ~ on pointer machine!
  - similar trick, splitting u directly instead of n, applied to stratified trees in [vEB – IPL 1977]

Saving space: [Vladimír Čunát – S.M. thesis 2011?]

- don't store empty clusters in vEB

$\Rightarrow$ V.clusters = hash table

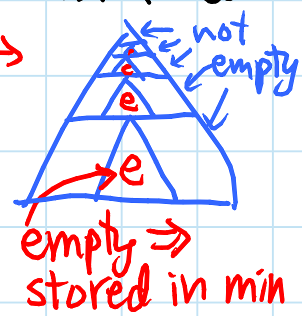
- $\Theta(1)$ w.h.p. e.g. via dynamic perfect hashing
- space = $O(\#$ nonempty "child" clusters $+ 1)$

- charge each table entry to min in child
- Insert cuts up element into $O(\lg w)$ min fields

$\Rightarrow O(n \lg w)$ space

- tight in worst case ($\Rightarrow$ not $O(n)$!)

- $O(n)$ space via indirection as above



x-fast trees: [Willard – IPL 1983]

- don't store $\emptyset$s in simple tree view
- store hash table of root-to-1 paths

or one per length (as in Willard)

viewed in binary: $\emptyset$ = left, 1 = right

i.e. prefixes of elements in S

- $O(\lg w)$ query via binary search as before
- $\Theta(w)$ update as before
- $\Theta(n\,w)$ space

y-fast trees: [Willard – IPL 1983]

= x-fast trees

+ indirection as above

- $O(\lg w)$ query still
- $O(\lg w)$ amortized w.h.p. update
- $O(n)$ space