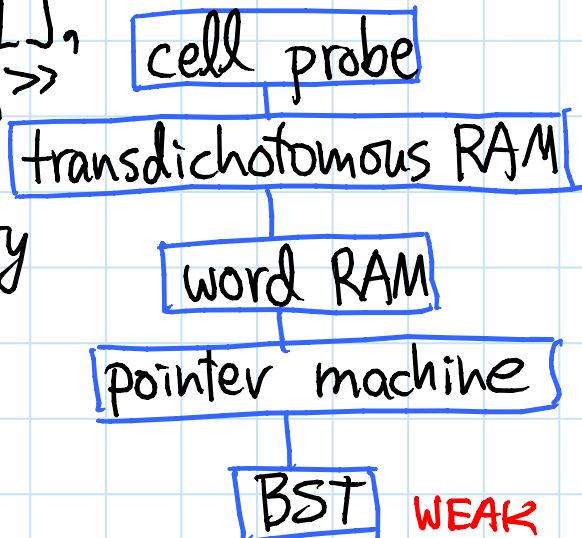


TODAY: Integers & van Emde Boas

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

Models for integer data structures:

- word =  $w$ -bit integer  $\in \mathcal{U} = \{0, 1, \dots, u-1\}$   <sup>$u = 2^w$</sup> 
  - ↑ 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:  $[], +, -, *, /, \% , <, >, \&, |, \wedge, \sim, \ll, \gg$ 
  - 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.  $\Theta(u)$  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  $\searrow$  - min:  $\leq O(\sqrt{\lg n})$  w.h.p.  $\Theta(n)$  space  
- cell probe lower bound:  $O(n \text{ poly} \lg n)$  space  $\Rightarrow \Omega(\min\{\log_w n, \frac{\lg w}{\lg \frac{\lg w}{\lg \lg n}}\})$   
[Patrascu & Thorup - STOC 2006 & SODA 2007]

$\Rightarrow$  vEB optimal for  $w = O(\text{poly} \lg n)$   
& fusion trees optimal for  $w = 2^{\Omega(\sqrt{\lg n} \lg \lg n)}$

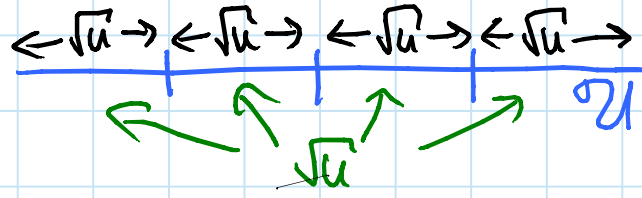
- pointer machine, word specified by <sup>node</sup> pointer:  
- van Emde Boas:  $O(\lg \lg u)$  / op.  $\Theta(u)$  space  
- lower bound:  $\Omega(\lg \lg u)$  / op.  $\Omega(u)$  space

[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  $U$  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  $\uparrow$  mod  $\uparrow$

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

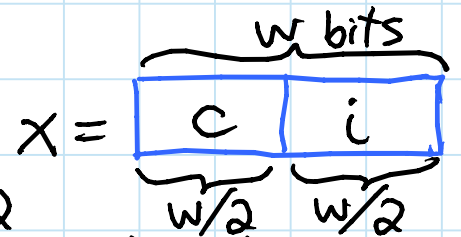
- binary perspective:

- split bits in half

-  $c = \text{high order} = x \gg w/2$

-  $i = \text{low order} = x \& ((1 \ll w/2) - 1)$

-  $x = (c \ll w/2) | i$



- recursive vEB  $V$  of size  $u$ :

-  $V.\text{cluster}[i] = \text{vEB of size } \sqrt{u}$  for  $0 \leq i < \sqrt{u}$

-  $V.\text{summary} = \text{vEB of size } \sqrt{u}$  &  $w' = w/2$

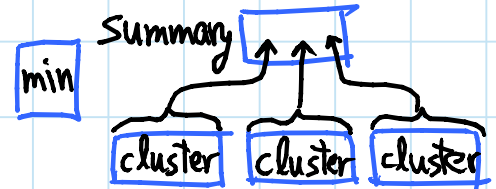
- stores which clusters  $c$  are nonempty

-  $V.\text{min} = \text{minimum element in } V$ ,

not stored recursively

OR None if  $V$  is empty

-  $V.\text{max} = (\text{copy of}) \text{max. element in } V$



Successor( $V, x = \langle c, i \rangle$ ):

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

Insert( $V, x = \langle c, i \rangle$ ):

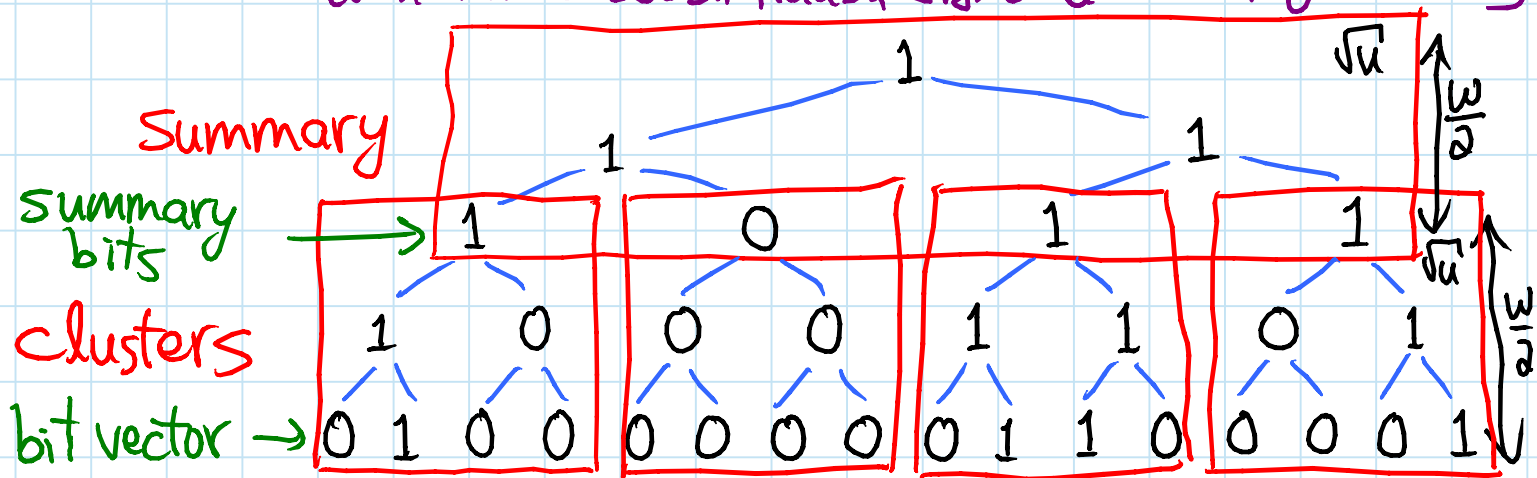
- if  $V.min = None$ :  $V.min = V.max = x$ ; return  $O(1)$
- if  $x < V.min$ : swap  $x \leftrightarrow V.min$
- if  $x > V.max$ :  $V.max = x$
- if  $V.cluster[c].min = None$ :  
Insert( $V.summary, c$ )  $\Rightarrow$  next call is  $O(1)$
- Insert( $V.cluster[c], i$ )

because min not stored recursively

Delete( $V, x = \langle c, i \rangle$ ):

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

Tree view: expand recursion [VEB-FOCS 1975]  
 [van Emde Boas, Kaas, Zijlstra - Math. Sys. Th. 1977]

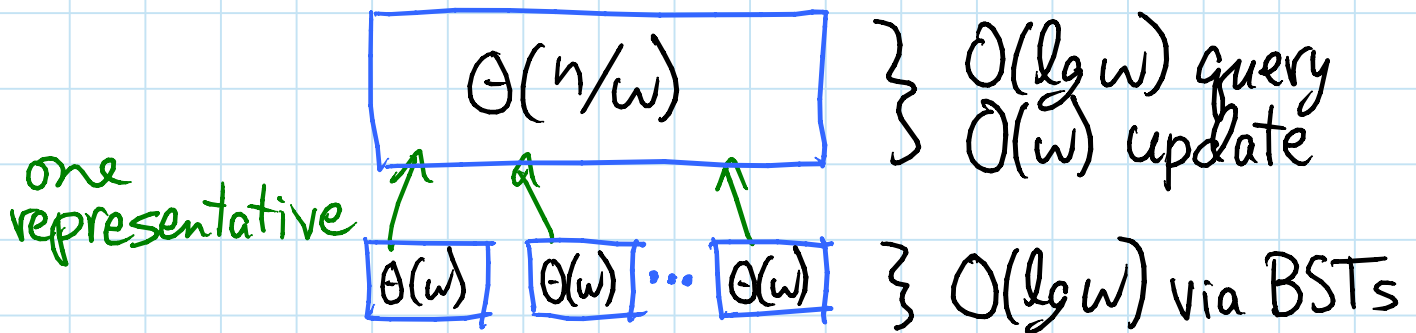


- node = OR of children
- ⇒ path from leaf  $x$  to root is monotone
- ⇒ could binary search for  $0 \rightarrow 1$  transition
- max/min of last 0's left/right sibling is predecessor/successor of  $x$  (if  $x \notin S$ )
- store sorted linked list on elements to find successor/predecessor
- ⇒ query 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  $i = 0, 1, \dots, \lg w$
- 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 [VEBK77]

"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  $\Theta(w)$



- 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  
 $\Rightarrow \Theta(w)$  space  $\sim$  on pointer machine!
- similar trick, splitting  $u$  directly instead of  $n$ , applied to stratified trees in [VEB - IPL 1977]

## Saving space: [Bender, Demaine, Farach-Colton?]

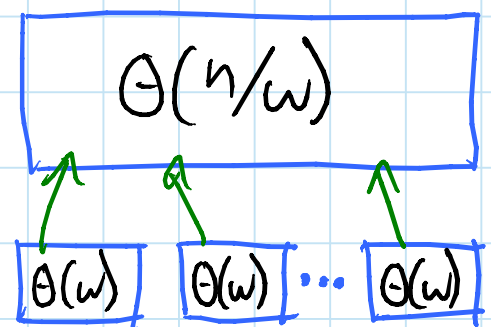
- start from  $\sqrt{EB}$
- don't store empty clusters
- $\Rightarrow$  V. clusters = hash table
  - $\Theta(1)$  w.h.p. e.g. via dynamic perfect hashing
  - space =  $O(\# \text{ nonempty "child" clusters} + 1)$
- charge each table entry to min in child
- $\Rightarrow$  each element charged once
- $\Rightarrow O(n)$  space

## 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 viewed in binary:  $\emptyset = \text{left}, 1 = \text{right}$   $\hookrightarrow$   
(as in Willard) i.e. prefixes of elements in  $S$
- $O(\lg w)$  query via binary search as before
- $\Theta(w)$  update as before
- $\Theta(nw)$  space

## y-fast trees: [Willard - IPL 1983]

- x-fast trees + indirection as above
- $O(\lg w)$  query still
- $O(\lg w)$  amortized update
- $O(n)$  space



MIT OpenCourseWare  
<http://ocw.mit.edu>

6.851 Advanced Data Structures  
Spring 2012

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.