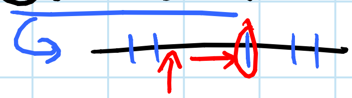
6.046	Lecture 4	Feb. 19, 2015

TODAY: van Emde Boas [Peter, 1974]

- series of improved data structures [based on personal communication with Michael Bender, 2001]
- Insert, Successor
- Delete
- space

Goal: maintain n elements among $\{0, 1, \ldots, u-1\}$ subject to Insert, Delete, Successor in $O(\lg \lg u)$ time/op.



- if $u = n^{O(1)}$ or $n^{\lg^{O(1)} n}$ then $O(\lg \lg n)$ time/op.!
  - exponentially faster than balanced search trees
  - cooler queries than hashing
- application: network routing tables ($u = 2^{32}$ in IPv4)
  $= \{$range of IP addresses $\rightarrow$ port to send$\}$

Where might $O(\lg \lg u)$ bound arise?

- binary search over $\lg u$ elements
- recurrences: $T(\lg u) = T\left(\frac{\lg u}{2}\right) + O(1)$
  $$\boxed{T(u) = T(\sqrt{u}) + O(1)}$$

We'll develop van Emde Boas data structure by a series of improvements on a very simple data structure:

① Bit vector: V[x] = is x in the set?

e.g.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1

$u = 16$
$n = 4$

- Insert/Delete: $O(1)$:)
- Successor/Predecessor: $O(u)$:(

② Split universe into clusters: $\sqrt{u}$ of size $\sqrt{u}$

e.g.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1

V.cluster[0] V.cluster[1] ... V.cluster[$\sqrt{u}-1$]

nonempty: 1 0 1 1

- if $x = i\sqrt{u} + j$ then $V[x] = V.\text{cluster}[i][j]$
  $0 \le j < \sqrt{u}$

$\Rightarrow$ define $\begin{cases} \text{low}(x) = x \bmod \sqrt{u} & = j \\ \text{high}(x) = \lfloor x/\sqrt{u} \rfloor & = i \end{cases}$

$\text{index}(i,j) = i\sqrt{u} + j$

= high & low-order halves in binary

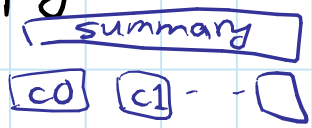
x: $\underbrace{1\,0}_{\text{high}(x) = 2}\,\underbrace{0\,1}_{\text{low}(x) = 1}$ = 9

- Insert: set V.cluster[high(x)][low(x)] } $O(1)$
  mark cluster high(x) nonempty } $O(1)$
- Successor:
  - look within cluster high(x) } $O(\sqrt{u})$
  - else find next nonempty cluster i } $O(\sqrt{u})$ *
  - find min j in that cluster } $O(\sqrt{u})$
  - return index(i, j)

better! $O(\sqrt{u})$

③ **Recurse:** 3 ops. in Successor are recursive Successors!

- V.cluster[i] = size-$\sqrt{u}$ van Emde Boas, $0 \le i < \sqrt{u}$
- V.summary = size-$\sqrt{u}$ van Emde Boas
- V.summary[i] = is V.cluster[i] nonempty?



Insert(V, x):

Insert(V.cluster[high(x)], low(x)) $\quad T(\sqrt{u})$

Insert(V.summary, high(x)) $\quad T(\sqrt{u})$

$$\Rightarrow T(u) = 2T(\sqrt{u}) + O(1)$$
$$T'(\lg u) = 2T'\left(\frac{\lg u}{2}\right) + O(1)$$
$$= O(\lg u)$$

Successor(V, x):

```
i = high(x)
j = Successor(V.cluster[i], low(x))        T(√u)
if j = ∞:
    i = Successor(V.summary, i)            T(√u)
    j = Successor(V.cluster[i], -∞)        T(√u)
return index(i, j)
```

$$\Rightarrow T(u) = 3T(\sqrt{u}) + O(1)$$
$$T'(\lg u) = 3T'\left(\frac{\lg u}{2}\right) + O(1)$$
$$= O((\lg u)^{\lg 3})$$
$$= O(\lg^{1.585} u)$$

!

- need to reduce to one recursion!

④ Maintain min & max of every structure:

- $O(1)$ overhead in Insert: if $x < V.min$: $V.min = x$
  if $x > V.max$: $V.max = x$

Successor(V, x):

```
  i = high(x)
  if low(x) < V.cluster[i].max:
       j = Successor(V.cluster[i], low(x))
  else: i = Successor(V.summary, high(x))
       j = V.cluster[i].min
  return index(i, j)
```

$\Rightarrow T(u) = T(\sqrt{u}) + O(1)$
$= O(\lg \lg u)$ ☺

⑤ Don't store min recursively:

- Successor checks for min specially:
  if $x < V.min$: return V.min

Insert(V, x):

empty case

costs $O(1)$

```
  if V.min = None:  V.min = V.max = x: return
  if x < V.min:  swap x ↔ V.min
  if x > V.max:  V.max = x
  if V.cluster[high(x)].min = None:    (previously empty)
       Insert(V.summary, high(x)) *
  Insert(V.cluster[high(x)], low(x))
```

* if both calls, then second costs $O(1)$ (empty case)

$\Rightarrow T(u) = O(\lg \lg u)$ ☺

⑥ Delete(V, x):
    if x = V.min: (find new min)
        i = V.summary.min
        if i = None: V.min = V.max = None   } empty now
                       return   costs O(1)
        x = V.min = index(i, V.cluster[i].min)   } unstore new min
    Delete(V.cluster[high(x)], low(x))
    if V.cluster[high(x)].min = None: } empty now
        Delete(V.summary, high(x))   * second call
update V.max {
    if x = V.max:
        if V.summary.max = None: } just min now
            V.max = V.min
        else: i = V.summary.max
            V.max = index(i, V.cluster[i].max)

* if make second call, then first call was cheap (just deleted a min)

$\Rightarrow T(u) = O(\lg \lg u)$

Lower bound: [Pătrașcu & Thorup 2007]

$\Omega(\lg \lg u)$ for $u = n^{\lg^{O(1)} n}$

& space $= O(n \operatorname{polylg} n)$

– even static (just Successor, no Insert/Delete)

7 <u>Space</u>: improve from current $\Theta(u)$ to $O(n \lg\lg u)$

- only create nonempty clusters
  - if V.min becomes None, deallocate V
- V.cluster = hash table of nonempty clusters (recall from 6.006; and see Lecture 8)
- insert may create new structure (fill min) $\Theta(\lg\lg u)$ times (each empty insert)
  - can really happen [Vladimír Čunát]
- charge pointer to structure (and associated hash-table cell) to the structure

$\Rightarrow O(n \lg\lg u)$ space (but randomized)

CHARGING AMORTIZATION ~
SEE NEXT LECTURE (5)

8 <u>Indirection</u> further reduces to $O(n)$ space

- store vEB structure with $n = O(\lg\lg u)$ using BST or even array
  - $\Rightarrow O(\lg\lg n)$ time once in base case
- $O(n/\lg\lg u)$ such structures (disjoint)
  - $\Rightarrow O\left(\frac{n}{\lg\lg u} \cdot \lg\lg u\right) = O(n)$ space for small
- larger structures "store" pointers to them
  - $\Rightarrow O\left(\frac{n}{\lg\lg u} \cdot \lg\lg u\right) = O(n)$ space for large
- details: split/merge small structures