6.046 (2015) Van Emde Boas Trees

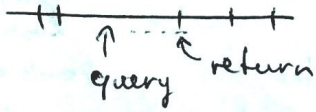
lecture 4

- series of improved DSs
- Insert, Delete, Successor
- space

size of universe U $= u$

Goal: maintain n elements among $\{0, 1, \ldots, u-1\}$ subject to Insert, Delete, Successor

Successor: in $O(\lg \lg u)$ time



, predecessor is symmetric

$\lg \lg 2^{64} = \lg 64 = 6$

my comment

$O(n \lg n)$ ← size of universe in radix sort

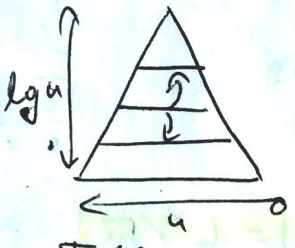
if $u = n^{O(1)}$ or $n^{\lg^{O(1)} n}$, then $\lg \lg u = O(\lg \lg n)$

- application: network router: ip range → send to port y

mark beginnings of IP ranges and use predecessor

Where might $O(\lg \lg u)$ come from?

→ binary search on levels of tree



$T(k) = T(\frac{k}{2}) + O(1)$

$= O(\lg k)$

$T(\lg u) = T(\frac{\lg u}{2}) + O(1)$

$= O(\lg \lg u)$

$T'(u) = T'(\sqrt{u}) + O(1)$

$= O(\lg \lg u)$

intuition

take a problem of size u

split into problems of size $\sqrt{u}$

and recurse on one of them

① Bit vector = array of size u

0 = absent 1 = present

$u = 16$

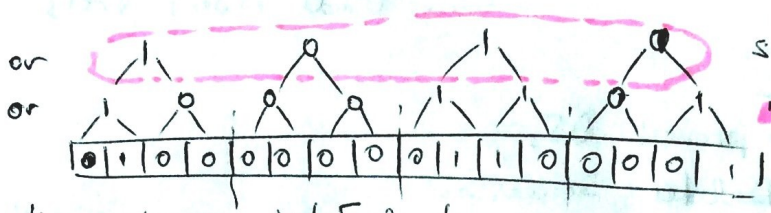
$n = 4$

array

0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1

Insert / Delete: $O(1)$

Successor: $O(u)$



my comment: recursion see vEB [redacted] tree. Summary has all info about "cluster bit vector" similar

② split the universe into $\sqrt{u}$ clusters of size $\sqrt{u}$.

my comment [redacted] idea in ideal non-rand skip lists balancing

Insert: $O(1)$
check/change bits on 2 levels

skip list
$\lg(n)$ levels, each $\lg_{\sqrt{n}}$
$2 \leftarrow 4 \leftarrow 8 \leftarrow 16$

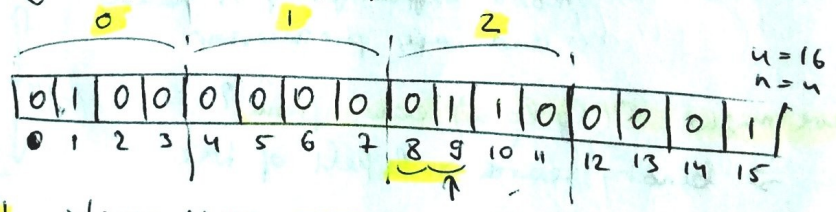
vEB
repeated square rooting
$2 \leftarrow 4 \leftarrow 16$

Successor (x): $O(\sqrt{u})$
- look in x's cluster
- look for next 1 in the summary vector
- look for first 1 in that cluster

Terminology:

if $x = i\sqrt{u} + j$ $\quad i$: cluster number

$0 \le j < \sqrt{u}$ $\quad j$: position within that cluster



e.g. $x = 9$

$9 = 2 \cdot \sqrt{16} + 1$

go two clusters in the summary vector, then multiply by the size of each cluster and continue with 0, 1.

$high(x) = \lfloor x/\sqrt{u} \rfloor$ integer division → i

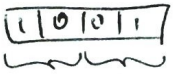
$low(x) = x \bmod \sqrt{u}$ remainder → j

$index(i,j) = i\sqrt{u} + j$ use i and j to go back to x

← x representation

Why high/low? $O(\lg(u))$ long

$x = 9$



$high(x) = 2$ → high half of the bits

$low(x) = 1$ → low half of the bits

divide the bit vector of 9 in 2 halves and get high half and low half.

very efficient

6.046 (2015)
Lecture 4

③ recurse : V = size u

- $V.cluster[i]$ = size $\sqrt{u}$ $\quad 0 \le i \le \sqrt{u}-1$
- $V.summary$ = size $\sqrt{u}$

vEB

vEB

vEB

Insert (V, x)

low(x) becomes x in the next Insert call.

```
Insert(V.cluster[high(x)], low(x))
Insert(V.summary, high(x))
```

my comment: (sub)clusters which are not empty, recursively for each cluster that is recursed on

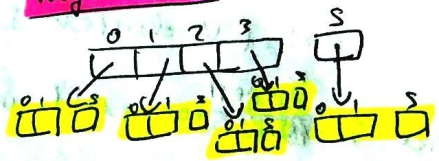
high(x), low(x)

$T(u) = 2T(\sqrt{u}) + O(1)$

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

$= O(\lg u)$

my comment



Successor (V, x)

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

$O((\lg u)^{\lg 3})$

④ DS augmentation

store min and max → every vEB structure knows its min and max.

clusters & summaries

Insert (V, x)

```
if x < V.min:
    V.min = x
if x > V.max
    V.max = x
Insert ...
Insert ...
```

shell $O(\lg u)$ so far

Successor (V, x)

if $x < V.min$ return V.min

```
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)
```

only 1 recursion on $\sqrt{u}$

$O(\lg \lg u)$

(5) don't store min recursively (equivalent to lazy propagation, never move down)

if vEB structure is empty, set V.min = V.max = x and stop.

Insert (V, x)

```
if V.min = None:          // inserting into an empty structure
    V.min = V.max = x
    return
if x < V.min: swap x ⟷ V.min   // every item except min is recursively inserted
if x > V.max: V.max = x
if V.cluster[high(x)].min = None:   // if cluster is empty need to [redacted] update summary
    Insert(V.summary, high(x))
Insert(V.cluster[high(x)], low(x))
```

still 2 [redacted] Insert calls in worst case

But if Insert(V.summary, high(x)) is called V.cluster[high(x)] was empty

Thus [redacted] Insert(V.cluster[high(x)], low(x)) will only update the min and max of V.cluster[high(x)]

⇒ in each case only 1 recursive call!

⇒ $O(\lg \lg u)$

6.046 (2015)
Lecture 4

(3)

```
Delete (V, x)
  if x = V.min:
    i = V.summary.min
    if i = None:
      V.min = V.max = None
      return // min deleted
    x = V.min = index(i, V.cluster[i].min)
    // deleting min, but it is not the only elt; actual deletion in following code
  ① Delete(V.cluster[high(x)], low(x)) // undo Insert
  ② if V.cluster[high(x)].min = None:
      Delete(V.summary, high(x))
    if x = V.max: // at this point max was just deleted
      if V.summary.max = None:
        V.max = V.min
      else:
        i = V.summary.max
        V.max = index(i, V.cluster[i].max)
```

my comment: V.min is not in V.summary!

triggers ② in the preceding Delete call

my comment: of new x in sub trees!

if ② leads to a recursive call of Delete, it must be true that ① the last item in V.cluster[high(x)] was deleted, which is V.cluster[high(x)].min, in constant time

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

Lower bound

$\Omega(\lg\lg u)$ for $u = n^{\lg O(1) n}$ & space $O(n \cdot \text{poly}\lg n)$

(6) Space is $O(u)$

- Only store non-empty clusters

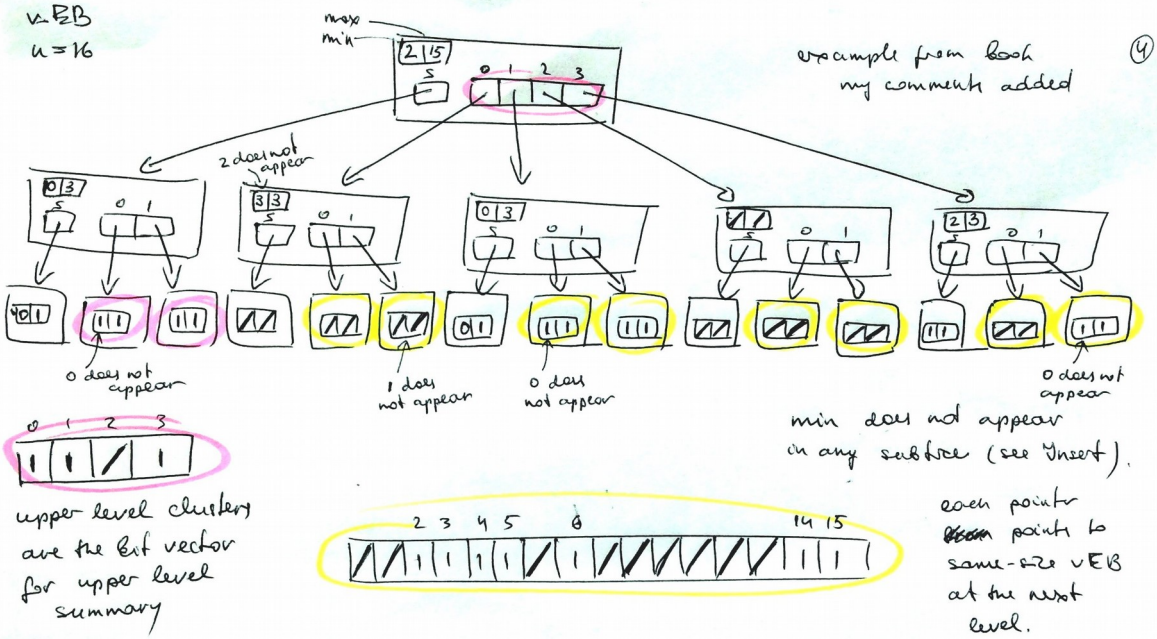
$\Rightarrow$ V.cluster = hash table, not array

[redacted] runtime bound to expectation bound!

but space goes down

- $O(n \lg\lg u)$ space, fix $\Rightarrow O(n)$

vEB
$u = 16$



example from book
my comments added

min does not appear in any subtree (see Insert).

upper level clusters are the bit vector for upper level summary

each pointer points to same-size vEB at the next level.

4 clusters of 4
each has 2 clusters of 2

Bit vector: 16 → 4 of 4 → each 2 of 2

256 → 16 of 16 → each 4 of 4 → each 2 of 2