CS 224

lecture 1

static – no insertions / deletions
dynamic – insertions / deletions

Static Predecessor

- data structure represents set S of items $\{x_1, \dots x_n\}$
- Query $pred(z) = \max\{x \in S : x < z\}$
- want low space, fast query

Example soln ($O(n)$ space):

sorting:
- get max
- insert all elts
- start with max, get predecessors

static : store numbers sorted (in array), do binary search ← $\Theta(\lg n)$
dynamic : $O(\lg n)$ query using balanced BST (rb-tree, AVL), $O(\lg n)$ for updates.

Comparison model for sorting $\Theta(n \lg n)$
but this is not how computers work → e.g. 32–64 bit words

bitwise operations (XOR)
bit shifting etc...
multiplication

$\underbrace{\qquad}_{\text{not just comparisons}}$ → Word RAM model

Word RAM model

- items are integers in $\{0, 1, \dots 2^{w}-1\}$
- w = "word size"
  $u = 2^w - 1$ universe
- assume that pointers fit in a word
- space $\geq n$ (n items)
  $w \geq \lg(\text{space}) \geq \lg n$ ← distinction between: w close to $\lg n$ or much larger than $\lg n$

Two data structures

① Van Emde Boas tree (FOCS '75)

update/query $\Theta(\lg w)$ time

$\Theta(u)$ space, 64-bit machine → $\Theta(2^{64})$

can be made $\Theta(n)$ with randomization ← hash tables (my comment)

[redacted] y fast tries, same bounds (Willard, IPL '83)

② Fusion trees (Fredman, Willard JCSS '93)

Query in time $\Theta(\log_w n)$ (can be made dynamic) $\log_w n = \frac{\lg n}{\lg w} \leq \frac{\lg n}{\lg \lg n}$ ($w \geq \lg n$)

and linear space

⟹ knowing w, achieve $\min\{\underset{\text{VEB/y fast}}{\lg w}, \underset{\text{Fusion}}{\lg_w n}\} \leq \sqrt{\lg(n)}$

[redacted] with dynamic Fusion trees ⟹ sort in $O(n\sqrt{\lg n})$

assume $\lg w = \lg_w n = \frac{\lg n}{\lg w}$

$(\lg w)^2 = (\lg n)$

Faster sorting (same model, Word RAM model)

my comment

vEB [redacted] provides $O(\lg\lg u)$ updates $\Rightarrow$ $O(n \lg\lg u)$ sort.

$O(n \lg\lg n)$ deterministic (Han, STOC '02)

$O(n\sqrt{\lg\lg n})$ randomized, in expectation (Han, Thoup, FOCS '02)

Open question of linear-time sorting is possible in the word RAM model.

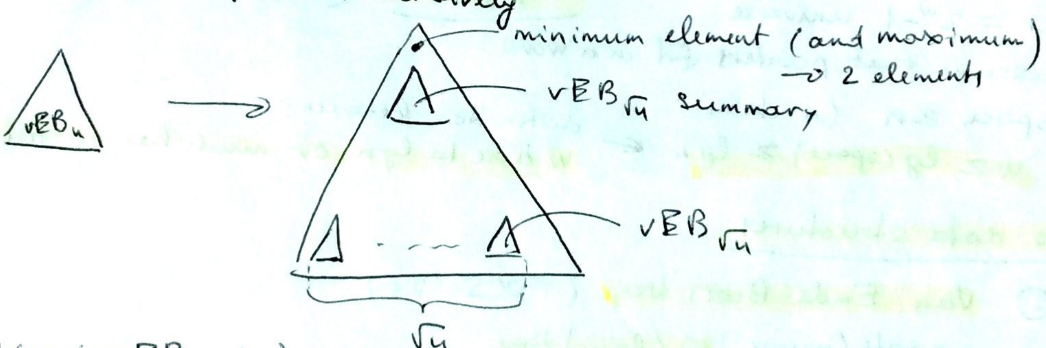
Word RAM

Word RAM: assume that given x, y fitting in a word each, we can do: $+$ $/$ $*$ $-$, $\sim$ $\wedge$ $|$ $\oplus$, $\gg$ $\ll$ in const. time

- $/$: int division, rounds down
- $*$: fits in 2 words
- $\sim$ $\wedge$ $|$ $\oplus$: bitwise
- $\gg$ $\ll$: bit shifting
- $+$ $/$ $*$ $-$: integer arithmetic

Van Emde Boas Tree (VEB tree)

- VEB tree defined recursively



Fields of vEB_u (V)

- $\sqrt{u}$ - size array V.cluster[0], ..., V.cluster[$\sqrt{u}-1$]
  - $vEB_{\sqrt{u}}$ data structure
- V.summary is a $vEB_{\sqrt{u}}$ instance
- V.min / V.max are integers in $\{0, \dots, u-1\}$

Let $x \in \{0, 1, \dots, u-1\}$

Ex: $x = \underbrace{1001}_{c}\,\underbrace{0011}_{i} = \langle c, i \rangle$, $c, i \in \{0, \dots, \sqrt{u}-1\}$

x in base $\sqrt{u}$

my comments (bug fixes)

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

```
if x > V.max
    return V.max
else if V.cluster[c].min < x :        ← must be i
    return Pred(V.cluster[c], i)       ← return c·√u + Pred(V.cluster[c], i)
                                                 (u of subtree)
else
    c = Pred(V.summary, c)
    return V.cluster[c].max            ← return c·√u + V.cluster[c].max
                                                 (u of subtree)
```

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

```
if V = ∅
    V.min ← x , return       // min not stored in subtrees   }  also need
if x < V.min                                                  }  to handle
    swap(x, V.min)           // min not stored in subtrees   }  max
if V.cluster[c].min = ∅
    Insert(V.summary, c)
Insert(V.cluster[c], i)
```

Pred time:

$T(u) = T(\sqrt{u}) + O(1)$ only one recursion

$\Rightarrow T(u) = O(\lg \lg u) = O(\lg w)$ $\quad u = 2^{\lg w}$

Insert time: if V.cluster[c].min = ∅ or true

Insert (V.cluster[c], i) = O(1)

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

Space of vEB:

$$S(u) = (\sqrt{u}+1)\, S(\sqrt{u}) + O(1) \quad \leftarrow \text{min \& max}$$

$$\Rightarrow S(u) = \Theta(u)$$

Improve space: in a vEB data structure, have a hash table

my comment: instead of an array of pointers to empty and non-empty clusters

- keys are cluster ID's c ...
- value is pointer to corresponding non-empty cluster

Claim: vEB with hash table uses $\Theta(n)$ space

Pf: Charge the cost of storing (c, pointer to cluster c) to minimum element of cluster c.
Each $x \in S$ is charged exactly once.

short aside:

Dictionary problem:

- store (key, value) pairs
- query (k) returns val associated with key k or null if k is not associated.
- insert (k, v) associates val v with key k

Dynamic dictionary is possible with
$\Theta(n)$ space
$\Theta(1)$ worst case query
$\Theta(1)$ expected insertion

(Dietzfelbinger et al.)

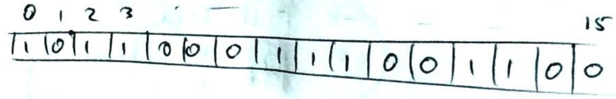
$\Theta(1)$ insertion with high probability is possible

my comment
$n = O(m) \Rightarrow$ must grow m as n grows, [redacted] $\alpha = O(1)$ then search is $\Theta(1)$ in expectation!
[redacted] here $\Theta(1)$ worst case search! [redacted]
↳ modification [redacted] beyond simple chaining and open-addressing

sketch, because same bounds

Another solution (x, y - fast tries)

bit array of length n



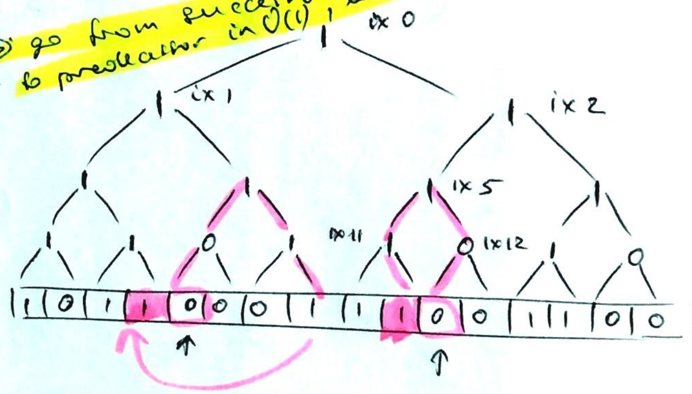
- make binary tree where each node stores OR
- store all the 1's in a doubly linked list ⟹ go from successor to predecessor in $O(1)$, both for 0's and 1's

Time: $O(\lg n)$

- On any path from leaf to root bits are monotone

0 0 0 1 1 1 1 1

to find the first 1, can do binary search



successor ⟹ predecessor in doubly LL

- store tree as an array
  root at index 0
  node v has left child at $2v+1$
  right child at $2v+2$

Binary search on following pointers ⟹ can find kth ancestor in constant time by doing $\gg k$

$O(\lg \lg n)$ Pred time

- could also, for each node, store its 2^kth ancestor for each $k = 0 \ldots \lg\lg n$

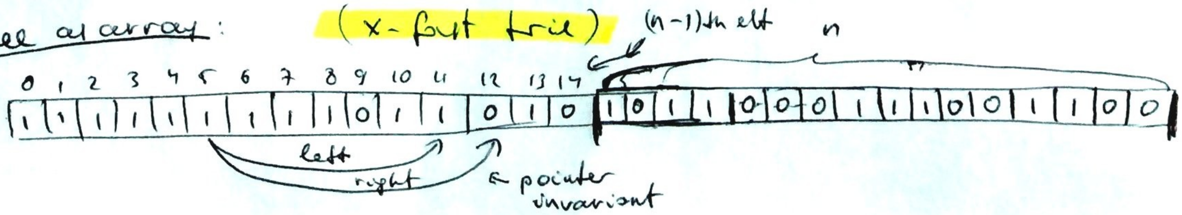
$O(n \lg\lg n)$ Space

- To save space, only store 1's in a hash table ← similar space solution to vEB

For each level of tree, hash table stores locations of 1's

⟹ space $\Theta(nw)$

tree as array: (x-fast trie)



from x-fast to y-fast

- use "indirection"