Adv. Data Structures

Van Emde Boas trees

Recap

- HW2 posted, due next Friday
- No class on Friday this week (happy HW-ing)
- Will have at least 1 more HW (after break)
- Project proposals: due April 2 (<u>no</u> exceptions) (2-3 pages - see webpage for details)

Current data structure:

What if we restrict inputs?

Goal: Have a bounded set of possible elements, & want to store which ones are in my set.

ie: subset of 32-bit integer

or list of names (all ≤ 30 chars)

Operations

- insert(x)
- find(x)
- delete(x)
- max/min
- successor(x)
- predecessor(x)

Tiered Bitvector:

Put a summary on top of the vector. u/B

OR the bits

1	0	1	0	1	1	0	0
00100010	00000000	00011000	00000000	00000100	11110111	00000000	00000000

B

How to search/update:

succ: check for next value in x's block

if none, move up & scan upper tier (until 1)

Move down & find min in low block

Runtime: $B + \frac{u}{B} + B$

$= O(B + \frac{u}{B})$

How to find "best" value for B?

What about deleting?

1	0	1	0	~~1~~ 0	1	0	0
001000~~1~~0 0	00000000	00011000	00000000	00000~~1~~00 0	11110111	00000000	00000000

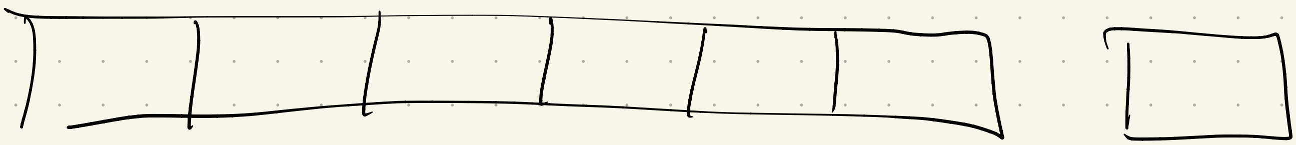
- 1 delete in bottom $O(1)$
- Is-empty
  - if empty, delete top $(0 \to 1)$

Runtime:

$O(\sqrt{u})$

So: tiering helped! ($U \to \sqrt{U}$)

Can we improve even more?



$\sqrt{U}$ blocks
each $\sqrt{U}$ size

summary,
size
$\sqrt{U}$

Recurse!

For each block of size $\sqrt{U}$, apply the same construction:

$U^{1/4}$ size blocks,
plus summary

Picture:

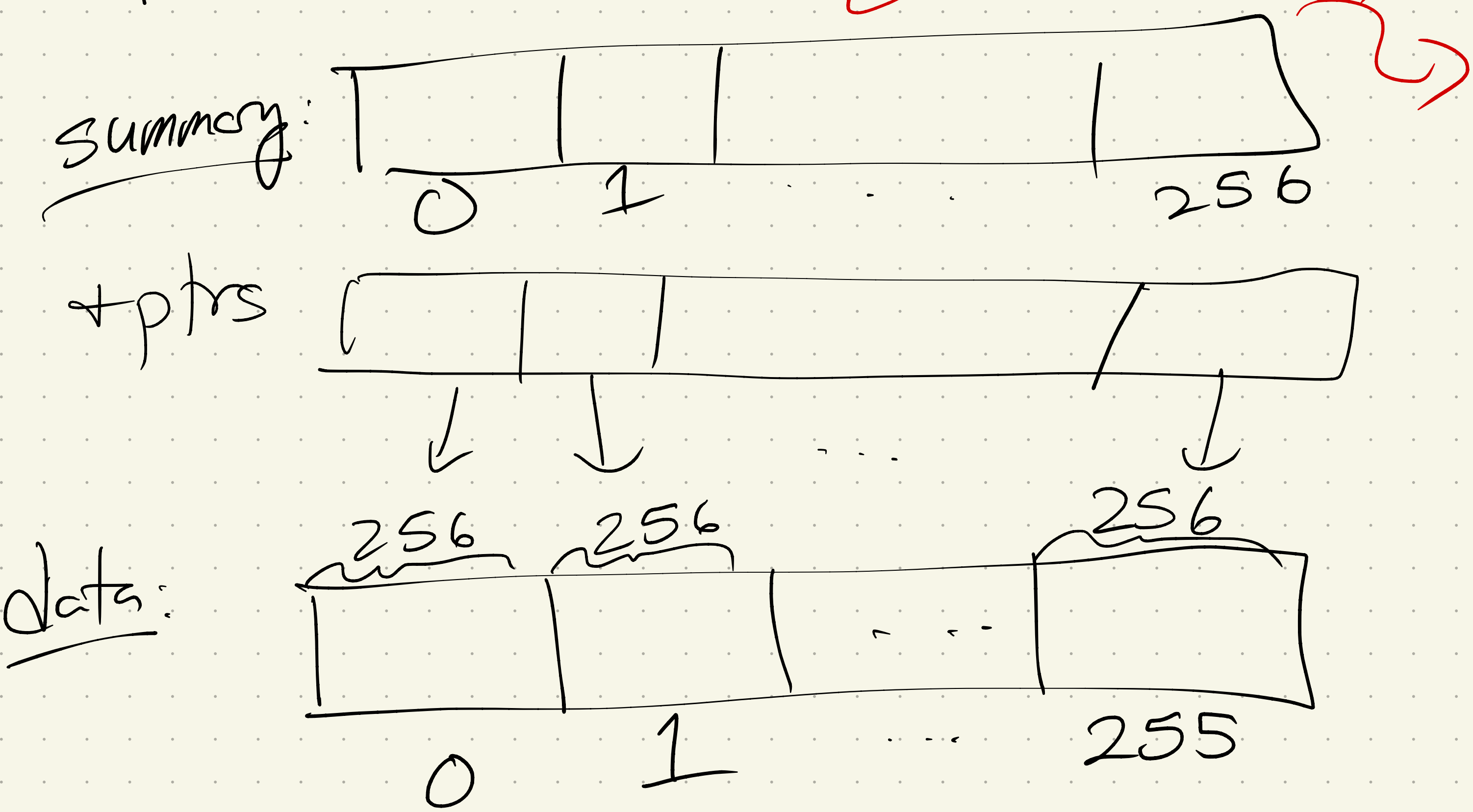
Suppose we have ASCII!

$U = 65{,}536$

$\sqrt{U} = 256 \; (\& \; U^{1/4} = 16)$

Before:

just an arry

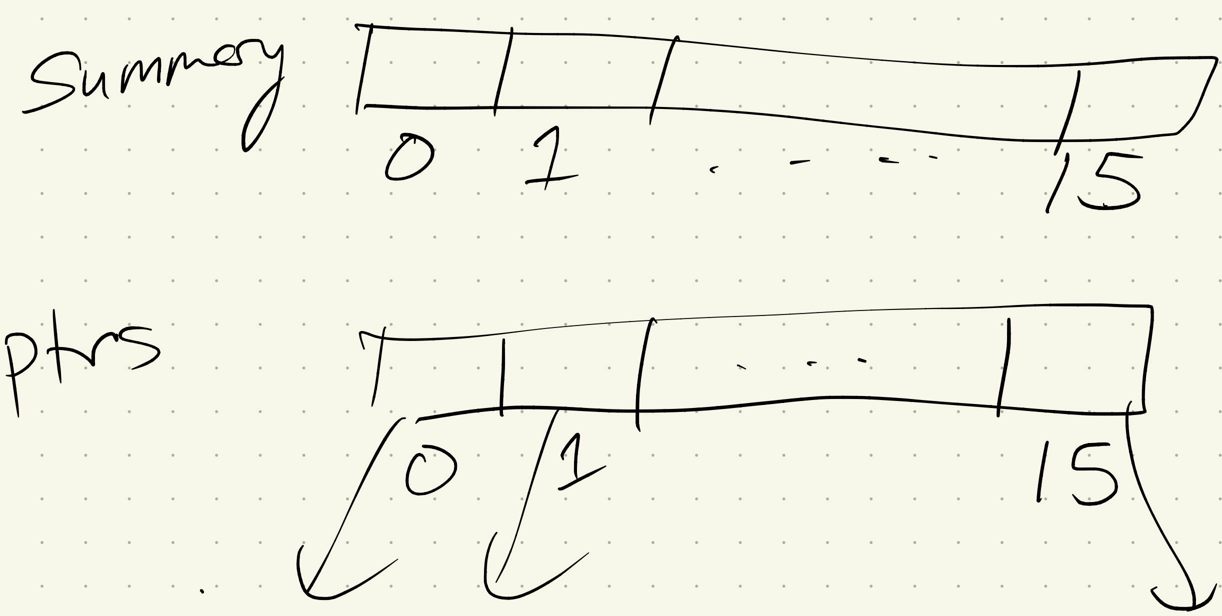


Change: recursively store summary & each level

Summary + data blocks:
each size 256

Apply same construction:

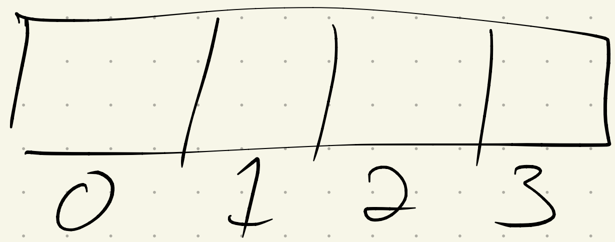
$$\sqrt{256} = 16$$

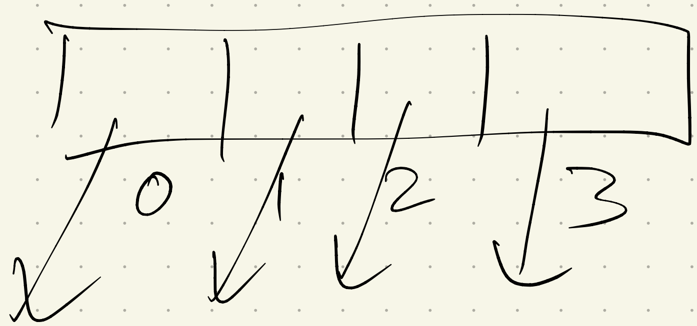


Each of <u>those</u> is size 16.

$$\sqrt{16} = 4$$

So:

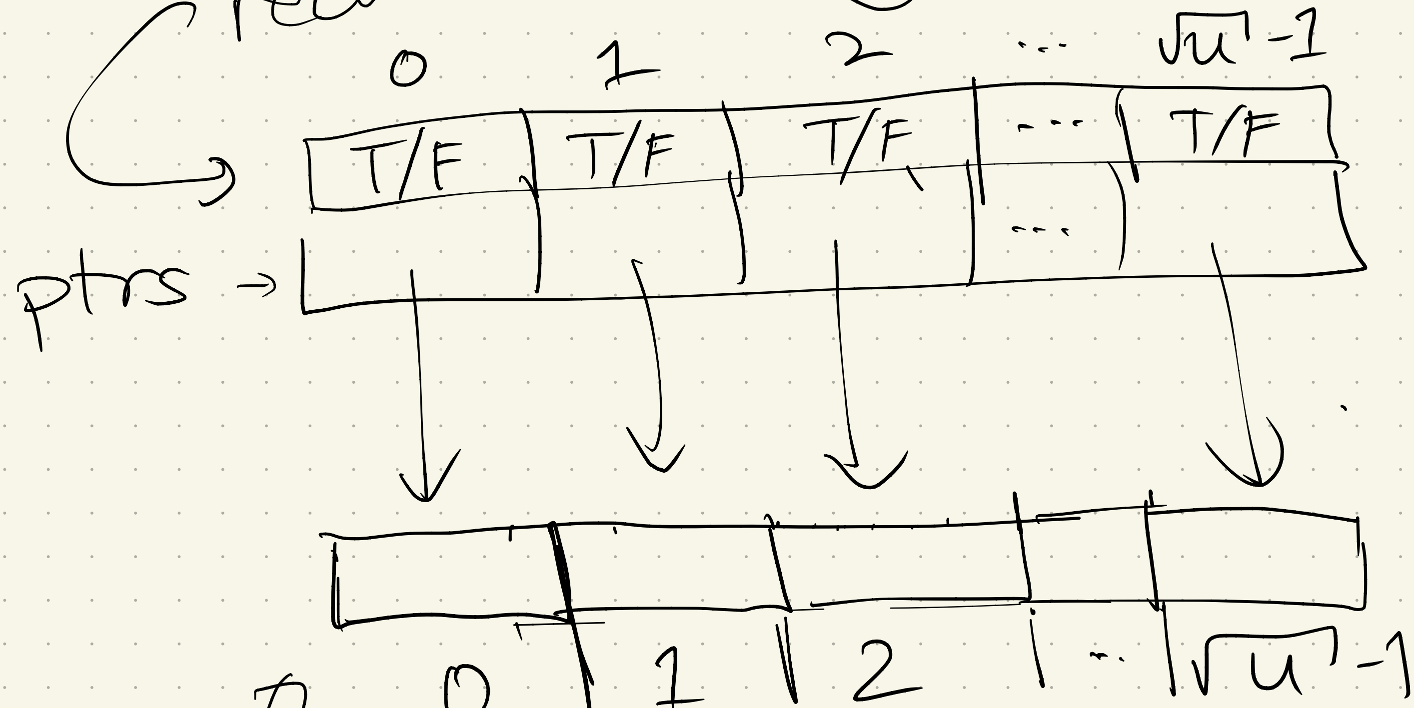




(& stop when ≤ 2)

Details:

recursive summary of size $\sqrt{u}$



each is a DS of size $\sqrt{u}$

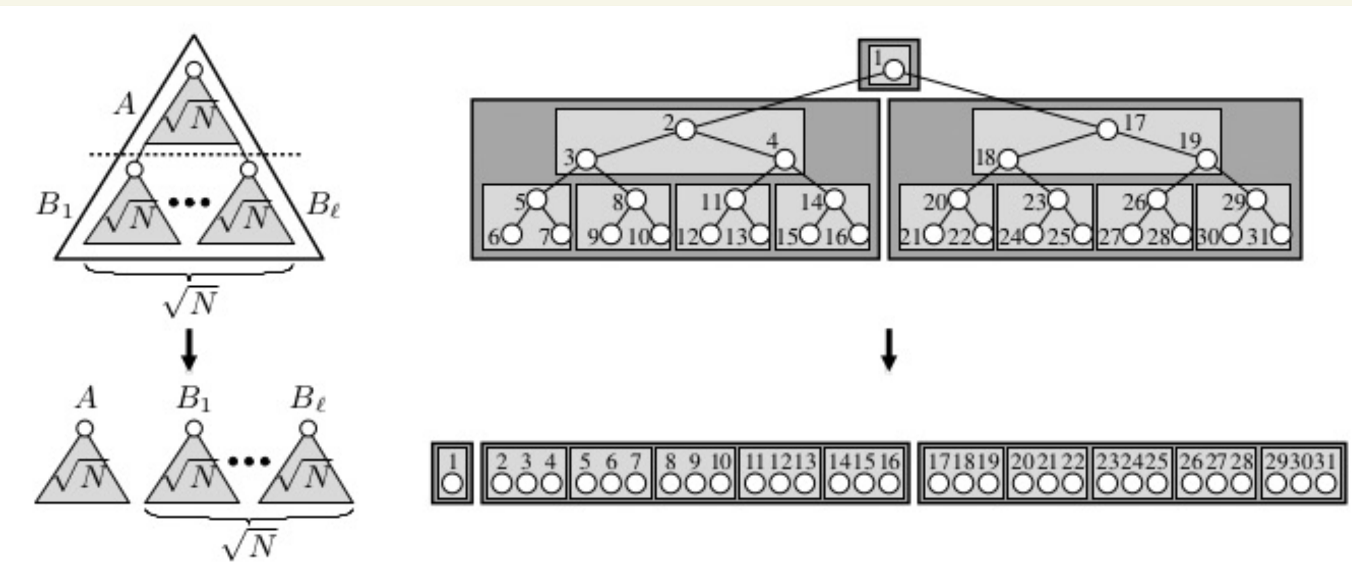


FIG. 2.1. *The van Emde Boas layout. Left: in general; right: of a tree of height 5.*

Now: Implementation!

Lookup(x): 1 lookup
(find element i's T/F in
the $(x \bmod u^{1/2})$ spot
in $\lfloor x/u^{1/2} \rfloor^{th}$ bitvector

→ summary is useless

$$L(u) = 1 + L(\sqrt{u})$$

Insert: ≤ 2 inserts in
smaller data structure
(plus an isEmpty)

IsEmpty(): 1 recursive isEmpty

$S(u) = 1 + S(\sqrt{u})$

Min/Max(): 2 recursive calls

one on summary

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

↳ then on that level recursive structure

Succ/~~Pred~~(x):

max in bottom level,

if max == x,

recursive succ on summary data struc,

& then min in its lower level

Delete(x): 1 delete,

1 isEmpty, & (maybe) another delete on summary

The recursion:

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

or

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

Use domain transformation (link posted):

Let $S(k) = T(2^k) = T(u)$

so $k = \log u$

$\Rightarrow S(k) = 2S(k/2) + 1$

or $= S(k/2) + 1$

& solve

$\log u$

or $\log \log u$

The takeaway:

$O(\log U)$ worst case $\ddot{\frown}$

$O(\log \log U)$ lookups

vs: ~~$O(U)$~~ + $O(1)$

~~$O(\sqrt{U})$~~

but:

U is size of universe!

If $n \geq \log U$,

we beat BST in lookups!

(since $\log n \geq \log \log U$)

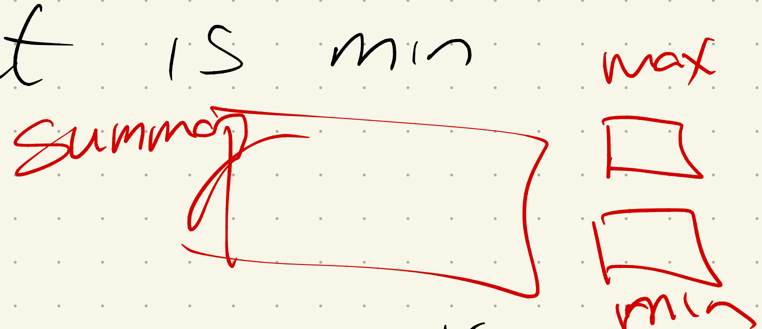
"proto VEB trees)

van Emde Boas tree:

A slight modification of our tiered bitvectors.

Besides summary & $\sqrt{u}$ pointers to next level, we'll also store min & max <ins>separately</ins>. (at each level)

Lookups are unchanged (except we also check if target is min or max)



Important: min & max are <ins>only</ins> stored in special field.

(this changes the code...)

The Good:

- Min, max, & isEmpty are now $O(1)$ time!

  vs. $O(\log U) + O(\log\log U)$ before

- Lookup is unchanged:

  $O(\log\log U)$

  If x isn't max or min, then query $\left\lfloor \frac{x}{U^{1/2}} \right\rfloor$th DS in slot $x \bmod U^{1/2}$

The bad:

- Need to change insert, delete, & succ/pred.

Insert(x):

Basically the same
(≤ 2 inserts in $\sqrt{U}$ DS)

But:
- max + min
- empty case

First attempt:

If tree is empty or size 1:
change max [or / and?] min

Else:
Check max + min
(+ update if needed)
"x" might be old max or min

Then insert(x ~~mod $U^{1/2}$~~)
into ~~$\lfloor x/U^{1/2} \rfloor$~~th DS

If it wasn't empty
insert into summary also

$\Rightarrow$ Runtime: ~~$T(U) = 2T(\sqrt{U}) + O(1)$~~

Doing better:

An observation: If tree is empty, insert runs in $O(1)$ time.

Recall:

If low level is empty:
insert twice
otherwise:
insert once

→ It was empty!!

New recurrence:

$$I(u) = 1 + 1 + I(\sqrt{u})$$

$$\Longrightarrow O(\log \log u)$$ ü

Delete:

Similar setup:

$O(1)$ If size is 1, update min/max & done

Else if min (or max) is deleted, replace with min (or max) of first (or last) non-empty block, & recursively delete that.

$1 + P(\sqrt{u})$ $\Downarrow$ $O(\log \log u)$

Else:

delete $x \bmod u^{1/2}$ from correct subtree

if empty, delete $\lfloor \frac{x}{u^{1/2}} \rfloor$ from summary

Key: again, only delete twice if one was empty!

New recurrence:

$$D(u) = 1 + D(\sqrt{u}) + 1$$

$$= O(\log \log u)$$

Successor(x):

If tree is empty or $x > max$:

declare failure

else if tree $\lfloor \frac{x}{u^{1/2}} \rfloor$ is not empty & $x <$ max in that tree, recursively call successor on that tree

else:

Find successor of $\lfloor \frac{x}{u^{1/2}} \rfloor$ in summary

if it exists, return min in that tree

otherwise return max of summary

Runtime:

$$S(u) = 1 + S(\sqrt{u})$$

$$\Rightarrow \log \log u$$

Takeaway: MAGIC!!

Runtime is $O(\log \log U)$, (or $O(1)$ for min, max & isEmpty)

If $n \gg \log U$: exponentially faster than a BST.

The catch:

Space & hidden big-O

Other cool thing: cache oblivious

Next time:

Switching focus slightly:

- heap variants
  (binomial + Fibonacci heaps)
- and suffix trees