Lecture 9: Integers

- word RAM model (we want to look into the bits of the representation)

predecessor problem

- maintain a set S of n words out of a universe U of size 2^w where
- insert ($x \in U$) w is the word length (e.g. $w = 32$ for 32bit int)
- delete ($x \in S$)
- predecessor ($x \in U$) = $\max\{y \in S, y \le x\}$

assume $w \ge \log n$ which allows to store n in one word

(Note: in comparison model we know that $\Theta(\log n)$ is the answer, LB: sort, UB: BBST)

Technique	Operation	Space		
Van Emde Boas tree (1975)	$O(\log w)$	$O(U)$		
+ hashing	$O(\log w)$ w.h.p.	$\Theta(n)$	} good if w is small	} today
y-fast trees	$O(\log w)$ w.h.p.	$\Theta(n)$		
fusion trees {static / dynamic	$O(\frac{\log n}{\log w})$ w.h.p.	$\Theta(n)$	} good if w is large	} next week

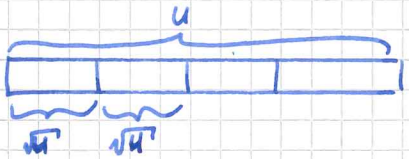
↳ all combined: $\min(\log w, \frac{\log n}{\log w}) \le \sqrt{\log n}$ so provably faster than Bal Bin Search Trees

Van Emde Boas Trees

How to get $\log\log U$? Binary search on the word length?

Main idea is to target this recurrence:

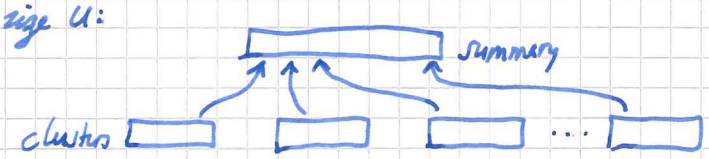
$T(U) = T(\sqrt{U}) + O(1)$ or binary search on w by splitting U into $\sqrt{U}$ clusters of size $\sqrt{U}$

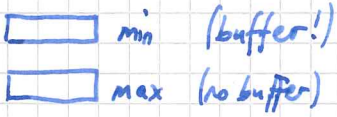


word $x = \langle c, i \rangle$

$c = \lfloor x/\sqrt{U} \rfloor$	$c = x >> (w/2)$
$i = x \,\%\, \sqrt{U}$	$i = x \,\&\, ((1 << (w/2)) - 1)$
$x = c \cdot \sqrt{U} + i$	$x = (c << (w/2)) \mid i$
hard to do on CPU	easier by just halving word length

VEB of size U:



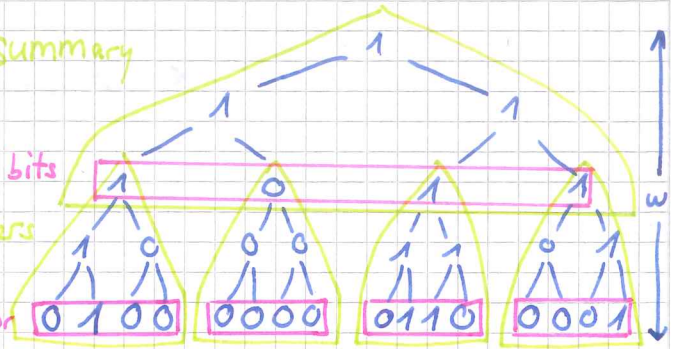


- $V.cluster[i]$ = VEB of size $\sqrt{U}$, $0 \le i \le \sqrt{U}$
- $V.summary$ = VEB of size $\sqrt{U}$ containing the indices of non-empty clusters
- $V.min$ = minimum element in V (not stored recursively!)
- $V.max$ = maximum element in V (also stored recursively, just auxiliary information)

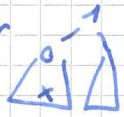
```
successor(V, x = <c,i>)        ← if x < v.min: return v.min
    if i < V.cluster[c].max:                          } search in lower bits
        return <c, successor(v.cluster[c], i)>
    else:
        c' = successor(v.summary, c)                  } search in higher bits
        return <c', v.cluster[c'].min>

insert(V, x = <c,i>):
    if v.min = None: v.min = v.max = x; return   (*)  } update min/max
    if x < v.min: swap x ↔ v.min
    if x > v.max: v.max = x
    if v.cluster[c].min = None                        } recursive insertion
        insert(v.summary, c)                          } If both recursive calls are made, then the second
    insert(v.cluster[c], i)                           } is in O(1), by (*) → so at most one non-trivial
                                                        recursive call!

delete(V, x = <c,i>)
    if x = v.min:
        if x = v.max:
            v.max = v.min = None; return (*)          } update min
        c = v.summary.min
        i = v.cluster[c].min
        v.min = <c,i>
    delete(v.cluster[c], i)                           } recursive deletion
    if v.cluster[c].min = None:                       } (again: if two calls are made, one is trivial by (*))
        delete(v.summary, c)
    if v.summary.min = None:                          } update max
        v.max = v.min
    else:
        c' = v.summary.max
        v.max = <c', v.cluster[c'].max>
```

→ recurrence $T(2^w) = T(2^{w/2}) + O(1)$ → $T(2^w) = \log(w)$

Saving Space: Expanded tree view:



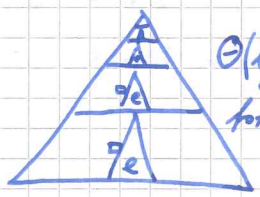
first idea: (90's)
- node = OR of its children
- trie store min/max per subtree
- linked list of 1-bits
- update: $O(w)$
- query: bin search on the leaf-root-path for 1 pattern $\rightarrow O(\log w)$



↝ problem: $O(w)$ update and space $O(U)$

second idea: (2011)
- don't store empty clusters in VEB
- v. clusters = hash table $\Theta(1)$ w.h.p. (FKS perfect hashing)
- space = $O(\#\text{non-empty clusters} + 1)$
- change each table entry to min in its child

↝ improves space to $O(n \cdot \lg w)$

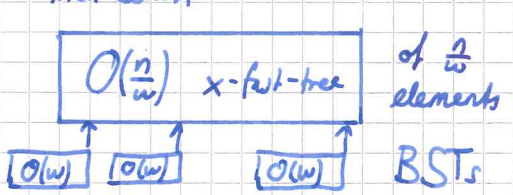


third idea: (x-fast trees 1977)
- take tree view idea from idea 1
- store hash table of each level 1's position

↝ gives $O(\log w)$ query, $O(w)$ update in $O(n \cdot w)$ space (w.h.p.)

finally: y-fast trees: x-fast trees + indirection

- $O(\log w)$ query
- $O(\log w)$ amortized update w.h.p.
- $O(n)$ space



query: top: $O(\log w)$, bottom $O(\log w)$
update: $O(\log w)$ bottom, $O(w)$ amortized to $O(w)$ changes
space: $O(\frac{n}{w} \cdot w + n) = O(n)$

same indirection idea works on the second idea to get to $O(n)$ space