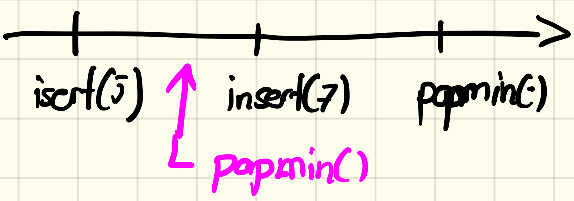
L5 20.3.17

# Topic: Temporal II
## Retroactive DS

A data structure ≃ sequence of updates




A <u>retroactive</u> D.S. allows changes to sequence

- Insert(t, "op(...)")
- Delete(t)

} everywhere

- Query(t, "op(..)")

↳ partial r.: query "NOW"

↳ full r.: query any t

(1) Commutative Updates

$$\text{Insert}(t, x) = \text{Insert}(\text{now}, x)$$

(2) Invertible Updates

$$\text{Delete}(t) = \text{Insert}(\text{now}, x^{-1})$$

(1) + (2) $\Rightarrow$ partial r. easy.

E.g.)
- Hashing (assuming no collisions)
- Array with arithmetic updates $A[i] \mathrel{+}= \Delta$
- ordered set

What do we need for full RA.?

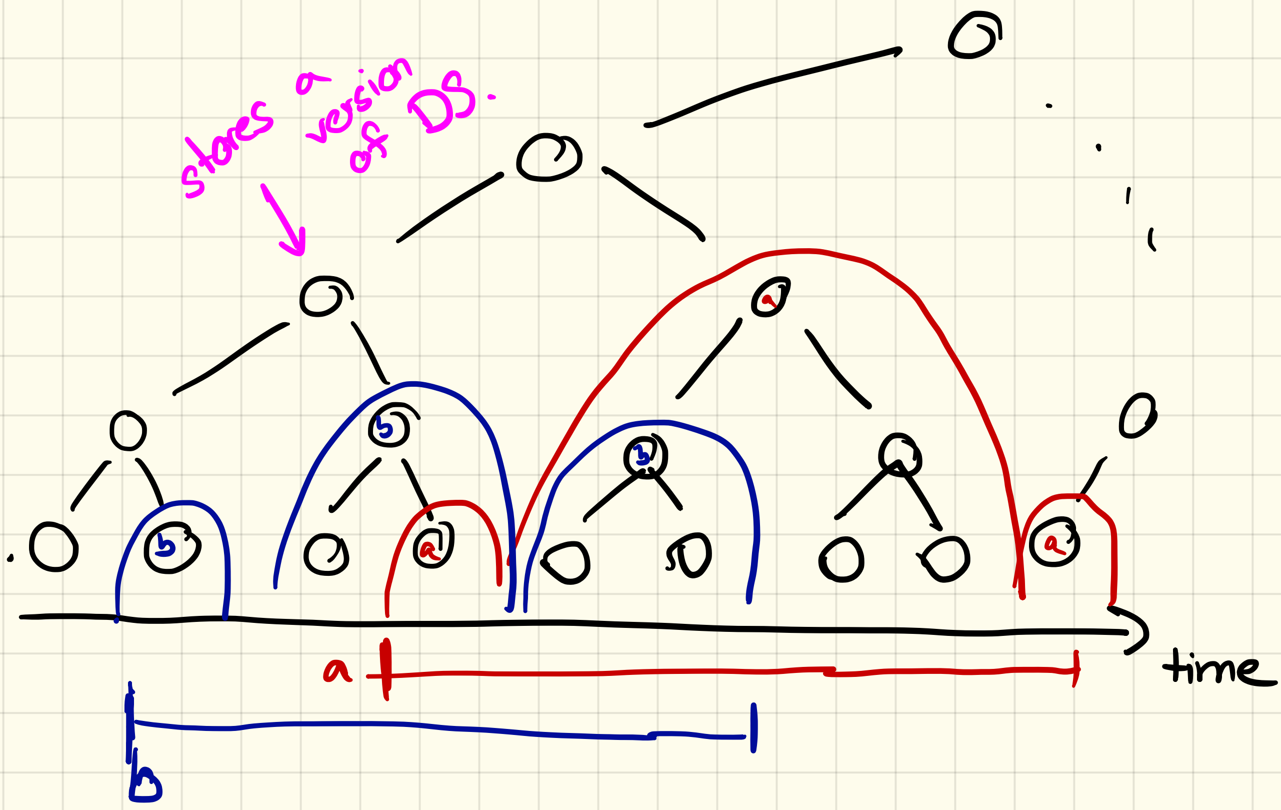
$\rightarrow$ Decomposable $\underbrace{\text{Search Queries}}_{\text{Set + Queries}}$

$$\text{Query}(x, A \cup B) = \text{Query}(x, A) \circ \text{Query}(x, B)$$

E.g.) • Nearest neighbor
• Successor

$\Rightarrow$ Full RA with $\log(m)$ overhead
$\uparrow$ # updates.

Solution: Segment tree



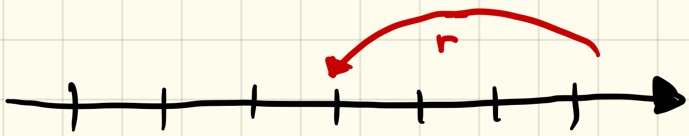
Ins, Del $\rightarrow$ modify element existence interval

$\rightarrow$ $O(\log m)$ steps / update

Query($t$) $\rightarrow$ walk up from $t$ to root

What about general transformations?

- Rollback method:




unwind r ops, perform update, re-do r ops.

$\longrightarrow$ Overhead $O(r)$

E.g.) Maintain: $X, Y$

Updates: $X = \alpha$, $Y \mathrel{+}= \Delta$, $Y = X \cdot Y$

$X = x$, $Y += a_n$, $Y = X \cdot Y$, $Y += a_{n-1}$, $Y = X \cdot Y$, ...., $Y += a_0$

$\rightarrow Y = a_n x^n + \dots + a_0$

alg. pec. trees

Update Query $\rightarrow$ Eval of polynomial $\Omega(n)$

Priority queue: Insert( ), Pop-Min( )

Demaine et. al. 2003:

partially retroactive $O(\log n)$/OP

proof: nice, but (currently) omitted

Succesor query:

| | |
|---|---|
| Partial | $O(\log m)$ |
| Full | $O(\log^2 m)$ ← fully decomposable |
| | $O(\log m)$ ← hard (= ugly, convoluted) |

Giora & Kaplan 09

**Aim:** list order maintenance

$\mathcal{O}(1)$: Insert, Remove

Query$(x, y)$: $x \overset{?}{<} y$

Start with a different problem

**Label space maintenance**:

Maintain linked list with monotone labels (integers)

E.g.)

$5 \leftrightarrow 7 \leftrightarrow 9$

$\Downarrow$ Ins

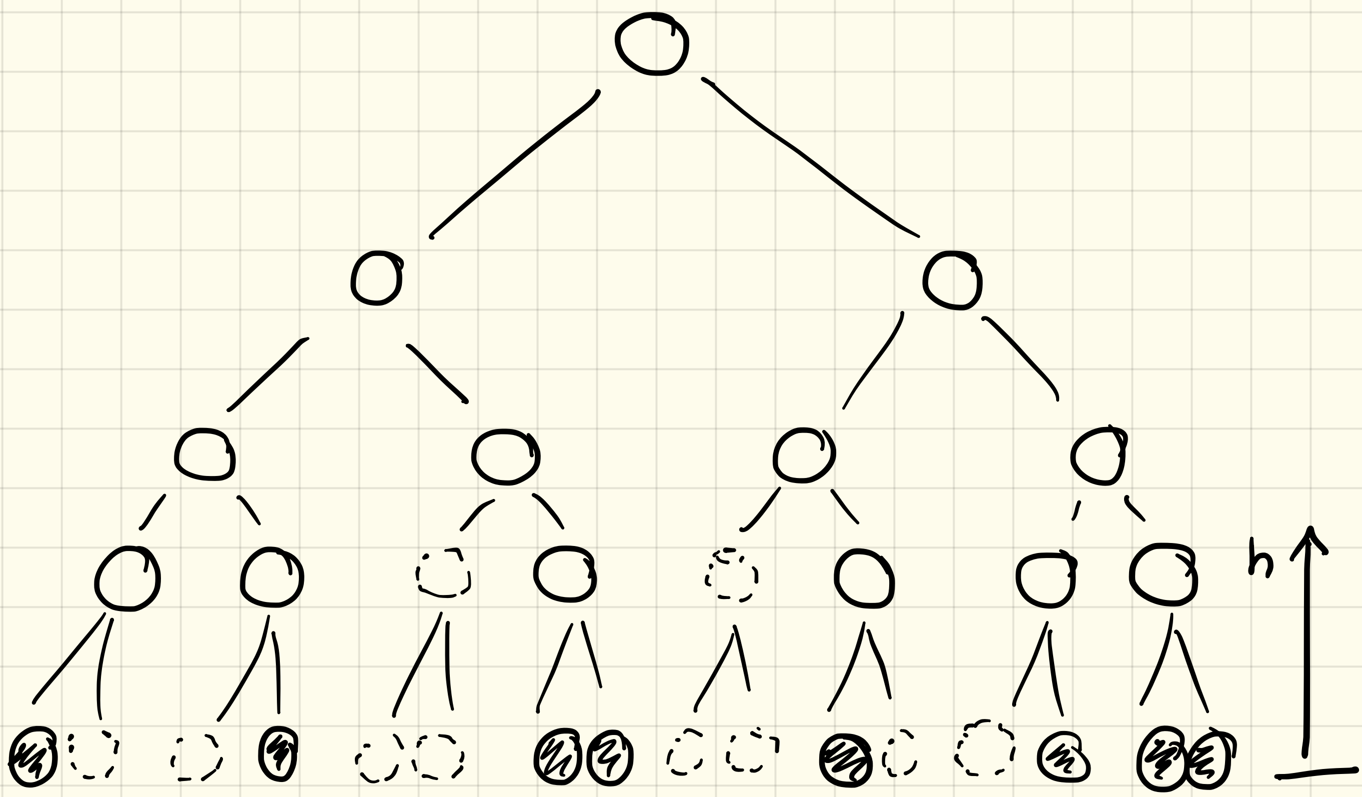
$5 \leftrightarrow 6 \leftrightarrow 7 \leftrightarrow 9$

$\Downarrow$ Ins

$4 - 5 - 6 - 42 - 109$

| Label space | Label size | Time |
|---|---|---|
| $\Theta(n)$ - $\Theta(n \log n)$ | $\Theta(\log n)$ | $\Theta(\log^2 n)$ |
| $\Theta(n^{1+\varepsilon})$ - $\Theta(\mathrm{poly}(n))$ | $\Theta(\log n)$ | $\Theta(\log n)$ today |
| $\Theta(2^n)$ | $\Theta(n)$ | $\Theta(1)$ |

trivial
insert $\rightsquigarrow$ take average




Sparse tree of labels = Trie of labels

Space below node at height h = #space = $2^h$

#usage = number of used nodes below

density of node = $\#usage / \#space \overset{!}{\leq} 1/\alpha^h$

where $1 < \alpha < 2$.

Update: Delete ✓

Insert

(1) Try to squeeze element

(2) Walk up the tree until you find node X with ~~good~~ small density

↳ rebalance whole subtree

Cost = $\mathcal{O}(2^h/\alpha^h)$

Idea:

Single child of X: Y is also balanced

Child density - child threshold

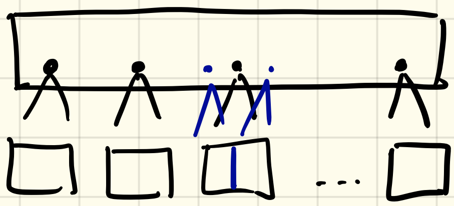
$\geq 1/\alpha^{h-1} - 1/\alpha^h = \Theta(1/\alpha^h)$

→ $\Omega(2^h/\alpha^h)$ in a single child to make it unbalanced.

⇒ $\mathcal{O}(1)$ amortized per anchestor

$\Rightarrow$ $O(\log n)$ amortized in total.

Next: List order maintenance



1) $m/\log n$ labels
2) $\log n$ labels

1 level of indirection

1) $\Theta(\log n)$ time, label size $\Theta(\log n)$
2) $\Theta(1)$ time, label size $\Theta(\log n)$

Label = ( top label, bottom label)

box too large : split it
box too small : merge it w. neighbor

Update: Bottom $O(1)$

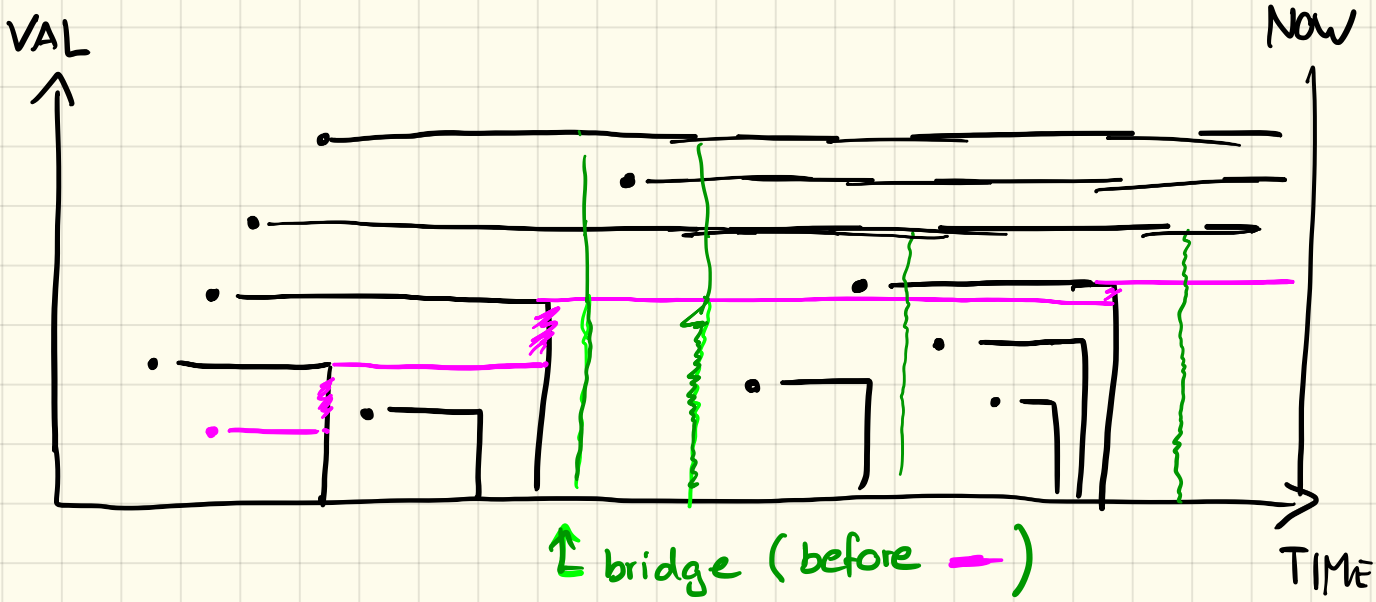
Top $O(\log n)$ worst case,

$O(1)$ amortized

L6 (part 2) 27.3.17

# Partially Retroactive Priority Queue

$O(\log n)$/OP (oth. sort below $n \log n$)




LView: insert = rightward ray

del-min = upward ray

Insert($t$, "insert($k$)") inserts value into $Q_{now}$

" delete("delete-min)

$= \max\{k, k' : k' \text{ deleted at time} \geq t\}$

Bridge at time $t$ : iff $Q_t \subseteq Q_{now}$

$t'$ is bridge preceeding time $t$

$\max\{k' \notin Q_{now} \quad k' \text{ inserted} \geq t'\}$

- Store $Q_{now}$ as Balanced BST (by value)
- Store BBST on leaves = inserts (by time)

  $\forall$ $\text{node}_x$ $\max\{k' \in Q_{now} : k' \in x \text{ subtree}\}$
- Store BST on leaves = updates

  +1 Insert(k) $\quad k \in Q_{now}$

  -1 Del-Min

  0 Insert(k) $\quad k \notin Q_{now}$

  Node store subtree sums, min/max of prefix sums