Today: temporal data structures II
- partial retroactivity
- full retroactivity
- nonoblivious retroactivity

Think: time travel

Retroactivity: [Demaine, Iacono, Langerman - T. Alg. 2007]
- traditional DS formed by sequence of updates
- allow changes to that sequence (destroying old ver.)
- maintain linear timeline

\[ t = \emptyset \rightarrow \text{insert(5)} \rightarrow \text{insert(7)} \rightarrow \text{delete-min} \rightarrow \cdots \rightarrow \text{now} \]

Dr. Who, Timecop, Back to the Future

- ops:
  - Insert(\( t, \text{"op(--)"} \)): retroactively do op() at time \( t \)
  - Delete(\( t \)): retroactively undo op. at time \( t \)
  - Query(\( t, \text{"op()"} \)): execute query at time \( t \) (relative to current timeline only)

- time specified as index, or via order-maintenance DS
- partial retroactivity: Query only in present (last \( t \))
- full retroactivity: Query at any time "Q" in Star Trek
**Easy case:**
- commutative updates: \( x \cdot y = y \cdot x \)
  \( \Rightarrow \text{Insert}(t, x) \equiv x \text{ in present} \)
- invertible updates: \( x \cdot x^{-1} = 0 \)
  \( \Rightarrow \text{Delete}(t) \equiv x^{-1} \text{ in present} \)
  \( \Rightarrow \text{partial retroactivity easy (update in present)} \)

- e.g. hashing, or array with \( A[i] \leftarrow \Delta \)
- e.g. search problem: maintain set \( S \) of objects
  subject to \( \text{query}(x, S) \) for object \( x \)
  & insert/delete objects
- decomposable search problem: [Bentley & Saxe - 1980]
  \( \text{query}(x, A \cup B) = \text{f(query}(x, A), \text{query}(x, B)) \)
- e.g. nearest neighbor, successor, point location
  
- full retroactivity in \( O(lg m) \) factor overhead
  via segment tree:

\[ \text{balanced BST} \]

- time interval maps to \( O(lg m) \) subtree intervals
- Insert/Delete modify element’s existence interval
  \( \Rightarrow O(lg m) \) updates to DSs in nodes
- Query combines \( O(lg m) \) searches via \( f \)
General transformations: 

- rollback method: retro. op. r time units in past with factor-r overhead via logging ("undo persistence")

- lower bound: $\Omega(r)$ overhead can be necessary
  - DS maintains two values $X$ & $Y$, initially $\emptyset$
  - ops: $X = x$, $Y = \Delta$, $Y = X \cdot Y$, query: return $Y$
  - $O(1)$ time/op. in "straight-line program" model
  - $Y = a_n$, $Y = X \cdot Y$, $Y = a_{n-1}$, $Y = X \cdot Y$, ..., $Y = a_0$

  Computes poly. $a_n x^n + a_{n-1} x^{n-1} + ... + a_0$ [Horner's Rule]

- Insert($t = \emptyset$, "$X = x$") changes $X$ value

- evaluating degree-$n$ polynomial requires $\Omega(n)$ worst-case arithmetic ops. in any field, independent of $a_i$ preprocessing, in "history-independent algebraic decision tree"

  $\Rightarrow$ integer RAM $\Rightarrow$ generalized real RAM

  [Frandsena, Hansenb, Miltersen – I&C 2001]

- cell-probe lower bound: $\Omega(\sqrt[2]{r})$
  - DS maintains $n$ words; arithmetic updates +&
  - compute FFT using $O(n \log n)$ ops.
  - changing $w_i$ requires $\Omega(n)$ cell probes

  [Frandsena et al. 2001]

- OPEN: $\Omega(r/\text{poly} \log r)$ cell-probe lower bound?
Priority queues: [Demaine, Iacono, Langerman 2003]

- insert & delete-min, partially retroactive in \(O(\log n/\log\log n)\) per operation
- assume keys inserted only once
- \(L\) view: insert = rightward ray
delete-min = upward ray

Also, Delete("delete-min")

- Insert\((t, "insert(k)")\) inserts into \(Q\) now
  \[ \max \{ k, k' \} \) k' deleted at time \( \geq t \}

- Bridge at time \( t \) if \( Q_t \subseteq Q\) now

- If \( t' \) is the bridge preceding time \( t \)
  then \( \max \{ k' \mid k' \text{ deleted at time } \geq t \}\)
  \[ = \max \{ k' \in Q_{t'} \mid k' \text{ inserted at time } \geq t' \}\]
- store Qnow as balanced BST; one change/update
- store balanced BST on leaves = insertions, ordered by time, augmented with
  \forall node x: max\{k', \in Qnow \mid k' inserted in x's subtree\}
- store balanced BST on leaves = updates, ordered by time, augmented with
  0 for insert(k) with k \in Qnow
  +1 for insert(k) with k \notin Qnow
  -1 for delete-min

& subtree sums & subtree min/max prefix sums

\psi bridge = prefix summing to 0
\psi can find preceding bridge, change to Qnow in O(lg n) time

Other structures:
- queue: O(1) partial, O(lg m) full
- deque: O(lg m) full
- union-find (incremental connectivity): O(lg n) full
- priority queue: O(\sqrt{m} lg m) full
  [via general partial->full transform \times O(\sqrt{m})]
- successor: O(lg m) partial via search
  O(lg^2 m) full via decomposable search
  O(lg m) full [Giora & Kaplan - T.Als. 2009]
  \subseteq uses fractional cascading \[L3\]
  & van Emde Boas \[L11\]
Nonoblivious retroactivity: [Acar, Blelloch, Tangwongsan - CMU TR 2007]
- in algorithmic use of DS (e.g. priority queue in Dijkstra)
  updates performed depend on results of queries
  ⇒ put queries on timeline too
- retroactive update may change result of future queries
- new retro DS query: time of earliest error
- assume that algorithm corrects errors by further retroactive updates (e.g. Delete & re-Insert query)
  in increasing time order always ≤ errors
  - idea: just rerunning what’s changed of algorithm

Priority queue: insert, delete, & min in $O(\log n)$ time/op.

- invariant: all crossings involve horiz segments
  with left endpoint left of all errors
- maintain lowest leftmost crossing
  = leftmost lowest crossing
- Assume keys inserted only once
- Maintain earliest floating error on each key row
- Maintain priority queue on all errors by time
  \( \Rightarrow \) always know earliest error

- Insert \((x, \text{"min"})\): upward ray shot
  = fully retroactive successor \((-\infty) \preceq O(lg m)\)
  = fully retroactive insert, delete, min
    (decomposable search problem \(\sim\) but then \(lg^2 m\))

- Insert \((x, \text{"insert(y)"})\) / Delete \((x, \text{"delete(y)"})\):
  rightward ray shot to find earliest crossing
  (if lower than existing lower left crossing)
  = fully retroactive successor \((x) \preceq O(lg m)\)
  \(\cdots\) when all inserts are at time \(-\infty\)

- Insert \((x, \text{"delete(y)"})\) / Delete \((x, \text{"insert(y)"})\):
  - if was lowest crosser, find next by upward ray shot from leftmost crosser query
  - rightward ray shot to find earliest floater

- Delete \((x, \text{"min"})\):
  - if floating: rightward ray shot to next in row
  - if leftmost crosser: find next by upward ray shot for next min query (successor among queries)