Lecture 7

Temporal data structures (2 kinds)

branching-universe model of time travel

Persistence: keep all versions of DS
- operations specify which version
- update creates new version (instead of modifying)
- 4 levels:
  1. partial persistence: update only latest version
     ⇑ ⇒ versions linearly ordered (think Déjà Vu)
  2. full persistence: update any version
     ⇑ ⇒ versions form tree
  3. confluent persistence: combinators make new
     ⇑ version from >1 given version ⇒ version DAG
  4. functional: never modify nodes; only create new

Partial persistence: [Driscoll, Sarnak, Sleator, Tarjan - JCSS 1989]
- given pointer-machine DS
- suppose $\le p$ nodes point to any node, $p = O(1)$
- store reverse pointers for most recent version
- allow $\le p$ (time, field, value) mods. in a node
- when update changes a field:
  - if node not full, just add mod.
  - else: copy node-with-mods, recurse on rev. ptrs.
- Φ = # full latest nodes ⇒ $O(1)$ amortized overhead

Full persistence: [Driscoll et al. again]

- again assume pointer machine, $\le p$ incoming ptrs/node
- version list = pre-order traversal of version tree
- list-order DS: insert node after specified node
  order query: is node x before y?
  $O(1)$ time/op. [Dietz & Sleator – STOC 1987] [L19]
- space for up to $2p$ mods. in a node
- when a node is full, split into two roughly half-full nodes (like B-trees)
- Φ = # full nodes $\Rightarrow O(1)$ amortized cost
- linked list of nodes representing DS node
- second phase to update reverse pointers

- $O(1)$ worst-case partial persistence [Brodal – NJC 1996]

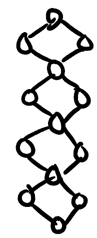
OPEN: $O(1)$ worst-case full persistence?

- $O(\lg\lg n)$ fully persistent array $\Rightarrow$ any RAM DS [Dietz – WADS 1989]

OPEN: matching lower bound? what about partial?
- possibly solved by Pătraşcu et al. (unpublished)

Confluent persistence

- functional data structures e.g. Okasaki 2003 book
- e.g. deques with concat. in $O(1)$/op. [Kaplan, Okasaki, Tarjan - SICOMP 2000]
  double-ended queues (push/pop front/back)
- general transformation: [Fiat & Kaplan - JAlg. 2003]
  - $d(v)$ = depth of node v in version DAG
  - $e(v) = 1 + \lg(\#\text{ paths from root to } v)$
  - overhead: $\lg(\#\text{updates}) + \max_v e(v)$
  - poor when $e(v) = 2^{\#\text{updates}}$ e.g. →



    - can make exponential-size DS this way
    - in this case still exponentially better than nonpersist.

OPEN: when can you do better?
- lists with split & concatenate?
- trees
- general pointer machine
- arrays with... cut & paste?

Retroactivity [Demaine, Iacono, Langerman - SODA 2004 & TAlg 2007]

- traditional DS formed by sequence of updates
- allow changes to that sequence
- maintain linear timeline

$\boxed{\text{round-trip time travel}}$

- operations:
  - Insert(t, "op(·)"): retroactively do op. @ time t
  - Delete(t): retroactivity undo op. @ time t
  - Query(t, "op(··)"): execute op. query @ time t
- partial retroactivity: Query only in present (last t)
- full retroactivity: Query at any time

Easy cases:

- commutative updates: $x, y \equiv y, x$
  - $\Rightarrow$ Insert(t, x) can just do x in present
- invertible updates: $x, x^{-1} \equiv \emptyset$
  - $\Rightarrow$ Delete(t) can just do x^{-1} in present
- e.g.: insert/delete keys; array with $A[i] \mathrel{+}= \Delta$

$\Rightarrow$ partial retroactivity is easy

- search problem: maintain set S of objects subject to query(x, S) for object x & insert/delete objects $\Rightarrow$ commutative & invertible
- decomposable search problem: [Bentley & Saxe - J.Alg. 1980]

  $\text{query}(x, A \cup B) = f(\text{query}(x, A), \text{query}(x, B))$

  - e.g.: nearest neighbor, successor, point location
- full retroactivity in $O(\lg n)$ overhead via segment tree

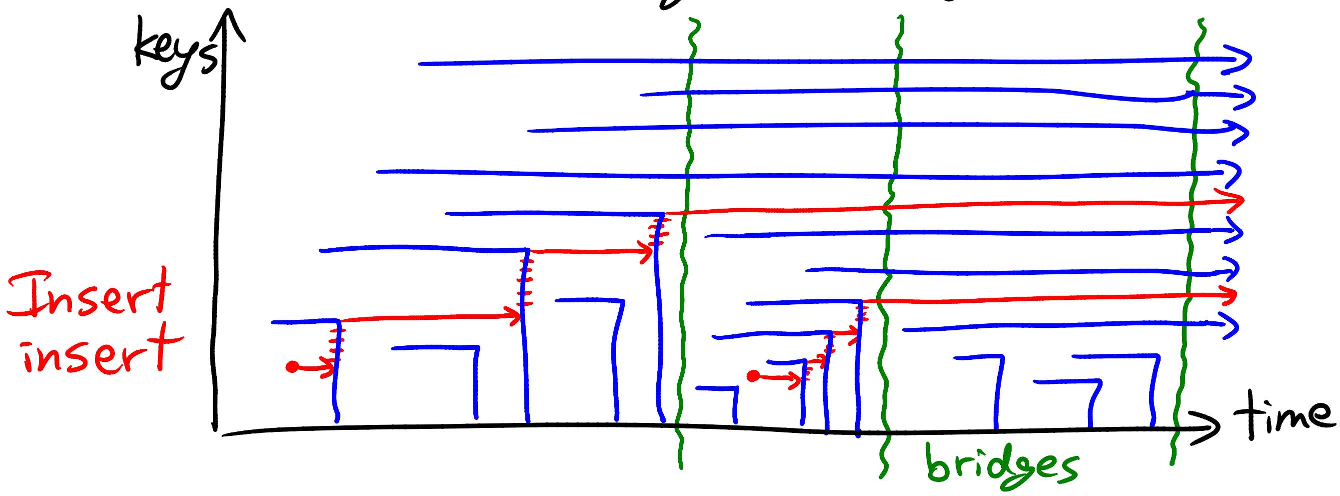
General transformations:

- rollback method: retro. op. @ r time units in past with factor-r overhead, via logging ("undo persistence")
- lower bound: $\Omega(r)$ can be necessary! :(
  - DS maintains two values, X & Y, initially 0
    - $\text{setX}(x)$: $X \leftarrow x$ — $\text{addY}(\Delta)$: $Y \leftarrow Y + \Delta$
    - $\text{mulXY}()$: $Y \leftarrow X \cdot Y$ — $\text{query}()$: return Y
  - trivial $O(1)$ time/op. in "straight-line program model"
  - $\text{addY}(a_n)$, $\text{mulXY}()$, $\text{addY}(a_{n-1})$, $\text{mulXY}()$, …, $\text{addY}(a_0)$ computes the polynomial $a_n x^n + a_{n-1} x^{n-1} + \dots + a_0$
  - Insert($t=0$, "setX(x)") changes x value
  - requires $\Omega(n)$ arithmetic ops. over any field, even with any infinite subset such as integers, independent of a_i preprocessing, in worst case, in "history-independent algebraic decision tree" $\Rightarrow$ integer RAM $\Rightarrow$ generalized real RAM [Frandsen, Hansen, Miltersen — I&C 2001]
- cell-probe lower bound: $\Omega(\sqrt{r/\lg r})$
  - DS maintains n words; arithmetic updates $+/\cdot$
  - compute FFT in $O(n \lg n)$
  - changing w_i has $\Omega(\sqrt{n})$ LB [Frandsen et al. 2001]

$\boxed{\text{OPEN: } \Omega(r/\text{poly}\lg r) \text{ cell-probe lower bound?}}$

Priority queues: insert, delete-min (not commut.)

- partial retroactivity in $O(\lg n)$/op.
- assume keys only inserted once
- **L view:** insert = rightward ray, delete-min upward



- Insert(t, "insert(k)") inserts into Q_{now} $\max\{k, k' \in Q_{now} \mid k' \text{ deleted at time} \geq t\}$ ← hard to maintain
- bridge at time t if $Q_t \subseteq Q_{now}$
- if t' is last bridge before time t, then $\max\{k' \mid k' \text{ deleted at time} \geq t\}$ $= \max\{k' \notin Q_{now} \mid k' \text{ inserted at time} \geq t'\}$
- store BST on leaves = insertions, ordered by time with $\max\{k' \notin Q_{now} \mid k' \text{ inserted in } x\text{'s subtree}\}$ $\forall x$
- store BST on leaves = updates, ordered by time with 0 for insert(k), $k \in Q_{now}$, +1 for insert(k), deleted, −1 for delete-min & subtree sums
  $\Rightarrow$ bridge = prefix summing to 0
- store Q_{now} explicitly: one change/update

Other structures:

- queue: $O(1)$ partial, $O(\lg m)$ full
- deque: $O(\lg n)$ full
- union-find (incremental connectivity): $O(\lg m)$ full
- priority queue: $O(\sqrt{m} \lg m)$ full
  (via general partial → full transform, $\cdot O(\sqrt{m})$)
- successor: $O(\lg m)$ partial trivial,
  $O(\lg^2 m)$ full easy
  - $O(\lg m)$ full [Giora, Kaplan - SODA 2007]