Temporal data structures (2 kinds)

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
     \[ \Rightarrow \text{versions linearly ordered} \]
  2. **full persistence**: update any version
     \[ \Rightarrow \text{versions form tree} \]
  3. **confluent persistence**: combinators make new
     version from >1 given version \[ \Rightarrow \text{version DAG} \]
  4. **functional**: never modify nodes; only create new

Partial persistence: \[ \text{[Driscoll,Sarnak,Sleator,Tarjan-JCSS 1989]} \]
- given pointer-machine DS
- suppose \( p \) nodes point to any node, \( p=O(1) \)
- store reverse pointers for most recent version
- allow \( p \) (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.
- \( \Phi = \# \text{full latest nodes} \Rightarrow O(1) \text{ amortized overhead} \)
Full persistence: [Driscoll et al. again]
- again assume pointer machine, ≤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] [L.19]
- 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 ⇒ 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 ⇒ 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 (unpublished)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, double-ended queues (push/pop front/back) Tarjan - SICOMP 2000]
- general transformation: [Fiat & Kaplan - JAlgo. 2003]
  - \( d(v) \) = depth of node \( v \) in version DAG
  - \( e(v) = 1 + \log \) (# paths from root to \( v \))
  - overhead: \( \log \) (# updates) + \( \max \ e(v) \)
  - poor when \( e(v) = 2 \) (# updates) e.g.
  - can make exponential-size DS this way
  - in this case still exponentially better than nonpersistent.

OPEN: when can you do better?
- lists with split & concatenate?
- trees
- general pointer machine
- arrays with... cut & paste?
Retroactivity [Demaine, Iacono, Langerman, & Tarjan 2007]
- Traditional DS formed by sequence of updates
- Allow changes to that sequence
- Maintain linear timeline
- Operations:
  - Insert(t, “op(·)“): retroactively do op at time t
  - Delete(t): retroactivity undo op at time t
  - Query(t, “op(·)”): execute op query at time t
- Partial retroactivity: Query only in present (last t)
- Full retroactivity: Query at any time

Easy cases:
- Commutative updates: x, y = y, x
  \[ \Rightarrow \text{Insert}(t, x) \text{ can just do } x \text{ in present} \]
- Invertible updates: x, x^{-1} = \emptyset
  \[ \Rightarrow \text{Delete}(t) \text{ can just do } x^{-1} \text{ in present} \]
- E.g.: insert/delete keys; array with A[i] += \Delta
  \[ \Rightarrow \text{Partial retroactivity is easy} \]
- Search problem: maintain set S of objects subject to query(x, S) for object x
  \& insert/delete objects \Rightarrow \text{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(log n) overhead via segment tree
General transformations:

- **rollback method**: retro. op @ t 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
  - \( \text{set } X(x): X \leftarrow x \)
  - \( \text{add } Y(\Delta): Y \leftarrow Y + \Delta \)
  - \( \text{mul } XY(): Y \leftarrow X \cdot Y \)
  - \( \text{query } (): \text{return } Y \)

- trivial \( O(1) \) time/op in "straight-line program model"
- \( \text{add } Y(a_n), \text{mul } XY(), \text{add } Y(a_{n-1}), \text{mul } XY(), \ldots, \text{add } Y(a_0) \)
  - computes the polynomial \( a_nx^n + a_{n-1}x^{n-1} + \ldots + a_0 \)
- \( \text{Insert}(t=0, \"set } X(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"

  \[ \Rightarrow \text{integer RAM } \Rightarrow \text{generalized real RAM} \]

  [Frandsen, Hansenb, Miltersen-I&C 2001]

- **cell-probe lower bound**: \( \Omega(\sqrt{r/lg r}) \)
  - DS maintains \( n \) words; arithmetic updates \(/\)
  - compute FFT in \( O(n lg n) \)
  - changing \( w_i \) has \( \Omega(\sqrt{n}) \) LB [Frandsen et al. 2001]

**OPEN**: \( \Omega(r/poly lg r) \) 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 Qnow
  max \{k, k' \in Qnow \mid k' deleted at time ≥ t\} hard to maintain
- bridge at time t if Qt ⊆ Qnow
- if t' is last bridge before time t,
  then max \{k \mid k' deleted at time ≥ t\}
  = max \{k' \in Qnow \mid k' inserted at time ≥ t'\}
- store BST on leaves = insertions, ordered by time
  with max \{k' \in Qnow \mid k' inserted in x's subtree\} \forall x
- store BST on leaves = updates, ordered by time
  with 0 for insert(k), k \in Qnow, +1 for insert(k), deleted,
  -1 for delete-min & subtree sums
  ⇒ bridge = prefix summing to 0
- store Qnow 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 \(\rightarrow\) full transform, \(\circ O(\sqrt{m})\))
- successor: \(O(lg m)\) partial trivial, \(O(lg^3 m)\) full easy
- \(O(lg m)\) full [Giora, Kaplan - SODA 2007]