

TODAY: temporal data structures

- persistence
- retroactivity

think:
time travel

Persistence:

- keep all versions of DS
- operations specify which version
- update creates new version
(never modify a version)
- 4 levels:

branching-
universe model

TV series
Flash Forward?

① partial persistence:

update only latest version
⇒ versions linearly ordered

movie
Déjà Vu

part 1

Déjà Vu

Part 2

② full persistence:

update any version

⇒ versions form a tree

Pullman's book
Subtle Knife?

③ confluent persistence:

can combine >1 given version into new V.

⇒ versions form a DAG

movie Primer?

④ functional:

never modify nodes; only create new
(pointer machine)

Version of DS represented by node ptr.

Partial persistence: [Driscoll, Sarnak, Sleator, Tarjan
any pointer-machine DS
with $\leq p = O(1)$ pointers to any node (in any version)
can be made partially persistent
with $O(1)$ amortized multiplicative overhead
(& $O(1)$ space / change)]

- JCSS 1989

Proof:

- store reverse pointers for nodes in latest version
- allow $\leq p+1$ (version, field, value) mods. in a node (using $p=O(1)$)
- to read node.field at version v , check for mods with time $\leq v$
- when update changes $\text{node.field} \leftarrow x$
 - if node not full: add mod. (now, field, x)
 - else: create node with mods. applied (& no mods.)

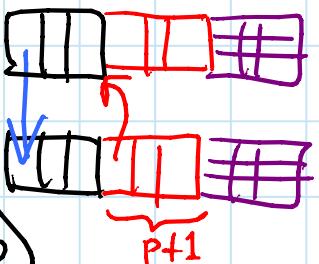
update back pointers to this node

(found via pointers)

in worst case, they

all move $\Rightarrow p$ mods. } recursively change pointers to
 \Rightarrow still not full } this node (found via back ptrs.)

- $\Phi = \#$ full (p mods.) nodes in latest version
- $\Rightarrow O(p) = O(1)$ amortized cost per change \square



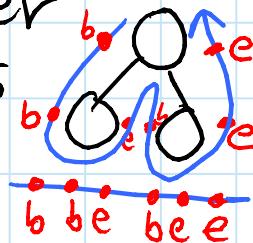
- $O(1)$ worst-case [Brodal - NJC 1996] [Driscoll ...]
- $O(\lg(\#_{\text{changes}}))$ reading slowdown for p unbounded

Full persistence: ditto [Driscoll et al. 1989]

- linearize tree of versions via in-order traversal, marking begin & end times of each version
- store begin & end times in order-maintenance DS: [L21: Dietz & Sleator - STOC 1987]
 - insert time before/after specified time (like linked list)
 - does time s precede time t ?
(\Rightarrow is version v an ancestor of v' ?)
in $O(1)$ time/op.
- \Rightarrow can tell which mods. apply to desired version
- when node is full, split into two nodes each roughly half full: (like B-tree node)
then recursively update pointers & back pointers to this node
- allow up to $2(p+c+1)$ mods. ($c = \# \text{ptrs.}/\text{node}$)
 \Rightarrow even if half full ($p+c+1$) & all c ptrs. move & all p back ptrs. move, still not full
- $\Phi = \# \text{ full nodes} \Rightarrow O(1)$ amortized cost

OPEN: $O(1)$ worst case?

- $O(\lg \lg n)$ fully persistent arrays \Rightarrow RAM DS [Dietz - WADS 1989]
- matching lower bound [Patrascu - unpub. 2008]
- OPEN: what about partial persistence?



Confluent persistence:

- functional DSs [Okasaki - book 2003]
 - e.g. deques with concat. in $O(1)$ /op.
 - double-ended queues [Kaplan, Okasaki, Tarjan - SICOMP 2000]
- logarithmic separation from pointer-machine DS [Pippenger - TPLS 1997]
- general transformation: [Fiat & Kaplan - J.Alg. 2003]
 - $d(v)$ = depth of version v in version DAG
 - $e(v) = 1 + \lg(\# \text{ paths from root to } v)$
 - overhead: $\lg(\#\text{updates}) + \max_v e(v)$ time & space
 - poor when $e(v) \sim 2^{\#\text{updates}}$ e.g.:
 - can make exponential-size version in this way
 - ⇒ still exponentially better than naive
 - $\max_v e(v)$ lower bound ASSUMING all nodes addressable at all times ~ UNREASONABLE: normally have to navigate to nodes
 - tries with $O(1)$ fingers, local nav. & subtree copy/delete [Demaine, Langerman, Price - Algorithmica]

method

path copying

1. functional

1. confluent

2. functional

2. confluent

finger move

time space

$\lg \Delta$ \emptyset

$\lg \Delta$ $\lg \Delta$

$\lg \lg \Delta$ $\lg \lg \Delta$

$\lg \Delta$ \emptyset

$\lg \lg \Delta$ \emptyset

modification

(time = space)

depth

$\lg \Delta$

$\lg \lg \Delta$

$\lg n$

$\lg n$

} local mods.

} cheap

} globally balanced

OPEN: better transformation with $O(1)$ fingers?
maintained to present
separations?
functional transformations?

OPEN: lists with split & concatenate?

OPEN: arrays with e.g. copy & paste?

Retroactivity: [Demaine, Iacono, Langerman - T.Alg. 2007]

- traditional DS formed by sequence of updates
- allow changes to that sequence
- maintain linear timeline
- ops:
 - Insert($t, "op(\dots)"$): retroactively do op() at time t
 - Delete(t): retroactively undo op. at time t
 - Query($t, "op(\dots)"$): execute query at time t
- partial retroactivity: Query only in present (last t)
- full retroactivity: Query at any time

Timecop, Back to the Future

round-trip
model

Easy cases:

- commutative updates: $x, y \equiv y, x$
 $\Rightarrow \text{Insert}(t, x) \equiv x \text{ in present}$
- invertible updates: $x, x^{-1} \equiv \emptyset$
 $\Rightarrow \text{Delete}(t) \equiv x^{-1} \text{ in present}$
- e.g. hashing, or array with $A[i] += \Delta$
 \Rightarrow partial retroactivity easy
- search problem: maintain set S of objects
subject to $\text{query}(x, S)$ for object x } comm. & invertible
& insert/delete objects }
- decomposable search problem: [Bentley & Saxe - J.A. 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: [Demaine et al. 2003]

- rollback method: retro. op. r time units in past with factor- r overhead via logging ("undo persistence") movie Retroactive
- lower bound: $\Omega(r)$ overhead can be necessary
 - DS maintains two values X & Y , initially \emptyset
 - $\text{set } X(x)$: $X \leftarrow x$
 - $\text{add } Y(\Delta)$: $Y \leftarrow Y + \Delta$
 - $\text{mul } XY()$: $Y \leftarrow X \cdot Y$
 - $\text{query}()$: return Y
 - $O(1)$ time/op. in "straight-line program" model
 - $\text{add } Y(a_n), \text{mul } XY(), \text{add } Y(a_{n-1}), \text{mul } XY(), \dots, \text{add } Y(a_0)$
computes poly. $a_n x^n + a_{n-1} x^{n-1} + \dots + a_0$ [Cramer's rule]
 - $\text{Insert}(t = \emptyset, \text{"set } X(x) \text{"})$ 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

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

- cell-probe lower bound: $\Omega(\sqrt{r} / \lg r')$
 - DS maintains n words; arithmetic updates $+, \cdot$
 - compute FFT using $O(n \lg n)$ ops.
 - changing w_i requires $\Omega(\sqrt{n})$ cell probes

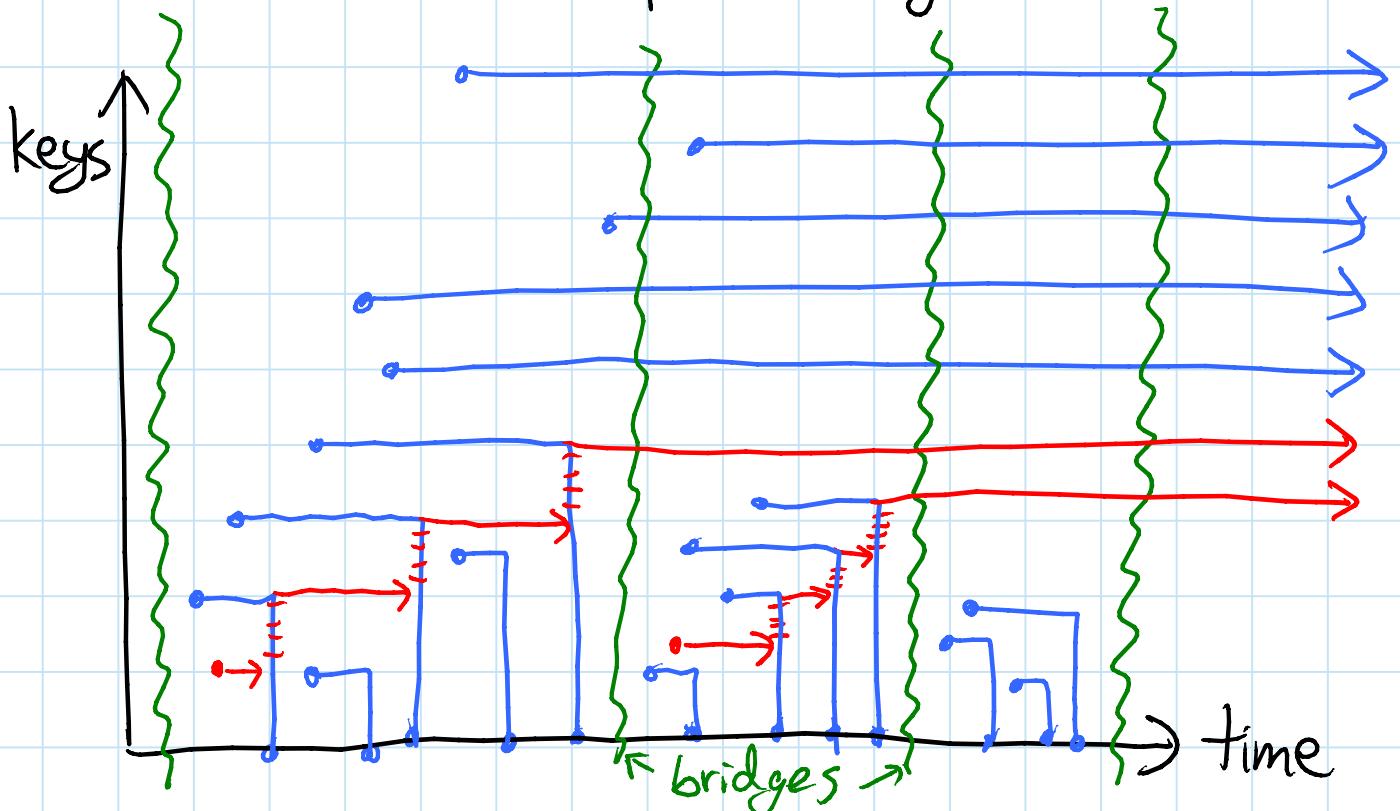
[Frandsen et al. 2001]

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

Priority queues: [Demaine et al. 2003]

partial retroactivity in $O(\lg n)$ /op.

- assume keys inserted only once
- L view: insert = rightward ray
delete-min = upward ray



- $\text{Insert}(t, \text{"insert}(k)")$ inserts into Q_{now}
 $\max \{ k, k' \in Q_{\text{now}} \mid k' \underline{\text{deleted}} \text{ at time } \geq t \}$
hard to maintain
- bridge at time t if $Q_t \subseteq Q_{\text{now}}$
- if t' is the bridge preceding time t
 then $\max \{ k' \mid k' \underline{\text{deleted}} \text{ at time } \geq t \}$
 $= \max \{ k' \in Q_{\text{now}} \mid k' \underline{\text{inserted}} \text{ at time } \geq t' \}$

- store Q_{now} as balanced BST; one change/update
- store balanced BST on leaves = insertions, ordered by time, augmented with
 $\forall \text{node } x: \max\{k' \in Q_{\text{now}} \mid k' \text{ inserted in } x\text{'s subtree}\}$
- store balanced BST on leaves = updates, ordered by time, augmented with
 - 0 for $\text{insert}(k)$ with $k \in Q_{\text{now}}$
 - $+1$ for $\text{insert}(k)$ with $k \notin Q_{\text{now}}$
 - -1 for delete-min

& subtree sums

\Rightarrow bridge = prefix summing to \emptyset

\Rightarrow can find preceding bridge, change to Q_{now} in $O(\lg n)$ time

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 OPEN: better?
- successor: $O(\lg m)$ partial trivial
 $O(\lg^2 m)$ full easy
 $O(\lg m)$ full [Giora & Kaplan - T.Als. 2009]

(\Rightarrow optimal dynamic vertical

ray shooting among horizontal

line segments) OPEN: general?

