Partial & full persistence: review

Example: AVL tree + partial persistence
\( p = 1 \) (no parent pointers)
\[ \Rightarrow \leq 2 \text{ mods./node} \]

**v0:**

**v1:** insert 9

**v2&3:** update heights

**v4&5&6:** rotate
v6': overflow 7

v6'': update pointers

v7: update heights

v7': overflow 5 & update pointers
Node splitting: full persistence
- overflowing node represents various versions
- look at linearized versions
- split represented versions in half linearly
- second half is closed under descendants \( \Rightarrow \) put in new node

(Can't split out single subtree of \( \frac{1}{2} \) or \( \frac{1}{3} \) the size)
Potential analysis: full persistence
- \( \leq 2(d+p+1) \) mods. per node
  - out-degree in-degree
- potential \( \Phi = c \cdot \sum_{\text{node}} (d+p+1) - \min \{d+p+1, \# \text{empty mod. slots}\} \)
  - \( = \sum_{\text{node}} \# \text{used mods. in second half of node} \)
- each update \( i \) has:
  - actual cost \( t_i = c \cdot (1 + \# \text{overflows}) \)
    - orig. field node splits
    - potential change \( \Delta \Phi_i = \Phi_i - \Phi_{i-1} \)
      - \( = -c \cdot \# \text{overflows} \cdot \# \text{emptied 2\text{-}nd half slots} \)
      - \( d+p+1 \)
      - \( + c \cdot \# \text{overflows} \cdot \# \text{pointers to "both" nodes} \)
- no \( 1+ \) for mod causing split, which turns into a \( d \) pointer
- key: pointers in \( d+p+1 \) mods (of both nodes) have reverse pointers that can be updated directly to just one of the nodes
  - amortized cost \( a_i = t_i + \Delta \Phi_i \)
    - \( = c(1+o) - c \cdot o \cdot (d+p+1) + c \cdot o \cdot (d+p) \)
    - \( = c = O(1) \)
- care about total cost \( \Sigma t_i \) \( \rightarrow \) telescoping sum
  - know \( 0 = \Sigma a_i = \Sigma (t_i + \Delta \Phi_i) = \Sigma t_i + \Phi_m - \Phi_0 \)
  - \( \Rightarrow \Sigma t_i = \Phi_0 - \Phi_m \leq \# \text{initial nodes} \cdot (d+p+1) \)
  - \( \leq 0 \)
⇒ \( O(1) \) amortized cost per update
Partially retroactive priority queue: review

- ordered by key: \( Q_{\text{now}} \) as balanced BST
- ordered by time:
  - BBST where leaves = updates \((\text{insert}+\text{delete})\)
  - leaf stores \( \emptyset \) for insert\((k)\) where \( k \in Q_{\text{now}} \)
  - \( +1 \) for insert\((k)\) where \( k \notin Q_{\text{now}} \)
  - \( -1 \) for delete-min

\( \Rightarrow \) prefix sum \( \emptyset \) means all to-be-deleted items have been deleted \( \equiv \text{BRIDGE} \)

- each node stores subtree sum & min & max prefix sum within subtree

\( \Rightarrow \) can find bridge preceding given leaf \( x \):
  - compute prefix sum for \( x \):
    - \( \leq \) left subtrees of path to root
  - look for subtree whose prefix sum range hits \( \emptyset \) when adding previous \( \Delta s \)
  - walk down \( \Rightarrow \) & Delete\((t, \text{"delete-min"})\)

- Insert\((t, \text{"insert}(k)\text{"") inserts into \( Q_{\text{now}} \)
  - max key deleted after \( t \) (or \( k \) if larger)
  = max key \& \( Q_{\text{now}} \) inserted after last bridge

\( \Rightarrow \) store in each node the max key \& \( Q_{\text{now}} \) inserted in the subtree \( \Rightarrow \) & Insert\((t, \text{"delete-min"})\)

- Delete\((t, \text{"insert}(k)\text{"") deletes from \( Q_{\text{now}} \)
  - min key \( \in Q_{\text{now}} \) inserted before next bridge \( > t \)

\( \Rightarrow \) successor bridge + min insert augmentation
Problem: transform any partially retroactive DS
- retro. updates in $O(m)$
- present queries in $O(n)$
into a fully retroactive DS with $O(\sqrt{m})$ overhead:
- retro. updates in $O(\sqrt{m} \cdot U(m))$
- retro. queries in $O(\sqrt{m} \cdot U(m) + O(n_t))$