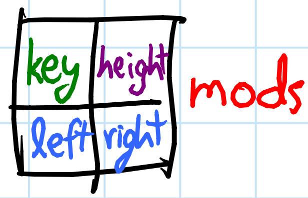
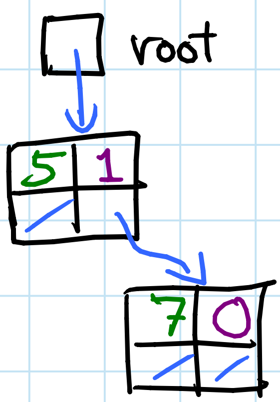
6.851	Class 2	Feb. 12, 2014

Partial & full persistence: review

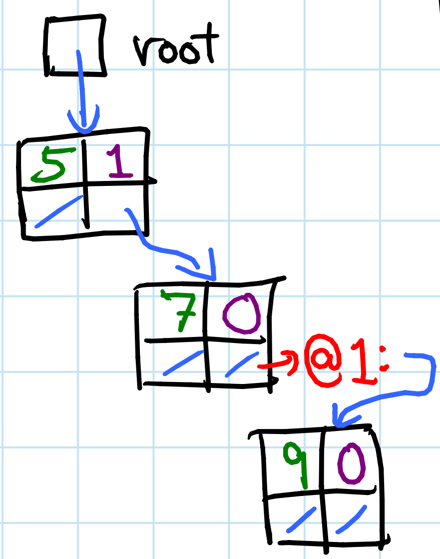
Example: AVL tree + partial persistence
$p = 1$ (no parent pointers)
$\Rightarrow \leq 2$ mods./node



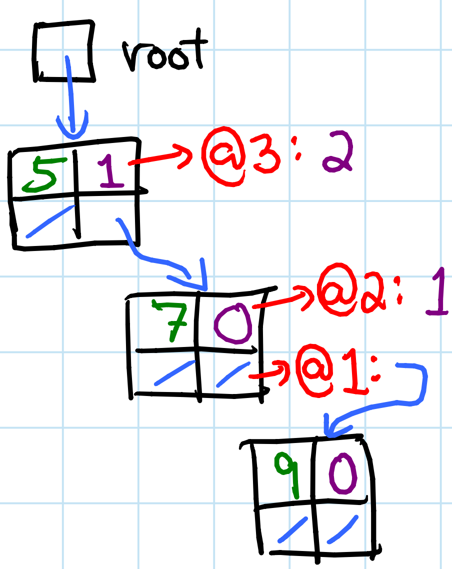
v∅:



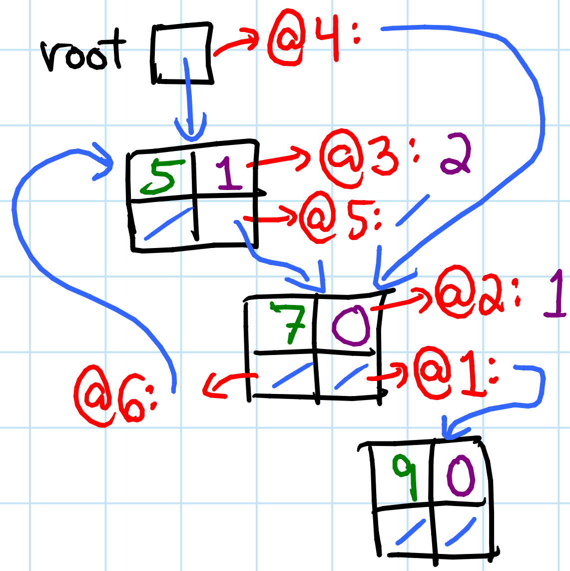
v1: insert 9



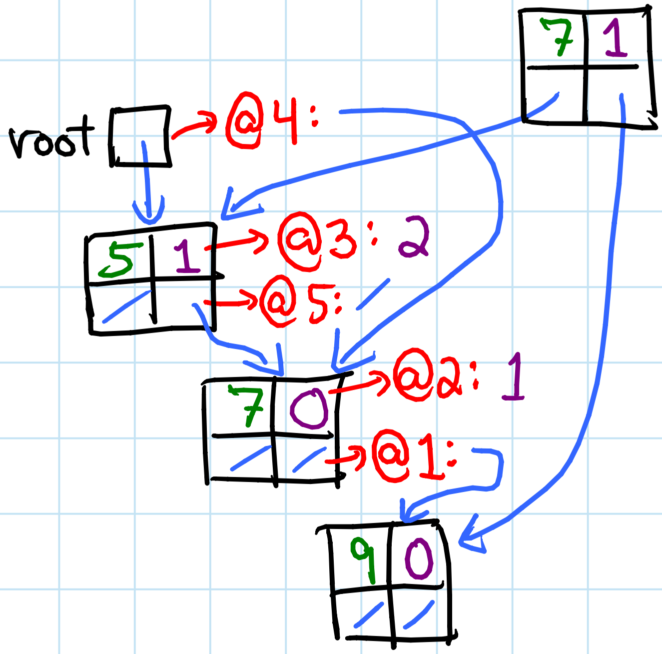
v2&3: update heights



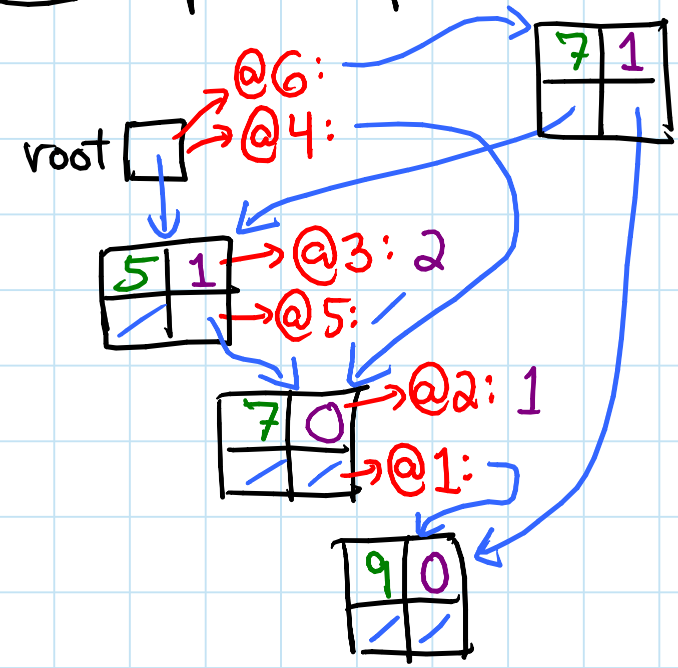
v4&5&6: rotate



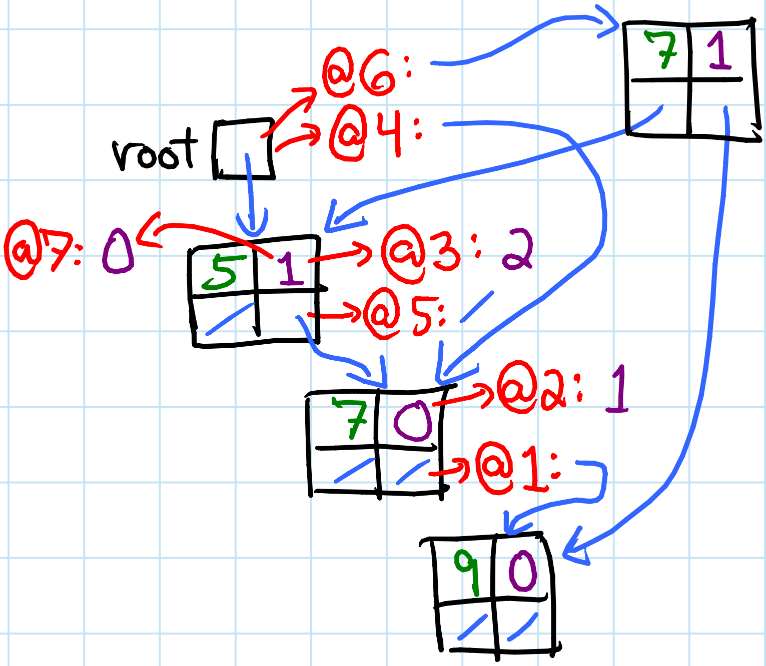
v6': overflow 7



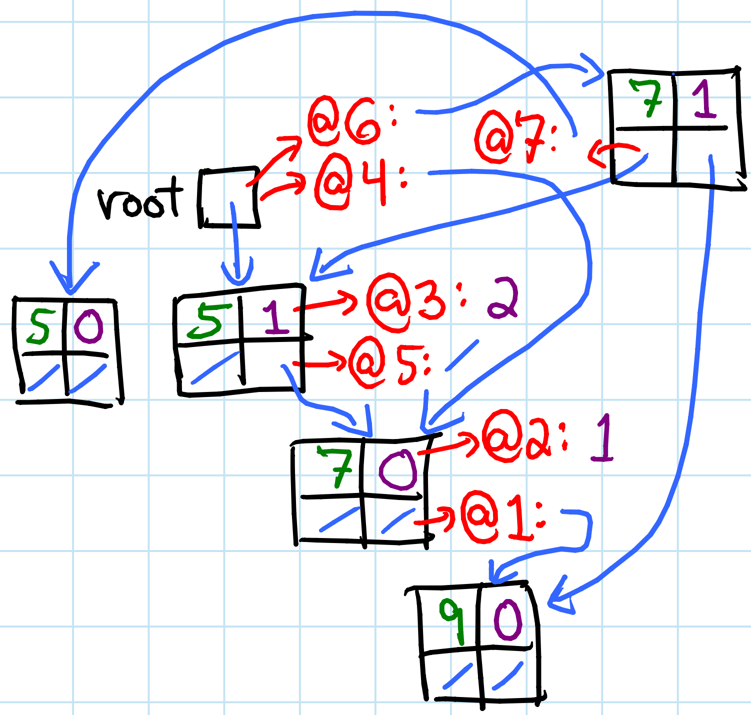
v6'': update pointers



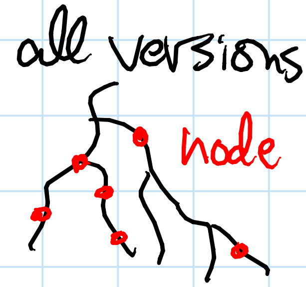
v7: update heights



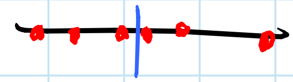
v7': overflow 5
& update pointers



Node splitting: full persistence

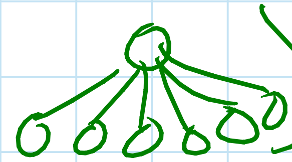


- overflowing node represents various versions
- look at linearized versions
- split represented versions in half linearly



- second half is closed under descendants ⇒ put in new node

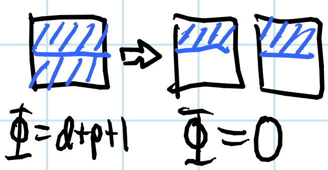
(Can't split out single subtree of 1/2 or 1/3 the size:



)

Potential analysis: full persistence

- $\le 2(\underbrace{d}_{\text{out-degree}}+\underbrace{p}_{\text{in-degree}}+1)$ mods. per node
- potential $\Phi = c\cdot\sum_{\text{node}}\left((d+p+1) - \min\{d+p+1, \#\text{empty mod. slots}\}\right)$
  $= \sum_{\text{node}}(\#\text{used mods. in second half of node})$
- each update i has:
  - <u>actual cost</u> $t_i = c\cdot(\underbrace{1}_{\text{orig. field}} + \underbrace{\#\text{overflows}}_{\text{node splits}})$
  - <u>potential change</u> $\Delta\Phi_i = \Phi_i - \Phi_{i-1}$



$$= -c\cdot\#\text{overflows}\cdot\underbrace{\#\text{emptied 2nd-half slots}}_{d+p+1}$$
$$+\, c\cdot\#\text{overflows}\cdot\underbrace{\#\text{pointers to "both" nodes}}_{\underbrace{d}_{\text{forward}\Rightarrow\text{back}}+\underbrace{p}_{\text{back}}}$$

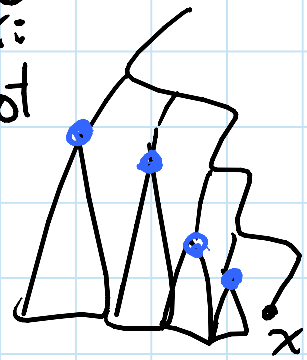
no 1+ for mod causing split, which turns into a d pointer

  - <u>key</u>: pointers in $d+p+1$ mods (of both nodes) have reverse pointers that can be updated directly to just one of the nodes
  - <u>amortized cost</u> $a_i = t_i + \Delta\Phi_i$
    $= c(1\cancel{+o}) - c\cdot o\cdot(\cancel{d+p}+\cancel{1}) + \cancel{c\cdot o\cdot(d+p)}$
    $= c = O(1)$
- care about <u>total cost</u> $\sum t_i$ → telescoping sum
- know $0 = \sum a_i = \sum(t_i + \Delta\Phi_i) = \sum t_i + \Phi_m - \Phi_0$
  $\Rightarrow \sum t_i = \Phi_0 \underbrace{- \Phi_m}_{\le 0} \le \#\text{initial nodes}\cdot(d+p+1)$

$\Rightarrow O(1)$ amortized cost per update $\square$

Partially retroactive priority queue: review

- ordered by key: Q_{now} as balanced BST
- ordered by time:
  - BBST where leaves = updates (insert+delete)
    - leaf stores $\emptyset$ for insert(k) where $k \in Q_{now}$
      $+1$ for insert(k) where $k \notin Q_{now}$
      -1 for delete-min
  - $\Rightarrow$ prefix sum 0 means all to-be-deleted items have been deleted $\equiv$ 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: Σ left subtrees of path to root
    - look for subtree whose prefix sum range hits $\emptyset$ when adding previous Δs
    - walk down



- Insert(t, "insert(k)") & Delete(t, "delete-min") inserts into Q_{now}
  - max key deleted after t (or k if larger)
  - = max key $\notin Q_{now}$ inserted after last bridge
  - $\Rightarrow$ store in each node the max key $\notin Q_{now}$ inserted in the subtree
- Delete(t, "insert(k)") & Insert(t, "delete-min") deletes from Q_{now}
  - min key $\in Q_{now}$ inserted before next bridge $> t$
  - $\Rightarrow$ successor bridge + min insert augmentation

Problem: transform any partially retroactive DS

- retro. updates in $U(m)$
- present queries in $Q(n)$

into a fully retroactive DS with $O(\sqrt{m})$ overhead:

- retro. updates in $O(\sqrt{m}\,U(m))$
- retro. queries in $O(\sqrt{m}\,U(m) + Q(n_t))$