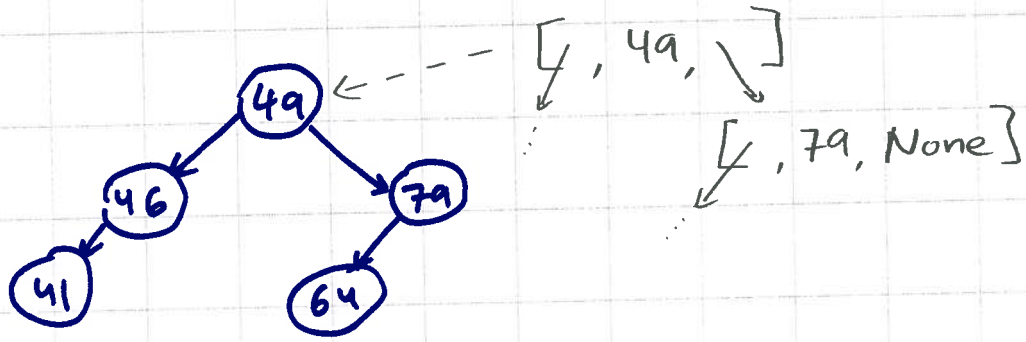
6.006 Lecture 4: BST's & Balanced BSTs

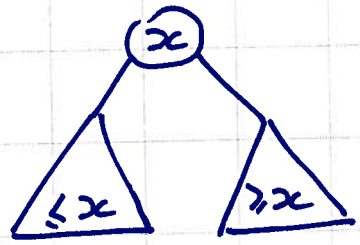
- BST's & BST operations in $\Theta(h)$ time
- Balance $\Rightarrow$ height $= \Theta(\lg n)$
- AVL Trees

$\rightarrow$ CLRS 13.1 & 13.2, but Red-Black trees, not AVL trees

Binary Search Trees



BST property:

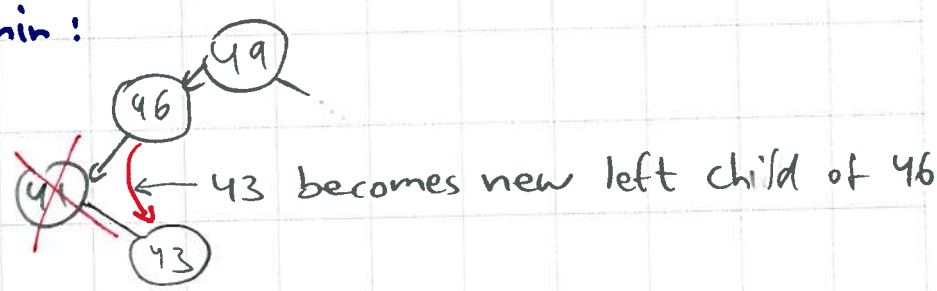


BST operations start at the root and only visit vertices along a path to a leaf

find

findmax (min): go right (left) until can't

delete min:



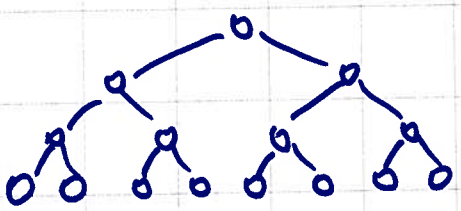
next-larger: a bit trikier

delete: even more :-)

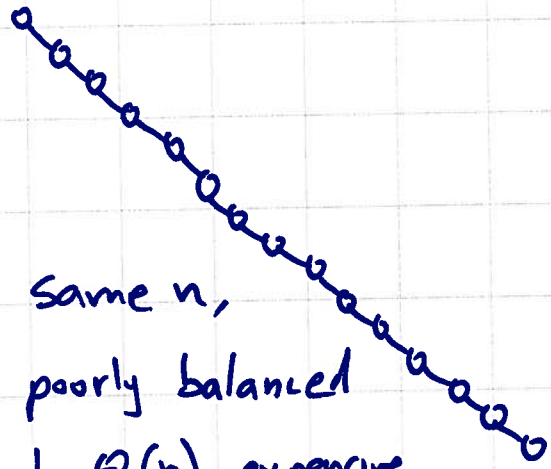
rank: easy if we augment nodes with size of subtree

Balanced Trees

operations cost bounded by path from root to a leaf; how long can the path be? (Height of tree is defined as longest such path).



perfectly balanced
$h = \Theta(\lg n)$



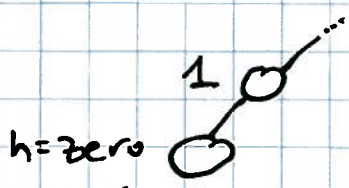
same n,
poorly balanced
$h = \Theta(n)$, expensive operations

Balanced BST Strategy

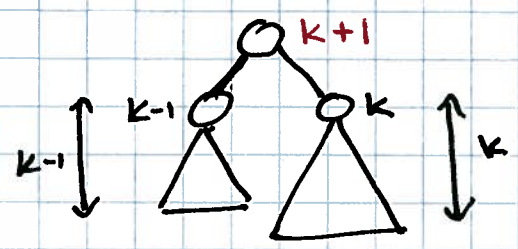
- Augment every node with some property
- Define a local invariant on property
- Show (prove) that invariant guarantees $\Theta(\lg n)$ heigh
- Design algorithms to maintain property & to fix up the invariant.

AVL trees (Adel'son-Vel'skii-Landis 1962)

- Property: height, # edged to most distant leaf
- invariant: heights of left & right children differ by at most ± 1
  (define height of None as -1)



(None rule)



- Thm: Height of AVL tree $\leq 2\lg n$

Proof: Let N_h be min # nodes in height-h AVL t.

$$N_h = N_{h-2} + N_{h-1} + 1 \quad \text{(picture above)}$$

$$> 2N_{h-2}$$

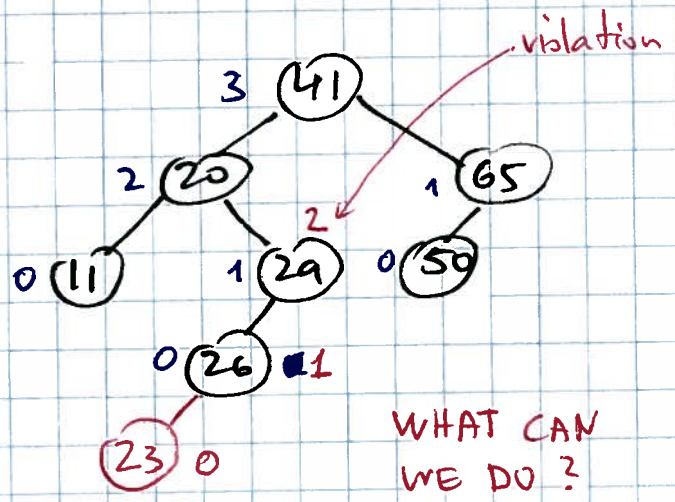
$$N_0 = 1 \Rightarrow N_h > 2^{h/2} \Rightarrow \frac{h}{2} < \lg N_h \Rightarrow h < 2\lg N_h$$

For a given tree with height h and N nodes,

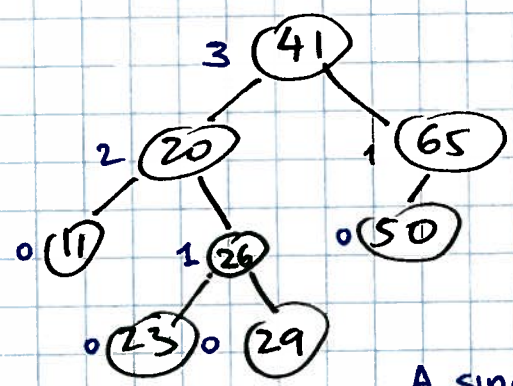
$N > N_h$ so $h < 2\lg N_h < 2\lg N$

- When you insert (delete, etc), the invariant may get violated.

Example: Insert(23)

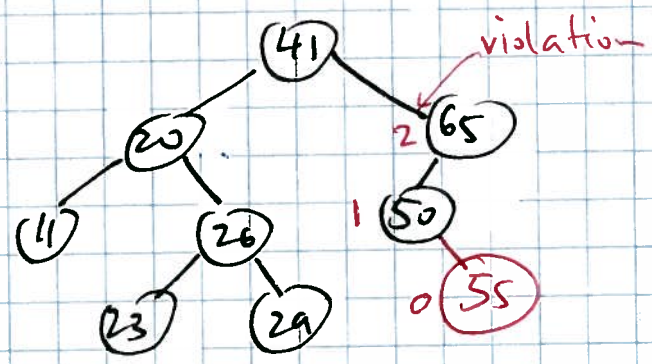


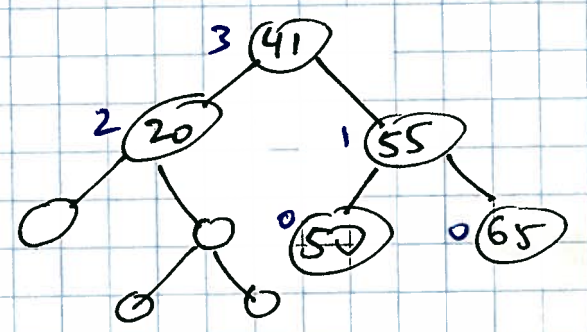
WHAT CAN WE DO ?



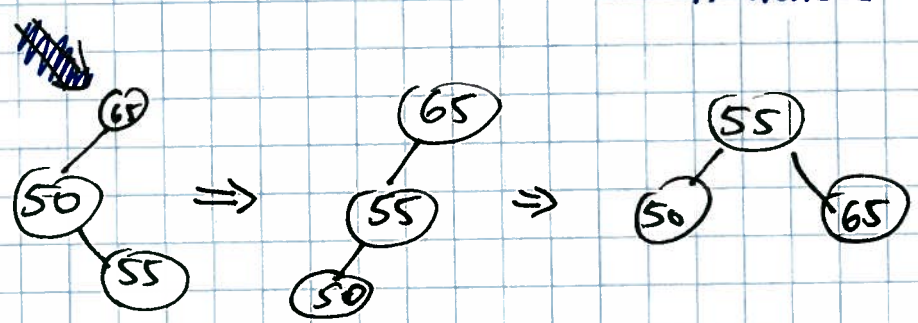
A single rotation on 29

Example: Insert(55)



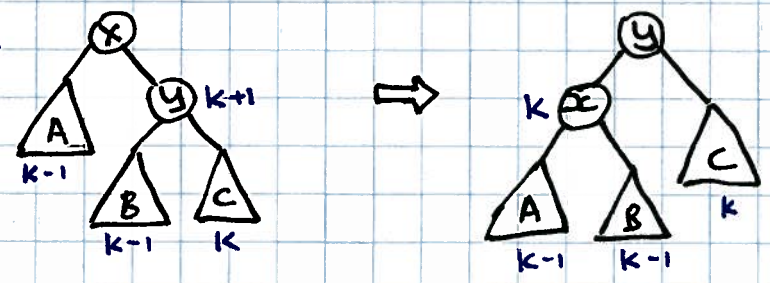


A double rotation

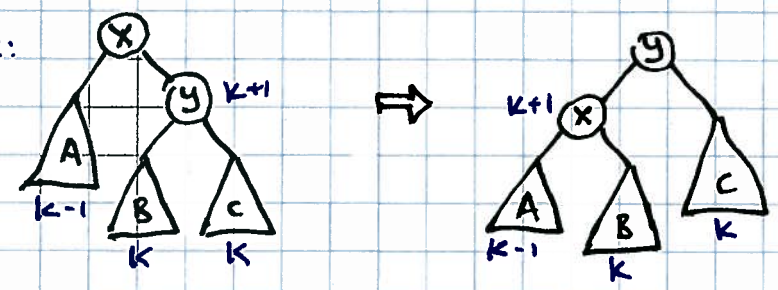


- General Rotation Rules

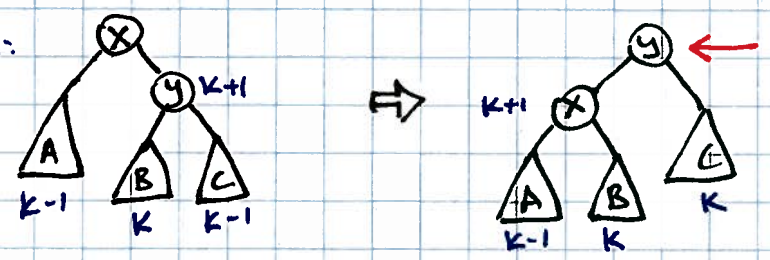
Case 1:



Case 2:

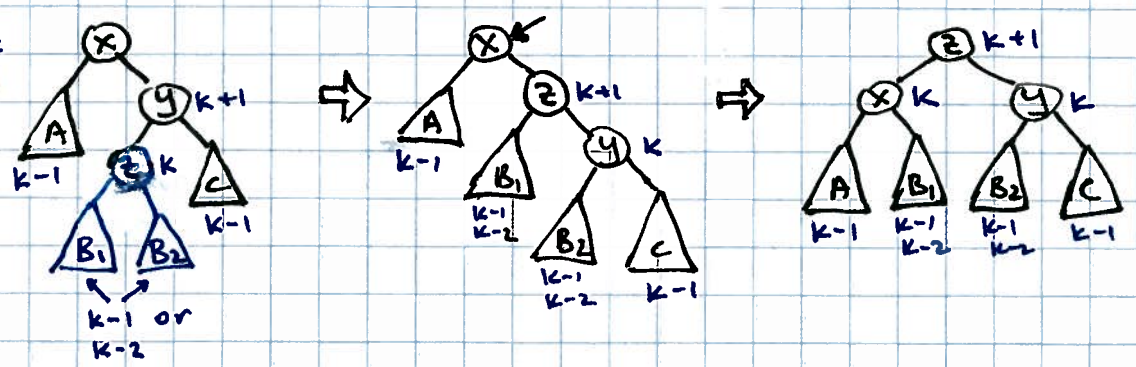


Case 3:



← Violates our property; not a useful rotation on this tree

Case 3: (again)



- Here we assumed that right subtree of x is higher other case is symmetric
- We also assumed that x is off-balance by 1
- During insertions, this is always the case & we can fix up the balance going up the tree in $\Theta(\lg n)$ time.

can also do delete, deletemin etc in $\Theta(\lg n)$ time

- In general, you can use rotations to transform any shape tree to any other shape using rotations.

Other search trees

- AVL trees
- 2-3 trees ⇒ B-trees — balanced but not binary; used in databases
- weight-balanced trees a.k.a. BB[α]
- red-black trees — nodes augmented with just 1 bit! CLRS ch. 13
- skip lists — randomized; fast with high probability
- Splay trees — amortized analysis; individual ops can be expensive, but in a sequence of ops the average cost is always low.
- scapegoat trees
  Rivest & Galperin
  amortized insert, delete

Why did people invent so many trees?

- B-trees . very shallow, very little random access
- BB[α] : tune insert/delete vs search
- red-black: only 1 extra bit per node
- splay, scapegoat : no extra data in nodes, lower constants, but only amortized guarantees

(Insertions into lists in Python take constant amortized time; once in a while, the list is copied to a larger array to make room for more insertions; when do you trigger this?)

- van Emde Boas trees: $O(\lg \lg u)$ operations for integer key $1 \ldots u$