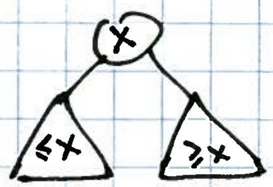
6.006 Lecture 4: Balanced BSTs

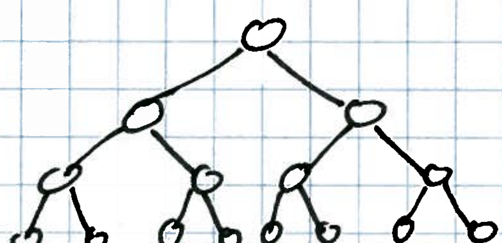
- Balance $\Rightarrow h = \Theta(\lg n) \Rightarrow \Theta(\lg n)$ operations
- AVL trees: definition, ~~balance, height, insert, rotati~~
  height
  rotations, insert
- "Big picture" on data structures
- CLRS 13.1 & 13.2, (red-black trees, not AVL trees)

Review

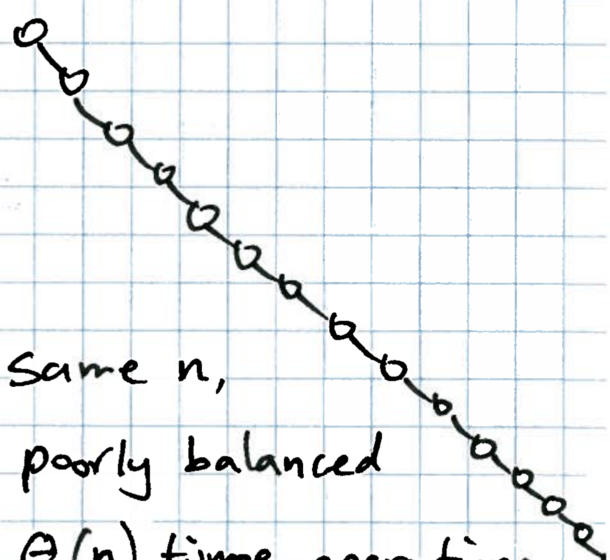
- BST property



- Operations (find, insert, delete, ...) take $\Theta(h)$ time where h is the height of the tree (# edges downward to a leaf).



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



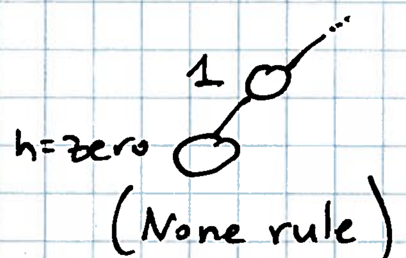
same n,
poorly balanced
$\Theta(n)$ time operations

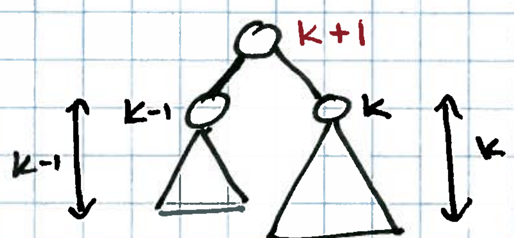
Balanced BST Strategy

- Augment every node with some property
- Define a local invariant on property
- Show (prove) that invariant guarantees ~~to~~ $\Theta(\lg n)$ 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)





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

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

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

$> 2 N_{h-2}$

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

For a given tree with height h and N nodes,

$N \geq N_h$ so $h < 2 \lg N_h \leq 2 \lg N$

- A tighter analysis (side note)

h	0	1	2	3	4	5	6	7	8	9	10
	boundary cond.										
$1+N_{h-2}+N_{h-1}=N_h$	$\boxed{1}$	$\boxed{2}$	4	7	12	20	33				
$F_{h-2}+F_{h-1}=F_h$		$\boxed{1}$	$\boxed{1}$	2	3	5	8	13	21	34	

$$N_h = F_{h+3} - 1$$

$$F_h \approx \frac{\varphi^h}{\sqrt{5}} \qquad \varphi = \frac{1+\sqrt{5}}{2} \approx 1.618 \quad \text{golden ratio}$$

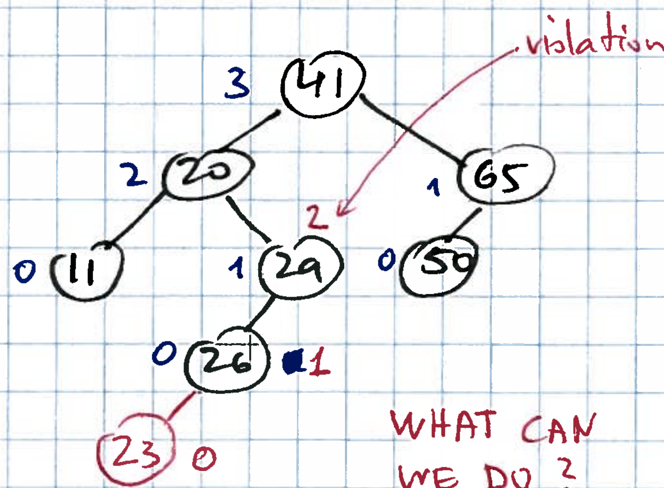
$$h \approx \log_\varphi F_h + \log_\varphi \sqrt{5}$$

$$h+3 \approx \log_\varphi (N+1) \pm \text{const}$$

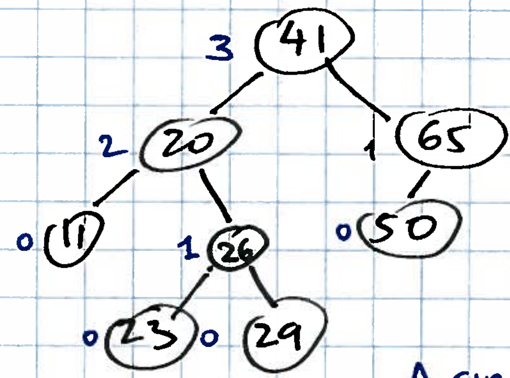
$$h \approx \log_\varphi (N+1) \pm \text{const} \approx \frac{\lg(N)}{\lg(\varphi)} \pm \text{const} \approx 1.44 \lg(N) \pm \text{const}$$

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

Example: Insert(23)

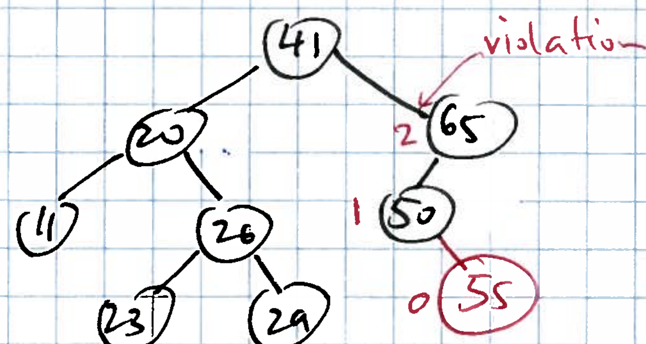


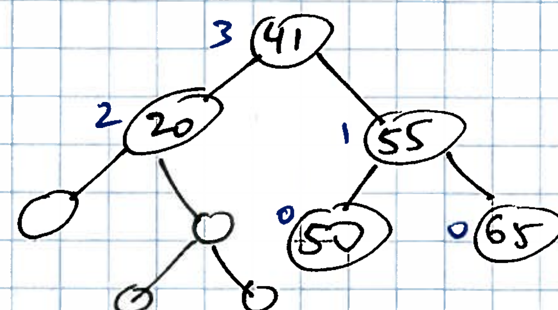
WHAT CAN WE DO?



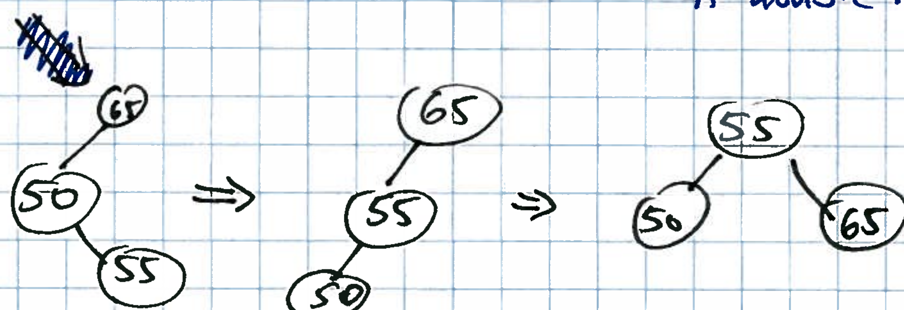
A single rotation on 29

Example: Insert(55)



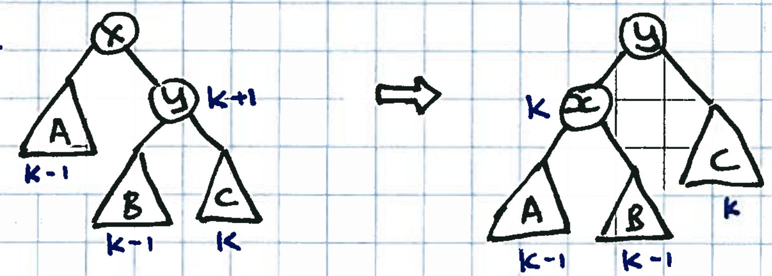


A double rotation

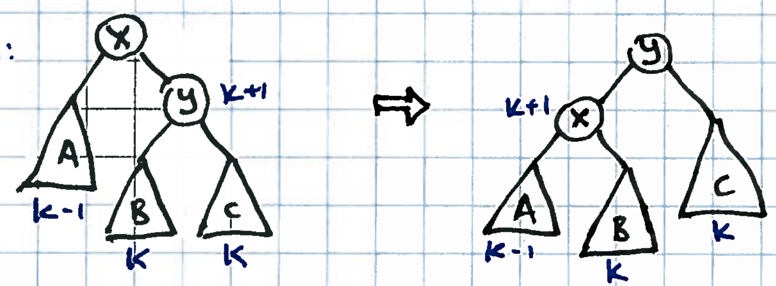


- General Rotation Rules

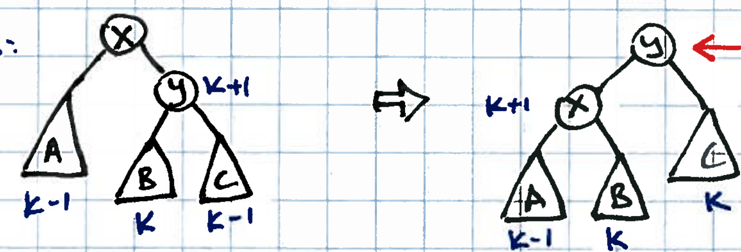
Case 1:



Case 2:

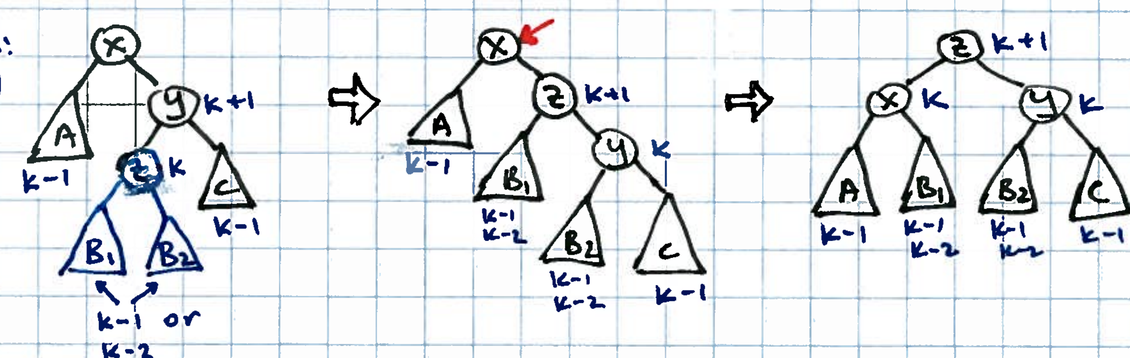


Case 3:



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 $1 \ldots u$

- The big picture: Abstract Data Types (ADT)

- Data-structure problem defines what operations to support

e.g.: Priority Queue

- create empty queue
- insert(x)
- retrieve & delete minimum element

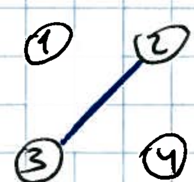
is this good enough for runway reservation? almost but not quite.

- Choice of DS driven by ADT requirements but not determined by them: BSTs are OK for priority queues but so are heaps.

- Another example: Union-Find

  - create a collection of sets $\{\{1\}, \{2\}, \dots \{n\}\}$
  - union(i, j) : merge the sets that contain i and j
  - find(i) : return a representative of the set containing i

init(4)
find(3) → 3
union(3,2)
find(3) → 2
find(2) → 2
⋮



- We specified the ADT completely but did not describe a DS; see CLRS ch. 21.