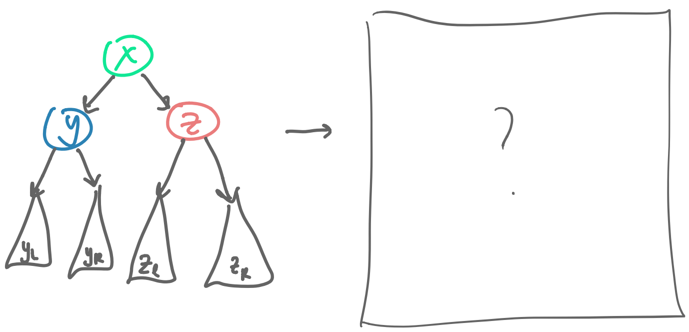
THE ROAD UP TO NOW...

- Recursive data structures (Linked list, Trees)
- Operations on data structures + Time complexity (Search for number in tree? $O(\log n)$)

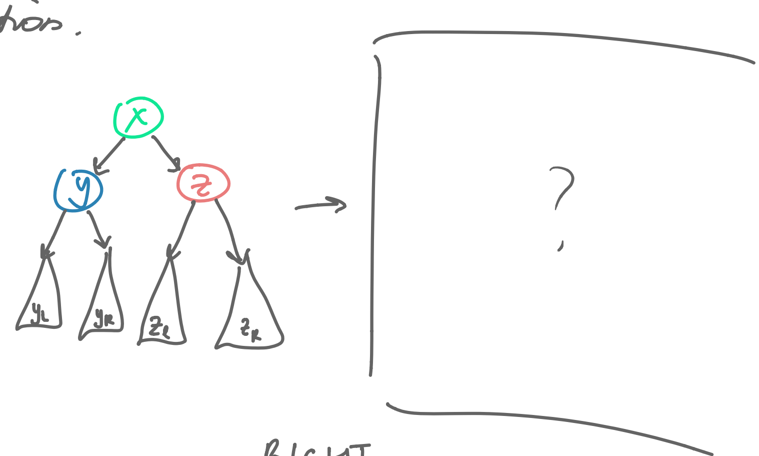
SOME BASIC PRIMITIVES: Rotation.

GOAL - 'reorganise' the tree whilst maintaining the search tree condition, FAST → $O(1)$

VARIANTS - Left rotation and right rotation.



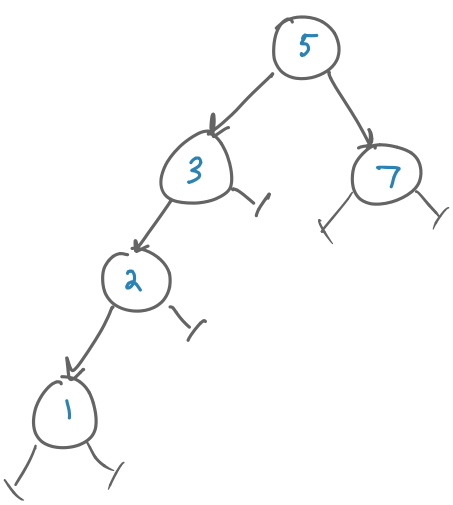
LEFT



RIGHT

Constant number of pointers change in each rotation ⇒ CONSTANT TIME.

USEFULNESS? → One side is taller than the other, rotate in opposition!



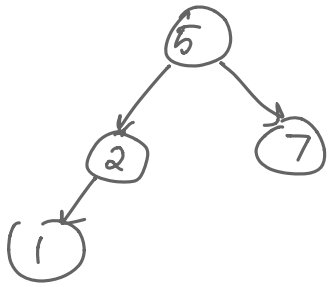
TASK: USING ONE rotation height balance the tree.

height balance: forall nodes. $|h_L - h_R| \le 1$

PRELIMINARY TO SPLAY TREES → INSERT AT ROOT.

YOU KNOW: INSERT AT LEAF

↳ Insert a new value into the tree as a leaf following BST condition.



INSERT AT ROOT:

① INSERT VALUE AT LEAF.

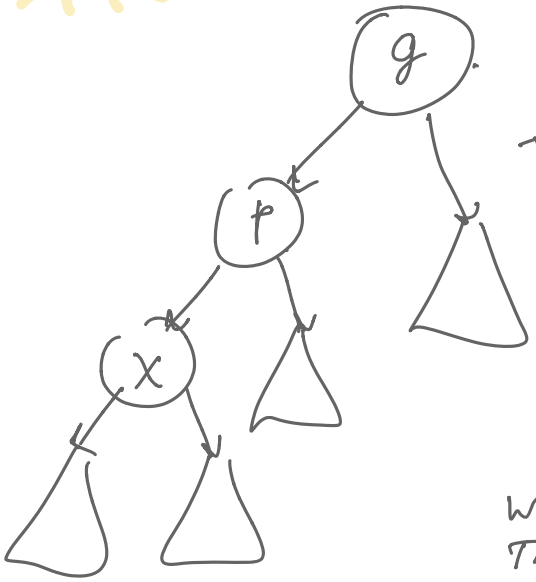
TASK: Simulate on insertion of 0.

② Rotate in opposition to what side you recursed on e.g. t->left = insert(v, t->left) ⇒ ROTATE RIGHT!

SPLAY TREES

Same insertion method as insertion at root.

EXCEPT!



THEN

① Rotate right on g node (rotates on g→p edge)

② Rotate right on p node (rotates on p→x edge)

What would an insertion at root do?

WE CALL THIS MODIFIED VERSION OF AMENDING THE TREE THE SPLAY OPERATION.

On insertion of a value → insert at leaf → perform SPLAY.

On finding a node → perform SPLAY to closest target.

vs.

- Locality "more likely" to search again.

- Suppose you search for nodes in ascending order. Complexity?

- $\Theta(\log n)$ "amortized" time.

AUGMENTING DATA STRUCTURES

- Adding some EXTRA data on an existing structure for efficient implementation of ops.
  e.g. DOUBLY LINKED LISTS.

The AVL TREE

Motivation - if we can keep the height of the tree 'h' bounded above by $\log|n|$ then subsequent operations on the tree proportional to 'h' are bounded by $\log|n| \Rightarrow$ WORST CASE $O(\log|n|)$ operations!

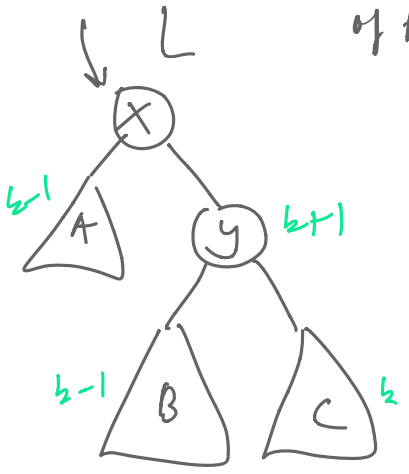
GOAL - Insert $O(\log n) \leftarrow O(h)$
Search $O(\log n) \leftarrow O(h)$

OBSERVATION - For all nodes in the tree
let h_l = height(node → left)
h_r = height(node → right)
Then maintain invariant:

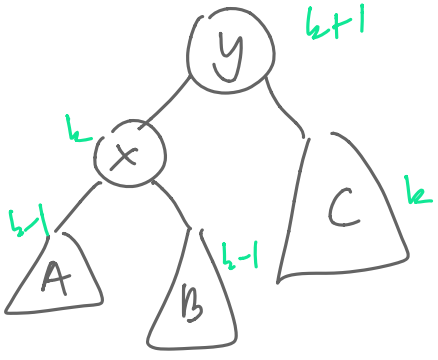
$$|h_l - h_r| \leq 1$$

consider tree

h - height of the tree.



rotateL(X)



① Let x be the node that first violates the AVL condition.

② Assume x is 'right heavy' (taller on the right)

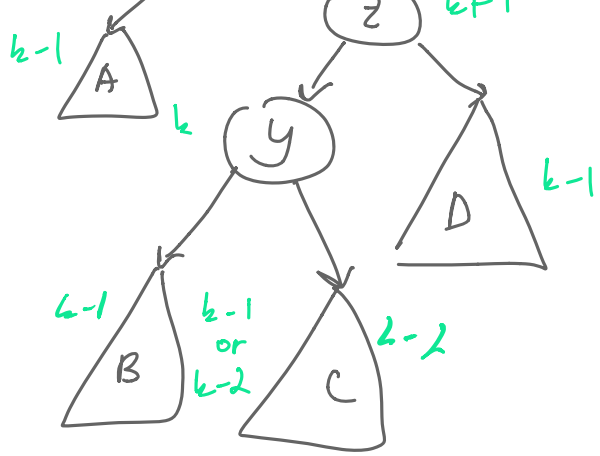
③ Perform Left rotation to counteract

④ Similar idea if x is 'left heavy' (symmetric)

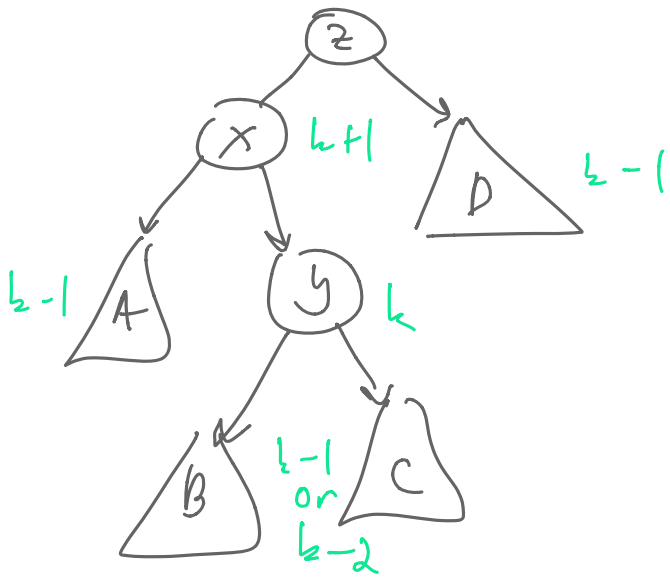
HANG ON...

Do we just rotate in opposition to the side that's taller?

→ most cases yes, but consider...



Ok... just rotate left... right?



Why did this happen?

Before we were right right now we're right left!

↳ rotate to be right right.

Notice before we rotate left, it would be good to rebalance the z→y edge. Here ① rotateR(z)

② rotateL(x)

OKAY, TWO DOWN ONE TO GO!

Ok... does this do anything?

worst case when every node has h_l and h_r differ by 1. Let N_H be number of nodes in height-h AVL tree.

$N_h = N_{h-1} + N_{h-2} + 1$ ← root node

root

N_{h-1} ← differ by 1. N_{h-2} ← differ by 1.

$> 2 N_{h-2}$ (solve the recurrence)

$N_H > 2^{h/2}$

$\log(N_H) > \log(2^{h/2})$

$2 \log(N_H) > h \Rightarrow h < 2\log(N_H)$ ← which is n!!!

$\frac{h < 2\log(N_H)}{\log(n)}$

Okay... Last balanced tree!

2-3-4 TREES

Motivation: Construct a weight balanced tree that performs well w.r.t. cache utilization.

Applications: Linux ext3/4 Filesystem, Database Implementation, Dis. FS (systems!)

(new document)