

Balanced Search Trees

- Resources available:

kevin-miao.com

Today

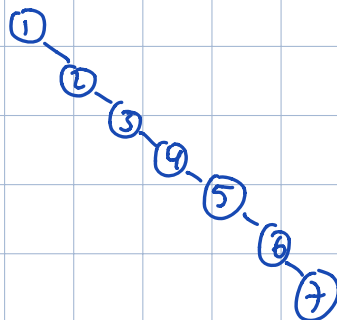
- Quiz Q2: Review
- Mini Review: Balanced Search Trees
 - 2-3-4 Trees
 - LLRB
 - * Rotations
 - * Insertion

mini Review

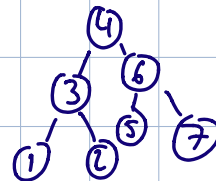
- Why balanced trees?

Efficiency at all times

add 1, 2, 3, 4, 5, 6, 7



vs



$O(\log n)$

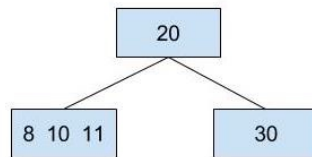
* 2-3-4 trees

* Can have 2, 3 or 4 children

Insertion * Go down like BST, add to leaf

* If invalid tree, pop up middle
repeat until valid

valid is when for all nodes, the $items \leq 3$



Variants (B-Trees)

→ 2-4

→ 2-3-4-5

→ 2-3 Trees

Red Black trees ↔ 2-3-4 Trees

* Middle Element ↔ black node reachable from parent

* left ↔ red left of black

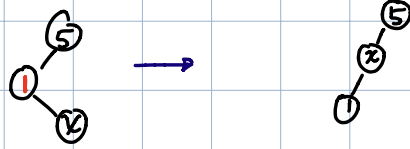
* right ↔ red right of black

left leaning red Black Trees ↔ 2-3 Trees

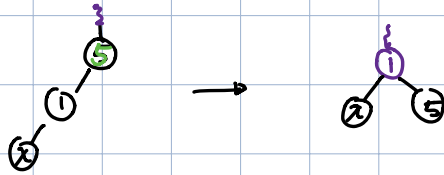
* left → red : left of black

Operations

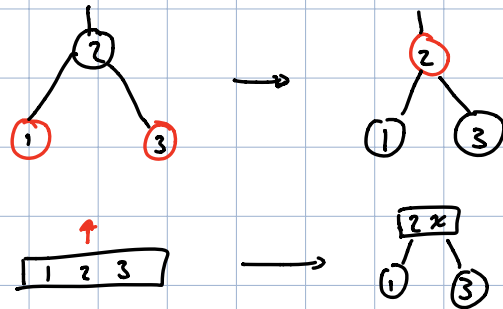
① Rotate Left (1)



② Rotate Right (5)

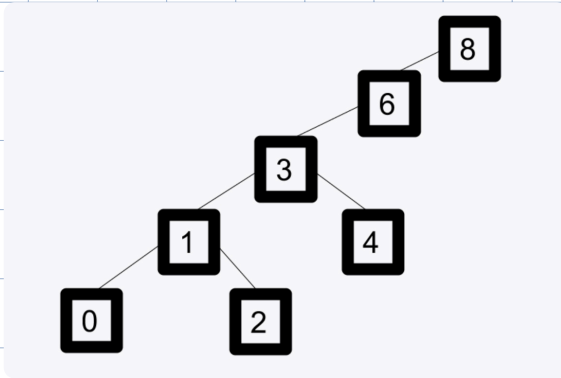


③ Flip Color



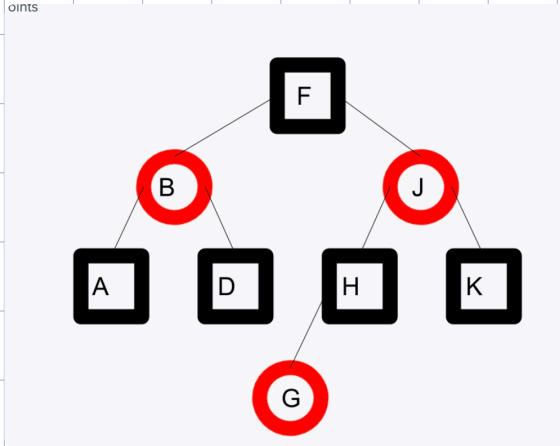
Quiz

Q3
a)



Which sequence of rotations will balance this tree? (Height = 3)

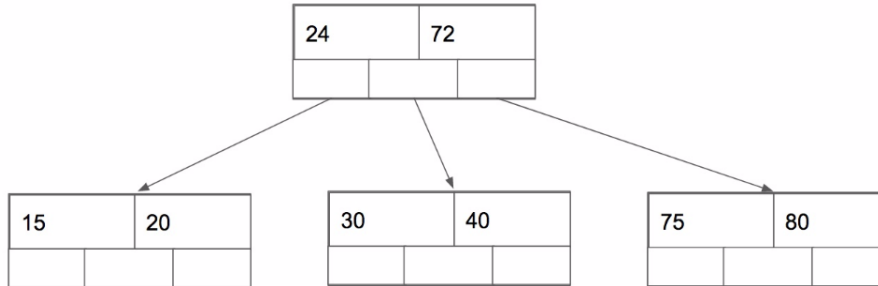
b)



Convert this tree to a 2-3-4 tree.

4 All about Trees

1. Why does a binary search tree have a worst case runtime of $\theta(n)$ for *contains*?
2. Give a sequence of operations, such that if they were inserted in the order they appear, would result in a "poor" binary search tree.
3. Examine this B-tree with order 3. Mark the paths taken when the user calls *contains*(40).



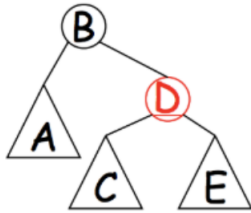
4. Now call *insert*(35), and draw the resulting tree.
5. What property of a B-tree rectifies problems of binary search trees, such as the one in 1.1? Why would you not use a B-tree?

5 The Holy LLRB Invariant

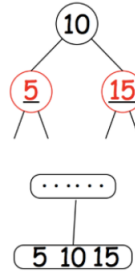
RB Tree Invariants: Node labels are in order from left to right. All paths through the tree contain the same number of black nodes. No red nodes have red parents. As a result, the height of a RB tree with n nodes is $O(\log n)$.

LLRB trees must also maintain the following invariant (in addition to the regular red-black invariant):

No right-leaning trees (black parent with right red child):



No "4-nodes" (black parent with two red children):



1. What are the "fixups" for the two cases above in order to preserve the LLRB invariant (i.e. what operations do we perform on each tree to ensure it is a proper LLRB)?

Consider the following RB tree:



2. Draw the tree after applying all necessary fixups to make it a proper LLRB tree.

3. Next, insert 10 into the tree, and apply all fixups to preserve the LLRB invariant.

4. Finally, draw the corresponding 2-3 tree.