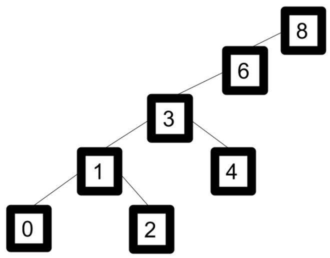
CS 61B Fall 2021

Balanced Search

Exam Prep Discussion 12: November 8, 2021

1 Balancing Trees

We are given the following extremely unbalanced search tree.



Select the minimum number of rotations in the correct order required to balance this tree. *Hint*: The resulting tree should have two layers of nodes below the root.

[ ] Rotate left on 8
[ ] Rotate right on 8
[ ] Rotate left on 6
[ ] Rotate right on 6
[ ] Rotate left on 4
[ ] Rotate right on 4
[ ] Rotate left on 3
[ ] Rotate right on 3
[ ] Rotate left on 2
[ ] Rotate right on 2
[ ] Rotate left on 1
[ ] Rotate right on 1
[ ] Rotate left on 0
[ ] Rotate right on 0

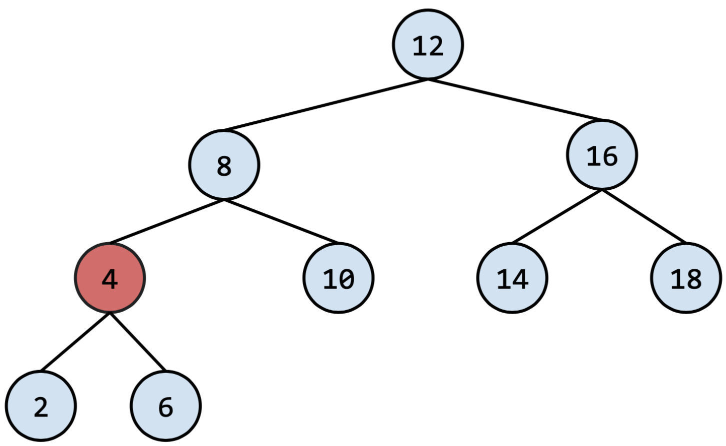
2 LLRBs

a) (2 Points). Perform the following insertions on the Left Leaning Red Black Tree (LLRB) given below. For each insertion, give the fix up operations needed. Recall a fix up operation is one of the following:

- `rotateLeft`
- `rotateRight`
- `colorFlip`
- change the root node to black.

Note that insertions are **dependent**. If only two operations are necessary, pick "None" for the third operation. If only one operation is necessary, pick "None" for the second and third operation. If no operations are necessary, pick "None" for all three operations.

If you put "None" for the "Operation applied", **leave the "Node to apply on" blank.** (Summer 2021 MT2)



i) (0.5 Points). Insert 17

	Operation applied	Node to apply on
1st operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ None	
2nd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ None	
3rd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ None	

ii) (0.5 Points). Insert 15. Note that insertions are dependent, so insert 15 into the state of the LLRB after the insertion of 17.

	Operation applied	Node to apply on
1st operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
2nd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
3rd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	

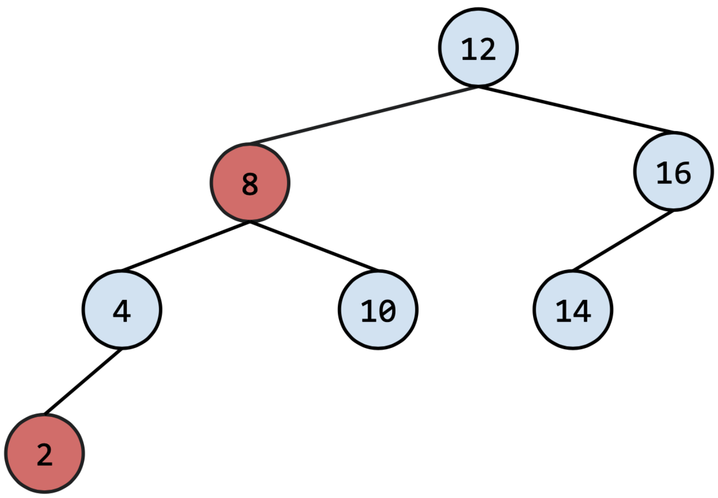
iii) (0.75 Points). Insert 13. Note that insertions are dependent, so insert 13 into the state of the LLRB after the insertion of 15.

	Operation applied	Node to apply on
1st operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
2nd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
3rd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	

iv) (0.75 Points). Insert 19. Note that insertions are dependent, so insert 19 into the state of the LLRB after the insertion of 13.

	Operation applied	Node to apply on
1st operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
2nd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	
3rd operation	◯ `rotateLeft()` ◯ `rotateRight()` ◯ `colorFlip()` ◯ change root to black ◯ `None`	

b) (1.5 Points). The tree below is **not** a valid LLRB (hint: to see why this is the case, draw the corresponding 2-3 tree) but it's close! In this part, we will try to *transform* it into a valid LLRB in two different ways. Note that each way acts **independently** of the previous. If a way isn't possible, put `impossible`. Recall that LLRBs **cannot** have duplicates.



i) (0.75 Points). Way 1: Remove a **single leaf** node from the tree. Which leaf node?

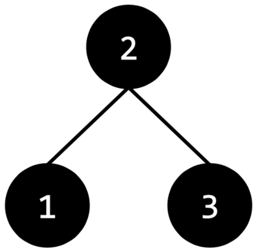
○ 2 ○ 4 ○ 8 ○ 10 ○ 12 ○ 14 ○ 16 ○ impossible

ii) (0.75 Points). Way 2: Flip the color of a **single** node. Which node?

○ 2 ○ 4 ○ 8 ○ 10 ○ 12 ○ 14 ○ 16 ○ impossible

3 Trees

The simple tree below can be a BST, 2-3 Tree, or even an LLRB!



a) (1 Point). Suppose it is a BST. Select all the insertion orderings that can produce the BST above. (Summer 2021 MT2)

☐ 1, 2, 3 ☐ 1, 3, 2 ☐ 2, 1, 3 ☐ 2, 3, 1 ☐ 3, 1, 2 ☐ 3, 2, 1
☐ None of the above

b) (1 Point). Now, suppose it is a 2-3 Tree. Select all the insertion orderings that can produce the 2-3 Tree above.

☐ 1, 2, 3 ☐ 1, 3, 2 ☐ 2, 1, 3 ☐ 2, 3, 1 ☐ 3, 1, 2 ☐ 3, 2, 1
☐ None of the above

c) (2.5 Points). Now, suppose it is an LLRB with only black nodes.
i) (0.75 Points). Select all the insertion orderings that can produce the LLRB above.

☐ 1, 2, 3 ☐ 1, 3, 2 ☐ 2, 1, 3 ☐ 2, 3, 1 ☐ 3, 1, 2 ☐ 3, 2, 1
☐ None of the above

ii (0.75 Points). Which insertion ordering requires the **minimum** number of `rotateLeft` and `rotateRight` calls. If multiple produce the minimum, select all.

☐ 1, 2, 3 ☐ 1, 3, 2 ☐ 2, 1, 3 ☐ 2, 3, 1 ☐ 3, 1, 2 ☐ 3, 2, 1
☐ None of the above

iii) (1 Point). Which insertion ordering requires **the maximum** number of `rotateLeft` and `rotateRight` calls. If multiple produce the maximum, select all.

☐ 1, 2, 3 ☐ 1, 3, 2 ☐ 2, 1, 3 ☐ 2, 3, 1 ☐ 3, 1, 2 ☐ 3, 2, 1
☐ None of the above