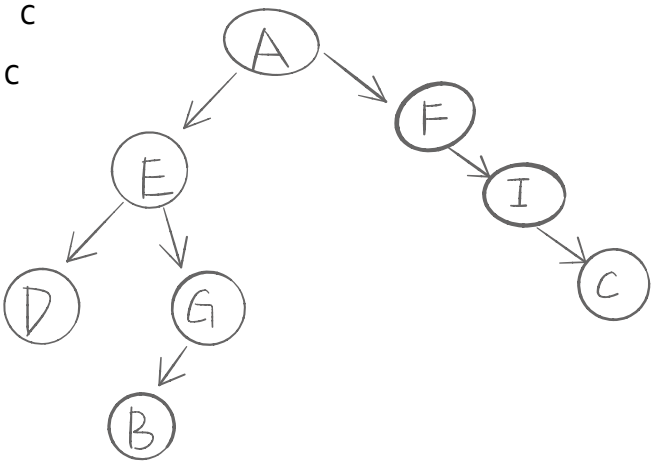
1. [3] Given the following pre-order and in-order traversals, reconstruct the appropriate binary search tree. **NOTE: You must draw a single tree that works for both traversals.**

Pre-order: A, E, D, G, B, F, I, C

In-order: D, E, B, G, A, F, I, C

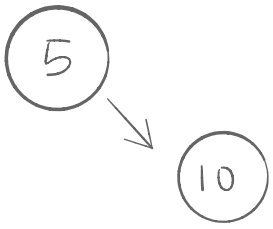


2. [3] Starting with an empty BST, draw each step in the following operation sequence. Assume that all removals come from the left subtree when the node to remove has two children.

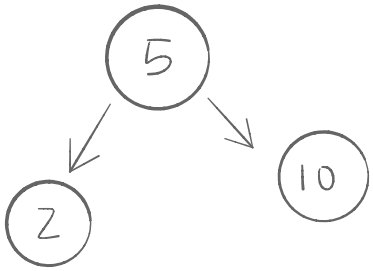
Insert(5), Insert(10), Insert(2), Insert(9), Insert(1), Insert(3), Remove(5).

Insert(5):

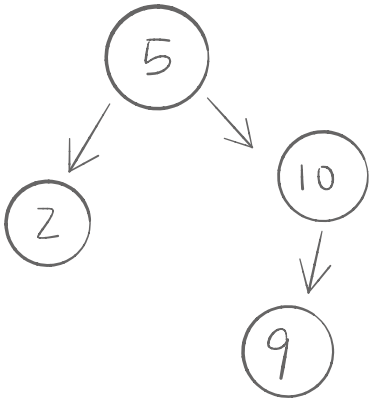
Insert(10):



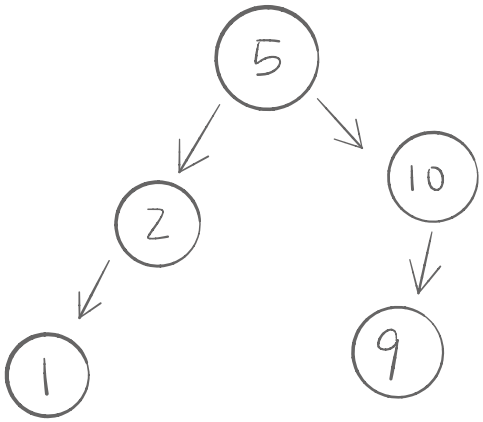
Insert(2):



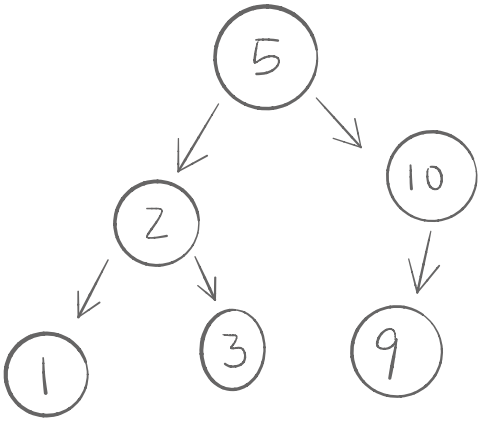
Insert(9):



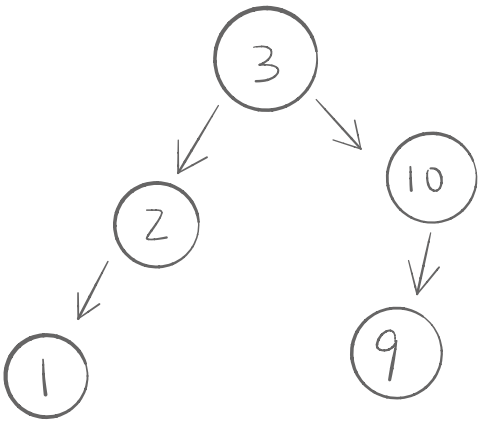
Insert(1):



Insert(3):



Remove(5):

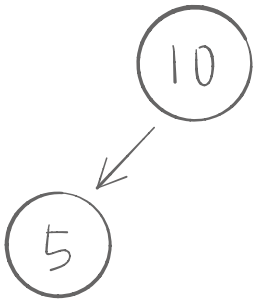


3. [3] Starting with an empty BST, draw each step in the following operation sequence. Assume that all removals come from the right subtree when the node to remove has two children.

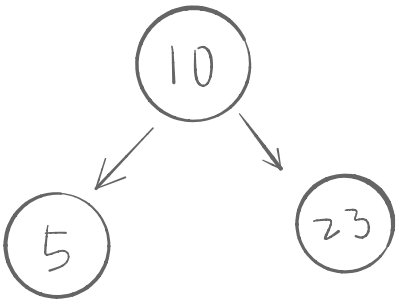
Insert(10), Insert(5), Insert(23), Insert(4), Insert(19), Insert(7), Insert(9), Insert(6), Remove(5).

Insert(10):

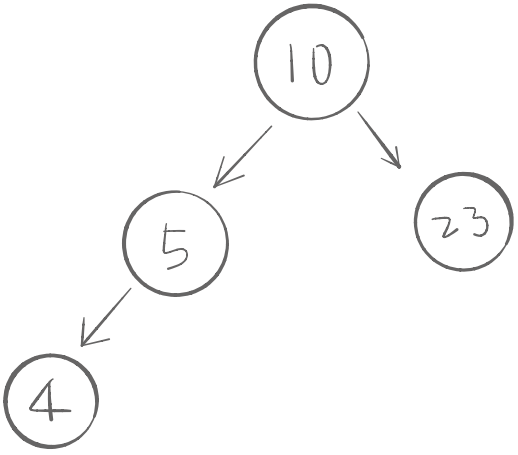
Insert(5):



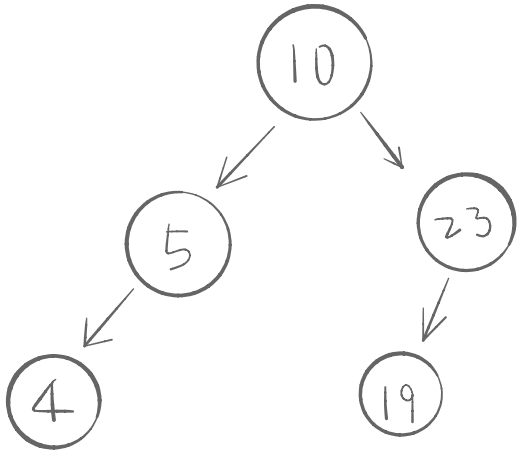
Insert(23):



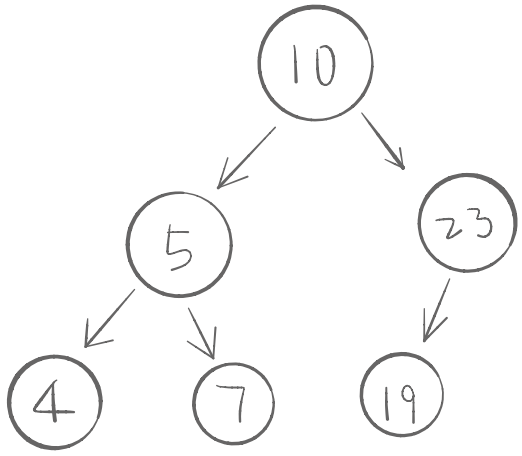
Insert(4):



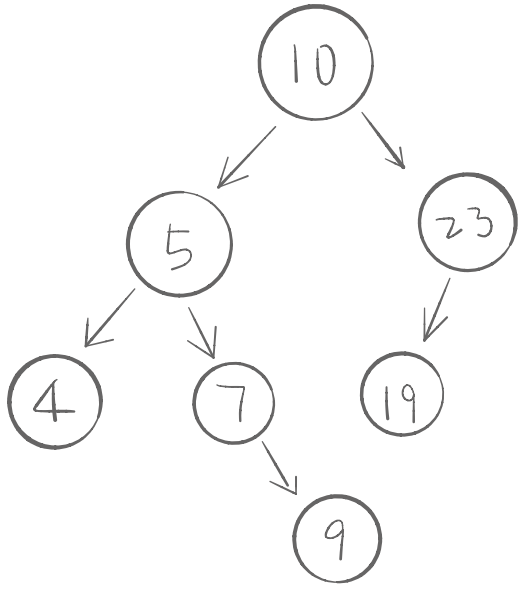
Insert(19):



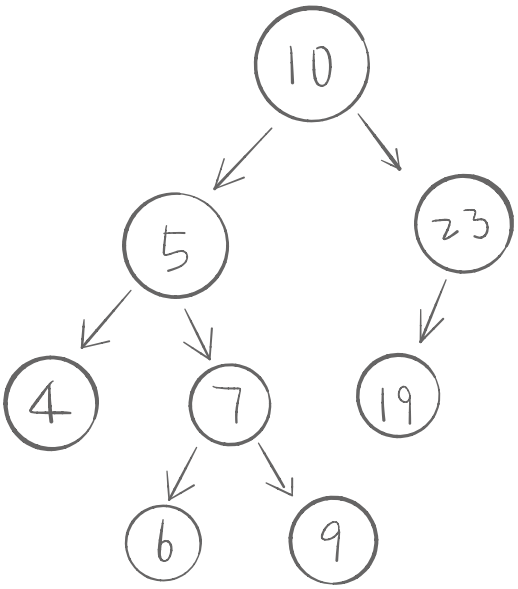
Insert(7):



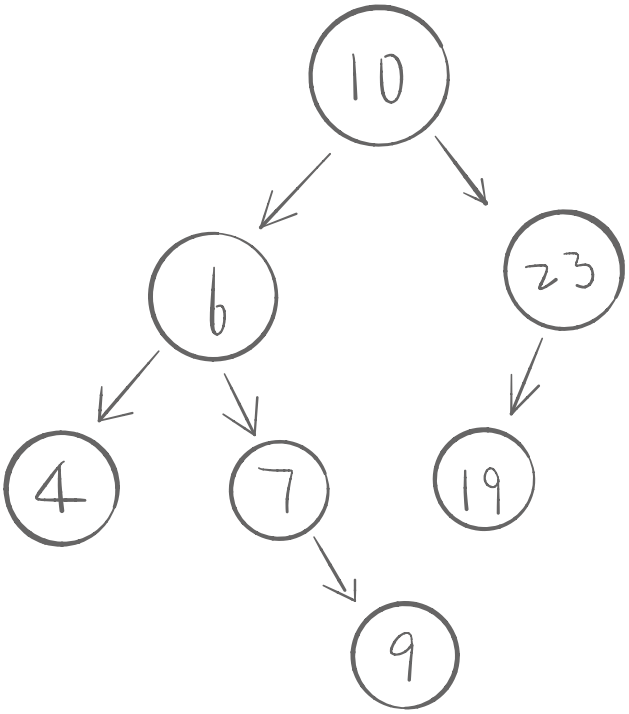
Insert(9):



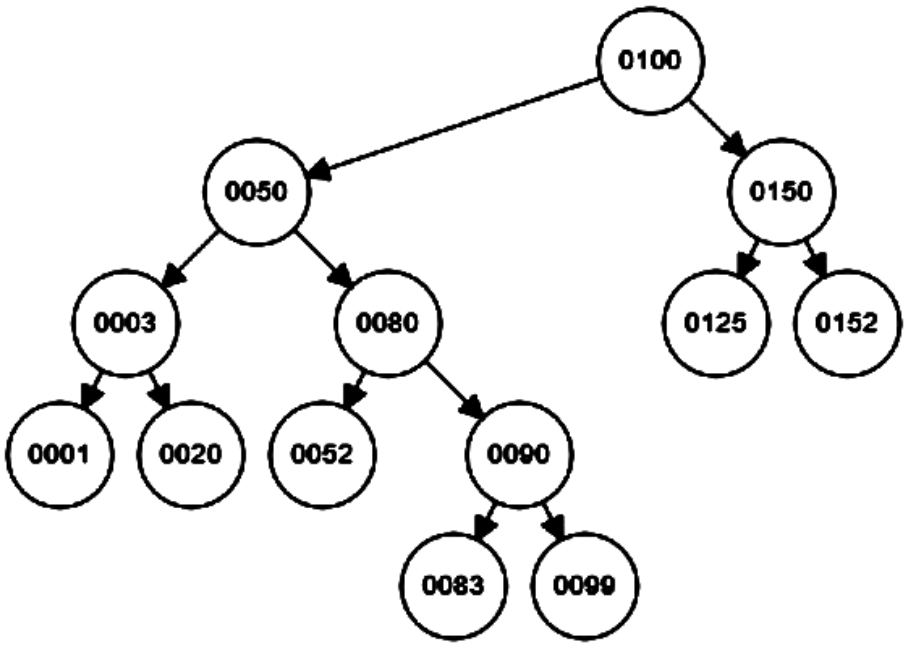
Insert(6):



Remove(5):



4. Given the following binary tree (where nullptr height == -1):



A. [1] What is the height of the tree?

4

B. [1] What is the depth of node 90?

3

C. [1] What is the height of node 90?

3

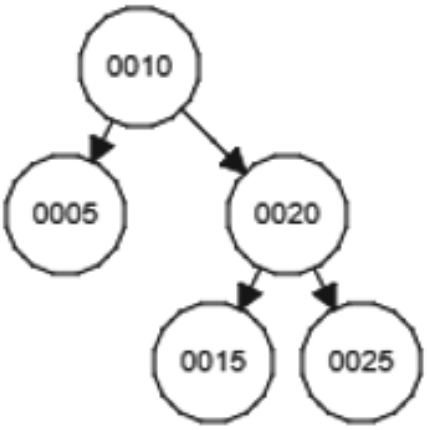
D. [3] Give the pre-order, in-order, and post-order traversal of this tree.

Pre-order: 100,50,3,1,20,80,52,90,83,99,150,125,152

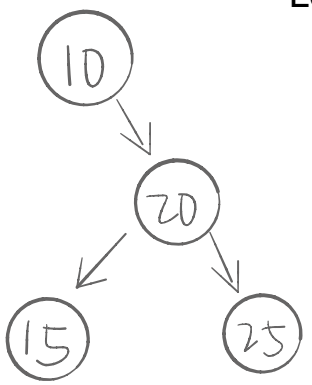
In-order:1,3,20,50,52,80,83,90,99,100,125,150,152

Post-order: 1,20,3,52,83,99,90,80,50,125,152,150,100

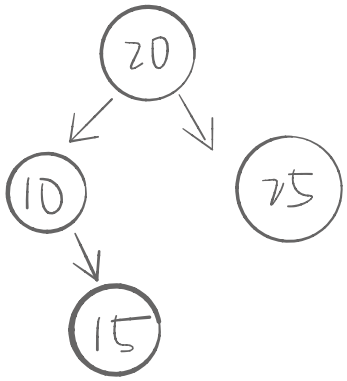
5. [3] Remove 5 from the following AVL tree; draw the results:



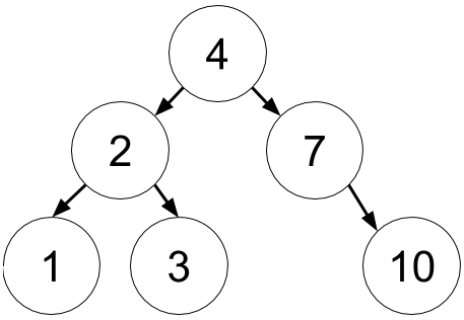
After removed:



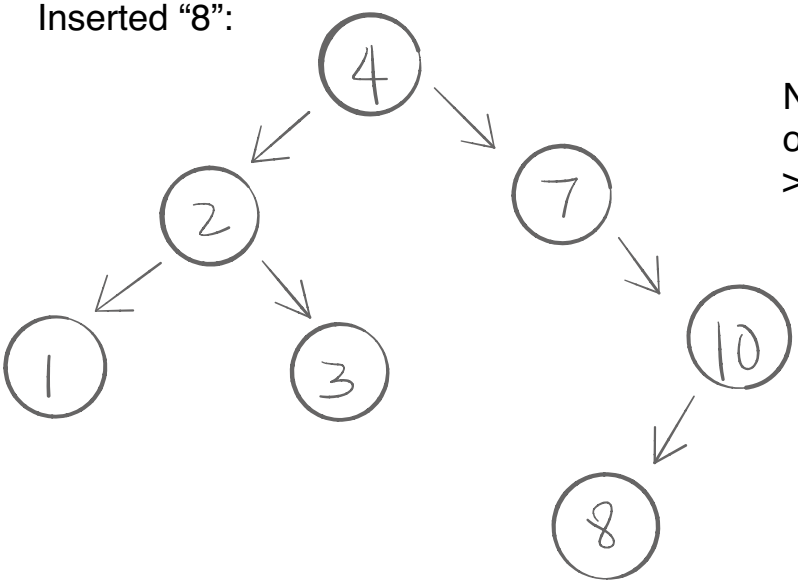
Left-rotation:



6. [3] Insert the value "8" into the following AVL tree; draw the result:

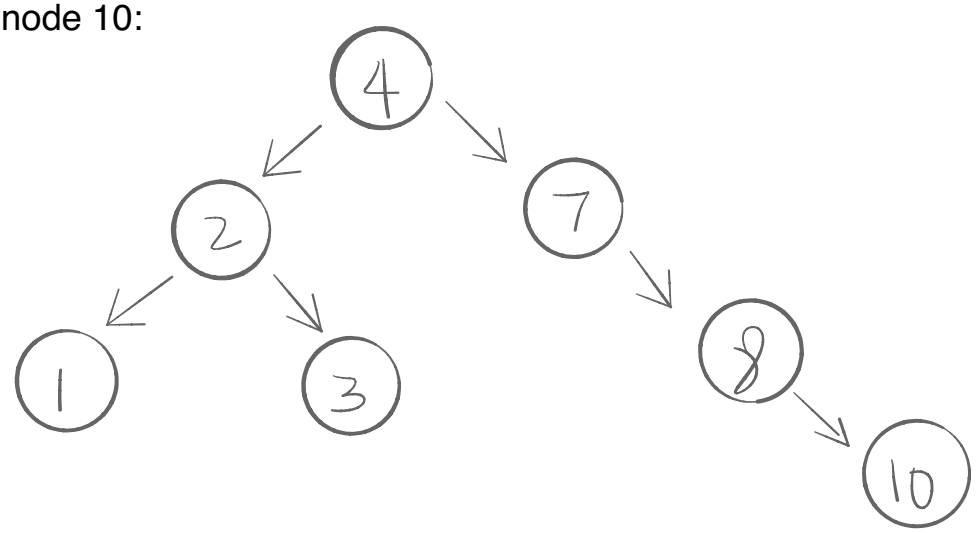


Inserted “8”:

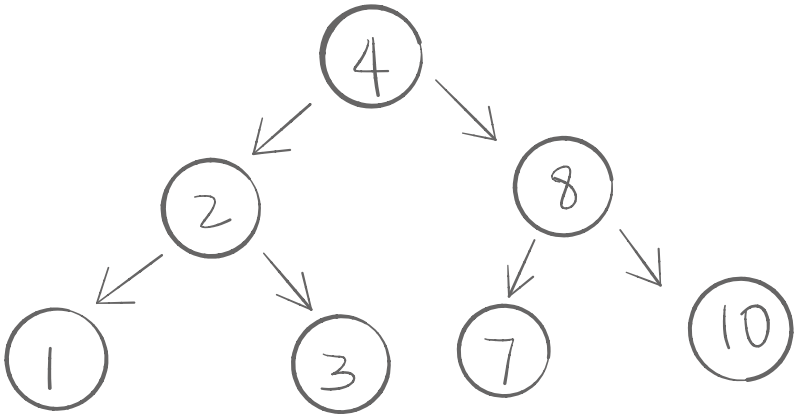


Node 7 has height of 0 in left sub tree, height of 2 in right sub tree, left height-right height >1, unbalanced.

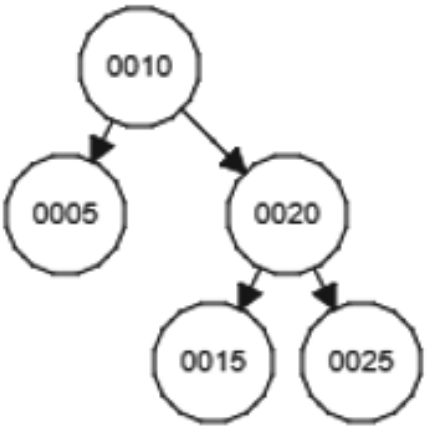
Right rotation at node 10:



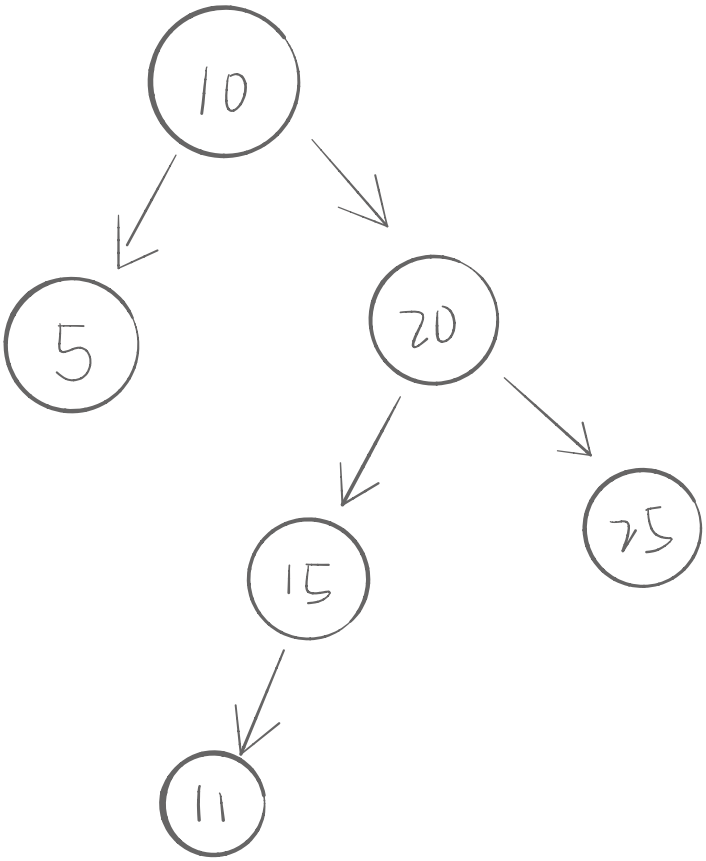
Left rotation at node 7:



7. [3] Insert the value “11” into the following AVL tree; draw the result:

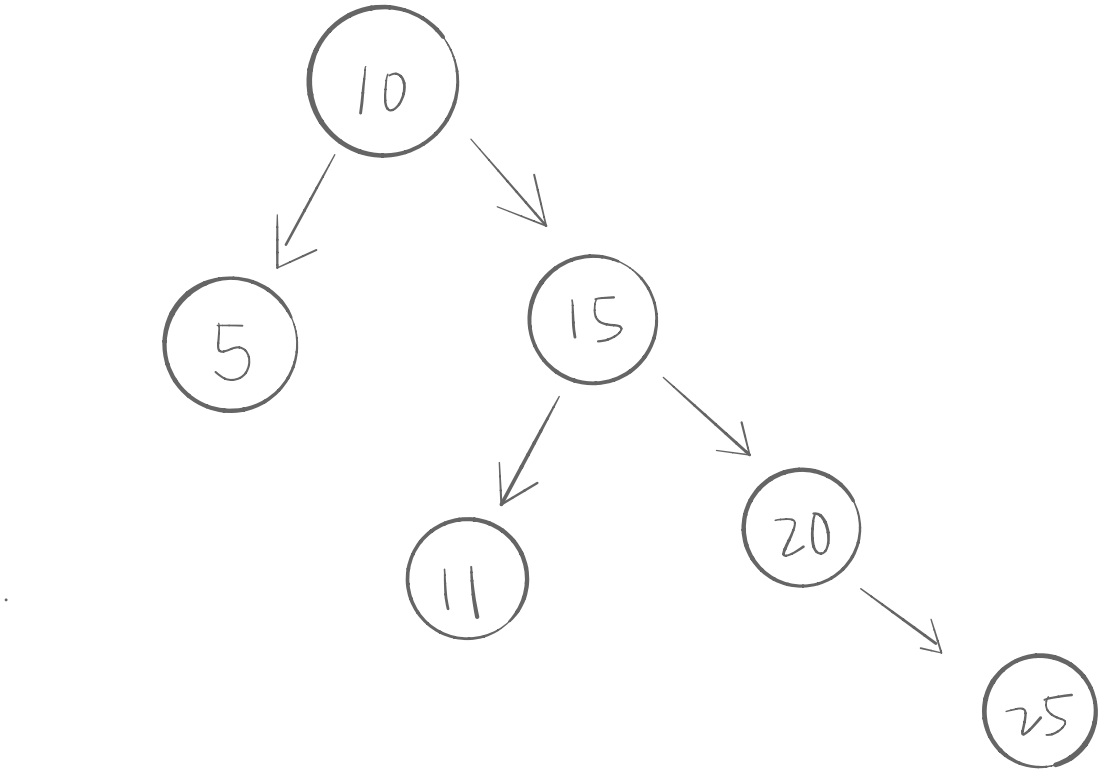


Inserted “11”:

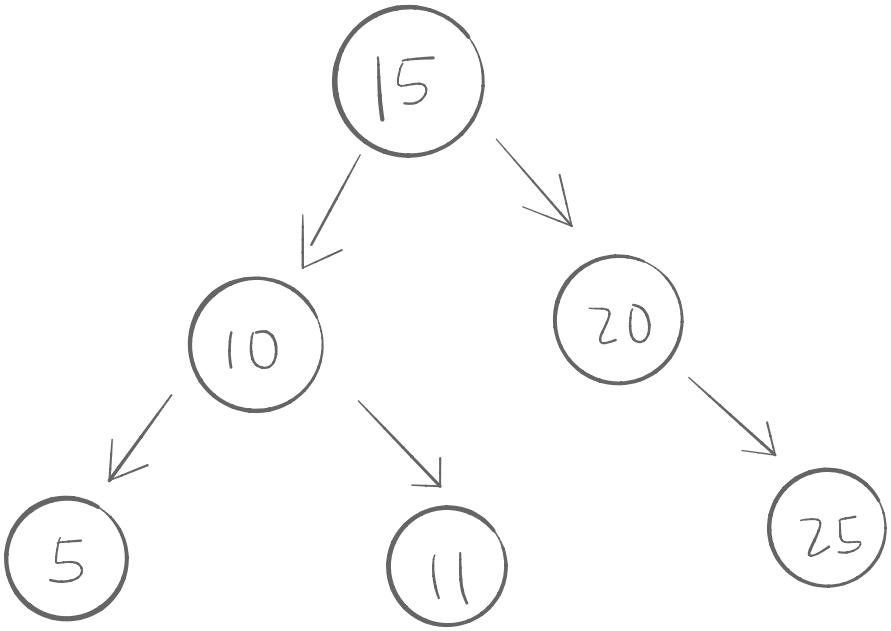


Node 10 has height of 1 at left sub tree and height of 3 at right sub tree, the difference between them is 2, which is unbalanced.

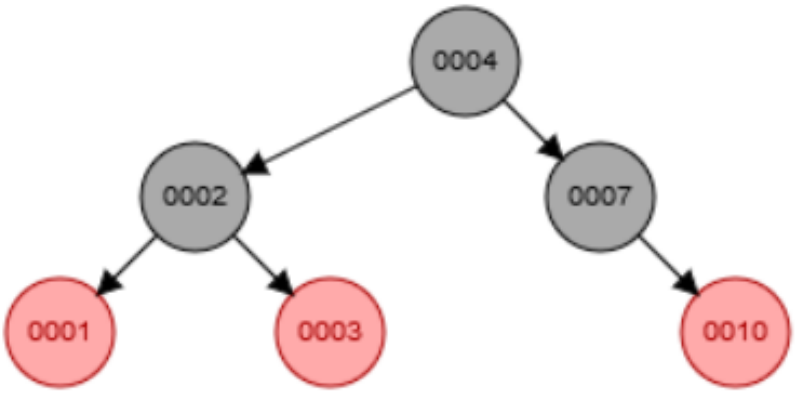
Right rotation at node 20:



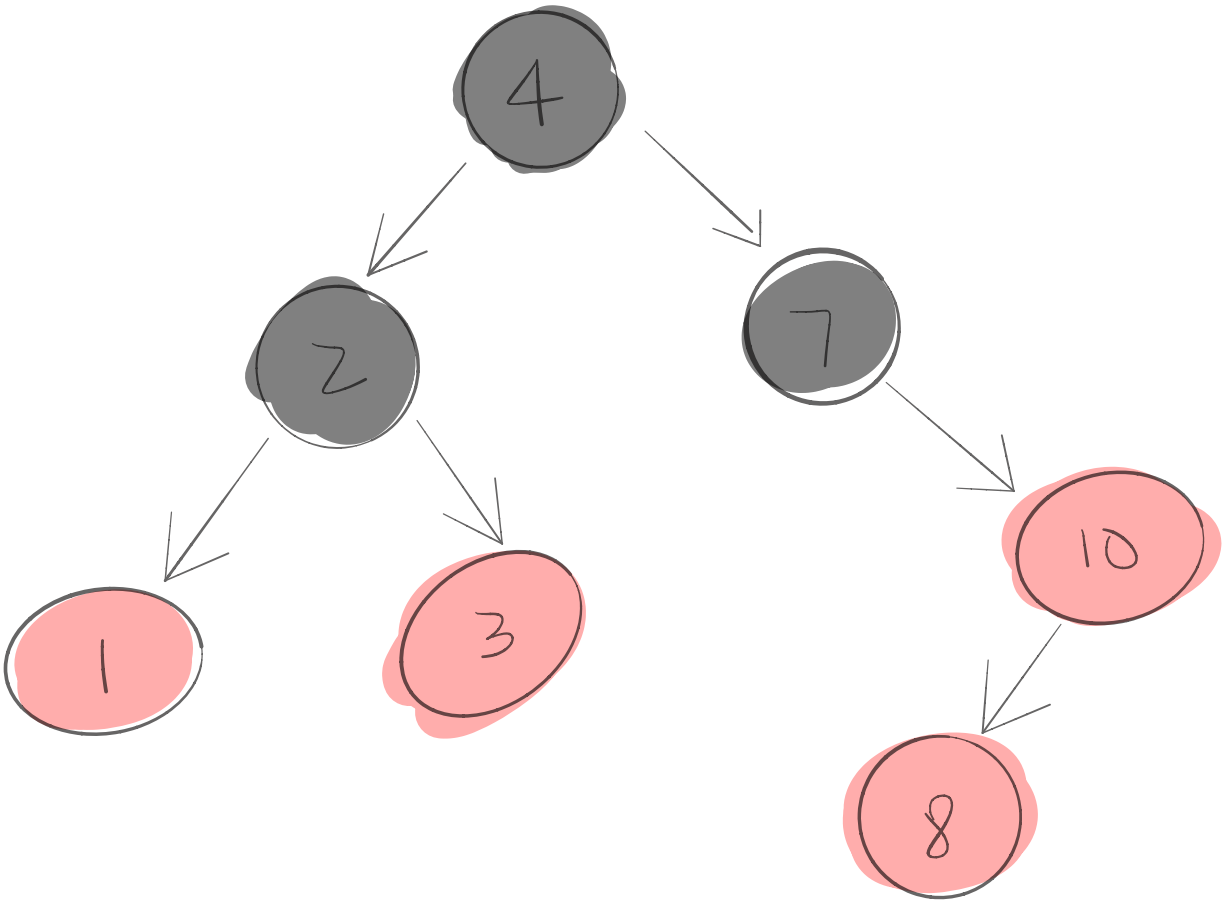
Left rotation at node 10:



8. [3] Insert the value "8" into the following Red-Black tree; draw the result. Use Double-circle to denote red nodes and single circle to denote black nodes.

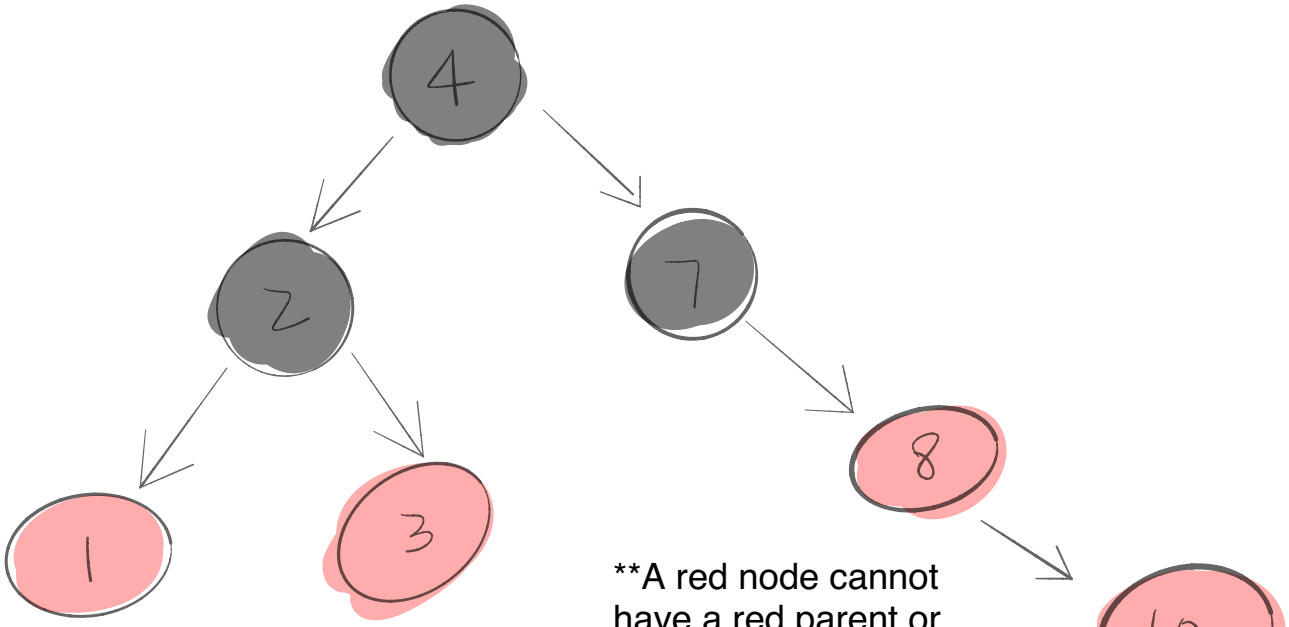


Inserted “8”: the new insert node is color of red



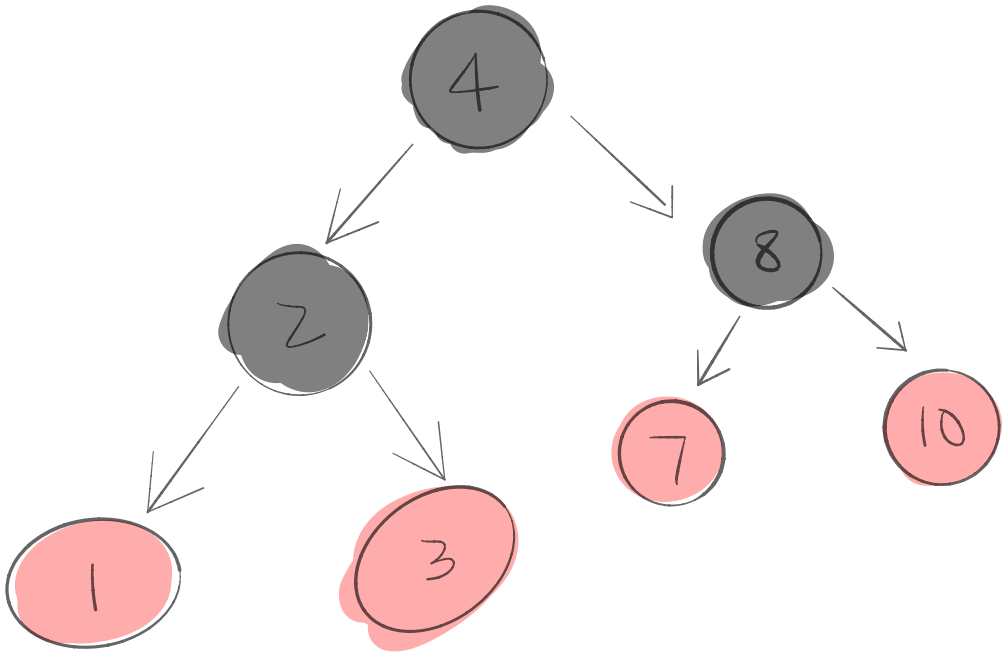
The left sub tree of node 7 is null, null node is black. We do rotation because of the black null node.

Right rotation at node 10: (Incomplete red-black tree)

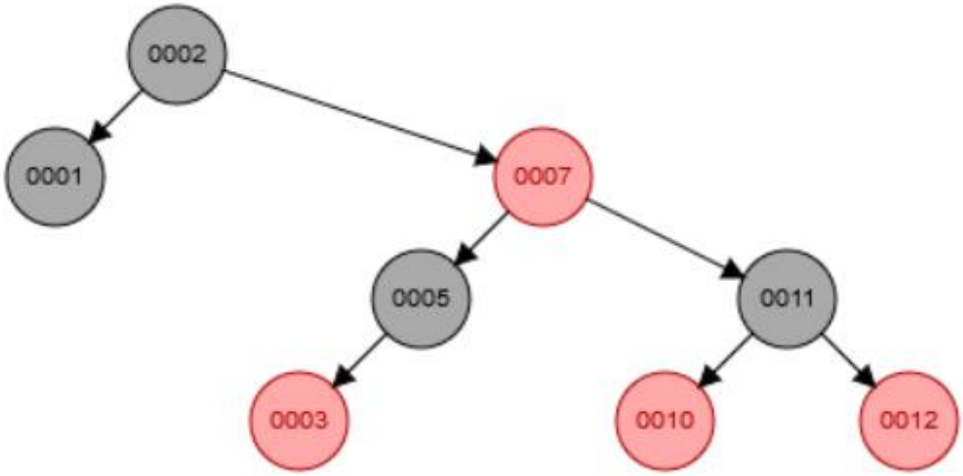


**A red node cannot have a red parent or red child, this is just showing the process step by step. We still need to do additional steps to adjust this insertion for the red-black tree.

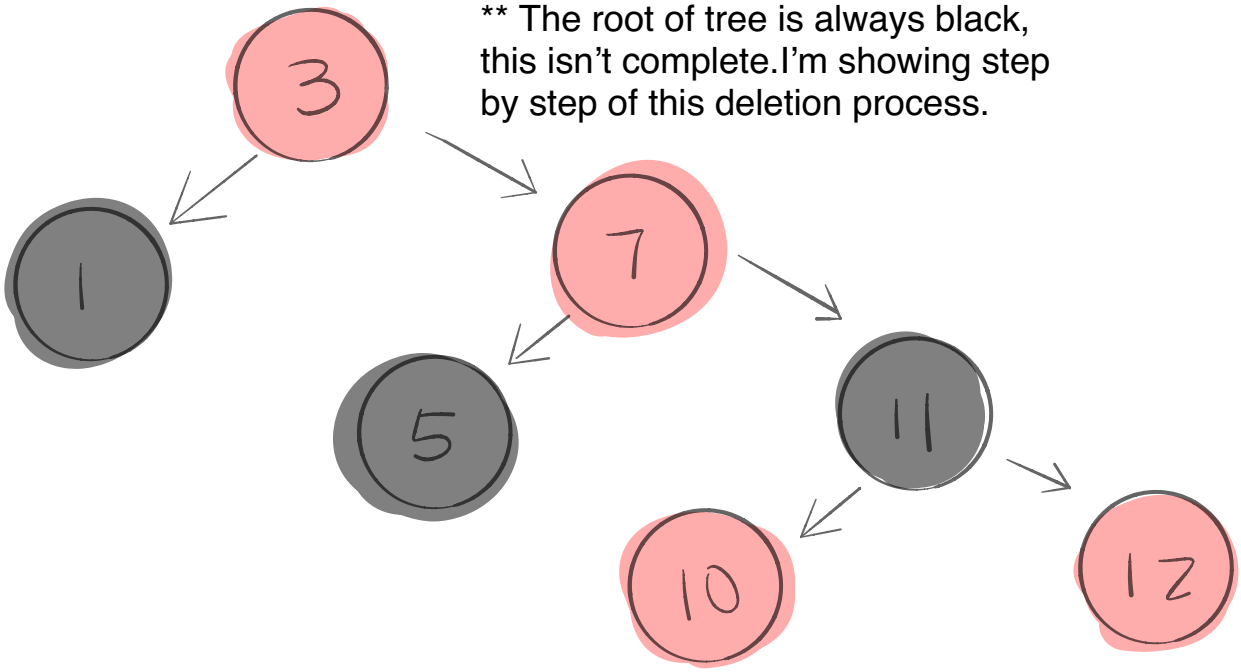
Left rotation at node 7, and change the color of node 8 to black. We have to follow the rule: every path from a node to any of its descendant null node has the same number of black nodes



9. [3] Delete the value "2" from the following Red-Black tree; draw the result. Use Double-circle to denote red nodes and single circle to denote black nodes.

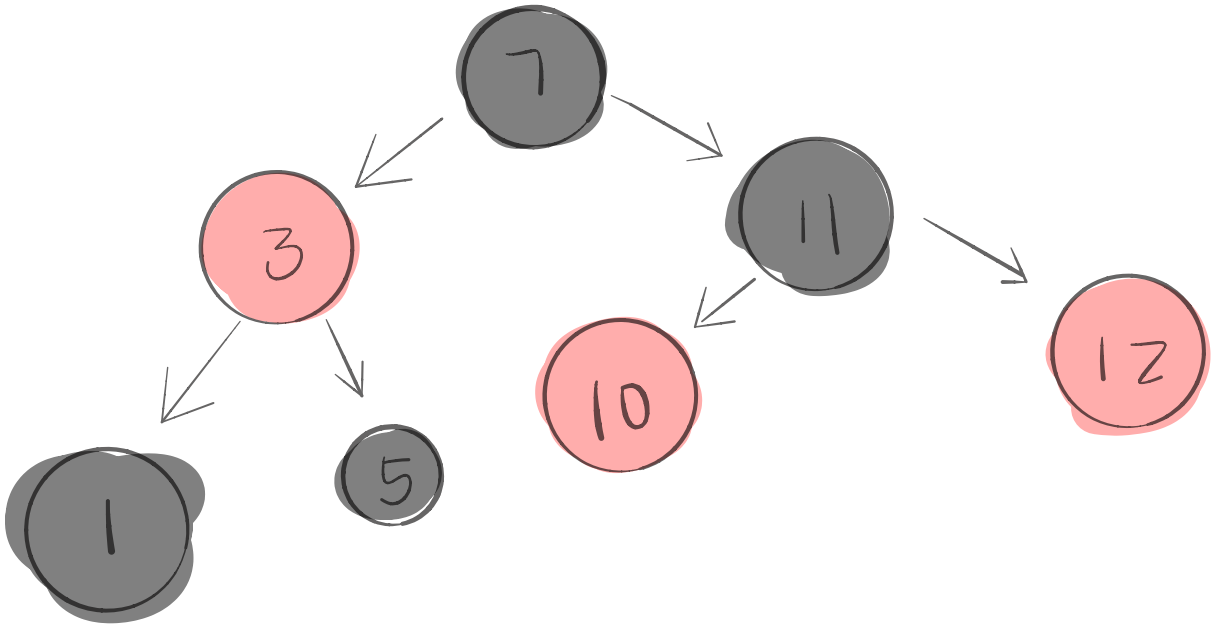


Deleted “2”: node 2 has two children, replace the node 2 by the minimum node element that greater than node 2.

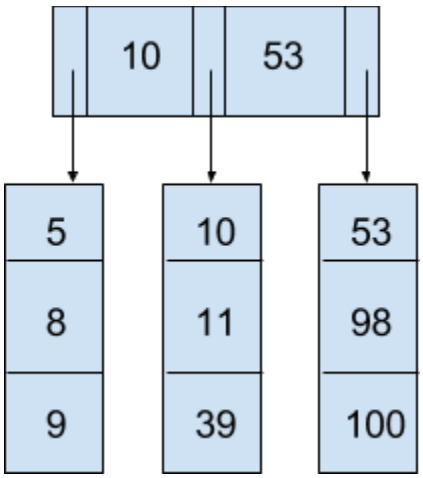


** The root of tree is always black, this isn’t complete.I’m showing step by step of this deletion process.

Left rotation at node 3:

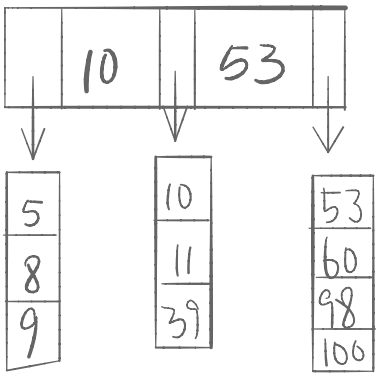


10. [4] Given the following B+ tree (M = 3, L = 3):

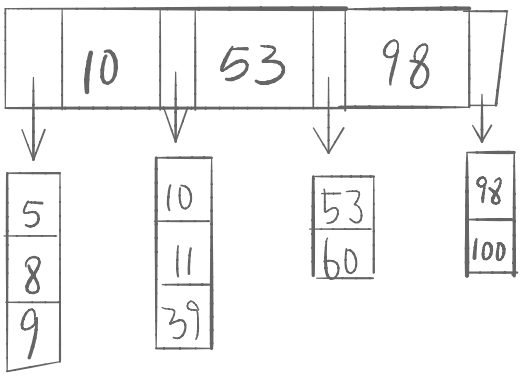


A) Insert 60 into the tree and draw the resulting B+ Tree:

Inserted 60:

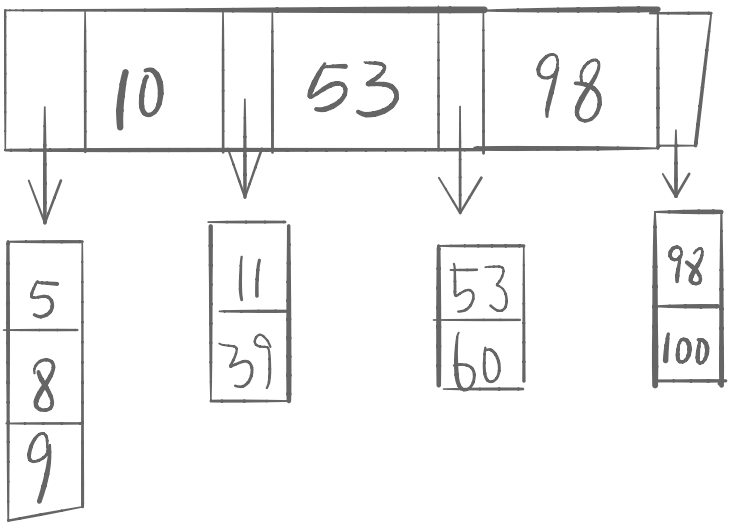


**All leaf nodes can have at most L=3 values and at least L/2= 3/2= 1.5



B) Based on the tree resulting from part (A), now remove 10 and draw the new tree:

Removed “10”:



11. [6] We are going to design our B+ Tree to be as optimal as possible for our old hard drives (since the management won't buy new ones, those cheapskates!). We want to keep the tree as short as we can, and pack each disk block in the filesystem as tightly as possible. We also want to access our data in sorted order for printing out reports, so each leaf node will have a pointer to the next one. See figure #1 on next page for a visualization of our tree.

CPU architecture: Intel Xeon with 64 bit cores
Filesystem: Ext4 with 4KB (4096 byte) blocks
The customer records are keyed by a random UUID of 128 bits

Block size = 4096 bytes
Key size = 1 bytes
Pointer = 2 bytes
Record size = 2 byte

Customer's Data record definition from the java file:

```
#include <uuid>
struct CustomerData {
      UUID uuid;          // Customer 128 bit key[1]
      char[32] name;      // Customer name (char is 2 bytes each)[2]
      int ytd_sales;      // Customer year to date sales
};
```

Calculate the size of the internal nodes (M) for our B-tree:

fs blocksize >= M · pointerbytes + (M−1) · keysize bytes
4096 >= M·(2) + (M−1)·16
4096 >= 2M + 16M − 16
M <= 228.44 bytes

There're several possible answers for the size of the internal nodes as long as the internal nodes M <= 228.44

Calculate the size of the B-tree leaf nodes (L) for this tree make sure to include the pointer (note CPU architecture!) to keep the list of leaf nodes:

fs blocksize >= L * size of Record + (optional next-L block pointer bytes)
32768 >= L·(128) + 128
32640 >= 128L
L <= 255

[1] https://docs.oracle.com/javase/7/docs/api/java/util/UUID.html
[2] http://cs-fundamentals.com/java-programming/java-primitive-data-types.php

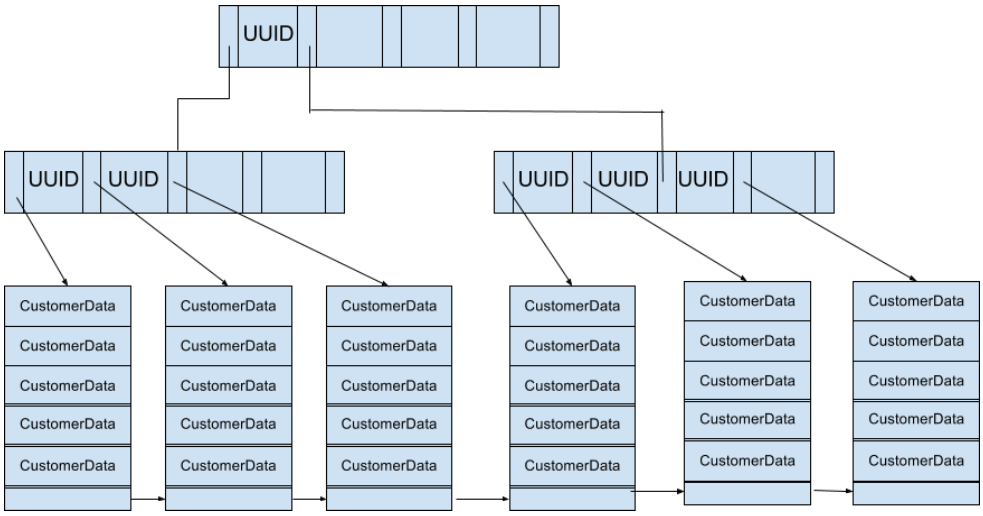


Figure #1: Visualization of our B+ Tree of height 2, customer data records, and pointers between the leaf nodes.

How tall (on average) will our tree be (in terms of M) with N customer records?

$$\log_{(M+1)} N < tall \leq \log_{M} N$$

If we insert 30,000 CustomerData records, how tall will be tree be?

$\log_{(5+1)} 30000 = 5.75$ $\log_{5} 30000 = 6.4$

$tall = 6$

If we insert 2,500,000 customers how tall will the tree be?

$\log_{(5+1)} 2500000 = 8.222$

$\log_{5} 2500000 = 9.15338$

$tall = 9.$