CptS 223 Homework #2 - All About Those Trees

Please complete the homework problems on the following page using a separate piece of paper or digitally as you see fit. Note that this is an individual assignment and all work must be your own. Be sure to show your work when appropriate.

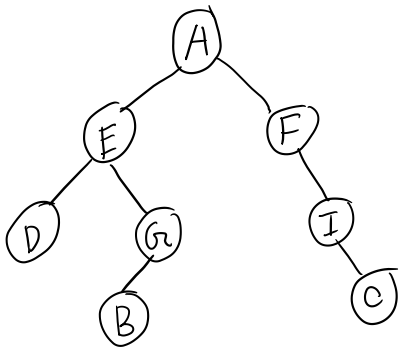
Yes, this is many pages, but each one is a relatively small amount of work. Normally, it's structure manipulations and some calculations.

Turn in this assignment by making a PDF of it. Then put that PDF into your git repository on the HW2 branch. Commit and push it to the server. On Blackboard, only turn in a file with the commit hash on the HW2 branch with your file. The grader will clone your repository, change to the HW2 branch and checkout your commit.

I suggest removing the original assignment files (I'm giving you the PDF, docx, and odt formats), so it's obvious to the grader what to grade. Just having the PDF of your answers there will make it clear which file you want them to look at.

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

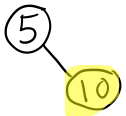


2. [3] Starting with an empty BST, draw each step in the following operation sequence. Assume that all removals come from the left subtree when the node to remove has two children.

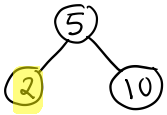
Insert(5), Insert(10), Insert(2), Insert(9), Insert(1), Insert(3), Remove(5).

Insert(5)

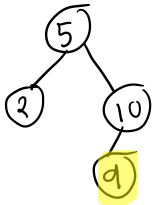
Insert(10) (5 < 10), insert right of 5.



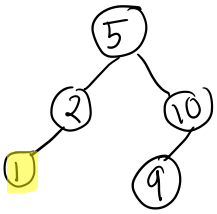
Insert(2) (5 > 2), insert left of 5.



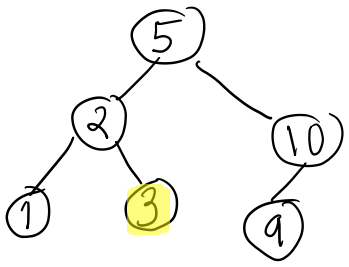
Insert(9) (5 < 9), more to the right and then (10 > 9), insert left of 10.



insert(1) (5 > 1), more to the left, and then (2 > 1), insert left of 2.

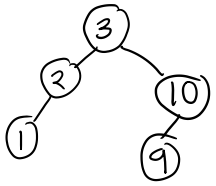


insert(3) $(5 > 3)$, move to the left, and then $(2 < 3)$, insert to the right of 2.



Remove(5) : If node to be removed has 2 children, we replace the node with the largest element of the left subtree and recursively remove the node.

We are removing 5 and the largest value in the left subtree is 3. So, replace 5 with 3 to be the new root of the tree.



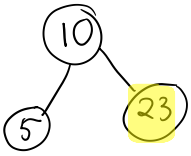
3. [3] Starting with an empty BST, draw each step in the following operation sequence. Assume that all removals come from the right subtree when the node to remove has two children.

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

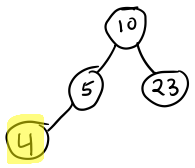
Insert(10)

Insert(5) $(5 < 10)$ move 5 to the left of 10.

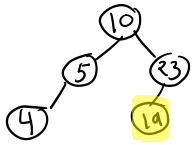
Insert(23) $(23 > 10)$ move 23 to the right of 10.



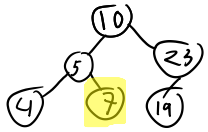
Insert(4) (4<10), move 4 to the left of 10 and then (4<5) move 4 to the left of 5.



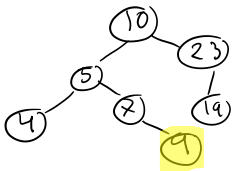
Insert(19) (19>10) move 19 right of 10 and then (19<23), moves 19 to the left of 23.



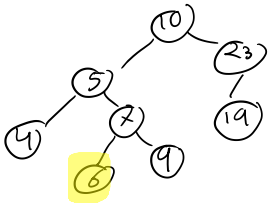
insert(7) (7<10) moves 7 to the left of 10. and then (7>5) moves 7 to the right of 5.



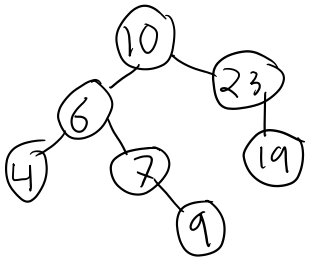
Insert(9) (9<10) moves 9 to the left of 10 and then (9>5) moves 9 to the right of 5. and then (9>7) move 9 to the right of 7.



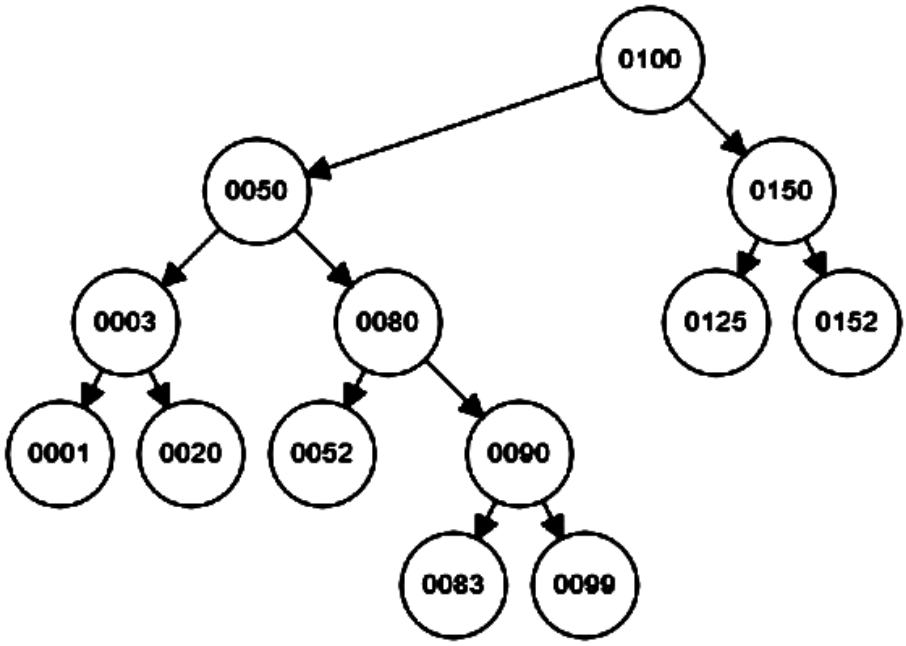
Insert(6) (6<10) moves 6 to the left of 10. and then (6>5) moves 6 to the right of 5 and then (6<7) move 6 to the left of 7.



remove(5) Since 5 has 2 children, we replace the node with the smallest value from the right subtree.



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?

1

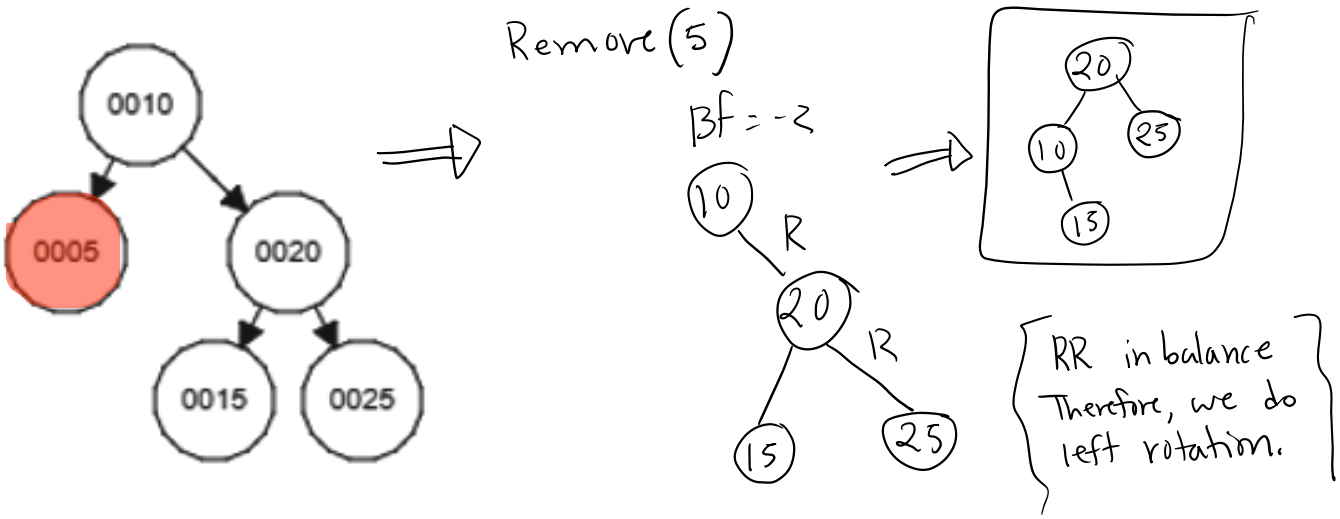
D. [3] Give the pre-order, in-order, and post-order traversal of this tree.

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

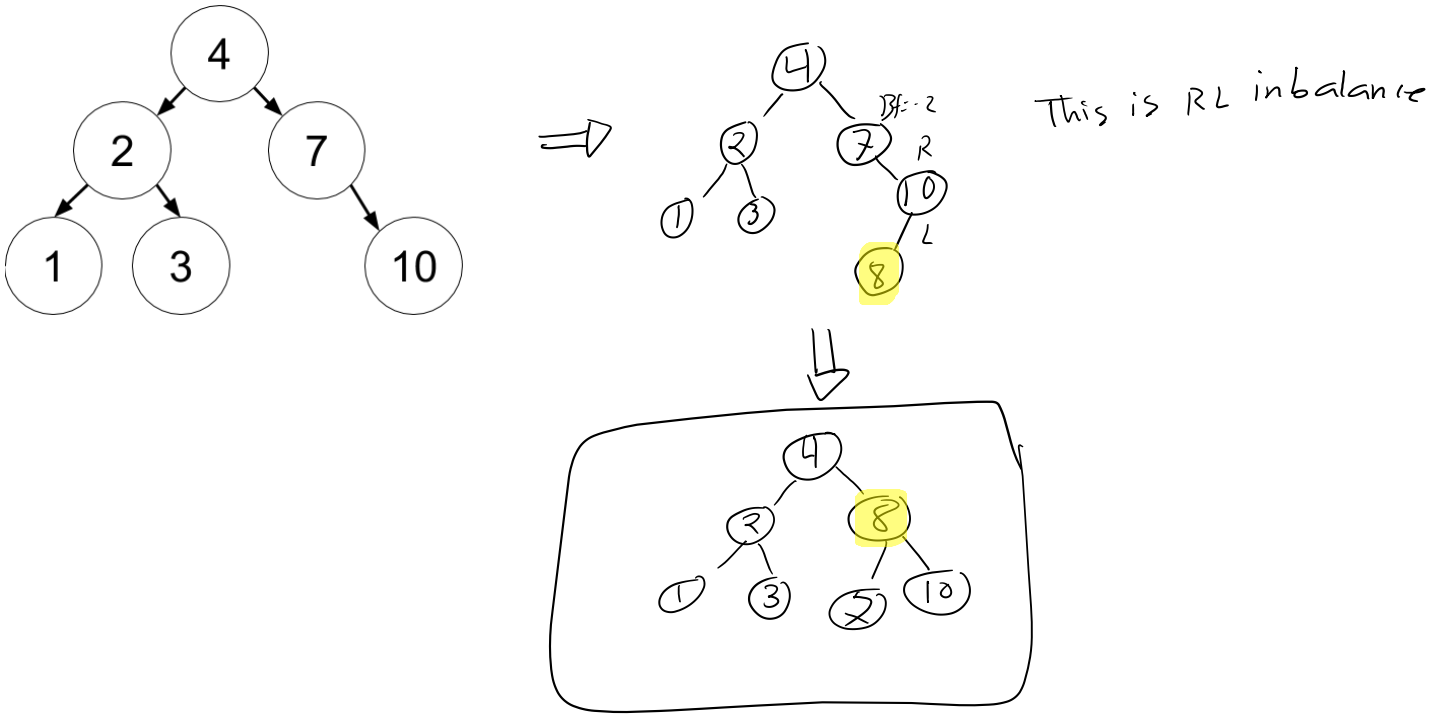
In-order: 1, 3, 20, 50, 52, 80, 83, 90, 99, 100, 125, 150, 152

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

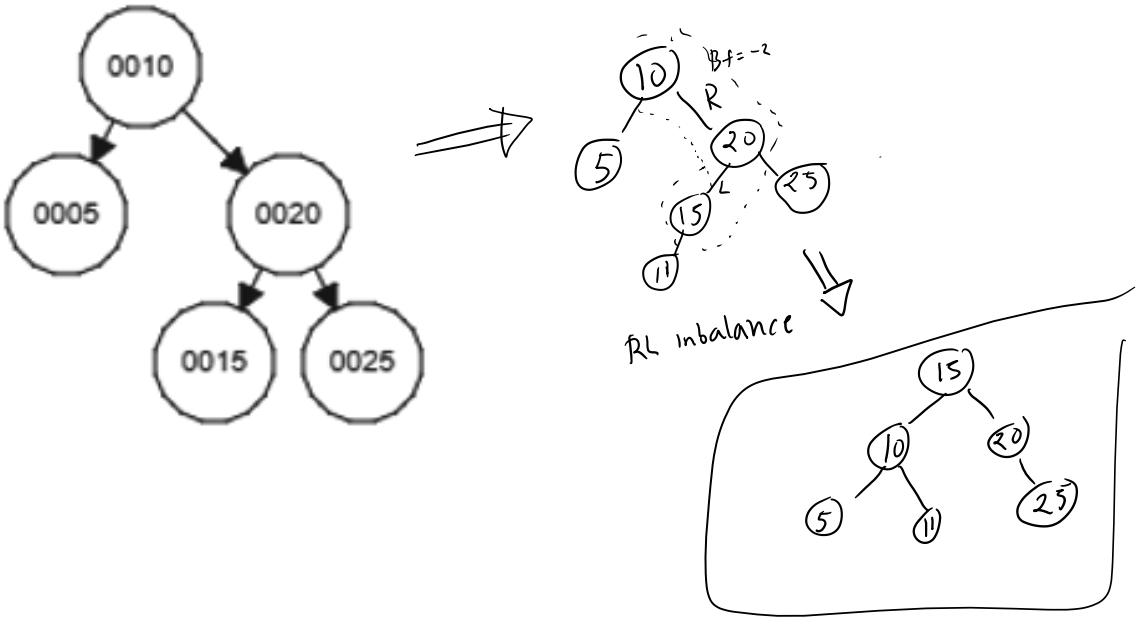
5. [3] Remove 5 from the following AVL tree; draw the results:



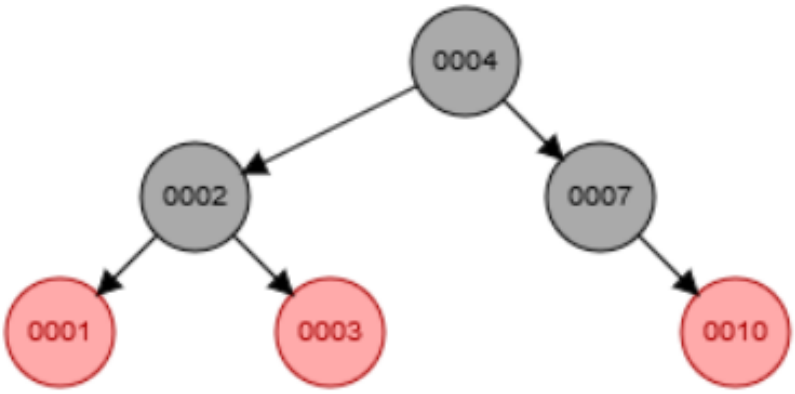
6. [3] Insert the value "8" into the following AVL tree; draw the result:



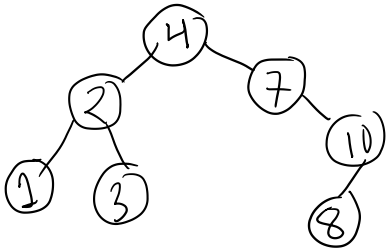
7. [3] Insert the value “11” into the following AVL tree; draw the result:



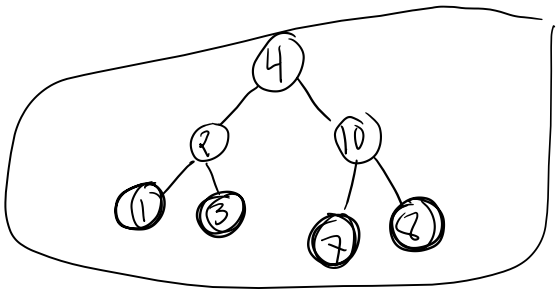
8. [3] Insert the value "8" into the following Red-Black tree; draw the result. Use Double-circle to denote red nodes and single circle to denote black nodes.



By inserting(8)

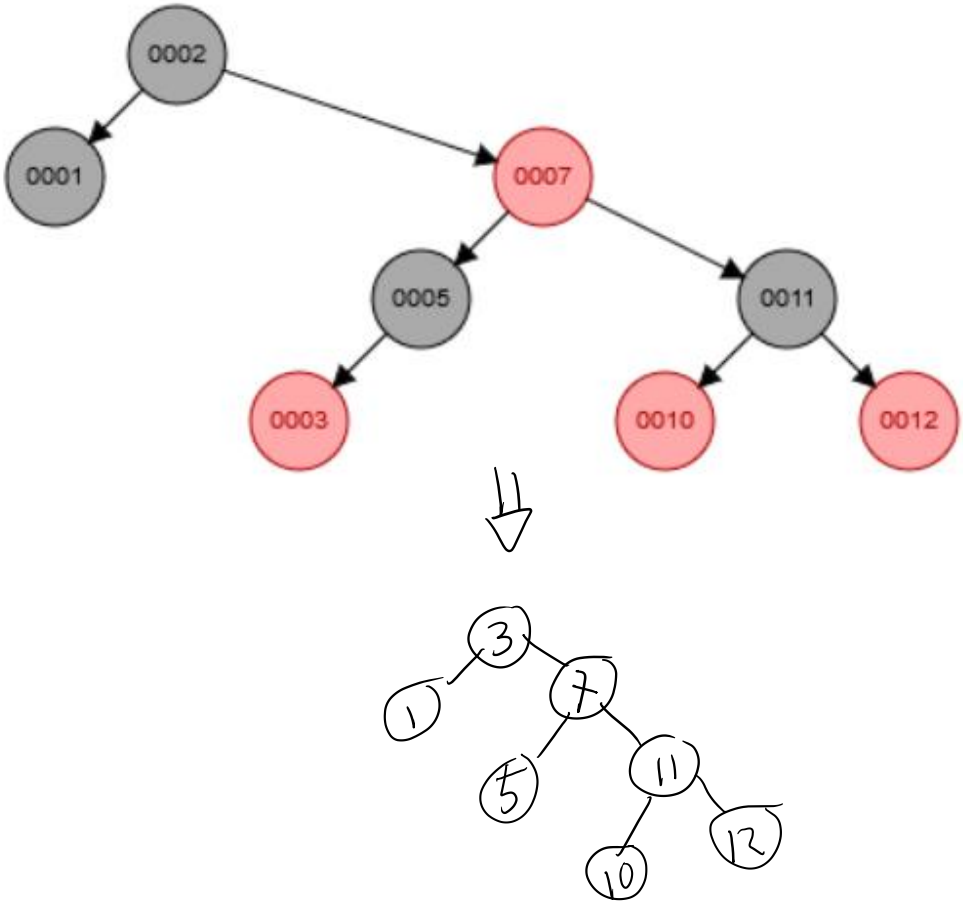


Rotation to balance the tree



After inserting "8", it will be unbalanced according to the property of AVL search tree. We have to perform RR rotation in order to balance the tree from tree-2 we get tree-3.

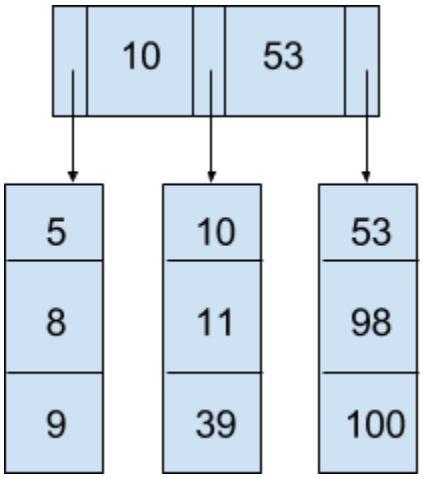
9. [3] Delete the value "2" from the following Red-Black tree; draw the result.
Use Double-circle to denote red nodes and single circle to denote black nodes.



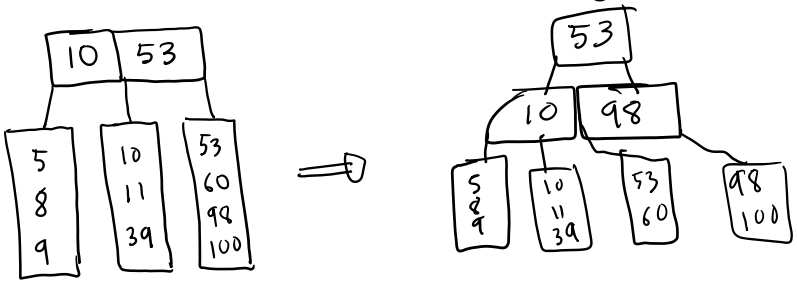
In order to deletion of node-2 in the right subtree is smallest element that is node 3. So we replace 2 by 3

For deletion, there is no child for node-3 so we don't need to worry about the left nodes.

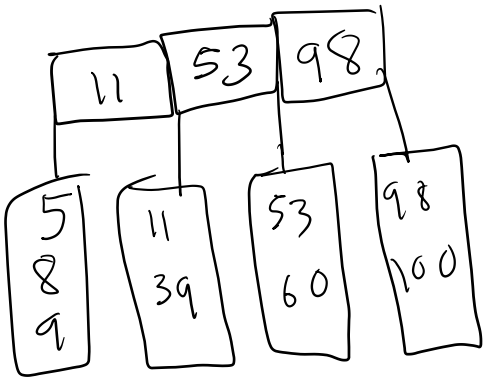
10. [4] Given the following B+ tree (M = 3, L = 3):



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



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



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

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:

The size of the internal nodes(m) is [3.] This is because in the given B tree structure there are 3 nodes avilable.

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:

The size of B-tree Leaf nodes (L) is [6.]

[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

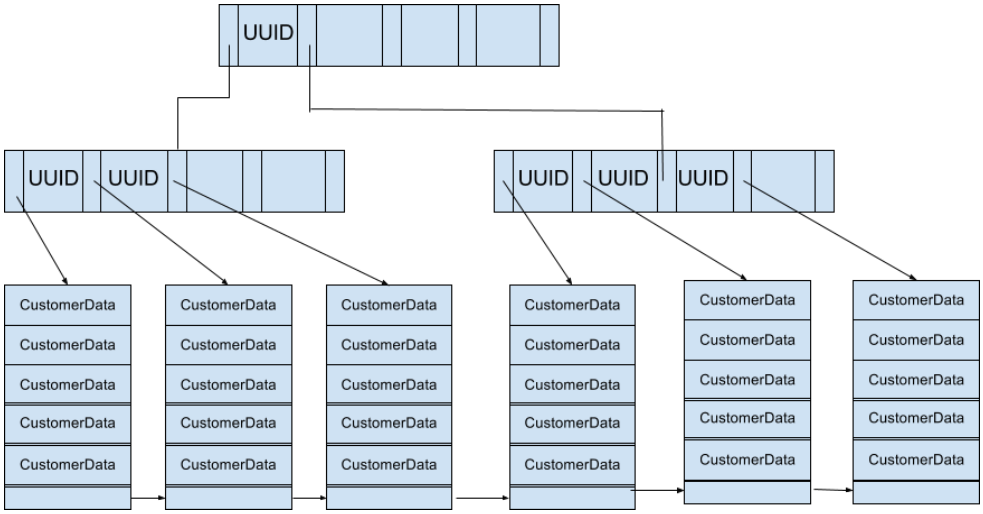


Figure #1: Visualization of our B+ Tree of height 2, customer data records, and pointers between the leaf nodes.

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

Since each B+ tree node can hold 5 pointers.

thus, $5^m < n \leq 5^{m+1}$, where height = m.

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

$5^6 < 30,000 \leq 5^{6+1}$

$15,625 < 30,000 \leq 78,125$

So, the height is 6

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

$5^9 < 2,500,000 \leq 5^{10}$

$1,953,125 < 2,500,000 \leq 9,765,625$

So, the height is 9