

Exercise on AVL trees

44251017 *HuangJiahui*

(1) Construct an AVL tree, by inserting the following keys in this order: 5, 4, 8, 6, 12, 11, 9, 7. Draw the AVL tree after each insertion.

Insert 5:

1		5
---	--	---

Insert 4:

1		5	
2		/	
3		4	

Balanced. No rotation needed.

Insert 8:

1		5	
2		/ \	
3		4 8	

Balanced. No rotation needed.

Insert 6:

1		5	
2		/ \	
3		4 8	
4			/
5			6

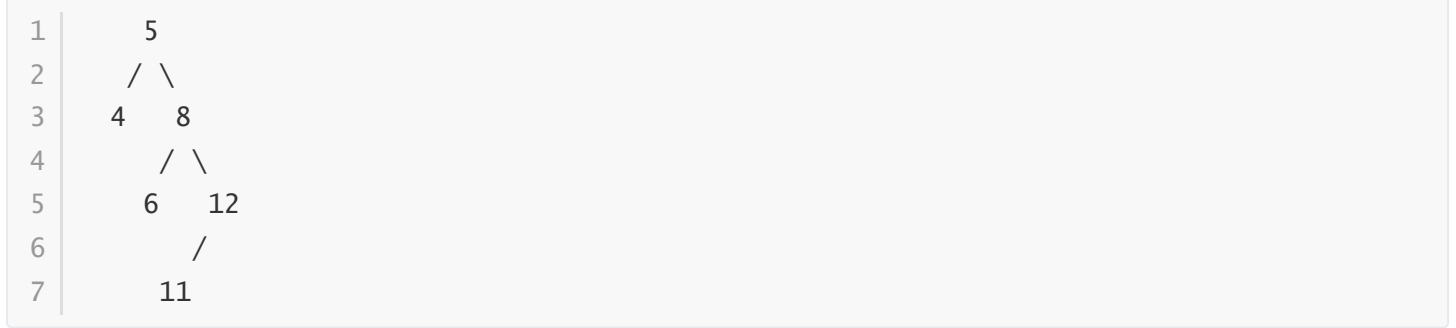
Still balanced. No rotation needed.

Insert 12:

1		5	
2		/ \	
3		4 8	
4			/ \
5		6 12	

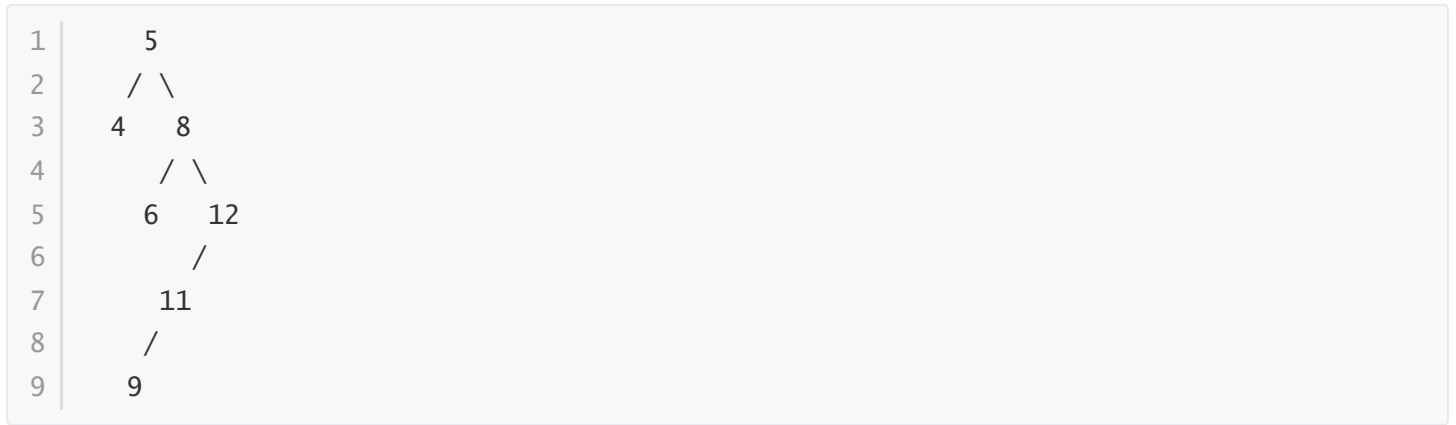
Balanced.

Insert 11:



Balanced.

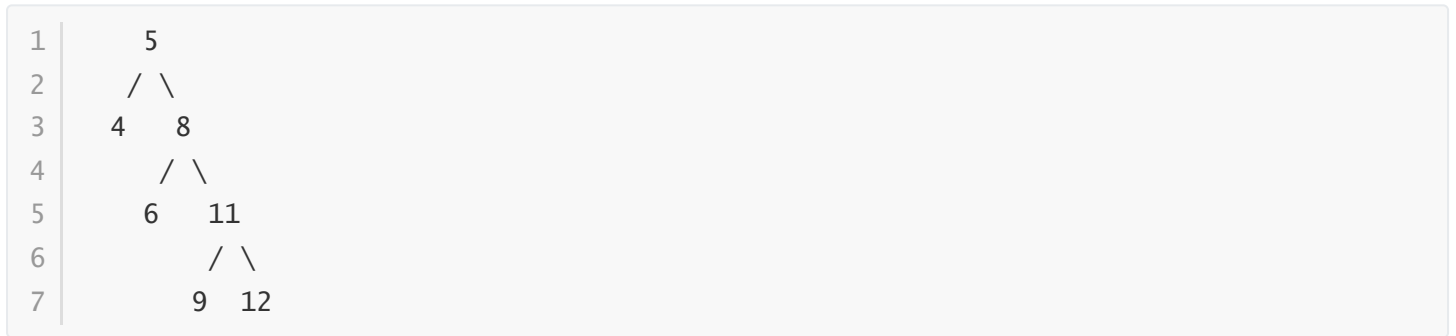
Insert 9:



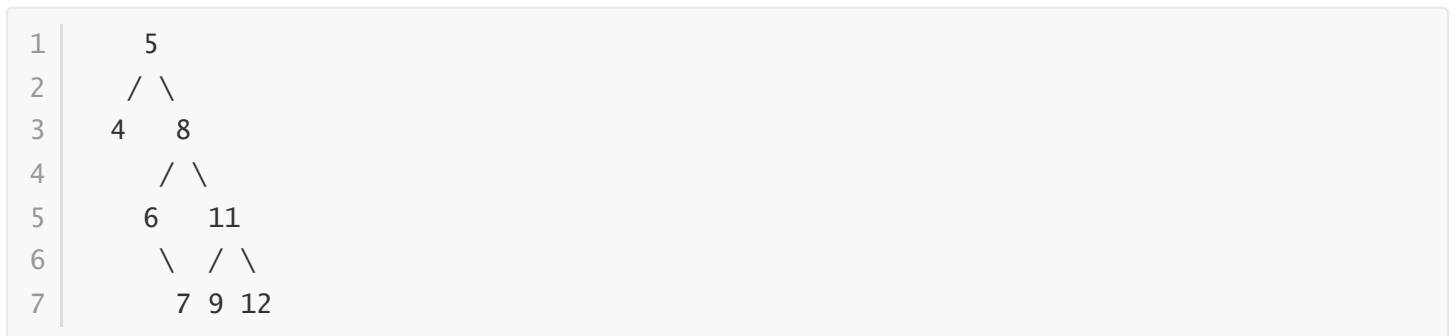
Now node 12 is unbalanced (left-heavy).

- Subtree: $12 \rightarrow 11 \rightarrow 9$
- Case: Left-Left imbalance \rightarrow single right rotation at 12

After rotation:



Insert 7:



Now node 6 is unbalanced (right-heavy).

Subtree: $6 \rightarrow 7$

Case: Right-right imbalance \rightarrow single left rotation at 6

After rotation

Final AVL tree:



Balanced.

(2) For a binary search tree, show an efficient algorithm to find the ceiling entry $\text{CeilingEntry}(k)$ such that returning an entry having smallest key that is no smaller than k . For accessing methods of tree use $\text{parent}(p)$, $\text{left}(p)$, $\text{right}(p)$, $\text{root}()$ (shown in Ch7-1).

```
1 Position<E> ceilingEntry(K k) {
2     Position<E> p = root();           // Start from the root
3     Position<E> candidate = null;     // Best candidate so far
4
5     while (p != null) {
6         if (key(p) == k) {
7             return p;
8         } else if (key(p) < k) {
9             p = right(p);              // Too small → go right
10        } else {
11            candidate = p;              // p could be the ceiling
12            p = left(p);                // Try to find smaller candidate
13        }
14    }
15    return candidate;                  // Return best ceiling found
16 }
```

(3) Show the worst case running time of your algorithm in (2). Here assume that the tree has n nodes and its height is h .

- Worst-case time: $O(h)$
- For AVL trees, height $h = O(\log n)$
- So the worst case running time is: $O(\log n)$