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:

```
5
```

Insert 4:

```
  5
 /
4
```

Balanced. No rotation needed.

Insert 8:

```
  5
 / \
4   8
```

Balanced. No rotation needed.

Insert 6:

```
    5
   / \
  4   8
     /
    6
```

Still balanced. No rotation needed.

Insert 12:

```
    5
   / \
  4   8
     / \
    6   12
```

Balanced.

Insert 11:

```
    5
   / \
  4   8
     / \
    6   12
       /
     11
```

Balanced.

Insert 9:

```
    5
   / \
  4   8
     / \
    6   12
       /
     11
    /
   9
```

Now node 12 is unbalanced (left-heavy).

- Subtree: 12 → 11 → 9
- Case: Left-Left imbalance → single right rotation at 12

After rotation:

```
    5
   / \
  4   8
     / \
    6   11
        / \
       9  12
```

Insert 7:

```
    5
   / \
  4   8
     / \
    6   11
     \  / \
      7 9  12
```

Now node 6 is unbalanced (right-heavy).
Subtree: 6 → 7
Case: Right-right imbalance → single left rotation at 6

After rotation

Final AVL tree:

```
    5
   / \
  4   8
     / \
    7   11
   /   / \
  6   9   12
```

Balanced.

---

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

```java
Position<E> ceilingEntry(K k) {
    Position<E> p = root();           // Start from the root
    Position<E> candidate = null;     // Best candidate so far

    while (p != null) {
        if (key(p) == k) {
            return p;
        } else if (key(p) < k) {
            p = right(p);             // Too small → go right
        } else {
            candidate = p;            // p could be the ceiling
            p = left(p);              // Try to find smaller candidate
        }
    }
    return candidate;                 // Return best ceiling found
}
```

---

(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)