12 第四讲 作业记录（二）

笔记本: 浙江大学《数据结构》
创建时间: 2025/4/6 16:36 **更新时间:** 2025/4/6 16:55
作者: panhengye@163.com
URL: https://pintia.cn/problem-sets/1873565885118418944/exam/problems/type/7?...

04-树5 Root of AVL Tree

提交结果

题目	用户	提交时间
04-树5	飞翔的小师弟	2025/04/06 16:36:22
编译器	**内存**	**用时**
Python (python3)	3248 / 65536 KB	15 / 400 ms
状态	**分数**	**评测时间**
答案正确	25 / 25	2025/04/06 16:36:23

评测详情

测试点	提示	内存(KB)	用时(ms)	结果	得分
0	fig 1 - LL	3036	15	答案正确	4 / 4
1	fig 2 - RR	3044	15	答案正确	4 / 4
2	fig 3 - RL	3148	15	答案正确	4 / 4
3	fig 4 - LR	3064	15	答案正确	4 / 4
4	深度LL旋转	3048	15	答案正确	4 / 4
5	最大N，深度RL旋转	3176	15	答案正确	4 / 4
6	最小N	3248	15	答案正确	1 / 1

【代码记录】

```
class AVLNode:
    def __init__(self, key):
        self.key = key
        self.height = 1
        self.left = None
        self.right = None


def height(node):
    if not node:
        return 0
    return node.height


def get_balance(node):
    if not node:
        return 0
    return height(node.left) - height(node.right)
```

```
def update_height(node):
    if not node:
        return
    node.height = max(height(node.left), height(node.right)) + 1


def right_rotate(y):
    x = y.left
    T2 = x.right

    # 执行旋转
    x.right = y
    y.left = T2

    # 更新高度
    update_height(y)
    update_height(x)

    # 返回新的根节点
    return x


def left_rotate(x):
    y = x.right
    T2 = y.left

    # 执行旋转
    y.left = x
    x.right = T2

    # 更新高度
    update_height(x)
    update_height(y)

    # 返回新的根节点
    return y


def insert(root, key):
    # 标准BST插入
    if not root:
        return AVLNode(key)

    if key < root.key:
        root.left = insert(root.left, key)
    elif key > root.key:
        root.right = insert(root.right, key)
    else:  # 键值已存在，AVL树不允许重复
        return root

    # 更新当前节点的高度
    update_height(root)

    # 获取平衡因子
    balance = get_balance(root)

    # 如果不平衡，则执行旋转

    # 左左情况
    if balance > 1 and key < root.left.key:
        return right_rotate(root)

    # 右右情况
    if balance < -1 and key > root.right.key:
        return left_rotate(root)

    # 左右情况
    if balance > 1 and key > root.left.key:
        root.left = left_rotate(root.left)
```

```
        return right_rotate(root)

    # 右左情况
    if balance < -1 and key < root.right.key:
        root.right = right_rotate(root.right)
        return left_rotate(root)

    # 返回未变更的节点指针
    return root


def main():
    n = int(input())
    keys = list(map(int, input().split()))

    root = None
    for key in keys:
        root = insert(root, key)

    print(root.key)


if __name__ == "__main__":
    main()
```

【整体思路】

1. **节点结构**:
   - 每个节点存储一个键值、高度信息以及左右子节点的引用
   - 高度定义为从该节点到最远叶子节点的最长路径长度
2. **平衡因子**:
   - 节点的平衡因子 = 左子树高度 - 右子树高度
   - 在AVL树中，每个节点的平衡因子必须在 {-1, 0, 1} 范围内
3. **旋转操作**:
   - 左旋：当右子树过高时使用
   - 右旋：当左子树过高时使用
   - 左右旋：先左旋左子树，再右旋当前节点
   - 右左旋：先右旋右子树，再左旋当前节点
4. **插入算法**:
   - 先按照普通二叉搜索树的方式插入节点
   - 自底向上更新节点高度和平衡因子
   - 根据平衡因子情况，选择合适的旋转操作来平衡树

【关键点解析1】

```
    # 左左情况
    if balance > 1 and key < root.left.key:
        return right_rotate(root)

    # 右右情况
    if balance < -1 and key > root.right.key:
        return left_rotate(root)

    # 左右情况
    if balance > 1 and key > root.left.key:
        root.left = left_rotate(root.left)
        return right_rotate(root)

    # 右左情况
    if balance < -1 and key < root.right.key:
```

```
        root.right = right_rotate(root.right)
        return left_rotate(root)
```

旋转模式	插入位置	调整方向	规律
LL	左子树过重	右旋	只转父亲，方向相反
RR	右子树过重	左旋	只转父亲，方向相反
LR	左子树的右子树过重	先左旋再右旋	先子再父，旋向同名
RL	右子树的左子树过重	先右旋再左旋	先子再父，旋向同名

【关键点解析2】

```
def right_rotate(y):
    x = y.left          # 设置x为y的左子节点
    T2 = x.right        # 保存x的右子树


    # 执行旋转
    x.right = y         # y成为x的右子节点
    y.left = T2         # x的原右子树成为y的左子树


    # 更新高度（先更新y再更新x，因为x现在是y的父节点）
    update_height(y)
    update_height(x)


    return x            # 返回新的根节点x
```

假设我们有以下树结构，其中Y是不平衡节点，X是Y的左子节点，T2是X的右子树：

旋转前

```
    Y
   / \
  X   T3
/ \
T1 T2
```

旋转后

```
    X
   / \
  T1  Y
     / \
    T2 T3
```