# avl平衡树基本操作的函数实现

2019年12月8日 14:01

## 左左单旋函数：

### 代码实现：

```
node* rotateLeft(node* root) {
    node* copy = root->left;
    root->left = copy->right;
    copy->right = root;
    return copy;
}
```

### 思路：

1. 先把root节点的左孩子复制在copy节点
2. root节点的左孩子位置存放copy节点的右孩子
3. copy节点的右孩子覆盖为root节点
4. 返回copy节点

## 左右双旋函数：

### 代码实现：

```
node* rotateLeftRight(node* root) {
    root->right = rotateLeft(root->right);
    return rotateRight(root);
}
```

### 思路：

1. 先对root节点的右孩子进行左左单旋
2. 步骤一后，root节点已变成右右单旋的情况
3. 对root节点进行右右单旋
4. 返回root节点

## 右右单旋函数：

### 代码实现：

```
node* rotateRight(node* root) {
    node* copy = root->right;
    root->right = copy->left;
    copy->left = root;
    return copy;
}
```

### 思路：

1. 先把root节点的右孩子复制到copy节点
2. root节点的右孩子位置存放copy节点的左孩子
3. copy节点的左孩子覆盖为root节点
4. 返回copy节点

## 右左双旋函数：

### 代码实现：

```
node* rotateRightLeft(node* root) {
    root->left = rotateRight(root->left);
    return rotateLeft(root);
}
```

### 思路：

1. 先对root节点的右孩子进行右右单旋
2. 步骤一后，root节点已变成左左单旋的情况
3. 对root节点进行左左单旋
4. 返回root节点

## 获取高度函数：

### 代码实现：

```
int getHeight(node* root) {
    if (root == NULL)
        return 0;
    return max(getHeight(root->left), getHeight(root->
    right)) + 1;
}
```

### 思路：

1. 递归出口为：当节点为空节点时，即没有高度，返回0；
2. 递归主体为：取左子树与右子树的中高度较高的+1作为高度返回

## 插入元素函数：

### 代码实现：

```
node* insert(node* root, int value) {
    if (root == NULL) {
        root = new node();
        root->data = value;
        root->left = root->right = NULL;
    }
    else if (value > root->data) {
        root->right=insert(root->right, value);
        if (getHeight(root->right) - getHeight(root->
        left) >= 2)
            root = value > root->right->data ?
            rotateRight(root) : rotateLeftRight(root);
    }
    else {
        root->left = insert(root->left, value);
        if (getHeight(root->left) - getHeight(root->
        right) >= 2)
            root = value < root->left->data ?
            rotateLeft(root) : rotateRightLeft(root);
    }
    return root;
}
```

### 思路：

1. 递归出口：当节点为空，开辟节点并存放数据
2. 递归主体1：当数据大于root节点的数据时，把数据插入右子树
   此时判断左右子树高度差，如果达到2，说明不平衡
   此时判断数据是插在了右子树的左边还是右边
   左边则做一次左右双旋，右边则做一次右右单旋
3. 递归主体2：当数据小于root节点的数据时，把数据插入左子树
   此时判断左右子树高度差，如果达到2，说明不平衡
   此时判断数据是插在了左子树的左边还是右边
   左边则做一次左左双旋，右边则做一次右左单旋
4. 最后返回插入结果