基础算法与数据结构（五） 树（4）平衡二叉树之AVL树

AVL树的定义

1. 平衡因子：δ(T) = |R| - |L| 若δ(T) = 0 那么这棵树是平衡的，其绝对值越小，说明树越平衡。
2. AVL树的定义： | δ(T) | ≤ 1

AVL树的性质

1. 一棵n个结点的AVL树的其高度保持在O(log2(n)),不会超过(3/2)*log2(n+1)
2. 一棵n个结点的AVL树的平均搜索长度保持在O(log2(n)).
3. 一棵n个结点的AVL树删除一个结点做平衡化旋转所需要的时间为O(log2(n)).

AVL的数据结构

基本同红黑树，把其中结点的染色编程了左右结点的高度差。

```
struct Node {
    Key key;
    int delta;
    Node *left, *right, *parent;
    Node(Key k) : key(k), delta(0), left(nullptr), right(nullptr),
parent(nullptr) {}

    virtual ~Node() {
        delete left;
        delete right;
    }

    void setLeft(Node* x) {
        left = x;
        if (x) x->parent = this;
    }

    void setRight(Node* x) {
        right = x;
        if (x) x->parent = this;
    }
```

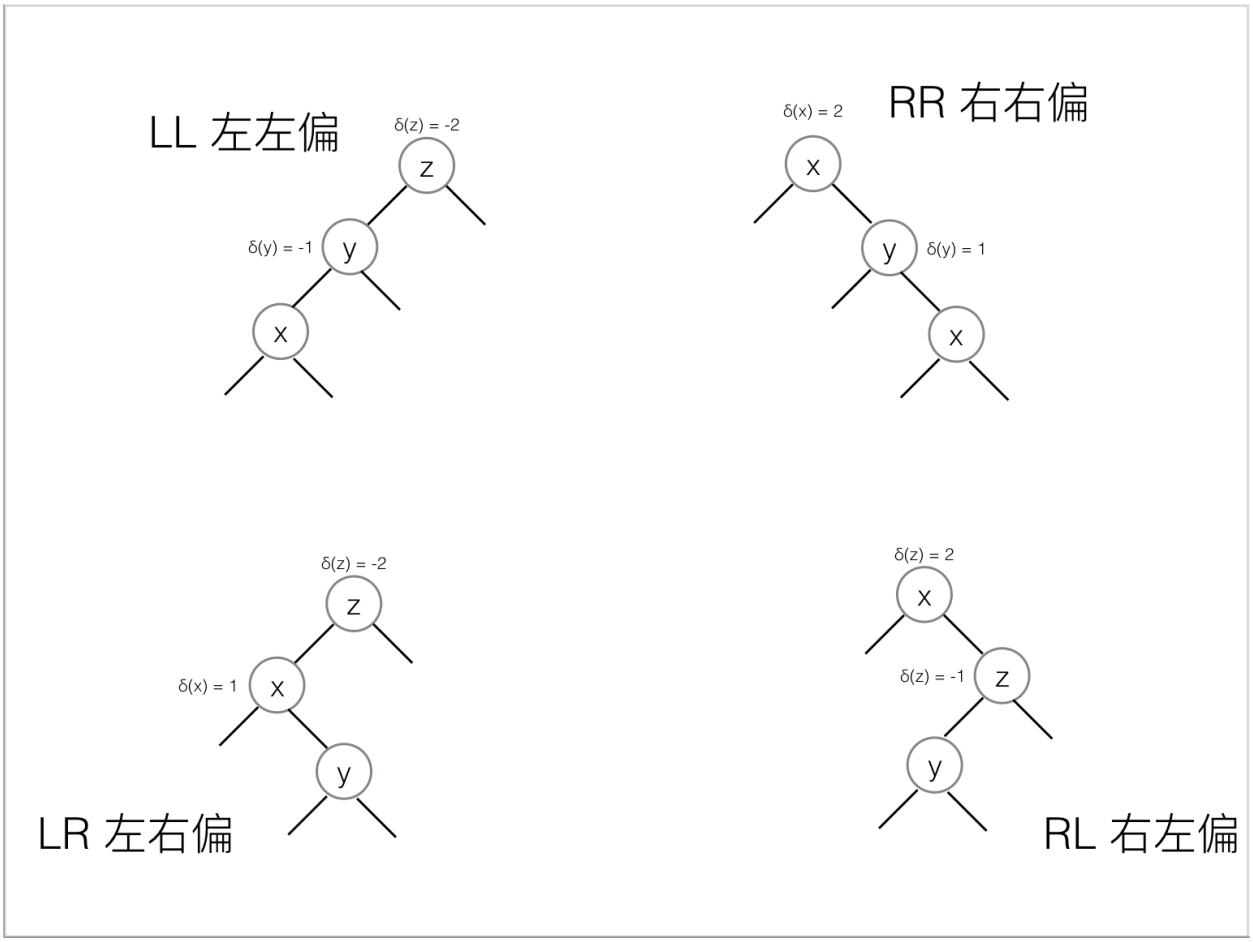
```
    void setChildren(Node* x, Node* y) {
        setLeft(x);
        setRight(y);
    }

    // parent <--> this  ==> parent <--> y
    void replaceWith(Node* y) {
        if (!parent) {
            if (y) y->parent = nullptr;
        } else if (parent->left == this) {
            parent->setLeft(y);
        } else {
            parent->setRight(y);
        }
        parent = nullptr;
    }
};
```

AVL树的插入

AVL树的插入主要难点同样在处理高度差大于1之后的问题。要处理这个问题就要进行树的旋转来的满足高度差小于1。

分为几种情况，转换如下：



LL偏：

导致最上方结点需要旋转的原因是左子树比右子树高度大1，且左子树的左子树比右子树大1，此时成为左-左偏

剩余的三种偏法看图容易理解。

偏的处理方法主要靠树的旋转，思想见代码。

```
Node* insertFix(Node* t, Node* x) {
    /*
     * denote d = delta(t), d' = delta(t'),
     *   where t' is the new tree after insertion.
     *
     * case 1: |d| == 0, |d'| == 1, height increase,
     *    we need go on bottom-up updating.
     *
     * case 2: |d| == 1, |d'| == 0, height doesn't change,
     *    program terminate
     *
     * case 3: |d| == 1, |d'| == 2, AVL violation,
     *    we need fixing by rotation.
```

```
     */
    int d1, d2, dy;
    Node *p, *l, *r;
    while (x->parent) {
        d2 = d1 = x->parent->delta;
        d2 += x == x->parent->left ? -1 : 1;
        x->parent->delta = d2;
        p = x->parent;
        l = x->parent->left;
        r = x->parent->right;
        if (abs(d1) == 1 && abs(d2) == 0) {
            return t;
        } else if (abs(d1) == 0 && abs(d2) == 1) {
            x = x->parent;
        } else if (abs(d1) == 1 && abs(d2) == 2) {
            if (d2 == 2) {
                if (r->delta == 1) { // right-right case
                    p->delta = 0;
                    r->delta = 0;
                    t = leftRotate(t, p);
                } else if (r->delta == -1) { // right-left case
                    dy = r->left->delta;
                    p->delta = dy == 1 ? -1 : 0;
                    r->left->delta = 0;
                    r->delta = dy == -1 ? 1 : 0;
                    t = rightRotate(t, r);
                    t = leftRotate(t, p);
                }
            } else if (d2 == -2) {
                if (l->delta == -1) { // left-left case
                    p->delta = 0;
                    l->delta = 0;
                    t = rightRotate(t, p);
                } else if (l->delta == 1) { // left-right case
                    dy = l->right->delta;
                    l->delta = dy == 1 ? -1 : 0;
                    l->right->delta = 0;
                    p->delta = dy == -1 ? 1 : 0;
                    t = leftRotate(t, l);
```

```
                    t = rightRotate(t, p);
                }
            }
            break;
        } else {
            printf("shouldn't be here. d1=%d, d2=%d\n", d1, d2);
            assert(false);
        }
    }
    return t;
}
```