数据结构与算法I 作业10

2019201409 于倬浩

12.3-6

```
Tree-Delete(T, z)
    if z.left == NIL
        Transplant(T, z, z.right)
    elseif z.right == NIL
        Transplant(T, z, z.left)
    else y = Tree-Maximum(z.left) //即改为在左子树中寻找前驱
        if y.p ≠ z
            Transplant(T, y, y.left) //Transplant需要改为为原方法的对称情况
            y.left = z.left //改为移动左子树
            y.left.p = y
        Transplant(T, z, y)
        y.right = z.right
        y.right.p = y
```

若改为前驱后继优先级相同的情况，可以在每次调用`Tree-Delete`的第三种情况前随机一个布尔变量，为0则执行寻找左子树的最大值，为1则执行寻找右子树的最小值。只需将原来的寻找后继的算法和以上代码合并起来即可。

12-1

- a. 若执行书中提到的`Tree-Insert`算法，则二叉树一定退化成每个节点只有右子树的一条链，树的高度退化为$\Theta(n)$，因此单次操作渐进时间复杂度为$\Theta(n)$，总时间复杂度$\Theta(n^2)$。
- b. 这种情况下，每次插入新的元素都会进入一个不同的子树，因此二叉树的形态类似满二叉树，树的高度维持在$\Theta(lgn)$，因此单次操作渐进时间复杂度为$\Theta(lgn)$，总时间复杂度$\Theta(nlgn)$。

- c. 此时树中只有树根一个节点。每次插入新元素，都会直接插入根的列表中。若单次插入列表的时间复杂度为$\Theta(1)$，那么单次操作的渐进时间复杂度为$\Theta(1)$，总时间复杂度$\Theta(n)$。
- d. 此时最坏情况下和朴素算法相同，即全部节点均为左子树/右子树，渐进时间复杂度为$\Theta(n)$；平均情况下，类似情况b，新节点插入两个子树的概率相同，树的期望深度为$\Theta(lgn)$，单次操作的期望时间复杂度为$\Theta(lgn)$，总时间复杂度$\Theta(nlgn)$。