**Lab5实验报告**

1. 总体思路

代码中用一个类实现搜索树，封装了树的各种方法。树的节点用结构体node实现，node有整型的value和key，以及3个node指针，分别为parent，left，right，分别指向父母、左孩子、右孩子。

二、部分函数的解释

（一）在删除节点的函数里

需要注意要分情况讨论

1. 没有孩子：直接删
2. 只有一个孩子：这个孩子顶上去
3. 有2个孩子：找successor去替换。（其中successor就是右孩子的情况也比较特殊，需要单独考虑，代码中有写）

注：

1. 注意在更换节点时parent的孩子指针域、孩子节点的parent指针域也要改变，容易忽略。2、另外，我的代码中替换节点是直接更改了其节点的父、子的指针指向，其实也可以直接改变要删除节点的key和value，改为要替换的节点的key和value，然后删除要替换的节点即可，这样可以避免更改要删除的节点的父、子的指针域。

（二）在三种遍历树得到value的序列的函数中

因为要得到一个vector<int>存储着value序列，所以要不断递归调用时传递这个vector，也可以在类里定义一个私有的vector<int>，这样在遍历时就不用传递vector<int>了，只是每次遍历前要清空这个vector。但是本次代码并没有采取这种方法，以上只是提供另一种方法的思路。

为了递归调用，所以还写了

//以下三个函数用于traversal\_函数的遍历：

vector<int> preorder(node\* p,vector<int> result);

vector<int> inorder(node\* p,vector<int> result);

vector<int> postorder(node\* p,vector<int> result);

但是由于其中没有用到类中的private数据，为了简化类的方法，所以设为了非成员函数（但是出于分装考虑也应该设为成员函数，这里并没有修改）。

而

    vector<int> traversal\_pre();//先序遍历

    vector<int> traversal\_in();//中序遍历

    vector<int> traversal\_post();//后序遍历

都设为了成员函数，因为用到了树的头结点head。

（三）其他函数解释具体见代码以及注释

1. 其他说明

说明文档中大部分函数要求返回的都是所找到的节点的value，为了复用性，我写了返回所找到的节点地址的函数，然后文档中的函数调用我自己写的返回指针的函数。