基础算法与数据结构（三） 树（2）二叉搜索树

二叉搜索树的定义

二叉搜索树（BST）每个结点均含有一个可比较的key和对应的value，左侧子树所有的key都比根节点小，右侧子树的所有key均比根节点大。

```
struct TreeNode{
  int key;
  int value;
  TreeNode* left;
  TreeNode* right;
  TreeNode(int key,int value) {
      this.key = key;
      this.value = value;
  }
}
```

二叉搜索树的性质

1. 二叉搜索树的中序遍历是有序的。（可以用于判断一个树是否为二叉搜索树）

二叉搜索树的操作

查找操作

用二叉搜索树为的就是搜索（有点废话的意思），利用二叉树可以将有序内容的查找的平均时间复杂度从O(n)降到O(lgn)，当然在最坏情况下如果树退化成了一个链表（均只有左子树或者右子树），那么就完全没有体现出树的作用。此时就要通过另一种在构造树时候进行平衡的方法来简化了。

废话不多说，上代码

```
int getValue(TreeNode* root,int key){
      if(!root) return null;
```

```
        if(key < root->key) return getValue(root->left,key);
        else if (key > root->key) return getValue(root->right,key)
        else return root->value;
}
```

插入操作

和查找操作类似，根据大小关系向下寻找，找到第一个符合条件的位置即可插入操作，返回一个新的结点的地址。
如果已经有key和要插入的key相同，那么默认更新这个值。

```
TreeNode* insert(TreeNode* root,int key,int value){
        if(!root) return new TreeNode(key,value);
        if(key < root->key) return insert(root->left,key,value);
        else if (key > root->key) return getValue(root->right,key,value)
        else{
            root->value = value;
            return root;
        };
}
```

删除操作

删除操作只要需要克服的问题就是再删除一个结点之后如何保持这个树的继续有序，即如何处理只能有一个接口但有两棵原先结点存在的子树。
1962年T.Hibbard提出了一个方法，就是从删除结点的右子树中寻找最小值作为这个删除结点的替代者。原因也很好理解，因为左子树一定比这个要替代的结点的key小，所以要在右子树中找出最小的，才能满足大于左子树，小于右子树。
同理，其实往左子树中寻找最大结点也是可以的，这两者应该随机的选用，这里展示前者的代码实现。

```
TreeNode* getMin(TreeNode* root){
        if(!root->left) return root;
        return getMin(root->left);
}

TreeNode* deleteMin(TreeNode* root){
```

```
        if(!root->left) return root->right;
        root->left = deleteMin(root->left);
        return root;
}

TreeNode*  deleteNode(TreeNode* root,int key){
        if(!root) return NULL;
        if(key < root->key)
            root->left = deleteNode(root->left,key);
        else if (key > root->key)
            root->right = deleteNode(root->right,key);
        else{
            //要删除的节点只有一个子树，直接返回。
            if(!root->right) return root->left;
            if(!root->left)  return root->right;
            //有两个子树，那么找到右子树中最小的,然后删除这个最小的值
            TreeNode* t = root;
            root = getMin(t->right);
            root->right = deleteMin(t->right);
            root->left = t->right;
        }
        return root;
}
```

典型问题

更新至2017.4.23（v1.0）

LeetCode 98

判断一棵树是否为有效的二叉搜索树。

利用之前的性质，进行中序遍历，如果遍历结果有序，那么就合法，否则不合法。

代码略。

LeetCode 108

利用同样的思想可以重建树。