《数据结构与算法》作业五、六

姓名：余弦 学号：2021112893 班级：2103401

1.二叉查找树

1)基本结构

```
typedef struct node{
    int data;
    node* left;
    node* right;
}Node;
```

2)插入某一元素

通过比较，比现在这个指针所指的元素小的元素插入到这个节点的左边，大的就插入到这个节点的右边，递归进行

```
    void insert(int n){
        if(root == nullptr){
            Node* n_node = new Node;
            n_node -> data = n;
            n_node -> left = nullptr;
            n_node -> right = nullptr;
            this -> root = n_node;
            return;
        }
        Node* p = this -> root;
        while(1){
            if(!p -> left && n < p -> data){
                Node* nnode = new Node;
                nnode -> data = n;
                nnode -> left = nullptr;
                nnode -> right = nullptr;
                p -> left = nnode;
                return;
            }
            else if(!p -> right && n >= p -> data){
                Node* nnode = new Node;
                nnode -> data = n;
                nnode -> left = nullptr;
                nnode -> right = nullptr;
                p -> right = nnode;
                return;
            }
            else if(p -> left && n < p -> data){
                p = p -> left;
                continue;
            }
            else if(p-> right && n >= p -> data){
                p = p -> right;
```

```
                continue;
            }
        }
    }
```

3)查找某一元素

```
    int BST_find(int n, int cnt){
        using namespace std;
        Node* p = this -> root;
        while(p){
            if(n == p -> data){
            return cnt;
            }
            if(n < p -> data) {
                cnt++;
                p = p->left;
            }else{
                cnt++;
                p = p -> right;
            }
        }
        return cnt;
    }
```

4)删除某一元素

若元素不在二叉查找树中则直接返回

若元素的节点为叶节点，则直接删除

若元素的节点只存在一个儿子节点（只存在左儿子或者右儿子），则用儿子节点代替当前节点

若儿子节点两个儿子都存在，则找到这个节点左子树的最大元素，替代当前节点的元素，并在左子树中删除这个最大元素

```
Node* BST_delete_in(Node* p,int n){
        using namespace std;
        if(!p) return p;
        if(n < p -> data){
            p -> left = BST_delete_in(p -> left, n);
            return p;
        }
        if(n > p -> data){
            p -> right = BST_delete_in(p -> right, n);
            return p;
        }
        // ///////
        if(!p -> left && !p -> right){
            delete p;
            return nullptr;
        }
        if(p -> left && !p -> right){
            Node* res = p -> left;
            //delete p;
```

```
            return res;
        }
        if(p -> right && !p -> left){
            Node* res = p -> right;
            //delete p;
            return res;
        }
        if(p -> left && p -> right){
            int m = Find_max(p -> left);
            p -> data = m;
            p -> left = BST_delete_in(p-> left,m);
            return p;
        }
    }
    int Find_max(Node* p){
        if(!p) return 0;
        if(!p->right) return p->data;
        return Find_max(p -> right);
    }
    void BST_delete(int n){
        this -> root = BST_delete_in(this->root, n);
    }
```

5)二叉查找树的排序算法

只需要中序遍历这个二叉查找树就可以完成排序操作

```
   void sort(Node* p){
       using namespace std;
       if(!p) return;
       sort(p -> left);
       cout << p -> data << ",";
       sort(p-> right);

   }
```

2.折半查找

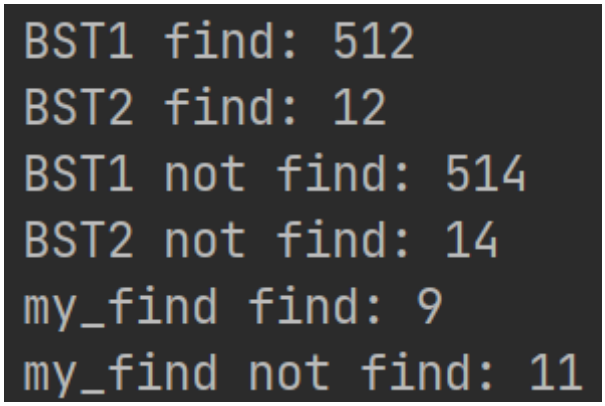
```
int my_find(int data_set[],int l, int r, int n,int cnt){
    using namespace std;
    if(l > r){
//        cout << "Not found!" <<endl;
        return cnt;
    }
    int mid = (l+r)>>1;
    if(data_set[mid] == n){
//        cout << "Fount it!"<<endl;
        return cnt;
    }
    if(data_set[mid] > n){

        return my_find(data_set, l, mid - 1, n, cnt + 1);
    }
    if(data_set[mid] < n){
```

```
17
18          return my_find(data_set, mid + 1, r, n,cnt + 1);
19      }
20  }
```

3.平均查找次数的比较

首先生成0-2048的有序随机数，添加到BST树中，然后打乱之后添加到另外一个BST中。

结果如下



BST1是有序序列构建的二叉查找树

BST2是无序序列构建的二叉查找树

题目中要求使用中序遍历顺序进行查找，所以只需要对data_set进行一次排序再进行查找即可。

可以看出在平均情况下，二叉查找树的效率和折半查找的效率相当。