

《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

```
1  typedef struct node{
2      int data;
3      node* left;
4      node* right;
5  }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

```
1  void insert(int n){
2      if(root == nullptr){
3          Node* n_node = new Node;
4          n_node -> data = n;
5          n_node -> left = nullptr;
6          n_node -> right = nullptr;
7          this -> root = n_node;
8          return;
9      }
10     Node* p = this -> root;
11     while(1){
12         if(!p -> left && n < p -> data){
13             Node* nnode = new Node;
14             nnode -> data = n;
15             nnode -> left = nullptr;
16             nnode -> right = nullptr;
17             p -> left = nnode;
18             return;
19         }
20         else if(!p -> right && n >= p -> data){
21             Node* nnode = new Node;
22             nnode -> data = n;
23             nnode -> left = nullptr;
24             nnode -> right = nullptr;
25             p -> right = nnode;
26             return;
27         }
28         else if(p -> left && n < p -> data){
29             p = p -> left;
30             continue;
31         }
32         else if(p -> right && n >= p -> data){
33             p = p -> right;
```

```

34         continue;
35     }
36 }
37 }

```

3)查找某一元素

```

1  int BST_find(int n, int cnt){
2      using namespace std;
3      Node* p = this -> root;
4      while(p){
5          if(n == p -> data){
6              return cnt;
7          }
8          if(n < p -> data) {
9              cnt++;
10             p = p->left;
11         }else{
12             cnt++;
13             p = p -> right;
14         }
15     }
16     return cnt;
17 }

```

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

```

1  Node* BST_delete_in(Node* p,int n){
2      using namespace std;
3      if(!p) return p;
4      if(n < p -> data){
5          p -> left = BST_delete_in(p -> left, n);
6          return p;
7      }
8      if(n > p -> data){
9          p -> right = BST_delete_in(p -> right, n);
10         return p;
11     }
12     // ///////
13     if(!p -> left && !p -> right){
14         delete p;
15         return nullptr;
16     }
17     if(p -> left && !p -> right){
18         Node* res = p -> left;
19         //delete p;

```

```

20         return res;
21     }
22     if(p -> right && !p -> left){
23         Node* res = p -> right;
24         //delete p;
25         return res;
26     }
27     if(p -> left && p -> right){
28         int m = Find_max(p -> left);
29         p -> data = m;
30         p -> left = BST_delete_in(p-> left,m);
31         return p;
32     }
33 }
34 int Find_max(Node* p){
35     if(!p) return 0;
36     if(!p->right) return p->data;
37     return Find_max(p -> right);
38 }
39 void BST_delete(int n){
40     this -> root = BST_delete_in(this->root, n);
41 }

```

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

```

1     void sort(Node* p){
2         using namespace std;
3         if(!p) return;
4         sort(p -> left);
5         cout << p -> data << ", ";
6         sort(p-> right);
7
8     }

```

2.折半查找

```

1     int my_find(int data_set[],int l, int r, int n,int cnt){
2         using namespace std;
3         if(l > r){
4             // cout << "Not found!" <<endl;
5             return cnt;
6         }
7         int mid = (l+r)>>1;
8         if(data_set[mid] == n){
9             // cout << "Fount it!"<<endl;
10            return cnt;
11        }
12        if(data_set[mid] > n){
13
14            return my_find(data_set, l, mid - 1, n, cnt + 1);
15        }
16        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 find: 512  
BST2 find: 12  
BST1 not find: 514  
BST2 not find: 14  
my_find find: 9  
my_find not find: 11
```

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。