

作业 5 查找结构与排序方法

1. 设计 BST 的左右链存储结构, 并实现 BST 插入、删除、查找和排序算法。

首先定义 BST 树结点的数据结构, 包含数据域以及左右儿子指针域, 具体如下:

```
5  typedef struct node
6  {
7      int data;
8      struct node *lchild,*rchild;
9  }BSTNode;
10 typedef BSTNode *BST;
```

而后 BST 的插入、删除、查找和排序均可以采用递归的思想

A. BST 插入

判断新节点在当前结点的左子树还是右子树上, 递归至空时申请新节点。

```
11 void InsertBST(BST &T,int x)
12 {
13     if(T == NULL){
14         T = new BSTNode;
15         T->data = x;
16         T->lchild = NULL;
17         T->rchild = NULL;
18     }
19     else if(x < T->data)
20         InsertBST(T->lchild, x);
21     else if(x > T->data)
22         InsertBST(T->rchild, x);
23 }
```

B. BST 建立

读取文件数据, 重复调用插入函数即可

```
25 void createBST(BST &T, char path[])
26 {
27     ifstream infile;
28     infile.open(path);
29     int x;
30     while(infile>>x)
31         InsertBST(T,x);
32     infile.close();
33 }
```

C. BST 查找

将待查数据与当前指针数据进行比较, 从而决定向左或向右遍历, 如若数据匹配或查找至空域 (查找失败) 即返回当前指针。引入 count 对查找长度进行记录。

```

49  BST search(int k, BST &T, int *count)
50  {
51      BST p = T;
52      (*count)++;
53      if((p == NULL) || (k == p->data))
54          return p;
55      if(k < p->data)
56          return (search(k, T->lchild, count));
57      else if(k > p->data)
58          return (search(k, T->rchild, count));
59  }

```

D. BST 删除

遍历至 BST 树的最左节点将其删除，并将其右子树继承至其父节点上。

```

35  int deletemin(BST &T)
36  {
37      int tmp; BST p;
38      if(T->lchild == NULL){
39          p = T;
40          tmp = T->data;
41          T = T->rchild;
42          delete p;
43          return tmp;
44      }
45      else
46          return (deletemin(T->lchild));
47  }

```

E. BST 排序

由于 BST 树的性质，其中序遍历序列即为其排序序列。

```

61  void Sort(BST T)
62  {
63      if(T != NULL){
64          Sort(T->lchild);
65          cout<<T->data<<endl;
66          Sort(T->rchild);
67      }
68  }

```

2. 实现折半查找算法。

与 BST 的思想类似，对有序序列按照待查找值与当前中值比较后递归比较前半部分或者后半部分。

```

70 int BinSearch(int x[], int low, int up, int k, int *count)
71 {
72     if(low > up) return 0;
73     else{
74         (*count)++;
75         int mid = (low + up)/2;
76         if(k < x[mid])
77             return BinSearch(x, low, mid-1, k, count);
78         else if(k > x[mid])
79             return BinSearch(x, mid+1, up, k, count);
80         else return mid;
81     }
82 }

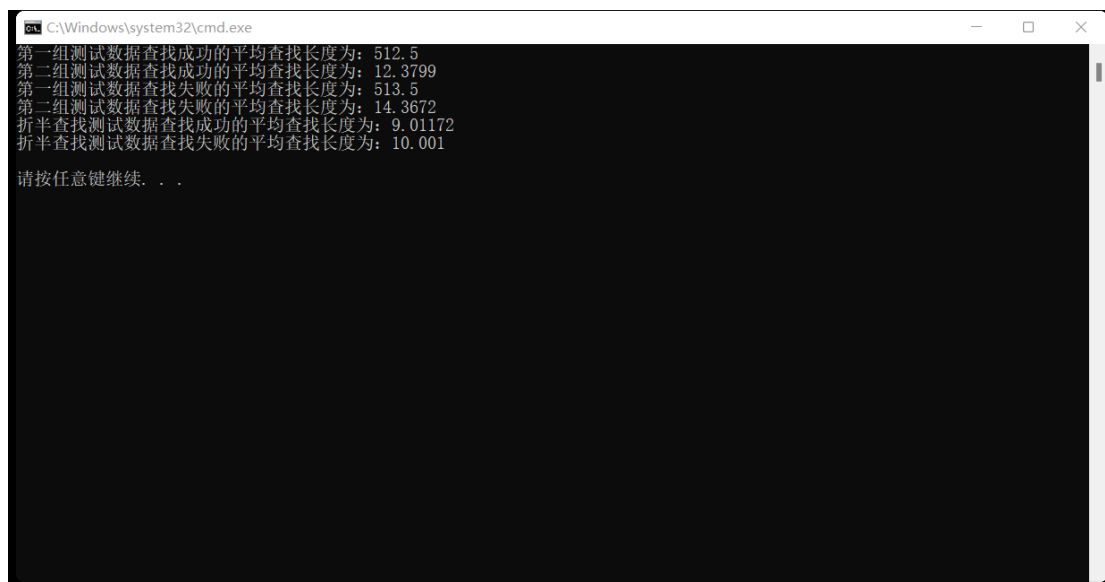
```

3. 实验比较：设计并产生实验测试数据，考察比较两种查找方法的时间性能，并与理论结果进行比较。

第一组数据：0 至 2048 之间的已排序的奇数序列

第二组数据：第 1 组测试数据的随机序列。

实验结果具体如下：



```

C:\Windows\system32\cmd.exe
第一组测试数据查找成功的平均查找长度为: 512.5
第二组测试数据查找成功的平均查找长度为: 12.3799
第一组测试数据查找失败的平均查找长度为: 513.5
第二组测试数据查找失败的平均查找长度为: 14.3672
折半查找测试数据查找成功的平均查找长度为: 9.01172
折半查找测试数据查找失败的平均查找长度为: 10.001
请按任意键继续. . .

```

理论情况下，第一组数据存入 BST 树时退化为单链表，但由于其有序性，可得查找成功的平均查找长度为 $\frac{\sum_{i=1}^n i}{n} = \frac{n+1}{2}$ ，查找失败的平均查找长度为 $\frac{n+1}{2} + 1$ ，与实验结果相吻合；第二组数据由于其随机性，其平均成功查找长度取决于二叉排序树的实际形态，取值范围为 $\log_2 n \sim \frac{n+1}{2}$ ，实验结果也表明随机性越高，BST 的查找效率越高。

而折半查找的平均成功查找长度为 $\frac{n+1}{n} \log_2(n+1) - 1$ ，查找失败的平均查找长度约为树的高度 $\log_2(n+1)$ ，实验结果显然也符合理论情形。

以上实验并不能完全说明就平均性能而言，BST 的查找与折半查找差不多，原因是 BST 的查找严重依赖于数据的随机性，在有序序列的情形下，折半查找是远远优于 BST 查找方式的，而在随机性较强的情况下，二者的查找性能差距较小，但折半查找需要满足有序性，因而很难说两种查找方式谁更优秀，实际情况实际分析，但就维护表的有序性而言，BST 的方式更为有效。