湖南大學

数据结构

课程实验报告

题　　目：基于 BST 实现静态查找表

学生姓名：魏子铖

学生学号：201726010308

专业班级：软件 1703

完成时间：2018.11.26

一、需求分析

0) 问题分析

查找的定义：给定一个值 k，在含有 n 个结点的表（或文件）中找出关键字等于给定值 k 的结点，若找到，则查找成功，输出该结点在表中的位置；否则查找失败，输出查找失败的信息。

静态查找表：仅对查找表进行查找操作，而不改变查找表中的数据元素。

现在我们给定一个静态查找表，对表中的数据执行查找操作。

需要实现的功能有：

i. 建立静态查找表。
ii. 实现静态查找表的查找功能。
iii. 对查找结果进行格式化输出。

1) 输入数据

【输入格式】

输入分两个部分，一部分是构建静态查找表的输入，一部分为测试查找功能的输入

- 第一行输入一个整数，表示静态查找表中的元素个数。
- 第二行输入 n 个整数，表示静态查找表中的每个元素值，用空格间隔。
- 接下来每一行输入一个整数，代表一次查找操作，输入-1 值来结束查找操作。

【输如样例】

9
37 24 42 7 32 40 45 2 120
37
42
7
45
120
99
36
-1

2) 输出数据

【输出格式】

对于每一次查找操作，输出查找成功或失败，以及查找次数。

【输出样例】

查找成功，查找次数为 1
查找成功，查找次数为 2
查找成功，查找次数为 3
查找成功，查找次数为 3
查找成功，查找次数为 4
查找失败，查找次数为 5
查找失败，查找次数为 4

3) 测试样例设计

样例一

样例输入

5

10 25 36 11 49
36
11
15
-1

样例输出

查找成功，查找次数为 3
查找成功，查找次数为 3
查找失败，查找次数为 4

设计理由：一般情况

样例二

样例输入

7
50 25 75 2 33 62 130
50
62
70
-1

样例输出

查找成功，查找次数为 1
查找成功，查找次数为 3
查找失败，查找次数为 4

设计理由：将 BST 设计成完全二叉树，查询效率最高

样例三

样例输入

5
11 22 33 44 55
33
55
66
-1

样例输出

查找成功，查找次数为 3
查找成功，查找次数为 5
查找失败，查找次数为 6

设计理由：将 BST 设计成链式结构，查询效率最低

样例四

样例输入

1
60
60
61
-1

样例输出

查找成功，查找次数为 1
查找失败，查找次数为 2
设计理由：BST 中只有一个结点的特殊情况

样例五

样例输入

7
45 50 20 45 50 20 45
45
50
20
-1

样例输出

查找成功，查找次数为 1
查找成功，查找次数为 2
查找成功，查找次数为 2
设计理由：BST 中出现多个重复元素，则只会查找到离根结点最近的元素

二、概要设计

1. 抽象数据类型

为实现上述功能，我们假设静态查找表中的数据都是整数，使用 int 类型来存储用户的输入，并将用户输入的值存储在 BST 相应的结点中。

抽象数据类型设计：

- 数据对象：一个整数，存在于静态查找表中。
- 数据关系：每个整数都是 BST 的一个结点，该结点有一个左结点和一个右结点。
- 基本操作：在构建 BST 时插入元素；静态查找表中的元素是否存在；获取静态查找表中元素的个数。
- **ADT：**

```
StaticSearchTable {
数据对象：D = { key_i | key∈N, i = 1, 2, 3, ……, n, 1≤n≤1000 }
数据关系：R = { key | key∈BST , key's leftchild < key < key's rightchild}
基本操作：
    void clear();
    //清空静态查找表中的元素，时间复杂度 O（n）
    void insert(const int&);
    //向静态查找表中插入元素，时间复杂度 O（logn）
    int find(const int&) const;
    //查找静态查找表中的元素，时间复杂度 O（logn）
    int size();
    //获取静态查找表中的元素个数，时间复杂度 O（1）
}
```

2. 算法的基本思想

对于各种功能：

1) 建立静态查找表
   基于 BST 来建立静态查找表，表中的每个元素值为整数。
2) 实现静态查找表的查找功能

在 BST 中执行递归查找，并根据递归深度返回查找次数。

3) 对查找结果格式化输出

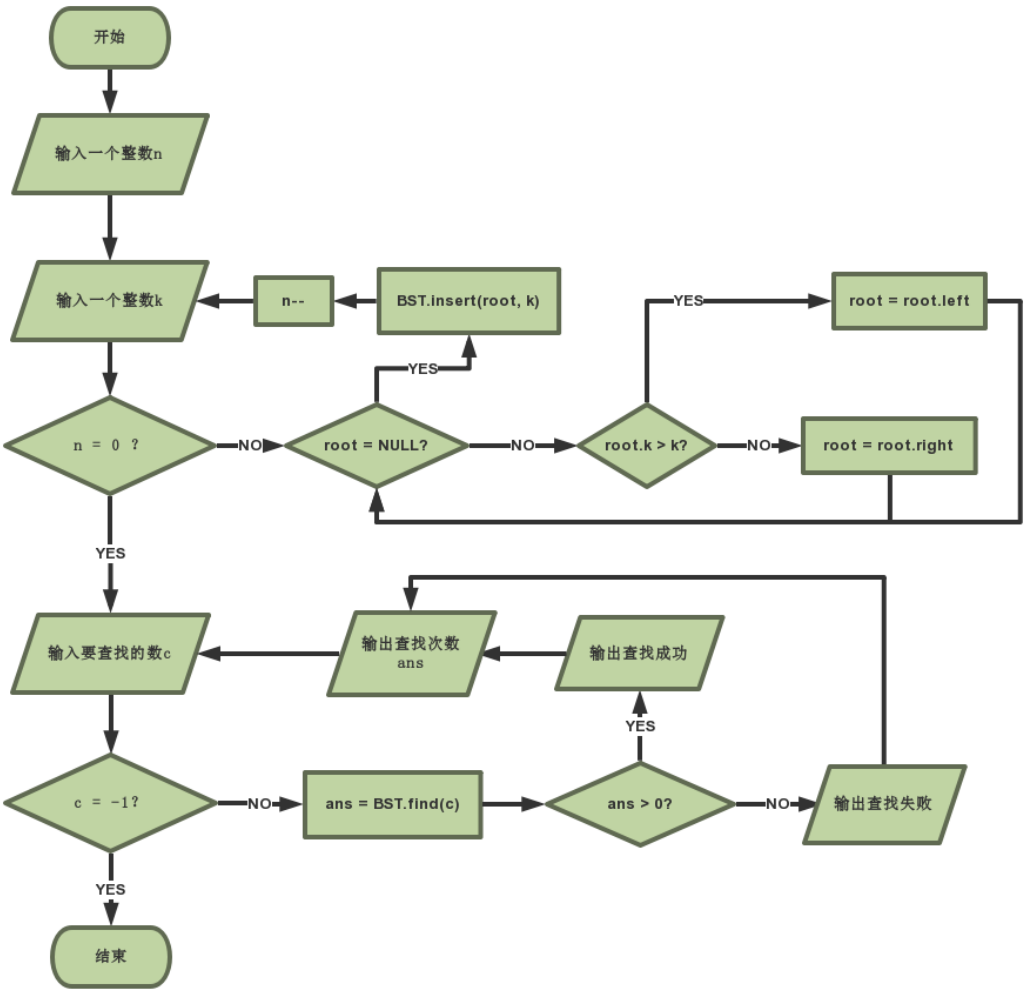
输出是要判断查找是否成功，输出相应的信息。

3. 程序的流程

程序由四个模块组成：

1) 输入模块：提示输入格式以及结束标志，输入分两个部分，一部分是构建静态查找表的输入，一部分为测试查找功能的输入。
2) 构建 BST：以链表形式实现 BST 的构建。
3) 查找模块：在 BST 中执行递归查找，返回值为查找次数。
4) 输出模块：格式化输出查找信息。

程序流程图如下：



三、详细设计

1. 物理数据类型

输入的数据为静态查找表的信息与查找信息，都是整数类型，由于静态查找表是基于 BST 的，满足树形特征，所以逻辑实现上可以采用链表的形式。

2. 输入和输出的格式

输入时有提示语句，说明输入的放法与结束条件，将每个元素存储在同一个 BST 中，形成一个静态查找表，并将查找结果格式化输出。

3. 算法的具体步骤

1) 基于 BST 建立静态查找表

对于二叉检索树的任何一个结点，该结点左子树的任何一个结点的值都小于该结点；该结点右子树任意一个结点的值都大于或等于该结点。要插入一个值，首先必须找出它应该放在树结构的什么地方。

```
Node* BST::insertHelp(Node* node, const int& k) {
    if (node == NULL) {
        return new Node(k);
        //树结构的该位置为空，则在该位置生成一个新结点，值为 k
    }
    if (k < node->getKey()) {
        node->setLeft(insertHelp(node->left(), k));
        //如果 k 小于该结点的值，则待插入的结点应该在该结点的左子树上
    }
    else {
        node->setRight(insertHelp(node->right(), k));
        //如果 k 大于等于该结点的值，则待插入的结点应该在该结点的右子树上
    }
    return node;
}
```

2) 查找表中的元素是否存在

比较待查找的值 k1 与当前结点的值 k2，若：

- k1 < k2：在该结点的左子树中查找。
- k1 = k2：查找成功。
- k1 > k2：在该结点的右子树中查找。
- k2 = NULL：查找失败。

```
int BST::findHelp(Node* node, const int& k) const {
    if (node == NULL) {
        return -1;
        //查找失败，返回负数
    }
    if (k < node->getKey()) {
        return findHelp(node->left(), k) * 2;
        //待查找的值小于当前节点的值，则在该结点的左子树中查找
    }
    else if (k > node->getKey()){
        return findHelp(node->right(), k) * 2;
        //待查找的值大于等于当前节点的值，则在该结点的右子树中查找
    }
    else {
        return 1;
        //查找成功，返回正数
    }
}
```

3) **格式化输出查找结果**

根据 find()函数返回值的正负判断查找是否成功。

```
int ans = StaticSearchTable.find(key);
if (ans > 0) {
    //结果为正数，表明 key 在静态查找表中
    std::cout << "查找成功，查找次数为" << ans << std::endl;
}
else {
    //结果为负数，表明 key 不在静态查找表中
    std::cout << "查找失败，查找次数为" << -ans << std::endl;
}
```

4. 算法的时空分析

1) 清空静态查找表中的元素，时间复杂度 O（n）。
2) 向静态查找表中插入元素，时间复杂度 O（logn）。
3) 查找静态查找表中的元素，时间复杂度 O（logn）。
4) 获取静态查找表中的元素个数，时间复杂度 O（1）。

四、调试分析

1.调试方案设计

调试目的：发现思维逻辑与代码实现上的区别，改进代码结构，排除语法逻辑上的错误。

样例：

9
37 24 42 7 32 40 45 2 120
37
42
7
45
36
-1

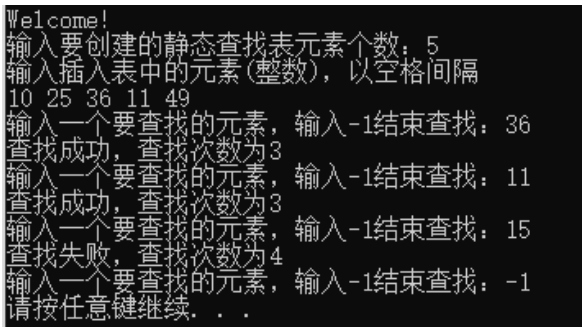
调试计划：设置好断点，注意观察每一步时各个变量的变化情况，找出错误的地方，然后改正；单步调试，更能准确定位出现错误的代码区域。

2.调试过程和结果，及分析

调试过程中由于指针指向了未知的区域，导致代码多次崩溃，发现是 insertHelp()方法出现了问题，排除错误后调试成功，输出了正确的结果。

五、测试结果

1. 样例一



一般情况，输出结果正确

2. **样例二**

```
Welcome!
输入要创建的静态查找表元素个数：7
输入插入表中的元素(整数)，以空格间隔
50 25 75 2 33 62 130
输入一个要查找的元素，输入-1结束查找：50
查找成功，查找次数为1
输入一个要查找的元素，输入-1结束查找：62
查找成功，查找次数为3
输入一个要查找的元素，输入-1结束查找：70
查找失败，查找次数为4
输入一个要查找的元素，输入-1结束查找：-1
请按任意键继续. . .
```

将 BST 设计成完全二叉树，查询效率最高，输出结果正确

3. **样例三**

```
Welcome!
输入要创建的静态查找表元素个数：5
输入插入表中的元素(整数)，以空格间隔
11 22 33 44 55
输入一个要查找的元素，输入-1结束查找：33
查找成功，查找次数为3
输入一个要查找的元素，输入-1结束查找：55
查找成功，查找次数为5
输入一个要查找的元素，输入-1结束查找：66
查找失败，查找次数为6
输入一个要查找的元素，输入-1结束查找：-1
请按任意键继续. . .
```

将 BST 设计成链式结构，查询效率最低，输出结果正确

4. **样例四**

```
Welcome!
输入要创建的静态查找表元素个数：1
输入插入表中的元素(整数)，以空格间隔
60
输入一个要查找的元素，输入-1结束查找：60
查找成功，查找次数为1
输入一个要查找的元素，输入-1结束查找：61
查找失败，查找次数为2
输入一个要查找的元素，输入-1结束查找：-1
请按任意键继续. . .
```

BST 中只有一个结点的特殊情况，输出结果正确

5. **样例五**

```
Welcome!
输入要创建的静态查找表元素个数：7
输入插入表中的元素(整数)，以空格间隔
45 50 20 45 50 20 45
输入一个要查找的元素，输入-1结束查找：45
查找成功，查找次数为1
输入一个要查找的元素，输入-1结束查找：50
查找成功，查找次数为2
输入一个要查找的元素，输入-1结束查找：20
查找成功，查找次数为2
输入一个要查找的元素，输入-1结束查找：-1
请按任意键继续. . .
```

BST 中出现多个重复元素，则只会查找到离根结点最近的元素，输出结果正确

六、实验日志

11/20

搭建好了静态查找表的结构框架，将实验内容分为三部分实现：

1.结点类（Node）：存储表中的值，有左结点与右节点。

2.二叉查找树类（BST)：一棵含有指向根节点指针的二叉树，树含有多个结点。

3.主程序（main)：向用户展示基于 BST 的静态查找表的功能。

11/23

完成了静态查找表的构建与查找功能，对递归建树与递归查找的操作方式有了更加深入地理解与体会，与常规的数组查找做对比，发现 BST 的查询次数平均值比数组要低很多，但 BST 还是存在缺陷：对于同一组数据，不同的输入顺序会使得树的形状不尽相同，对于同一组查找操作消耗的时间也不同，查找效率不稳定。

11/26

完成了整个代码工作，添加了格式化输出语句，不显得刻意呆板。

只要是从计算机、计算机网络中查找特定的信息，就需要在计算机中存储包含该特定信息的表。查找是许多程序中最消耗时间的一部分。因而，一个好的查找方法会大大提高运行速度。