数据结构

课程实验报告

题　　目：二叉搜索树

学生姓名：刘大卫

学生学号：201826010215

专业班级：软件 1802

完成时间：2019.10.29

一、 需求分析

1. 问题描述

查找表分为静态查找表和动态查找表两大类。静态查找表（Static Search Table)：只作查找操作的查找表；动态查找表（Dynamic Search Table)：在查找过程同时插入查找表中不存在的数据元素，或者从查找表中删除已经存在的某个数据元素。

要求基于课程“BST”，实现二叉查找树（BST)，实现一个静态查找表；在查找表中查找时，输出关键字的比较次数。

2. 问题分析

本题要求实现一个静态查找表，并且在其中查找时，输出关键字的比较次数。那么我们要做的首先是实现一个 BST 树的 ADT；然后编写主程序，在其中完成建树的操作，并且在其中进行查找的操作，输出比较的次数，总体来说本次实验是比较简单的。

3. 输入数据

采用输入的方式完成建树的操作。

文件中共有一行数据，均为 unsigned int 型的数据，数据与数据之间用空格区分，例如：

“9 6 3 8 5 2 74 1 9 5 3 9 7 2 3 6”

完成建立 BST 树的过程。

4. 输出数据

文件输入完成后，按照层次遍历输出 BST 树中的数据，每个数据一行，前面的空格个数表示数据的层数。

5. 测试样例

	1	2	3	4	5
输入	0	1 2 3 4 5 6 7 8 9	9 8 7 6 5 4 3 2 1	7 5 9 1 6 3 85 35 77 8 41 55 87 2 57 4 47	9 6 3 8 5 2 74 1 9 5 3 9 7 2 3 6

输出	0。	6 4 8 2 5 7 9 1 3。	6 4 8 2 5 7 9 1 3。	35 6 57 4 8 47 85 2 5 7 9 41 55 77 87 1 3	6 3 9 2 5 7 9 2 3 3 5 6 8 9 74 1。
设计目的	测试只有根节点的 BST 树，验证是否可以成功建立，查询。	测试只有右子树的 BST 树，验证是否可以成功建立，查询。	测试只有左子树的 BST 树，验证是否可以成功建立，查询。	测试没有重复数据的 BST 树，验证是否可以成功建立，查询。	测试有重复数据的 BST 树，验证是否可以成功建立，查询，只输出首次查询到的结果。

二、 概要设计

1. 抽象数据类型

为实现上述程序的功能，我们使用了 BST 树.BST 树的定义如下：若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值；若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值； 它的左、右子树也分别为二叉排序树。抽象数据类型设计：

数据对象：n 个 BST 树节点（BSTNode）；

数据关系：每个节点分为数据域与指针域两部分，数据域中定义本节点中的数据；指针域分为两部分，分别为左子树指针（lc）和右子树指针（rc）。其中左子树中的数据均小于本节点，右子树的数据均大于本节点。构成一棵 BST 树。

基本操作：本题主要应用的是 BST 树 ADT 中定义的建树操作和层次遍历。

2. 算法的基本思想

本题的主要目的是实现一棵 BST 树，并在其中进行查询的操作。所以我们只需要定义其中的插入和查询操作即可。需要注意的是，这些操作都是递归进行的我们需要深刻的理解递归的相关内容和定义。

建树主要算法如下：

1） 如果长度为 0，那么返回。

2） 否则节点赋值，建立左子树，递归建树，建立右子树，递归建树。

进行上述算法则可以成功解决问题。

3. 程序的流程

本体是简单的输入、处理与输出模型。按照“一、需求分析”中的输入输出格式进行即可。

三、 详细设计

1. 物理数据类型

具体的物理数据类型我们选择的是二叉检索树（BST），我们的实现基于课本上实现的 ADT，下面给出具体的物理结构：

```
class BSTNode {
private:
    int data; // 数据域
    BSTNode *lc; // 左子树指针
    BSTNode *rc; // 右子树指针
public:
    BSTNode() {} // 默认构造函数
    BSTNode(int e, BSTNode *l = NULL, BSTNode *r = NULL) {}
    // 带参数的构造函数
    int getData() {} // 获取数据
    BSTNode *left() {} // 获取左指针
    BSTNode *right() {} // 获取右指针
    void setLeft(BSTNode *b) {} // 设置左指针
    void setRight(BSTNode *b) {} // 设置右指针
    void setElement(const int &e) {} // 设置数据
};
```

在此基础上进行 BST 树的定义，在其中建立 BSTNode *root 还有查询函数 bool find(int &)和插入函数 void insert(int &)。

2. 输入输出的格式

采用文件输入的方式完成建树的操作，使得验收时可以专注于查找的测试。在控制台中先输入数据集的组数（1-5），例如“5”。

文件中共有一行数据，均为 unsigned int 型的数据，数据与数据之间用空格区分，例如：

“9 6 3 8 5 2 74 1 9 5 3 9 7 2 3 6”

完成建立 BST 树的过程。

3. 算法的时空分析

1) 进行插入操作时每次我们只需要比较的时下一层子树的其中一边，所以时间复杂度会降低，不再是线性表的$O(n)$，而是$O(\log n)$；
2) 同样的道理进行插入操作时，时间复杂度也为$O(\log n)$；
3) 但是，需要注意的是，我们这里的二叉检索树并不是平衡的，我们的时间复杂度是有可能达到$O(n)$的。

四、 调试分析

1. 调试方案设计

调试目的：

测试程序是否可运行，发现代码的语法错误、连接错误、逻辑错误和运算错误，是否有不严谨之处。

测试样例：

文件中的数据：

9 6 3 8 5 2 74 1 9 5 3 9 7 2 3 6

输入的要查询的数据：

9 74 38 -1

调试计划：

设定好断点，单步执行，查看 BST 中元素的变化情况，同时观察是否是符合有序的要求，在递归部分辅助以栈的演算，确定当前的执行环节，想办法对算法进行优化。对输出模块的各种情况进行完善，确定边界情况。寻找问题，检查遗漏。

设置：

在新建 BST 树后设置断点，进行单步执行，观察 BST 树中每一个元素的变化情况，查看是否正确（主要采用输出全部序列的方法进行调试工作）；针对递归部分的处理，通过栈辅助以纸面上的演算。及时进行修正，对代码的不完善之处进行修改。

2. 调试过程和结果以及分析

调试时发现，在插入时出现了整棵树无法建立的问题，发现在 insertHelp 函数的末尾没有返回值造成错误，及时加入了不满足条件时返回 root 的代码，即：return root，代码测试获得通过，各种功能的测试没有问题。

五、 实验日志

2019 年 10 月 29 日：

开始代码的编写工作，重新复习了二叉检索树的插入、删除、查询功能的原理以及代码的相关实现，完成 BSTNode 的编写与 BST 的 ADT 的编写。

2019 年 10 月 29 日：

基本完成 BST 中函数的编写工作，测试时发现了没有在 insertHelp 中及时返回 root 的问题进行了修复；

2019 年 10 月 29 日：

完成主程序的编写工作，设计完成测试样例，并且进行验证功能的完整性。

通过本题，我对于合理应用数据结构，提升算法有了更加深入的理解，也勾起了我对于数据结构学习的热情，相信我在这个学期中，一定可以学好数据结构，并且把他应用到以后的实践中去。更好的提升自己。