湖南大學

数据结构

课程实验报告

题　　目：BST

学生姓名　吴多智

学生学号　201626010520

专业班级　软件1605

完成日期　2017.11.8

一、 需求分析

（这个部分一般要撰写四个方面的内容）

1. 问题描述

基于教材内容，实现二叉树。同时，需要基于左子结点/右兄弟结点表示法或二叉链来实现二叉树 ADT,需要实现二叉树的各个基本操作。最后，编写一个 demo 程序，测试二叉树的各个基本操作的功能是否正常，实现二叉树的构建以及一种遍历。

2.输入输出数据

2.1 测试构建

输入整数，以-1 结束 Example： 1 3 2 4 6 8 8 9 8 -1

输出：二叉树前序遍历：1 3 2 4 6 8 9

二叉树中序遍历：1 2 3 4 6 8 9

2.2 查找

查找操作:(请输入要查找的值，输入-1 结束操作)

输入 3

输出 查找成功 2

3.测试样例设计

3.1 输入 8 //节点个数

34 76 45 18 26 54 92 65

45 //要查找的值

查找成功 3 //结果显示

34 //要查找的值

查找成功 1 //结果显示

100 //要查找的值

查找不成功 3 //结果显示

3.2 输入 10 //节点个数

34 76 45 18 26 54 92 65 99 100

92 //要查找的值

查找成功 3 //结果显示

101 //要查找的值

查找不成功 5 //结果显示

101 //要查找的值

查找成功 6 //结果显示

3.3 输入 8 //节点个数

34 76 45 18 26 54 92 65

1 //要查找的值

查找不成功 2 //结果显示

2 //要查找的值

查找不成功 3 //结果显示

3 //要查找的值

查找不成功 4 //结果显示

3.4 输入 0 //节点个数

1　　//要查找的值
查找不成功 0　　//结果显示
1　　//要查找的值
查找成功 1　　//结果显示
2　　//要查找的值
查找不成功 1　　//结果显示

3.5 输入　15　//节点个数
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
15　　//要查找的值
查找成功 15　　//结果显示
16　　//要查找的值
查找不成功 15　　//结果显示
3　　//要查找的值
查找成功 3　　//结果显示

二、概要设计

（这个部分一般要撰写三个方面的内容）

1.抽象数据类型

实验中，用链表实现二叉树，每个节点有数据域和左右子节点，同时，由于要实现的是二叉查找树，在插入时要注意维持二叉查找树的性质。

节点类：

```
//二叉查找树的节点结构
template <typename T>
struct BSNode
{
    BSNode(T t)
    : value(t), lchild(NULL), rchild(NULL){}

    BSNode() {};

    T value;
    BSNode<T>* lchild;
    BSNode<T>* rchild;
};
```

二叉查找树 ADT:

```
template <typename T>
class BSTree{
    public:
        BSTree();
        ~BSTree(){};

        void preOrder();
        void inOrder();
        void postOrder();
```

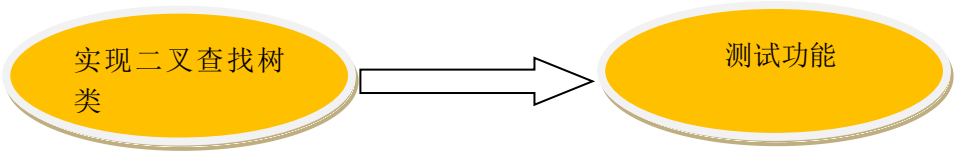
```
        void layerOrder();

        void insert(T key);       //插入指定值节点
        void remove(T key);       //删除指定值节点
        bool find(T key);
        //BSNode<T>* find(T key);
        void destory();           //销毁二叉树
        void print();             //打印二叉树
    private:
        BSNode<T>* root; //根节点
        void add(T t);
        void preOrder(BSNode<T>* p);
        void inOrder(BSNode<T>* p);
        void postOrder(BSNode<T>* p);
};
```

2.算法的基本思想

本实验主要实现二叉查找树，最主要的功能是插入与删除，插入操作中，类似与折半查找，比当前节点大就往右查找节点，比当前节点小，就往左查找，只到找到空节点或与要插入的值相等的节点；删除的操作，有点复杂，要根据节点的情况进行删除，详细内容看详细设计，同时还要注意删除的是不是根节点。

3.程序的流程



三、详细设计

（这个部分一般要撰写四个方面的内容）

1.物理数据类型

二叉树的每个节点是一个单独的类，且有两个指针分别指向左右孩子节点。底层是链表实现，删除、插入快。

2.输入和输出的格式

```
cout<<"创建二叉树：请输入数字，-1 代表添加结束。(相同的数字，只有一个会被添加）\n";
    BSTree<int> tree = BSTree<int>();
    int n = 0;
    while(n!=-1){
        cin>>n;
        if(n==-1)
            break;
        tree.insert(n);
    }

    cout<<"\n 二叉树前序遍历:";
```

```
tree.preOrder();
cout<<"\n 二叉树中序遍历:";
tree.inOrder();

cout<<"\n 插入操作(请输入要插入的值)：\n";
cin>>n;
if(tree.find(n))
    cout<<"The value is exixt.\n";
else
    tree.insert(n);

cout<<"二叉树前序遍历:";
tree.preOrder();
cout<<"\n 二叉树中序遍历:";
tree.inOrder();

cout<<"\n 删除操作(请输入要删除的值）:\n";
cin>>n;
if(!tree.find(n))
    cout<<"The value don't exixt.\n";
else
    tree.remove(n);

cout<<"二叉树前序遍历:";
tree.preOrder();
cout<<"\n 二叉树中序遍历:";
tree.inOrder();
```

3.算法的具体步骤

3.1 二叉树的添加

```
template <typename   T>
void BSTree<T>::add(T t){
    BSNode<T>* n = new BSNode<T>(t);
    BSNode<T>* cur = root;
    BSNode<T>* head;
    while(cur!=NULL){
        head = cur;             //循环查找到空节点
        if(cur->value < t){
            cur = cur->rchild;
        }else{
            cur = cur->lchild;
        }
    }

    if(head->value < t){       //判断与当前节点的大小，比当前节点大，添加到右节
```

```
            head->rchild = n;       //点，小就添加到左节点
        }else{
            head->lchild = n;
        }
}
```

3.2 二叉数的删除

```
template <typename T>
void BSTree<T>::remove(T t){
    if(find(t)){
        BSNode<T>* cur = root;
        BSNode<T>* parent = NULL;
        BSNode<T>* del = NULL;

        while(cur && cur->value!=t){     //循环找到要删除的节点
            parent = cur;
            if(cur->value >t ){
                cur = cur->lchild;
            }else{
                cur = cur->rchild;
            }
        }

        del = cur;

        if(cur == root){              //如果为根节点
            if(cur->lchild == NULL){
                root = cur->rchild;
            }else if(cur->rchild == NULL){
                root = cur->lchild;
            }else{
                BSNode<T>* precursor = cur->lchild;
                BSNode<T>* par = cur;
                while(precursor->rchild){
                    par = precursor;
                    precursor = precursor->rchild;
                }
                cur->value = precursor->value;
                if(par != cur){
                    par->rchild = precursor->lchild;
                }else{
                    par->lchild = precursor->lchild;
                }
                del = precursor;
            }
```

```
            }else{
                if(cur->lchild == NULL){
                    if(cur == parent->lchild){
                        parent->lchild = cur->rchild;
                    }else{
                        parent->rchild = cur->rchild;
                    }
                }else if(cur->rchild == NULL){
                    if(cur == parent->lchild){
                        parent->lchild = cur->lchild;
                    }else{
                        parent->rchild = cur->lchild;
                    }
                }else{
                    BSNode<T>* precursor = cur->lchild;
                    BSNode<T>* par = cur;
                    while(precursor->rchild){
                        par = precursor;
                        precursor = precursor->rchild;
                    }
                    cur->value = precursor->value;
                    if(par != cur){
                        par->rchild = precursor->lchild;
                    }else{
                        par->lchild = precursor->lchild;
                    }
                    del = precursor;
                }
            }

            delete del;
        }else{
            cout<<"The element is not exixt.\n";
        }
    }
```

4.算法的时空分析

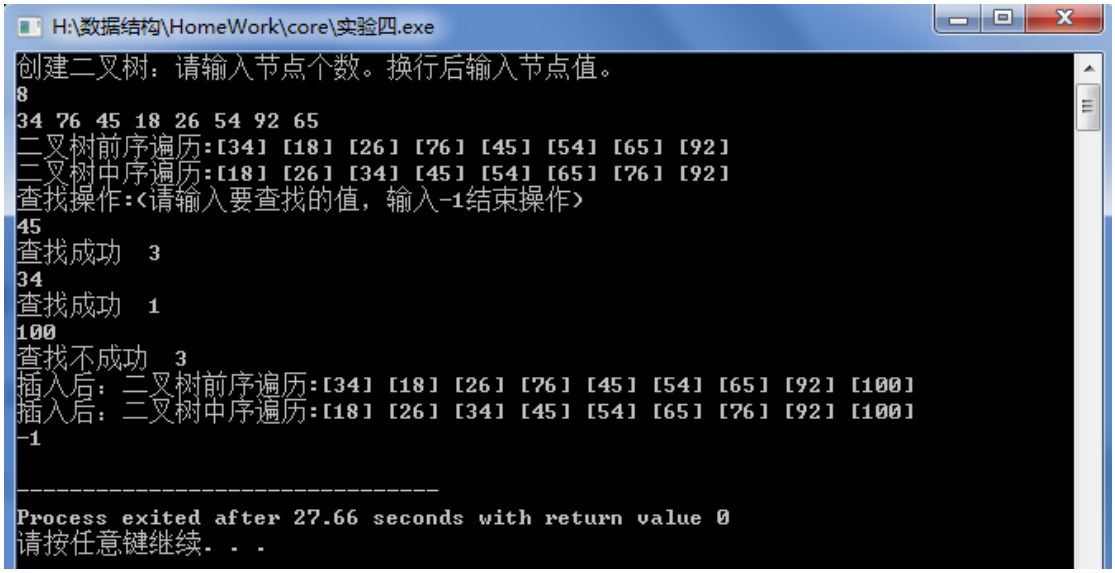
二叉查找数的添加、删除、查找的时间复杂度均为 $O(\log_2 n)$。

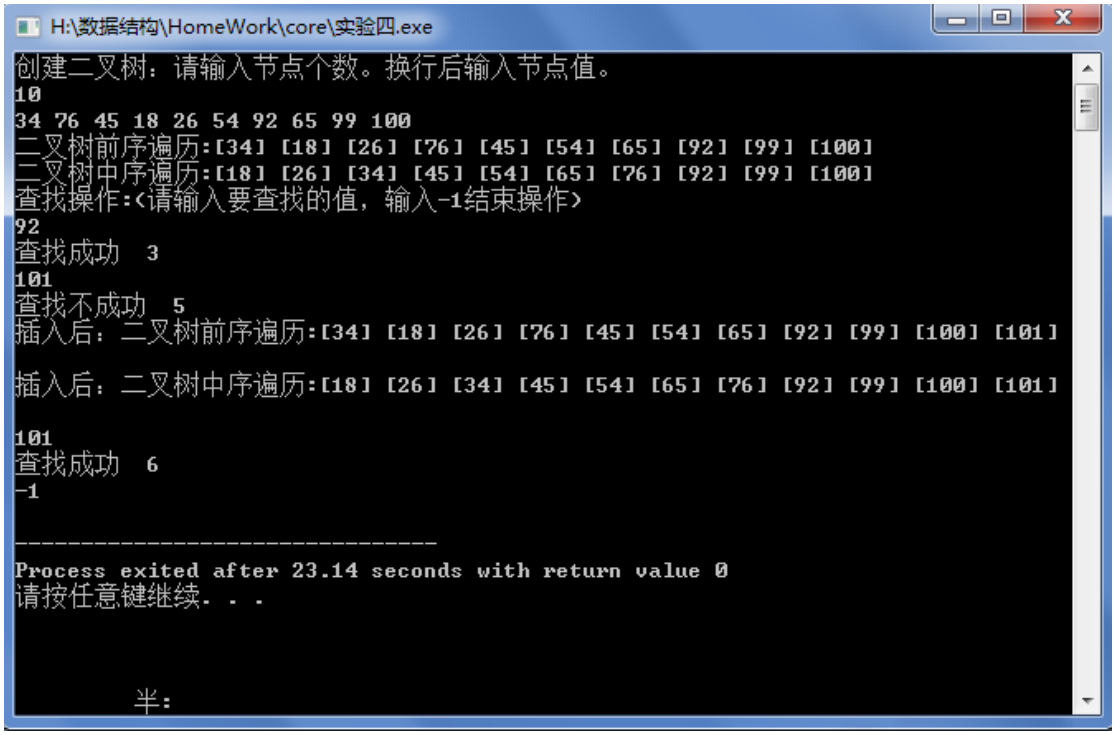
四、调试分析

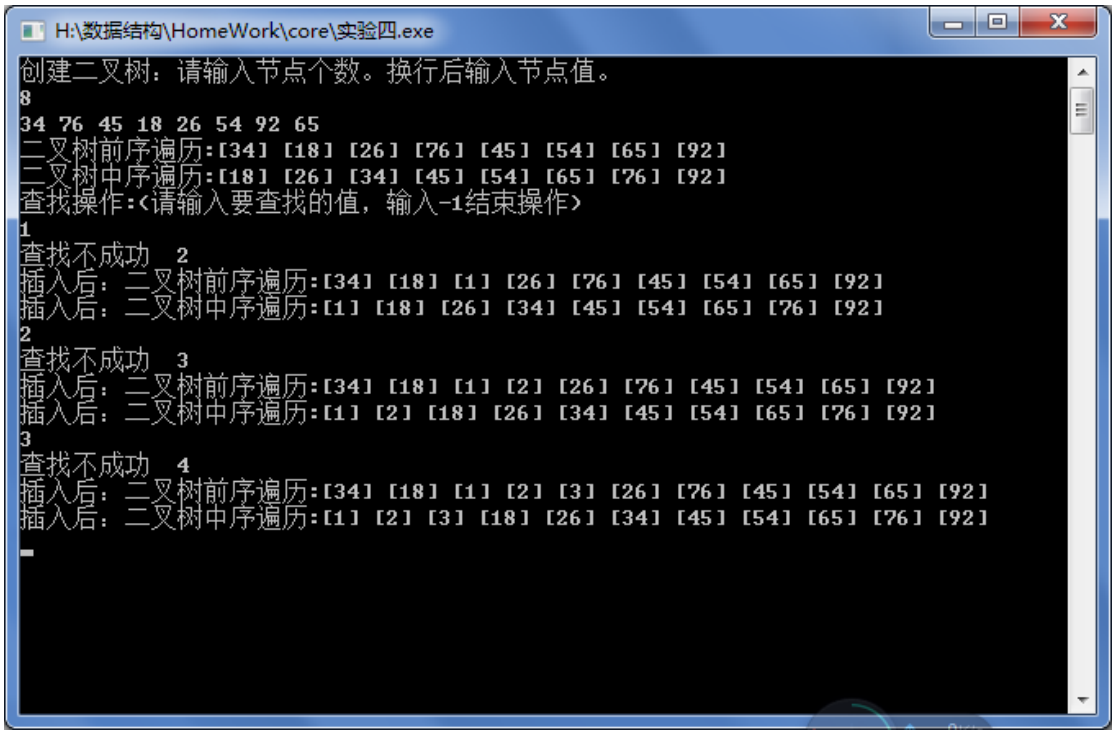
1.调试方案设计

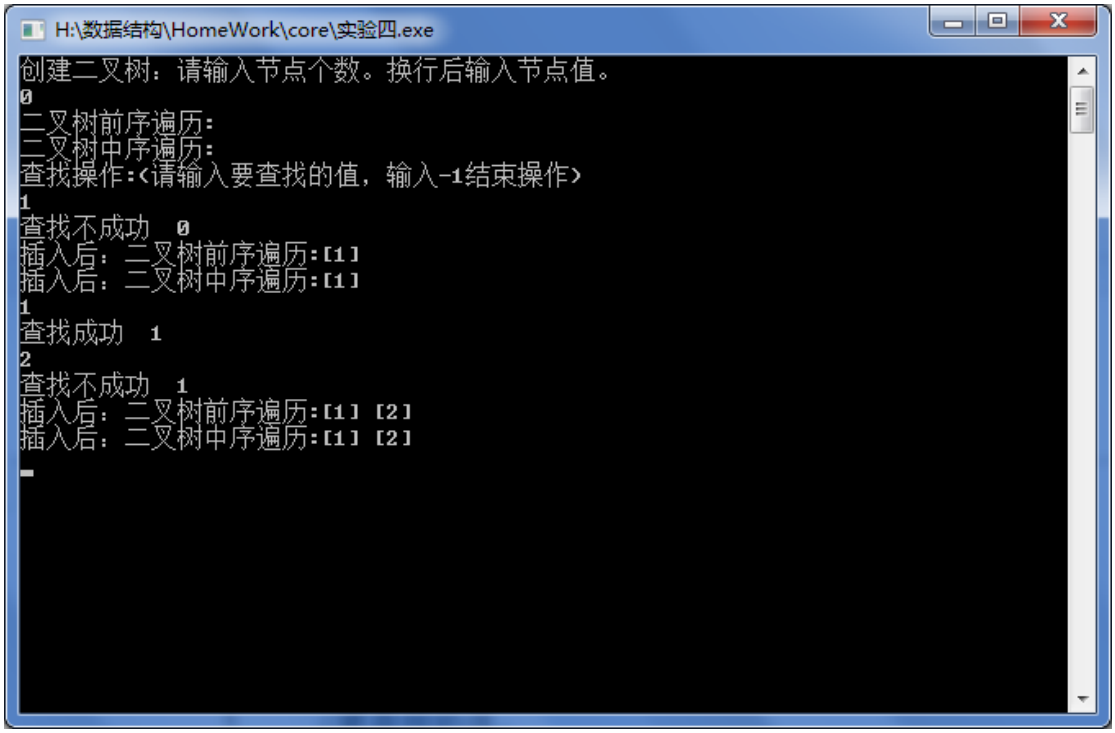
二叉树的插入和删除都设计到指针的操作，特别是删除操作，是要根据当前节点的情况进行不同的删除操作，需要通过调试，查看变量值，观察程序是否按照预期进行。

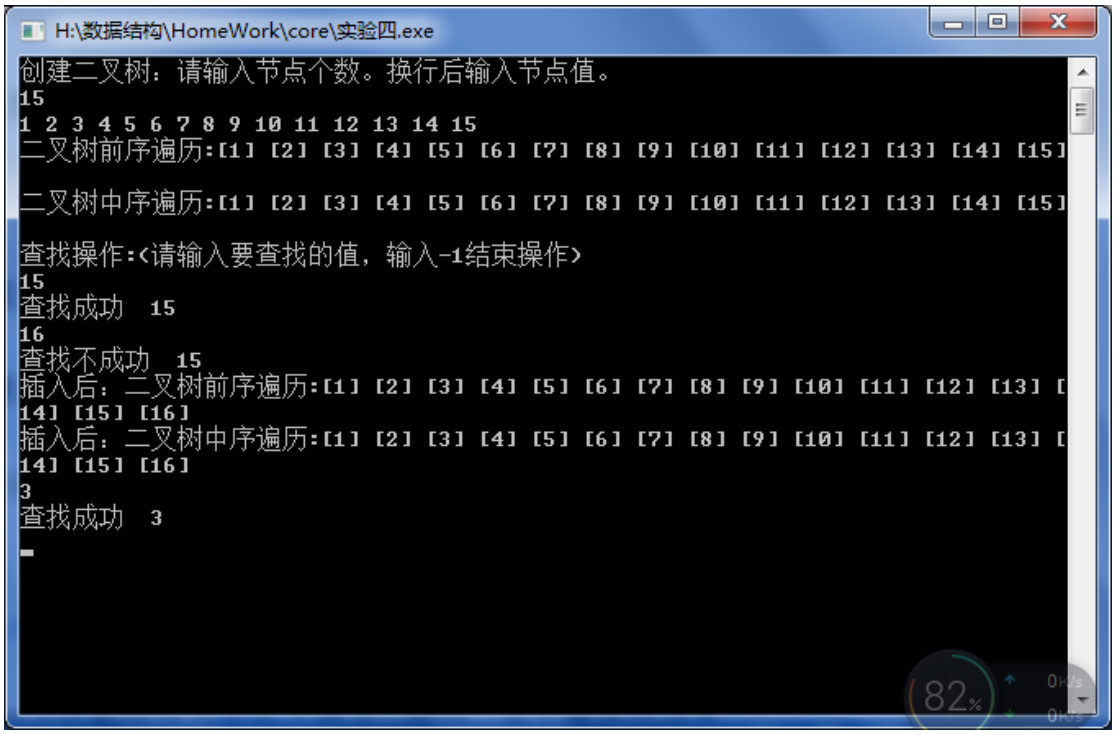
五、实验结果


H:\数据结构\HomeWork\core\实验四.exe
创建二叉树：请输入节点个数。换行后输入节点值。
8
34 76 45 18 26 54 92 65
二叉树前序遍历:[34] [18] [26] [76] [45] [54] [65] [92]
二叉树中序遍历:[18] [26] [34] [45] [54] [65] [76] [92]
查找操作:<请输入要查找的值，输入-1结束操作>
45
查找成功 3
34
查找成功 1
100
查找不成功 3
插入后：二叉树前序遍历:[34] [18] [26] [76] [45] [54] [65] [92] [100]
插入后：二叉树中序遍历:[18] [26] [34] [45] [54] [65] [76] [92] [100]
-1
Process exited after 27.66 seconds with return value 0
请按任意键继续. . .


H:\数据结构\HomeWork\core\实验四.exe
创建二叉树：请输入节点个数。换行后输入节点值。
10
34 76 45 18 26 54 92 65 99 100
二叉树前序遍历:[34] [18] [26] [76] [45] [54] [65] [92] [99] [100]
二叉树中序遍历:[18] [26] [34] [45] [54] [65] [76] [92] [99] [100]
查找操作:<请输入要查找的值，输入-1结束操作>
92
查找成功 3
101
查找不成功 5
插入后：二叉树前序遍历:[34] [18] [26] [76] [45] [54] [65] [92] [99] [100] [101]
插入后：二叉树中序遍历:[18] [26] [34] [45] [54] [65] [76] [92] [99] [100] [101]
101
查找成功 6
-1
Process exited after 23.14 seconds with return value 0
请按任意键继续. . .
半:


H:\数据结构\HomeWork\core\实验四.exe
创建二叉树：请输入节点个数。换行后输入节点值。
8
34 76 45 18 26 54 92 65
二叉树前序遍历:[34] [18] [26] [76] [45] [54] [65] [92]
二叉树中序遍历:[18] [26] [34] [45] [54] [65] [76] [92]
查找操作:<请输入要查找的值，输入-1结束操作>
1
查找不成功 2
插入后：二叉树前序遍历:[34] [18] [1] [26] [76] [45] [54] [65] [92]
插入后：二叉树中序遍历:[1] [18] [26] [34] [45] [54] [65] [76] [92]
2
查找不成功 3
插入后：二叉树前序遍历:[34] [18] [1] [2] [26] [76] [45] [54] [65] [92]
插入后：二叉树中序遍历:[1] [2] [18] [26] [34] [45] [54] [65] [76] [92]
3
查找不成功 4
插入后：二叉树前序遍历:[34] [18] [1] [2] [3] [26] [76] [45] [54] [65] [92]
插入后：二叉树中序遍历:[1] [2] [3] [18] [26] [34] [45] [54] [65] [76] [92]


H:\数据结构\HomeWork\core\实验四.exe
创建二叉树：请输入节点个数。换行后输入节点值。
0
二叉树前序遍历:
二叉树中序遍历:
查找操作:<请输入要查找的值，输入-1结束操作>
1
查找不成功 0
插入后：二叉树前序遍历:[1]
插入后：二叉树中序遍历:[1]
1
查找成功 1
2
查找不成功 1
插入后：二叉树前序遍历:[1] [2]
插入后：二叉树中序遍历:[1] [2]



六、实验心得

通过对二叉树的实验，加上自己的调试分析，看程序一步步执行，深刻发现二叉树插入的便捷性，且保持有序，查找的效率也是极高。但如果给出的数字的顺序是基本有序的的话，二叉树就会往一边倾斜，甚至演变成二叉链表。在这种情况下，可能需要平衡树或者红黑树来解决。