二叉排序树

1.项目简介

依次输入关键字并建立二叉排序树，实现二叉排序数的插入和查找功能。

2.项目功能简介

二叉排序树就是指将原来已有的数据根据大小构成一棵二叉树，二叉树中的所有结点数据满足一定的大小关系，所有的左子树中的结点均比根结点小，所有的右子树的结点均比根结点大。
二叉排序树查找是指按照二叉排序树中结点的关系进行查找，查找关键自首先同根结点进行比较，如果相等则查找成功；如果比根节点小，则在左子树中查找；如果比根结点大，则在右子树中进行查找。这种查找方法可以快速缩小查找范围，大大减少查找关键的比较次数，从而提高查找的效率。

3.核心代码及其功能

- **声明的变量**
  这里我创建了三个类，分别为Node为二叉树的节点结构，bTree为二叉搜索树结构，System为抽象整个程序的操作。下面为各个类的具体声明：

```
/*-----Node------*/
class Node{
public:
    Node():left(NULL),right(NULL){}          //初始化让左右两个节点为NULL
    ~Node(){}
    Node* left;                              //指向左子节点
    Node* right;                             //指向右子节点
    int value;                               //保存该节点的值
};
/*-------bTree-------*/
class bTree{                                 //抽象整个二叉排序树
public:
    bTree(){}
    ~bTree(){}
    void Create();                           //构建二叉排序树
    Node* getHead(){return Head;}            //获得二叉排序树的头节点
    void setHead(Node* node){Head=node;}     //设置二叉排序树的头节点
    void Insert(int number);                 //向树中插入新的元素
    bool Find(int number);                   //在二叉排序树中寻找是否有number元素
    void MidTraverse(Node* ptr);             //中序遍历二叉排序树
    void Display();                          //打印二叉排序树
private:
    Node* Head;                              //保存二叉排序树的头节点
};
/*------System-------*/
class System{                                //将整个程序操作抽象为系统
public:
    System();
    ~System(){}
    void Create();                           //构建二叉排序树
    void Insert();                           //插入操作
    void Find();                             //查找操作
    bTree bSortTree;                         //声明二叉排序树结构
};
```

- **创建二叉搜索树**

这里就是读入输入然后直接调用Insert函数将读入的数插入到二叉搜索树里就可以了，由于插入的地方是树的最下层节点，所以不会打乱树的原来结构。所以根据小于当前节点到左边，大于当前节点到右边的想法就可以得到Insert函数:

```
void bTree::Insert(int number){
    auto node=new Node;
    node->value=number;
    auto ptr=getHead();
    if(ptr){                                    //如果二叉排序树当前不为空则向下查找可插入的位置
        while(1){
            if(number<ptr->value){              //如果插入的数比当前节点小就进入左子树
                if(ptr->left){
                    ptr=ptr->left;
                }
                else{                           //如果没有左子树就直接插入其左子树位置
                    ptr->left=node;
                    break;
                }
            }
            else if(number>ptr->value){         //如果插入的数比当前节点大就进入右子树
                if(ptr->right){
                    ptr=ptr->right;
                }
                else{                           //如果没有右子树就直接插入其右子树位置
                    ptr->right=node;
                    break;
                }
            }
            else{                               //如果插入的数已经存在树中则返回
                cout<<"The input key<"<<number<<"> is have in!"<<endl;
                return;
            }
        }
    }
    else{
        setHead(node);
    }
}
```

然后通过调用Insert函数就可以得到Create函数的操作：

```
void bTree::Create() {
    int num=1;
    cout<<"Please input key to create Bsort_Tree:"<<endl;
    do{
        cin>>num;
        if(num==0){                             //遇到输入为0停止输入
            break;
        }
        else{
            Insert(num);                        //将读入的数插入树中
        }
    }while(1);
    Display();                                  //打印二叉排序树
}
```

- **查找操作**
  这里的查找也是遵循之前的想法，如果手上的数比当前节点的数小就到左子树里查找，如果手上的数比当前节点的数大就到右子树里查找。如果找到就返回true，没有找到就一直到ptr为NULL，返回false：

```
bool bTree::Find(int number){
    auto ptr=getHead();
    while(ptr){                                 //寻找到ptr为NULL时返回未找到false
        if(ptr->value>number){
            ptr=ptr->left;
```

```
        }
        else if(ptr->value<number){
            ptr=ptr->right;
        }
        else{
            return true;                    //找到则直接返回true
        }
    }
    return false;
}
```

- **打印操作**
  由于对于二叉排序树，其中序遍历出来的结果就是排序好的数列，所以我们选择使用中序遍历的方式对二叉树进行打印。这里中序遍历采用递归的方式。

```
void bTree::Display() {
    cout<<"Bsort_Tree is: "<<endl;
    auto head=getHead();
    MidTraverse(head);                      //调用中序遍历打印二叉排序树
    cout<<endl;
}
void bTree::MidTraverse(Node *ptr) {
    if(ptr){
        MidTraverse(ptr->left);
        cout<<ptr->value<<"->";
        MidTraverse(ptr->right);
    }
}
```

4.项目实例

> 以下代码均为顺序执行结果

- **初始界面**

```
/Users/kirito/CLionProjects/untitled/cmake-build-debug/untitled
**                  二叉排序树                    **
================================================
**               1.---建立二叉排序树              **
**               2.---插入元素                    **
**               3.---查询元素                    **
**               4.---退出程序                    **
================================================
```

- **创建二叉排序树**

```
Please select: 1
Please input key to create Bsort_Tree:
12 34 67 48 19 44 21 30 19 7 4 24 9 88 100 100 0
The input key<19> is have in!
The input key<100> is have in!
Bsort_Tree is:
4->7->9->12->19->21->24->30->34->44->48->67->88->100->
```

- **插入数字**

```
Please select:  2
Please input key which inserted: 90
Bsort_Tree is:
4->7->9->12->19->21->24->30->34->44->48->67->88->90->100->
```

- **查找数字**

```
Please select:  3
Please input key which searched: 90
search success!
Please select:  3
Please input key which searched: 110
110 not existed!
```

- **退出程序**

```
Please select:  4
press any key to continue



Process finished with exit code 0
```