

目录

目录

1.项目背景

2.需求分析

3.思路分析

4.核心代码说明

节点类

二叉排序树类

函数功能说明

返回根节点

中序遍历二叉树

插入元素

搜索元素

函数接口说明

5.使用方法和函数功能演示

建立二叉排序树

插入元素

搜索元素

退出程序

1.项目背景

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。

2.需求分析

本项目中，要求实现以下功能：

- 建立二叉排序树
- 二叉排序树插入元素
- 二叉排序树元素查询
- 有序输出二叉排序树

3.思路分析

二叉排序树的插入和搜索均可以通过递归方法实现，而有序输出则中序遍历一遍二叉树即可

插入：

从根节点开始，执行以下操作：

- 新元素与当前节点进行比较
 - 等于：显示该元素已存在，结束操作
 - 大于
 - 右子树存在：对右子树进行插入操作
 - 右子树不存在：作为右子节点插入
 - 小于
 - 左子树存在：对左子树进行插入操作
 - 左子树不存在：作为左子节点插入
- 中序遍历二叉树，输出结果

查询算法的思路与插入类似，不过当比较到叶子节点仍未找到相等元素则说明不存在，不需要在新建子节点

4.核心代码说明

节点类

```
1 class Node {
2 public:
3     int value;
4     Node* left_child;
5     Node* right_child;
6     Node(int m_value):value(m_value),left_child(NULL),right_child(NULL){}
7 };
```

二叉树的节点类，包含该节点值，左右节点指针，带参数的构造函数

二叉排序树类

```
1 class BST {
2 private:
3     Node* root;
4 public:
5     //Default constructor
6     BST():root(NULL){}
7     //Return the address of root
8     Node* getRoot() { return root; }
9     //Inorder traversal
10    void inorder(Node* node);
11    //Add new element
12    void addChild(Node* node,Node* new_node);
13    //Search node
14    void searchNode(Node* node,int value,bool& flag);
15 };
```

私有成员变量：根节点root

公有成员函数功能见代码注释

函数功能说明

返回根节点

因为根节点root为私有成员变量，该函数提供获取根节点地址的方法

```
1 Node* getRoot() { return root; }
```

中序遍历二叉树

采用递归方法实现

```
1 //Output the tree in inorder traversal
2 void BST::inorder(Node* node) {
3     if (node->left_child)
4         inorder(node->left_child);
5     cout << node->value << "->";
6     if (node->right_child)
7         inorder(node->right_child);
8 }
```

插入元素

采用递归方法实现，算法逻辑如下：

首先判断该节点是否存在，若不存在则直接结束

新元素与当前节点进行比较

- 等于：显示该元素已存在，结束操作
- 大于
 - 右子树存在：对右子树进行插入操作
 - 右子树不存在：作为右子节点插入
- 小于
 - 左子树存在：对左子树进行插入操作
 - 左子树不存在：作为左子节点插入

```
1 //Add child to the tree
2 void BST::addChild(Node* node, Node* new_node) {
3     if (!node && node!=root)
4         return;
5     if (node == root && node == NULL) {
6         root = new_node;
7         return;
8     }
9     if (node->value == new_node->value) {
10        cout << "The input key <" << node->value << "> have been in!" << endl;
11        return;
12    }
13    else if (new_node->value > node->value) {
14        if (!node->right_child) {
```

```

15     node->right_child = new_node;
16     return;
17 }
18 else
19     addChild(node->right_child, new_node);
20 }
21 else {
22     if (!node->left_child) {
23         node->left_child = new_node;
24         return;
25     }
26     else
27         addChild(node->left_child, new_node);
28 }
29 }

```

搜索元素

采用递归方式实现

首先判断节点是否存在，不存在则直接返回

若存在则进行值比较：

- 相等：输出搜索成功信息
- 小于：进入左子树进行搜索
- 大于：进入右子树进行搜索

```

1  //Search node in the tree
2  void BST::searchNode(Node* node, int value, bool& flag) {
3      if (node == NULL)
4          return;
5      else {
6          if (node->value == value) {
7              cout << "search success!" << endl;
8              flag = true;
9          }
10         else if (node->value > value)
11             searchNode(node->left_child, value, flag);
12         else
13             searchNode(node->right_child, value, flag);
14     }
15 }

```

函数接口说明

返回值类型	成员函数名	参数	属性	功能
\	BST	()	public	默认构造函数
Node*	getRoot	()	public	返回根节点地址
void	inorder	(Node* node)	public	中序遍历二叉树
void	addChild	(Node* node,Node* new_node)	public	插入元素
void	searchNode	(Node* node,int value,bool& flag)	public	搜索元素

5.使用方法和函数功能演示

双击 `b_sort_tree.exe` 启动程序，按照系统提示进行操作

建立二叉排序树

输入1选择该功能，随后以空格为分隔符输入元素，以0作为结尾标志（不会被计入元素），如果有重复元素，则系统只会保留先输入的一个，并提示该元素已存在。建立完成后，系统会按大小顺序输出二叉树。

```
C:\Users\aa\OneDrive\文档\数据结构实验\Data-Structure-Practice\exercise_9\b_sort_tree.exe
**      Binary-Sort-Tree      **
=====
**      1 --- Build the BST    **
**      2 --- Insert element   **
**      3 --- Search element   **
**      4 --- Exit             **
=====

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

Please select:
```

插入元素

输入2选择该功能，随后按提示输入要插入的元素，插入完成后有序输出二叉树

```
C:\Users\aa\OneDrive\文档\数据结构实验\Data-Structure-Practice\exercise_9\b_sort_tree.exe

**      Binary-Sort-Tree      **
=====
**      1 --- Build the BST    **
**      2 --- Insert element   **
**      3 --- Search element   **
**      4 --- Exit             **
=====

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

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->

Please select:
```

搜索元素

输入3选择该功能，按提示输入要查找的元素，如果搜索到，则会提示搜索成功，未搜索到则会提示搜索失败

```
C:\Users\aa\OneDrive\文档\数据结构实验\Data-Structure-Practice\exercise_9\b_sort_tree.exe
**      4 --- Exit      **
=====

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

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->

Please select:3
Please input key which searched:90
search success!

Please select:3
Please input key which searched:110
110 not exist!

Please select:
```

退出程序

输入4退出程序，再次运行则重启exe文件