项目说明文档

数据结构课程设计

——二叉排序树

作 者 姓 名：陆诚彬

学　　号：2254321

指 导 教 师：张颖

学院、 专业：软件学院 软件工程

同济大学

Tongji University

目录

1 项目背景

二叉排序树（Binary Search Tree，简称 BST）是计算机科学中的一种经典数据结构，用于存储数据的同时保持数据的有序性。其主要特点是每个节点都有一个键值，且每个节点的键值都大于其左子树上任意节点的键值，而小于右子树上任意节点的键值。这种特性使得二叉排序树在数据存储和检索方面表现出了卓越的性能。

随着数据量的增加和应用场景的复杂化，对数据处理效率的要求也日益增高。在这种背景下，二叉排序树作为一种高效的数据组织和处理方式，被广泛应用于数据库管理、内存管理、网络数据传输等众多领域。特别是在处理大量动态更新数据时，二叉排序树能够提供快速的插入、删除和查找操作，显著提高数据处理效率。

2 项目需求分析

2.1 功能需求

1) **建立二叉排序树：**能够接受用户输入的一系列关键字，并依据这些关键字构建二叉排序树。
2) **数据插入：**用户能够插入新的数据项到树中，插入操作需保持树的排序特性。
3) **数据查找：**提供一个有效的查找功能，能夠快速定位并返回特定的数据项。

2.2 非功能需求

1) **查找效率：**查找操作应尽可能高效，能够在对数时间复杂度内完成。
2) **内存管理：**树的构建和操作应有效管理内存，防止内存泄漏。
3) **输入界面：**提供简洁明了的界面，用户可以方便地输入关键字。
4) **输出展示：**清晰展示树的结构，以及查找和插入操作的结果。

2.3 项目输入输出需求

2.3.1 输入格式

输入相应的操作数，以及数据数字。

2.3.2 输出格式

输出相应的提示信息及二叉树架构。

2.3.3 项目示例

3 项目设计

3.1 数据结构设计

本程序的核心是构建和操作一个二叉搜索树（BST）。以下是数据结构的设计要点：

3.1.1 节点结构（Node）

目的：用于表示树中的单个节点。

结构：

1) 关键字（key）：整型，用于存储节点的值。
2) 左子节点指针（left）：指向左子树的指针。
3) 右子节点指针（right）：指向右子树的指针。
4) 构造函数：接受一个整数值并初始化节点。

3.1.2 二叉搜索树（BST）

目的：用于表示树中的单个节点。

结构：

1) 根节点（root）：指向整个树的根节点的指针，初始化为 nullptr。
2) 插入操作（insert）：递归地在树中插入新元素。
3) 搜索操作（search）：递归地在树中查找元素。
4) 中序遍历（inorderTraversal）：递归地进行中序遍历以按顺序显示树中的元素。

4 项目实现

4.1 创建节点实现

4.1.1 创建节点功能简介

功能描述：创建一个新的树节点。

4.1.2 创建节点核心代码

```
1.    Node(int value) {
2.            key = value;
3.            left = right = nullptr;
4.        }
```

4.2 插入元素实现

4.2.1 插入元素功能简介

功能描述：在二叉搜索树中插入一个新的元素。

4.2.2 建立家族核心代码

```
1.    void BinarySearchTree::insert(int key) {
2.        root = insertRec(root, key);
3.    }
4.
5.    Node* BinarySearchTree::insertRec(Node* root, int key) {
6.        if (root == nullptr) {
```

```
            return new Node(key);
        }
        if (key < root->key) {
            root->left = insertRec(root->left, key);
        } else if (key > root->key) {
            root->right = insertRec(root->right, key);
        }
        return root;
    }
```

4.3 搜索元素实现

4.3.1 搜索元素功能简介

功能描述：在二叉搜索树中搜索给定的元素。

4.3.2 搜索元素核心代码

```
    bool BinarySearchTree::search(int key) {
        return searchRec(root, key);
    }

    bool BinarySearchTree::searchRec(Node* root, int key) {
        if (root == nullptr) return false;
        if (root->key == key) return true;
        if (root->key < key) return searchRec(root->right, key);
        return searchRec(root->left, key);
    }
```

4.4 中序遍历实现

4.4.1 中序遍历功能简介

功能描述：以中序方式遍历二叉搜索树，并打印每个节点的键值。

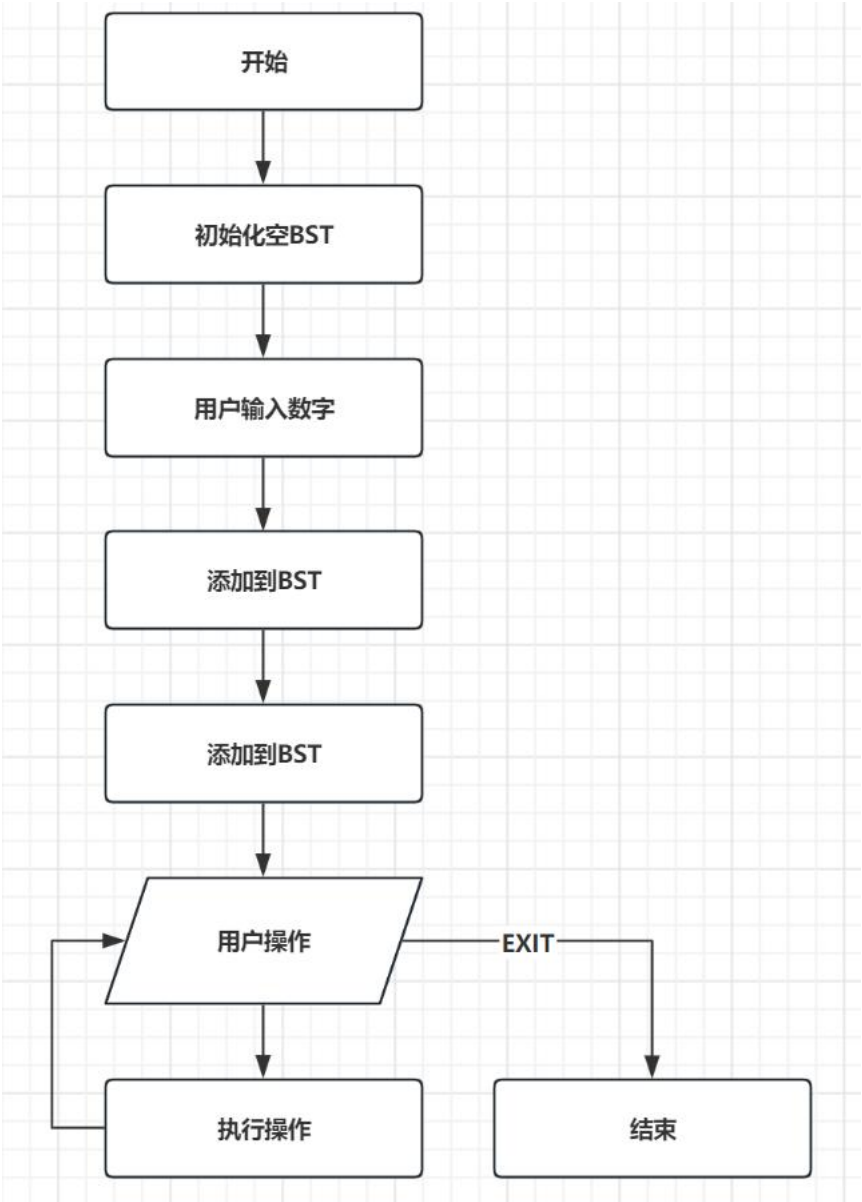
4.4.2 中序遍历核心代码

```
    void BinarySearchTree::inorderTraversal() {
        inorderTraversalRec(root);
        cout << endl;
    }

    void BinarySearchTree::inorderTraversalRec(Node* root) {
```

```
        if (root != nullptr) {
            inorderTraversalRec(root->left);
            cout << root->key << " ";
            inorderTraversalRec(root->right);
        }
    }
```

4.8 系统总体功能流程图



5 设计小结

5.1 数据结构和类设计

本项目以二叉搜索树（BST）作为核心数据结构，允许高效的搜索、插入和排序操作。BST 的每个节点都符合特定的顺序规则，使得树保持平衡，并实现了快速的查找和插入时间复杂度。

通过递归方法实现了 BST 的核心操作，包括插入、搜索和中序遍历，这些方法简化了复杂树结构的处理。

5.2 功能与非功能需求

程序采用面向对象的方法，将节点（Node）和二叉搜索树（BinarySearchTree）封装为类。这种封装提高了代码的模块化和重用性。

BinarySearchTree 类包含了管理树结构（如插入、搜索）的方法，而 Node 类表示树中的单个节点。

设计了一个简单直观的文本用户界面（TUI），允许用户执行基本操作，如构建树、插入节点、搜索节点和打印树。

通过循环菜单和用户输入，程序提供了与用户交互的方式，使用户能够轻松地管理和测试 BST。

在 BST 的操作中，递归方法虽然简化了代码，但也可能导致大数据集下的性能问题（如栈溢出）。因此，对于更复杂或大型的数据集，可能需要考虑迭代方法或其他数据结构。

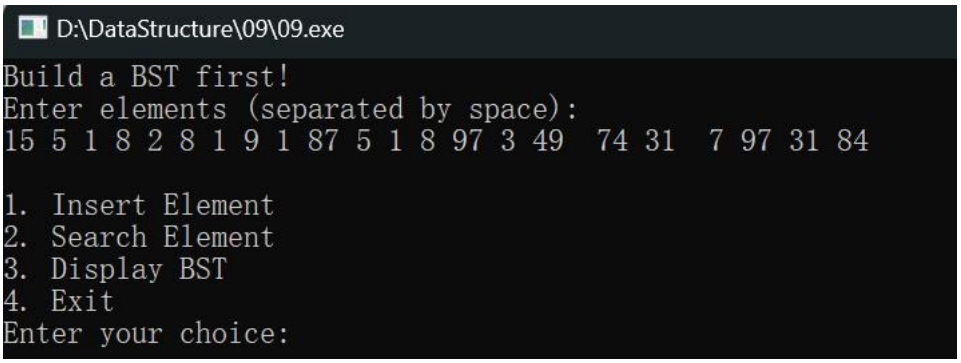
当前 BST 实现在最坏情况下可能会退化成链表，导致操作效率降低。未来可以考虑实现自平衡的二叉搜索树，以保持操作的最优时间复杂度。

总的来说，这个项目展示了基本的二叉搜索树操作和面向对象设计原则的应用，同时也提供了未来改进和扩展的多个方向。

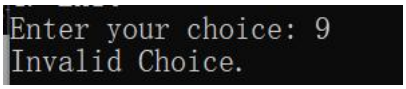
6 软件测试

6.1 输入测试

6.1.1 正常输入



6.1.2 输入超界/非法



结论：符合输入逻辑判断

6.2 输出测试

结论：符合输出逻辑，且正确。