数据结构实习报告

**题目：学生信息管理系统**

|  |  |
| --- | --- |
| **学生：** | 数据233 20231714314 范毅 |
| 数据231 20231714315 程浩 |
| 数据231 20231714319 刘翔 |
| **日期：** | 2024年12月 25 日 |

1. 问题分析

学生信息管理系统旨在对学生的基本信息进行高效管理，包括增、删、改、查、排序、筛选等功能。本系统的目标是通过用户友好的界面和清晰的操作流程，满足以下要求：

1. **数据录入与保存**：实现学生信息的录入、存储，并支持永久保存。
2. **信息查询**：支持按学号或姓名快速查找学生。
3. **信息管理**：包括学生信息的增、删、改功能。
4. **排序与筛选**：
   1. 支持按学号、姓名、班级或性别排序。
   2. 支持按班级或性别筛选。
5. **统计与展示**：实时显示筛选结果及对应人数，并能够完整显示所有学生的信息。
6. **数据存储**：学生数据保存于本地文件中，实现数据的持久化存储。

通过本系统的实现，可以有效解决手动管理学生信息的效率低、易出错等问题，为学校或教学单位提供方便快捷的解决方案。

1. 数据结构

##### ****1) 逻辑结构****

系统中采用了**二叉搜索树**（Binary Search Tree，BST）作为主要的数据结构，二叉搜索树是一种高效的查找和排序数据结构，其节点具有以下性质：

* 左子树节点的值均小于父节点。
* 右子树节点的值均大于父节点。

此外，通过集合操作临时存储筛选结果。

##### ****2) 物理结构****

* **二叉搜索树**：采用链式存储结构，节点中包含学生信息（数据域）以及左右子节点指针（地址域）。
* **本地文件**：学生数据存储在文本文件中，采用顺序存储结构。

1. 系统实现

**1. 节点类 Node**

节点类用于表示二叉搜索树的基本单元，每个节点存储一个学生对象及其左右子节点的引用。

public class Node {

Student student; // 数据域：存储学生信息

Node left, right; // 地址域：指向左右子节点的指针

/\*\*

\* 构造方法

\* @param student 包含学生信息的对象

\*/

public Node(Student student) {

this.student = student;

this.left = null;// 初始时左子节点为空

this.right = null;// 初始时右子节点为空

}

}

**2. 学生类 Student**

学生类表示单个学生的基本信息，包括班级、学号、姓名、性别和年龄，并提供对这些信息的操作方法。  
此外，实现了 Comparable 接口，以支持基于学号的排序。

**3. 二叉搜索树类 BinarySearchTree**

二叉搜索树用于存储学生信息，并提供对这些信息的增、删、改、查等操作。

3.1 成员变量

root：表示树的根节点。

3.2 插入操作

插入操作将学生信息插入到二叉搜索树中，按学号进行排序。

public void insert(Student student) {

root = insertRecursive(root, student);// 从根节点开始插入

}

private Node insertRecursive(Node current, Student student) {

if (current == null) {

// 当前节点为空，创建一个新的节点并返回

return new Node(student);

}

// 比较学号决定插入方向

if (student.compareTo(current.student) < 0) {

current.left = insertRecursive(current.left, student);// 插入左子树

} else if (student.compareTo(current.student) > 0) {

current.right = insertRecursive(current.right, student);// 插入右子树

}

return current;// 返回当前节点

}

3.3 删除操作

删除操作通过递归方式在树中找到匹配的节点，并根据情况（叶子节点、有一个子节点、有两个子节点）调整树结构。

public void delete(String id) {

root = deleteRecursive(root, id);// 从根节点开始递归删除

}

private Node deleteRecursive(Node current, String id) {

if (current == null) {

return null;// 树中不存在此学号，直接返回

}

// 找到要删除的节点

if (id.equals(current.student.getStudentId())) {

// 叶子节点

if (current.left == null && current.right == null) {

return null;

}

// 只有一个子节点

if (current.left == null) {

return current.right;

}

if (current.right == null) {

return current.left;

}

// 有两个子节点

Student smallestValue = findSmallestValue(current.right);

current.student = smallestValue;

current.right = deleteRecursive(current.right, smallestValue.getStudentId());

return current;

}

// 递归处理左子树或右子树

if (id.compareTo(current.student.getStudentId()) < 0) {

current.left = deleteRecursive(current.left, id);// 递归删除左子树

} else {

current.right = deleteRecursive(current.right, id);// 递归删除右子树

}

return current;

}

// 递归寻找左子树中最左的节点

private Student findSmallestValue(Node root) {

return root.left == null ? root.student : findSmallestValue(root.left);

}

3.4 查找操作

通过递归搜索方式，在树中查找匹配的学生信息。

public Student searchById(String id) {

return searchByIdRecursive(root, id);// 从根节点开始查找

}

private Student searchByIdRecursive(Node current, String id) {

if (current == null || current.student.getStudentId().equals(id)) {

return (current != null) ? current.student : null;// 当前节点为空，表示未找到

}

if (id.compareTo(current.student.getStudentId()) < 0) {

return searchByIdRecursive(current.left, id);// 左子树查找

} else {

return searchByIdRecursive(current.right, id);// 右子树查找

}

}

3.5 遍历操作

中序遍历（Inorder Traversal）实现对学生信息的有序输出。

public void inorderTraversal() {

inorderRecursive(root);// 从根节点开始遍历

}

private void inorderRecursive(Node current) {

if (current != null) {

inorderRecursive(current.left);// 遍历左子树

System.out.println(current.student);// 输出当前节点的学生信息

inorderRecursive(current.right);

}

}

3.6 筛选操作

使用遍历方式将符合条件的学生信息存入列表，并进行展示。

public List<Student> filterByCondition(String field, String value) {

List<Student> filteredList = new ArrayList<>();// 将树中所有节点按中序遍历提取为列表

filterRecursive(root, field, value, filteredList);

return filteredList;

}

private void filterRecursive(Node current, String field, String value, List<Student> resultList) {

if (current != null) {

filterRecursive(current.left, field, value, resultList);

// 筛选条件判断

switch (field) {

case "class":

if (current.student.getStudentClass().equals(value)) {

resultList.add(current.student);

}

break;

case "gender":

if (current.student.getGender().equals(value)) {

resultList.add(current.student);

}

break;

}

filterRecursive(current.right, field, value, resultList);

}

}

3.7 排序操作

二叉搜索树中默认按学号排序，通过中序遍历即可实现排序输出。若需要其他排序方式（如按姓名、班级），可将学生列表提取出来并应用自定义比较器。

public List<Student> getAllStudents() {

List<Student> students = new ArrayList<>();

inorderToList(root, students);

return students;

}

private void inorderToList(Node current, List<Student> list) {

if (current != null) {

inorderToList(current.left, list);

list.add(current.student);

inorderToList(current.right, list);

}

}

1. 运行结果（根据功能测试，每个测试数据运行的结果截图）

导入



添加



修改



删除



筛选



查找



排序



1. 总结

**分工情况**：

范毅负责数据结构选择、二叉搜索树的实现与测试。

程浩负责学生类设计、系统功能整合与用户交互界面开发。

刘翔负责数据导入导出模块、排序与筛选功能的实现。

**心得体会**：

通过本次项目开发，组员们熟练掌握了二叉搜索树的实现及应用。

理解了数据结构的选择对程序效率的影响，并学会了如何设计面向对象程序。

合作开发提升了团队沟通能力，分工协作提高了开发效率。