

数据结构课程设计

|  |
| --- |
| 题目 |
| 图书馆书籍管理系统模拟 |

|  |  |
| --- | --- |
| **学生姓名** | **罗毅** |
| **学生学号** | **302023315014** |
| **所在班级** | **软件工程2303** |
| **任课教师** | **王海霞** |
| **所在学院** | **计算机科学与技术学院** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **提交日期：** | **2024年12月** |  |

# 摘 要

本课程设计旨在模拟一个图书馆书籍管理系统（LBMS）。系统采用红黑树作为底层数据结构，以高效地存储和管理图书管理员信息及书籍信息。该系统包括两个主要子模块：图书馆管理员子模块和书籍管理子模块。图书馆管理员子模块实现了管理员的注册、登录以及对书籍的增删改查等功能；书籍管理子模块则支持书籍的增删改查操作，并且能够处理借阅状态更新。此外，系统还设计了菜单子模块和文件操作子模块，增强了系统的用户友好性和数据持久化能力。通过详细的设计思路与分析，系统不仅满足了基本的功能需求，还在测试过程中表现出良好的稳定性和可靠性。

关键字: 红黑树, 图书馆书籍管理系统, C++, 管理员, 书籍管理, 文件操作, 菜单, 数据持久化

# 目 录

[摘 要](#_Toc19209) 2

[目 录 3](#_Toc15755)

[第1章 课程设计内容与要求](#_Toc25850) 4

[1.1 问题描述 4](#_Toc6757)

[1.2 基本要求 4](#_Toc17262)

[1.3 实现提示 5](#_Toc22628)

[1.4 运行结果要求 5](#_Toc22628)

[第2章 设计思路与分析 5](#_Toc25984)

[2.1 系统总体设计 5](#_Toc9638)

[2.1.1 系统框架选择 6](#_Toc3184)

[2.2 系统功能设计 6](#_Toc2379)

[2.3 类的具体实现 1](#_Toc16197)3

[2.4 主程序的设计 2](#_Toc16197)4

[2.5 本章小结 2](#_Toc16197)5

[第3章 调试与分析 2](#_Toc23871)5

[3.1 主要难点分析 2](#_Toc24094)5

[3.1.1 红黑树的实现 2](#_Toc5450)5

[3.2 调试错误分析 2](#_Toc9178)5

[3.3 本章小结](#_Toc9178) 31

[第4章 测试结果与分析 3](#_Toc30624)2

[4.1 测试结果 3](#_Toc27354)2

[4.2 本章小结 4](#_Toc3825)7

[第5章 附录（源代码） 4](#_Toc32129)8

# 课程设计内容与要求

## 1.1 问题描述

在图书馆书籍管理系统中，管理员需要经常对图书信息进行查询和验证，如查询某本书是否在库、验证某本书的借阅状态等。由于图书信息的查询和验证频率很高，系统有必要有效地组织这些图书信息，从而快速查找和验证图书。另外，系统也会经常添加新入库的图书、删除损坏的图书和更新图书的借阅状态等操作，因此，系统必须采用动态结构，在添加、删除或更新后，依然能保证验证过程的快速。请采用相应的数据结构模拟图书馆书籍管理系统，其功能要求包括图书查询、图书状态更新、新书添加和旧书删除等。

## 1.2 基本要求

1、要求自己编程实现二叉树结构及其相关功能，以存储图书信息，不允许使用标准模板类的二叉树结构和函数。同时要求根据二叉树的变化情况，进行相应的平衡操作，即AVL平衡树操作，四种平衡操作都必须考虑。测试时，各种情况都需要测试，并附上测试截图；

2、要求采用类的设计思路，不允许出现类以外的函数定义，但允许友元函数。

3、要求采用多文件方式：.h文件存储类的声明，.cpp文件存储类的实现，主函数main存储在另外一个单独的cpp文件中。如果采用类模板，则类的声明和实现都放在.h文件中。

4、不强制要求采用类模板；不强制要求采用可视化窗口，但若采用可视化窗口可适当提高考核分数；要求源程序中有相应注释；

5、要求测试例子要比较详尽，各种极限情况也要考虑到，测试的输出信息要详细易懂，表明各个功能的执行正确；

## 1.3 实现提示

1、图书信息(即图书编号和借阅状态)可以存储在文件中，当程序启动时，从文件中读取所有的图书信息，并建立合适的查找二叉树；

2、查询过程时，需要根据图书的编号，检索整个二叉树，找到匹配的图书编号，进行验证；更新图书借阅状态时，也需要检索二叉树，找到匹配项后进行更新，同时更新文件中存储的图书借阅状态；

3、添加图书时，不仅需要在文件中添加，也需要在二叉树中添加相应的节点；删除图书时，也是如此；

4、图书馆书籍管理系统可以联系实际，扩展相应功能。

## 1.4 运行结果要求

要求有程序菜单，能够实现用户的登录管理、图书查询、添加图书、删除图书和更新图书借阅状态功能，实验报告要求有详细的功能测试截图。

# 第2章 设计思路与分析

## 2.1 系统总体设计

在图书馆书籍管理系统（LBMS）中，频繁的查询操作是常见的需求。如果依赖链表实现查询，在数据量很大的情况下，通过遍历查询的效率会非常低。因此，选择使用平衡二叉树来获得高效的搜索性能，将每次查询的时间复杂度控制在 O(log n) 。考虑到系统初始化时以及其他情况下存在的数据增删场景，选择红黑树用于图书管理员数据和书籍数据的存储。红黑树作为一种自平衡二叉查找树，可以在兼顾良好平衡性的前提下，同时拥有极佳的数据增删效率。

图书馆书籍管理系统（LBMS）主要有两个子模块，分别是**图书馆管理员子模块**与**书籍管理子模块**，分别用于实现图书馆管理员的账号登录与管理以及图书馆书籍的管理。在两个子模块中，分别设计图书馆管理员类和书籍类，利用红黑树的数据结构存储，实现增删改查的相应功能，对应地设计了图书馆管理员管理类和书籍管理类以实现相关功能。除此之外，还设计了**菜单子模块**和**文件操作子模块**，方便图书管理员更好的交互和信息管理。

### 2.1.1 系统框架选择

在Windows系统中使用c++语言，编译器版本为GCC 8.1.0，在Visual Studio Code 1.95.3版本进行开发。

## 2.2 系统功能设计

书馆书籍管理系统（LBMS）主要有两个子模块，分别是**图书馆管理员子模块**与**书籍管理子模块，**除此之外，还设计了**菜单子模块**和**文件操作子模块。**

### 2.2.1 图书馆管理员子模块

在该模块中，设计了图书馆管理员类和图书馆管理员管理类，主要实现的功能包括图书馆管理员的注册、登录以及管理员对书籍的相关管理。其中，每一个图书馆管理员的相关信息都存储在红黑树中。

2.2.1.1 图书馆管理员的注册

图书馆管理员的注册需要在正确输入秘钥之后方可注册相关信息，以下为注册的具体流程图。

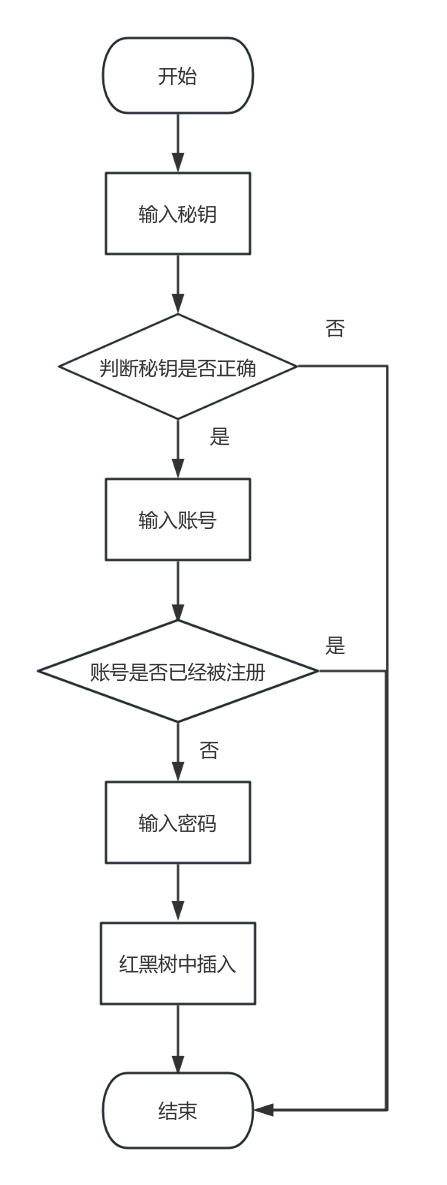


图1 图书馆管理员注册流程图

2.2.1.2 图书馆管理员的登录

图书馆管理员在登录时需要输入账号和密码，在红黑树中查找后确认账号存在并且密码成功之后方可登录，否则登录失败。以下为图书馆管理员登录的具体流程图。

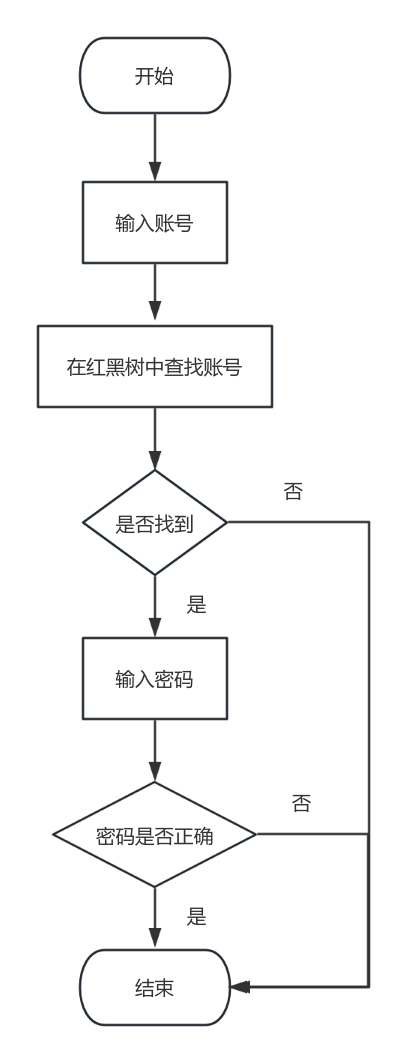
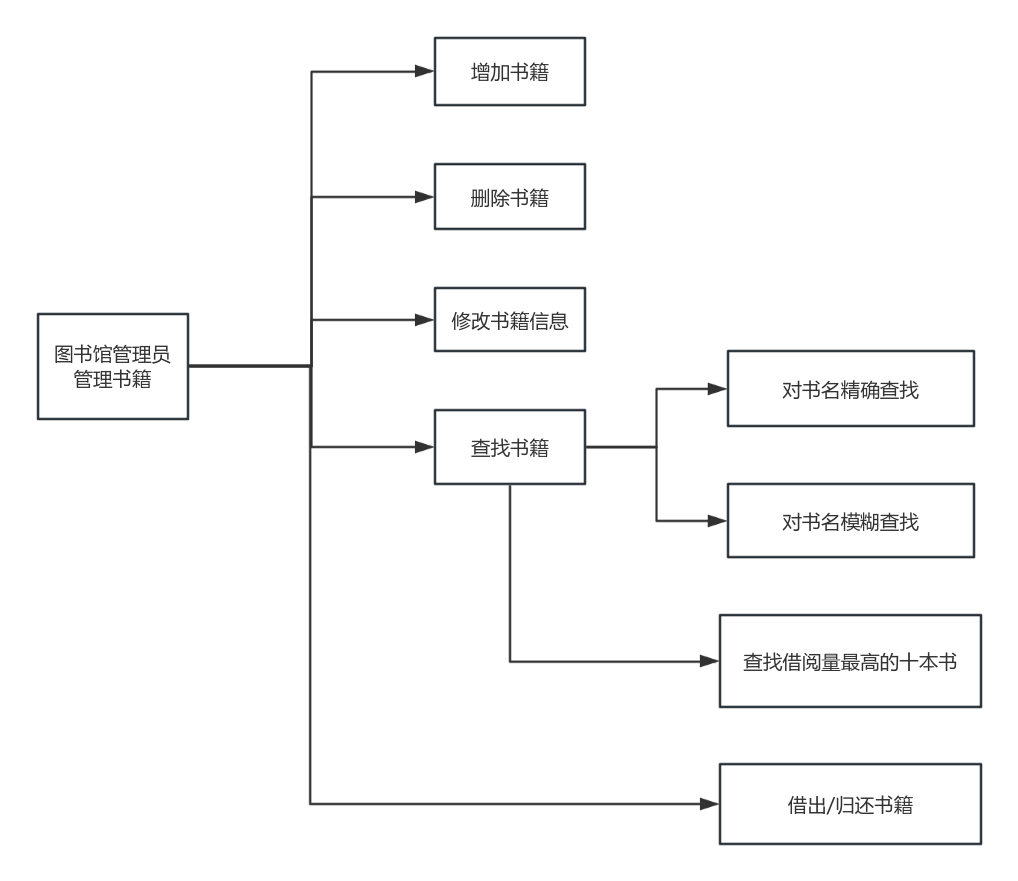


图2 图书馆管理员登录流程图

2.2.1.3 图书馆管理员对书籍的管理

图书馆管理员对书籍的管理主要体现在对书籍的增删改查以及书籍的借出和归还。其中增和删分别为对书籍的增加和删减，改为对书籍信息的改动。查为查询书籍的相关信息，支持通过书名精确搜索或者对书名进行模糊搜索。除此之外，还支持对借阅量最高的十本书籍的查找和对所有借出书籍的查找。对书籍的管理功能具体如以下流程图所示。



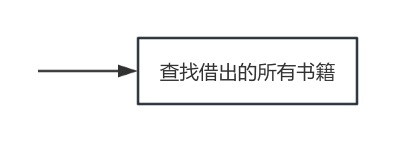
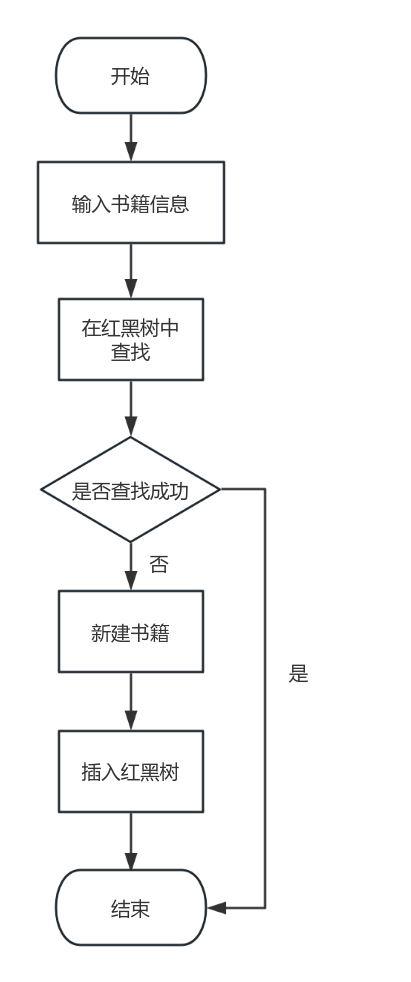


图3 图书馆管理员管理功能流程图



添加书籍的流程图如图所示：

图4 添加书籍流程图

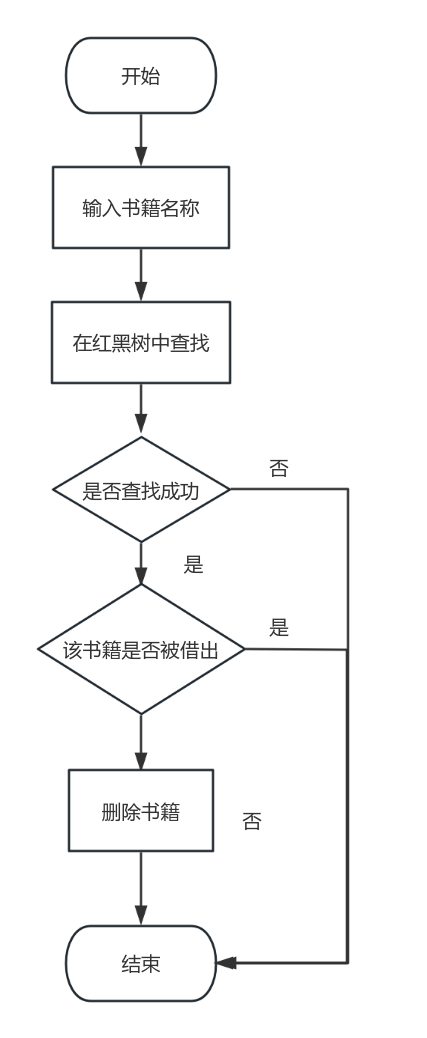
删除书籍需根据书籍的名称删除，流程图如下所示：

图5 删除书籍流程图

查找书籍、更新书籍信息与删除书籍的流程类似，输入书籍的名称，在红黑树中找到之后查找书籍会显示书籍相关信息，更新书籍信息会提示管理员输入新信息。

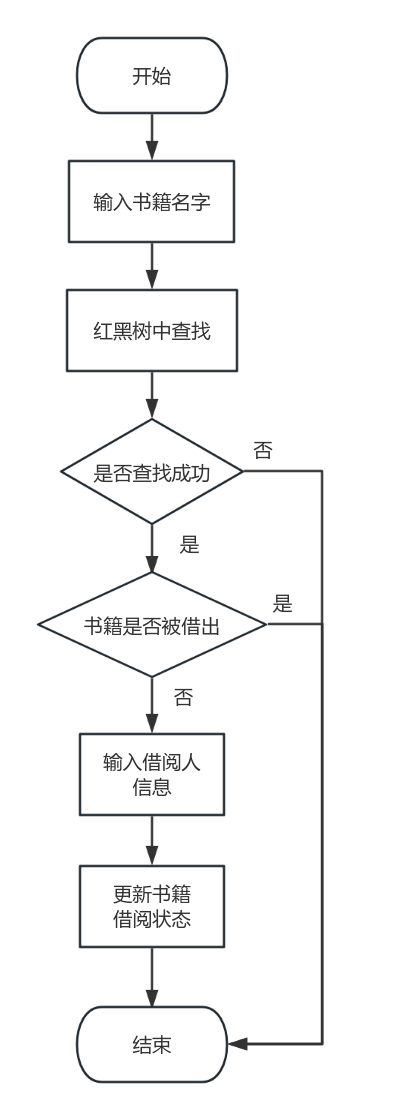
借阅书籍的流程图如下所示：

图6 借阅书籍流程图

归还书籍的流程与借阅书籍的流程类似。

### 2.2.2 书籍管理子模块

在该模块中，主要实现了书籍类和书籍管理类，将书籍存储在红黑树中进行，并且为图书管理员模块提供了对书籍增删改查的功能。

### 2.2.3 菜单子模块

采用了可读性较好的菜单模块，便于管理员交互。

### 2.2.4 文件操作子模块

该模块中，采用csv文件保存管理员账号和书籍的相关信息，可以将管理员信息和书籍信息保存在本地文件，当二者的信息通过程序改动之后，会将相应改动同时保存在本地文件。

## 2.3 类的具体实现

系统共包括6个类，分别是图书管理员类（admin）、图书管理员管理类（admin\_manager）、书籍类（book）、书籍管理类（book\_manager）、红黑树类（rbtree）、菜单类（menu）。由于图书管理员类和书籍类都存储在红黑树中，所以使用模板类实现红黑树类，可同时用于这两种数据类型。类之间的具体关系如下所示。

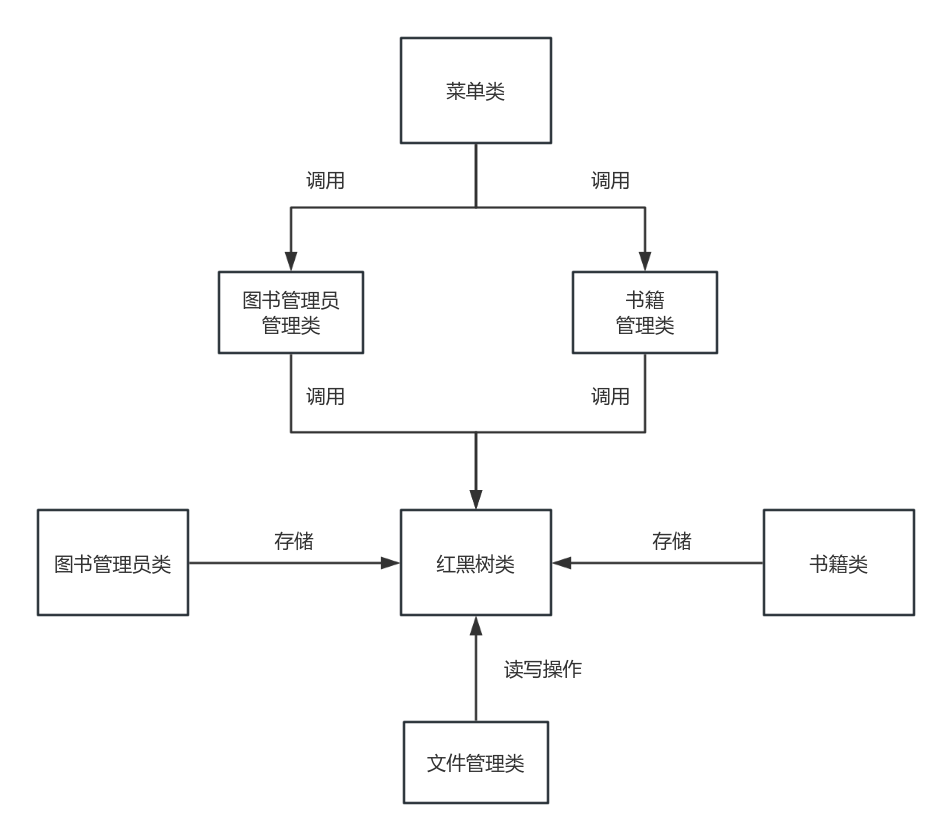


图7 类之间的关系图

2.3.1红黑树类（rbtree）

红黑树是一种特殊的二叉查找树，有以下几个特性：

1. 每个节点是红色或者是黑色
2. 根节点是黑色
3. 每个叶子节点为黑色，其中的叶子结点指的是为空的叶子结点
4. 如果一个节点是红色，它的子节点必须是黑色的
5. 从一个节点到每个叶子的路径上都包含相同数目的黑色节点。

红黑树示意图如下：



图8 红黑树示例

红黑树中比较复杂的操作主要有两个，分别是插入和删除。下面展开具体介绍。

2.3.1.1红黑树类的插入

1. 将红黑树当作一棵二叉查找树，将节点插入

由于红黑树本身就是一棵二叉查找树，插入节点之后，必然还是一棵二叉查找树，所以可以先按照二叉查找树的性质插入节点

1. 将插入的节点涂上红色

如果涂上红色，那么性质（5）自然满足，后续中的维护操作更加简单

1. 维护插入节点后的红黑树，使其重新成为一棵红黑树
2. 由于插入的是红色节点，所以性质（1）和性质（5）必然满足，而插入非空节点不会影响空的叶子节点，所以性质（2）必然满足
3. 如果插入之前是空树，需要把根节点的颜色涂为黑色，此时满足性质（3）。如果插入之前不是空树，必然满足性质（3）
4. 对于性质（4），需要分成多种情况讨论，示意图中的N、G、P、U分别表示当前插入节点、祖父节点、父节点、叔叔节点。

情况一：红黑树为空

此时直接插入节点作为根节点，并且将其修改为黑色

情况二：插入节点的父节点为黑色，无需任何操作

由于父节点为黑色，不会违反性质（4）

情况三：如果父节点P和叔叔节点U二者都是红色，（此时新插入节点N作为P的左子节点或右子节点都属于情形3），则我们可以将它们两个涂为黑色并重绘祖父节点G为红色（用来保持性质5）。现在我们的新节点N有了一个黑色的父节点P。因为通过父节点P或叔父节点U的任何路径都必定通过祖父节点G，在这些路径上的黑节点数目没有改变。

但是，红色的祖父节点G可能是根节点，这就违反了性质2，也有可能祖父节点G的父节点是红色的，这就违反了性质4。为了解决这个问题，需要对祖父节点G继续处理，即将其当作新加入的节点重新维护整棵红黑树。



图9 红黑树插入情况三

情况四：父节点P是红色而叔父节点U是黑色或空节点，并且新节点N 是其父节点P的右子节点而父节点P是祖父节点的左子节点。在这种情况 下，进行一次左旋调换新节点和其父节点的角色，接着，按情况5进行处理

（如果新节点N是父节点P的左子节点而父节点P又是祖父节点的右子节 点，考虑为同一种情况）



图10 红黑树插入情况四

情况五：父节点P是红色而叔父节点U是黑色或缺少，新节点N是其父节点的左子节点，而父节点P是祖父节点G的左子节点（新节点N是其父节点的右子节点，而父节点P是祖父节点G的右子节点，视为同一种情况）。在这种情况下，对祖父节点G进行一次右旋。以前的祖父节点G是黑色，否则父节点P就不可能是红色。切换父节点P和以前的祖父节点G的颜色。



图11 红黑树插入情况五

2.3.1.2红黑树类的删除

1. 删除

情况1：需要删除的节点有两个非空儿子

在删除节点的右子树中，寻找最小的节点为取代节点，取代要被删除的节点。此时，问题转化为删除这个取代节点。由于这个取代节点不可能有两个非空儿子，此时的问题转为情况2。

情况2：需要删除的节点有一个非空儿子或者没有非空儿子

将其非空子节点（如果没有非空儿子，则使用任意一个空儿子节点）替代该删除节点，和该删除节点的父节点连接。如果删除节点的颜色为黑色，此时性质（4）不满足，需要进行删除后的维护。

1. 删除后的维护（以下示意图中，删除节点的儿子为N，删除节点的父节点为P，N的兄弟为S，S的左儿子为SL，S的右儿子为SR。）

情况1：N是新的根

由于N的颜色是黑色，所以无需操作

情况2：S是红色。

对P左旋，把S转换成N的祖父，接着对调N的父亲和祖父的颜色。现在N有了一个黑色的兄弟和一个红色的父亲（它的新兄弟是黑色因为它是红色S的一个儿子），所以可以接下去按情形4、情形5或情形6来处理。



图12 红黑树删除维护情况二

情况3：N的父亲、S和S的儿子都是黑色的。

在这种情况下，将S涂为红色。此时，通过P的所有路径现在比不通过P的路径少了一个黑色节点，所以仍然违反性质5，所以从情形1开始，在P上重新平衡处理。



图13 红黑树删除维护情况三

情况4： S和S的儿子都是黑色，但是N的父亲是红色。

此时交换N的兄弟和父亲的颜色。这不影响不通过N的路径的黑色节点的数目，但是它在通过N的路径上对黑色节点数目增加了一，添补了在这些路径上删除的黑色节点。



图14 红黑树删除维护情况四

情况5：S是黑色，S的左儿子是红色，S的右儿子是黑色，而N是它父亲的左儿子。

对S做右旋，这样S的左儿子成为S的父亲和N的新兄弟。接着交换S和它的新父亲的颜色。所有路径仍有同样数目的黑色节点，但是现在N有了一个黑色兄弟，他的右儿子是红色的，这样就进入了情况6。



图15 红黑树删除维护情况五

情况6：S是黑色，S的右儿子是红色，而N是它父亲的左儿子。

此时，对N的父亲做左旋，这样S成为P的父亲和S的右儿子的父亲。交换P和S的颜色，并使S的右儿子为黑色。但是，N现在增加了一个黑色祖先：要么P变成黑色，要么P原来是黑色但这增加了一个黑色S祖父。所以，通过N的路径都增加了一个黑色节点，填补了原来黑色节点数量的减少。



图16 红黑树删除维护情况六

下面给出各个类的声明的代码截图

2.3.1.3红黑树类

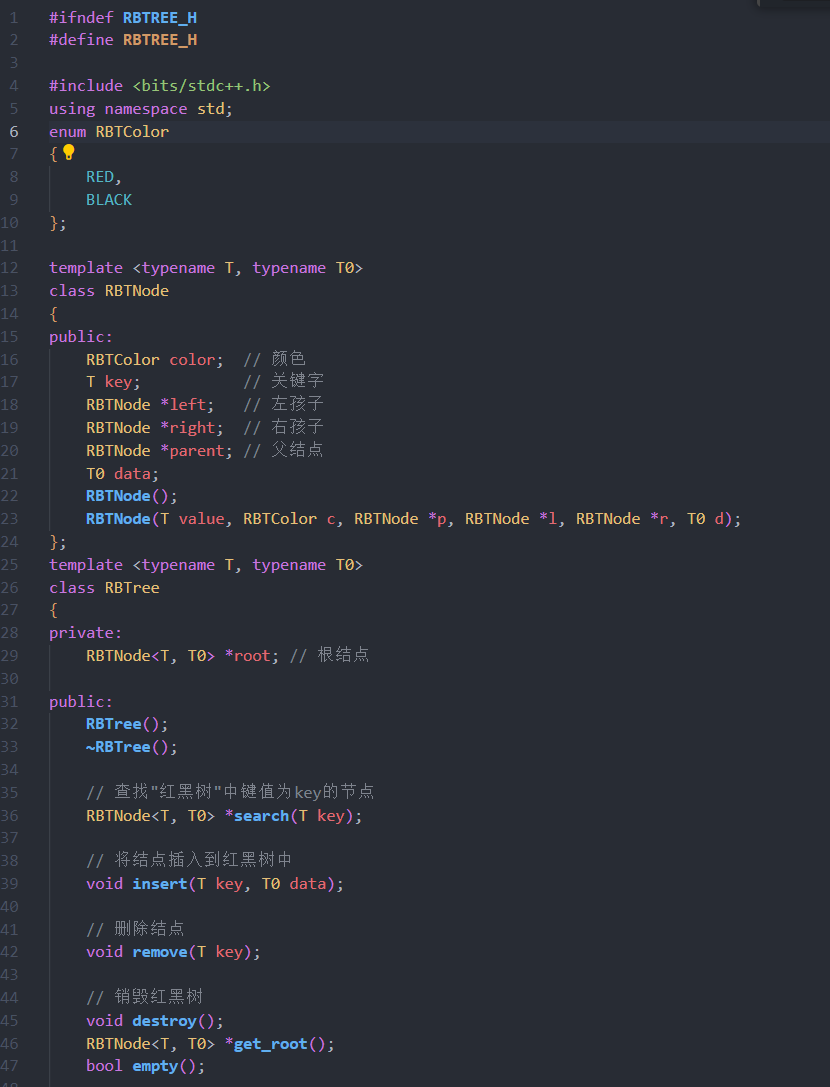


图17 红黑树节点类

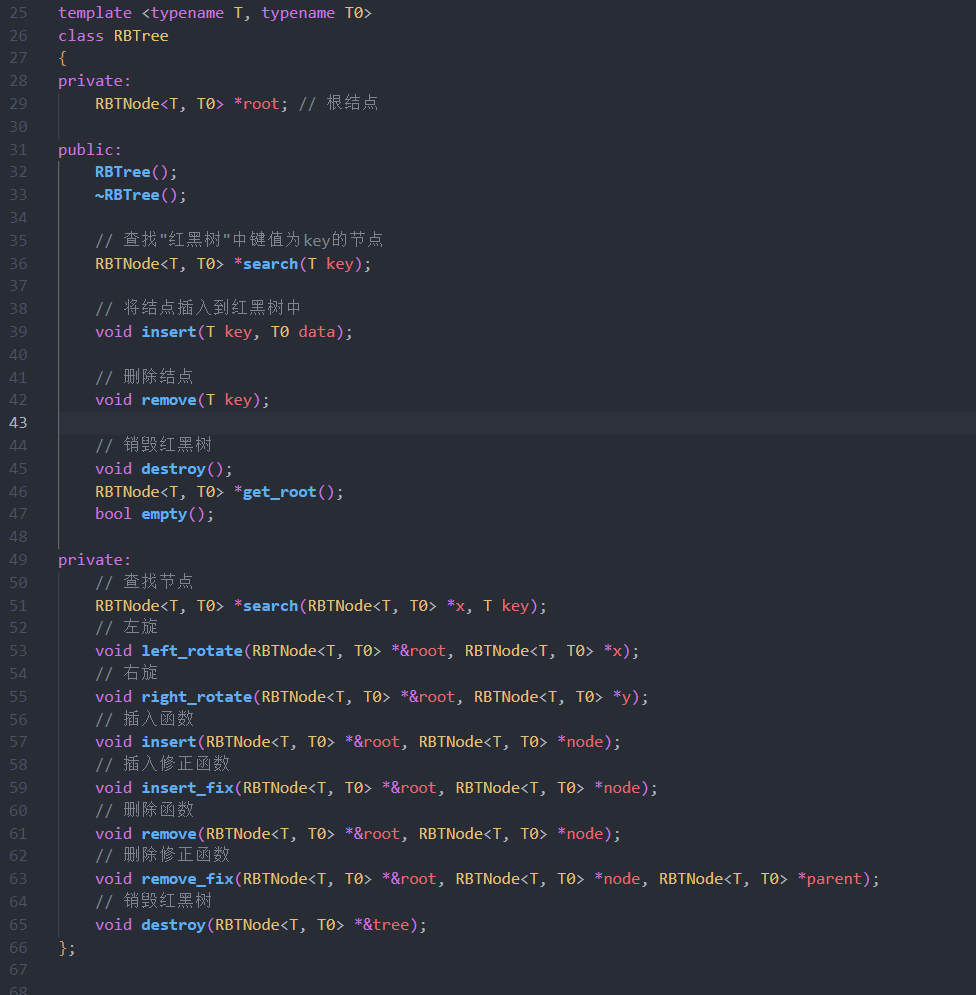
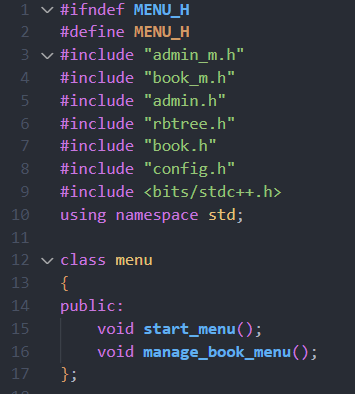


图18 红黑树类



2.3.1.4 menu类

图19 menu类

2.3.2 admin类

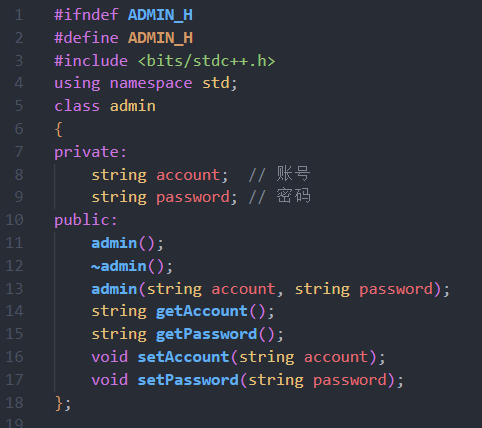


图20 admin类

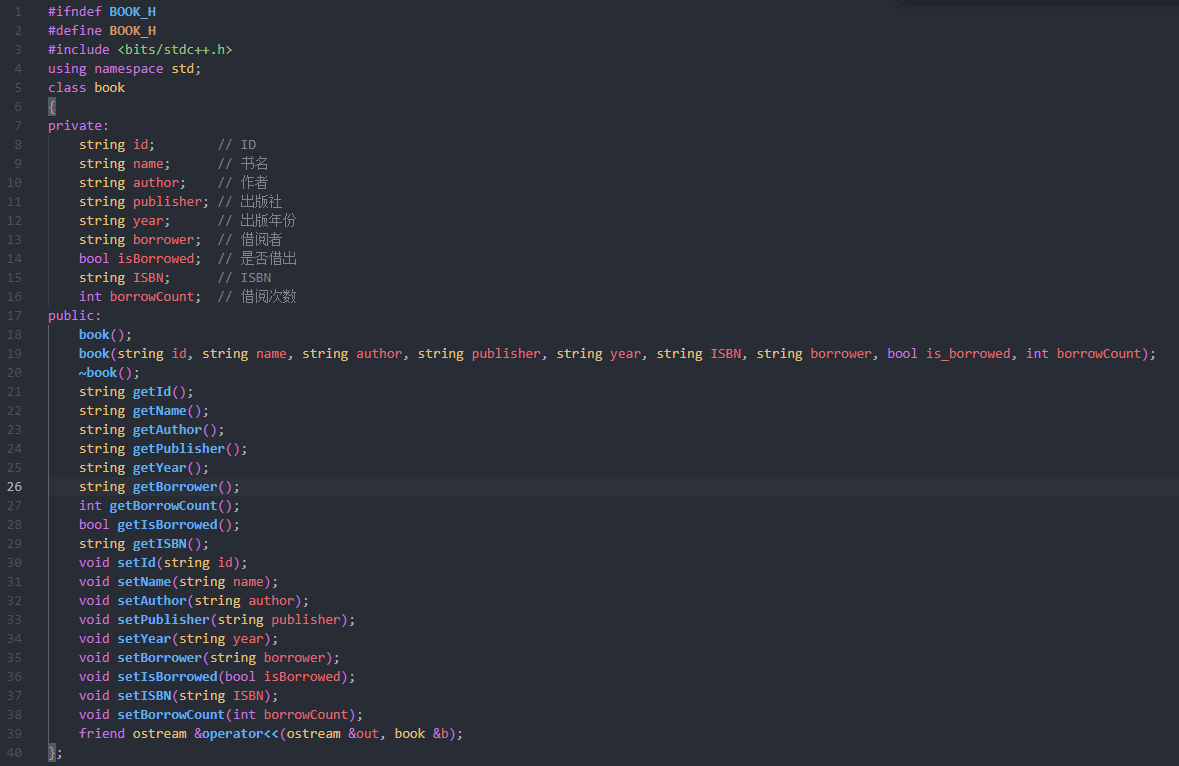
2.3.3 book类

图21 book类

2.3.4 admin\_manager类

用于实现对存储在红黑树中的admin的管理，其中红黑树的key为admin的账号。admin\_manager可以实现管理员注册和管理员登录功能，在创建时会根据本地文件自动初始化，析构时会自动将数据保存到本地文件。

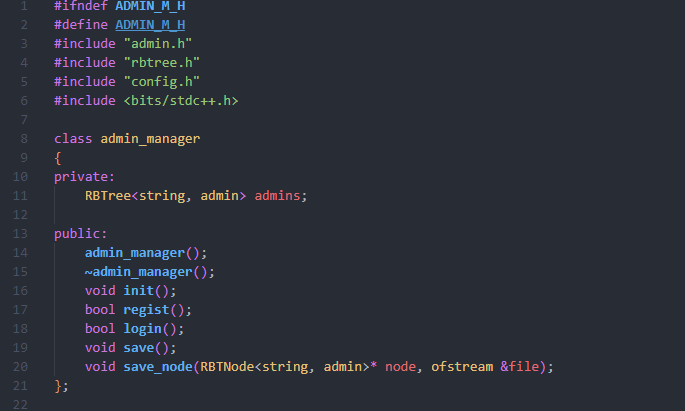


图22 admin\_manager类

2.3.5 book\_manager类

用于实现对存储在红黑树中的book的管理，其中红黑树的key为book的名字。book\_manager类可以实现增删改查以及借出书籍、归还书籍。同时也支持寻找所有已经借出的书籍、模糊搜索和寻找借阅量最大的十本书籍。在模糊搜索时，可以做到不区分大小写，寻找包含输入字符串的书名。在创建book\_manager时会自动根据本地文件初始化，在析构时会自动将数据保存到本地文件。

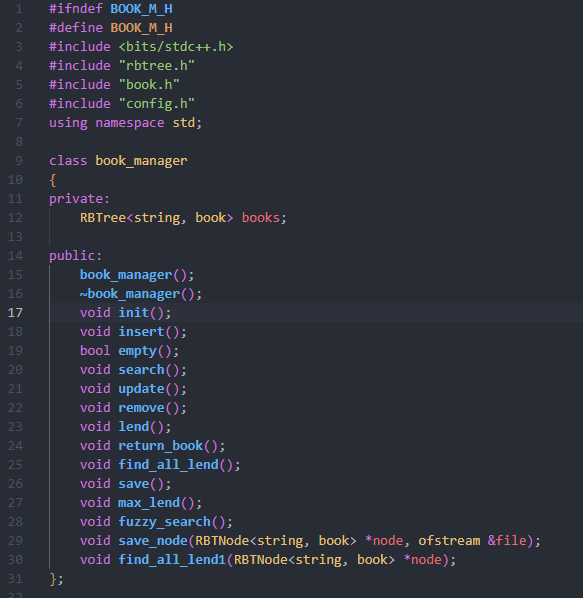


图23 book\_manager类

## 2.4 主程序的设计

在主程序中，通过调用start\_menu()方法启动交互界面。

## 2.5 本章小结

第2章中，详细介绍了图书馆书籍管理系统（LBMS）的设计思路与分析，主要围绕系统总体设计、系统功能设计以及类的具体实现进行了深入探讨。

系统总体设计部分，决定采用红黑树作为底层数据结构来存储图书管理员和书籍信息。这是因为红黑树不仅能够提供高效的搜索性能，而且在处理频繁的数据增删操作时也表现出色。

在系统功能设计方面，构建了两个核心子模块：图书馆管理员子模块和书籍管理子模块。图书馆管理员子模块涵盖了管理员的注册、登录及对书籍的管理功能，如增加、删除、修改和查询书籍信息。书籍管理子模块则专注于书籍的增删改查以及借阅状态的更新。为了增强系统的用户友好性，设计了直观的菜单子模块，并通过文件操作子模块实现了数据的持久化存储。

类的具体实现环节，定义了几个关键类，包括admin、admin\_manager、book、book\_manager、rbtree、menu。这些类之间协同工作，确保了系统的稳定性和可扩展性。

# 

# 第3章 调试与分析

## 3.1 主要难点分析

### 3.1.1 红黑树的实现

难点：红黑树的结构十分复杂，在插入和删除之后维护时需要考虑的情况很多，需要综合各个节点之间的关系和颜色进行旋转变色，实现难度较高。

解决：通过查阅博客和百科自学红黑树的相关知识，在确保充分了解红黑树的性质和构造之后进行红黑树类的实现。

## 3.2 调试错误分析

错误1：

错误内容：程序无法正常启动，报错信息提示无法找到头文件

解决方法：之间的导入路径发生错误，只写了头文件的名称，但是头文件和主程序不在一个文件夹。修改头文件的导入，添加上相对路径即可。



图24 错误截图



图25 修改后截图

错误2：

错误内容：无法正常启动程序，报错提示config.h中定义的变量无法被其他头文件调用。

解决方法：通过上网搜索资料，发现错误在于头文件之间互相引用，造成循环依赖的情况，出现报错。解决方法为将admin\_manager和book\_manager的具体对象实现放在admin\_m.h和book\_m.h的头文件中，config.h不调用这两个头文件，这样就不会形成循环依赖。



图26 错误截图

（这是config.h,在admin\_m.h和book\_m.h中也引用了config.h）

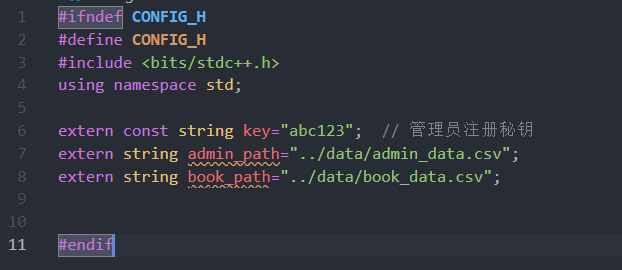


图27 修改后截图

错误3：

错误内容：无法正常启动，报错提示admin\_path和book\_path被初始化并且被声明为extern类型

解决方法：通过上网查阅资料，发现变量在头文件中声明extern时不能被初始化，于是将两个路径的赋值放在了主程序中。

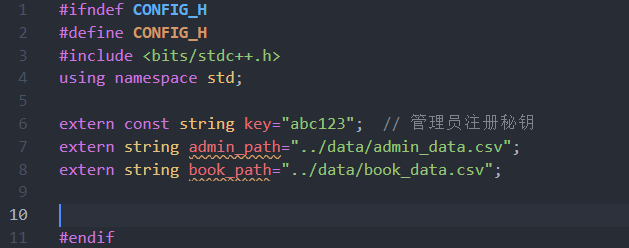


图28 错误截图

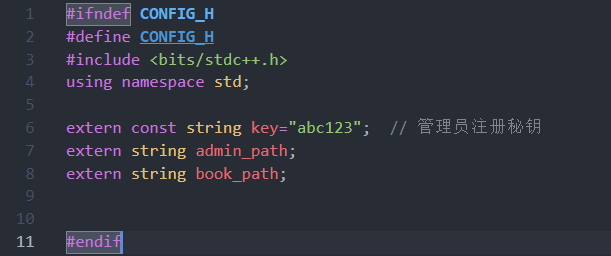


图29 修正后截图1

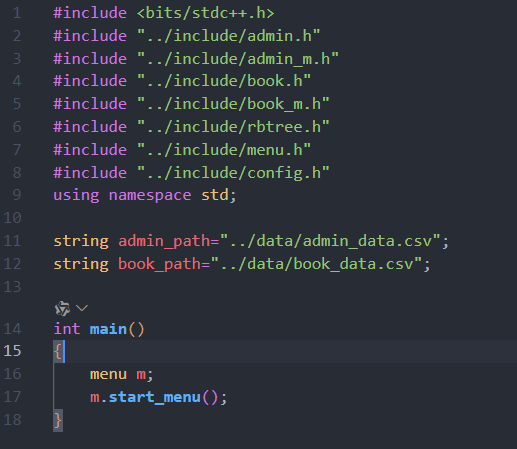


图30 修正后截图2

错误4：

错误内容：程序无法启动，提示admin\_m和book\_m的链接出现问题

解决方法：经过上网查阅资料，发现admin\_m和book\_m还需在cpp文件中实现定义。

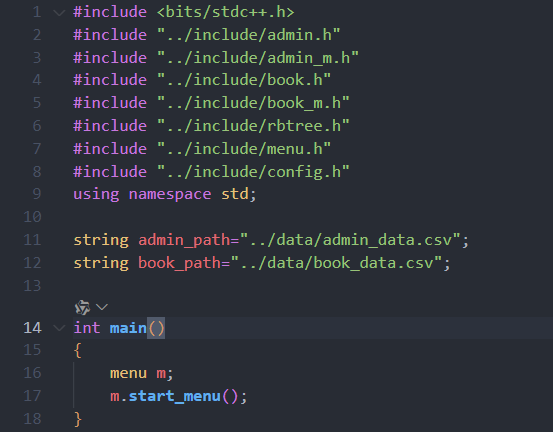


图31 错误截图



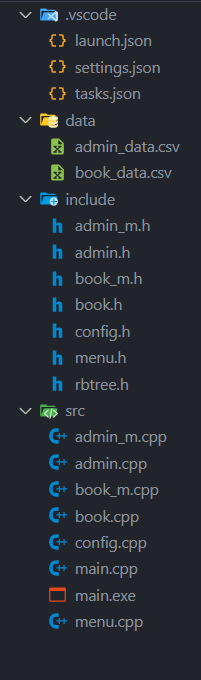
图32 修改后截图

## 

错误4：

错误内容：仔细阅读要求之后，发现代码设计出现问题。在头文件中，应该只出现类的声明，在cpp文件中实现类的定义，而模板类应该同时实现类的定义和声明。

解决方法：重新设计代码结构，设计新的cpp文件用于实现类的定义。



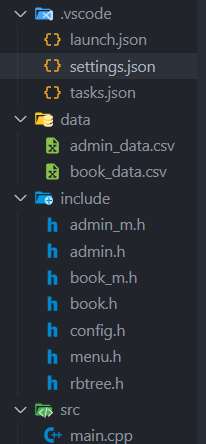


图33 错误截图

图34 修改后截图

## 3.3 本章小结

在本章中，针对图书馆书籍管理系统（LBMS）进行了详细的调试与分析。主要工作包括解决红黑树实现过程中遇到的复杂性问题，以及处理程序开发过程中出现的各种编译和链接错误。通过查阅相关资料并对代码进行调整，成功克服了这些难点，确保了系统的稳定运行。此外，还解决了头文件导入路径、循环依赖、变量初始化以及类定义等问题，从而保证了整个系统能够在无误的情况下正常启动和执行。

# 第4章 测试结果与分析

## 4.1 测试结果

该系统的所有功能均得到实现，并且对各种情况进行考虑。以下为具体的功能测试情况。

### 4.1.1 管理员注册功能

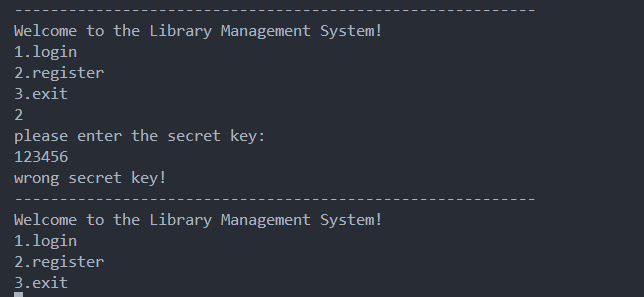


图35 秘钥错误

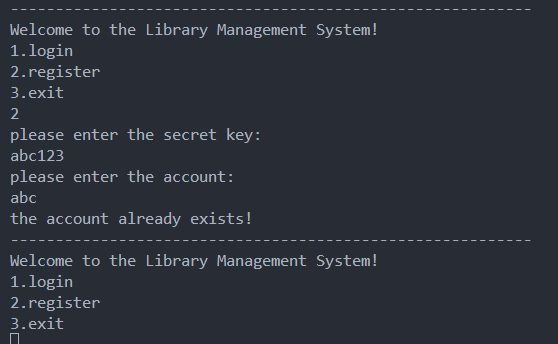


图36 账号已存在

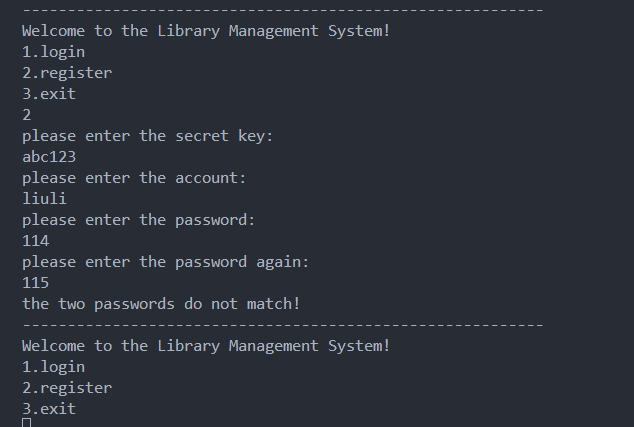


图37 两次密码不匹配

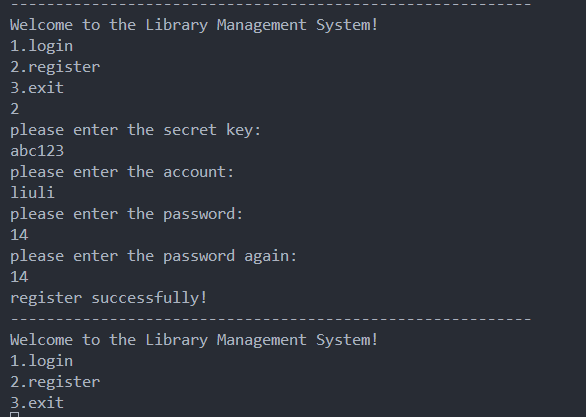


图38 注册成功

4.1.2 管理员登录功能

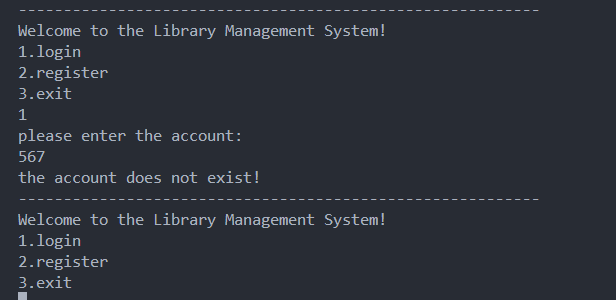


图39 管理员账号不存在

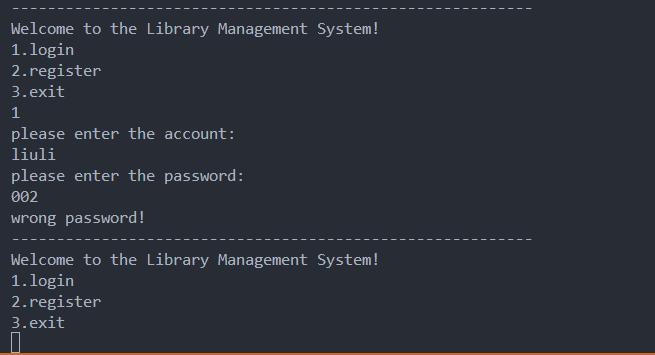


图40 密码错误

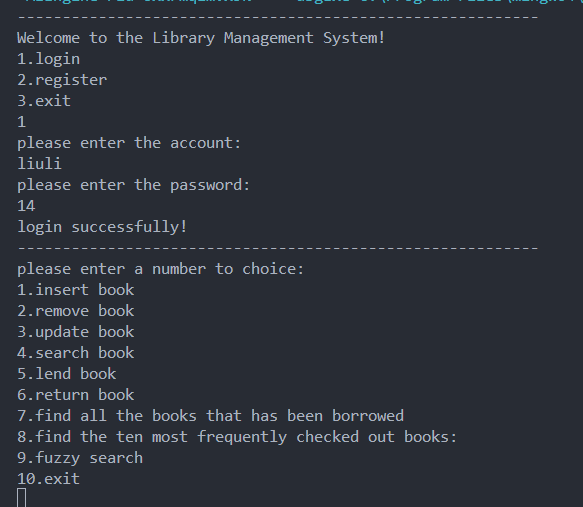


图41 登陆成功

4.1.3添加书籍

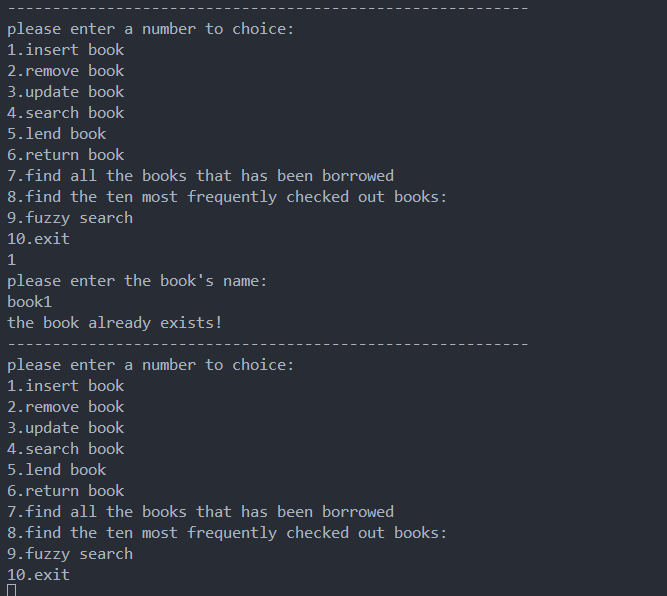


图42 该书籍已存在

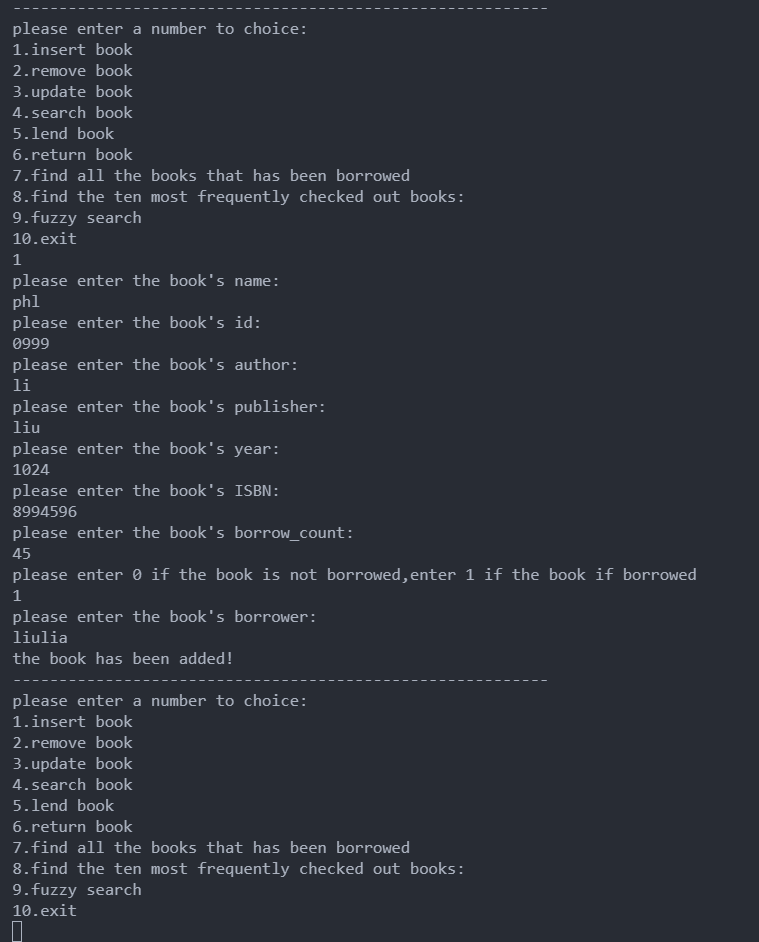


图43 成功加入书籍

4.1.4删除书籍

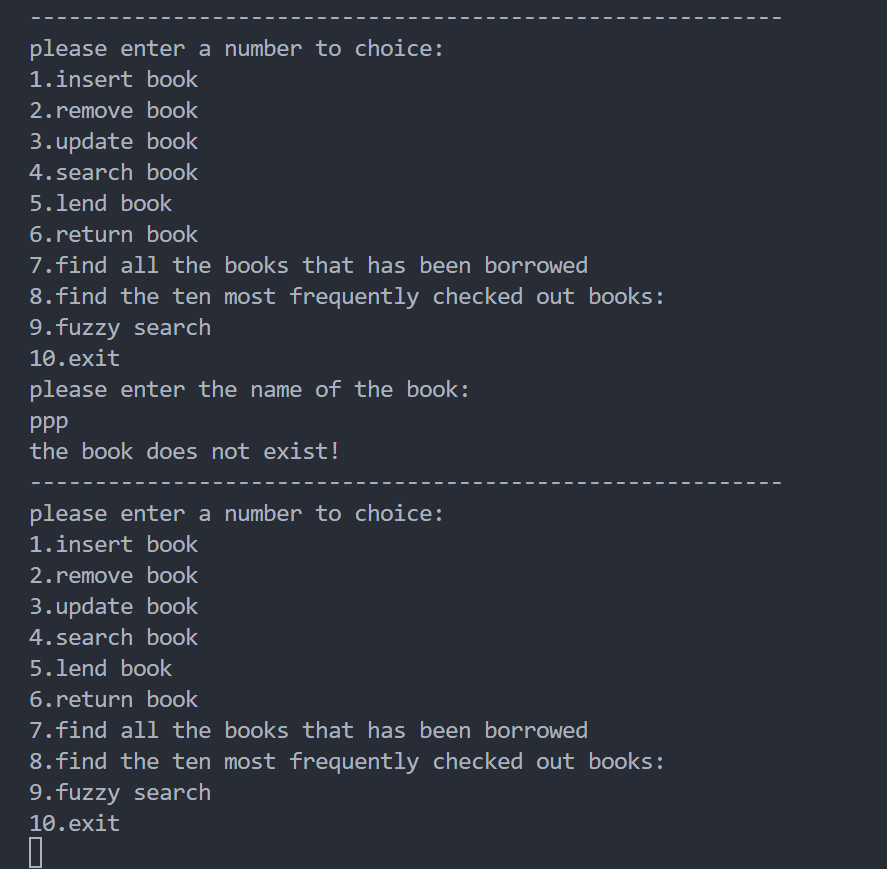


图44 要删除的书籍不存在

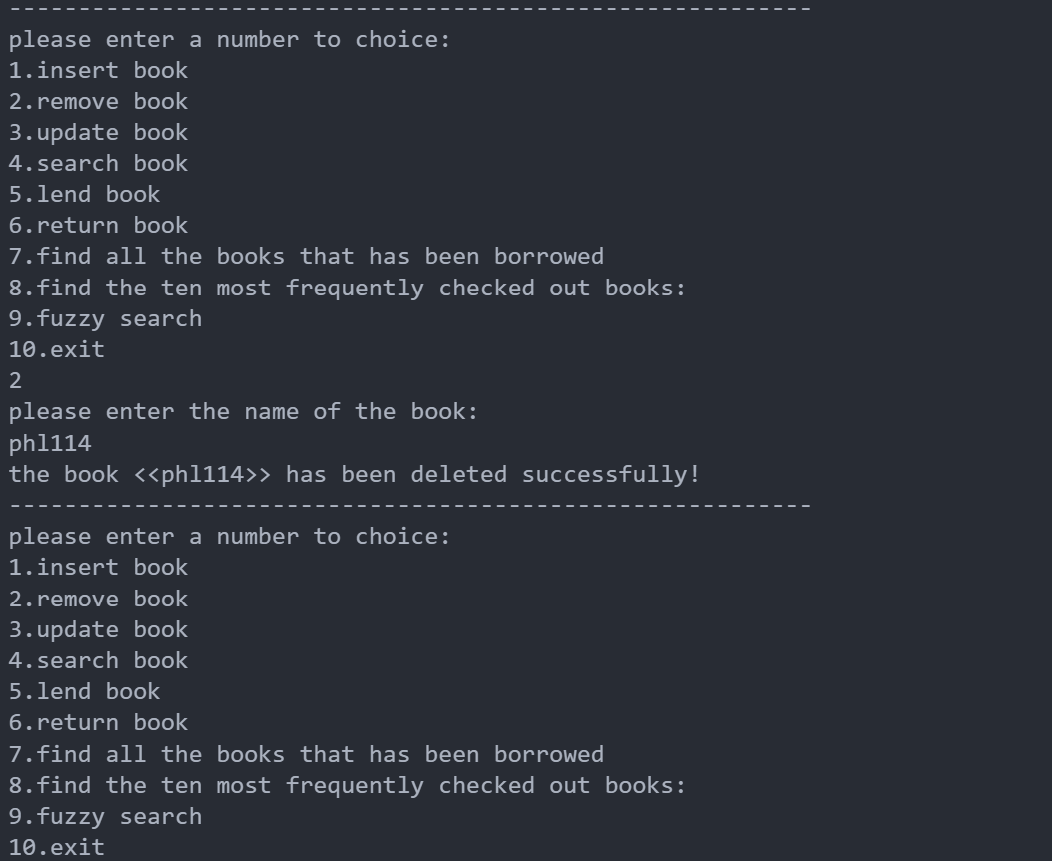


图45 成功删除书籍

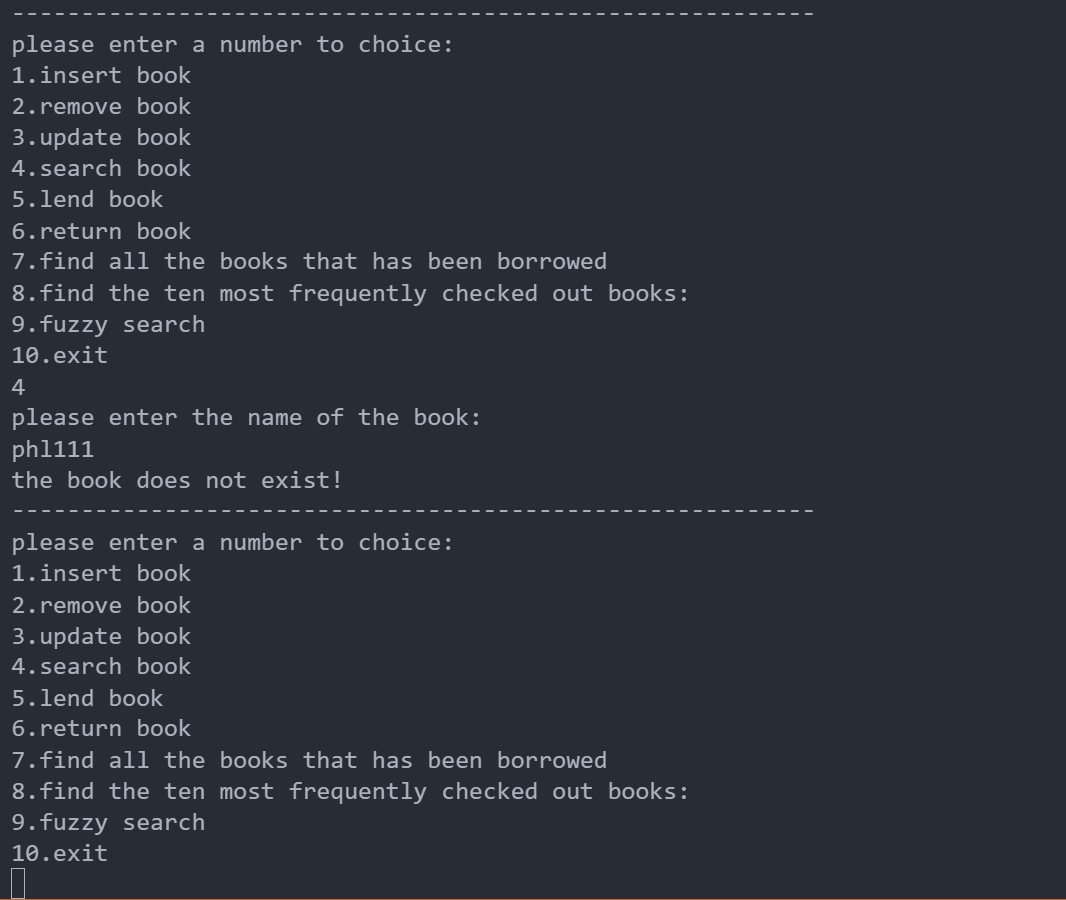
4.1.5 查找书籍（根据书名精确查找）

图46 查找书籍不存在

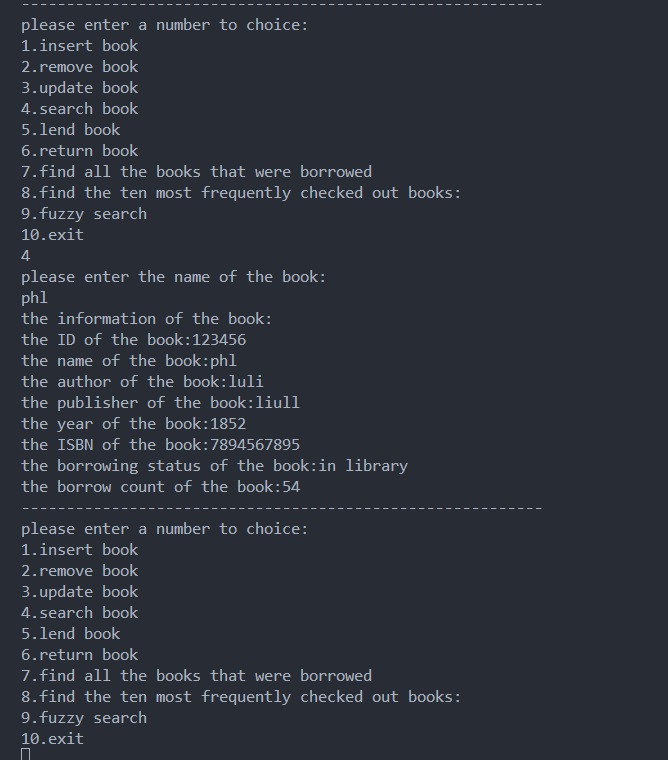


图47 成功查找书籍，显示书籍信息

4.1.6 更新书籍

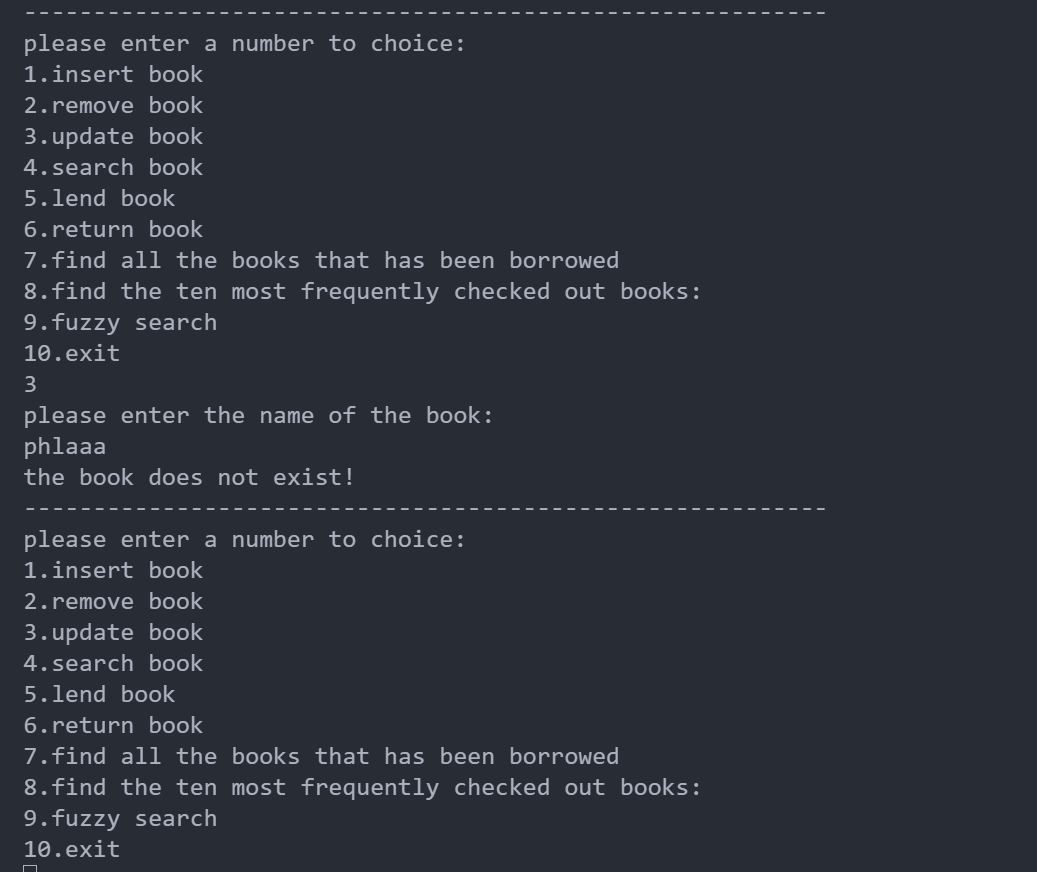


图48 要更新的书籍不存在

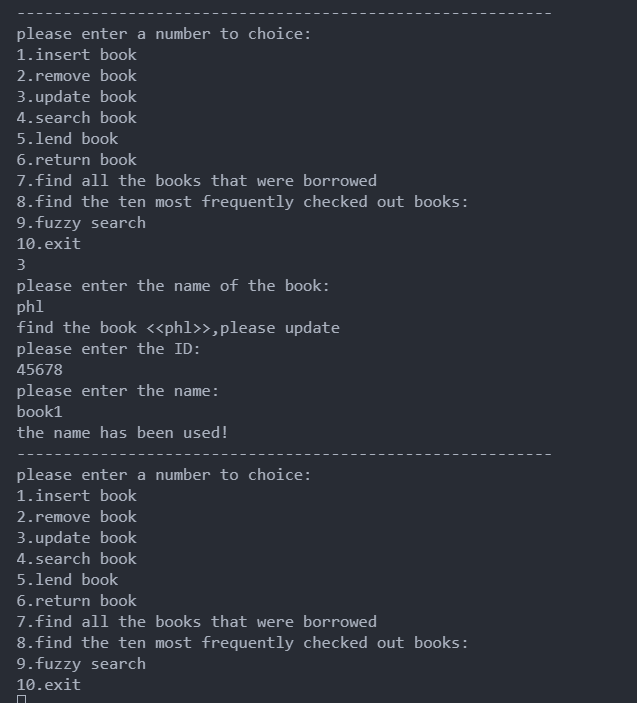


图49 更新的书籍名字与其他书籍重名

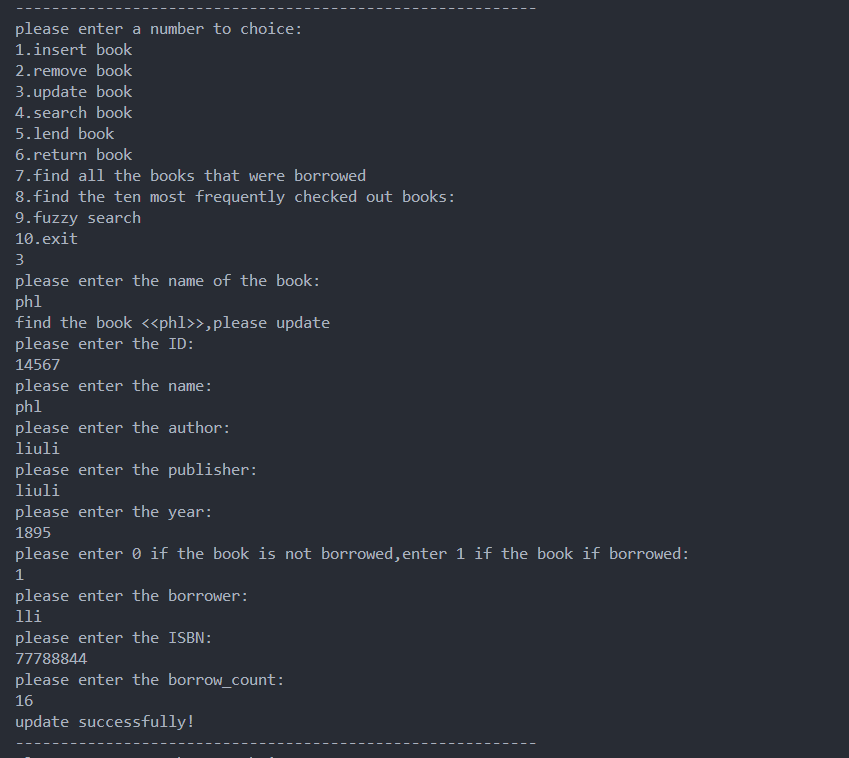


图50 成功更新书籍

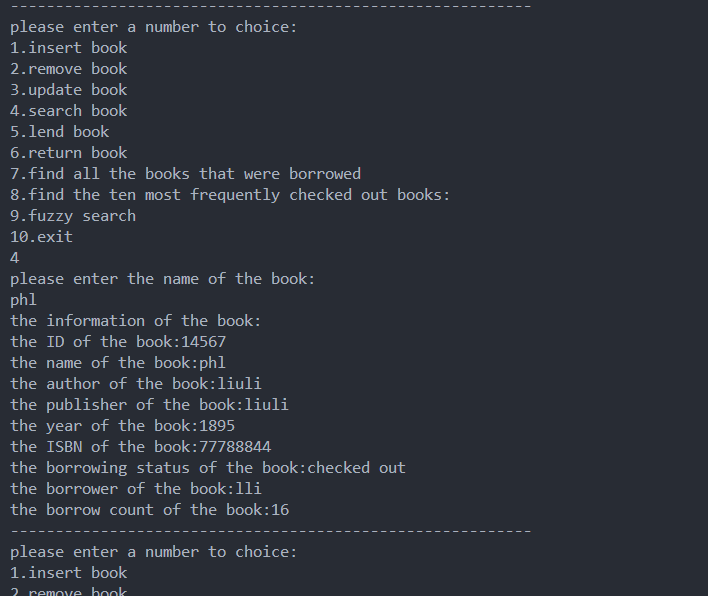


图51 更新后的书籍信息

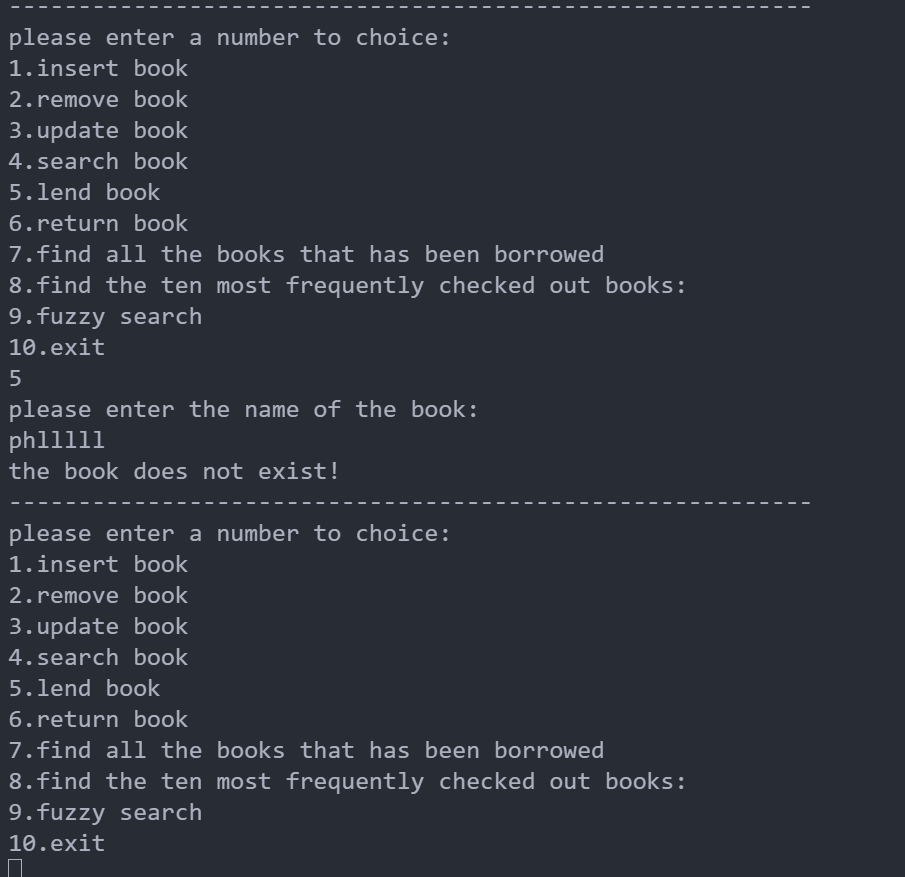
4.1.7 借阅书籍

图52 要借阅的书籍不存在

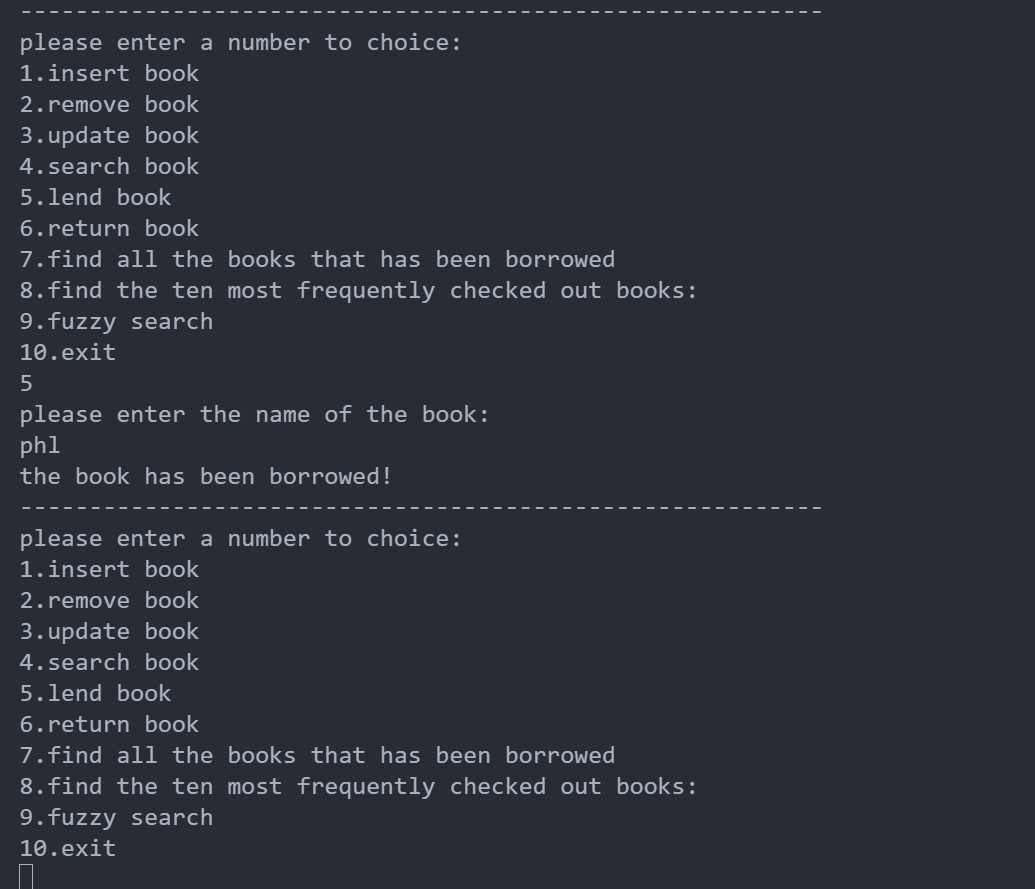


图53 要借阅的书籍已经被借走

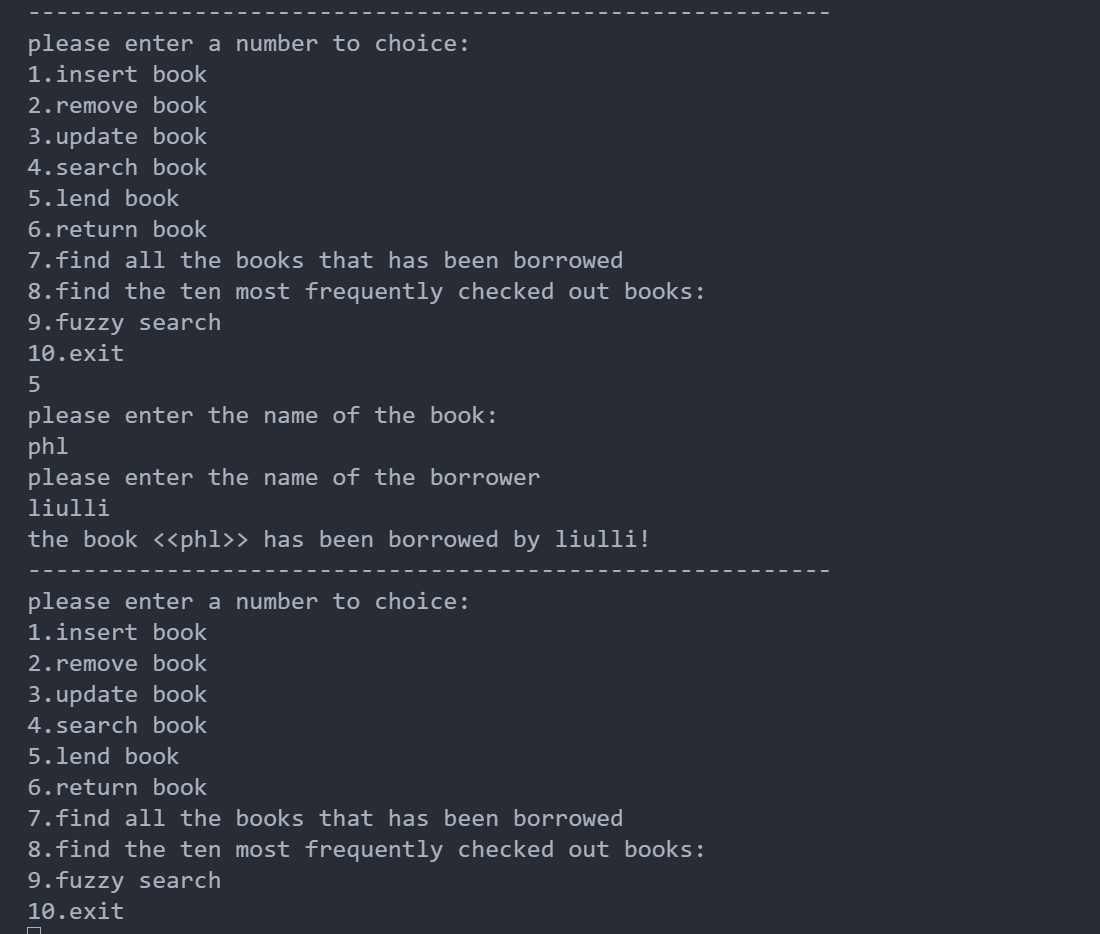


图54 成功借阅书籍

4.1.8 归还书籍

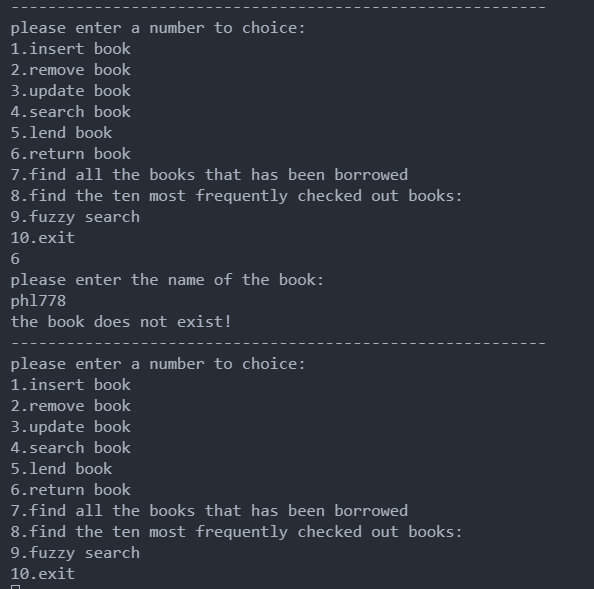


图55 要归还的书籍不存在

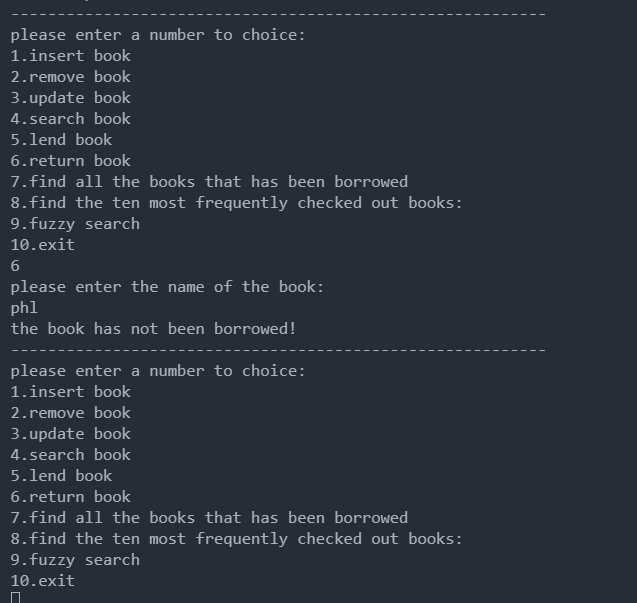


图56 要归还的书籍没有被借出

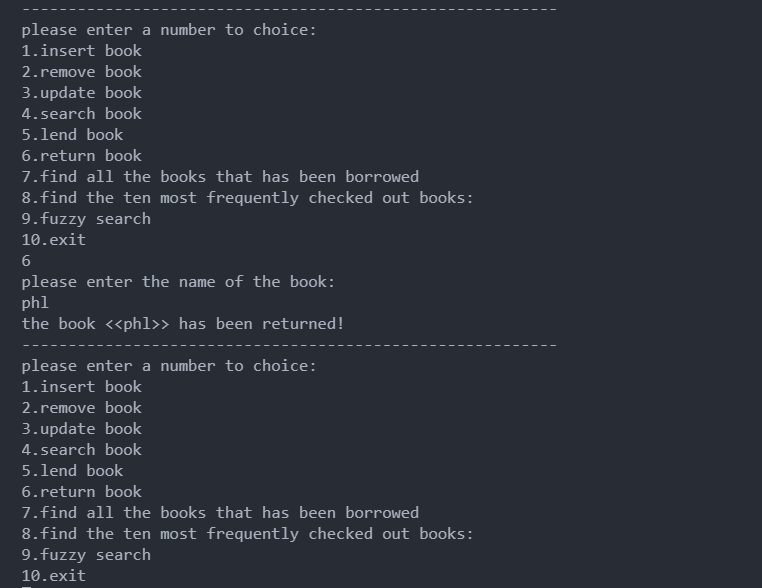


图57 成功归还书籍

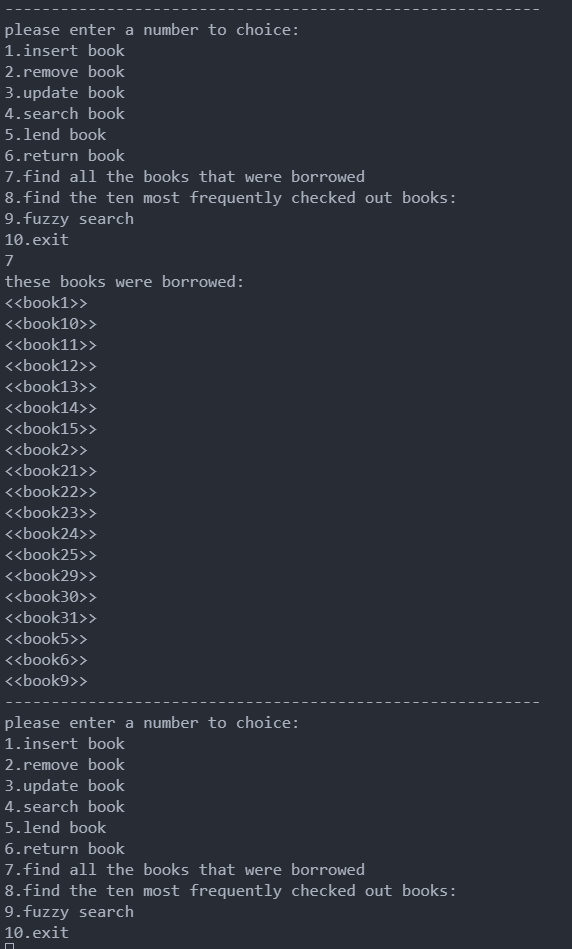
4.1.9查找所有被借出的书籍

图58 成功查找所有被借出的书籍

4.1.10查找借阅量最高的十本书籍

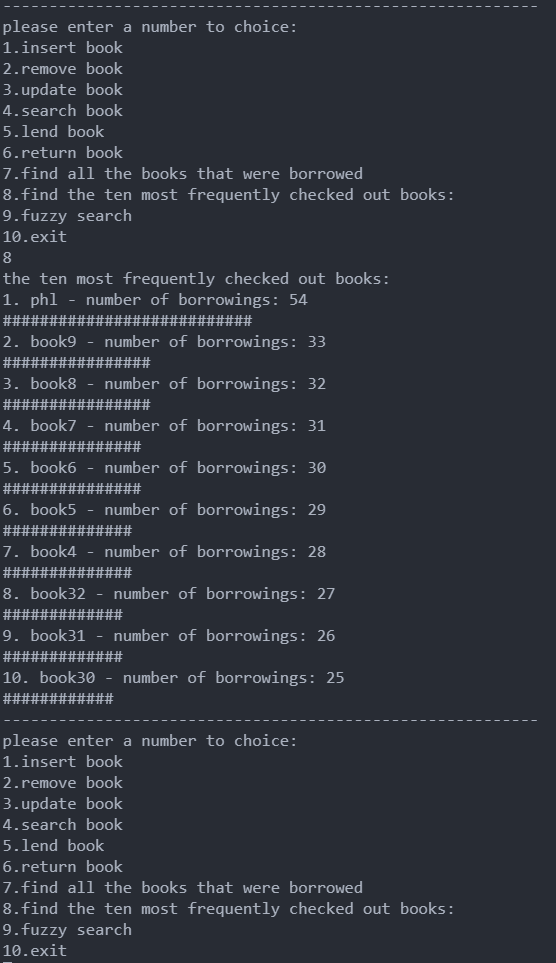


图59 成功查找借阅量最高的十本书籍

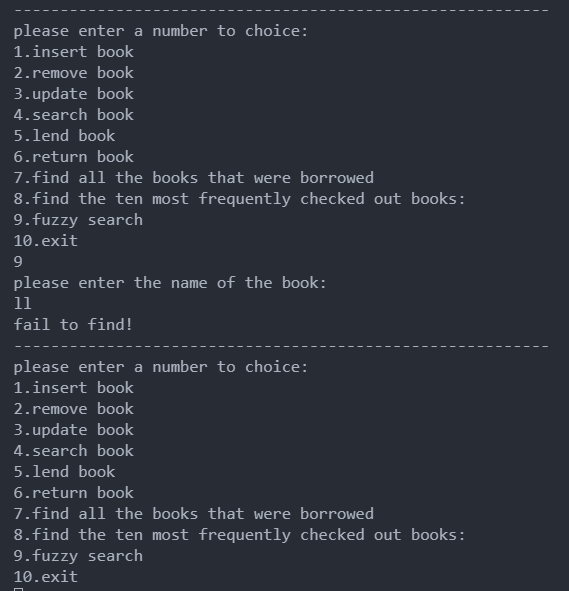
4.1.11模糊搜索

图60 模糊搜索失败

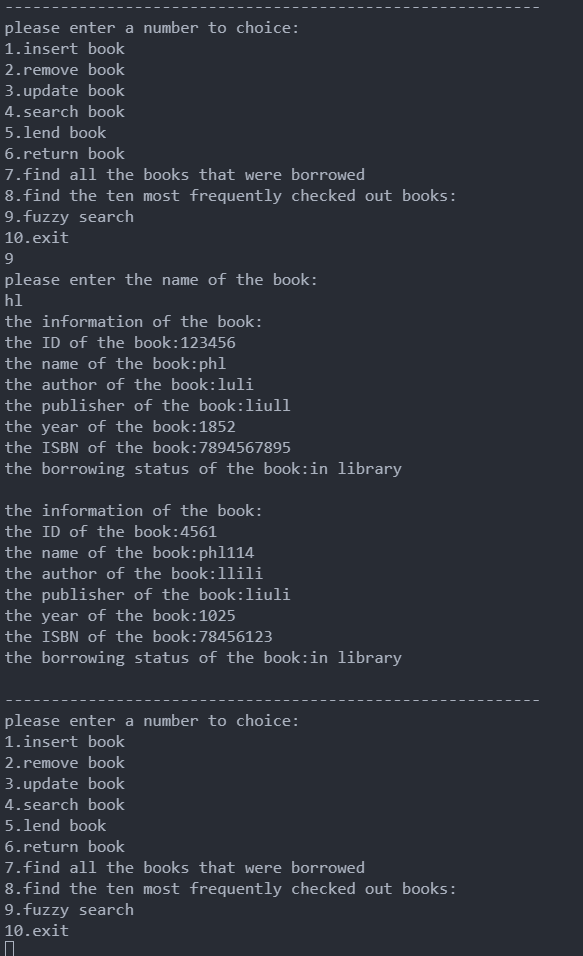


图61 模糊搜索成功

## 4.2 本章小结

在本章中，对图书馆书籍管理系统的所有功能进行了全面的测试，并考虑了各种可能的情况。测试结果表明，系统能够正确地支持管理员注册与登录、图书的添加、删除、查找、更新、借阅及归还等操作。通过对特定场景如账号已存在、密码不匹配、书籍不存在或已被借出等情况的测试，进一步验证了系统的健壮性和可靠性。此外，系统具备模糊搜索功能，可以有效地帮助管理员快速定位到所需信息。所有测试案例均得到了预期的结果，证明了该系统满足设计要求并具有良好的用户体验。

# 

# 第5章 附录（源代码）

# 1、admin\_m.h

#ifndef ADMIN\_M\_H

#define ADMIN\_M\_H

#include "admin.h"

#include "rbtree.h"

#include "config.h"

#include <bits/stdc++.h>

class admin\_manager

{

private:

    RBTree<string, admin> admins;

public:

    admin\_manager();

    ~admin\_manager();

    void init();

    bool regist();

    bool login();

    void save();

    void save\_node(RBTNode<string, admin>\* node, ofstream &file);

};

extern admin\_manager admin\_m;

#endif

1. admin.h

#ifndef ADMIN\_H

#define ADMIN\_H

#include <bits/stdc++.h>

using namespace std;

class admin

{

private:

    string account;  // 账号

    string password; // 密码

public:

    admin();

    ~admin();

    admin(string account, string password);

    string getAccount();

    string getPassword();

    void setAccount(string account);

    void setPassword(string password);

};

#endif

1. book\_m.h

#ifndef BOOK\_M\_H

#define BOOK\_M\_H

#include <bits/stdc++.h>

#include "rbtree.h"

#include "book.h"

#include "config.h"

using namespace std;

class book\_manager

{

private:

    RBTree<string, book> books;

public:

    book\_manager();

    ~book\_manager();

    void init();

    void insert();

    bool empty();

    void search();

    void update();

    void remove();

    void lend();

    void return\_book();

    void find\_all\_lend();

    void save();

    void max\_lend();

    void fuzzy\_search();

    void save\_node(RBTNode<string, book> \*node, ofstream &file);

    void find\_all\_lend1(RBTNode<string, book> \*node);

};

extern book\_manager book\_m;

#endif

1. book.h

#ifndef BOOK\_H

#define BOOK\_H

#include <bits/stdc++.h>

using namespace std;

class book

{

private:

    string id;        // ID

    string name;      // 书名

    string author;    // 作者

    string publisher; // 出版社

    string year;      // 出版年份

    string borrower;  // 借阅者

    bool isBorrowed;  // 是否借出

    string ISBN;      // ISBN

    int borrowCount;  // 借阅次数

public:

    book();

    book(string id, string name, string author, string publisher, string year, string ISBN, string borrower, bool is\_borrowed, int borrowCount);

    ~book();

    string getId();

    string getName();

    string getAuthor();

    string getPublisher();

    string getYear();

    string getBorrower();

    int getBorrowCount();

    bool getIsBorrowed();

    string getISBN();

    void setId(string id);

    void setName(string name);

    void setAuthor(string author);

    void setPublisher(string publisher);

    void setYear(string year);

    void setBorrower(string borrower);

    void setIsBorrowed(bool isBorrowed);

    void setISBN(string ISBN);

    void setBorrowCount(int borrowCount);

    friend ostream &operator<<(ostream &out, book &b);

};

#endif

1. config.h

#ifndef CONFIG\_H

#define CONFIG\_H

#include <bits/stdc++.h>

using namespace std;

extern string key;  // 管理员注册秘钥

extern string admin\_path;

extern string book\_path;

#endif

1. menu.h

#ifndef MENU\_H

#define MENU\_H

#include "admin\_m.h"

#include "book\_m.h"

#include "admin.h"

#include "rbtree.h"

#include "book.h"

#include "config.h"

#include <bits/stdc++.h>

using namespace std;

class menu

{

public:

    void start\_menu();

    void manage\_book\_menu();

};

extern menu m;

#endif

7、rbtree.h

#ifndef RBTREE\_H

#define RBTREE\_H

#include <bits/stdc++.h>

using namespace std;

enum RBTColor

{

    RED,

    BLACK

};

template <typename T, typename T0>

class RBTNode

{

public:

    RBTColor color;  // 颜色

    T key;           // 关键字

    RBTNode \*left;   // 左孩子

    RBTNode \*right;  // 右孩子

    RBTNode \*parent; // 父结点

    T0 data;

    RBTNode();

    RBTNode(T value, RBTColor c, RBTNode \*p, RBTNode \*l, RBTNode \*r, T0 d);

};

template <typename T, typename T0>

class RBTree

{

private:

    RBTNode<T, T0> \*root; // 根结点

public:

    RBTree();

    ~RBTree();

    // 查找"红黑树"中键值为key的节点

    RBTNode<T, T0> \*search(T key);

    // 将结点插入到红黑树中

    void insert(T key, T0 data);

    // 删除结点

    void remove(T key);

    // 销毁红黑树

    void destroy();

    RBTNode<T, T0> \*get\_root();

    bool empty();

private:

    // 查找节点

    RBTNode<T, T0> \*search(RBTNode<T, T0> \*x, T key);

    // 左旋

    void left\_rotate(RBTNode<T, T0> \*&root, RBTNode<T, T0> \*x);

    // 右旋

    void right\_rotate(RBTNode<T, T0> \*&root, RBTNode<T, T0> \*y);

    // 插入函数

    void insert(RBTNode<T, T0> \*&root, RBTNode<T, T0> \*node);

    // 插入修正函数

    void insert\_fix(RBTNode<T, T0> \*&root, RBTNode<T, T0> \*node);

    // 删除函数

    void remove(RBTNode<T, T0> \*&root, RBTNode<T, T0> \*node);

    // 删除修正函数

    void remove\_fix(RBTNode<T, T0> \*&root, RBTNode<T, T0> \*node, RBTNode<T, T0> \*parent);

    // 销毁红黑树

    void destroy(RBTNode<T, T0> \*&tree);

};

template <typename T, typename T0>

RBTNode<T,T0>::RBTNode() : left(NULL), right(NULL), parent(NULL){}

template <typename T, typename T0>

RBTNode<T,T0>::RBTNode(T value, RBTColor c, RBTNode<T,T0> \*p, RBTNode<T,T0> \*l, RBTNode<T,T0> \*r, T0 d)

{

    key = value;

    color = c;

    parent = p;

    left = l;

    right = r;

    data = d;

}

template <typename T, typename T0>

bool RBTree<T, T0>::empty()

{

    return root == NULL;

}

template <typename T, typename T0>

RBTNode<T, T0> \*RBTree<T, T0>::get\_root()

{

    return root;

}

template <typename T, typename T0>

RBTree<T, T0>::RBTree()

{

    root = NULL;

}

template <typename T, typename T0>

RBTree<T, T0>::~RBTree()

{

    destroy();

}

template <typename T, typename T0>

RBTNode<T, T0> \*RBTree<T, T0>::search(RBTNode<T, T0> \*x, T key)

{

    if (x == NULL || x->key == key)

        return x;

    if (key < x->key)

        return search(x->left, key);

    else

        return search(x->right, key);

}

template <typename T, typename T0>

RBTNode<T, T0> \*RBTree<T, T0>::search(T key)

{

    return search(root, key);

}

template <typename T, typename T0>

void RBTree<T, T0>::destroy(RBTNode<T, T0> \*&x)

{

    if (x == NULL)

        return;

    if (x->left != NULL)

        destroy(x->left);

    if (x->right != NULL)

        destroy(x->right);

    delete x;

    x = NULL;

}

template <typename T, typename T0>

void RBTree<T, T0>::destroy()

{

    destroy(root);

}

template <typename T, typename T0>

void RBTree<T, T0>::left\_rotate(RBTNode<T, T0> \*&root, RBTNode<T, T0> \*x)

{

    RBTNode<T, T0> \*y = x->right;

    if (y->left == NULL)

    {

        x->right = NULL;

    }

    else

    {

        x->right = y->left;

        y->left->parent = x;

    }

    y->parent = x->parent;

    if (x->parent == NULL)

    {

        root = y;

    }

    else

    {

        if (x->parent->left == x)

            x->parent->left = y;

        else

            x->parent->right = y;

    }

    y->left = x;

    x->parent = y;

}

template <typename T, typename T0>

void RBTree<T, T0>::right\_rotate(RBTNode<T, T0> \*&root, RBTNode<T, T0> \*y)

{

    RBTNode<T, T0> \*x = y->left;

    if (x->right == NULL)

    {

        y->left = NULL;

    }

    else

    {

        y->left = x->right;

        x->right->parent = y;

    }

    x->parent = y->parent;

    if (y->parent == NULL)

    {

        root = x;

    }

    else

    {

        if (y == y->parent->right)

            y->parent->right = x;

        else

            y->parent->left = x;

    }

    x->right = y;

    y->parent = x;

}

template <typename T, typename T0>

void RBTree<T, T0>::insert(T key, T0 data)

{

    RBTNode<T, T0> \*z = NULL;

    // 创建节点

    z = new RBTNode<T, T0>(key, BLACK, NULL, NULL, NULL, data);

    insert(root, z);

}

template <class T, class T0>

void RBTree<T, T0>::insert(RBTNode<T, T0> \*&root, RBTNode<T, T0> \*node)

{

    RBTNode<T, T0> \*y = NULL;

    RBTNode<T, T0> \*x = root;

    //  将红黑树当作一颗二叉查找树，将节点添加到二叉查找树中。

    while (x != NULL)

    {

        y = x;

        if (node->key < x->key)

            x = x->left;

        else

            x = x->right;

    }

    node->parent = y;

    if (y != NULL)

    {

        if (node->key < y->key)

            y->left = node;

        else

            y->right = node;

    }

    else

        root = node;

    // 设置节点的颜色为红色

    node->color = RED;

    // 将它重新修正为一颗二叉查找树

    insert\_fix(root, node);

}

template <typename T, typename T0>

void RBTree<T, T0>::insert\_fix(RBTNode<T, T0> \*&root, RBTNode<T, T0> \*node)

{

    // 将根节点设为黑色

    root->color = BLACK;

    RBTNode<T, T0> \*parent, \*gparent;

    // 若父节点存在，并且父节点的颜色是红色

    while ((parent = node->parent) != NULL && parent->color == RED)

    {

        gparent = parent->parent;

        // 若父节点是祖父节点的左孩子

        if (parent == gparent->left)

        {

            // 若叔叔节点是红色

            {

                RBTNode<T, T0> \*uncle = gparent->right;

                if (uncle != NULL && uncle->color == RED)

                {

                    uncle->color = BLACK;

                    parent->color = BLACK;

                    gparent->color = RED;

                    node = gparent;

                    continue;

                }

            }

            // 若叔叔是黑色，且当前节点是右孩子

            if (parent->right == node)

            {

                RBTNode<T, T0> \*tmp;

                left\_rotate(root, parent);

                tmp = parent;

                parent = node;

                node = tmp;

            }

            // 若叔叔是黑色，且当前节点是左孩子

            parent->color = BLACK;

            gparent->color = RED;

            right\_rotate(root, gparent);

        }

        else // 若父节点是祖父节点的右孩子

        {

            // 若叔叔节点是红色

            {

                RBTNode<T, T0> \*uncle = gparent->left;

                if (uncle != NULL && uncle->color == RED)

                {

                    uncle->color = BLACK;

                    parent->color = BLACK;

                    gparent->color = RED;

                    node = gparent;

                    continue;

                }

            }

            // 若叔叔是黑色，且当前节点是左孩子

            if (parent->left == node)

            {

                RBTNode<T, T0> \*tmp;

                right\_rotate(root, parent);

                tmp = parent;

                parent = node;

                node = tmp;

            }

            // 若叔叔是黑色，且当前节点是右孩子

            parent->color = BLACK;

            gparent->color = RED;

            left\_rotate(root, gparent);

        }

    }

}

template <typename T, typename T0>

void RBTree<T, T0>::remove(T key)

{

    RBTNode<T, T0> \*node;

    // 查找key对应的节点node，找到的话就删除该节点

    if ((node = search(root, key)) != NULL)

        remove(root, node);

}

template <typename T, typename T0>

void RBTree<T, T0>::remove(RBTNode<T, T0> \*&root, RBTNode<T, T0> \*node)

{

    RBTNode<T, T0> \*child, \*parent;

    RBTColor color;

    // 被删除节点的左右孩子都不为空的情况

    if ((node->left != NULL) && (node->right != NULL))

    {

        // 寻找被删节点的取代节点

        // 用它来取代被删节点的位置，然后再将被删节点去掉。

        RBTNode<T, T0> \*replace = node;

        // 获取右子树中的最小节点

        replace = replace->right;

        while (replace->left != NULL)

            replace = replace->left;

        // node不是根节点

        if (node->parent != NULL)

        {

            if ((node->parent)->left == node)

                (node->parent)->left = replace;

            else

                (node->parent)->right = replace;

        }

        else

            // node节点是根节点

            root = replace;

        // child是"取代节点"的右孩子，也是需要"调整的节点"。

        child = replace->right;

        parent = replace->parent;

        color = replace->color;

        // "被删除节点"是"它的后继节点的父节点"

        if (parent == node)

        {

            parent = replace;

        }

        else

        {

            // child不为空

            if (child != NULL)

                child->parent = parent;

            parent->left = child;

            replace->right = node->right;

            (node->right)->parent = replace;

        }

        replace->parent = node->parent;

        replace->color = node->color;

        replace->left = node->left;

        node->left->parent = replace;

        if (color == BLACK)

            remove\_fix(root, child, parent);

        delete node;

        return;

    }

    if (node->left != NULL)

        child = node->left;

    else

        child = node->right;

    parent = node->parent;

    // 保存"取代节点"的颜色

    color = node->color;

    if (child != NULL)

        child->parent = parent;

    // "node节点"不是根节点

    if (parent != NULL)

    {

        if (parent->left == node)

            parent->left = child;

        else

            parent->right = child;

    }

    else

        root = child;

    if (color == BLACK)

        remove\_fix(root, child, parent);

    delete node;

}

template <typename T, typename T0>

void RBTree<T, T0>::remove\_fix(RBTNode<T, T0> \*&root, RBTNode<T, T0> \*node, RBTNode<T, T0> \*parent)

{

    RBTNode<T, T0> \*other;

    while ((!node || node->color == BLACK) && node != root)

    {

        // 为左子结点

        if (parent->left == node)

        {

            other = parent->right;

            if (other->color == RED)

            {

                // 情况2：兄弟是红色的

                other->color = BLACK;

                parent->color = RED;

                left\_rotate(root, parent);

                other = parent->right;

            }

            if (((!other->left || (other->left)->color == BLACK)) &&

                (!other->right || (other->right)->color == BLACK))

            {

                // 情况3：父亲、兄弟和兄弟的两个儿子都是黑色的

                if (parent->color == BLACK)

                {

                    other->color = RED;

                    node = parent;

                    parent = node->parent;

                    continue;

                }

                // 情况4：兄弟和兄弟的两个儿子是黑色的，父亲是红色的

                else

                {

                    other->color = RED;

                    parent->color = BLACK;

                    break;

                }

            }

            else

            {

                if (!other->right || (other->right)->color == BLACK)

                {

                    // 情况5：兄弟是黑色的，兄弟的左孩子是红色，右孩子为黑色。

                    (other->left)->color = BLACK;

                    other->color = RED;

                    right\_rotate(root, other);

                    other = parent->right;

                }

                // 情况6：兄弟是黑色的，兄弟的右孩子是红色的，左孩子任意颜色。

                other->color = parent->color;

                parent->color = BLACK;

                (other->right)->color = BLACK;

                left\_rotate(root, parent);

                node = root;

                break;

            }

        }

        else

        {

            other = parent->left;

            if (other->color == RED)

            {

                // 情况2：兄弟是红色的

                other->color = BLACK;

                parent->color = RED;

                right\_rotate(root, parent);

                other = parent->left;

            }

            if ((!other->left || (other->left)->color == BLACK) &&

                (!other->right || (other->right)->color == BLACK))

            {

                // 情况3：父亲、兄弟和兄弟的两个儿子都是黑色的

                if (parent->color == BLACK)

                {

                    other->color = RED;

                    node = parent;

                    parent = node->parent;

                    continue;

                }

                // 情况4：兄弟和兄弟的两个儿子是黑色的，父亲是红色的

                else

                {

                    other->color = RED;

                    parent->color = BLACK;

                    break;

                }

            }

            else

            {

                if (!other->left || (other->left)->color == BLACK)

                {

                    // 情况5：兄弟是黑色的，兄弟的右孩子是红色，左孩子为黑色。

                    (other->right)->color = BLACK;

                    other->color = RED;

                    left\_rotate(root, other);

                    other = parent->left;

                }

                // 情况6：兄弟是黑色的，兄弟的左孩子是红色的，右孩子任意颜色。

                other->color = parent->color;

                parent->color = BLACK;

                (other->left)->color = BLACK;

                right\_rotate(root, parent);

                node = root;

                break;

            }

        }

    }

    if (node != NULL)

        node->color = BLACK;

}

#endif

8、admin\_m.cpp

#include <bits/stdc++.h>

#include "../include/admin.h"

#include "../include/admin\_m.h"

#include "../include/book.h"

#include "../include/book\_m.h"

#include "../include/rbtree.h"

#include "../include/menu.h"

#include "../include/config.h"

using namespace std;

admin\_manager::admin\_manager()

{

    init();

}

admin\_manager::~admin\_manager()

{

    save();

    admins.destroy();

}

void admin\_manager::init()

{

    ifstream file(admin\_path);

    if (!file.is\_open())

    {

        cerr << "unable to open admin\_data.csv!" << endl;

        return;

    }

    string line;

    getline(file, line);

    while (getline(file, line))

    {

        stringstream ss(line);

        string account, password;

        getline(ss, account, ',');

        getline(ss, password);

        admin admin\_new = admin(account, password);

        admins.insert(account, admin\_new);

    }

    file.close();

}

bool admin\_manager::regist()

{

    cout << "please enter the secret key:" << endl;

    string secret\_key;

    cin >> secret\_key;

    if (secret\_key != key)

    {

        cout << "wrong secret key!" << endl;

        return false;

    }

    cout << "please enter the account:" << endl;

    string account;

    cin >> account;

    if (admins.search(account))

    {

        cout << "the account already exists!" << endl;

        return false;

    }

    cout << "please enter the password:" << endl;

    string password, password\_check;

    cin >> password;

    cout << "please enter the password again:" << endl;

    cin >> password\_check;

    if (password != password\_check)

    {

        cout << "the two passwords do not match!" << endl;

        return false;

    }

    admin admin\_new = admin(account, password);

    admins.insert(account, admin\_new);

    cout << "register successfully!" << endl;

    save();

    return true;

}

bool admin\_manager::login()

{

    cout << "please enter the account:" << endl;

    string account;

    cin >> account;

    RBTNode<string, admin> \*node = admins.search(account);

    if (node == NULL)

    {

        cout << "the account does not exist!" << endl;

        return false;

    }

    else

    {

        cout << "please enter the password:" << endl;

        string password;

        cin >> password;

        if ((node->data).getPassword() == password)

        {

            cout << "login successfully!" << endl;

            return true;

        }

        else

        {

            cout << "wrong password!" << endl;

            return false;

        }

    }

}

// 保存树节点到文件

void admin\_manager::save\_node(RBTNode<string, admin> \*node, ofstream &file)

{

    if (node == NULL)

    {

        return;

    }

    // 递归保存左子树

    this->save\_node(node->left, file);

    // 保存当前节点的账号和密码

    file << (node->data).getAccount() << "," << (node->data).getPassword() << endl;

    // 递归保存右子树

    this->save\_node(node->right, file);

}

// 保存所有管理员数据到文件

void admin\_manager::save()

{

    ofstream file(admin\_path, ios::trunc); // 使用trunc模式，覆盖旧文件

    if (!file.is\_open())

    {

        cerr << "unable to open admin\_data.csv!" << endl;

        return;

    }

    file << "account,password" << endl;

    // 保存红黑树的根节点数据

    this->save\_node(admins.get\_root(), file);

    file.close(); // 关闭文件

}

admin\_manager admin\_m;

9、admin.cpp

#include <bits/stdc++.h>

#include "../include/admin.h"

#include "../include/admin\_m.h"

#include "../include/book.h"

#include "../include/book\_m.h"

#include "../include/rbtree.h"

#include "../include/menu.h"

#include "../include/config.h"

using namespace std;

admin::admin() {}

admin::~admin() {}

admin::admin(string account, string password)

{

    this->account = account;

    this->password = password;

}

string admin::getAccount()

{

    return account;

}

string admin::getPassword()

{

    return password;

}

void admin::setAccount(string account)

{

    this->account = account;

}

void admin::setPassword(string password)

{

    this->password = password;

}

10、book\_m.cpp

#include <bits/stdc++.h>

#include "../include/admin.h"

#include "../include/admin\_m.h"

#include "../include/book.h"

#include "../include/book\_m.h"

#include "../include/rbtree.h"

#include "../include/menu.h"

#include "../include/config.h"

using namespace std;

book\_manager::book\_manager()

{

    init();

}

book\_manager::~book\_manager()

{

    save();

    books.destroy();

}

void book\_manager::init()

{

    ifstream file(book\_path);

    if (!file.is\_open())

    {

        cerr << "unable to open book\_data.csv!" << endl;

        return;

    }

    string line;

    getline(file, line);

    while (getline(file, line))

    {

        stringstream ss(line);

        string id, name, author, publisher, year, ISBN, borrower, flag;

        int borrow\_count;

        bool is\_borrowed;

        getline(ss, id, ',');

        getline(ss, name, ',');

        getline(ss, author, ',');

        getline(ss, publisher, ',');

        getline(ss, year, ',');

        getline(ss, flag, ',');

        getline(ss, borrower, ',');

        getline(ss, ISBN, ',');

        if (flag == "0")

            is\_borrowed = false;

        else

            is\_borrowed = true;

        ss >> borrow\_count;

        if (books.search(name))

        {

            cout << "the book already exists!" << endl;

            return;

        }

        book book\_new = book(id, name, author, publisher, year, ISBN, borrower, is\_borrowed, borrow\_count);

        books.insert(name, book\_new);

    }

}

void book\_manager::insert()

{

    string id, name, author, publisher, x, year, ISBN, borrower;

    int borrow\_count;

    bool is\_borrowed;

    cout << "please enter the book's name:" << endl;

    cin >> name;

    if (books.search(name))

    {

        cout << "the book already exists!" << endl;

        return;

    }

    cout << "please enter the book's id:" << endl;

    cin >> id;

    cout << "please enter the book's author:" << endl;

    cin >> author;

    cout << "please enter the book's publisher:" << endl;

    cin >> publisher;

    cout << "please enter the book's year:" << endl;

    cin >> year;

    cout << "please enter the book's ISBN:" << endl;

    cin >> ISBN;

    cout << "please enter the book's borrow\_count:" << endl;

    cin >> borrow\_count;

    cout << "please enter 0 if the book is not borrowed,enter 1 if the book if borrowed" << endl;

    cin >> x;

    if (x == "0")

    {

        is\_borrowed = false;

        borrower = "null";

    }

    else

    {

        is\_borrowed = true;

        cout << "please enter the book's borrower:" << endl;

        cin >> borrower;

    }

    book book\_new = book(id, name, author, publisher, year, ISBN, borrower, is\_borrowed, borrow\_count);

    books.insert(name, book\_new);

    cout << "the book has been added!" << endl;

    save();

}

bool book\_manager::empty()

{

    return books.empty();

}

// 保存树节点到文件

void book\_manager::save\_node(RBTNode<string, book> \*node, ofstream &file)

{

    if (node == NULL)

    {

        return;

    }

    this->save\_node(node->left, file);

    // 保存当前节点的书籍信息

    book b = node->data;

    file << b.getId() << "," << b.getName() << "," << b.getAuthor() << ","

         << b.getPublisher() << "," << b.getYear() << "," << (b.getIsBorrowed() ? "1" : "0") << ","

         << b.getBorrower() << "," << b.getISBN() << "," << b.getBorrowCount() << endl;

    this->save\_node(node->right, file);

}

// 保存所有书籍信息到文件

void book\_manager::save()

{

    ofstream file(book\_path, ios::trunc); // 使用trunc模式，覆盖旧文件

    if (!file.is\_open())

    {

        cerr << "unable to open book\_data.csv!" << endl;

        return;

    }

    file << "id,name,author,publisher,year,isborrowed,borrower,ISBN,borrowcount" << endl;

    this->save\_node(books.get\_root(), file);

    file.close(); // 关闭文件

}

void book\_manager::find\_all\_lend1(RBTNode<string, book> \*node)

{

    if (node == NULL)

    {

        return;

    }

    this->find\_all\_lend1(node->left); // 递归遍历左子树

    // 获取当前节点的书籍信息

    book b = node->data;

    if (b.getIsBorrowed())

        cout << "<<" << b.getName() << ">>" << endl;

    this->find\_all\_lend1(node->right); // 递归遍历右子树

}

void book\_manager::find\_all\_lend()

{

    cout << "these books were borrowed:" << endl;

    // 调用递归函数，遍历红黑树

    this->find\_all\_lend1(books.get\_root());

}

void book\_manager::lend()

{

    cout << "please enter the name of the book:" << endl;

    string key;

    cin >> key;

    // 查找书籍

    RBTNode<string, book> \*node = books.search(key);

    if (node == NULL)

    {

        cout << "the book does not exist!" << endl;

        return;

    }

    book b = node->data;

    // 检查是否已经借出

    if (b.getIsBorrowed())

    {

        cout << "the book has been borrowed!" << endl;

        return;

    }

    // 让用户输入借阅人的信息

    cout << "please enter the name of the borrower" << endl;

    string borrower;

    cin >> borrower;

    // 更新书籍的借阅状态

    b.setIsBorrowed(true);

    b.setBorrower(borrower); // 设置借阅人

    // 记录借阅次数

    b.setBorrowCount(b.getBorrowCount() + 1);

    // 更新树中的书籍信息

    node->data = b;

    cout << "the book <<" << b.getName() << ">> has been borrowed by " << borrower << "!" << endl;

    save();

}

void book\_manager::return\_book()

{

    cout << "please enter the name of the book:" << endl;

    string key;

    cin >> key;

    // 查找书籍

    RBTNode<string, book> \*node = books.search(key);

    if (node == NULL)

    {

        cout << "the book does not exist!" << endl;

        return;

    }

    book b = node->data;

    // 检查是否已借出

    if (!b.getIsBorrowed())

    {

        cout << "the book has not been borrowed!" << endl;

        return;

    }

    // 更新书籍的借阅状态为未借出

    b.setIsBorrowed(false);

    b.setBorrower("null"); // 清空借阅人

    // 更新树中的书籍信息

    node->data = b;

    cout << "the book <<" << b.getName() << ">> has been returned!" << endl;

    save();

}

void book\_manager::search()

{

    cout << "please enter the name of the book:" << endl;

    string name;

    cin >> name;

    // 查找书籍

    RBTNode<string, book> \*node = books.search(name);

    if (node == NULL)

    {

        cout << "the book does not exist!" << endl;

        return;

    }

    // 找到书籍后，显示书籍信息

    book b = node->data;

    cout << b;

}

void book\_manager::update()

{

    cout << "please enter the name of the book:" << endl;

    string name;

    cin >> name;

    // 查找书籍

    RBTNode<string, book> \*node = books.search(name);

    if (node == NULL)

    {

        cout << "the book does not exist!" << endl;

        return;

    }

    // 找到书籍后，提示用户进行修改

    book b = node->data;

    cout << "find the book <<" << b.getName() << ">>,please update" << endl;

    string id, author, publisher, year, ISBN, borrower, is\_borrowed;

    int borrow\_count;

    bool flag;

    cout << "please enter the ID: " << endl;

    cin >> id;

    cout << "please enter the name: " << endl;

    cin >> name;

    if (name != b.getName() && books.search(name) != NULL)

    {

        cout << "the name has been used!" << endl;

        return;

    }

    cout << "please enter the author: " << endl;

    cin >> author;

    cout << "please enter the publisher:  " << endl;

    cin >> publisher;

    cout << "please enter the year: " << endl;

    cin >> year;

    cout << "please enter 0 if the book is not borrowed,enter 1 if the book if borrowed:" << endl;

    cin >> is\_borrowed;

    if (is\_borrowed == "0")

    {

        flag = false;

        borrower = "null";

    }

    else

    {

        flag = true;

        cout << "please enter the borrower: " << endl;

        cin >> borrower;

    }

    cout << "please enter the ISBN: " << endl;

    cin >> ISBN;

    cout << "please enter the borrow\_count: " << endl;

    cin >> borrow\_count;

    // 更新书籍信息

    b.setId(id);

    b.setAuthor(author);

    b.setPublisher(publisher);

    b.setYear(year);

    b.setISBN(ISBN);

    b.setName(name);

    b.setIsBorrowed(flag);

    b.setBorrower(borrower);

    b.setBorrowCount(borrow\_count);

    // 更新树中的书籍

    node->data = b;

    cout << "update successfully!" << endl;

    save();

}

void book\_manager::remove()

{

    cout << "please enter the name of the book:" << endl;

    string name;

    cin >> name;

    // 查找书籍

    RBTNode<string, book> \*node = books.search(name);

    if (node == NULL)

    {

        cout << "the book does not exist!" << endl;

        return;

    }

    book b = node->data;

    // 如果书籍已经借出，提示不能删除

    if (b.getIsBorrowed())

    {

        cout << "the book has been borrowed,so it can not be deleted!" << endl;

        return;

    }

    // 删除书籍

    books.remove(name);

    cout << "the book <<" << name << ">> has been deleted successfully!" << endl;

    save();

}

void book\_manager::fuzzy\_search()

{

    cout << "please enter the name of the book:" << endl;

    string input;

    cin.ignore();        // 忽略之前输入的换行符

    getline(cin, input); // 获取用户输入的书名

    // 将用户输入的字符串转为小写，方便不区分大小写进行搜索

    transform(input.begin(), input.end(), input.begin(), ::tolower);

    bool found = false;

    // 遍历树中的所有书籍

    RBTNode<string, book> \*node = books.get\_root();

    stack<RBTNode<string, book> \*> s;

    while (!s.empty() || node != NULL)

    {

        if (node != NULL)

        {

            s.push(node);

            node = node->left;

        }

        else

        {

            node = s.top();

            s.pop();

            book b = node->data;

            string book\_name = b.getName();

            // 将书名转为小写，以进行不区分大小写的匹配

            transform(book\_name.begin(), book\_name.end(), book\_name.begin(), ::tolower);

            // 检查书名是否包含用户输入的字符串（模糊匹配）

            if (book\_name.find(input) != string::npos)

            {

                found = true;

                cout << b;

                cout << endl;

            }

            node = node->right;

        }

    }

    if (!found)

    {

        cout << "fail to find!" << endl;

    }

}

void book\_manager::max\_lend()

{

    // 存储所有书籍的名称和借阅次数

    vector<pair<string, int>> book\_lends;

    // 遍历红黑树中的每本书

    RBTNode<string, book> \*node = books.get\_root();

    stack<RBTNode<string, book> \*> s;

    while (!s.empty() || node != NULL)

    {

        if (node != NULL)

        {

            s.push(node);

            node = node->left;

        }

        else

        {

            node = s.top();

            s.pop();

            book b = node->data;

            string book\_name = b.getName();

            int borrow\_count = b.getBorrowCount();

            book\_lends.push\_back(make\_pair(book\_name, borrow\_count));

            node = node->right;

        }

    }

    // 按照借阅次数降序排序

    sort(book\_lends.begin(), book\_lends.end(),

         [](const pair<string, int> &a, const pair<string, int> &b)

         {

             return a.second > b.second; // 按照借阅次数降序排序

         });                             // 排序规则：借阅次数更多的排在前面

    // 输出借阅次数最多的十本书

    cout << "the ten most frequently checked out books:" << endl;

    int max\_display = min(10, (int)book\_lends.size());

    for (int i = 0; i < max\_display; i++)

    {

        string book\_name = book\_lends[i].first;

        int borrow\_count = book\_lends[i].second;

        cout << i + 1 << ". " << book\_name << " - number of borrowings: " << borrow\_count << endl;

        int bar\_length = borrow\_count / 2; // 每5次借阅占1个字符宽度，调整此值来控制条形图的比例

        for (int j = 0; j < bar\_length; j++)

        {

            cout << "#";

        }

        cout << endl;

    }

}

book\_manager book\_m;

1. book.cpp

#include <bits/stdc++.h>

#include "../include/admin.h"

#include "../include/admin\_m.h"

#include "../include/book.h"

#include "../include/book\_m.h"

#include "../include/rbtree.h"

#include "../include/menu.h"

#include "../include/config.h"

using namespace std;

book::book()

{

    id = "";

    name = "";

    author = "";

    publisher = "";

    year = "";

    borrower = "null";

    isBorrowed = false;

    ISBN = "";

    borrowCount = 0;

}

book::book(string id, string name, string author, string publisher, string year, string ISBN, string borrower, bool is\_borrowed, int borrowCount)

{

    this->id = id;

    this->name = name;

    this->author = author;

    this->publisher = publisher;

    this->year = year;

    this->ISBN = ISBN;

    this->isBorrowed = is\_borrowed;

    this->borrower = borrower;

    this->borrowCount = borrowCount;

}

book::~book() {}

int book::getBorrowCount()

{

    return borrowCount;

}

void book::setBorrowCount(int borrowCount)

{

    this->borrowCount = borrowCount;

}

string book::getId()

{

    return id;

}

string book::getAuthor()

{

    return author;

}

string book::getBorrower()

{

    return borrower;

}

string book::getYear()

{

    return year;

}

string book::getISBN()

{

    return ISBN;

}

bool book::getIsBorrowed()

{

    return isBorrowed;

}

string book::getName()

{

    return name;

}

string book::getPublisher()

{

    return publisher;

}

void book::setId(string id)

{

    this->id = id;

}

void book::setName(string name)

{

    this->name = name;

}

void book::setAuthor(string author)

{

    this->author = author;

}

void book::setPublisher(string publisher)

{

    this->publisher = publisher;

}

void book::setYear(string year)

{

    this->year = year;

}

void book::setBorrower(string borrower)

{

    this->borrower = borrower;

}

void book::setIsBorrowed(bool isBorrowed)

{

    this->isBorrowed = isBorrowed;

}

void book::setISBN(string ISBN)

{

    this->ISBN = ISBN;

}

ostream &operator<<(ostream &out, book &b)

{

    out << "the information of the book:" << endl;

    out << "the ID of the book:" << b.id << endl;

    out << "the name of the book:" << b.name << endl;

    out << "the author of the book:" << b.author << endl;

    out << "the publisher of the book:" << b.publisher << endl;

    out << "the year of the book:" << b.year << endl;

    out << "the ISBN of the book:" << b.ISBN << endl;

    if (b.isBorrowed)

    {

        out << "the borrowing status of the book:checked out" << endl;

        out << "the borrower of the book:" << b.borrower << endl;

    }

    else

        out << "the borrowing status of the book:in library" << endl;

    out << "the borrow count of the book:" << b.borrowCount << endl;

    return out;

}

1. config.cpp

#include <bits/stdc++.h>

#include "../include/admin.h"

#include "../include/admin\_m.h"

#include "../include/book.h"

#include "../include/book\_m.h"

#include "../include/rbtree.h"

#include "../include/menu.h"

#include "../include/config.h"

using namespace std;

string key="abc123";  // 管理员注册秘钥

string admin\_path="../data/admin\_data.csv";

string book\_path="../data/book\_data.csv";

1. main.cpp

#include <bits/stdc++.h>

#include "../include/admin.h"

#include "../include/admin\_m.h"

#include "../include/book.h"

#include "../include/book\_m.h"

#include "../include/rbtree.h"

#include "../include/menu.h"

#include "../include/config.h"

using namespace std;

int main()

{

    m.start\_menu();

}

1. menu.cpp

#include <bits/stdc++.h>

#include "../include/admin.h"

#include "../include/admin\_m.h"

#include "../include/book.h"

#include "../include/book\_m.h"

#include "../include/rbtree.h"

#include "../include/menu.h"

#include "../include/config.h"

using namespace std;

void menu::start\_menu()

{

    while (1)

    {

        cout << "----------------------------------------------------------\n";

        cout << "Welcome to the Library Management System!" << endl;

        char x;

        cout << "1.login" << endl;

        cout << "2.register" << endl;

        cout << "3.exit" << endl;

        cin >> x;

        bool flag;

        switch (x)

        {

        case '1':

            flag = admin\_m.login();

            if (flag)

            {

                manage\_book\_menu();

            }

            break;

        case '2':

            admin\_m.regist();

            break;

        case '3':

            return;

        default:

            cout << "invalid input" << endl;

        }

    }

}

void menu::manage\_book\_menu()

{

    while (true)

    {

        int x;

        cout << "----------------------------------------------------------\n";

        cout << "please enter a number to choice:\n";

        cout << "1.insert book" << endl;

        cout << "2.remove book" << endl;

        cout << "3.update book" << endl;

        cout << "4.search book" << endl;

        cout << "5.lend book" << endl;

        cout << "6.return book" << endl;

        cout << "7.find all the books that were borrowed" << endl;

        cout << "8.find the ten most frequently checked out books:" << endl;

        cout << "9.fuzzy search" << endl;

        cout << "10.exit" << endl;

        cin >> x;

        switch (x)

        {

        case 1:

            book\_m.insert();

            break;

        case 2:

            book\_m.remove();

            break;

        case 3:

            book\_m.update();

            break;

        case 4:

            book\_m.search();

            break;

        case 5:

            book\_m.lend();

            break;

        case 6:

            book\_m.return\_book();

            break;

        case 7:

            book\_m.find\_all\_lend();

            break;

        case 8:

            book\_m.max\_lend();

            break;

        case 9:

            book\_m.fuzzy\_search();

            break;

        case 10:

            return;

        default:

            cout << "invalid input" << endl;

        }

    }

}

menu m;