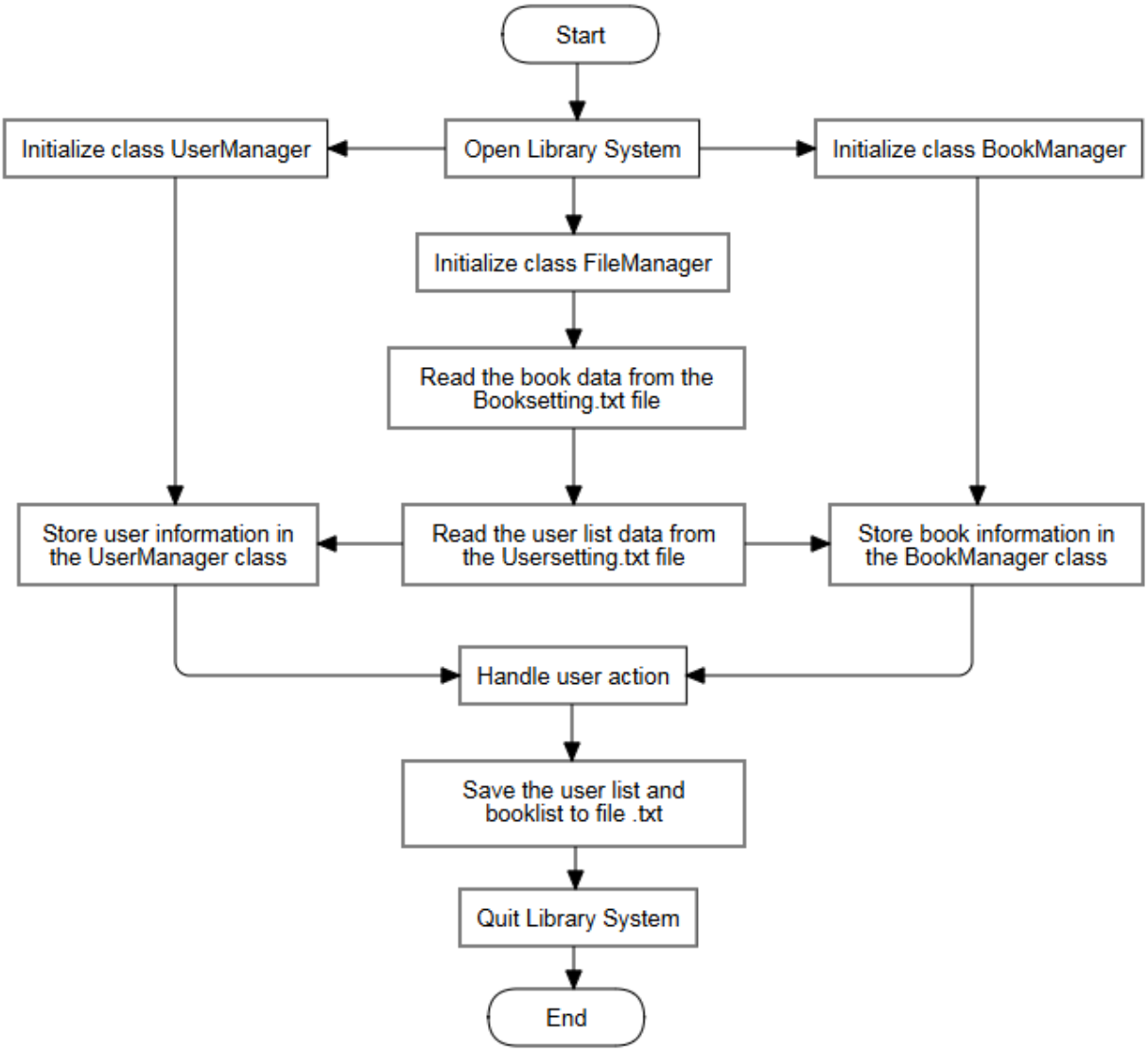
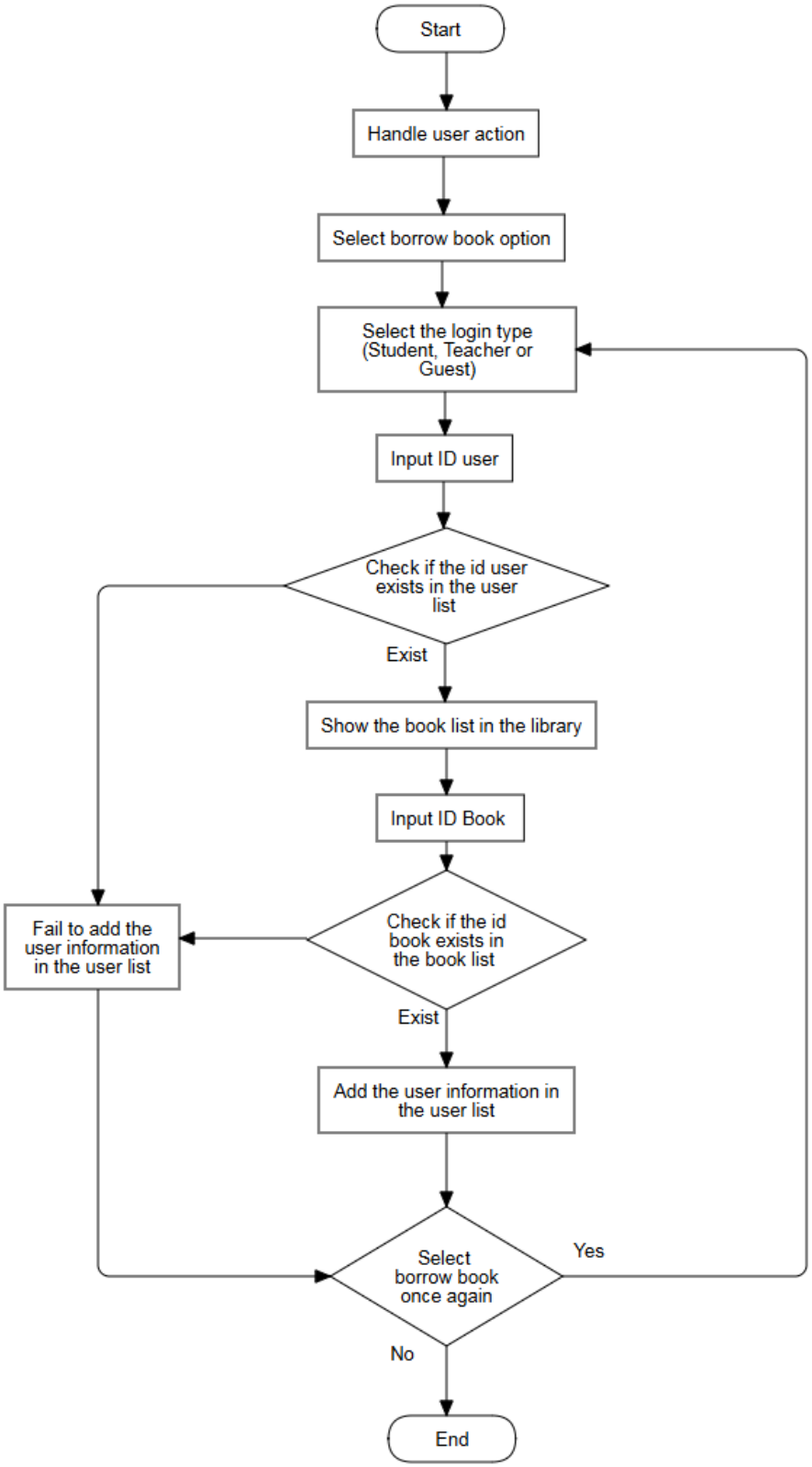
I. Luồng xử lí

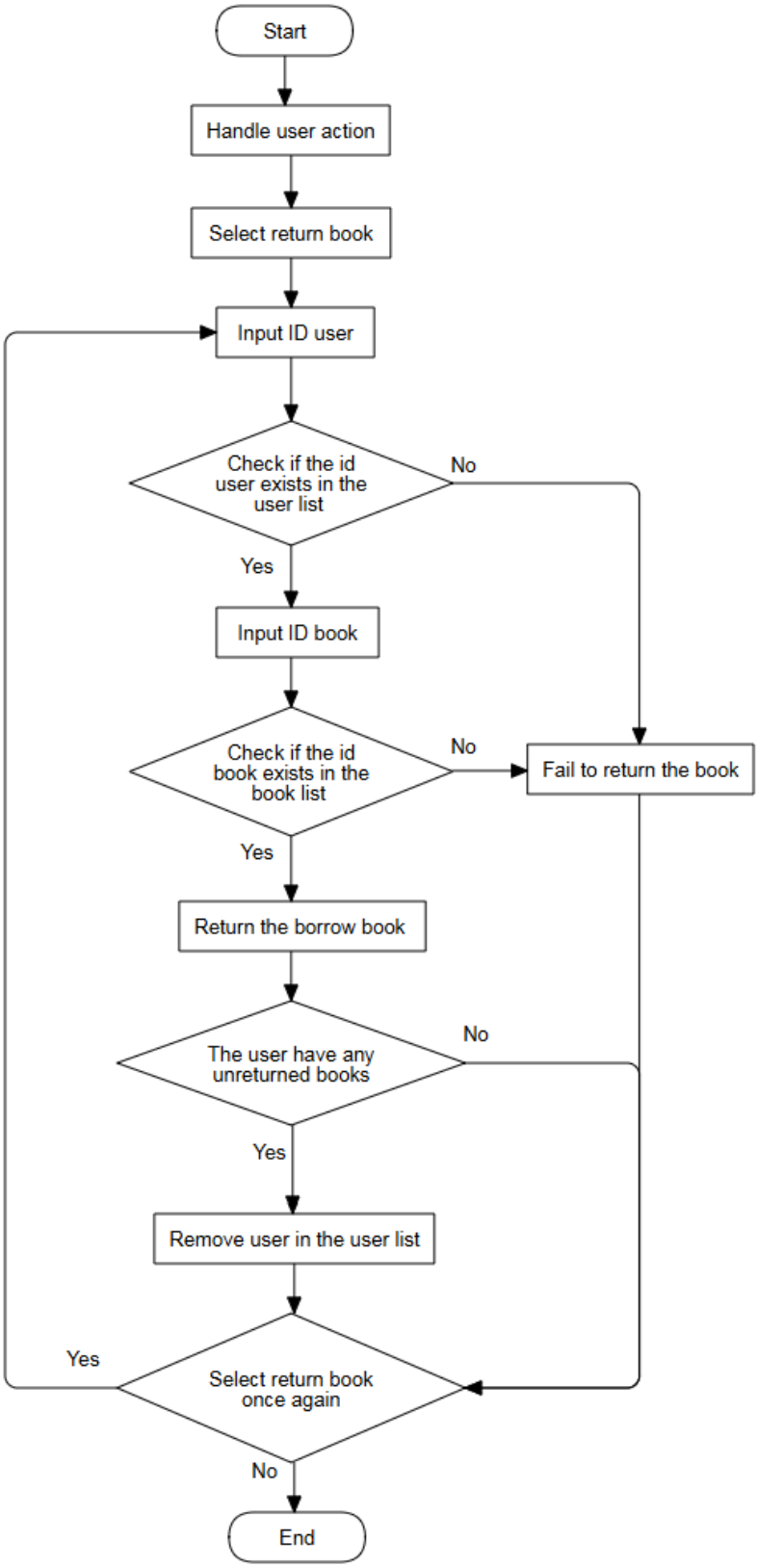
1. Khởi động hệ thống



2. Mượn sách


Start
Handle user action
Select borrow book option
Select the login type (Student, Teacher or Guest)
Input ID user
Check if the id user exists in the user list
Exist
Show the book list in the library
Input ID Book
Check if the id book exists in the book list
Fail to add the user information in the user list
Exist
Add the user information in the user list
Select borrow book once again
Yes
No
End

3. Trả sách


Start
Handle user action
Select return book
Input ID user
Check if the id user exists in the user list
No
Yes
Input ID book
Check if the id book exists in the book list
No
Fail to return the book
Yes
Return the borrow book
The user have any unreturned books
No
Yes
Remove user in the user list
Select return book once again
Yes
No
End

II. Class Diagram

III. Lựa chọn các giải pháp cho các tính năng

1. UserManager: Quản lí danh sách các người dùng mượn sách và thực hiện các chức năng quàn lí thông tin người dùng.

- Thêm thông tin người dùng mượn sách
- Xóa thông tin người dùng mượn sách
- Tìm thông tin người dùng mượn sách
- Cập nhật thông tin người dùng mượn sách khi họ mượn lần tới
- In ra danh sách thông tin người dùng mượn sách
- In ra danh sách người mượn sách quá 30 ngày

1.1 Cấu trúc dữ liệu để lưu trữ thông tin người dùng sẽ được sử dụng là: std::set

• Tính năng thêm thông tin user là tính năng quyết định để xem UserManager sử dụng cấu trúc dữ liệu nào để quản lí. Dựa trên luồng nhập thông tin User như sau:

Mảng được sắp xếp khi đọc file → nhập thông tin người dùng → lưu vào danh sách → mảng chưa được sắp xếp → nhập thông tin người dùng lần tới → kiểm tra id người dùng (tìm kiếm) → tạo người dùng mới nếu id chưa tồn tại hoặc không tạo thêm người dùng mới khi đã tồn tại id người dùng.

• Bài toán đặt ra: khi thêm User vào mảng, vị trí đó có thể chưa đúng thứ tự - mảng cũng chưa được sắp xếp. Do đó, hoặc là không sắp xếp mảng và tìm kiếm lần lượt các phần tử User trong mảng hoặc là phải liên tục sắp xếp mảng sau khi phần tử được thêm vào danh sách và tìm kiếm nhị phân để kiểm tra **id user** trong mảng. Ta phải chọn các thao tác sắp xếp và tìm kiếm sao cho tính năng input User có độ phức tạp là nhỏ nhất.

• Theo luồng nhập thông tin User, nếu ta sử dụng mảng hoặc vector để quản lí thì:

- TH1: Sử dụng tìm kiếm tuyến tính

+ Thêm phần tử user **O(1)** → Nhập thông tin user lần tới → Mảng không sắp xếp - tìm kiếm tuần tự duyệt qua danh sách và kiểm tra id user: độ phức tạp O(n) sẽ chậm khi n lớn. Cách này sẽ nhanh hơn so với **TH2** vì kể cả số lượng người dùng ít hoặc nhiều. Tại sao? Hãy xem cách tiếp theo.

- TH2: Sử dụng thuật toán sắp xếp và tìm kiếm nhị phân

+ Thêm phần tử user **O(1)** → Nhập thông tin user lần tới → Mảng sắp xếp nếu sử dụng thuật toán có độ phức tạp **O(nlogn)** - tìm kiếm nhị phân **O(logn)** để kiểm tra id user: độ phức tạp **O(nlogn.logn)**.

+ **TH2** tốn thêm thời gian vì phải sắp xếp trước khi tìm kiếm nhị phân. Nếu ta cứ input user và sắp xếp lại mảng, kể cả khi sử dụng thuật toán **insertion sort** – trường hợp tốt nhất của thuật toán sẽ là **O(n) - v**ì mảng đã được sắp xếp trước khi chèn vào một phần tử. Thì độ phức tạp sẽ là **O(nlogn) –** vẫn không bằng **TH1. TH2** có thể nhanh hơn **TH1** nếu mảng chỉ sắp xếp 1 lần và có nhiều lượt tìm kiếm (tỉ lệ theo số lượng mảng). Nhưng **TH2** luôn phải sắp xếp sau khi nhập thông tin user để kiểm tra id bằng tìm kiếm nhị phân. Vì thế, cách này lại chậm hơn **TH1**.

- TH3: Sử dụng cấu trúc dữ liệu **std::set** hoặc xây dựng cây nhị phân tự cân bằng

+ **std::set** được xây dựng theo cấu trúc cây nhị phân tự cân bằng nên các thao tác chèn, xóa, tìm kiếm sẽ luôn là **O(logn).**

+ Ở cách cuối cùng, khi nhập thông tin user và thêm vào danh sách, ta sẽ tốn **O(logn)** để chèn vào danh sách nhưng các phần tử đã được lưu sẵn theo thứ tự tăng dần. Nên khi nhập thông tin user khác và kiểm tra id vào lần tới, ta không cần phải sắp xếp và chỉ sẽ tốn **O(logn)** để tìm kiếm id có trong danh sách. Do đó, chi phí để chèn phần tử vào set là **O(logn)** (danh sách đã được sắp xếp) và chi phí tìm kiếm sẽ là **O(logn).** Tổng sẽ là: **O(2.logn) ~ O(logn). TH3** sẽ có chi phí ít hơn rất nhiều so với **TH1** và **TH2** bất kể số lượng n ít hay nhiều.

Vì vậy, **TH3** là cách hiệu quả nhất.

2. BookManager: Quản lí danh sách các sách có trong thư viện và thực hiện các chức năng như:

+ Hiển thị sách

+ Tìm kiếm để người dùng mượn sách

+ Tìm kiếm theo tên sách, tên tác giả, thể loại sách để hiển thị sách.

+ In số lượng sách mượn nhiều nhất

2.1 Cấu trúc dữ liệu sử dụng để quản lí các sách trong thư viện

• **Bài toán:** Danh sách sẽ được lưu thứ tự tăng dần theo ID Book và không thay đổi thứ tự sắp xếp trong runtime. Sử dụng cấu trúc dữ liệu để lưu danh sách sẽ hợp lí khi tìm kiếm ID book để người dùng mượn sách?

- Nếu tìm kiếm để người dùng mượn sách: vì danh sách đã được sắp xếp và không thêm các sách vào thư viện (mảng vẫn giữ thứ tự sắp xếp) nên:

+ Sử dụng mảng hoặc **vector** và tìm kiếm nhị phân sẽ tốn **O(logn)**

+ Sử dụng std::list: tìm kiếm sẽ tốn **O(n)**

+ Sử dụng cấu trúc dữ liệu **std::set** và tìm kiếm sẽ tốn **O(logn)** khi lấy cuốn sách đó.

Vậy ta có thể sử dụng mảng hoặc vector để quản lí sách và sử dụng tìm kiếm nhị phân để tìm kiếm sách (cho user mượn sách).

2.2. Tìm kiếm theo tên sách, tên tác giả hay thể loại sách để hiển thị:

+ Mảng được sắp xếp theo thứ tự của ID nên nếu sử dụng tìm kiếm nhị phân để tìm kiếm theo yêu cầu, ta phải sort mảng. Cách này có thể nhanh hơn nếu các giá trị yêu cầu của sách là không trùng nhau. Nhưng thông tin tên tác giả hoặc thể loại sách có thể trùng nhau nên việc sort mảng và tìm kiếm nhị phân sẽ mất nhiều lần nếu cứ muốn tìm kiếm theo cách này, độ phức tạp sẽ là **O(nlogn.logn)**

+ Vì trong việc tìm kiếm, các giá trị có thể trùng nhau. Ta có thể duyệt lần lượt và kiểm tra các phần tử trong mảng theo yêu cầu, độ phức tạp sẽ là **O(n)** cho việc tìm kiếm này.

Vậy ta có thể sử dụng tìm kiếm tuyến tính để tìm kiếm theo tên sách, tên tác giả hay thể loại sách.

2.3 In các sách được mượn nhiều nhất

- Cách 1: Ta có thể tìm kiếm số lượng sách được mượn nhiều nhất trong mảng với **O(logn)** trước. Sau đó duyệt qua mảng **- O(n)** để kiểm tra số lượng các sách và in ra các sách được mượn nhiều nhất. Độ phức tạp sẽ là **O(nlogn)**.

- Cách 2: Cập nhật lượng sách lớn nhất trước khi duyệt

+ Khi người dùng mượn sách, thư viện sẽ cập nhật số lượng sách mượn lớn nhất của các đầu sách.

+ Số lượng sách lớn nhất sẽ luôn được cập nhật và ta không cần thiết phải tìm kiếm số lượng sách trong thư viện trước khi duyệt qua mảng.

+ Thời gian truy xuất sẽ là **O(n).**

- Cách 3: Sử dụng mảng phụ

+ Vẫn cập nhật lượng sách lớn nhất trước khi duyệt. Việc này sẽ thực hiện khi có người dùng mượn sách, ta sẽ so sánh số lượng mới của sách đó với số lượng lớn nhất hiện tại trong mảng. Lúc này, mảng luôn lưu trữ các vị trí index của các sách mượn nhiều nhất, nhưng số lượng mượn của các sách mượn trong mảng là bằng nhau. Vậy ta chỉ lấy chỉ số index đầu tiên trong mảng để so sánh với số lượng sách còn lại đó mà người dùng đã mượn trong thư viện mà thôi:

- Nếu lớn hơn, thay thế toàn bộ mảng phụ và thêm index hiện tại vào (vì đây là sách có số lượng lớn nhất mới).
- Nếu bằng, thêm index vào mảng phụ.
- Nếu nhỏ hơn, không làm gì với mảng phụ.

+ Sử dụng một mảng phụ để lưu các chỉ số (index) của những đầu sách có số lượng lớn nhất trong thư viện.

+ Khi cần truy xuất các đầu sách đó, ta không phải duyệt toàn bộ danh sách (ListBook) nữa.

+ Thay vào đó ta chỉ cần duyệt mảng phụ, vốn chỉ chứa k phần tử – tức là số lượng sách có số lượng lớn nhất. Thời gian truy xuất: **O(k).** Chi phí không gian: **O(n).**

Vậy cách 3 sẽ là tốt nhất nếu cần truy xuất nhiều lần và chương trình có bộ nhớ đủ nhiều.

2.4 In thông tin người dùng mượn những sách đã quá 30 ngày

Ta có một mảng vector để lưu các giao dịch của các user đã mượn sách. Khác với cấu trúc set chỉ để lưu trữ và sử dụng cho sắp xếp và tìm kiếm thì mảng vector sẽ dùng cho việc lưu trữ các thông tin giao dịch của user theo thứ tự của ngày mượn sách.

- Nhập ngày hệ thống trong thư viện

- Duyệt và kiểm tra các giao dịch mà người dùng mượn sách, lấy ngày mượn của sách tương ứng và tính toán số ngày đã mượn

- In ra màn hình

3. User: Lớp User quản lí danh sách mà user mượn sách. Nó sẽ có các thao tác để tìm kiếm sách đã mượn, trả về danh sách các sách mượn và hiển thị danh sách đó.

- Trong thông tin User sẽ được lưu một mảng vector<BorrowTransaction>. BorrowTransaction là lớp lưu thông tin về lượt mượn sách, ngày mượn sách khi user thao tác chức năng mượn sách trong thư viện. Điều này tuân thủ quy tắc S trong SOLID – một lớp chỉ nên thực hiện một chức năng nhất định.

• Sử dụng tìm kiếm tuyến tính để kiểm tra sách đã mượn

- Khi trả sách, ta sử dụng tìm kiếm tuyến tính để tìm kiếm thông tin sách mà người dùng trả. Thư viện sẽ kiểm tra thông tin của user trước, sau đó là thông tin id của sách của người dùng đó đã mượn.

- Việc tìm kiếm tuần tự trong mảng vector có thể phù hợp sử dụng với số lượng sách mượn là nhỏ. Vì người dùng sẽ mượn sách không quá nhiều – tầm 10 sách trở xuống. Do đó, thao tác tìm kiếm trên mảng sách mượn của người dùng, ta có thể sử dụng tìm kiếm tuyến tính.