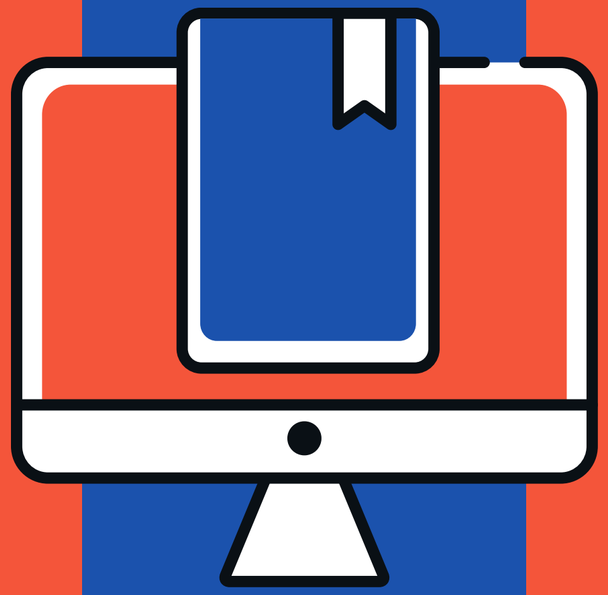
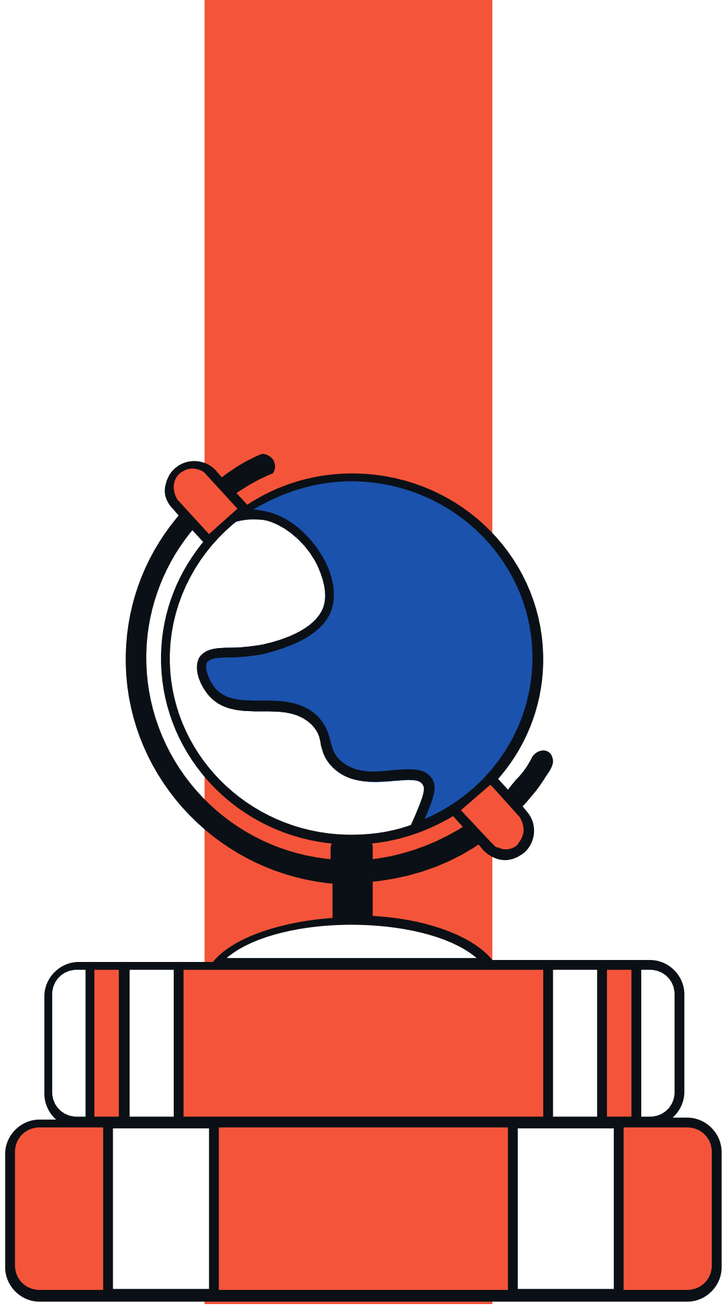


Library Management System

By: Shantanu Mukund Satpute (121CS0234

Table Of Content

https://github.com/shantanusatpute/Project



About Myself

I am Shantanu Satpute, a student currently in the pre-final year of my Bachelor's degree in Computer Science and Engineering at NIT Rourkela. Coming from Pune, Maharashtra, I have a deep interest in the intersection of technology and exploration. My expertise lies in areas such as Data Structures, Algorithms, and Database Management Systems. I strive to constantly enhance my skills in these domains.

Project Description

Description

The Library Management System is a software application designed to simplify the management of a library's operations and resources. It provides a digital platform for librarians to efficiently handle tasks such as adding new students, managing student records, tracking issued and returned books, and performing searches for student information.

Key Features

- **Student Management:** The system allows librarians to add new students to the library database. Each student record typically includes details such as name, ID number, and stream. The system provides functionalities for adding, removing, and searching for student records.

- **Book Management:** Librarians can maintain a catalog of available books in the library. Book details, such as title, author, and genre, can be stored in the system. Additionally, the system keeps track of the number of copies available and the number of copies issued to students.

- **Book Issuance and Return**: The system facilitates the issuance and return of books to students. Librarians can search for a student, check if the student is eligible to borrow a book, and update the book's status accordingly. Similarly, when a student returns a book, the system updates the availability status.

- **Display and Search:** The system allows librarians to display a list of students in the library along with their details. Librarians can search for specific students based on their name, ID, or other attributes. This functionality helps in quickly retrieving student information.

Classes Used

Class: Node

- **Purpose**: Represents a generic node in the library management system
- **Description**: The Node class serves as the base class for other node types in the library management system. It is designed to be a generic node that can hold different types of data. It includes a default constructor and destructor.

Class: Student

- **Purpose**: Represents a student in the library management system.
- **Description**: The Student class represents a student and their information in the library management system. It includes member variables to store the student's name, ID number, stream, and book information. The constructor initializes the student object with the provided name, ID number, and stream.

Class: library_management

- **Purpose**: Manages the library operations.
- **Description**: The library_management class is responsible for managing the library operations. It includes a root node pointer to maintain the student data structure.

Functions Used

- **void insert(Node* newNode)**
  - Purpose: Inserts a student into the library system.
  - Description: This function takes a pointer to a Node object representing a student and inserts it into the library. If the library is empty, the student is added as the root. However, if there is already a student in the library, it displays an error message indicating that only one student can be inserted in this version.
- **bool containsNode(const string& name)**
  - Purpose: Checks if a student with a given name exists in the library system.
  - Description: This function checks if a student with the specified name exists in the library. It utilizes the searchStudent function to perform the search operation. If a matching student is found, it returns true; otherwise, it returns false.
- **Node* searchStudent(Node* current, const string& name)**
  - Purpose: Searches for a student with a given name in the library system.
  - Description: This function recursively searches for a student with the specified name in the library system. It takes a pointer to the current node and the name to search for. If the current node is a Student type and its name matches the given name, it returns the student node. If the current node is not a Student type or the names don't match, it continues the search recursively. If no match is found, it returns nullptr.

Functions Used

- **void remove(const string& name)**
  - Purpose: Removes a student with a given name from the library system.
  - Description: This function removes a student with the specified name from the library. If the student is found (using the containsNode function), it deletes the root node and sets it to nullptr. If the student is not found, it displays an error message indicating that the student was not found in the library.
- **void display(Student* students, int& count)**
  - Purpose: Displays the information of all students in the library system.
  - Description: This function displays the information of all students in the library. It takes an array of Student objects and a reference to an integer count. The function retrieves the details of the student in the root node and stores them in the students array. It increments the count to keep track of the number of students. Note that in this version, only one student can be added, so the count will be either 0 (if no student is added) or 1 (if a student is added).
- **void issueBook()**
  - Purpose: Searches for a student with a given name in the library system.
  - Description: This function recursively searches for a student with the specified name in the library system. It takes a pointer to the current node and the name to search for. If the current node is a Student type and its name matches the given name, it returns the student node. If the current node is not a Student type or the names don't match, it continues the search recursively. If no match is found, it returns nullptr.

Functions Used

- **void returnBook()**
  - Purpose: Simulates returning a book by a student.
  - Description: This function simulates the process of a student returning a book to the library. It prompts the user to enter the name of the student. It then searches for a student with the given name using the searchStudent function. If a matching student is found, it displays a message indicating that the book has been returned by the student. If no match is found, it displays an error message indicating that the

Data Structures used

- **Data Structure: Binary Tree**
    - Purpose: Organises the student records in a hierarchical structure.
    - Description: The library management system utilizes a binary tree data structure to organize the student records. The root node represents the main student in the library, and additional students cannot be added in this version.

Why Binary Tree?

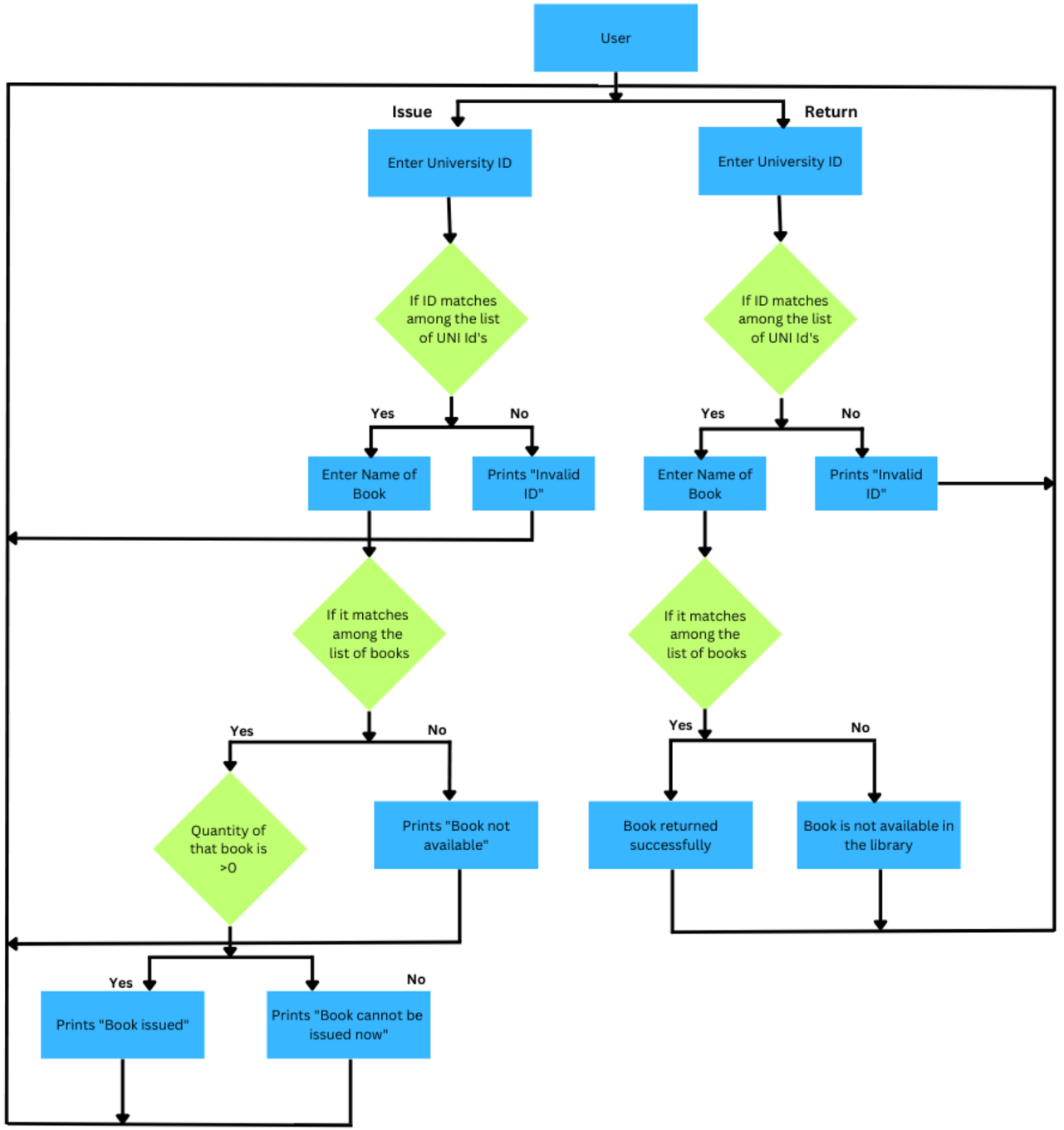
1. **Simplicity:** A binary tree is a relatively simple data structure to implement and understand. It provides a hierarchical structure for organizing data.
2. **Basic functionality:** The code focuses on basic functionalities such as adding a single student, searching for a student, displaying student information, and issuing/returning a book. For these simple operations, a binary tree can be used as a straightforward container.
3. **Future expansion:** Although the current code manages only one student, the choice of using a binary tree might suggest the intention to expand the system in the future to handle multiple students. The binary tree structure could be extended to support more advanced operations like searching, inserting, and deleting multiple student records efficiently.

Alternatives to Binary Tree

- **Linked Lists**
  - Each node of the linked list can represent a student record.
  - The linked list can be used to store and manage a collection of student records.
  - Insertion, deletion, and traversal operations can be performed efficiently.
  - Searching for a specific student may require traversing the linked list linearly.

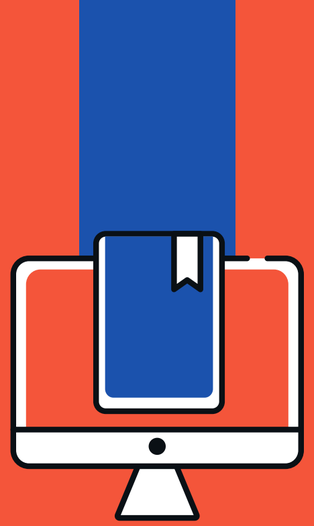
- **Arrays**
  - An array can be used to store a fixed-size collection of student records.
  - Each element of the array can represent a student record.
  - Random access to elements allows for efficient retrieval of student information.
  - Insertion and deletion operations may require shifting elements, resulting in potential inefficiencies.

Flowchart


User
Issue
Return
Enter University ID
Enter University ID
If ID matches among the list of UNI Id's
If ID matches among the list of UNI Id's
Yes
No
Yes
No
Enter Name of Book
Prints "Invalid ID"
Enter Name of Book
Prints "Invalid ID"
If it matches among the list of books
If it matches among the list of books
Yes
No
Yes
No
Quantity of that book is >0
Prints "Book not available"
Book returned successfully
Book is not available in the library
Yes
No
Prints "Book issued"
Prints "Book cannot be issued now"

Conclusion

I would like to conclude this project by stating that it was through this project that I learned how to effectively apply theoretical data structures in practical real-life scenarios.



Thanks

By: Shantanu Satpute

https://github.com/shantanusatpute/Project