Jaypee Institute of Information Technology

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING AND INFORMATION TECHNOLOGY

PROJECT REPORT

PROJECT TITLE: LIBRARY MANAGEMENT SYSTEM

Name	Enroll of Students
Ansh Pandey	23104019
Aryan Bhardwaj	23104014

Under Supervision of: Ms. Neha, Ms. Mayuri Gupta

Course Name: Data Structures and Algorithms Lab
Course Code: 18B15CS211
Program: B. Tech. IT
2nd Year
3rd semester

Declaration

I hereby declare that the project titled “Library Management System” is an original work carried out by me under the guidance of Ms. Neha and Ms. Mayuri Gupta . The project is submitted in partial fulfillment of the requirements for the course Data Structures and Algorithms Lab at JIIT Noida. The information provided in this report is authentic to the best of my knowledge, and all references to external sources are duly acknowledged. I further declare that this project has not been submitted to any other institution for the award of any degree or diploma. I take full responsibility for any errors or omissions in this report.

Library Management System

Abstract

This project aims to develop a comprehensive Library Management System (LMS) using C++ programming language. The LMS is designed to efficiently manage library operations, including book management, member management, book issuing and returning, and searching and sorting books. The system utilizes various data structures and algorithms to ensure optimal performance and functionality.

The LMS incorporates features such as adding, deleting, searching, and sorting books using data structures like arrays, linked lists, and binary search trees. It also manages library members using queues or linked lists and employs stacks or queues to handle book issuing and returning processes. To facilitate efficient searching and sorting, the system implements searching algorithms (Linear Search, Binary Search) and sorting algorithms (Selection, Bubble, Insertion). Additionally, a Binary Search Tree (BST) is used to store and manage book information, enabling efficient searching.

The implementation plan involves creating book and member structures, using appropriate data structures for book and member management, implementing functions for book management, member management, issuing and returning books, and developing a menu-driven program for user interaction. The C++ code structure outlines the classes and functions necessary for the system's functionality.

Key C++ Concepts and Their Applications

1. Data Structures:

- **Arrays:** Used for storing collections of books or members temporarily, especially when sorting or searching is required.
- **Linked Lists:** Employed to manage dynamic lists of books or members, particularly when frequent insertions and deletions are needed.
- **Binary Search Trees (BSTs):** Used to store and manage book information efficiently, allowing for efficient searching, insertion, and deletion based on book attributes.

2. Algorithms:

- **Searching Algorithms:**

    - **Linear Search:** Used for simple searching of books in unsorted arrays.
    - **Binary Search:** Used for efficient searching of books in sorted arrays or BSTs.
- **Sorting Algorithms:**
    - **Selection Sort:** Used for sorting books by different attributes (e.g., title, author).

3. Object-Oriented Programming (OOP):

- **Classes and Objects:** Used to model real-world entities like books and members, encapsulating their data and behavior.
- **Inheritance:** Potentially used to create derived classes from base classes (e.g., inheriting from a generic Book class to create specific book types).
- **Polymorphism:** Might be employed to implement different behaviors for objects of the same class (e.g., different ways to search for books based on search criteria).

4. Other C++ Features:

- **Pointers:** Used to dynamically allocate memory for data structures like linked lists.
- **Functions:** Used to modularize the code and improve readability.
- **Input/Output Streams:** Used for user interaction and file handling.
- **Control Flow (if-else statements, loops):** Used to control the program flow.

5. Advanced Algorithms:

- **Greedy Algorithms:**
    - **Floyd Warshall Algorithm:** Can be used to find the shortest path or least cost delivery route for managing book delivery to various branches.
- **Dynamic Programming:**
    - **Longest Common Subsequence (LCS):** Can be used in advanced search features to find the most similar books based on title or content by matching similar patterns.
- **String Algorithms:**
    - **Naïve String Matching:** Used for simple searching of books by title or author name.
    - **Rabin-Karp Algorithm:** Used for faster substring searching, especially useful for finding books with matching keywords or phrases.

Problem Statement and Objectives

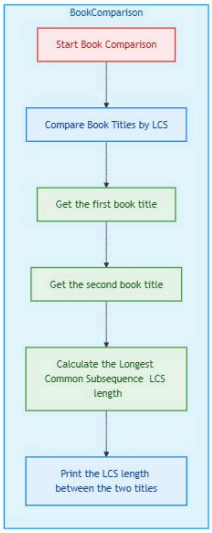
The increasing demand for an efficient and accessible library system calls for a solution that can effectively manage the wide array of operations involved, such as book issue and return, inventory management, and tracking of issued books. The existing manual systems are prone to errors, time-consuming, and lack real-time accessibility. This Library Management System is designed to automate and streamline these operations by providing a software-based solution for the library. The system aims to minimize human intervention, reduce errors, and ensure efficient management of library resources.

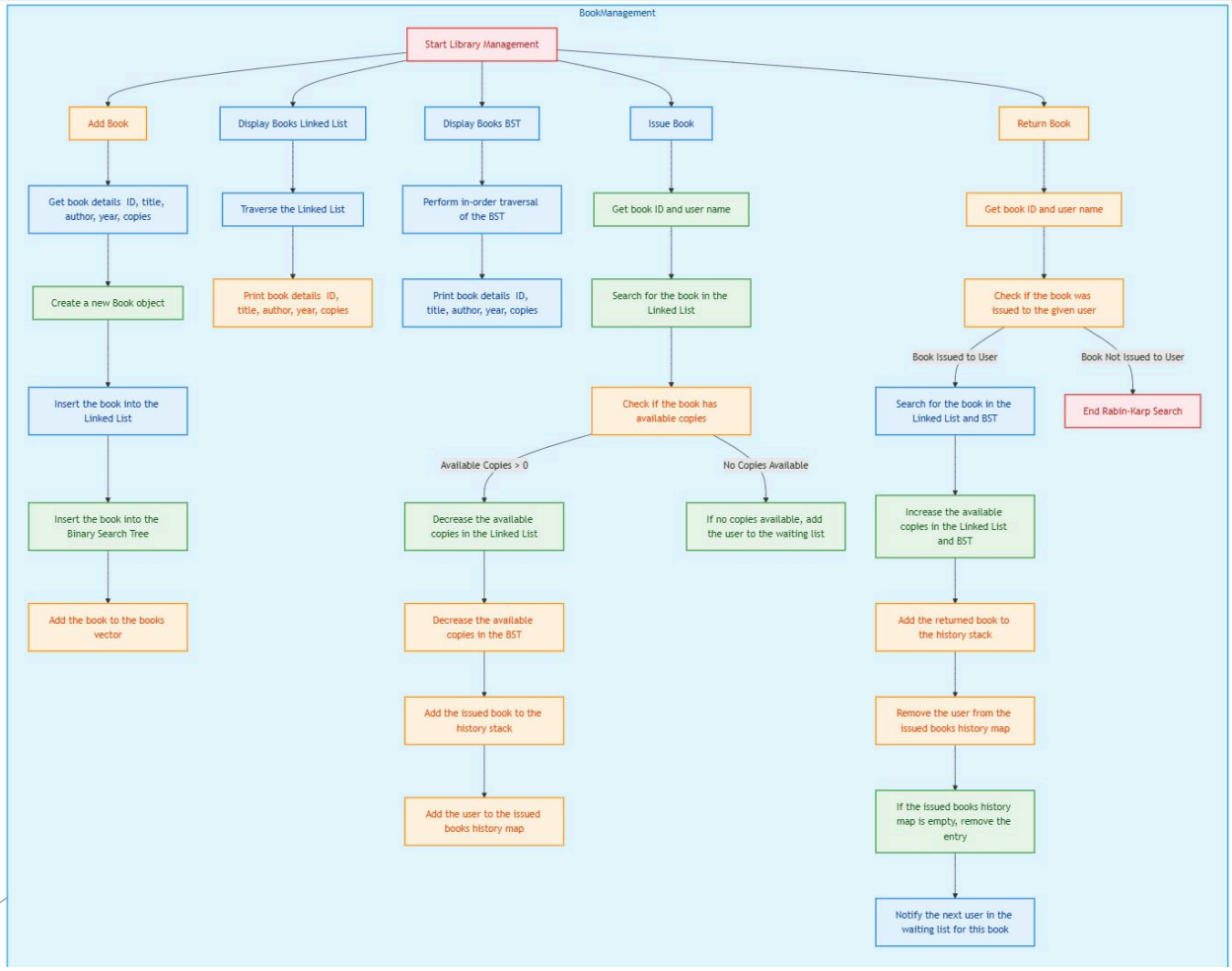
Objectives of the Project:

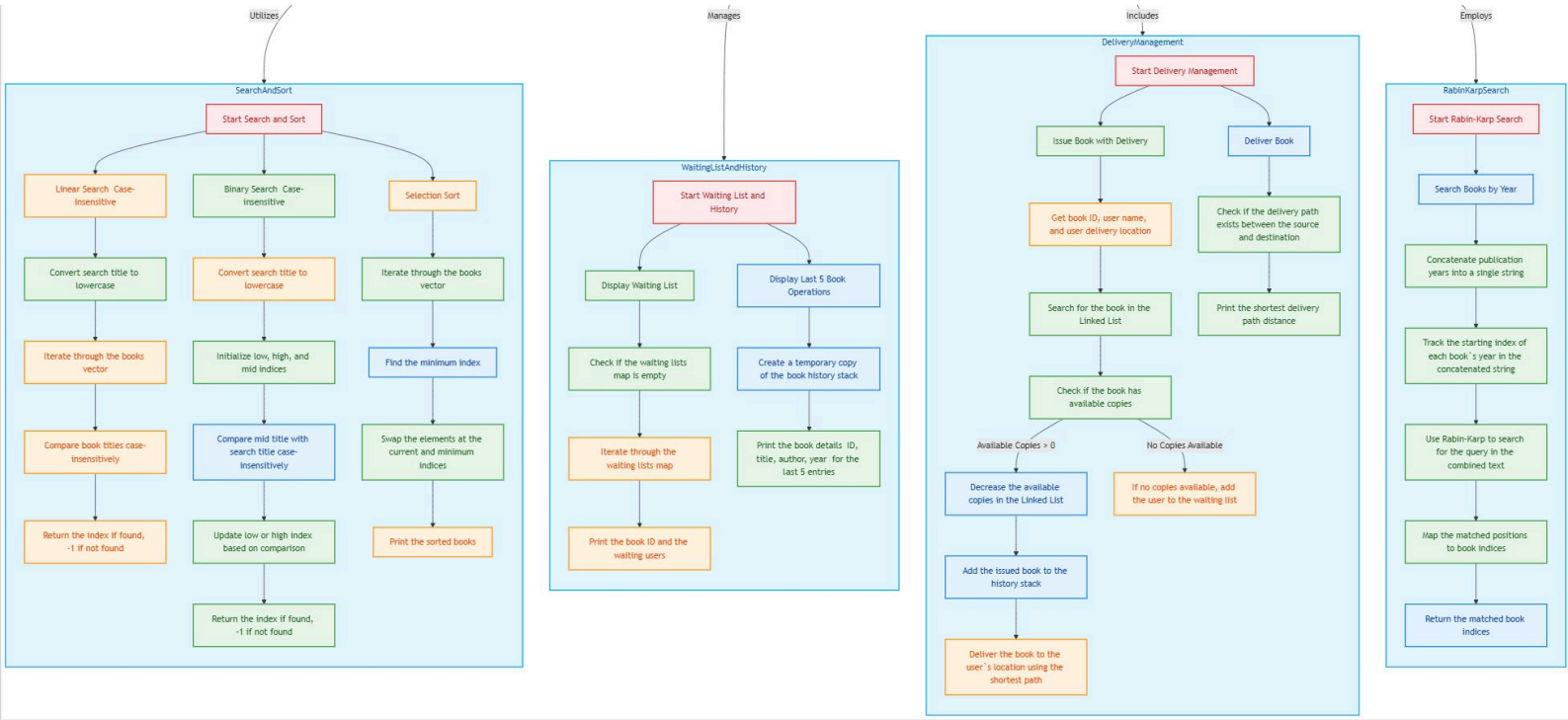
1. To develop an automated system for managing books in the library.
2. To facilitate easy book issue and return processes.
3. To incorporate book search and inventory tracking functionalities.
4. To implement a user-friendly interface for both library staff and patrons.
5. To integrate the delivery functionality for book issues, ensuring better service.
6. To use algorithms like Floyd-Warshall for optimizing book delivery routes.

This project aims to create a system that enhances the efficiency of library operations, ensuring smooth user experiences and better resource management.

FLOW OF THE PROGRAM


BookComparison
Start Book Comparison
Compare Book Titles by LCS
Get the first book title
Get the second book title
Calculate the Longest Common Subsequence LCS length
Print the LCS length between the two titles


BookManagement
Start Library Management
Add Book
Get book details ID, title, author, year, copies
Create a new Book object
Insert the book into the Linked List
Insert the book into the Binary Search Tree
Add the book to the books vector
Display Books Linked List
Traverse the Linked List
Print book details ID, title, author, year, copies
Display Books BST
Perform in-order traversal of the BST
Print book details ID, title, author, year, copies
Issue Book
Get book ID and user name
Search for the book in the Linked List
Check if the book has available copies
Available Copies > 0
Decrease the available copies in the Linked List
Decrease the available copies in the BST
Add the issued book to the history stack
Add the user to the issued books history map
No Copies Available
If no copies available, add the user to the waiting list
Return Book
Get book ID and user name
Check if the book was issued to the given user
Book Issued to User
Search for the book in the Linked List and BST
Increase the available copies in the Linked List and BST
Add the returned book to the history stack
Remove the user from the issued books history map
If the issued books history map is empty, remove the entry
Notify the next user in the waiting list for this book
Book Not Issued to User
End Rabin-Karp Search


Utilizes
SearchAndSort
Start Search and Sort
Linear Search Case-insensitive
Binary Search Case-insensitive
Selection Sort
Convert search title to lowercase
Convert search title to lowercase
Iterate through the books vector
Iterate through the books vector
Initialize low, high, and mid indices
Find the minimum index
Compare book titles case-insensitively
Compare mid title with search title case-insensitively
Swap the elements at the current and minimum indices
Return the index if found, -1 if not found
Update low or high index based on comparison
Print the sorted books
Return the index if found, -1 if not found
Manages
WaitingListAndHistory
Start Waiting List and History
Display Waiting List
Display Last 5 Book Operations
Check if the waiting lists map is empty
Create a temporary copy of the book history stack
Iterate through the waiting lists map
Print the book details ID, title, author, year for the last 5 entries
Print the book ID and the waiting users
Includes
DeliveryManagement
Start Delivery Management
Issue Book with Delivery
Deliver Book
Get book ID, user name, and user delivery location
Check if the delivery path exists between the source and destination
Search for the book in the Linked List
Print the shortest delivery path distance
Check if the book has available copies
Available Copies > 0
No Copies Available
Decrease the available copies in the Linked List
If no copies available, add the user to the waiting list
Add the issued book to the history stack
Deliver the book to the user`s location using the shortest path
Employs
RabinKarpSearch
Start Rabin-Karp Search
Search Books by Year
Concatenate publication years into a single string
Track the starting index of each book`s year in the concatenated string
Use Rabin-Karp to search for the query in the combined text
Map the matched positions to book indices
Return the matched book indices

SOURCE CODE:-

```
#include <iostream>
#include <string> #include
<vector> #include
<algorithm> #include
<map> #include <queue>
#include <stack> #include
<cctype> #include
<limits.h> using
namespace std; class
Book {
public:
    int id; string
    title;
    string author;
    int  publicationYear;  int
    availableCopies;
    Book() {}
    Book(int id, string title, string author, int year, int copies)
        :  id(id),  title(title),  author(author),  publicationYear(year),  availableCopies(copies)  {}
    string getTitle() const {
        return title;
    }
    int getPublicationYear() const {
        return publicationYear;
    }
};
struct     Node     {
    Book        data;
    Node* next;
    Node(Book b) : data(b), next(nullptr) {}
};
class LinkedList { public:
    Node* head;
    LinkedList() : head(nullptr) {} void
    insert(Book b) {
        Node* newNode = new Node(b); if
        (!head) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != nullptr) {
                temp = temp->next;
            }
```

```
            temp->next = newNode;
        }
    }
    void display() {
        Node* temp = head; while
        (temp != nullptr) {
            cout << "Book ID: " << temp->data.id << ", Title: " << temp->data.title
                << ", Author: " << temp->data.author << ", Year: " << temp->data.publicationYear
                << ", Available Copies: " << temp->data.availableCopies << endl;
            temp = temp->next;
        }
    }
    void remove(int bookID) { if
        (!head) return;
        if (head->data.id == bookID) {
            Node* temp = head;
            head = head->next;
            delete temp;
            return;
        }
        Node* prev = head; Node*
        curr = head->next;
        while (curr != nullptr && curr->data.id != bookID) { prev
            = curr;
            curr = curr->next;
        }
        if (curr != nullptr) {
            prev->next = curr->next;
            delete curr;
        }
    }
    Book* searchByID(int bookID) {
        Node* temp = head;
        while (temp != nullptr) {
            if (temp->data.id == bookID) {
                return &temp->data;
            }
            temp = temp->next;
        }
        return nullptr;
    }
};
queue<string>      waitingList;
stack<Book>      bookHistory;
struct BSTNode {
    Book data;
    BSTNode* left;
    BSTNode* right;
```

```
    BSTNode(Book b) : data(b), left(nullptr), right(nullptr) {}
};
class BST {
public:
    BSTNode* root; BST() :
    root(nullptr) {}

    void insert(Book b) {
        root = insertRec(root, b);
    }

    BSTNode* insertRec(BSTNode* node, Book b) { if
        (node == nullptr) {
            return new BSTNode(b);
        }
        if (b.id < node->data.id) {
            node->left = insertRec(node->left, b);
        } else {
            node->right = insertRec(node->right, b);
        }
        return node;
    }

    void inOrder(BSTNode* node) { if
        (node != nullptr) {
            inOrder(node->left);
            cout << "Book ID: " << node->data.id << ", Title: " << node->data.title
                << ", Author: " << node->data.author << ", Year: " << node->data.publicationYear
                << ", Available Copies: " << node->data.availableCopies << endl;
            inOrder(node->right);
        }
    }

    BSTNode* searchByID(BSTNode* node, int bookID) { if
    (node == nullptr || node->data.id == bookID) {
        return node;
    }
    if (bookID < node->data.id) {
        return searchByID(node->left, bookID);
    } else {
        return searchByID(node->right, bookID);
        }
    }

    void display() {
        inOrder(root);
    }
```

```
};
void selectionSort(vector<Book>& books) { int
    n = books.size();
    for (int i = 0; i < n - 1; i++) { int
        minIdx = i;
        for (int j = i + 1; j < n; j++) {
            if (books[j].title < books[minIdx].title) {
                minIdx = j;
            }
        }
        swap(books[i], books[minIdx]);
    }
    for(auto i:books)
    cout << "Book ID: " << i.id << ", Title: " << i.title
                << ", Author: " << i.author << ", Year: " << i.publicationYear
                << ", Available Copies: " << i.availableCopies << endl;
}
string toLowerCase(const string& str) { string
    result = str;
    transform(result.begin(), result.end(), result.begin(), [](unsigned char c) {
        return tolower(c);
    });
    return result;
}
int linearSearch(vector<Book>& books, string title) { string
    searchTitle = toLowerCase(title);
    for (int i = 0; i < books.size(); i++) {
        if (toLowerCase(books[i].title) == searchTitle) { return i;
        }
    }
    return -1;
}
int binarySearch(vector<Book>& books, string title) { string
    searchTitle = toLowerCase(title);
    int low = 0, high = books.size() - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        string midTitle = toLowerCase(books[mid].title);

        if (midTitle == searchTitle) return mid;
        if (midTitle < searchTitle) low = mid + 1; else
        high = mid - 1;
    }
    return -1;
}
map<int, queue<string>> issuedBooksHistory;
```

```
map<int, queue<string>> waitingLists;
void issueBook(LinkedList& list, BST& tree, int bookID, string user) { Book*
    book = list.searchByID(bookID);
    if (book != nullptr) {
        if (book->availableCopies > 0) {
            book->availableCopies--;

            BSTNode* bookTreeNode = tree.searchByID(tree.root, bookID); if
            (bookTreeNode != nullptr) {
                bookTreeNode->data.availableCopies--;
            }

            cout << "Book issued successfully to " << user << endl;


            bookHistory.push(*book); if
            (bookHistory.size() > 5) {
                bookHistory.pop();
            }


            issuedBooksHistory[bookID].push(user);

        } else {
            cout << "No copies available. Adding " << user << " to the waiting list for book ID "
                 << bookID << "." << endl;
            waitingLists[bookID].push(user);
        }
    } else {
        cout << "Book not found." << endl;
    }
}
void returnBook(LinkedList& list, BST& tree, int bookID, string user) {

    if (issuedBooksHistory.find(bookID) == issuedBooksHistory.end() ||
issuedBooksHistory[bookID].empty()) {
        cout << "This book was not issued to anyone or is invalid." << endl; return;
    }
    queue<string>& issuedQueue = issuedBooksHistory[bookID]; bool
    isUserValid = false;


    queue<string> tempQueue = issuedQueue;
    while (!tempQueue.empty()) {
        if (tempQueue.front() == user) {
            isUserValid = true;
            break;
```

```
        }
        tempQueue.pop();
    }

    if (!isUserValid) {
        cout << "This book was not issued to " << user << ". Return rejected." << endl; return;
    }

    Book* bookList = list.searchByID(bookID);
    BSTNode* bookTree = tree.searchByID(tree.root, bookID); if

    (bookList != nullptr && bookTree != nullptr) {

        bookList->availableCopies++;
        bookTree->data.availableCopies++;

        cout << "Book returned successfully by " << user << "." << endl;
        bookHistory.push(*bookList);
        if (bookHistory.size() > 5) {
            bookHistory.pop();
        }
        issuedQueue = issuedBooksHistory[bookID];
        while (!issuedQueue.empty()) {
            if (issuedQueue.front() == user) {
                issuedQueue.pop();
                break;
            }
            issuedQueue.pop();
        }
        if (issuedQueue.empty()) {
            issuedBooksHistory.erase(bookID);
        }
        if (waitingLists.find(bookID) != waitingLists.end() && !waitingLists[bookID].empty()) {
            string waitingUser = waitingLists[bookID].front();
            cout << "Notifying " << waitingUser << " from the waiting list for book ID " << bookID
<< "." << endl;
            waitingLists[bookID].pop();


            if (waitingLists[bookID].empty()) {
                waitingLists.erase(bookID);
            }
        }
    } else {
        cout << "Book not found in the system." << endl;
    }
}
```

```
int naiveStringMatching(const string& text, const string& pattern) { int n
    = text.length();
    int m = pattern.length();


    for (int i = 0; i <= n - m; i++) { int
        j = 0;
        while (j < m && text[i + j] == pattern[j]) { j++;
        }

        if (j == m) {
            return i;
        }
    }
    return -1;
}
void addBook(vector<Book>& books, LinkedList& list, BST& tree, Book b) { books.push_back(b);
    list.insert(b);
    tree.insert(b);
}
int LCS(string str1, string str2) { int
    m = str1.length();
    int n = str2.length();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));


    for (int i = 1; i <= m; i++) { for
        (int j = 1; j <= n; j++) {
            if (str1[i - 1] == str2[j - 1])
                dp[i][j] = dp[i - 1][j - 1] + 1; else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }

    return dp[m][n];
}
void compareBooksByTitle(string book1, string book2) { int
    lcsLength = LCS(book1, book2);
    cout << "Longest Common Subsequence Length between '" << book1 << "' and '" <<
book2 << "': " << lcsLength << endl;
}
void floydWarshall(vector<vector<int>>& dist, int V) { for
    (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) { for
            (int j = 0; j < V; j++) {
```

```
            if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX && dist[i][k] + dist[k][j] < dist[i][j]) {
                dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
}
void deliverBook(const vector<vector<int>>& dist, int source, int destination) { if
    (dist[source][destination] == INT_MAX) {
        cout << "Delivery not possible between these locations." << endl;
    } else {
        cout << "Shortest delivery path from branch " << source << " to location " << destination
             << " is " << dist[source][destination] << " units." << endl;
    }
}
void issueBookWithDelivery(LinkedList& list, int bookID, string user, const vector<vector<int>>&
dist, int userLocation, int libraryBranch) {
    Book* book = list.searchByID(bookID); if
    (book != nullptr) {
        if (book->availableCopies > 0) {
            book->availableCopies--;
            cout << "Book issued successfully to " << user << endl;
            bookHistory.push(*book);
            if (bookHistory.size() > 5) {
                bookHistory.pop();
            }


            deliverBook(dist, libraryBranch, userLocation);
        } else {
            cout << "No copies available. Adding " << user << " to the waiting list." << endl;
            waitingList.push(user);
        }
    } else {
        cout << "Book not found." << endl;
    }
}

vector<int> rabinKarpSearch(const string& text, const string& pattern) {
    vector<int> result;
    int m = pattern.length(); int
    n = text.length(); const int d
    = 256;
    const int q = 101;

    int i, j; int p
    = 0; int t =
    0;
```

```
    int h = 1;


    for (i = 0; i < m - 1; i++) { h
        = (h * d) % q;
    }


    for (i = 0; i < m; i++) {
        p = (d * p + pattern[i]) % q; t =
        (d * t + text[i]) % q;
    }


    for (i = 0; i <= n - m; i++) { if

        (p == t) {

            for (j = 0; j < m; j++) {
                if (text[i + j] != pattern[j]) {
                    break;
                }
            }


            if (j == m) {
                result.push_back(i);
            }
        }

        if (i < n - m) {
            t = (d * (t - text[i] * h) + text[i + m]) % q;


            if (t < 0) {
                t = t + q;
            }
        }
    }

    return result;
}
    vector<int> searchBooksByYear(const vector<Book>& books, const string& yearQuery) {
    vector<int> matchedBookIndexes;
    string combinedText = "";


    for (int i = 0; i < books.size(); i++) {
```

```
        combinedText += to_string(books[i].getPublicationYear()) + " ";
    }


    vector<int> result = rabinKarpSearch(combinedText, yearQuery);


    for (int index : result) {
        matchedBookIndexes.push_back(index);
    }

    return matchedBookIndexes;
}

int main() { vector<Book>
    books; LinkedList list;
    BST tree;


    int V = 4; vector<vector<int>>
    dist = {
        {0, 5, INT_MAX, 10},
        {5, 0, 3, INT_MAX},
        {INT_MAX, 3, 0, 1},
        {10, INT_MAX, 1, 0}
    };


    floydWarshall(dist, V);


    addBook(books, list, tree, Book(1, "The God of Small Things", "Arundhati Roy", 1997, 2));
    addBook(books, list, tree, Book(2, "Autobiography Of Ansh", "Ansh Pandey", 2016, 1));
    addBook(books, list, tree, Book(3, "The Ramayana", "Valmiki", 500, 3));
    addBook(books, list, tree, Book(4, "The Guide", "R.K. Narayan", 1968, 4));
    addBook(books, list, tree, Book(5, "Gitanjali", "Rabindranath Tagore", 1910, 0));

    int choice; do
    {
        cout << "\nLibrary Management System\n"; cout
        << "1. Add Book\n";
        cout << "2. Display Books (Linked List)\n";
        cout << "3. Display Books (BST)\n";
        cout << "4. Issue Book\n"; cout
        << "5. Return Book\n";
        cout << "6. Search Book (Linear Search)\n"; cout
        << "7. Search Book (Binary Search)\n";
```

```
cout << "8. Sort Books (Selection Sort)\n";
cout << "9. Display Waiting List\n";
cout << "10. View Last 5 Book Operations (History)\n"; cout
<< "11. Compare two book titles by LCS\n";
cout << "12. Issue a Book with Delivery\n";
cout << "13. Search Books by Year (Rabin-Karp)\n"; cout
<< "0. Exit\n";
cout << "Enter your choice: "; cin
>> choice;

switch (choice) {
   case 1: {
      int id, year, copies;
      string title, author;
      cout << "Enter Book ID: "; cin >> id;
      cout << "Enter Title: "; cin.ignore(); getline(cin, title); cout
      << "Enter Author: "; getline(cin, author);
      cout << "Enter Publication Year: "; cin >> year; cout
      << "Enter Available Copies: "; cin >> copies; Book
      b(id, title, author, year, copies); addBook(books, list,
      tree, b);
      break;
   }
   case 2:
      list.display();
      break;
   case 3:
      tree.display(); break;
   case 4: { int
      id;
      string user;
      cout << "Enter Book ID to issue: "; cin >> id;
      cout << "Enter User Name: "; cin.ignore(); getline(cin, user);
      issueBook(list, tree, id, user);
      break;
   }
   case 5: { int
      id;
      string user;
      cout << "Enter Book ID to return: "; cin >> id;
      cout << "Enter User Name: "; cin.ignore(); getline(cin, user);
      returnBook(list, tree, id, user);
      break;
   }
   case 6: { string
      title;
      cout << "Enter title to search: "; cin.ignore(); getline(cin, title);
```

```
    int index = linearSearch(books, title);
    if (index != -1) cout << "Book found: " << books[index].title << endl;
    else cout << "Book not found.\n";
    break;
}
case 7: { string
    title;
    cout << "Enter title to search: "; cin.ignore(); getline(cin, title); sort(books.begin(),
    books.end(), [](Book a, Book b) { return a.title < b.title; }); int index =
    binarySearch(books, title);
    if (index != -1) cout << "Book found: " << books[index].title << endl;
    else cout << "Book not found.\n";
    break;
}
case 8:
    selectionSort(books);
    sort(books.begin(), books.end(), [](Book a, Book b) { return a.title < b.title; }); cout
    << "Books sorted by title.\n";
    break;
case 9:
    cout << "Waiting List:\n"; if
    (waitingList.empty()) {
        cout << "No one in the waiting list.\n";
    } else {
        queue<string> tempQueue = waitingList; while
        (!tempQueue.empty()) {
            cout << tempQueue.front() << endl;
            tempQueue.pop();
        }
    }
    break;
case 10: {
    cout << "Last 5 Book Operations (Issued/Returned):\n";
    stack<Book> tempHistory = bookHistory;
    int count = 0;
    while (!tempHistory.empty() && count < 5) { Book
        b = tempHistory.top();
        cout << "Book ID: " << b.id << ", Title: " << b.title
             << ", Author: " << b.author << ", Year: " << b.publicationYear << endl;
        tempHistory.pop();
        count++;
    }
    break;
}
 case 11: {
    string book1, book2;
    cout << "Enter first book title: "; cin.ignore();
```

```
            getline(cin, book1);
            cout << "Enter second book title: ";
            getline(cin, book2);
            compareBooksByTitle(book1, book2); break;
        }
         case 12: { int
        id; string user;
        int userLocation;
        cout << "Enter Book ID to issue: "; cin >> id;
        cout << "Enter User Name: "; cin.ignore(); getline(cin, user);
        cout << "Enter User Delivery Location (0-3): "; cin >> userLocation;
        issueBookWithDelivery(list, id, user, dist, userLocation, 0);
        break;
      }
      case 13: { string
   yearQuery;
   cout << "Enter publication year to search for: ";
   cin.ignore(); getline(cin, yearQuery);
   vector<int> foundBooks = searchBooksByYear(books, yearQuery); if
   (foundBooks.empty()) {
      cout << "No books found for the year " << yearQuery << ".\n";
   } else {
      cout << "Books found for the year " << yearQuery << ":\n"; for
      (int index : foundBooks) {
         cout << books[index].getTitle() << endl;
      }
   }
   break;
}
        case 0:
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice.\n";
            break;
      }
   } while (choice != 0);

   return 0;
```

OUTPUT SNAPSHOTS:-

```
Library Management System
1. Add Book
2. Display Books (Linked List)
3. Display Books (BST)
4. Issue Book
5. Return Book
6. Search Book (Linear Search)
7. Search Book (Binary Search)
8. Sort Books (Selection Sort)
9. Display Waiting List
10. View Last 5 Book Operations (History)
11. Compare two book titles by LCS
12. Issue a Book with Delivery
13. Search Books by Year (Rabin-Karp)
0. Exit
Enter your choice: 1
```

```
Enter your choice: 1
Enter Book ID: 29
Enter Title: WIMPY KID
Enter Author: ARYAN
Enter Publication Year: 2000
Enter Available Copies: 20
```

```
Enter your choice: 2
Book ID: 1, Title: The God of Small Things, Author: Arundhati Roy, Year: 1997, Available Copies: 2
Book ID: 2, Title: Autobiography Of Ansh, Author: Ansh Pandey, Year: 2016, Available Copies: 1
Book ID: 3, Title: The Ramayana, Author: Valmiki, Year: 500, Available Copies: 3
Book ID: 4, Title: The Guide, Author: R.K. Narayan, Year: 1968, Available Copies: 4
Book ID: 5, Title: Gitanjali, Author: Rabindranath Tagore, Year: 1910, Available Copies: 0
Book ID: 29, Title: WIMPY KID, Author: ARYAN, Year: 2000, Available Copies: 20
```

```
Enter your choice: 3
Book ID: 1, Title: The God of Small Things, Author: Arundhati Roy, Year: 1997, Available Copies: 2
Book ID: 2, Title: Autobiography Of Ansh, Author: Ansh Pandey, Year: 2016, Available Copies: 1
Book ID: 3, Title: The Ramayana, Author: Valmiki, Year: 500, Available Copies: 3
Book ID: 4, Title: The Guide, Author: R.K. Narayan, Year: 1968, Available Copies: 4
Book ID: 5, Title: Gitanjali, Author: Rabindranath Tagore, Year: 1910, Available Copies: 0
```

```
Enter your choice: 4
Enter Book ID to issue: 2
Enter User Name: ANSH
Book issued successfully to ANSH
```

```
Enter your choice: 5
Enter Book ID to return: 3
Enter User Name: ANSH
This book was not issued to anyone or is invalid.
```

```
Enter your choice: 6
Enter title to search: The Ramayana
Book found: The Ramayana
```

```
0. Exit
Enter your choice: 7
Enter title to search: The Ramayana
Book found: The Ramayana
```

```
Book ID: 2, Title: Autobiography Of Ansh, Author: Ansh Pandey, Year: 2016, Available Copies: 1
Book ID: 5, Title: Gitanjali, Author: Rabindranath Tagore, Year: 1910, Available Copies: 0
Book ID: 1, Title: The God of Small Things, Author: Arundhati Roy, Year: 1997, Available Copies: 2
Book ID: 4, Title: The Guide, Author: R.K. Narayan, Year: 1968, Available Copies: 4
Book ID: 3, Title: The Ramayana, Author: Valmiki, Year: 500, Available Copies: 3
Books sorted by title.
```

```
Enter your choice: 11
Enter first book title: The God of Small Things
Enter second book title: The Ramayana
Longest Common Subsequence Length between 'The God of Small Things' and 'The Ramayana': 7
```

```
Enter your choice: 12
Enter Book ID to issue: 1
Enter User Name: aryan
Enter User Delivery Location (0-3): 1
Book issued successfully to aryan
Shortest delivery path from branch 0 to location 1 is 5 units.
```

```
0. Exit
Enter your choice: 10
Last 5 Book Operations (Issued/Returned):
Book ID: 4, Title: The Guide, Author: R.K. Narayan, Year: 1968
Book ID: 4, Title: The Guide, Author: R.K. Narayan, Year: 1968
Book ID: 3, Title: The Ramayana, Author: Valmiki, Year: 500
Book ID: 2, Title: Autobiography Of Ansh, Author: Ansh Pandey, Year: 2016
Book ID: 1, Title: The God of Small Things, Author: Arundhati Roy, Year: 1997
```

```
Enter your choice: 13
Enter publication year to search for: 2016
Books found for the year 2016:
Autobiography Of Ansh
```

```
Enter your choice: 9
Waiting List:
Book ID: 5
Username - aryan
```

Conclusion and Future Work

In conclusion, the Library Management System provides a reliable and efficient method for managing the various activities of a library. By automating the process of book issue and return, as well as implementing real-time tracking of books, the system significantly reduces errors and saves time. Additionally, the integration of the
Floyd-Warshall algorithm for book delivery optimizes the delivery process, enhancing the overall service experience for library patrons. The system is user-friendly, easy to maintain, and scalable, making it suitable for libraries of different sizes.

Future Work:

The current version of the Library Management System can be further enhanced by adding additional features such as:

1. Implementing user authentication to restrict access to sensitive operations.
2. Adding a digital payment system for fines and late returns.
3. Integrating an online catalog and remote book reservation system.
4. Enhancing the delivery system by incorporating real-time tracking and notifications.
5. Implementing a recommendation system based on users' borrowing history.

These future enhancements will contribute to making the system even more efficient, secure, and user-friendly, providing libraries with a more comprehensive solution for managing their resources.

References

1. GeeksforGeeks. "Floyd-Warshall Algorithm". Available at: https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/
2. GeeksforGeeks. "Longest Common Subsequence | Dynamic Programming". Available at: https://www.geeksforgeeks.org/longest-common-subsequence-dp-4/
3. GeeksforGeeks. "Naive String Matching Algorithm". Available at: https://www.geeksforgeeks.org/naive-algorithm-for-pattern-searching/
4. GeeksforGeeks. "Binary Search Tree | Set 1 (Search and Insertion)". Available at: https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/
5. GeeksforGeeks. "Linked List | Set 1 (Introduction)". Available at: https://www.geeksforgeeks.org/linked-list-set-1-introduction/