DATA STRUCTURES AND ALGORITHMS PROJECT

CONSTANTINESCU MARIA

421G

Library Management System

The Library Management System (LMS) project manages a library's inventory. The core of the system relies on a Binary Search Tree (BST) data structure to store and manipulate information about books.

The addition of books to the library is supported, where users input details such as title, author, and a unique code for each book. The BST ensures the organized storage of books based on their titles, allowing for quick and efficient search and retrieval.

Search functionality is a key feature, enabling users to search for books by either title or author. The BST structure ensures fast and accurate search results, displaying comprehensive information about the located books.

Deletion of books from the library is supported, with the BST structure dynamically adjusting to maintain organizational integrity. Users are prompted to enter the title of the book they wish to delete.

The system provides an option to display all books in the library in a sorted order. This feature utilizes an in-order traversal of the BST, offering users a view of the library's contents.

The system's efficient use of a Binary Search Tree ensures that library operations are scalable, with searches, insertions, and deletions having a time complexity of O(log n).

The Library Management System project provides an effective solution for librarians to manage, search, and organize their book inventory.

#include <iostream>  
#include <string>  
#include <cstring>  
using namespace std;  
  
// Structure to represent a book  
typedef struct Book {  
 string title;  
 string author;  
 int cod;  
 Book(std::string t, std::string a, int c) : title(t), author(a), cod(c) {}  
}BOOK;  
  
// Node structure for the binary search tree  
typedef struct TreeNode {  
 Book book;  
 TreeNode \* left;  
 TreeNode \* right;  
 // Constructor that accepts a Book object  
 TreeNode(const Book& b) : book(b), left(nullptr), right(nullptr) {}  
}treenode;  
  
// Binary Search Tree class  
class LibraryManagementSystem {  
private:  
 TreeNode \*root;  
  
 TreeNode \* insert(TreeNode \*node, const Book &book);  
  
 TreeNode \*remove(TreeNode \*node, const string &title);  
  
 TreeNode \*findMin(TreeNode \*node);  
  
 TreeNode \*searchtitle(TreeNode \*node, const string &key);  
  
 TreeNode \*searchautor(TreeNode \*node, const string &key);  
  
 void inorderTraversal(TreeNode \*node);  
  
  
public:  
 LibraryManagementSystem() : root(nullptr) {} //constructor binary search tree is empty  
  
 void addBook(const Book &book);  
  
 void deleteBook(const string &title);  
  
 void searchBookByName(const string &title);  
  
 void searchBookByAutor(const string &author);  
  
 void displayBooks();  
  
 // Destructor  
 ~LibraryManagementSystem();  
};  
  
// Function to insert a book into the BST  
TreeNode \* LibraryManagementSystem::insert(TreeNode \*node, const Book &book) {  
 if (node == nullptr)  
 return new TreeNode(book);  
  
 if (book.title < node->book.title) {  
 node->left = insert(node->left, book);  
 } else if (book.title > node->book.title) {  
 node->right = insert(node->right, book);  
 }  
 // Return the updated node after insertion  
 return node;  
}  
  
  
TreeNode\* LibraryManagementSystem::findMin(TreeNode\* node) {  
 while (node->left != nullptr) {  
 node = node->left;  
 }  
 return node;  
} //leftmost node in a subtree contains the smallest value in that subtree  
  
// Function to remove a book from the BST  
TreeNode\* LibraryManagementSystem::remove(TreeNode\* node, const string& title) {  
 if (node == nullptr) {  
 return nullptr;  
 }  
  
 if (title < node->book.title) {  
 node->left = remove(node->left, title); //recursive call of the remove function on the left subtree  
 } else if (title > node->book.title) {  
 node->right = remove(node->right, title);  
 } else {  
 // Node with only one child or no child  
 if (node->left == nullptr) {  
 TreeNode\* temp = node->right;  
 delete node;  
 return temp;  
 } else if (node->right == nullptr) {  
 TreeNode\* temp = node->left;  
 delete node;  
 return temp;  
 }  
 // Node with two children  
 TreeNode\* temp = findMin(node->right); //The minimum node in the right subtree is the node with the smallest value greater than the current node  
 node->book = temp->book;  
 node->right = remove(node->right, temp->book.title);  
 }  
  
 return node;  
}  
  
  
// Function to search for a book by title in the BST  
TreeNode\* LibraryManagementSystem::searchtitle(TreeNode\* node, const string& key) {  
 if (node == nullptr)  
 return nullptr;  
  
 if (node->book.title == key) {  
 return node;  
 }  
  
 if (key < node->book.title) {  
 return searchtitle(node->left, key);  
 }  
  
 return searchtitle(node->right, key);  
}  
  
TreeNode\* LibraryManagementSystem::searchautor(TreeNode\* node, const string& key) {  
 if (node == nullptr)  
 return nullptr;  
  
 if (node->book.author == key) {  
 return node;  
 }  
  
 if (key < node->book.author) {  
 return searchautor(node->left, key);  
 }  
  
 return searchautor(node->right, key);  
}  
void LibraryManagementSystem::inorderTraversal(TreeNode\* node) {  
 if (node != nullptr) {  
 inorderTraversal(node->left);  
 cout << "Title: " << node->book.title << ", Author: " << node->book.author << endl;  
 inorderTraversal(node->right);  
 }  
}  
  
void LibraryManagementSystem::addBook(const Book& book) {  
 root = insert(root, book);  
 // cout << "Book added successfully." << endl;  
}  
  
void LibraryManagementSystem::deleteBook(const string& title) {  
 root = remove(root, title);  
 cout << "Book deleted successfully." << endl;  
  
}  
  
void LibraryManagementSystem::searchBookByName(const string& title) {  
 TreeNode\* result = searchtitle(root, title);  
 if (result != nullptr) {  
 cout << "Title: " << result->book.title << endl;  
 cout << "Author: " << result->book.author << endl;  
 } else {  
 cout << "Book not found." << endl;  
 }  
}  
  
void LibraryManagementSystem::searchBookByAutor(const string& autor) {  
 TreeNode \*result = searchautor(root, autor);  
 if (result != nullptr) {  
 cout << "Title: " << result->book.title << endl;  
 cout << "Author: " << result->book.author << endl;  
 } else {  
 cout << "Book not found." << endl;  
 }  
}  
  
void LibraryManagementSystem::displayBooks() {  
 cout << "Library Books:" << endl;  
 inorderTraversal(root);  
 cout << endl;  
}  
// Destructor implementation  
LibraryManagementSystem::~LibraryManagementSystem() {  
 //cout << "Destructor was called" << endl;  
}  
  
  
int main() {  
 LibraryManagementSystem library;  
  
 // Adding books to the library  
 library.addBook(Book("The Catcher in the Rye", "J.D. Salinger", 12345));  
 library.addBook(Book("To Kill a Mockingbird", "Harper Lee", 67890));  
 library.addBook(Book("1984", "George Orwell", 11111));  
 library.addBook(Book("The Adventures of Tom Sawyer", "Mark Twain", 64825));  
 library.addBook(Book("The Wizard of Oz", "L. Frank Baum", 23340));  
 library.addBook(Book("Anna Karenina", "Leo Tolstoy", 97432));  
 int x;  
 do{  
 cout<<endl;  
 cout << "1.Display all books" << endl;  
 cout << "2.Search book by name" << endl;  
 cout << "3.Search book by author" << endl;  
 cout << "4.Deleting a book from the library" << endl;  
 cout << "5.Exit" << endl;  
 cout<<endl;  
  
 cin >> x;  
  
 if (x == 1) {  
 library.displayBooks();}  
 else if (x == 2) {  
 cin.ignore();  
 string y;  
 cout << "Enter name of the book:";  
 getline(cin,y);  
 library.searchBookByName(y);  
 } else if (x == 3) {  
 cin.ignore();  
 string z;  
 cout << "Enter author of the book:";  
 getline(cin,z);  
 library.searchBookByAutor(z);  
 }  
 else if(x == 4){  
 cin.ignore();  
 string w;  
 cout << "Enter name of the book you want to delete:";  
 getline(cin,w);  
 library.deleteBook(w);  
 library.displayBooks();  
 }  
 else if(x==5)  
 cout << "Exiting the program." << endl;  
 }while(x!=5);  
 return 0;  
}













