```cpp
#include <iostream>
#include <string>
#include <stdexcept>
#include <conio.h>
#include <windows.h>
#include <unordered_map>
#include <queue>
#include <vector>
#include <climits>
#include <algorithm>
#include <fstream>
#include <sstream>

using namespace std;

class LoginSystem
{
private:
    string username;
    string password;

public:
    void showWelcomeScreen()
    {
        system("cls");
        cout << "\n\n";
        cout << "\t\t\t=====================================================";
        cout << "\n\n\t\t\t\tWELCOME TO LIBRARY MANAGEMENT SYSTEM";
        cout << "\n\n\t\t\t=====================================================";
        getch();
    }

    void authenticateUser()
    {
        const string validUsernames[] = {"treena", "takia", "arjuna"};
        const string validPasswords[] = {"123456", "123456", "123456"};
        const int numUsers = sizeof(validUsernames) / sizeof(validUsernames[0]);

        while (true)
        {
            try
            {
                system("cls");
                cout << "\n\n";
                cout << "\t\t\t============================";
                cout << "\n\n\t\t\t\tLOGIN PANEL";
                cout << "\n\n\t\t\t============================";
                cout << "\n\n\n ENTER USER NAME: ";
                cin >> username;
                cout << "\n\n ENTER PASSWORD: ";
                password = "";
                for (int i = 1; i <= 6; i++)
                {
                    password += getch();
                    cout << "*";
                }
                bool authenticated = false;
                for (int i = 0; i < numUsers; i++)
                {
                    if (username == validUsernames[i] && password == validPasswords[i])
                    {
                        authenticated = true;
                        break;
                    }
                }

                if (authenticated)
                {
                    cout << "\n\n\n\t\t\tCONGRATULATIONS! LOGIN SUCCESSFUL";
                    cout << "\n\n\n\t\t\t\tLOADING";
                    for (int i = 1; i <= 6; i++)
                    {
                        Sleep(100);
                        cout << ".";
                    }
                    Sleep(100);
                    system("cls");
                    cout << "\n\n";
                    cout << "\t\t\t==========================";
                    cout << "\n\n\t\t\tMAIN MENU";
                    cout << "\n\n\t\t\t==========================\n";
                    break;
                }
                else
```

```
                {
                    throw invalid_argument("Invalid login credentials");
                }
            }
            catch (const invalid_argument& e)
            {
                cout << "\n\n\n\t\t\t" << e.what();
            }
            catch (const exception& e)
            {
                cout << "\n\n\n\t\t\tAn error occurred: " << e.what();
            }
            getch();
            getch();
        }
    }
};

struct Book
{
    int id;
    string title;
    string author;
    bool isBorrowed;
    Book* next;
};

struct Member
{
    int id;
    string name;
    Book* borrowedBooks;
    int borrowedCount;
    Member* next;
};

class BookList
{
private:
    Book* head;
    const string bookDataFile = "books.txt";

public:
    BookList() : head(nullptr) {}

    void addBook(int id, string title, string author)
    {
        try
        {
            Book* newBook = new Book{id, title, author, false, nullptr};
            newBook->next = head;
            head = newBook;

            ofstream outFile(bookDataFile, ios::app);
            if (outFile.is_open())
            {
                outFile << id << "," << title << "," << author << endl;
                outFile.close();
            }
            else
            {
                cerr << "Unable to open file for writing." << endl;
            }
        }
        catch (const bad_alloc& e)
        {
            cerr << "Memory allocation failed: " << e.what() << endl;
        }
    }

    void displayBooks()
    {
        ifstream inFile(bookDataFile);
        if (inFile.is_open())
        {
            string line;
            while (getline(inFile, line))
            {
                stringstream ss(line);
                string idStr, title, author;
                getline(ss, idStr, ',');
                getline(ss, title, ',');
                getline(ss, author, ',');

```

```
169                 int id = stoi(idStr);
170
171                 cout << "Name: " << title << " , Author: " << author << " , ID: " << id << endl;
172             }
173             inFile.close();
174         }
175         else
176         {
177             cerr << "Unable to open file for reading." << endl;
178         }
179     }
180
181     Book* searchBook(int id)
182     {
183         Book* temp = head;
184         while (temp != nullptr)
185         {
186             if (temp->id == id)
187             {
188                 return temp;
189             }
190             temp = temp->next;
191         }
192         return nullptr;
193     }
194
195     void quickSort(Book* low, Book* high)
196     {
197         if (low == nullptr || high == nullptr || low == high || low->next == high) return;
198
199         Book* pivot = partition(low, high);
200         quickSort(low, pivot);
201         quickSort(pivot->next, high);
202     }
203
204     Book* partition(Book* low, Book* high)
205     {
206         string pivot = high->title;
207         Book* i = low->next;
208         Book* j = low;
209         while (i != high)
210         {
211             if (i->title < pivot)
212             {
213                 j = (j == nullptr) ? low : j->next;
214                 swap(j->title, i->title);
215                 swap(j->id, i->id);
216                 swap(j->author, i->author);
217                 swap(j->isBorrowed, i->isBorrowed);
218             }
219             i = i->next;
220         }
221         j = (j == nullptr) ? low : j->next;
222         swap(j->title, high->title);
223         swap(j->id, high->id);
224         swap(j->author, high->author);
225         swap(j->isBorrowed, high->isBorrowed);
226         return j;
227     }
228
229     void sortBooks()
230     {
231         if (head == nullptr) return;
232
233         Book* tail = head;
234         while (tail->next != nullptr)
235         {
236             tail = tail->next;
237         }
238
239         quickSort(head, tail);
240     }
241
242
243     Book* getHead()
244     {
245         return head;
246     }
247
248     Book* binarySearch(string title)
249     {
250         sortBooks();
251         Book* low = head;
252         Book* high = nullptr;
```

```
253
254             while (low != high)
255             {
256                 Book* mid = low;
257                 int count = 0;
258                 while (mid->next != high)
259                 {
260                     mid = mid->next;
261                     ++count;
262                 }
263                 for (int i = 0; i < count / 2; ++i)
264                 {
265                     mid = mid->next;
266                 }
267                 if (mid->title == title)
268                 {
269                     return mid;
270                 }
271
272                 else if (mid->title < title)
273                 {
274                     low = mid->next;
275                 }
276                 else
277                 {
278                     high = mid;
279                 }
280             }
281             return nullptr;
282         }
283     };
284
285     class MemberList
286     {
287     private:
288         Member* head;
289         const string memberDataFile = "members.txt";
290
291     public:
292         MemberList() : head(nullptr) {}
293
294         void addMember(int id, string name)
295         {
296             try
297             {
298                 Member* newMember = new Member{id, name, nullptr, 0, nullptr};
299                 newMember->next = head;
300                 head = newMember;
301
302                 ofstream outFile(memberDataFile, ios::app);
303                 if (outFile.is_open())
304                 {
305                     outFile << id << "," << name << endl;
306                     outFile.close();
307                 }
308                 else
309                 {
310                     cerr << "Unable to open file for writing." << endl;
311                 }
312             }
313             catch (const bad_alloc& e)
314             {
315                 cerr << "Memory allocation failed: " << e.what() << endl;
316             }
317         }
318         void displayMembers()
319         {
320             ifstream inFile(memberDataFile);
321             if (inFile.is_open())
322             {
323                 string line;
324                 while (getline(inFile, line))
325                 {
326                     stringstream ss(line);
327                     string idStr, name;
328                     getline(ss, idStr, ',');
329                     getline(ss, name, ',');
330
331                     int id = stoi(idStr);
332
333                     cout << "Name: " << name << " , ID: " << id << " , Borrowed books: ";
334
335                     Member* member = searchMember(id);
336                     if (member)
```

```
                {
                    cout << member->borrowedCount;
                }
                else
                {
                    cout << "0";
                }
                cout << endl;
            }
            inFile.close();
        }
        else
        {
            cerr << "Unable to open file for reading." << endl;
        }
    }


    Member* searchMember(int id)
    {
        Member* temp = head;
        while (temp != nullptr)
        {
            if (temp->id == id)
            {
                return temp;
            }
            temp = temp->next;
        }
        return nullptr;
    }

    void borrowBook(int memberId, Book* book)
    {
        Member* member = searchMember(memberId);
        if (member && book && !book->isBorrowed)
        {
            book->isBorrowed = true;
            Book* borrowedBook = new Book{book->id, book->title, book->author,
book->isBorrowed, member->borrowedBooks};
            member->borrowedBooks = borrowedBook;
            member->borrowedCount++;
            cout << "Book borrowed successfully." << endl;
        }
        else
        {
            cout << "Error: Member not found or book already borrowed." << endl;
        }
    }

    void returnBook(int memberId, int bookId)
    {
        Member* member = searchMember(memberId);
        if (!member)
        {
            cout << "Member not found." << endl;
            return;
        }

        Book* prev = nullptr;
        Book* curr = member->borrowedBooks;
        while (curr != nullptr && curr->id != bookId)
        {
            prev = curr;
            curr = curr->next;
        }

        if (curr != nullptr)
        {
            if (prev != nullptr)
            {
                prev->next = curr->next;
            }
            else
            {
                member->borrowedBooks = curr->next;
            }

            delete curr;
            member->borrowedCount--;
            cout << "Book returned successfully." << endl;
        }
        else
        {
```

```
            cout << "Book not found in member's borrowed books." << endl;
        }
    }

    Member* bfsSearchByID(int id)
    {
        if (!head)
            return nullptr;

        queue<Member*> q;
        unordered_map<int, Member*> visited;

        q.push(head);
        visited[head->id] = head;

        while (!q.empty())
        {
            Member* current = q.front();
            q.pop();

            if (current->id == id)
                return current;

            Member* neighbor = current->next;
            while (neighbor)
            {
                if (visited.find(neighbor->id) == visited.end())
                {
                    q.push(neighbor);
                    visited[neighbor->id] = neighbor;
                }
                neighbor = neighbor->next;
            }
        }

        return nullptr;
    }
};

class LibraryGraph
{
private:
    unordered_map<int, unordered_map<int, int>> adjacencyList;

public:

    void addEdge(int memberId1, int memberId2, int weight)
    {
        adjacencyList[memberId1][memberId2] = weight;
        adjacencyList[memberId2][memberId1] = weight;
    }

    vector<int> findShortestPath(int memberId1, int memberId2)
    {
        unordered_map<int, int> distance;
        unordered_map<int, int> parent;
        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

        for (auto it = adjacencyList.begin(); it != adjacencyList.end(); ++it)
        {
            int member = it->first;
            distance[member] = INT_MAX;
            parent[member] = -1;
        }

        distance[memberId1] = 0;
        pq.push({0, memberId1});

        while (!pq.empty())
        {
            int u = pq.top().second;
            pq.pop();

            for (auto it = adjacencyList[u].begin(); it != adjacencyList[u].end(); ++it)
            {
                int v = it->first;
                int weight = it->second;

                if (distance[v] > distance[u] + weight)
                {
                    distance[v] = distance[u] + weight;
                    parent[v] = u;
                    pq.push({distance[v], v});
                }
```

```
            }
        }

        vector<int> shortestPath;
        int current = memberId2;
        while (current != -1)
        {
            shortestPath.push_back(current);
            current = parent[current];
        }
        reverse(shortestPath.begin(), shortestPath.end());

        return shortestPath;
    }
};

void printMenu()
{
    cout << "\n1. Add Book" << endl;
    cout << "2. Add Member" << endl;
    cout << "3. Display Books" << endl;
    cout << "4. Display Members" << endl;
    cout << "5. Borrow Book" << endl;
    cout << "6. Return Book" << endl;
    cout << "7. Search Book by Title" << endl;
    cout << "8. Search Member by ID" << endl;
    cout << "9. Find Shortest Path Between Members" << endl;
    cout << "10. Exit" << endl;
    cout << "Enter your choice: ";
}

int main()
{
    LoginSystem loginSystem;
    loginSystem.showWelcomeScreen();
    loginSystem.authenticateUser();

    BookList books;
    MemberList members;
    LibraryGraph libraryGraph;

    libraryGraph.addEdge(1, 2, 1);
    libraryGraph.addEdge(1, 3, 1);
    libraryGraph.addEdge(2, 4, 1);
    libraryGraph.addEdge(3, 4, 1);
    libraryGraph.addEdge(4, 5, 1);

    int choice;
    do
    {
        printMenu();
        cin >> choice;
        switch (choice)
        {
        case 1:
        {
            int id;
            string title, author;
            cout << "\nEnter Book ID: ";
            cin >> id;
            cout << "Enter Book Title: ";
            cin.ignore();
            getline(cin, title);
            cout << "Enter Book Author: ";
            getline(cin, author);
            books.addBook(id, title, author);
            cout << "Book added successfully.\n" << endl;
            break;
        }
        case 2:
        {
            int id;
            string name;
            cout << "\nEnter Member ID: ";
            cin >> id;
            cout << "Enter Member Name: ";
            cin.ignore();
            getline(cin, name);
            members.addMember(id, name);
            cout << "Member added successfully.\n" << endl;
            break;
        }
        case 3:
        {
```

```
            cout << "\nList of Books:\n" << endl;
            books.displayBooks();
            cout << "\n\n";
            break;
        }
        case 4:
        {
            cout << "\nMembers of the library:\n" << endl;
            members.displayMembers();
            cout << "\n\n";
            break;
        }
        case 5:
        {
            int memberId, bookId;
            cout << "\nEnter Member ID: ";
            cin >> memberId;
            cout << "Enter Book ID: ";
            cin >> bookId;
            Book* book = books.searchBook(bookId);
            if (book)
            {
                members.borrowBook(memberId, book);
            }
            else
            {
                cout << "Book not found.\n" << endl;
            }
            break;
        }
        case 6:
        {
            int memberId, bookId;
            cout << "\nEnter Member ID: ";
            cin >> memberId;
            cout << "Enter Book ID: ";
            cin >> bookId;
            members.returnBook(memberId, bookId);
            break;
        }
        case 7:
        {
            string title;
            cout << "\nEnter Book Title: ";
            cin.ignore();
            getline(cin, title);
            Book* book = books.binarySearch(title);
            if (book)
            {
                cout << "Book found: ID=" << book->id << ", Title=" << book->title << ",
Author=" << book->author << endl;
            }
            else
            {
                cout << "Book not found.\n" << endl;
            }
            break;
        }
        case 8:
        {
            int id;
            cout << "\nEnter Member ID to search: ";
            cin >> id;
            Member* member = members.searchMember(id);
            if (member)
            {
                cout << "Member found: ID=" << member->id << ", Name=" << member->name << ",
Borrowed Books=" << member->borrowedCount << endl;
            }
            else
            {
                cout << "Member not found.\n" << endl;
            }
            break;
        }
        case 9:
        {
            int memberId1, memberId2;
            cout << "\nEnter ID of first member: ";
            cin >> memberId1;
            cout << "Enter ID of second member: ";
            cin >> memberId2;

            vector<int> shortestPath = libraryGraph.findShortestPath(memberId1, memberId2);
```

```
670
671                 if (shortestPath.empty())
672                 {
673                     cout << "There is no path between the given members." << endl;
674                 }
675                 else
676                 {
677                     cout << "Shortest path between members: ";
678                     for (int memberId : shortestPath)
679                     {
680                         cout << memberId << " ";
681                     }
682                     cout << endl;
683                 }
684                 break;
685             }
686             case 10:
687             {
688                 cout << "Exiting program.\n";
689                 break;
690             }
691             default:
692             {
693                 cout << "Invalid choice. Please enter a number between 1 and 10.\n";
694                 break;
695             }
696             }
697         }
698         while (choice != 10);
699
700         return 0;
701     }
```