A Platform for Smart Book Search using Structured Programming Approach

Lekiya Kasepa, Dr. Aravind Mohan
Department of Computer Science, McMurry University, Abilene, TX 79697

Abstract

In today's digital era, we have access to a lot of information available on the web about books, but there are limitations to easily and quickly finding the books that we need. Existing recommendation systems are often inadequate, leading to frustrating user experience when searching for the right books. This makes internet users struggle to find the books that they want to read, and authors get pushed under the radar because of the limitations of the current recommendation systems. The Smart Book Search system (SBS) addresses these challenges by allowing users to effectively search for their book of choice by using various methods, including title, author, keyword, or recommendation search. By enhancing the user experience, the SBS also improves author visibility. Utilizing our algorithm, the SBS can efficiently find books that match user interest based on search history. Specifically, it recommends the top five authors by analyzing the user's previous searches and inputs. To validate the effectiveness of our system, we conducted a series of experiments that demonstrated its accuracy in finding and ranking books based on user search criteria. Additionally, we used C, a structured programming language, to efficiently implement our algorithm and develop our system based on the dataset we collected from Kaggle (Goodreads).

Introduction

Reading plays a big role in expanding user knowledge and improving literacy. Even though the internet provides access to a vast digital library of books, finding the right one can still be a tough task for many readers. If a reader keeps getting the wrong suggestions every time they search for books, it might push them towards losing interest in reading, which is a disadvantage for both readers and authors. Popular platforms like Goodreads, Library Thing, and Book Browse try to help users search for books, but they often fall short. Many of these sites have limited search options or only provide basic recommendations that don't connect to what users want to read. This can lead to frustration, as users may end up seeing lists of books that are not related to their interests. For instance, if someone is looking for a fantasy novel but ends up receiving suggestions for unrelated genres, the experience can be disappointing. On top of that, many systems don't keep track of the books a user has previously searched for, so they can't provide better recommendations based on past choices. The Smart Book Search system (SBS) aims to address these issues by offering a more user-friendly way to discover books. With SBS, readers can search using keywords, author names, or even partial titles, making it much easier to find what they're looking for. For example, if a student wants to find a history book titled, American History by Alan Brinkley, they can easily search using the title or just keywords like "American" and "History." The system quickly provides the right book without dragging the user through a long process. Additionally, SBS lets users find other books by the same author with just a simple search for the author's name. This means that if the student is interested in more works by Alan Brinkley, they will get a list right away. We implemented the SBS system using C, a structured programming language and conducted experiments to verify the efficiency and accuracy of the tool.

Methodology

The Smart Book Search system (SBS) was implemented in C and uses three CSV files: Books.csv (Table 1) containing titles, authors, and page counts, Users.csv with user profile like username, password and full name, and History.csv (Table 2), which stores user search activity. First, users are prompted to login using their username and password. After a successful log in, users are later prompted to select a search type (choice) which can either be title, author, keyword, or page count, and enter a value afterwards. The tool then scans the Books.csv for matches, displays the matched results, and stores them in the History.csv for future recommendations. For example, when Mark101 searched for keyword atlas, all books related to the keyword were retrieved from the books file (Table 1) and later shown under recommendations (Table 3). The SBS algorithm (Table 4) was implemented within our tool using C Programming language. Our algorithm first accepts the User Profile, Search criteria (Choice), Books, and Search History as input. Next, the algorithm performs either search by attribute or recommendation to generate the result R which is a set of tuples that includes the titles and author names.

Table 1: Books

Title	Author	Page Count
Layer Cake	J.J Connolly	344
Basin and Range	John McPhee	240
Atlas	David Mitchell	529
Atlas Shrugged	Leonard Peikoff	1168
We the Living	Ayn Rand	464
Angels Fall	Nora Roberts	391
Absolute C++	Walter J. Savitch	943
Valley of Silence	Nora Roberts	318

Table 2: User Search History

User Id	Value	Attribute
Chris10101	Layer Cake	Title
Chris10101	943	Page Count
Luke608	John McPhee	Author
Luke608	We the Living	Title
Mark101	Atlas	Keyword
Mark 101	Nora Roberts	Author

Table 3: Recommendations

User Id	Title	Author
Chris10101	Layer Cake	J.J. Connolly
Chris10101	Absolute C++	Walter J. Savitch
Luke608	Basin and Range	John McPhee
Luke608	We the Living	Ayn Rand
Mark101	Atlas	David Mitchell
Mark101	Atlas Shrugged	Leonard Peikoff
Mark101	Angels Fall	Nora Roberts
Mark101	Valley of Silence	Nora Roberts

Table 4: Smart Book Search Algorithm

```
SBS (User Profile P, Choice C, Books B, Search History S
Result, R <Title, Author> = empty
Status = Validate(P)
If (Status == Success)
  Switch (C)
    Case Title, Author, PageCount, Keyword:
      prompt(searchValue)
      While (B.Item != NULL)
        book = B.nextItem()
        If (book.getAttribute(C) == searchValue)
          R.add(book.title, book.author)
          S.add(R)
        End If
      End While
    Case Recommendation:
      Authors = S.getAuthors(P)
      For author in Authors
        R.add(B.getTitles(author), author)
      End For
  End Switch
End If
return R
```

Experiments

To help visualize the efficiency of the SBS tool, we used three charts to know the time, frequency and number of books the tool can recommend to users based of their search history. Figure 1 shows that as the number of books increases, the average response time rises, particularly for keyword and page count searches, while author or title searches remain fast. Figure 2 indicates that as users search more, recommendations for top authors increases, but this varies among users due to personal preferences. Figure 3 reveals that more searches lead to an increase in books recommended, that is more relevant to the users. Overall, our system increases the response time and enhance the book recommendation process.

Figure 1: Average Response Time Distribution

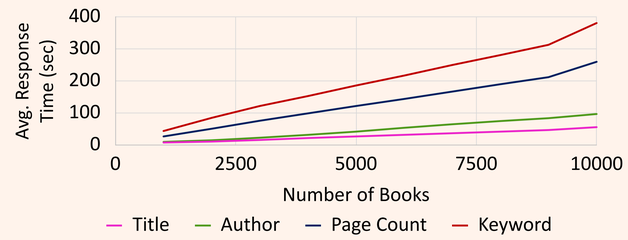


Figure 2: Total Frequency of 5 Recommended Authors

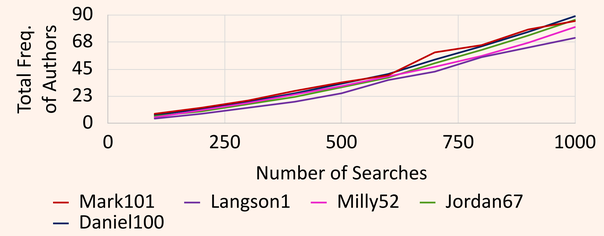
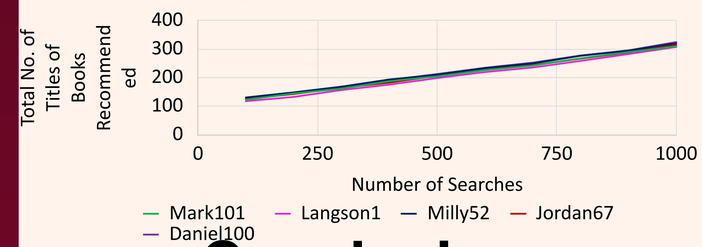


Figure 3: Total No. Titles of Books Recommended



Conclusion

The Smart Book Search system improves the book search experience of users by accurately finding and ranking books based on user search criteria and user interest. We implemented our system and tested it with multiple use cases to validate that the tool can both successfully search using different attributes like title, author, page count, and keywords, and identify the top five authors and recommend books that are relevant to the user and help the users discover more books they might be interested in. We conducted a series of experiments to show that our tool can search and recommend books efficiently and accurately.