COP3530 Project 3 Report

Administrative

Team Name:
JDK's Book Search Engine

Team Members + Github user names:
Kent Phipps - https://github.com/StrongKs
Jack Crew - https://github.com/realjackcrew
Dane Dickerson - https://github.com/Dane314pizza

Link to GitHub repo: https://github.com/StrongKs/DSA_Project3
Link to Video demo: https://youtu.be/gbegRLFNhdw

Extended and Refined Proposal

Problem:
The problem that the project "Book Search Engine" aims to solve is the difficulty users face in finding specific books and discovering new titles in a vast and ever-growing pool of available literature.

Motivation:
The project seeks to create a centralized and efficient search engine that simplifies the process of locating and accessing detailed information about books.

Features implemented:
Searching using either custom made hashmap or trie which parses GoogleBookAPIDataSet. Can search for books by either book's title or book's authors.

Description of data:
Has 68,000 book entries that describe book titles, identification number, authors, categories, averageRating, maturityRating, publishedDate, and pageCount. This sums the dataset to about 544,000 data points that was parsed to construct the hash or trie data structure.

Tools/Languages/APIs/Libraries used:

Collaboration - GitHub

Language - C++

Data set - GoogleBookAPIDataset.txt (tab separated text)

- Used tabs over CSV because some of the data fields like titles and authors use commons which made parsing difficult.

Libraries - SFML

- Simple and easy to use front end interface that allows for text field lookup with toggle selection of hashmap or trie for data structure and authors or titles for search up field.
- A new window will pop up with results of all the books with text field specification along with data structure set up and search function duration times.

Algorithms implemented:

Trie

- 26-ary nodes to account for every character. When a book is added to a node there is a vector of book pointers where it is pushed back into.
- After construction of Trie, it can call the parsing function on the GoogleBookAPIDataset.txt to insert all books into the data structure. The time duration to add book data into Trie is recorded as an attribute and displayed.
- To search for a book with a key, based on the parsing specification, it can look by title or author. It iterates through every char in the key and follows the nodes until it gets to the last char in the key. It then performs BFS from that point on to find all books with the key being the prefix. The time duration to search and retrieve a book is recorded as attribute and displayed

Hashmap

- Implemented as a vector of string, book vector pairs.
- Parses in input from the same .txt file, updating all the values of each book struct. Key is inputted as either title or author depending on which parameter is selected for search. Time is also recorded for analysis.
- Search (retrieve function) is conducted by the hash value of the key. All matching keys are returned (in case of several entries of the same book title or author). Time is also recorded for analysis.

Additional Data Structures/Algorithms used:

No other significant data structures or algorithms were used.

Distribution of Responsibility and Roles:
Kent Phipps - Trie API implementation
Jack Crew - Hashmap API implementation
Dane Dickerson - SFML front end, integrating both APIs

Analysis

Changes Made After Proposal:

- Simplified user interface by reducing search parameters such as genre and publishing year as book shelf.
- Made a new window for search results. This allows for simpler user experience having all results apart from the search engine page.
- Instead of using GoogleBookAPI directly from Google we fund a simple .txt that made it easier to implement for parsing into data structures(hash and trie).

Time Complexity Analysis:
Unless otherwise specified, ‘n’ is defined as the number of books in the data set.

- **Trie**
    - Parsing - O(n)
        - In order to retrieve all books from the data set, we must iterate over every book(row) in order to collect their attributes: title, author, publish data, etc.
    - Searching - O(n)
        - In the worst case where the user enters nothing into the search text field then the Trie will perform a BFS from the root nodes and iterate through every possible book and return it.
- **Hashmap**
    - Parsing - O(n)
        - In order to retrieve all books from the data set, we must iterate over every book(row) in order to collect their attributes: title, author, publish data, etc.
    - Searching - O(n)
        - In the worst case, the user conducts a search in a dataset where all of the titles or all of the authors are the same, which requires the program to iterate through and return every value.

Reflection

Overall Experience:
It was a very educational experience to be able to collaborate with highly effective peers that are able to contribute to a project. None of us had worked extensively with either of these data structures before, so it was exciting to see how much we were learning as the development process went along. We also enjoyed working with SFML again in a much less intense project than Minesweeper.

Challenges:
Using the GoogleBooksAPI was difficult to implement. After discussing with a TA, we determined it would be simpler and more effective to use a text file that incorporated all the data from the API. We found the text file through Kaggle.

Changes we would make to the project/workflow:
Finalizing UI/UX features earlier would have allowed us to be able to test and develop the graphical interface better. In addition, starting earlier would have allowed more time to beautify the front end and implement more fun features in the back end.

What each member has learned:
Kent Phipps - How to create a simple API for UX/UI designers to integrate into the front end interface.
Jack Crew - How to contribute simultaneously with others through GitHub and parse values in a tab separated data set.
Dane Dickerson - How to collaborate with others through GitHub and implement others APIs into a graphical interface.

References

- GoogleBooksAPIDataset - Kaggle Data Set
  https://www.kaggle.com/datasets/chameleonastronaut/google-books-api-dataset?resource=download
- Trie Data structure - Geeks for Geeks
  https://www.geeksforgeeks.org/trie-insert-and-search/
- Hash Map definition and complexity - baeldung
  https://www.baeldung.com/cs/hash-tables
- Hash Function - Geeks for Geeks
  https://www.geeksforgeeks.org/hash-functions-and-list-types-of-hash-functions