CSC212 Project

Time Complexity

Faisal Almashouq 444105697

Saud Alkathery 444102094

Table of Contents:

Abstract:

Extracting, organizing, and querying data is the cornerstone of every business in this day and age. This project involved these steps to ensure usability of such databases. The program takes in data from the .csv file and sorts it into three different structures, indexing, reverse indexing, and reverse indexing via BST. Afterwards comes querying through boolean operations (AND, OR) and ranking. Finally, we developed a simple UI menu that allows for querying through different indexing methods. Inferring the difference between linked list and binary search tree implementations regarding time complexity, space complexity, and ordering.

Introduction:

Developing, managing, and querying spreadsheet data is a crucial part of business, as such data can be difficult to read. Developing a query simplifies data, ensuring only required data is available for view. In this project, we are required to create an efficient indexing system for querying spreadsheet data. To do so, we should export data from the spreadsheet into java, use data structures for indexing and ordering, then allow querying by boolean methods and ranking retrieval.

Background:

Indexing refers to setting an index for each object. It is a way to manage data and store it in a simpler manner. Inverse indexing implies that each part of an object contains its index. It is extremely useful if some data repeats itself. They are implemented through Linked Lists and Binary Search Trees **(BST)**, which are two data structures that contain data and help navigate and query between them. Querying is the process of searching for data through filtering systems. Boolean retrieval is a filtering system for query that filters results based on input, and the boolean operator used (AND, OR). Ranking retrieval retrieves data and ranks its frequency.

Design and Implementation:

Designing the program went through different stages. The program needed to load in the spreadsheet and stopwords, then indexed through different types of indexing, then developing the boolean and ranking retrievals. Finally, it is all fixed together in the user interface menu.

Loading Spreadsheet:

Class **FileDataEntry** allows the .csv spreadsheet file and the stop words file to be an input. It loads in IDs and its corresponding string text. The stop words are then filtered. Lastly, all data is sorted into the three types of indexing, which are normal indexing, inverted indexing, and inverted indexing using BST.

Indexing:

From the previous step, all words are inserted into three data structures. One data structure is a linked list for normal indexing rows. It isn't the most efficient way to store data, as it uses too much of the memory, as well as the time complexity takes longer for lookup. The other method is called inverse indexing, which saves words once, then each repetition of a word becomes an index. A lot more efficient than normal indexing, but still isn't optimal. The final method is reverse indexing via BST. This allows for quick searching and insertion. It is the most optimal solution for the boolean & ranking retrievals. Classes **Rows**, **InvertedIndex**, and **InvertedIndexBST** are responsible for each method respectively.

Boolean & Ranking:

Boolean retrieval refers to retrieving the row ID of two words/tokens based on the operator's constrictions. The AND operator requires words to have the same row ID, if so, it returns that row. On the other hand, the OR operator retrieves all the row IDs of both words. Ranking retrieval refers to rank each word by its frequency, with precedence from lower to higher row ID. Both retrieval methods allow for quick filtration of query.

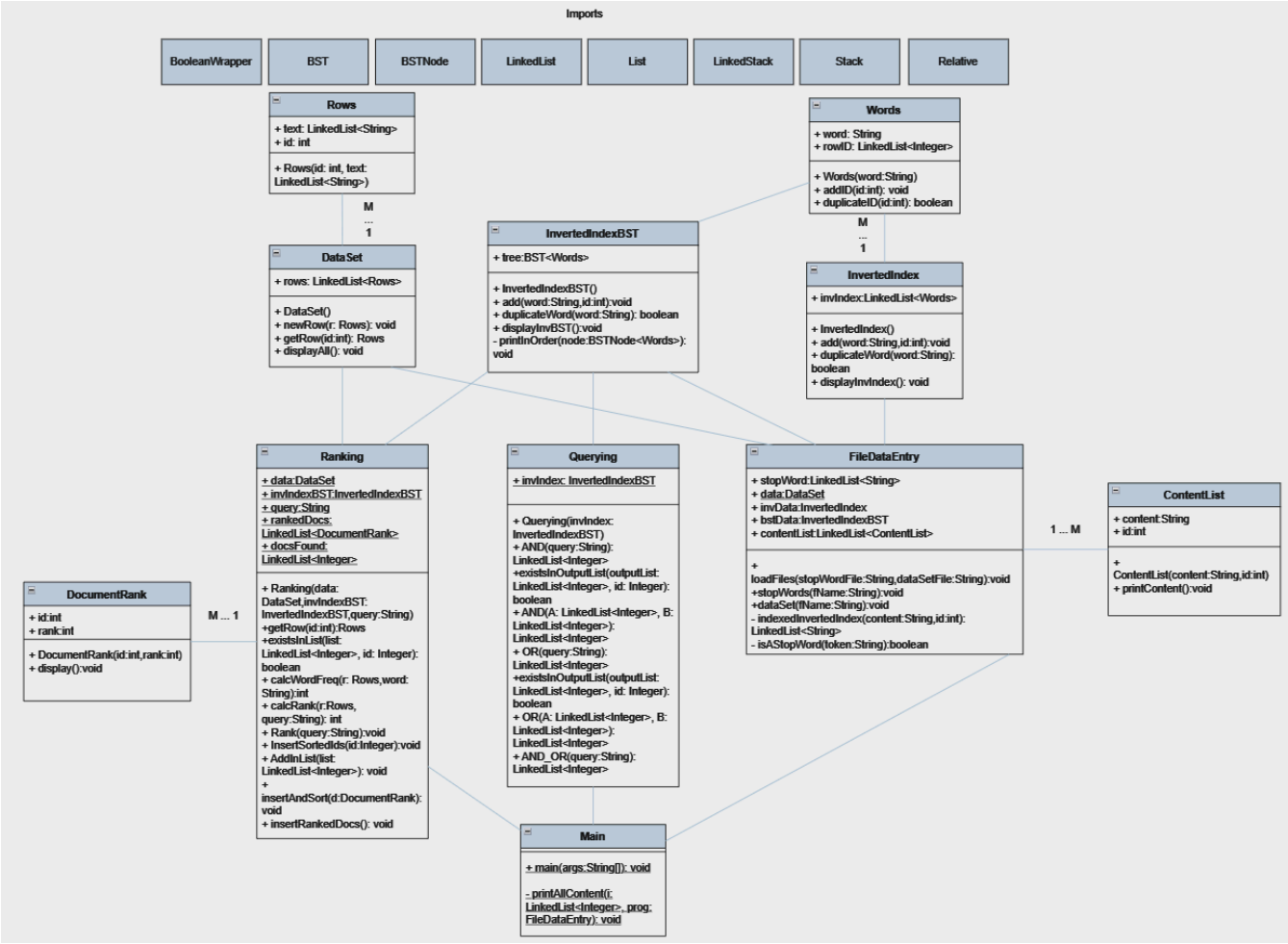
Query Design:

Utilizing the aforementioned retrieval methods, the query is executed from the simple user interface menu. It links the indexing methods with filtration by retrievals, then displays it on the console.

User Interface:

Managed in the **Main** class, it allows users to pick the query option. It operates by a do-while loop that has 3 main options, which are display all, query by boolean retrieval, and query by ranking retrieval. Display all displays all data in the indexing types.

UML:



A UML diagram is a visual representation of the program, noting only methods, instance variables, and relations between classes. Note that the above classes are pre-made imported classes.

Analysis:

Analyzing index, inverted index via linked list, and inverted index via BST performances indicates expanding each method's advantages and flaws.

Indexing allows for a simple implementation of data structuring which saves each row ID and the corresponding text split word by word in another linked list. It has an average case time complexity insertion of O(n) and search by row ID of O(n), but searching for a specific word has $O(n^2)$ or $O(n \times m)$, as it has to move through each row's text linked list. It has a space complexity of $O(n^2)$ or $O(n \times m)$. So, it is simple, but inefficient for large datasets. Ordering is preserved.

Inverted indexing takes in each word once, and with each repetition it adds the new row ID. It has an average case time complexity insertion of O(n). Searching in all cases requires O(n). Space complexity of such implementation would require O(n). Order of words is almost preserved, as duplicates are discarded and only the row ID is added.

Lastly is the inverted indexing using BST, which inserts words and is sorted by the precedence of the letters with regards to ASCII convention. BSTs are split into two types, balanced and unbalanced. It is expected that data isn't necessarily balanced, as words are inserted and not ordered. Best case of insertion and search is O(log n) and the average/worst case is O(n). It also requires O(n) space complexity. Order of words isn't preserved.

This proves that in best cases, which is when the tree is balanced, BSTs are better than linked lists, and in worst/average cases, they both have a O(n) time complexity.

Discussion:

Strengths:

Although the idea of the project was new to us, the implementation was simpler than expected, as we have had prior training with regards to Java. Our coding team was credible, so this project was exactly at our level. Importing data structures was simple. Unifying the work was not an issue, as we took turns in working. Even when we were working concurrently, we just copied the added work to each other.

Limitations:

One of the group members had an unforeseen issue, which sadly forced him to drop the term. Importing .csv files into Java was challenging, as we did not know how to read data from the spreadsheet directly. Instructions of the deliverables were unclear, as we had to second guess ourselves often, and inquire other students of their approach.

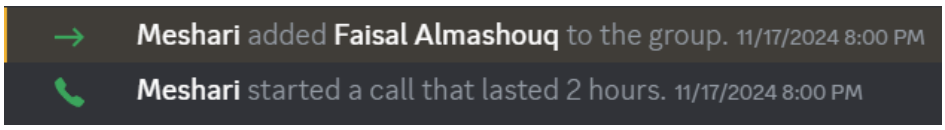
Conclusion:

This project explored the process of creating a fully functional query program via indexing, inverted indexing with linked lists, and inverted indexing with BSTs to efficiently query data from a spreadsheet. Each method has distinct advantages and limitations, with BST proving to be the most optimized for query with regards to the spreadsheet. Although indexing provided simplicity, it had redundancy with repetitive words and worse space and time complexities. Moreover, inverted indexing improved efficiency by reducing redundancy, which is helpful for repetitive data, however it isn't optimized for large scale querying. These findings demonstrate the importance of choosing the right data structure based on requirements of the querying process. Overall, this project highlights practical applications of indexing methods and retrievals in data management.

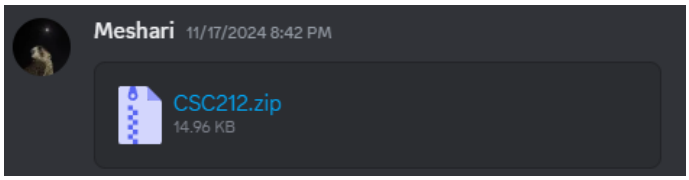
Appendix:

We were not familiar with github, as we did not know the purpose of it, so we decided that we should work on eclipse. With each member working alone then transfering the files between us with our progress. At the end, we collected our code and uploaded it into github to submit. However, as we now understand the main purpose of it, which is to verify our commitment on this project by the dates, we realized that we should've started the project on github originally. The following is a solution to verify our progress and to show our commitment:

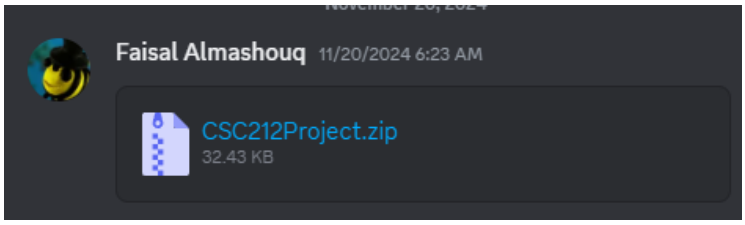
- We created the group on 17/11/2024, and joined a call together that evening at 8:00pm.



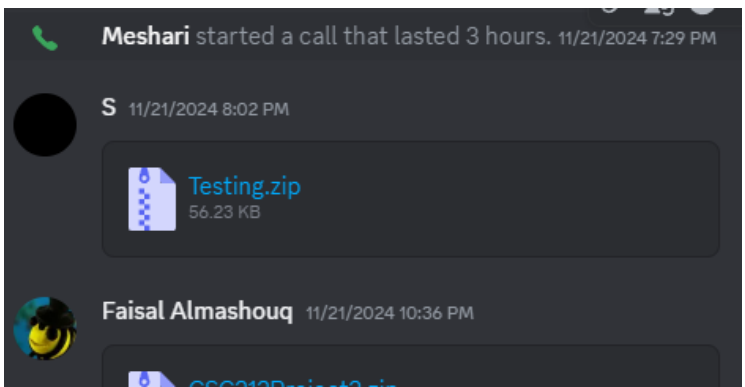
- We divided the work evenly between us, ensuring collaboration.
- We had a quick unified draft between us later that day.



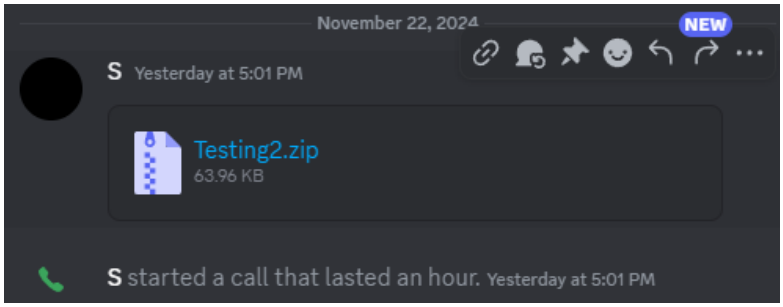
- Each person worked alone for 3 days, then we sent progress on 20/11/2024.



- The next day, we all joined in on a call, and had 2 more progression files.



- The next day, we sent our progress yet again, then joined a call to review.



- The last draft was unified at 6:11pm on 22/11/2024.