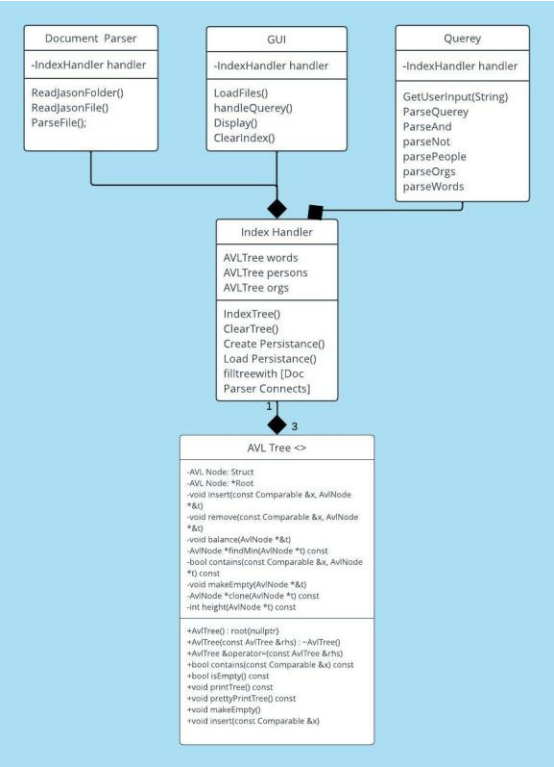
High-level Design of Solution

- UML class diagram, etc.



- High-level pseudo code for each component.

GUI:

If the user selects a choice 1 - 5

The user will be shown the action preformed.

else

The interface will prompt you to continue or exit.

endif

Document parser:

The JSON files will be read in.

If the file cannot be found it will prompt the user.

The file will be parsed into the required sections.

The files will be turned into individual strings the program can use

Strings will be sent to the index to be saved into the trees.

The JSON files will close.

Index:

Trees are created for Words, Persons, and Orgs

Files are then created for each Tree for reading purposes

If the GUI selects to Write the index

Files will be read to the tree.

If the GUI selects to read the index

Files will be read to the tree.

- Your Timeline to Finish the Project

Activity	Planned Completion Date	Actual Completion Date	Time Spent (Hours)
Compete initial UML and class descriptions	11/4/2022	11/4/2022	3 hours
Complete File IO and Data Transformation classes	11/09/2022	11/09/2022	16 Hours
Complete AVL tree class (make suitable for inverted file index)	11/13/2022	11/13/2022	16 Hours
Persistent Index	11/15/2022	11/18/2022	11 Hours
Complete query processor and user interface	11/25/2022	12/3/2022	19 Hours
Complete relevancy rankings	11/25/2022	11/27/2022	9 Hours
Finalize code	11/28/2022	11/27/2022	22 Hours
Complete final testing	12/1/2022	12/4/2022	6 Hours
Finalize documentation	12/1/2022	12/4/2022	8 Hours

Used Data Structures and explanation of what data structures are used where. For each, state - time and space complexity (Big-Oh), - why it is the appropriate data structure and what other data structure could have been used.

a

AVL Tree: Handles our strings that come from the document parser that split into words, orgs, persons. The AVL tree allows for quick access to stored items due to its unique structure and ability to only look at half of the data in the tree. Maintaining a balance ensures that the node heights never differ by more than 1. We were able to further optimize our AVL Tree by adding iterative functions for insert, store, and find. The AVL tree was really the only option for this task as it provided the best option for a fast look-up of terms. The AVL tree has a Big-Oh notation of O(log N).

Stack: We used a stack within the AVL tree functions for insert and store tree. A stack was particularly useful for these functions since we needed a container that was best suited for a last-in-first out operation. For the insert function, a stack was used to store nodes that needed to be balanced on the way back up the tree once a new node was inserted. For the store tree function a stack also worked well for storing each node that we passed while doing in-order traversal. The top of the stack was popped and placed into the index vector which was used to print out our persistent files. A vector could have also been used for this, but the stack seemed better suited for LIFO than a vector. The Big-Oh notation for our stacks is O(N).

String: Turns the text we read from files into items we can work on in the program to be parsed and separated into corresponding trees.

Vector: We use vectors to hold tokenized words that will be split into single strings later and we also use them throughout the program as our go-to container when we need to store results from our queries. A vector is also used to store all the words entered in a user search. Vectors are a natural data structure to use due to their utility and ease of use. We substituted maps for vectors whenever possible for improved performance. The Big-Oh for our vectors in O(N).

Unordered Map: We used an unordered map for stop words, the document index and within our tree nodes to link our documents with the number of times a specific word appears in that document. This allowed for a critical linkage between the words, the documents they appeared in, and the number of times they appeared. We could have used vectors for this linkage, but the map provided improved performance. The Big-Oh notation for unordered maps is O(1).

User Documentation

- How to use the software.

The Gui is what holds the program together and links each part to one another. Initially, the menu prompts you to select from 5 different options. Those options are; creating an index, writing and reading to persistence, entering a query, and output statistics. The GUI is what allows us to connect the entire program together

Our AVL Tree starting point was the tree provided by professor Hahsler. The tree provided used recursive functions which is an effective way to minimize the amount of code, but it does not have the best performance. To improve performance, we decided to create iterative functions whenever possible, especially for the frequently used functions (e.g., insert, find, etc.). We also added a remove function for additional functionality, but it is a recursive implementation.

The tree can be loaded in two primary ways: the document parser and the persistent files. The document parser method takes a long time to go through numerous documents, so it is not the best method for performance, but this is unavoidable when starting with new documents. The document parser passes in words, people, and organizations as well as document numbers which

are eventually stored in the tree nodes after the word class creates word objects consisting of words, document numbers and occurrences. The persistent data option stores information loaded into a tree in text files so that a tree can be quickly re-populated with that data. Using persistent data improves the loading time.

When a search term is entered, the tree is quickly scanned for the word/person/organization. Once the tree finds the word, it returns the document map showing the documents and occurrences for that word, and this information is used by the query processor to provide the most relevant search results.

The primary function of the Document Parser is to read each document and turn the information into an easy-to-use format for the tree.

The document parser will take the document's text and read them in. After the document is read in, there will be a “tokens” vector that is created. This vector will be populated by having all the text tokenized into individual strings. Before the words are pushed into the “tokens” vector the words are compared against the stop word text file. This helps reduce the workload of the trees and only keeps important words needed for searches. Once this is completed the document parser will help organize the words into “Text,” “Persons”, and “Organizations” From here the trees will be sent to the index handler.

The primary function of the index is to organize all the key information from the documents (keywords, people, and organizations) and load it into its respective AVL tree for quick access. Additionally, the index can store all the information within the AVL trees into output files (persistent data) which can then be quickly loaded into AVL trees. The index consists of two main classes within our program: the Word class and the Index class. The primary data structures used for the index are the AVL tree and the unordered map.

The Word class creates the word objects which are then loaded into AVL trees. All word objects consist of a word, person, or organization, a document number, and the number of occurrences a word, person, or organization appears in that document. For comparing word objects correctly, the word class requires overloaded comparison operators which compare the string element of the word objects, so the program can efficiently store and search for words within the trees. The document number and occurrences are stored within an unordered map, with the document number being the key value. This allows for efficient searching when looking for relevant documents for any word. Additionally, as the tree is being loaded, the insertDoc function within the Word class is called which checks occurrences for each word for a particular document. If the function sees a new document where the word is not present, it will create a new pair within the unordered map listing the document number and an occurrence of 1. If the function sees an additional occurrence of the same word in the same document, it will simply increment the occurrence value for that respective document. The Word class is also responsible for the output stream that organizes the information within word objects, so it can be easily output into persistent data files.

The Index class has two primary functions: one function is responsible for loading the tree with word objects and the other function is responsible for generating the persistent data files. The loadTree function is called by the file reader class which passes the following parameters: word, person, or organization, document number, character value indicating word, person, or organization, and an AVL tree. Based on this information, the function will create a tree node for new words and load it into its appropriate tree. For repeat words, it calls the insertDoc function from the word class which will help pair the document number with an accurate number of occurrences. Finally, the persistent data files are populated with the generateFiles function. This function calls the storeTree function from the AVLTree class which takes all the information contained within a tree and then stores it in a vector. The contents of this vector are then used to populate the persistent data files in an organized manner, so it can easily be read to quickly repopulate the trees.

The query processor is responsible for initiating the searches within the trees and providing relevant search results for the user. The query processor can perform three distinct types of searches:

1. Simple queries – these are one-word/people/organization searches which require the query processor to find the particular word and return the relevant documents. The entire document map for a word is returned and then loaded into a vector which is then sorted in descending order by occurrences. The top 15 results are then displayed to the user.
2. AND queries – these are multiple word searches where the query processor looks for an intersection of the documents between the multiple words. Once this intersection is determined the sum of the occurrences of the words for a document are considered. The documents (top 15) with highest occurrences are then displayed to the user.
3. Mixed queries – these queries are like AND queries in that they involve multiple words, but they differ due to the different types of terms. For example, there are word searches mixed with people searches, so the query processor needs to determine the appropriate tree to search based on the information entered by the user. This can also be done with organizations as well. A similar process is followed to determine and display the most relevant results.

- Some example queries with the results.

```
Enter your choice(1-6):4
you have selected Enter a query
Please enter search term(s):
money
Found 1 document(s) containing search term 'money'

Top 15 search results by relevancy:
1.
Eaton Vance Closed-End Funds Release Estimated Sources Of Distributions
2018-02-28T18:54:00.000+02:00
cnbc.com

Enter document id to display text: 1
BOSTON, Feb. 28, 2018 /PRNewswire/ -- The Eaton Vance closed-end funds listed below released today the estimated sou
rces of their February distributions (each a "Fund"). This press release is issued as required by the Funds' managed
 distribution plan (Plan) and an exemptive order received from the U.S. Securities and Exchange Commission. The Boar
d of Trustees has approved the implementation of the Plan to make monthly or quarterly, as noted below, cash distrib
utions to common shareholders, stated in terms of a fixed amount per common share. This information is sent to you f
or informational purposes only and is an estimate of the sources of the February distribution. It is not determinati
ve of the tax character of a Fund's distributions for the 2018 calendar year. Shareholders should note that each Fun
d's total regular distribution amount is subject to change as a result of market conditions or other factors.
IMPORTANT DISCLOSURE : You should not draw any conclusions about each Fund's investment performance from the amount
of this distribution or from the terms of each Fund's Plan. Each Fund estimates that it has distributed more than it
s income and net realized capital gains; therefore, a portion of your distribution may be a return of capital. A ret
urn of capital may occur for example, when some or all of the money that you invested in each Fund is paid back to y
ou. A return of capital distribution does not necessarily reflect each Fund's investment performance and should not
be confused with "yield" or "income." The amounts and sources of distributions reported in this notice are only esti
mates and are not being provided for tax reporting purposes. The actual amounts and sources of the amounts for accou
nting and/or tax reporting purposes will depend upon each Fund's investment experience during the remainder of its f
iscal year and may be subject to changes based on tax regulations. Each Fund will send you a Form 1099-DIV for the c
alendar year that will tell you how to report these distributions for federal income tax purposes.
```

Performance

- Statistics for indexing all documents and timing.

```
Enter your choice(1-6):5
you have selected View all statistics
Numer of documents 6
Number of words indexed 367
Number of persons indexed 16
Number of orgs indexed 9
```

Test Cases:

Tree:

Contains and Insert	Remove
Clear	Find Value
Empty	Copy Constructor
Find Minimum	Copy Assignment Operator

Memory Leaks:

No memory leaks were detected by valgrind

Valgrind output

<status>

<state>RUNNING</state>

<time>00:00:00:00.061 </time>
</status>
<status>
<state>FINISHED</state>
<time>00:00:00:00.525 </time>
</status>
<errorcounts>
</errorcounts>
<suppcounts>
</suppcounts>

Test Persistence Data:

```
accounting  3   2
actual  3   2
addingfile  6   2
addition    6   2
amounted    5   4
amounts 3   6
andor   3   2
annual  3   22
annualized  3   24
appealed    6   2
approved    3   2
arrive  6   2
asset   3   2
assuming    3   2
attributed  6   2
average 3   2
barely  6   2
based   1   2
based   3   4
bdi 6   4
berlin  6   2
bilateral   6   2
binds   6   2
bloc    6   2
```

- Create Index