Information Retrieval: Indexing

Session: 2021 - 2025

Submitted by:

Abdullah Nasir

2021-CS-113

Supervised by:

Dr. Khaldoon Syed Khurshid

Department of Computer Science

University of Engineering and Technology Lahore

Pakistan

Contents

1 Introduction

Indexing is the process for ordering data in such a way that it will be easily and quickly retrievable when someone wants to know a specific piece of information. For text files, this procedure would include reading through documents word by word, noting which terms were found, and storing the details in an ordered format. This gives fast search and retrieval because the index may immediately find words, frequency, and even the positions within the documents. This makes indexing prevent scanning of whole files for a specific query and becomes highly optimized to become one of the fundamental techniques in applications like search engines and document management systems.

2 Goal Definition

To achieve this objective, the assignment is meant for the development of a powerful indexing system capable of document retrieval both by name and content. By coming up with indexes, I intend to facilitate easy location of information out of several text files into one single document, thus enabling speedy keyword-based searches within the content as well as the option to identify files by specific names. It identifies certain data points in a vast collection of documents that a person wants to locate or retrieve easily, having designed the system to improve search accuracy and speed.

3 Document Collection

The documents that are used for indexing are **text files** i.e. with .txt extension. The content for these documents has been taken from **Wikipedia**. There a total of six documents.

1. science
2. plants
3. animals
4. computer
5. AI
6. Pokemon

Each of these files contain data related to the name of the document.

4 Implementation Plan

I plan to use **Python** as the programming language as instructed by my instructor and also because of the ease of use of this language. There will be no libraries used for the implementing the core functionality of the indexer. Only those that are strictly required will be used.

5 Approaches

In my first attempt to indexing, I came up with an index for each word every time it appeared in the text; this was with the intention of having all possible keywords and focusing on getting as much indexed data as possible. However, within one or two iterations, I realized that this approach produced a large amount of indexed data most of which was irrelevant and a lot redundant.

To fine-tune this, I developed another tactic that incorporated a list of stop words. These stop words refer to ”is,” ”am,” ”are,” ”this,” and ”there,” among others, which tend to add less meaning in search-oriented contexts. In addition, I built lists of verb and noun suffixes such that it could allow only the more pertinent words to be included. This only managed to screen out most irrelevant words but was still not exact in pinpointing exactly what mattered.

6 Backend Concept

6.1 Index Creation

While creating indexes, several steps are applied to filter and normalize words before they are included in the final index. The first step involves a check for each word against a predefined stop words list stored in an external file. If the word matches the stop words list, then the word is simply ignored and not included in the index. We normalize words that meet this filter by removing the common prefixes and suffixes, hence returning the base form of the word. It means reading prefix and suffix lists from external files, so it is easy to update and maintain these lists. The specific punctuation marks - ., ,, ?, !, (, and - are stripped from the base form of each word using a custom character-processing function that picks apart each and every character, for a careful step ensuring only meaningful normalized words are indexed in this pre-indexing step, thus reducing redundancy and irrelevant terms, improving search efficiencies and accuracy. These words in the base-form, cleaned are then added to the index, hence making it ready for efficient retrievals of the documents based on content.

6.2 Searching by Document

The program implements a straightforward lookup to allow the user to search for a document using its name. The program takes the user's query, checks if the file he or she is searching for exists within the list of the indexed file names. If it does, the program opens the file location by calling the open_file_location function that opens the directory containing the file by the system's file explorer in which the user can easily get the file.

6.3 Searching by Content

The program also allows one to search by document content. In this case, the user types in a query consisting of words. The program now searches in its indexed words for all matches. If there is at least one word from the query that exists in the index, it aggregates the counts as well as line numbers of documents that contain a word. This ranking is based on the word occurrence count in descending order. This makes it easy for the user to determine which documents best match the search query. This function assists in fetching documents that carry plenty of the query terms.

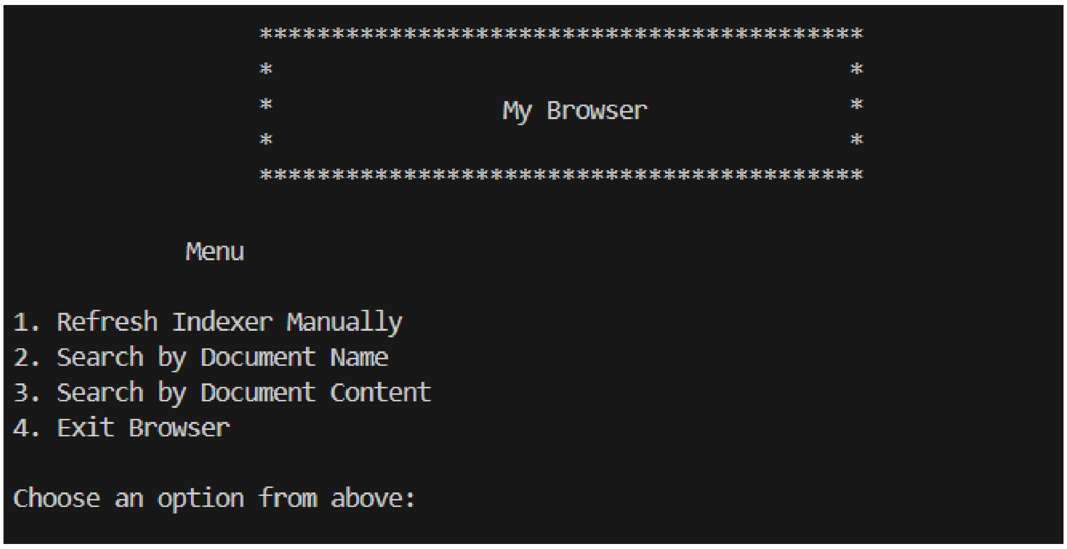
7 User Interface

This assignment was completed without creating a graphical user interface, rather it uses command line interface. Despite lacking all the glitters of graphics, I have tried my best to implement a user interface that is acceptable by the user.

7.1 Home Screen

The first thing that appears on the command line is the Home Screen for the browser. It consists of the following options for the user:

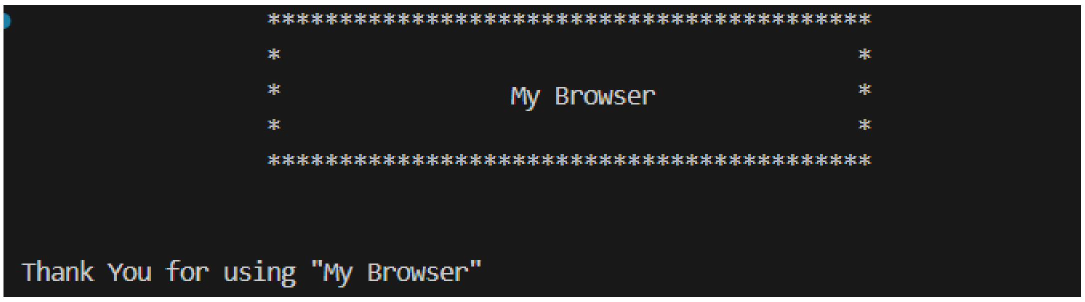
1. **Refresh Indexer Manually**: This option allows the user to manually update or refresh the indexing of all documents in the folder. This is useful if new files have been added or existing files have been modified.
2. **Search by Document Name**: This option lets the user search for a specific document by entering its name. It provides a quick way to locate and open a particular file based on its title.
3. **Search by Document Content**: This feature enables content-based search within documents. The user can enter a keyword or phrase, and the program will retrieve documents containing the specified terms.
4. **Exit**: This option allows the user to exit or close the program.



```
                  ******************************************
                  *                                        *
                  *                My Browser              *
                  *                                        *
                  ******************************************


             Menu

1. Refresh Indexer Manually
2. Search by Document Name
3. Search by Document Content
4. Exit Browser

Choose an option from above:
```

7.2 Exit Screen

After the user has used the browser and plans to exit it, the programmer seems grateful that his browser was useful.



```
                  ******************************************
                  *                                        *
                  *                My Browser              *
                  *                                        *
                  ******************************************


 Thank You for using "My Browser"
```

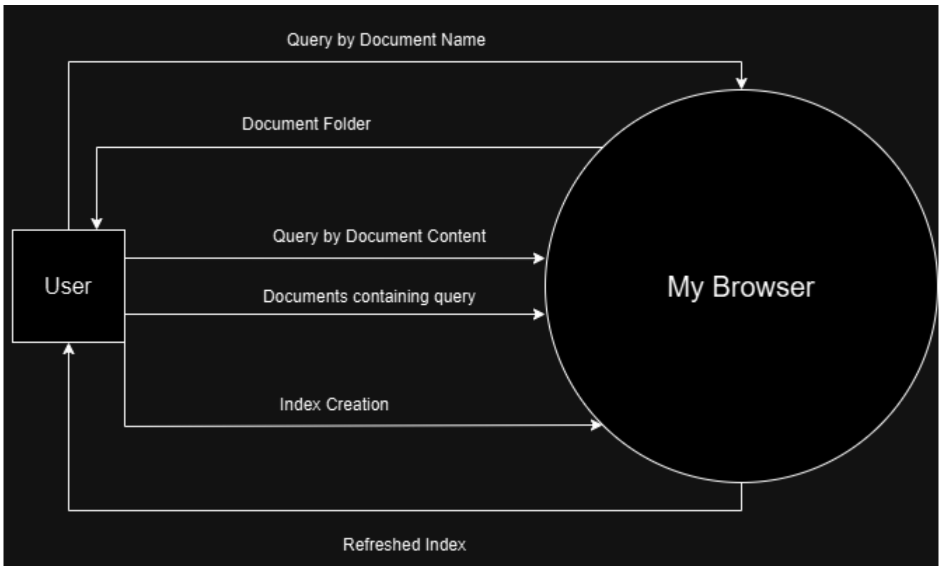
8 Shortcomings

Although I was able to successfully complete this assignment, There are several shortcomings to this type of approach.

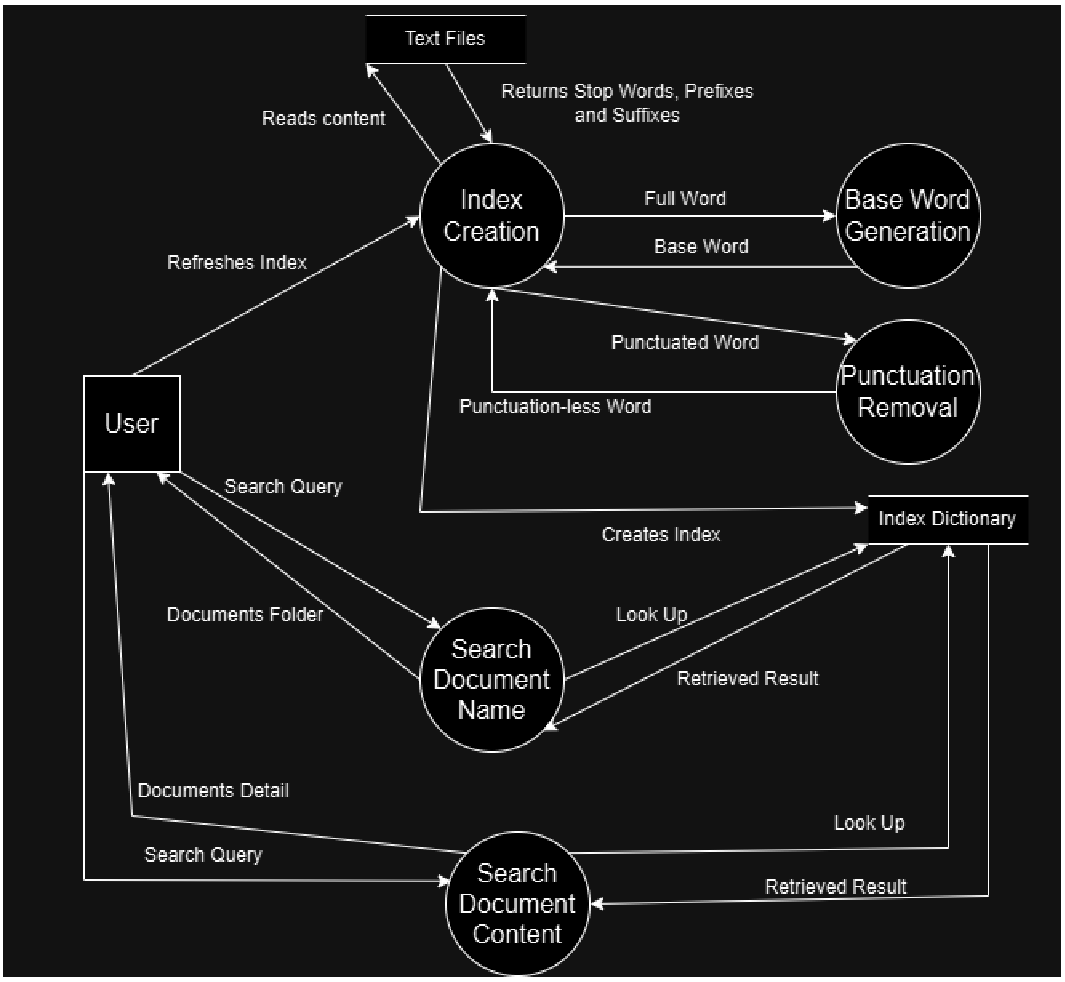
- The current indexer does not refresh automatically, multi-threading may be needed in the future.
- If using a file for stop words, we may need to add many words to it which makes the size too big to be controlled.
- Similarly, for the prefixes and suffixes, many terms will be added to files.
- Another issue with suffixes is that sometimes removing just the last common suffix is not enough or sometimes removing the suffix removes the actual word.
- A limited number of punctuation is being removed right now.
- No handling for typing errors for the user.

9 Data Flow Diagrams

9.1 DFD level 0



9.2 DFD level 1


Text Files
Reads content
Returns Stop Words, Prefixes and Suffixes
Index Creation
Full Word
Base Word
Base Word Generation
Refreshes Index
Punctuated Word
Punctuation-less Word
Punctuation Removal
User
Search Query
Creates Index
Index Dictionary
Documents Folder
Look Up
Retrieved Result
Search Document Name
Documents Detail
Look Up
Search Query
Retrieved Result
Search Document Content