Information Retrieval

Ex01

Binary Search Engine

Hessam Koucheki | 9812358032

Bu-Ali Sina University

Introduction

This project involves building a search engine for searching in multiple documents. The main objective of this project was to develop a tool that can quickly and accurately search for a specific word or phrase in a collection of documents and return the relevant results. To achieve this, I implemented a posting list that stores the terms and their corresponding documents. This allowed the search engine to efficiently index and retrieve the documents that contain the search term. Overall, this project provided me with valuable experience in developing information retrieval systems and utilizing data structures for efficient search operations.

Reading Files

Here, I explain how I wrote a code can read the documents.

First, I set the file names that I want to load into my program. Please note that because files were really huge, to test the program I just use the content of only one file, as well others are commented.

```
data = {}
file_names = ('lyrl2004_tokens_train.dat',
              #   'lyrl2004_tokens_test_pt2.dat',
              #   'lyrl2004_tokens_test_pt3.dat',
              )
```

After that, I we reach here:

```
for file_name in file_names:
    with open(file_name, 'r') as f:
        x = f.readlines()

    ID = ''
    for line in x:
        if '.I' in line:
            ID = line.replace('.I ', '').strip()
            data[ID] = ''
            print((ID + '\n') * (int(ID) % 100_000 == 0), end='')
        elif line == '.W\n':
            continue

        else:
            data[ID] += line + ' '
        # input()
```

This code reads multiple text files and extracts relevant information from them to build a data dictionary. Here's a step-by-step breakdown of what each part of the code does:

- The first line of code starts a for loop that iterates over a list of file names (stored in the variable file_names).

- The second line opens the current file in read-only mode ('r') using a with block. This ensures that the file is properly closed after it has been read. The *readlines()* method reads the entire file and returns a list of lines.

- The ID variable is initialized as an empty string. This variable will store the current document ID as the loop processes each line of the file.

- The next block of code is a for loop that iterates over each line in the file (stored in the x variable). For each line, the code checks if it contains the string '.I' (indicating the start of a new document). If it does, the code extracts the document ID from the line and stores it in the ID variable. It also initializes an empty string in the data dictionary with the document ID as the key. To keep track of the code progress, we check If the document ID is a multiple of 100,000, the code prints the ID.

- If the line is the string '.W', the code skips the line and continues with the next line.

- Otherwise, the code appends the current line to the text associated with the current document ID in the data dictionary. The += operator concatenates the current line with the existing text and adds a space at the end.

Overall, this code reads multiple files and extracts the document IDs and text content from them, storing the data in a dictionary. The code also includes a print statement to display the document IDs for every 100,000th document to monitor progress while processing large numbers of files. (I got shocked when I see that chatGPT could comprehend that the print statement was to *'monitor the progress for large number of files'*...!!!!!)

Create Dictionary/Vocabulary

In here, I explain how I created the vocabulary list or dictionary.

```
## Create Dictionary/Vocabulary
vocab_list = {}

for key, val in data.items():
    vocab_list[key] = set(val.split())


print('A total of {} documents has been tokenized ... !\n'.format(
    len(vocab_list)))
from itertools import chain
tokens = *chain(*vocab_list.values()),
print('A total of {} tokens has been extracted ... !\n'.format
      (len(tokens)))
```

This code creates a dictionary of unique words (or tokens) found in a collection of documents. Here's a step-by-step breakdown of what each part of the code does:

- The first line initializes an empty dictionary called vocab_list. This dictionary will store the unique words found in each document.

- The for loop iterates over each item in the data dictionary (created in the previous code snippet). For each document, the code splits the text into individual words using the split() method and creates a set of unique words. The document ID is used as the key in the vocab_list dictionary, and the set of unique words is stored as the value.

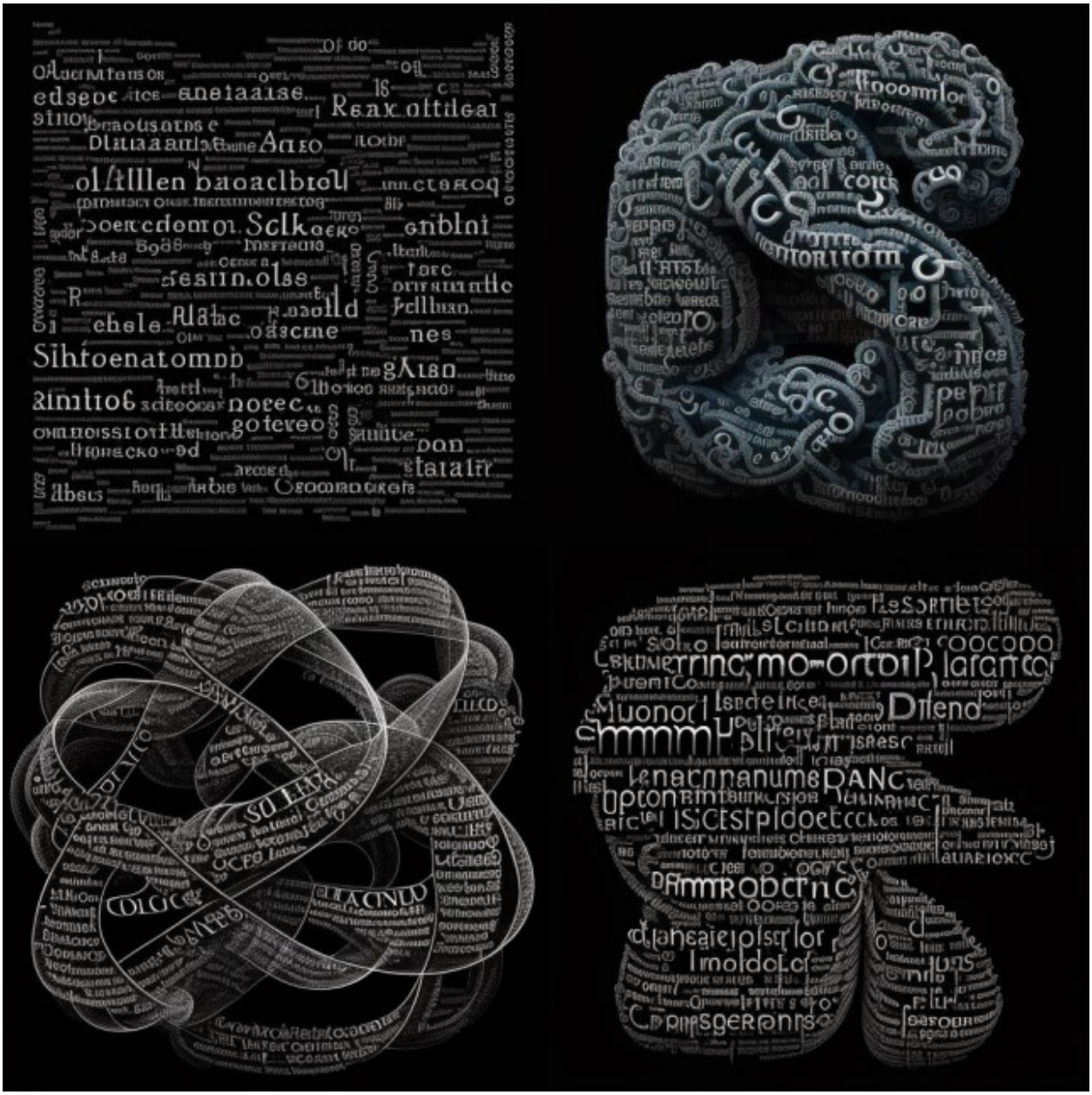
- After processing all the documents, the code prints a message indicating the total number of documents that have been tokenized (i.e., had their words extracted and stored in vocab_list).

- The next line imports the chain() function from the itertools module. The chain() function is used to concatenate multiple lists or sets into a single iterable object.

- The tokens variable is assigned the result of calling chain() with the values in vocab_list as arguments. The * operator before chain() unpacks the values in vocab_list into separate arguments. This effectively flattens the nested sets of words from each document into a single iterable object of all unique words.

Overall, this code creates a dictionary of unique words in a collection of documents and calculates the total number of unique words extracted from the documents.

Creating Posting List



Now, it’s time to explain the posting list section.

```
## Creating Posting List
print('Creating Posting List')
posing_list = {}
for token in tokens[:1000]:
    posing_list[token] = []
    for doc_id, doc_txt in vocab_list.items():
        if token in doc_txt:
            posing_list[token].append(doc_id)

print('Finished Creating Posting List.')
```

This code creates a posting list for the tokens extracted in the previous code snippet. Here's a step-by-step breakdown of what each part of the code does:

- The posing_list variable is initialized as an empty dictionary. This dictionary will store the posting list for each token (i.e., the documents that contain the token).

- The for loop iterates over the first 1000 tokens in the tokens iterable (created in the previous code snippet). For each token, the code initializes an empty list as the value for the token in the posing_list dictionary. The number 1000 is selected for test as the larger numbers will take much more time to compute.

- The next for loop iterates over each item in the vocab_list dictionary. For each document, the code checks if the current token is in the document's set of words (i.e., if the token appears in the document). If it

does, the code appends the document ID to the list of documents associated with the current token in the posing_list dictionary.

- After processing all the documents for the current token, the code moves on to the next token and repeats the process until 1000 tokens have been processed.

Overall, this code creates a posting list for a subset of tokens extracted from a collection of documents. The posting list indicates which documents contain each token and can be used for text search and analysis purposes. Note that this code only processes a subset of the tokens (the first 1000), so the full posting list for all tokens would require changing this number.

Writing to file

This is not an important part to describe separately, but Midjourney works amazing... :))))

```
## Write Posing List to the file
with open('Posting_list.dat', 'w') as f:
    for token, post_list in posing_list.items():
        f.write(token + ': ' + '→'.join(post_list) + '\n\n')
```

This code writes all the result into a file called Posting_list in a format of linkedlist.

Interactive Search Engine

Functions

Here, we defined a few functions to work with.

Intersection

```
def intersection(lst1, lst2):
    return set(lst1) & set(lst2)
```

The first function intersection(lst1, lst2) takes two lists as input and returns the intersection of the two lists as a set. In this case, it is used to find the documents that contain all the words in the user's query.

Show_doc

```
def show_doc(word):
    print('Enter the document ID to open it.\
        enter \\all to show all the documents number. \
                Enter "\\return" to return to main menu')
    doc_id = input('>> ')
    if doc_id == '\all':
        if word == 0:
            print(intersec_list)
        else:
            print(posing_list[word])

    print('Enter the document ID to open it.\
        enter \\all to show all the documents number. \
                Enter "\\return" to return to main menu')
    doc_id = input('>> ')
```

```
  while doc_id != '\\return':
      if data.get(doc_id):
          print(border)
          print(border)
          print(data[doc_id])
          print(border)
          print(border)
          print('Enter another document ID to open it.\
          Enter "\\return" to return to main menu')
          doc_id = input('>> ')

      else:
          print('You may entered a wrong document Id. Try again.\
              Enter the document ID to open it. \
              Enter "\\return" to return to main menu')
          doc_id = input('>> ')
```

The second function show_doc(word) is used to display the contents of a document. It takes a word as input to handle batch file show. This function prompts the user to enter a document ID. In the main program, to make it consice, we only show the first 50 results to the user, so If the user enters '\all',

it prints out a list of all document IDs containing the input word (if word != 0, we comprehend that the user query is containing only one word, then it prints out the documents containing the input word; if word == 0, we understand that we have to deal with intersection_list, thus it prints out the documents number in the intersect_list). If the user enters a valid document ID, it prints out the contents of that document. The function then prompts the user to enter another document ID or '\return' to return to the main menu. If the user enters an invalid document ID, it prompts the user to try again.

Main Part

First prompt

```
border = '=' * 150
if __name__ == '__main__':
    while True:
        print(border)
        print('Type a word to search in documents. \
            Search multiple words by using the + sign. \
            Type "\exit" to finish the program')
        word = input('>> ')

        if word == '\exit':
            break
```

This code is the main part of a search engine that allows users to search for words in a set of documents. Here is an explanation of the code:

This code allows users to search for words in a set of documents. The program first prompts the user to input a word to search for. The user can also search for multiple words by using the + sign to join the words. If the user inputs the "\exit" command, the program terminates.

handle multiple word search

```
# handle multiple word search
if '+' in word:
    words = [x.strip() for x in word.split('+')]
    bundle = [[posing_list.get(x)] for x in words]
    try:
        intersec_list = bundle.pop()
        # merge all lists
        while bundle:
            intersec_list = intersection(intersec_list, bundle.pop())
            # flatten the list
            while len(intersec_list) == 1:
                intersec_list = intersec_list[0]

    except:
        pass

    # # flatten the list
    while len(intersec_list) == 1:
                intersec_list = intersec_list[0]

    print(intersec_list[:min(len(intersec_list), 50)])
    show_doc(0)
```

If the user searches for multiple words, the program generates a list of documents that contain all of the searched words. The program outputs the first 50 documents that contain all of the searched words and prompts the user to select a document ID to open. If the user inputs "\all", the program outputs all of the document IDs that contain all of the searched words.

handle single word

```
elif posing_list.get(word):
    print(border)
    print('The word {} has {} occurance in all documents, here are the
    print(posing_list[word][:min(len(posing_list[word]), 50)])

    show_doc(word)
    print(border)
```

If the user searches for a single word, the program outputs the number of occurrences of the word in all documents and the first 50 documents that contain the word. The program then prompts the user to select a document ID to open.

Handle Not found

```
# Not found
else:
    print('No such a word found in our database. \
        Try searching other words! ')
```

If the searched word is not found in any document, the program prompts the user to search for other words.

Running code

In here, we work with the program and explore a few functionallity of it.

menu

```
A total of 23149 documents has been loaded...!

A total of 23149 documents has been tokenized...!

A total of 1757801 tokens has been extracted...!

Creating Posting List
Finished Creating Posting List.
============================================================================================================
============================================================
Type a word to search in documents.             Search multiple words by using the + sign.
            Type "\exit" to finish the program
>>
```

At the first run, we can see this result. This run, cause the file Posting_list.dat to be generated.

Single word search

```
============================================================================================================
============================================================
Type a word to search in documents.             Search multiple words by using the + sign.
            Type "\exit" to finish the program
>> level
============================================================================================================
============================================================
The word level has 2482 occurance in all documents, here are the top documents:
['2286', '2291', '2296', '2298', '2314', '2320', '2322', '2324', '2326', '2331', '2336', '2
338', '2339', '2340', '2342', '2352', '2355', '2374', '2398', '2408', '2431', '2432', '2439
', '2455', '2475', '2520', '2533', '2534', '2538', '2554', '2556', '2557', '2588', '2611',
'2635', '2672', '2680', '2693', '2697', '2699', '2706', '2720', '2726', '2727', '2728', '27
33', '2738', '2739', '2763', '2764']
Enter the document ID to open it.        enter \all to show all the documents number.
           Enter "\return" to return to main menu
>>
```

If we search for a word, like level, we can see the result of the search.

```
Enter the document ID to open it.        enter \all to show all the documents number.
          Enter "\return" to return to main menu
>> 2635
================================================================================================
============================================================
================================================================================================
============================================================
compan compan compan early early crimea tenfold cormick cormick cormick cormick cormick epi
c epic aktash aktash aktash kyrmtexasnaft lviv api grav paraffin sulphur karen gas gas matu
s interest march liquid
 begin undertak term spot ukrain ukrain quart produc produc drill drill percent local canad
 well level janu janu sold sold million long control contract greek greek crud oil oil oil
oil
 oil oil increas rise current current buy financ financ ronald london destin ll ll program
energ greec greec bpd basi problem develop develop export export export export contact day
degree great
 field refin refin plac barrel intern secur joint ventur promis process need weather newsro
om partn partn presid content independ establ



================================================================================================
============================================================
================================================================================================
============================================================
```

After entering a documner ID to show, this would show up all of the document on the screen.

```
>> 2697
================================================================================================
============================================================
================================================================================================
============================================================
early market market market market econom econom econom econom econom econom econom quiet sl
ip unanim tuesday tuesday hard clos clos clos clos find high immed opinion interest interes
t interest rate
 rate rate rate rate rate month low low low low low low low low begin year year year year y
ear year ahead remaind mccarth data data data followthrough bank crb stil
 stil stil stil stil point point uneas extrem wari trend aber resourc glassm glassm glassm
imply good mid poitn discount firm road dole dole wait peopl peopl week week week week
 ston fund downsid yield yield yield yield report usual percent percent percent percent per
cent percent percent percent percent percent meet meet meet meet meet strong anal anal anal
 anal expect expect
 expect lead govern treasur treasur treasur treasur bill bill chas shift fell fell level le
vel level pric pric pric pric pric gain index fomc fomc fomc futur futur test thursday open
 push end end leav leav drop candid benchmark sum monday monday remain jim edg commod commo
d feder feder bureau reserv hurt narrow stabl august public public pow robust finish unempl
oy past
 financ commit made made rose rose import cent prov cite pret cut cut effort research resea
rch friend assoc senior senior inflat inflat notic growth spend move fact heavy event sharp
 bob
 job job job note note note claim claim claim claim poll direct direct outcom rang basi bas
i lift settl wary sell consum divid josh vote session session session policymak policymak e
mphas
 comment budget secur eve fed fed fed fed bond bond bond bond found chang aver strateg morn
 polic polic polic advis releas releas unchang unchang republ surpr presid presid activ ope
rat
 desk doubt assum reluc offic sign slow trad trad tax independ hedg
```

We can keep going and ask for other documents at that moment as well.

Bu typing \return, we navigate to the search menu.

Multiple word search

```
============================================================================================
============================================================
Type a word to search in documents.             Search multiple words by using the + sign.
           Type "\exit" to finish the program
>> level + begin
['2286', '2287', '2295', '2305', '2307', '2309', '2313', '2328', '2334', '2335', '2348', '2
364', '2415', '2439', '2447', '2455', '2458', '2461', '2478', '2502', '2508', '2510', '2511
', '2518', '2520', '2528', '2536', '2538', '2539', '2550', '2570', '2573', '2631', '2633',
'2635', '2655', '2665', '2678', '2686', '2692', '2697', '2702', '2738', '2774', '2792', '28
00', '2811', '2818', '2839', '2887']
Enter the document ID to open it.        enter \all to show all the documents number.
           Enter "\return" to return to main menu
>>
```

After searching the words **level** and **begin**, we can see the result as well.

By typing the document ID, we can see the document.

```
>> 2697
============================================================================================
============================================================
============================================================================================
============================================================
early market market market market econom econom econom econom econom econom econom quiet sl
ip unanim tuesday tuesday hard clos clos clos clos find high immed opinion interest interes
t interest rate
 rate rate rate rate rate month low low low low low low low low begin year year year year y
ear year ahead remaind mccarth data data data followthrough bank crb stil
 stil stil stil stil point point uneas extrem wari trend aber resourc glassm glassm glassm
imply good mid poitn discount firm road dole dole wait peopl peopl week week week week
 ston fund downsid yield yield yield yield report usual percent percent percent percent per
cent percent percent percent percent percent meet meet meet meet meet strong anal anal anal
 anal expect expect
 expect lead govern treasur treasur treasur treasur bill bill chas shift fell fell level le
vel level pric pric pric pric pric gain index fomc fomc fomc futur futur test thursday open
 push end end leav leav drop candid benchmark sum monday monday remain jim edg commod commo
d feder feder bureau reserv hurt narrow stabl august public public pow robust finish unempl
oy past
 financ commit made made rose rose import cent prov cite pret cut cut effort research resea
rch friend assoc senior senior inflat inflat notic growth spend move fact heavy event sharp
 bob
 job job job note note note claim claim claim claim poll direct direct outcom rang basi bas
i lift settl wary sell consum divid josh vote session session session policymak policymak e
mphas
 comment budget secur eve fed fed fed fed bond bond bond bond found chang aver strateg morn
 polic polic polic advis releas releas unchang unchang republ surpr presid presid activ ope
rat
 desk doubt assum reluc offic sign slow trad trad tax independ hedg
```

Not found

```
>> hello
No such a word found in our database.                Try searching other words!
================================================================================
```

If we search a word that doesn't exists in our database, we would get an error.

With thanks and respect,

Hessam Koucheki

9812358032

The document has been accomplished

with the help of ChatGPT and MidJourney