Search Engine

TEAM MEMBERS:

MUHAMMAD WALEED ABDULLAH (348883)

MUHAMMAD TALHA (338796)

USAMA QURESHI (338954)

Tools Used:

Python

Modules Used:

- json
- os
- nltk
- Tkinter

Components:

1. **Forward Index:**

The forward index is generated by generate_forward_index function which takes path to data as an argument. The structure of forward index is hashed docID and wordID as key and a list containing two sublists. One for storing hit counts for word in title and other for storing hit count for content and position of hits in that doc.

```
[[3273015187, 0], [[1, 1], [0, 1, 2]]]
[[3273015187, 1], [[1, 1], [0, 1, 9]]]
[[3273015187, 2], [[1, 1], [0, 2, 7, 45]]]
[[3273015187, 3], [[1, 1], [0, 2, 8, 46]]]
[[3273015187, 4], [[1, 1], [0, 1, 17]]]
[[3273015187, 5], [[1, 1], [0, 1, 41]]]
[[3273015187, 6], [[1, 1], [0, 1, 42]]]
[[3273015187, 7], [[1, 1], [0, 1, 43]]]
[[3273015187, 8], [[1, 1], [0, 2, 6, 40]]]
[[3273015187, 9], [[1, 1], [0, 0]]]
[[3273015187, 10], [[1, 0], [0, 1, 1]]]
[[3273015187, 11], [[1, 0], [0, 1, 3]]]
[[3273015187, 12], [[1, 0], [0, 1, 4]]]
```

2. **Inverted Index:**

The inverted index is generated by inverted_index_generator function. The structure of inverted index is same as forward index. The only difference is that inverted index is sorted by wordID.

```
[[3273015187, 0], [[1, 1], [0, 1, 2]]]
[[1812025179, 0], [[1, 1], [0, 0]]]
[[2324107447, 0], [[1, 0], [0, 2, 79, 156]]]
[[4057307547, 0], [[1, 1], [0, 2, 47, 67]]]
[[2124223359, 0], [[1, 1], [0, 0]]]
[[1118006972, 0], [[1, 0], [0, 1, 2550]]]
[[4010602117, 0], [[1, 0], [0, 6, 503, 564, 620, 643, 789, 1038]]]
[[3294628077, 0], [[1, 0], [0, 1, 98]]]
[[4229926718, 0], [[1, 1], [0, 1, 10]]]
[[3794842426, 0], [[1, 0], [0, 1, 480]]]
[[1901351923, 0], [[1, 0], [0, 2, 451, 1781]]]
[[3953214014, 0], [[1, 1], [0, 1, 52]]]
[[3828554757, 0], [[1, 1], [0, 1, 23]]]
[[2713256931, 0], [[1, 1], [0, 2, 1, 26]]]
[[4112524127, 0], [[1, 1], [0, 2, 5, 27]]]
```


3. **Lexicon**:

The lexicon contains mappings of word to a list which contains word ids and its offset in bytes into inverted index. At top of this file, we store the word count in lexicon.


```
{"word_count": [40537, 0], "lockdown": [0, 0], "protest": [1, 20153],
```


4. **Document Index:**

The document index file contains mappings of doc ids to their urls.


```
{"3273015187": "https://21stcenturywire.com/2020/04/15/reopennc-lockdown-protest-called-non-essential-activity-by-raleigh-police/",
```


Procedure:

Indexing:

In forward indexing, we first open the file lexicon in read mode and load its data into dictionary named lexicon. Secondly, the file document index is opened in read mode and all its data is loaded into a dictionary named document_indices. This file is then opened in write mode so new docs that are indexed can have their doc ids written to it. Then we open 300 files called forward barrels and store their file pointers in a list named forward_barrels. One by one we open each json file in folder given for indexing. Similarly we create 300 dictionaries each implicitly referred as forward dict and create a list named forward_dicts which contains these forward dictionaries.

From the loaded data from the json file, we loop through each article. For each article, we store its doc id and hash it using function named crc32 in zlib module. We then check if the document was previously

indexed or not, if not then store the hashed doc id and its url as key value pair in document_indices dictionary. Then the content of article is parsed. In parsing we pick valid words using regex expression, then lower it and split it into words. After that we remove stop words and then stem the words. Similar operation is performed on title part of article. For each word in title and content, we check if word is in lexicon if not we add word and list containing word id and default value for offset into inverted index as key value pair.

From the word id we compute to which barrel the word should belong and check if tuple of doc id and word id is already in barrel or not. If not in that barrel, we store tuple of hashed doc id and word id and a list which acts as hitlist as key value pair in forward dict which will be later written to corresponding forward barrel. In this list, there are two sub lists, one which contains hiltist for title and other content. The hitlist for title means that in the doc, the word occurs in title how many times. For this title hitlist, we only store how many times word occurs in title of that doc. In hitlist list for content, we store hit count as well as position of where in doc that word occurs. In case if tuple of doc id and word id is already in barrel, we just append the new position where word occurs to the corresponding sub list and increment hit count.

Lastly, one by one we write 300 forward dictionaries to forward barrels. Then we write document indices dictionary to file document index. The file lexicon is opened in write mode and new lexicon with new words is written to the file. Lastly, the function forward_index_generator returns a list containing doc count, and time taken to index.

Sorting:

To create inverted index, we open forward barrel one by one and create inverted barrel. To create inverted barrels, we use the function inverted_index_generator. In this function, we open lexicon file in read mode and load its content to lexicon dictionary and all its keys in a list named lexicon_keys. One by one we open inverted barrel. If file previously exist, its previous content is loaded line by line into a list named inverted_list then we open the corresponding forward barrel and append its content to the inverted_list. Now inverted barrel file is opened in write mode and inverted_list is sorted on basis of word id using function sort which uses counting sort to sort the list. Lastly we write the sorted list to inverted barrel and during this procedure, the offset value of word id in inverted index is also added to lexicon. The new lexicon is written to the file.

Searching:

For searching, there are two function search_lexicon and searchWords. The search_lexicon function takes a word as an argument and searches for a word in lexicon, if found it returns value corresponding to word as key which is a lost containing word. The searchWords function takes list of words entered in search query. One by one it calls search_lexicon function to get word_id and offset value into inverted index in form of a list. These results are appended to a list named word_ids. A dictionary named documents is created which store doc id and IR score of the doc and hit list containing word position.

For each word id in word, its barrel number is computed and the corresponding inverted barrel is opened and using the offset value into inverted barrel, the file pointer is moved the required number of bytes using seek function. Then each line is read afterwards and each entry is decomposed into docID, titleHitList, titleHits, contentHitList, contentHits. Then IR score for each doc is computed. Firstly, the content hits and title hits scaled by factor 5 are added to IR score count of that doc. Then position of each hit of a word in each doc is compared one by one with other word hits in that doc and according to proximity, a certain weight is added. If difference in position of two words in a doc is one, weight added is 10, if difference is less than or equal to 10 it means that they are in a sentence so weight added is 8 and if difference is less than or equal to 100 then weight assigned is 4 and else 2 is added.

Finally the document named dictionary is sorted on basis of IR score and result is displayed.