Wikipedia Search Engine

By - Sanyam Sharma 2K18/SE/115

Submitted to - Dr. Rajiv Mishra

INTRODUCTION

1- Implemented a search engine on the wikipedia corpus in python using k-way merge sort and added relevance ranking using tf-idf score.

2- Used inverted index data structure which is derived from term document matrix structure. We also used offsets to access required values in inverted indexes efficiently.

3-The corpus is in XML format and over 75 GB in size , which made it difficult to implement as the whole corpus cannot be accommodated in RAM at once.

4- The project used the concepts of algorithm design and analysis such as merging sorted files, tree-traversal, pattern matching and searching.

Inverted Indexes

Token	Document Id
Harry	1, 2
Potter	1, 2
And	1, 2
The	1, 2
Half	1
Blood	1
Prince	1
Deathly	2
Hallows	2

Inverted index

1- The inverted index data structure is derived from the Term-document matrix in which cell i,j stores a boolean value representing whether token i is present in document j or not.

2- This results in a sparse matrix and takes up a lot of space which is not feasible for the large corpuses. Therefore, just like adjacency list is derived from sparse matrix in graph representation , in a similar fashion Inverted index data structure is derived from Term Document matrix.

3- We have however used a slightly different version of the inverted index by pairing the document IDs with the corresponding token frequencies to assign TF-IDF score later.

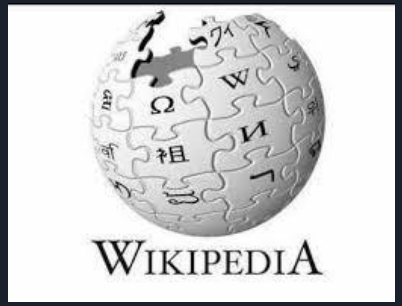
Offsets

1-Offsets consist of pointer to a particular word in the inverted index so that it can be accessed efficiently later on.

2- We have created an offset for titles so as to access titles in constant time while returning the queries.

3- We have also created an offset for each category in order to calculate final tf-idf scores efficiently.

Traversal



1- We iterated over the XML file using in-built XML parser in python and extracted tokens from Infobox, category box and title from a particular wikipedia page using regular expressions.

2- We created a three dictionaries , one for each category for each page storing the frequency of the words , and for every 50,000 pages we write the inverted index in a text file , which is stored in hard disk freeing up the RAM in order to load another batch of pages.

3- After creating all the inverted indexes and storing them as text file in hard disk we merge them using k-way merge sort and calculating TF-IDF scores alongside, in order to create the final inverted indexes which will be used to fetch the query.

Merging the files

1- We created 413 text files for each of the three categories, which stored the inverted indexes in given format :- {word1} : {<DocID1,freq1>,<DocID2,freq2>,.....}

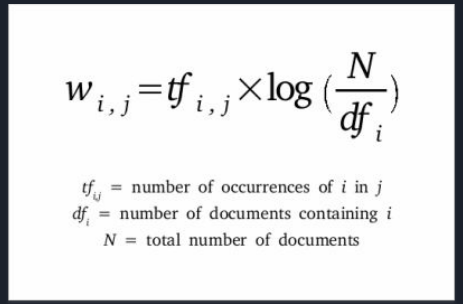
2- We treat each inverted index file as a sorted array and insert the first element of each file in a heap to begin with.

3- We then write the min element in the final inverted index of the corresponding category and also maintain the word offset.

4- Using page_count , freq and docid we calculate TF-IDF scores which will be covered in the next slide.

5- The final inverted index will stored in the following format : {word1} : {<DocID1,TF-IDF score 1>,<DocID2,TF-IDF score 2>.....}

TF-IDF Score



1-TF-IDF score consists of product of 2 values , the term frequency and the inverse document frequency.

2-The term frequency represents the impact of the particular word in the current document irrespective of the weight of the word.

3- The inverse document frequency represents the weight of the word considering the number of documents it is present in, the more rare a word, the more specific its meaning is to a particular document.

4- The product of tf and idf gives us a pretty good estimation of relevance of the returned links to a given query.

Retrieving Results

We use the query words and iterate over the corresponding list to that word in the inverted indexes by retrieving the position using word offsets and add up the tf-idf scores of the documents and sort them according to tf-idf scores and return the top 10 titles.

Thank you.