Ambrose Tuscano
CMSC 676 Information Retrieval
Spring 2020
Project 4 Write Up

Introduction:

In the past Project we had indexed the 503 documents, creating an index and a posting file. Indexed file had indexes arranged in alphabetic order of the tokens.

After an Information Retrieval system has processed and indexed the documents, a service must be provided for a user to query these documents containing the particular token via the index file. The retrieval system takes in queries which are lists of words and then matches the query words against the inverted file to come up with document weights.

Implementation:

So, as the aim was to take a query, and find out which documents have the tokens in them along with their relevance to the document.

I started with what I had from the previous Project 3. Since I had the indexing and posting file already done what I was left with was just loading the files and then querying them and finding the relevant posting for the query.

I started by loading the indexing file and posting file in memory respectively and adding their contents in a string data type, later I used split by new line to create array of both index and postings so it would be easy to access them.

So, I had an index and a posting array in memory.

Next part was for confirming the necessary inputs in the command line argument passed to run the system.

The tricky part was taking a query and then retrieve it. Here, I also dealt with the error that may be caused due to uppercase of query tokens, as all words in indexed were in lower case, this I did by lowercasing the query. The part of using Weighted Query as instructed in the Project documentation, from what I could understand was that tokens carrying a higher weight would outlie tokens with lower weight as mentioned for Weighted(Wt) query. To do that, we would take the weight for the command line by and multiply it with the idf we get from the posting file. So by doing this the token having more weight in query would get more idf value. And the whole query output will change accordingly.

Similarly, For normal query each Query token would take the same weight(I generalized it to be 1 for any unweighted token, so while multiplying with idf we get the original idf). I maintained the Query token along with its weight in a dictionary for easy use.

Next, I worked on retrieval of the Query token from the Index, to do so I created a retrieved hash map. I iterated the token along with weight till the query had tokens(i.e. using items() function).

Now for each word in the query I checked if it was indexed, by running through the index array. For this part, I could have had options like binary search the array for faster results, but I didn't go with them as already due to the smaller documents corpus size the running time was less. If we are dealing with huge corpus, binary search through tokens would be a great alternative as I had already indexed the files in alphabetic order. For binary search it would have had been better if I had taken Index input as a dictionary rather than an array that I took during this Implementation.

Going further, I found the index(positon) of the particular token in the index array, and using the same index, I found the number of postings and starting position of the Token document postings in the postings file. This I did by adding +1 and +2 to the index of the token, as we had saved the index file as: token, number of postings, index in posting file in the index array.

Next from the posting array, I found the postings using the index in posting file plus the number of posting file. I named this as the document array. Now for Individual entries in the document array, I split the entries to create strings of documents and idf.

And then, we create a hash map dictionary where Retrieved document is stored as the key and the weight gets stored as the value. If there is a document, then we calculate its weight using the idf value from posting file and multiplying it by the weight in query.

Next, was sorting the retrieved dictionary. I did that using the weight value as the key, to sort the dictionary and save it as a list with document and weight as a single tuple. Sorting was done using the weight in descending order. So, while printing individual the tuples, I print the individual items of the tuples thus getting in the required for of:

Document weight

If no documents are found in the retrieved dictionary, then the sort list is empty, so I print that the words are not indexed.

Also, for as only top ten documents were required, so I print only the first ten tuples in the sorted list.

Conclusion:

As the number of Files indexed reduces, the time taken to retrieve the postings also reduced. The time taken to run the system over a indexed corpus of 100 files was average 0.0138282299041748 seconds, whereas for the 503 file corpus, it averaged at 0.05585002899169922 seconds of the 5 queries times, I tested for each. For 200,300 and 400 files, it was in the same linear increase.

So, taking into consideration the small time needed to do the retrieval, I didn't think of using binary search over the alphabetically sorted index array. But that could be done for huge corpus, saving about half the time as it would be O(log n).

Outputs :

Word : Diet

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py  diet
{'diet': 1}
Is found in 7 documents
018.html 0.06321153450712644
058.html 0.007797251304806207
263.html 0.006420047555086119
252.html 0.0044651950080547625
009.html 0.004172454876568497
050.html 0.0031316337150508614
353.html 0.0009469827250873783
Time : 0.04687309265136719
```

Word : international affairs

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py international affairs
{'international': 1, 'affairs': 1}
Is found in 128 documents
219.html 0.029520045832838328
161.html 0.017931244968597396
138.html 0.016992277748958007
205.html 0.01656776156974037
247.html 0.01529186913474678
125.html 0.012610315005673707
143.html 0.011689242860504975
117.html 0.011551180936955705
243.html 0.010508199661027
197.html 0.008979341876011473
Time : 0.05684661865234375
```

Word : Zimbabwe

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py Zimbabwe
{'zimbabwe': 1}
Is found in 0 documents
Word not found in documents indexed
Time : 0.04587697982788086
```

Word : computer network

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py computer network
{'computer': 1, 'network': 1}
Is found in 75 documents
060.html 0.041535845662355644
140.html 0.02641404842357841
156.html 0.02603811814638891
128.html 0.018668575577566696
181.html 0.01641693935309502
135.html 0.012931877874043598
380.html 0.012763134509139113
501.html 0.01210000026891636
502.html 0.011711889395360239
221.html 0.010476458024541647
Time : 0.05485224723815918
```

Word : hydrotherapy

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py hydrotherapy
{'hydrotherapy': 1}
Is found in 2 documents
299.html 0.0019175678699444327
043.html 0.0003642671529015482 3
Time : 0.05186009407043457
```

Word : identity theft

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py identity theft
{'identity': 1, 'theft': 1}
Is found in 23 documents
379.html 0.011999948179847628
301.html 0.006303321194676861
245.html 0.006288198063211705
328.html 0.004410839705059978
298.html 0.002559093259594828
397.html 0.0015977328412257004
292.html 0.001426734785914103
362.html 0.0013863584065299174
235.html 0.0013532223010876904
027.html 0.0012461586880689 61
```

Weighted Queries :

As asked for, I implemented the Weighted Query as per my own understanding.

Below, I have shown the changes that weights had on the query, to do so I used the two queries implemented above: Diet and International Affairs, as it would be easy to compare the differences.

Diet : (Single Word Query)

Under Normal Weight :

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py  diet
{'diet': 1}
Is found in 7 documents
018.html 0.06321153450712644
058.html 0.007797251304806207
263.html 0.006420047555086119
252.html 0.0044651950080547625
009.html 0.004172454876568497
050.html 0.0031316337150508614
353.html 0.0009469827250873783
Time : 0.04687309265136719
```

Weighted Query

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py Wt 0.2 diet
{'diet': 0.2}
Is found in 7 documents
018.html 0.012642306901425288
058.html 0.0015594502609612415
263.html 0.001284009511017224
252.html 0.0008930390016109526
009.html 0.0008344909753136995
050.html 0.0006263267430101723
353.html 0.0001893965450174757
Time : 0.0538175106048584
```

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py Wt 0.4 diet
{'diet': 0.4}
Is found in 7 documents
018.html 0.025284613802850575
058.html 0.003118900521922483
263.html 0.002568019022034448
252.html 0.0017860780032219051
009.html 0.001668981950627399
050.html 0.0012526534860203447
353.html 0.0003787930900349514
Time : 0.05088067054748535
```

For, Positive weightes, It was observed that the ordering didn't change, just the weights got multiplied as expected.

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py Wt -0.4 diet
{'diet': -0.4}
Is found in 7 documents
353.html -0.0003787930900349514
050.html -0.0012526534860203447
009.html -0.001668981950627399
252.html -0.0017860780032219051
263.html -0.002568019022034448
058.html -0.003118900521922483
018.html -0.025284613802850575
Time : 0.050825595855712891
```

But, for the Negative weight, It was observed that order inverted, that was because of the change of the weights falling in the negative number line.

International affairs: (Multi Word Query)

Under Normal Weight :

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py international affairs
{'international': 1, 'affairs': 1}
Is found in 128 documents
219.html 0.029520045832838328
161.html 0.017931244968597396
138.html 0.016992277748958007
205.html 0.01656776156974037
247.html 0.01529186913474678
125.html 0.012610315005673707
143.html 0.011689242860504975
117.html 0.011551180936955705
243.html 0.010508199661027
197.html 0.008979341876011473
Time : 0.05684661865234375
```

Under Customized weight:

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py Wt 0.4 international 0.3 affairs
{'international': 0.4, 'affairs': 0.3}
Is found in 128 documents
219.html 0.008856013749851498
161.html 0.007172497987438959
138.html 0.006796911099583203
205.html 0.006627104627896148
247.html 0.006116747653898712
125.html 0.005044126002269483
143.html 0.00467569714420199
117.html 0.004620472374782282
243.html 0.0042032798644108
197.html 0.0035917367504045895
Time : 0.05082297325134277
```

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py Wt 0.3 international 0.5 affairs
{'international': 0.3, 'affairs': 0.5}
Is found in 128 documents
219.html 0.014760022916419164
161.html 0.0053793734905792185
138.html 0.0050976833246874015
205.html 0.004970328470922111
247.html 0.004587560740424034
125.html 0.003783094501702112
143.html 0.0035067728581514926
117.html 0.0034653542810867115
243.html 0.0031524598983081
229.html 0.002902251697048712
Time : 0.0548405647277832
```

In Multi Word Queries, It was observed that the documents change as per the weights changes in the query, this was what was required in the Project requirements.

Having a negative weight on one word had more profound effect, and thus highlighted the positive(generally saying more weighed) query.

```
C:\Users\ambro\Desktop\retrieve>python retrieval.py Wt 0.3 international -0.5 affairs
{'international': 0.3, 'affairs': -0.5}
Is found in 128 documents
161.html 0.0053793734905792185
138.html 0.0050976833246874015
205.html 0.004970328470922111
247.html 0.004587560740424034
125.html 0.003783094501702112
143.html 0.0035067728581514926
117.html 0.0034653542810867115
243.html 0.0031524598983081
197.html 0.002693802562803442
108.html 0.0023661289983764105
Time : 0.08380317687988281

C:\Users\ambro\Desktop\retrieve>python retrieval.py international
{'international': 1}
Is found in 115 documents
161.html 0.017931244968597396
138.html 0.016992277748958007
205.html 0.01656776156974037
247.html 0.01529186913474678
133.html 0.01437151448900819
125.html 0.012610315005673707
143.html 0.011689242860504975
117.html 0.011551180936955705
243.html 0.010508199661027
197.html 0.008979341876011473
Time : 0.07879400253295898
```