PHASE 4 – INFORMATION RETRIEVAL QUERY INDEX PROJECT

NAME: KRITHIKA VERMA

CAMPUS ID: TR12417

APPROACH

The output Dictionary, posting files and Term document dictionary from assignment 3 are used to evaluate scores of documents.

Strategy:

I have followed Term-at-a-time approach as it does not have any restrictions on the number of words in the query. While Doc-at-a-time is good for query with two words but not very useful for a multi-word query. The complexity achieved with Doc-at-a-time is O(n^2) and is not efficient.

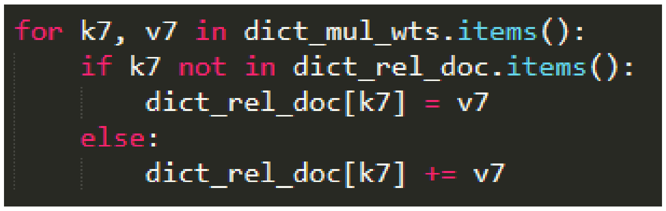
Algorithm:

1. Query terms weights are fetched (if provided in query). This goes through all through preprocessing of down casing and removing stop words. I have used a map(dictionary) for storing the term VS weights.
2. Query is iterated over term by term.
3. Each query term is iterated over the term document matrix which consists of token as key and values as posting list containing DocId and TF-IDF.
4. For a matching query term, its DocId and term weights multiplied by query term weight are stored in another map.
5. All the relevant documents containing the query terms are updated with calculated weights taking query term factor into account.
6. Steps 3 to 5 are repeated for all the terms found in the query.
7. Display the top ten relevant documents.

Data Structures:

Maintaining partial document weights using an array is not efficient because the relevant documents will be much less than the total number of documents in the corpus. Having an array with 0 weights for all the irrelevant documents is not an efficient way and proper utilization of memory.

So, I have used a Hash Table to maintain the document-query similarity scores (weights). An entry is created in the table whenever a new relevant document is encountered for a query term matching. The score is updated each time the same document is encountered while evaluating the other terms.



```
for k7, v7 in dict_mul_wts.items():
    if k7 not in dict_rel_doc.items():
        dict_rel_doc[k7] = v7
    else:
        dict_rel_doc[k7] += v7
```

Complexity:

O (q+n+N)

q → Number of terms in query

n → length of posting list, corresponding to each query term

N → Number of documents in the corpus

Reading Term Document Matrix (TDM)

My algorithm searches for the query term in TDM. Once the terms are found, corresponding DocId and term weights are fetched and stored in the dictionary. The loop breaks once the required information is fetched. I chose this method because it avoids the need to completely search and traverse the whole TDM across 503 documents and take it to in-memory before finding the information. There are higher chances that the term will be found much before and there is no need to traverse the whole TDM until 503 documents.

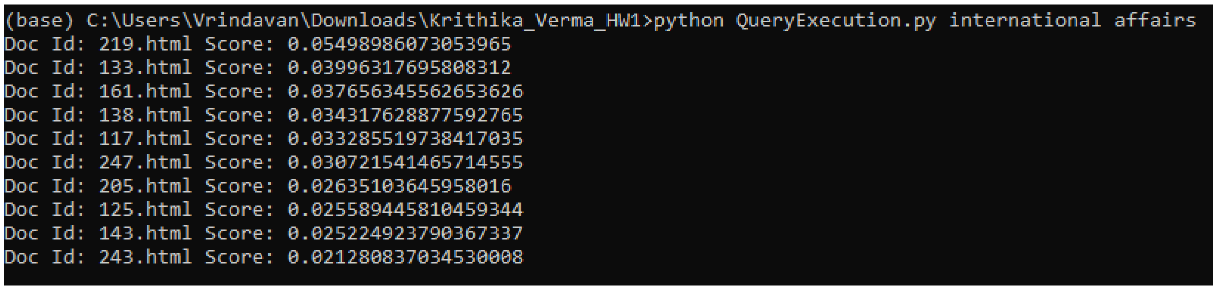
Query Term Weights (Extra 5% bonus)

Query terms are assigned weights to give importance to terms that seems more important while fetching the relevant documents. The query terms that are common and has more occurrence than the rest does not contribute any significant meaning to query should be assigned minimal weight because they should be given minimal importance while getting the relevant documents.

The weights of the term are provided in the input query. If the user is providing weights, then query starts with ‘wt’. Sample query = ‘wt 0.3 dog 0.6 rat 0.9 bird’ signifies weights of dog, rat and bird as 0.3, 0.6 and 0.9 respectively. This means that documents containing rat and bird will be given more importance than dog while calculating document similarity score.

<u>Sample Input and Output</u>

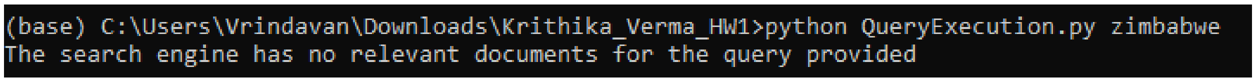
1. Query = 'international affairs'



```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python QueryExecution.py international affairs
Doc Id: 219.html Score: 0.05498986073053965
Doc Id: 133.html Score: 0.03996317695808312
Doc Id: 161.html Score: 0.037656345562653626
Doc Id: 138.html Score: 0.034317628877592765
Doc Id: 117.html Score: 0.033285519738417035
Doc Id: 247.html Score: 0.030721541465714555
Doc Id: 205.html Score: 0.02635103645958016
Doc Id: 125.html Score: 0.025589445810459344
Doc Id: 143.html Score: 0.025224923790367337
Doc Id: 243.html Score: 0.021280837034530008
```

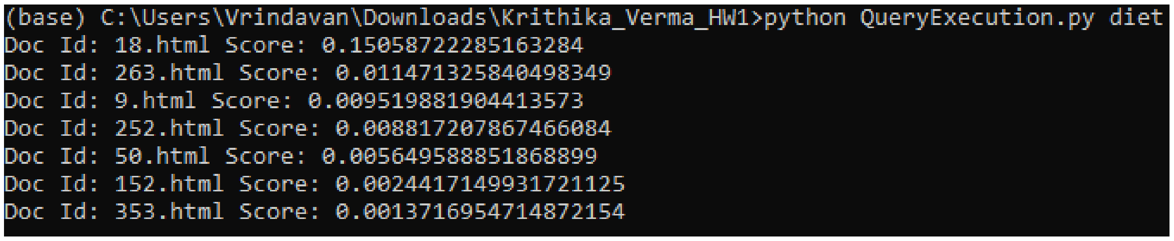
2. Query = 'Zimbabwe'

The search engine has no relevant documents for query provided.



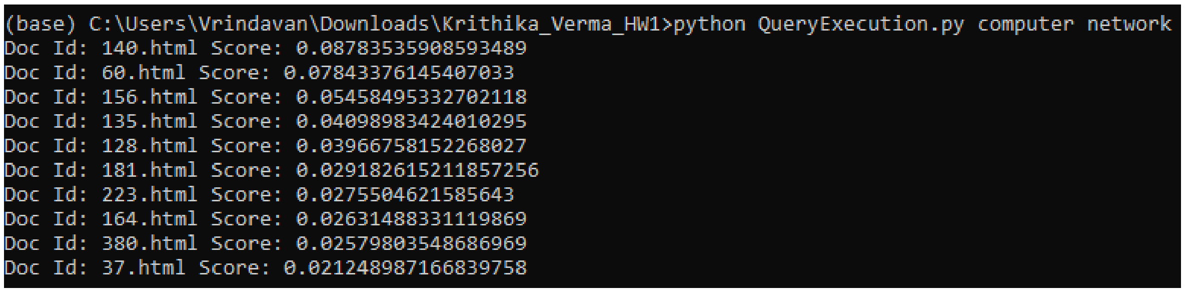
```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python QueryExecution.py zimbabwe
The search engine has no relevant documents for the query provided
```

3. Query = 'diet'



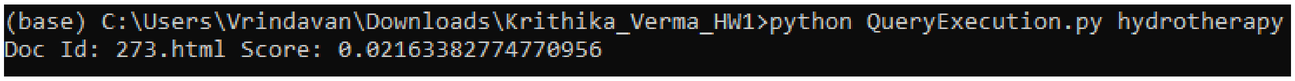
```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python QueryExecution.py diet
Doc Id: 18.html Score: 0.15058722285163284
Doc Id: 263.html Score: 0.011471325840498349
Doc Id: 9.html Score: 0.009519881904413573
Doc Id: 252.html Score: 0.008817207867466084
Doc Id: 50.html Score: 0.005649588851868899
Doc Id: 152.html Score: 0.0024417149931721125
Doc Id: 353.html Score: 0.0013716954714872154
```

4. Query = 'computer network'



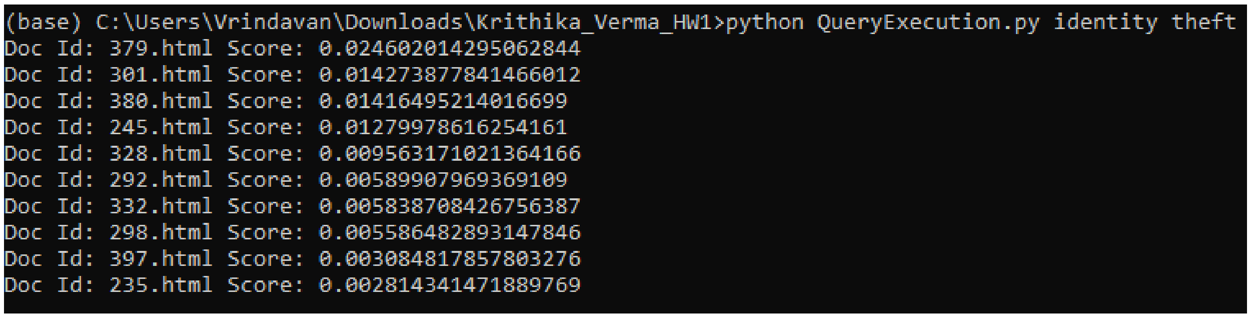
```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python QueryExecution.py computer network
Doc Id: 140.html Score: 0.08783535908593489
Doc Id: 60.html Score: 0.07843376145407033
Doc Id: 156.html Score: 0.05458495332702118
Doc Id: 135.html Score: 0.04098983424010295
Doc Id: 128.html Score: 0.03966758152268027
Doc Id: 181.html Score: 0.029182615211857256
Doc Id: 223.html Score: 0.0275504621585643
Doc Id: 164.html Score: 0.02631488331119869
Doc Id: 380.html Score: 0.02579803548686969
Doc Id: 37.html Score: 0.021248987166839758
```

5. **Query = ‘hydrotherapy’**



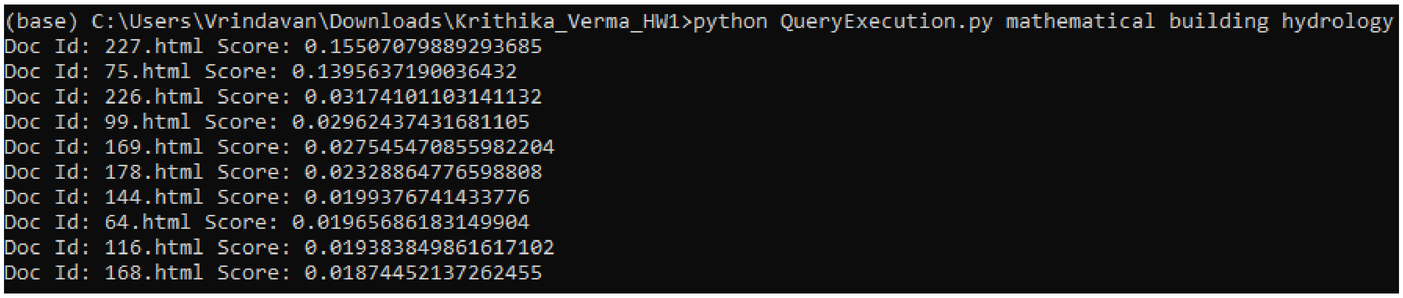
```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python QueryExecution.py hydrotherapy
Doc Id: 273.html Score: 0.02163382774770956
```

6. **Query = ‘identity theft’**



```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python QueryExecution.py identity theft
Doc Id: 379.html Score: 0.024602014295062844
Doc Id: 301.html Score: 0.014273877841466012
Doc Id: 380.html Score: 0.01416495214016699
Doc Id: 245.html Score: 0.01279978616254161
Doc Id: 328.html Score: 0.009563171021364166
Doc Id: 292.html Score: 0.00589907969369109
Doc Id: 332.html Score: 0.005838708426756387
Doc Id: 298.html Score: 0.005586482893147846
Doc Id: 397.html Score: 0.003084817857803276
Doc Id: 235.html Score: 0.002814341471889769
```

7. **Query = ‘mathematical building hydrology’**

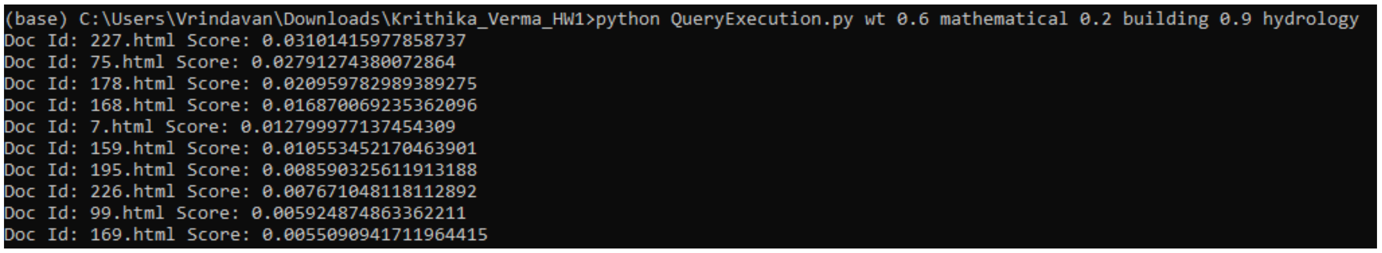


```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python QueryExecution.py mathematical building hydrology
Doc Id: 227.html Score: 0.15507079889293685
Doc Id: 75.html Score: 0.1395637190036432
Doc Id: 226.html Score: 0.03174101103141132
Doc Id: 99.html Score: 0.02962437431681105
Doc Id: 169.html Score: 0.027545470855982204
Doc Id: 178.html Score: 0.02328864776598808
Doc Id: 144.html Score: 0.0199376741433776
Doc Id: 64.html Score: 0.01965686183149904
Doc Id: 116.html Score: 0.019383849861617102
Doc Id: 168.html Score: 0.01874452137262455
```

In this example, word ‘building’ is relatively common and occurs 61 documents of the corpus, the second most common word is ‘mathematical’ that occurs in 10 documents and then ‘hydrology’ occurs in 4 documents. So, the word ‘hydrology’ is most important and should be highly weighted, and ‘building’ should be least weighted.

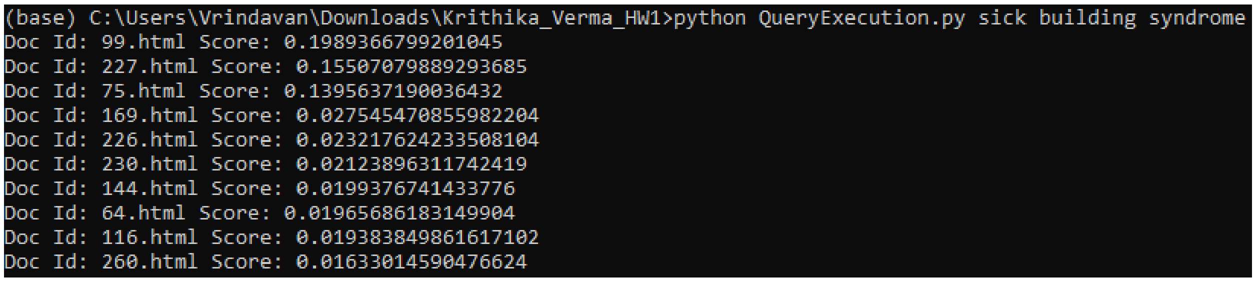
After applying the weights to the query term, we see that a new document ‘7.html’ has been added because of the presence of relatively important term (hydrology). Also, the rank of documents 168.html, 178.html has got better because of the same logic.

8. **Query = ‘wt 0.6 mathematical 0.2 building 0.9 hydrology’ [first string is wt to indicate that weights are present in the query]**



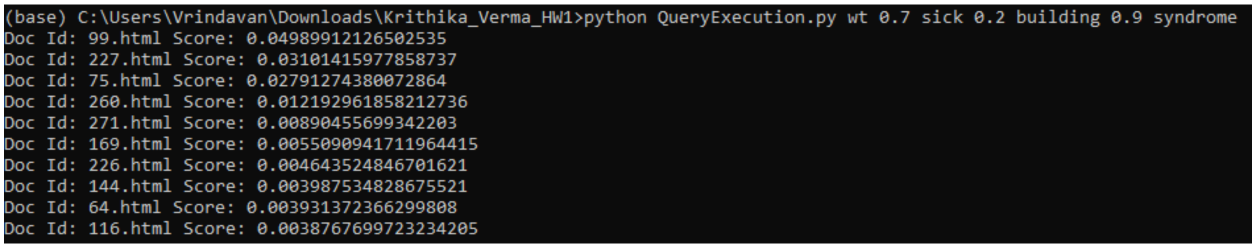
```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python QueryExecution.py wt 0.6 mathematical 0.2 building 0.9 hydrology
Doc Id: 227.html Score: 0.03101415977858737
Doc Id: 75.html Score: 0.02791274380072864
Doc Id: 178.html Score: 0.020959782989389275
Doc Id: 168.html Score: 0.016870069235362096
Doc Id: 7.html Score: 0.012799977137454309
Doc Id: 159.html Score: 0.010553452170463901
Doc Id: 195.html Score: 0.008590325611913188
Doc Id: 226.html Score: 0.007671048118112892
Doc Id: 99.html Score: 0.005924874863362211
Doc Id: 169.html Score: 0.0055090941711964415
```

9. **Query = ‘sick building syndrome’**



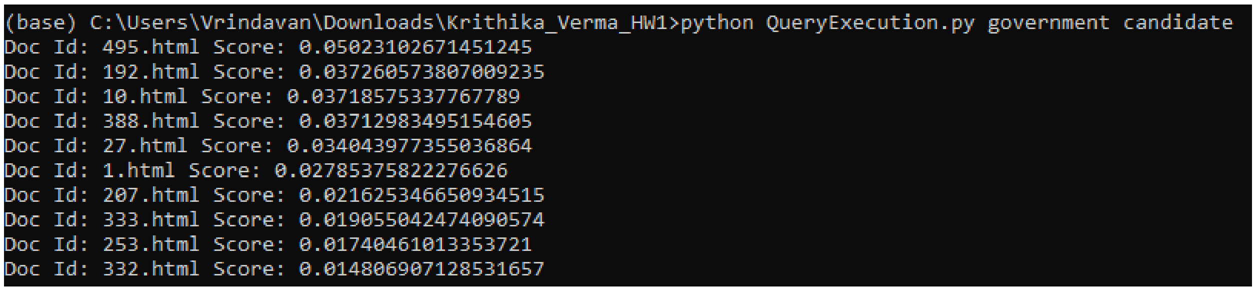
```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python QueryExecution.py sick building syndrome
Doc Id: 99.html Score: 0.1989366799201045
Doc Id: 227.html Score: 0.15507079889293685
Doc Id: 75.html Score: 0.1395637190036432
Doc Id: 169.html Score: 0.027545470855982204
Doc Id: 226.html Score: 0.023217624233508104
Doc Id: 230.html Score: 0.02123896311742419
Doc Id: 144.html Score: 0.0199376741433776
Doc Id: 64.html Score: 0.01965686183149904
Doc Id: 116.html Score: 0.019383849861617102
Doc Id: 260.html Score: 0.01633014590476624
```

10. **Query = ‘wt 0.7 sick 0.2 building 0.9 syndrome’**



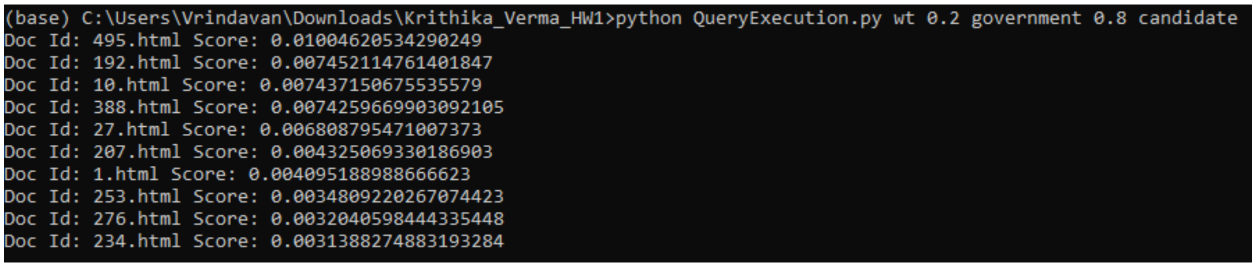
```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python QueryExecution.py wt 0.7 sick 0.2 building 0.9 syndrome
Doc Id: 99.html Score: 0.04989912126502535
Doc Id: 227.html Score: 0.03101415977858737
Doc Id: 75.html Score: 0.02791274380072864
Doc Id: 260.html Score: 0.012192961858212736
Doc Id: 271.html Score: 0.00890455699342203
Doc Id: 169.html Score: 0.005509094171196441 5
Doc Id: 226.html Score: 0.004643524846701621
Doc Id: 144.html Score: 0.003987534828675521
Doc Id: 64.html Score: 0.003931372366299808
Doc Id: 116.html Score: 0.003876769972323420 5
```

11. **Query = ‘government candidate’**



```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python QueryExecution.py government candidate
Doc Id: 495.html Score: 0.05023102671451245
Doc Id: 192.html Score: 0.037260573807009235
Doc Id: 10.html Score: 0.03718575337767789
Doc Id: 388.html Score: 0.03712983495154605
Doc Id: 27.html Score: 0.034043977355036864
Doc Id: 1.html Score: 0.02785375822276626
Doc Id: 207.html Score: 0.021625346650934515
Doc Id: 333.html Score: 0.019055042474090574
Doc Id: 253.html Score: 0.01740461013353721
Doc Id: 332.html Score: 0.014806907128531657
```

12. **Query = ‘wt 0.2 government 0.8 candidate’**



```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python QueryExecution.py wt 0.2 government 0.8 candidate
Doc Id: 495.html Score: 0.01004620534290249
Doc Id: 192.html Score: 0.007452114761401847
Doc Id: 10.html Score: 0.007437150675535579
Doc Id: 388.html Score: 0.007425966990309210 5
Doc Id: 27.html Score: 0.006808795471007373
Doc Id: 207.html Score: 0.004325069330186903
Doc Id: 1.html Score: 0.004095188988666623
Doc Id: 253.html Score: 0.003480922026707442 3
Doc Id: 276.html Score: 0.003204059844433544 8
Doc Id: 234.html Score: 0.003138827488319328 4
```