**Methodology and Algorithm:**

Here we build a command line retrieval engine on top of inverted files created in assignment 3.

A query is passed as system argument on the command line, top 10 documents that match the terms in the query is retrieved. The input to the cmdline is given as

e.g. Retrieve.py identity theft

**Preprocessing:**

The words identity and theft are taken as system arguments. These arguments are tokenized using word tokenizer. It then passed through functions which would remove stopwords, special characters (to make sure that characters typed by mistake, do not go through the retrieve process. Once the query is cleaned it is ready to go through retrieval.

As we have already created the dictionary file, which contains the term, number of documents in which the term occurs and the first position of the term in postings file. We have also created the postings file which contains the document id and the weight of the term in that document. This would be present at the position mentioned as the third entry in the dictionary file. As we have the number of documents that contains the term we know hoe many entries from the position is for a particular term.

A Boolean ‘OR’ similarity is used to get matches for the query term. This is chosen over a Boolean ‘AND’ because if ‘AND’ is used then we would get very few document matches, as the match success would require every term in the query to be present in a particular document for that document to be retrieved in the search, whereas for Boolean ‘OR’ any or all query terms present in a document would make the document a match.

**Step 1:**

The cleaned terms from the query is searched through for a match in the dictionary file, if a match is found the consecutive 2 entries after the term is saved in memory as frequency and position.

**Step 2:**

The postings file is opened, and we point the pointer to the position obtained in the previous step as the starting point. The end point would be the at (position + frequency). At every line in the postings file there is a document Id and term weight for a particular term. We store the values from the starting to end pointer in this file as a dictionary, which contains the document id as it’s key and term and term weights as values.

Once these values are collected for every term’s every document, a dictionary like below is obtained.

Retrieve.py federal legislation privacy

{'002': {'federal': '0.0754801882523086', 'legislation': '0.0806600068006253', 'privacy': '0.5313165271225008'}, '019': {'federal': '0.006932355649209566'}, '027': {'federal': '0.009245593647985413', 'legislation': '0.0038000274478011396'}, '052': {'federal': '0.055512620456282705'},…….}

All the 3 terms in the query are present in document 002.html, we form a dictionary with 002 as key and value is a collection of dictionaries with term as key (federal, legislation and privacy) and term weights as it’s values.

Once this is obtained, to populate documents with query words with high relevance, here it is high term weights, we calculate the sum of term weights .

This calculation is done for every document obtained in the previous step. We end up with a dictionary with document id (document name) as key and it’s term weights as values as shown. If a term does not exits, then a query no available message is printed.

**Result Evaluation:**

{'002': 0.6874567221754346, '019': 0.006932355649209566, '027': 0.013045621095786553, '052': 0.055512620456282705, '054': 0.035066211349316756, '106': 0.06224876759128133, '133': 0.03891735900729121, '150': 0.20413557449626976,…….}

For the terms federal, legislation and privacy in document 002 term weights are '0.0754801882523086, '0.0806600068006253' and '0.5313165271225008' , the total weight for the document 002 is the sum of these weights which is ‘0.6874567221754346’, which is exactly what is obtained in the final dictionary for 001.

The final document retrieval for federal legislation privacy is as below. The top 10 (the highest 10 weights )

002.html 0.6874567221754346

388.html 0.2833128204317538

333.html 0.24829699310296618

150.html 0.20413557449626976

246.html 0.17781127337201777

238.html 0.16657856111222036

240.html 0.1330287544866451

253.html 0.12492439621792988

382.html 0.11265130894278333

419.html 0.11127917612038993

We see that document 002 is scored the highest as we have all the 3 terms in this document leading to highest sum of weights.

As Boolean ‘OR’ is used we do get documents which contain only 1 or only 2 out of the total words in the query like for e.g.

'027': {'federal': '0.009245593647985413', 'legislation': '0.0038000274478011396'},

'388': {'federal': '0.27219058215780434', 'privacy': '0.011122238273949482'},

'419': {'federal': '0.11127917612038993'}

Document 388 is showing up in the top 10 results even if the word legislation is not present, this is because Boolean ‘OR’ is used and also the term weights of deferral and privacy is quiet high in this document. Similarly document 419 has only the term ‘federal’ in it , but still shows up in the top 10 results, if a stricter search is needed then we can use Boolean ‘AND’ and only like 002 containing all the terms would be populated and not the ones like 419 or 388.

**Time complexity:**

The time complexity of this algorithm as dictionary is used is o(n\*m) as we have a nested for loop, (outer loop) where each query term is looked for in dictionary file, (inner for loop) for each match in dictionary file, a match in postings file is looked for. Therefore, the run time depends on the number of terms in the query as well as the number of documents it occurs in. The limit of the first for loop is the number of terms in the query(n), the limit of the second for loop is the number of documents the query occurs in (m)

This could be reduced if a different data structure is used (like hash map) or if the match from dictionary file is saved in a separate file and looked up.

**The result for the queries in the assignment page with it’s are as below:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| diet | international affairs | Zimbabwe | computer network | hydrotherapy | identity theft |
| 018.html 0.30783816259722085  273.html 0.07423965084803699  252.html 0.03699466911088855  263.html 0.03538253447588365  278.html 0.03390954827544883  009.html 0.031065195349262713  050.html 0.014675098717917431  152.html 0.01187446511343224  353.html 0.008474495387445845 | 247.html 0.2183984327712473  161.html 0.1936344080547357  243.html 0.16553361364605806  133.html 0.13528822026165582  138.html 0.11499162456426368  125.html 0.1033452215860122  205.html 0.10026719860516754  197.html 0.0984150509797869  331.html 0.08829244141853973  143.html 0.08505047899058087 | Empty | 156.html 0.3003972800923021  060.html 0.2704563910483915  181.html 0.18042605059046574  223.html 0.13682738905989061  164.html 0.13136726088960643  140.html 0.1278151110097693  380.html 0.12706122030613315  275.html 0.11036438212052091  315.html 0.09825513651798348  135.html 0.0845115432767512 | 273.html 0.06568338823443066 | 379.html 0.0873819288166209  380.html 0.05945635863958928  301.html 0.05899863955267281  245.html 0.05727328511102279  328.html 0.04330775026985283  298.html 0.03357676873480773  292.html 0.026871202453194316  028.html 0.02611484099033487  332.html 0.022265261727930325  019.html 0.019566898011157324 |