REPORT ON INFORMATION RETREIVAL SYSTEM

**Ranking documents:**

The IR system uses Vector Space Model for ranking the documents with respect to a query. The query is supposed to be present in the same vector space as the documents.

For scoring, the system uses **cosine similarity** score on weighted terms.

For weighing terms, **tf-idf** weighting scheme is used (Formula: *tf \* log10 (N / dft)*).

It is important to note that, I calculated cosine similarity on the normalized weighted terms and **not on original weighted terms** as suggested in the lecture slides for *performance reasons*. So, the ranks may be different, but the relative order of documents remains unchanged.

**IR Engine design:**

At top – level, the program works with the following containers which are built while parsing different files:

1. Hashtable <String, Boolean> **stopWords** (Stores list of stop words)
2. Hashtable <String, Document> **documentIndex** (Stores list of documents)
3. Hashtable <String, Word> **wordIndex** (Stores list of words)
4. Hashtable <Integer, Document> **queryIndex** (Stores list of queries)
5. HashMap <Document, ArrayList<Document>> **manual\_entries** (Stores list of documents (value) which are relevant to document (key)

The following data structures were defined to supplement the top-level containers.

**Word:**

The data structure Word implemented in **Word.java** has the following fields.

* A word has a name.
* A count (number of occurrences in all documents).
* A Map which has Document name as Key (To track number of documents in which word has occurred) and a boolean value. (Is present or not)

Appropriate getters and setters are used in the class. We say two words are the same, if they share the same name which is implemented in equals () method.

**Document:**

A document implemented in **Document.java** has the following fields:

* An ID which stores unique number of the ID.
* A Name (Every document has a name)
* A Map (Key – Words present in that document, Value – weight of that word).

We say two documents are equal if they share the same name and ID. Note that both data structures implement Comparable Interface and override toString() methods.

**Weight:**

Each word in a document is associated with weight which holds the following information:

* Term frequency (Number of times word has occurred in a document)
* TF-IDF weight // tf - idf score.
* Normalized score // Eucilidean normal form.

The indexer is built on the fly which means that you update the forward and inverted index as you encounter each word. This would result in better performance as you are not parsing the file twice or looping through the entire collection each time.

The time taken to construct and evaluate queries is roughly around **5 to 6 seconds.**

(On a system running Windows OS, i5 processor and 12 GB RAM). I believe this performance can still be improved by parallelizing certain parts of code. Although, complete parallelization is not achievable as we are parsing text only in a certain context which means parallelization at file level is not achievable.

There are 32843 distinct words and 5368 documents in the ft911 folder.

The dependency diagram is as follows: A screenshot of a computer

Description generated with very high confidence

**System Performance against different parameters:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Number of top docs requested (k) | Query consideration | Number of Hit Docs (For 4 Queries in topics.txt) | Number of hit docs which are relevant | Total number of docs which are relevant | Precision | Recall |
| 0 (Considering scores > 0) | Title | 2557 | 16 | 22 | 0.625733% | 72.7272% |
| 0 | Title + Desc | 7625 | 19 | 22 | 0.249180% | 86.363636% |
| 0 | Title + Desc +Narrative | 11878 | 21 | 22 | 0.176797% | 95.4545% |
| 0 | Title + Narrative | 11255 | 21 | 22 | 0.1865% | 95.4545% |
| 10 | Title | 40 | 12 | 22 | 30% | 54.54% |
| 10 | Title + Desc | 40 | 11 | 22 | 27.5% | 50% |
| 10 | Title + Desc + Narrative | 40 | 11 | 22 | 27.5% | 50% |
| 10 | Title + Narrative | 40 | 14 | 22 | 35% | 63.6363% |
| 50 | Title | 200 | 14 | 22 | 7% | 63.63% |
| 50 | Title + Desc | 200 | 16 | 22 | 8% | 72.72% |
| 50 | Title + Desc + Narrative | 200 | 19 | 22 | 9.5% | 86.36% |
| 100 | Title | 400 | 14 | 22 | 3.5% | 63.63% |
| 100 | Title + Desc | 400 | 16 | 22 | 4% | 72.72% |
| 100 | Title + Desc + Narrative | 400 | 21 | 22 | 5.25% | 95.4545% |
| 1000 | Title | 4000 | 18 | 22 | 0.449% | 81.81% |
| 1000 | Title + Desc | 4000 | 18 | 22 | 0.449% | 81.81% |
| 1000 | Title + Desc + Narrative | 4000 | 21 | 22 | 0.525% | 95.4545% |
| 2000 | Title | 8000 | 19 | 22 | 0.2375% | 86.3636% |
| 2000 | Title + Desc | 8000 | 19 | 22 | 0.2375% | 86.3636% |
| 2000 | Title + Desc + Narrative | 8000 | 21 | 22 | 0.2625% | 95.4545% |

From analyzing different parameter settings, it seems like considering only title and narrative in the query and retrieving Top 10 documents gives decent results.