## INDEXER.PY

This program created simple inverted index of the given corpus.

Input: Corpus file in below format:
    # document_id
    tokens in the document

Output : File containing simple inverted index and token count of each

Document in the corpus file

**Language/ Version used :** Python 2.7

**Libraries used :** sys, re, sets, json

**Data Structures used :**

**doc_token_count** : dictionary to store the token counts for each doc_id.

Key : doc_id, value, value: no. of tokens in each doc

**doc_token_collection** : dictionary to store the tokens in each document

key : doc_id, value: tokens in the document

**inverted_index** : dictionary to store inverted index

key: word, value : term frequency of the word in each document

**ds** : list to dump json, ds=[inverted_index,doc_token_count]

**Pseudo code:**

```
1. Input file is read , split based on “#” and doc_token_collection and
   doc_token_count data structures are prepared.
2. To build an inverted index :
   For each document in doc_token_collection,
       For each word in a document,
           If first occurrence of document or word, initialize the inverted_index
           dictionary value, otherwise increase the counter for the word.
3. List ds is prepared and output file is written using json.dump
```

## BM25.PY

This program implement a small search engine using BM25 ranking model.

Input: index file : an inverted index of a corpus, queries_file: test queries

Output: Top 100 document IDs and their BM25 scores for each test query in the queries file in the following format:

query_id Q0 doc_id rank BM25_score system_name

**Language/ Version used :** Python 2.7

**Libraries used :** sys, re, sets, json, math, collections

**Data Structures used :**

**bm25 :** Dictionary to store bm25 scores,key : doc_id , value : bm25 score

ds, inverted_index, doc_token_count (o/p of indexer.py) as input

**Pseudo code:**

```
1. Load the inveted_index and token_count dictionaries from the index_file
2. Below is the formula for bm25_score :
   $\sum_{i\in Q} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)} \cdot \frac{(k_1+1)f_i}{K+f_i} \cdot \frac{(k_2+1)qf_i}{k_2+qf_i}$
   where the summation is now over all terms in the query;
   R, r = 0
   ni = number of documents in which term i found
   N = Total number of documents
   fi = frequency of term i in the document
   qfi = frequency of term i in the query
   $K = k_1((1-b)+b\cdot\frac{dl}{avdl})$
    k1=1.2, b=0.75, k2=100
3. Calculate N, avdl.
   For each query in the test queries:
       For each term in the query:
           Get tf i.e. term frequency of the term from inverted_index dict
           For each doc_id in tf :
               Calculate dl for the doc_id, qfi, bm25 score using above formula
               and update the bm25 dict
4. Sort the results based on bm25 scores and write the top 100 results into
   output file for each query.
```