COMP472 – Project 3 DEMO

Rhina Kim – 40130779

November9, 2022

# **This program is composed of 6 python files:**

* ***main.py***: pipeline execution of subproject I and II. It will begin with asking for user input and executes the subproject section which user has chosen.
* ***queries.py***: all the sample queries given by professor and custom queries created by me for experiment purpose resides in this file. Subproject II imports this file to extract all the available sample queries.
* ***reuters.py***: all about extracting Reuter’s collection and making list of *term-docID* pairs. (Extract, tokenize, make *term-docID* pairs from the raw texts)
* ***s1\_utils*.*py***: all the utility functions for Subproject I. It includes functions for:
  + Naïve indexer
  + SPIMI indexer
  + Remove duplicates for term-docID pairs
  + Check if result inverted index from Naïve and SPIMI indices are identical
* ***s2\_utils.py***: all the utility functions for Subproject II. It includes functions for:
  + *tftd* (number of term t in each documents d) computation
  + *Ld* (document length / total number of words in document d) computation
  + *L\_avg* (average document length for whole collection) computation
  + *N* (total number of documents) computation
  + dft (number of documents in collection that has term t) computation
  + *idft* (idft weighting of the query term t) computation
  + *RSVd* (given the document d, how relevant the term t is) computation
  + Intersection: Boolean search (AND)
  + Union: Boolean search (OR)
  + Input query processing (tokenization and removing stop words)
* ***file\_output.py***: function for outputting result to specified filename.

# Constants: (*main.py*, Line 8)

DIRECTORY = "../reuters21578\_extracted/"

OUTPUT\_DIRECTORY = "outputs\_test/"

* *DIRECTORY*: input directory to be read which stores all the *Reuters* corpus
* *OUTPUT\_DIRECTORY*: all pipeline outputs are written inside this folder

Sample Queries: (*queries.py*, Line 247)

sample\_queries1 = "America"

sample\_queries2 = "population"

sample\_queries3 = "South Korea and Japan"

sample\_queries4 = "Democrats' welfare and healthcare reform policies"

sample\_queries5 = "Drug company bankruptcies"

sample\_queries6 = "George Bush"

def get\_sample\_queries():

return [sample\_queries1, sample\_queries2, sample\_queries3, sample\_queries4, sample\_queries5]

Sample queries consist of queries that are given by our professor and other custom queries. It includes single queries, multiple queries, and queries with punctuations. This will help perform our project with any type of queries.

**Inputs:** (*main.py*, Line 247)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* P3 Assignment \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Using \"Reuters21578\" as a corpus (set of documents) to experiment...")

print("[1] Subproject(1-A): Experiment execution time for Naive indexer and SPIMI indexer")

print("[2] Subproject(1-B): Compile an inverted index for both Naive and SPIMI indexers (without compression)")

print("[3] Subproject(2-Ranked): Create an BM25 probabilistic search engine and provide rank results based on custom input queries")

print("[4] Subproject(2-Unranked): Create an Boolean search engine and provide unranked results based on custom boolean input queries")

print("[5] Run everything on above pipelines")

print("[Q] Exit")

while True:

choice = input(">>> Enter from option [1] to [5] or Q to exit: ")

if choice == '1': S1\_A()

elif choice == '2': S1\_B()

elif choice == '3': S2\_Ranked()

elif choice == '4': S2\_Unranked()

elif choice == '5':

S1\_A()

S1\_B()

S2\_Ranked()

S2\_Unranked()

break

elif choice == 'Q' or choice == 'q':

exit()

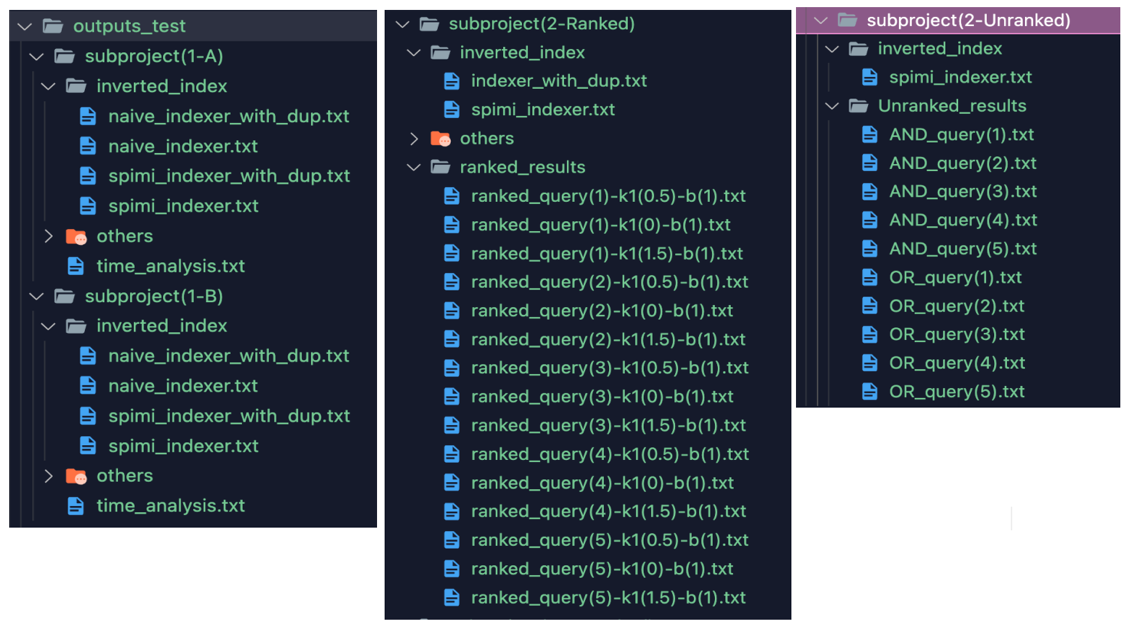
else:

print("Invalid option, enter again.")

We can choose to execute any subproject parts in this assignment by answering input prompt.

# **Outputs:**

For much simplified looking outputs, refer to ***Deliverables/P3/outputs\_test***



For subproject I (a), the execution time for Naïve indexer and SPIMI indexer is explained inside ***time\_analysis.txt***.

For subproject I (b), the compiled inverted index results (list of terms and its postings list) for both Naïve and SPIMI index are indicated under ***/inverted\_index*** directories with 2 versions: allowing duplicated *term-docID* and without duplicated *term-docID.*

For subproject II with ranked query search, all the results returned by BM25 are listed under ***/ranked\_results*** directories.

Query numbers are decided based on the *sample\_queries()* function in *query.py*.

The format of the BM25 result output is as following:

**Query Search:** # query string given

South Korea and Japan

**Query Tokens:** # tokenized query

['South', 'Korea', 'Japan']

# Tuning parameters

k1 = 0

b = 1

{documentID: score} # document ID ranked from its top score

-------------------

19377: 4.014072347869391

17207: 4.014072347869391

17083: 4.014072347869391

16964: 4.014072347869391

16957: 4.014072347869391

16935: 4.014072347869391

…

For subproject II with Boolean unranked retrieval, all the results returned by boolean query intersection (AND) and boolean query union (OR) are listed under ***/unranked\_results*** directories.

The format of the BM25 result output is as following:

**Query Search:**

South Korea and Japan

**Query Tokens:**

['South', 'Korea', 'Japan']

1568 union postings (OR) found.

Ranked list of retrieved documentIDs:

{documentID: # queries found}

-----------------------------

{

"903": 3,

"1385": 3,

"1499": 3,

"1656": 3,

"1772": 3,

"1902": 3,

…

}

# **Outputs on Command Prompt:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* P3 Assignment \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Using "Reuters21578" as a corpus (set of documents) to experiment...

[1] Subproject(1-A): Experiment execution time for Naive indexer and SPIMI indexer

[2] Subproject(1-B): Compile an inverted index for both Naive and SPIMI indexers (without compression)

[3] Subproject(2-Ranked): Create an BM25 probabilistic search engine and provide rank results based on custom input queries

[4] Subproject(2-Unranked): Create a Boolean search engine and provide unranked results based on custom Boolean input queries

[5] Run everything on above pipelines

[Q] Exit

>>> Enter from option [1] to [5] or Q to exit:

**===================== Subproject(1-A) =====================**

(1) Collecting Reuters files...

(2) Reading and extracting documents from Reuters files...

\* Reading ../reuters21578\_extracted/reut2-004.sgm \*

- 1000 documents have been successfully extracted.

(3) Tokenizing all documents...

- Number of term-docId pairs: 129102

(4) Reduced the number of terms in test corpus to 10000.

**Naive Indexer:**

\* Removing duplicates...

- 4008 postings have been removed due to duplicate.

\* Sorting list of term-docId pairs...

\* Creating inverted index...

--> Total number of distinct terms(type) in dictionary: 2816

**==> Execution Time: 0.005676984786987305**

**SPIMI Indexer:**

\* Removing duplicates...

- 4008 postings have been removed due to duplicate.

\* Creating inverted index...

\* Sorting inverted index hash-table...

--> Total number of distinct terms(type) in dictionary: 2816

**==> Execution Time: 0.005117177963256836**

**Naive Indexer allowing duplicate term-docID pairs:**

\* Sorting list of term-docId pairs...

\* Creating inverted index...

--> Total number of distinct terms(type) in dictionary: 2816

**==> Execution Time: 0.0073621273040771484**

**SPIMI Indexer allowing duplicate term-docID pairs:**

\* Creating inverted index...

\* Sorting inverted index hash-table...

--> Total number of distinct terms(type) in dictionary: 2816

**==> Execution Time: 0.005326032638549805**

Results successfully outputted.

Both Naive Indexer and Spimi Indexer results are **identical.**

**===================== Subproject(1-B) =====================**

(1) Collecting Reuters files...

(2) Reading and extracting documents from Reuters files...

\* Reading ../reuters21578\_extracted/reut2-004.sgm \*

\* Reading ../reuters21578\_extracted/reut2-010.sgm \*

\* Reading ../reuters21578\_extracted/reut2-011.sgm \*

\* Reading ../reuters21578\_extracted/reut2-005.sgm \*

…

- 21578 documents have been successfully extracted.

(3) Tokenizing all documents...

- Number of term-docId pairs: 2597951

**Naive Indexer:**

\* Removing duplicates...

- 986711 postings have been removed due to duplicate.

\* Sorting list of term-docId pairs...

\* Creating inverted index...

--> Total number of distinct terms(type) in dictionary: 58405

**==> Execution Time: 5.011843204498291**

**SPIMI Indexer:**

\* Removing duplicates...

- 986711 postings have been removed due to duplicate.

\* Creating inverted index...

\* Sorting inverted index hash-table...

--> Total number of distinct terms(type) in dictionary: 58405

**==> Execution Time: 1.6352832317352295**

**Naive Indexer allowing duplicate term-docID pairs:**

\* Sorting list of term-docId pairs...

\* Creating inverted index...

--> Total number of distinct terms(type) in dictionary: 58405

**==> Execution Time: 3.6659250259399414**

**SPIMI Indexer allowing duplicate term-docID pairs:**

\* Creating inverted index...

\* Sorting inverted index hash-table...

--> Total number of distinct terms(type) in dictionary: 58405

**==> Execution Time: 1.174767017364502**

Results successfully outputted.

Both Naive Indexer and Spimi Indexer results are **identical.**

**===================== Subproject(2-Ranked) - RSVd =====================**

(1) Collecting Reuters files...

(2) Reading and extracting documents from Reuters files...

\* Reading ../reuters21578\_extracted/reut2-004.sgm \*

\* Reading ../reuters21578\_extracted/reut2-010.sgm \*

\* Reading ../reuters21578\_extracted/reut2-011.sgm \*

…

- 21578 documents have been successfully extracted.

(3) Tokenizing all documents...

- Number of term-docId pairs: 2597951

S2 SPIMI Indexer:

\* Removing duplicates...

- 986711 postings have been removed due to duplicate.

\* Creating inverted index...

\* Sorting inverted index hash-table...

--> Total number of distinct terms(type) in dictionary: 58405

==> Execution Time: 1.6821367740631104

S2 Indexer allowing duplicate term-docID pairs:

\* Creating inverted index with duplicates...

\* Sorting inverted index...

--> Total number of distinct terms(type) in dictionary: 58405

Computing tftd...

Computing Ld...

Computing N...

Computing Ldavg...

Processing input queries...

Computing RSVd for query(1) with (k1=0, b=1)...

Computing RSVd for query(2) with (k1=0, b=1)...

Computing RSVd for query(3) with (k1=0, b=1)...

Computing RSVd for query(4) with (k1=0, b=1)...

Computing RSVd for query(5) with (k1=0, b=1)...

Results successfully outputed.

**===================== Subproject(2-Unranked) - Boolean =====================**

(1) Collecting Reuters files...

(2) Reading and extracting documents from Reuters files...

\* Reading ../reuters21578\_extracted/reut2-004.sgm \*

\* Reading ../reuters21578\_extracted/reut2-010.sgm \*

\* Reading ../reuters21578\_extracted/reut2-011.sgm \*

\* Reading ../reuters21578\_extracted/reut2-005.sgm \*

…

- 21578 documents have been successfully extracted.

(3) Tokenizing all documents...

- Number of term-docId pairs: 2597951

S2 SPIMI Indexer:

\* Removing duplicates...

- 986711 postings have been removed due to duplicate.

\* Creating inverted index...

\* Sorting inverted index hash-table...

--> Total number of distinct terms(type) in dictionary: 58405

==> Execution Time: 1.6330490112304688

Processing input queries...

**Computing Boolean Search with Intersection (AND) for query(1)...**

**- There are not enough query tokens to perform boolean intersection.**

**Computing Boolean Search with Union (OR) for query(1)...**

**- There are not enough query tokens to perform boolean union.**

**Computing Boolean Search with Intersection (AND) for query(2)...**

**- There are not enough query tokens to perform boolean intersection.**

**Computing Boolean Search with Union (OR) for query(2)...**

**- There are not enough query tokens to perform boolean union.**

**Computing Boolean Search with Intersection (AND) for query(3)...**

**- 59 intersection postings (AND) found.**

**Computing Boolean Search with Union (OR) for query(3)...**

**- 1568 union postings (OR) found.**

**Computing Boolean Search with Intersection (AND) for query(4)...**

**- 0 intersection postings (AND) found.**

**Computing Boolean Search with Union (OR) for query(4)...**

**- 576 union postings (OR) found.**

**Computing Boolean Search with Intersection (AND) for query(5)...**

**- 0 intersection postings (AND) found.**

**Computing Boolean Search with Union (OR) for query(5)...**

**- 5275 union postings (OR) found.**

Results successfully outputed.

# **BF25 Implementation:**

*# Number of terms t in each documents d*

def TFtd(index):

index\_TFtd = {}

for term, postings in index.items():

term\_freq\_dict = dict(Counter(postings))

*# sort by docIDs*

sorted\_term\_freq\_dict = dict(sorted(term\_freq\_dict.items(), key=lambda t: t[0]))

*# append to index*

index\_TFtd[term] = sorted\_term\_freq\_dict

*# sort by terms*

index\_TFtd = dict(sorted(index\_TFtd.items(), key=lambda x: x[0]))

*# return dictionary of {term: {docID1: term\_freq, docID2: term\_freq, ...}, ...}*

return index\_TFtd

def get\_TFtd\_val(index, docID, input\_term):

TFtd\_val = 0

for term, postings in index.items():

if term == input\_term:

tf\_dict = dict(Counter(postings))

if docID in tf\_dict:

TFtd\_val = tf\_dict[docID]

return TFtd\_val

*# Document Length (Total number of words in document)*

def LD(documents):

dict\_LD = {}

*# compute number of words in each documents (document length)*

for docID, text in documents.items():

*# get tokens*

tokens = S2\_get\_tokens\_list(text)

*# assign freq tokens to docID*

dict\_LD[docID] = len(tokens)

*# sort by docIDs*

dict\_LD = dict(sorted(dict\_LD.items(), key=lambda x: x[0]))

*# return dictionary of {docID: docLength, ...}*

return dict\_LD

def get\_LD\_val(documents, input\_docID):

LD\_val = 0

for docID, text in documents.items():

*# compute number of words in given document ID (document length)*

if docID == input\_docID:

*# get tokens*

tokens = S2\_get\_tokens\_list(text)

*# get freq of tokens val*

LD\_val = len(tokens)

return LD\_val

*# Average document length for the whole collection*

def LD\_avg\_compute(documents, N):

*# total number of words in the collection*

total\_num\_words = 0

*# count number of words in each documents (document length)*

for text in documents.values():

*# get tokens*

tokens = S2\_get\_tokens\_list(text)

*# add to total number of words in the collection*

total\_num\_words += len(tokens)

*# compute LD Average*

return total\_num\_words / N

*# Total number of documents in the collection*

def N\_compute(documents):

return len(documents)

*# Number of documents in collection that certain term occurs in*

def DFt\_compute(index, term):

DFt\_val = 0

if term in index:

DFt\_val = len(index[term])

return DFt\_val

*# idf weighting of the query term present*

def iDFt\_compute(N, DFt):

return log(N/DFt)

*# Given the document, how relevant the term is*

def RSVd\_compute(documents, index, query\_tokens, variables, k1=0, b=1):

RSVd\_val = 0

RSVd\_dict = {}

*# get all necessary variables*

N\_val = variables['N']

LD\_avg\_val = variables['Lavg']

dict\_LD = variables['Ld']

index\_tftd = variables['tftd']

for docID in documents:

*# get Ld value*

LD\_val = dict\_LD[docID]

for token in query\_tokens:

*# if tftd exists (tftd != 0)*

if token in index\_tftd and docID in index\_tftd[token]:

*# get tftd value*

TFtd\_val = index\_tftd[token][docID]

*# get dft value*

DFt\_val = DFt\_compute(index, token)

*# compute weighting*

**iDFt = iDFt\_compute(N\_val, DFt\_val)**

**x = (k1 + 1) \* TFtd\_val**

**y = k1 \* ((1 - b) + b \* (LD\_val / LD\_avg\_val)) + TFtd\_val**

**RSVd\_val += log(iDFt) \* (x / y)**

*# collect RSVd scores into dictionary*

RSVd\_dict[docID] = RSVd\_val

RSVd\_val = 0 *# initialize*

*# Sort the result from highest rank*

RSVd\_dict = dict(sorted(RSVd\_dict.items(), key=lambda x: (x[1], x[0]), reverse=True))

return RSVd\_dict

# **Boolean Search Implementation – Intersection (AND):**

***# Boolean search (AND)***

def intersection(query\_tokens, index):

postings\_total = []

common\_postings = []

message = ""

if len(query\_tokens) >= 2:

*# Get postings list for every tokens first*

for token in query\_tokens:

if token in index:

postings = index[token]

postings\_total.append(postings)

*# Get Intersection*

common\_postings = sorted(list(set.intersection(\*[set(postings) for postings in postings\_total])))

*# If intersection postings found*

if common\_postings:

message = str(len(common\_postings)) + " intersection postings (AND) found." *# output*

else:

message = "0 intersection postings (AND) found." *# output*

else:

message = "There are not enough query tokens to perform Boolean intersection."

*# make dictionary to store info*

AND\_info = {'postings': common\_postings, 'message': message}

return AND\_info

# **Boolean Search Implementation – Union (OR):**

***# Boolean search (OR)***

def union(query\_tokens, index):

postings\_total = []

union\_postings = []

union\_postings\_ranked = {}

message = ""

*# If enough query tokens exist*

if len(query\_tokens) >= 2:

*# Get postings list for every tokens first*

for token in query\_tokens:

if token in index:

postings = index[token]

postings\_total.append(postings)

*# Get Union*

union\_postings = sorted(list(set.union(\*[set(postings) for postings in postings\_total])))

*# If union postings found*

if union\_postings:

*# Compute rank of how many keywords union docs contain*

for docID in union\_postings:

freq\_docID = sum(postings.count(docID) for postings in postings\_total)

union\_postings\_ranked[docID] = freq\_docID

*# Get Union Rank*

union\_postings\_ranked = dict(sorted(union\_postings\_ranked.items(), key=lambda x:x[1], reverse=True))

*# Output*

message = str(len(union\_postings)) + " union postings (OR) found."

else:

*# Output*

message = "0 union postings (OR) found."

else:

message = "There are not enough query tokens to perform boolean union."

*# make dictionary to store info*

OR\_info = {'postings': union\_postings, 'ranked\_postings': union\_postings\_ranked, 'message': message}

return OR\_info