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.

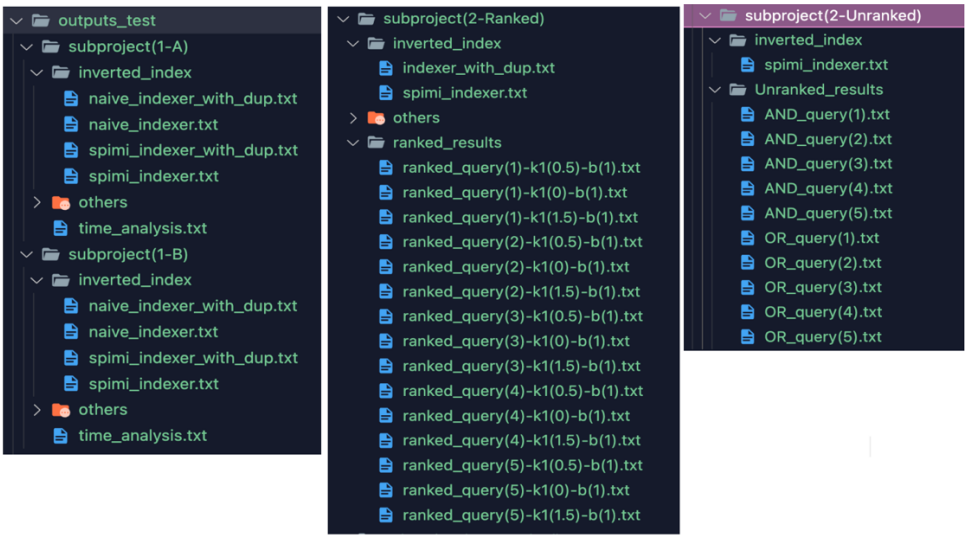
Inputs: (*main.py*, Line 247)

```
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
```