Tanuj Gupta:

Code: Github

A) Design decisions and high-level software architecture

Teck Stack

The language used: **java**
Project management tool: **maven**
IDE: **Intellij**
Version control: **Github**

Programming tools or libraries

Lucene version 9.1.0 ->
Lucene-core, lucene-analyzers-common, lucene-queryparser

Design decisions and high-level software architecture

The program is written following object-oriented principles to make it more manageable and readable. The program is divided into seven packages:

1. Parsing:

This package parses query, document, and relevance file. To hold different properties of each query and document, I have created two classes(value objects), namely ParsedDocument and ParsedQuery

2. VO:

This package defines all the classes to hold objects. It follows popular design principle of the value object. It has 3 main classes:
a) ParsedQuery: to hold properties of each query
b) ParsedDocument: to hold properties of each parsed document
c) LogFileResultRow: to hold a single row of final results in the format defined for logfile

3. SearchEngine:

It is the main package which has 3 classes:

a) **Indexer:** This class indexes all the parsed documents in byteBufferIndex (in

RAM). It uses lucenes standard analyzer that removes stop worlds and perform stemming and lemmetization

b) **Reader and Searcher:** This class is used to query index created by indexer. In constructor, I also calculate document Frequencuies for all the terms to be used while ranking document using my own algorithm.

4. Ranking:

This package defines all the 5 ranking algorithms, namely (Boolean, TF, TF-IDF, Relevance feedback, and Own algorithm)

5. VoTransformation:

This package is used for conversion among different class objects. For instance, function convertParsedDocumentToDocument function inside VoTransformations Class is used to convert my own created ParsedDocument object to lucenes document object.

6. Run:

This package is used to run everything. It contains main method, where every java program starts. This is what make all the required function calls.

7. Comparator:

This package defines comparison functions for objects necessary to be able to sort them.

B) Implementation of ranking algorithms:

1. Boolean: Program uses used lucenes booleanSimilarity for Boolean ranking
2. Tf-Idf: Program uses used lucenes classicSimilarity that implements interface TfIDFSimilarity for Boolean Tf-Idf:
3. Tf: I have created my own class called TfSimilarity for this. TfSimilarity also implements TfIDFSimilarity interface like lucenes classicSimilarity class. The only difference is that the method that returns idf is overridden and made to return 1 in TfSimilarity.
4. Relevance Feedback: For this, I have used Pseudo_Relevance_Feedback. The program first fetches the top 5 documents using lucenes Tf-Idf method. Then the original query is expanded using these 5 documents. This query is then used to fetch the top 50 final documents using lucenes Tf-Idf method.

C) Strengths and weaknesses of your design:

Strengths:

1. The program uses a well-established Lucene library to build the search engine. It gives a lot of in-built functionalities that we can use to add more enhanced features to the current code.
2. Java follows an object-oriented pattern, which makes application development very understandable and developer-friendly.

Weaknesses:

1. Using Lucene library also comes with a big disadvantage:
   Here the developer has less control over so many aspects. Understanding the internals of Lucene is very difficult and may hinder writing one's own optimization on so many aspects.
   Lucene also requires a lot of knowledge of its classes to be able to use them.
2. Java doesn't provide many Machine learning modules and libraries. So, adding machine learning is challenging when using Java. In that case, the combination of python and java seems more suitable, like pylucene.

D) Customized new algorithm:

- The algorithm is the combination of pseudo-relevance-feedback, relevance-feedback, and optimized tf-idf
- It follows these three steps in order:
    1. **Relevance-feedback:**
       1.1) Randomly pick two relevant documents for a query from the relevance-file shared
       1.2) Use these two documents to expand the query
    2. **Pseudo-relevance-feedback:**
       2.1) Now Use the expanded query generated in 1st step to fetch the top 5 documents using normal tf-idf.
       2.2) Among these five documents, select the ones that are different from the two documents used in the first step.
       2.3) Use these selected documents to further expand the query.
    3. **Optimized tf-idf**

- Optimized tf-idf uses properties of a query to enhance ranking. Every query in the query file given has three properties, namely, query_number(num), title, and description(desc).
- Instead of giving equal weightage to every term in a query, Optimized tf-idf gives more weightage to terms in the title, unlike normal tf-idf.
- To give extra weights to the terms in the title, the algorithm simply multiples constant C with term_frequency to give the term more weight. Value for constant C is decided as below:

a) If the term is present in both the title and description, multiply the term frequency by 3, i.e., constant C=3
b) If the term is present in the title, multiply the term frequency by 2, i.e, constant C=3
c) If the term is only present in the description, constant C=1. That is, the term frequency is used as-is.

E) Experimental results:

1. **Boolean:**

```
tanujgupta@Tanujs-MacBook-Air trec_eval %  ./trec_eval
s/logfileBoolean.txt
runid                   all     Boolean
num_q                   all     63
num_ret                 all     3150
num_rel                 all     3205
num_rel_ret             all     439
map                     all     0.0622
gm_map                  all     0.0110
Rprec                   all     0.1209
bpref                   all     0.1609
recip_rank              all     0.4682
iprec_at_recall_0.00    all     0.5020
iprec_at_recall_0.10    all     0.2271
iprec_at_recall_0.20    all     0.1073
iprec_at_recall_0.30    all     0.0492
iprec_at_recall_0.40    all     0.0206
iprec_at_recall_0.50    all     0.0198
iprec_at_recall_0.60    all     0.0164
iprec_at_recall_0.70    all     0.0000
iprec_at_recall_0.80    all     0.0000
iprec_at_recall_0.90    all     0.0000
iprec_at_recall_1.00    all     0.0000
P_5                     all     0.2603
P_10                    all     0.2270
P_15                    all     0.2063
P_20                    all     0.1889
P_30                    all     0.1619
P_100                   all     0.0697
P_200                   all     0.0348
P_500                   all     0.0139
P_1000                  all     0.0070
tanujgupta@Tanujs-MacBook-Air trec_eval %
```

2. **Tf:**

```
tanujgupta@Tanujs-MacBook-Air trec_eval %  ./trec_eval
s/logfileTf.txt
runid                   all     Tf
num_q                   all     63
num_ret                 all     3150
num_rel                 all     3205
num_rel_ret             all     505
map                     all     0.0658
gm_map                  all     0.0137
Rprec                   all     0.1337
bpref                   all     0.1701
recip_rank              all     0.4630
iprec_at_recall_0.00    all     0.4989
iprec_at_recall_0.10    all     0.2498
iprec_at_recall_0.20    all     0.1251
iprec_at_recall_0.30    all     0.0507
iprec_at_recall_0.40    all     0.0350
iprec_at_recall_0.50    all     0.0169
iprec_at_recall_0.60    all     0.0031
iprec_at_recall_0.70    all     0.0000
iprec_at_recall_0.80    all     0.0000
iprec_at_recall_0.90    all     0.0000
iprec_at_recall_1.00    all     0.0000
P_5                     all     0.2698
P_10                    all     0.2429
P_15                    all     0.2233
P_20                    all     0.2087
P_30                    all     0.1910
P_100                   all     0.0802
P_200                   all     0.0401
P_500                   all     0.0160
P_1000                  all     0.0080
tanujgupta@Tanujs-MacBook-Air trec_eval %
```

3. **Tf-IDF:**

```
tanujgupta@Tanujs-MacBook-Air trec_eval %  ./trec_eval
s/logfileTfidfBoth.txt
runid                   all     tfidf-Both
num_q                   all     63
num_ret                 all     3150
num_rel                 all     3205
num_rel_ret             all     709
map                     all     0.1101
gm_map                  all     0.0363
Rprec                   all     0.1958
bpref                   all     0.2447
recip_rank              all     0.5549
iprec_at_recall_0.00    all     0.6105
iprec_at_recall_0.10    all     0.3992
iprec_at_recall_0.20    all     0.2392
iprec_at_recall_0.30    all     0.1447
iprec_at_recall_0.40    all     0.0869
iprec_at_recall_0.50    all     0.0215
iprec_at_recall_0.60    all     0.0048
iprec_at_recall_0.70    all     0.0020
iprec_at_recall_0.80    all     0.0000
iprec_at_recall_0.90    all     0.0000
iprec_at_recall_1.00    all     0.0000
P_5                     all     0.3587
P_10                    all     0.3317
P_15                    all     0.3122
P_20                    all     0.2992
P_30                    all     0.2624
P_100                   all     0.1125
P_200                   all     0.0563
P_500                   all     0.0225
P_1000                  all     0.0113
tanujgupta@Tanujs-MacBook-Air trec_eval %
```

4. **Relevance Feedback:**

```
[tanujgupta@Tanujs-MacBook-Air trec_eval %  ./trec_eval /
s/logfilePRF.txt

runid                   all     PseudoRelevanceFeedback
num_q                   all     63
num_ret                 all     3150
num_rel                 all     3205
num_rel_ret             all     391
map                     all     0.0632
gm_map                  all     0.0139
Rprec                   all     0.1200
bpref                   all     0.1438
recip_rank              all     0.5343
iprec_at_recall_0.00    all     0.5786
iprec_at_recall_0.10    all     0.2630
iprec_at_recall_0.20    all     0.1170
iprec_at_recall_0.30    all     0.0315
iprec_at_recall_0.40    all     0.0139
iprec_at_recall_0.50    all     0.0095
iprec_at_recall_0.60    all     0.0000
iprec_at_recall_0.70    all     0.0000
iprec_at_recall_0.80    all     0.0000
iprec_at_recall_0.90    all     0.0000
iprec_at_recall_1.00    all     0.0000
P_5                     all     0.3397
P_10                    all     0.2825
P_15                    all     0.2286
P_20                    all     0.2048
P_30                    all     0.1667
P_100                   all     0.0621
P_200                   all     0.0310
P_500                   all     0.0124
P_1000                  all     0.0062
tanujgupta@Tanujs-MacBook-Air trec_eval %
```

5. **Own:**

```
tanujgupta@Tanujs-MacBook-Air trec_eval %  ./trec_eval
s/logfileOwn.txt

runid                   all     Own
num_q                   all     63
num_ret                 all     3150
num_rel                 all     3205
num_rel_ret             all     708
map                     all     0.1119
gm_map                  all     0.0377
Rprec                   all     0.1953
bpref                   all     0.2446
recip_rank              all     0.5769
iprec_at_recall_0.00    all     0.6281
iprec_at_recall_0.10    all     0.4097
iprec_at_recall_0.20    all     0.2544
iprec_at_recall_0.30    all     0.1496
iprec_at_recall_0.40    all     0.0856
iprec_at_recall_0.50    all     0.0180
iprec_at_recall_0.60    all     0.0046
iprec_at_recall_0.70    all     0.0019
iprec_at_recall_0.80    all     0.0000
iprec_at_recall_0.90    all     0.0000
iprec_at_recall_1.00    all     0.0000
P_5                     all     0.3587
P_10                    all     0.3286
P_15                    all     0.3037
P_20                    all     0.2825
P_30                    all     0.2608
P_100                   all     0.1124
P_200                   all     0.0562
P_500                   all     0.0225
P_1000                  all     0.0112
tanujgupta@Tanujs-MacBook-Air trec_eval %
```

Patterns observed:

- Recall increases with rank.
- Precision decrease with the rank.
- Pseudo relevance and relevance feedback seem to have higher precision in the selection of the top documents in comparison to other algorithms.
- Although algorithms have similar performance when selecting higher-ranked documents.
- The terms in the title seem to have more importance than the terms in the description.
- The performance of the five algorithms is as follows:

    Own > Relevance feedback > Pseudo relevance feedback > Tf-Idf > Tf > Boolean

F) What you have learned from this assignment:

1. **Implementing search engine:**
    a. Learned how to configure a Lucene project in Java
    b. Leaned to create both filesystem and ram index using Lucene
    c. Learned to update Lucene index to hold extra useful values such as document frequency, term vectors, positional index etc
    d. Learned to query from the index using Lucene
    e. Learned to use various ranking algorithms given by Lucene to rank fetched documents
    f. Learned to update/override Lucene classes to create own ranking algorithms

2. **Application development design principles:**
    a. Learned to create a maven project in Java.
    b. Learned to use SOLID design principles.
    c. Learned various concepts like value object, POJO(Plain old java objects) etc while developing search engine application
    d. Learned basic oops concepts like polymorphism, inheritance etc

3. **Evaluation:**
    a. Got on hands experience with evaluating search engines.
    b. Learned to use reac_eval to evaluate search engine performance.
    c. Learned to use various evaluation results to come up with a better algorithm.
    d. Learned various patterns of recall/precision shown by search engine ranking algorithms.