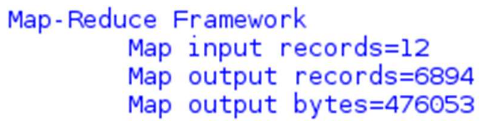
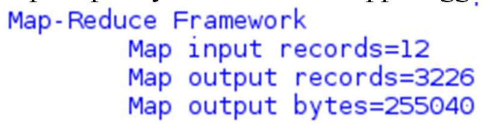
Functionality
Part 1
1. Case folding
Improvement to indexing: Removing tuples between which the only difference is capitalization, therefore saves space of the extra key and reduce size of the whole inverted index.
Limitations: Some special terms contain upper case letters by themselves. Un-capitalize them will get them not being able to be distinguished from normal lower phrases that contains same word(s), therefore cause difficulties in later processing.

2. In-mapper aggregation
Improvement to indexing: Reduces size of mapper output as space formerly used for duplicate tuples is saved now. Of course, size of input of reduce function is also reduced and it greatly helps reduces running time of reducer.



```
Map-Reduce Framework
        Map input records=12
        Map output records=6894
        Map output bytes=476053
```

pic 1- map output byte without in-mapper aggregation



```
Map-Reduce Framework
        Map input records=12
        Map output records=3226
        Map output bytes=255040
```

pic 2- map output byte with in-mapper aggregation
Before this is implemented, running time is around 2.6~3.6s; after it's implemented, running time is around 2.6~3.1s. Theoretically, there will be improvements in running time as map output bytes are almost halved as can be seen from pic 1 and pic 2, however, this improvements is not really obvious as size of this document set is still small and a majority of running time is used on data I/O rather than actual data processing.

3. Stopword removal
Improvement to indexing: Reduces size of mapper output and eventually the final inverted index because all spaces for saving information of those words are saved. Also, it makes the inverted index be more information-intensive, as a larger percent of words in the inverted index is more helpful for selecting a document.

4. Stemming
Improvement to indexing: Helps with information concentration in inverted index as it merges information of all variances together under the tuple of stem word. Of course, size of inverted index is also reduced as extra keys that are variants of a stem will disappear.
Limitations: Current Porter stemmer cannot well understand context, causing the problem of over-stemming and under-stemming.

5. Numerical and special expressions handling
Improvement to indexing: in most cases these expressions merely appear in a small number of documents, thus being great helper for locating what we want. By keeping them we have better indicator of documents.
Limitations: Currently I am using regex to filter out unwanted strings and preserve tokens I need. It gives bi-product like meaningless strings that composed of letters and digits. Better way to further filter these out is to be discovered.

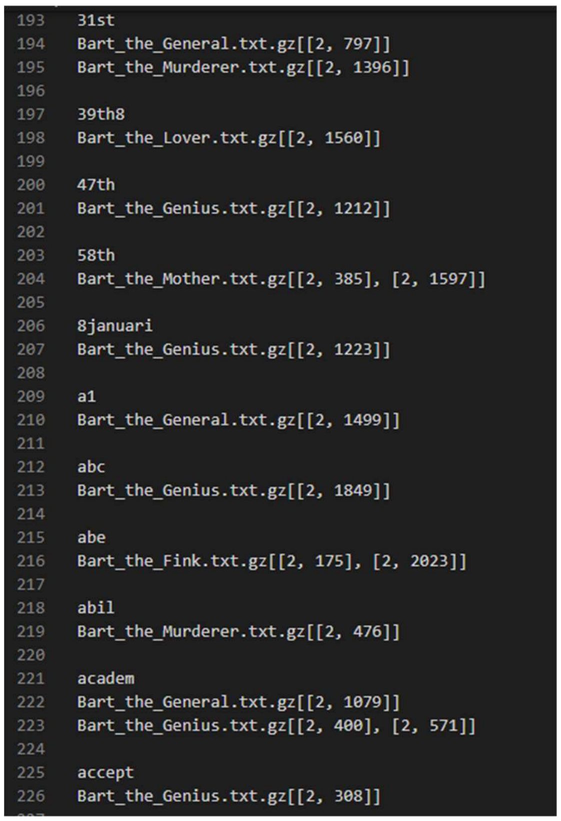
Part 2
1. Position indexing is implemented.
Improvement to indexing: With position of occurrence recorded, we can directly locate a token

rather than going through the whole document to look for it. It helps with calculation of tf-idf, as number of indices in a document of a token is its term frequency in the document. Also, it helps find multi-word terms. If we query for such a term, for every document, we can see that if required single-word sub-terms are next to each other and are in the same sequence as they appear in the required multi-word term.
Limitations: When files are larger than a split, then sequence of each split can no longer be preserved by mappers of the file. In this case, current method of counting line number and column number will no longer work. In that case, a global program for indexing should be implemented before mapreduce jobs start and indexing information should be sent to mappers. Also, in my implementation, indices are saved as strings, which cannot be used directly. When getting an index, extra steps are to take to convert the string to a pair of integers.

2. tf-idf calculation is implemented.
Improvement: With tf-idf, ranked retrieval can be done, allowing user to use human language queries and number of results is controllable.
Limitations: tf-idf assumes that terms are independent in documents, which is usually not the case. Also, currently in my implementation, tf-idf score is stored together with indices, which is not ideal. Ideally, value for each key should be a tuple with 3 elements: (document name, list of indices, tf-idf score).

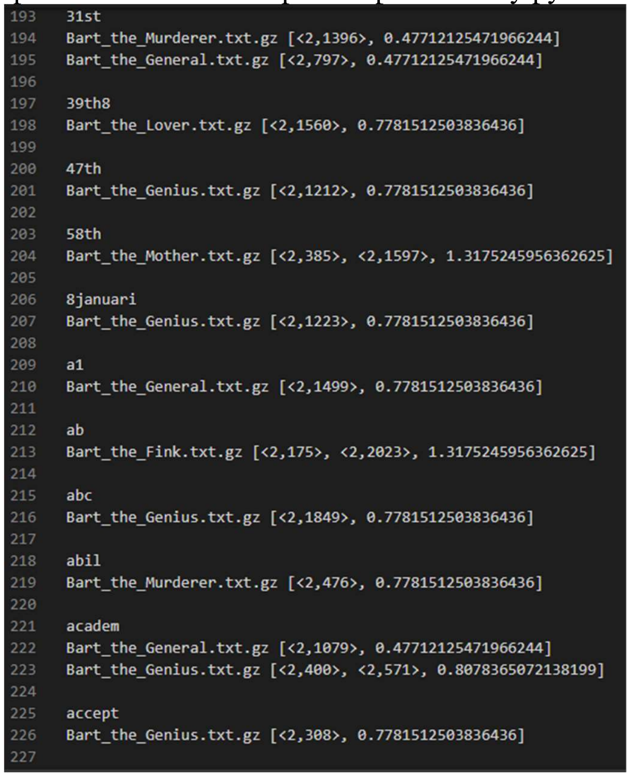
Performance
iii.
Inverted index summarization pattern and filtering pattern, characterized by stopword removal, are used in in the code.
Summarization patterns are a more straightforward pattern than any other patterns, as grouping data together is the core function of the Mapreduce paradigm. This means we can make fully use of the framework and there is relatively fewer work left for implementation, thus program running can be sped up. With the help of in-mapper aggregation, further acceleration can be achieved.
In filtering pattern, we only need a subset of data. In this coursework, we only care about those tokens that have actual meanings and somehow represent the document. So I am not further processing stopwords and single characters. This helps scale down workload of both mapper and reducer function and help speed up the program.
iv.
Lack of custom partitioner may hinder performance, because inverted indexes are rather susceptible to hot spots in index keys. If we are facing a larger document set, some tokens may have much more tuples than others after lemmatization due to their various forms. In this case, the reducer for these tokens are going to be particularly busy. This can slow down the entire job as some reducers will be more time-consuming than others.

Evaluation
Evaluation of results is done by reimplement in python the functionalities I have achieved in mapreduce to check if results produced will remain the same if the whole progress is done linearly. Refactoring from mapreduce to python is achievable as all functions we use in Java have equivalent substitutions in python. Comparison on python output file and log.txt produced by my mapreduce program shows that relevant information about tuples are the same.
Below are extractions of the 2 files.



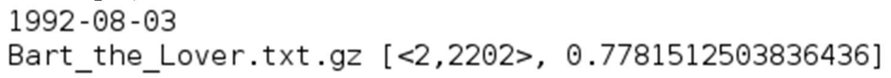
```
31st
Bart_the_General.txt.gz[[2, 797]]
Bart_the_Murderer.txt.gz[[2, 1396]]

39th8
Bart_the_Lover.txt.gz[[2, 1560]]

47th
Bart_the_Genius.txt.gz[[2, 1212]]

58th
Bart_the_Mother.txt.gz[[2, 385], [2, 1597]]

8januari
Bart_the_Genius.txt.gz[[2, 1223]]

a1
Bart_the_General.txt.gz[[2, 1499]]

abc
Bart_the_Genius.txt.gz[[2, 1849]]

abe
Bart_the_Fink.txt.gz[[2, 175], [2, 2023]]

abil
Bart_the_Murderer.txt.gz[[2, 476]]

academ
Bart_the_General.txt.gz[[2, 1079]]
Bart_the_Genius.txt.gz[[2, 400], [2, 571]]

accept
Bart_the_Genius.txt.gz[[2, 308]]
```

pic 3-extraction of output file produced by python



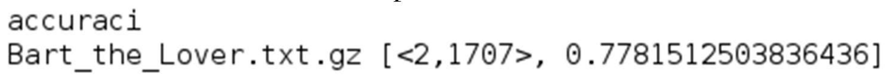
```
31st
Bart_the_Murderer.txt.gz [<2,1396>, 0.47712125471966244]
Bart_the_General.txt.gz [<2,797>, 0.47712125471966244]

39th8
Bart_the_Lover.txt.gz [<2,1560>, 0.7781512503836436]

47th
Bart_the_Genius.txt.gz [<2,1212>, 0.7781512503836436]

58th
Bart_the_Mother.txt.gz [<2,385>, <2,1597>, 1.3175245956362625]

8januari
Bart_the_Genius.txt.gz [<2,1223>, 0.7781512503836436]

a1
Bart_the_General.txt.gz [<2,1499>, 0.7781512503836436]

ab
Bart_the_Fink.txt.gz [<2,175>, <2,2023>, 1.3175245956362625]

abc
Bart_the_Genius.txt.gz [<2,1849>, 0.7781512503836436]

abil
Bart_the_Murderer.txt.gz [<2,476>, 0.7781512503836436]

academ
Bart_the_General.txt.gz [<2,1079>, 0.47712125471966244]
Bart_the_Genius.txt.gz [<2,400>, <2,571>, 0.8078365072138199]

accept
Bart_the_Genius.txt.gz [<2,308>, 0.7781512503836436]
```

pic 4-extraction of output file produced by mapreduce, containing same tuples with those in pic 3 (973 words above) sample output (extracted form log.txt that is created when running the program, providing a more structured view of output):



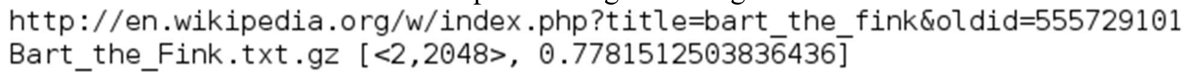
```
1992-08-03
Bart_the_Lover.txt.gz [<2,2202>, 0.7781512503836436]
```

pic 5-date



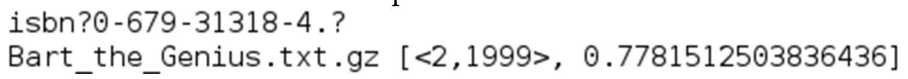
```
accuraci
Bart_the_Lover.txt.gz [<2,1707>, 0.7781512503836436]
```

pic 6-wrong stemming



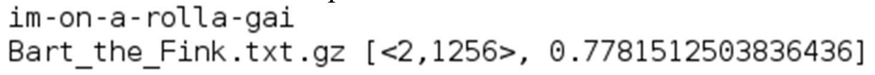
```
http://en.wikipedia.org/w/index.php?title=bart_the_fink&oldid=555729101
Bart_the_Fink.txt.gz [<2,2048>, 0.7781512503836436]
```

pic 7-URL



```
isbn?0-679-31318-4.?
Bart_the_Genius.txt.gz [<2,1999>, 0.7781512503836436]
```

pic 8-isbn number



```
im-on-a-rolla-gai
Bart_the_Fink.txt.gz [<2,1256>, 0.7781512503836436]
```

pic 9-term with hyphen

explanation: Every line has a token and its indices. Indices may consist of detailed information from one document (e.g. {Bart_the_Fink.txt.gz=[<2,1341>, 0.7781512503836436]}) or multiple documents (e.g. {Bart_the_Lover.txt.gz=[<2,1500>, <2,1513>, 0.8078365072138199], Bart_the_Murderer.txt.gz=[<2,1311>, 0.47712125471966244]}) . For information from every document contains document name, a list of position indices that current term occurs in the document and tf-idf score of the term in the document.

Interesting thing:
pic 5: format of date is preserved.
pic 6: tokens created by stemmer that cannot be found in original document.
pic 7: format of URL is preserved.
pic 8: format of ISBN number is preserved.
pic 9: format of token with hyphen is preserved.