

SPIMI Project 1

Concordia University
COMP 479
Fall 2019

Prepared by
Eve Raya, 40016485

Packages, Classes, methods and Folders

BlockHandling (pkg): This package has a java class with the same name that includes two functions:

- *addToBlock(String token, String docID)*
 - This function is used to add the token with its' respective docId's found to a block
- *getBlock(String blockName)*
 - This function is designed to retrieve a block with its' respective LinkedHashMap that includes the terms with their posting lists.

FileHandling (pkg): This package has a java class with the same name that includes two functions:

- *sendToBlock(int blockNumber):*
 - This function takes the block number and creates a new block file with the corresponding number and outputs the non-sorted dictionary to it.
- *sendToFinalFile(TreeMap sortedList, int final_file_number):*
 - This function uses a TreeMap to sort the keys alphabetically and sends every 25,000 terms to a final block text file.
- *DeleteFiles():*
 - This function is used at the beginning of the Main.java to delete any previous blocks in the DISK folder.

Common (pkg): This package has an abstract java class with the same name that includes the following, the abstract class is used to be able to instantiate the same variable in more than one class differently:

- Getters and setter for the *SortedDictionnary* using a *TreeMap<String, LikedHashSet<String>>*
- Getters and setters for the *PostingList* using *LinkedHashSet<String>*
- Getters and setters for the *Dictionnary* using *HashMap<String, LinkedHashSet<String>>*

CompressionTable (pkg): This package has a java class with the same name that includes the following:

- *noNumber:*
 - count of numbers removed
- *case_fold:*
 - count of case_fold done
- *stop_words:*
 - count of stop words removed
- *percent_numbers_from_unfiltered:*
 - % of numbers from unfiltered
- *percent_casefolding_from_unfiltered:*
 - % of case folding from unfiltered
- *percent_stopwords_from_unfiltered:*
 - % of stopwords from unfiltered
- *percent_previous_numbers:*
 - % of numbers from previous
- *percent_previous_case_folding:*
 - % of case folding from previous

- *percent_previous_stopwords*:
 - % of stopwords from previous
- *printTable()*:
 - prints the compression table

Preprocess (pkg): This package has a java class with the same name that includes the following:

- *ArrayList<String> Stopwords*:
 - has a list of stopwords we will compare to later
- *Remove_stopwords(String token)*:
 - method to remove stopwords and count them
- *Remove_numbers(String token)*:
 - method to remove digits and count them
- *Case_folding(String token)*:
 - method to convert to lower case and count the conversions
- *Punctuation(String token)*:
 - method to remove unnecessary punctuation

Query(pkg): This package has a java class with the same name that includes the following:

- *searchTerm(String query)*:
 - searches the FinalBlocks for 1 query term and outputs the posting list
- *searchDictionary(String query)*:
 - searches the FinalBlocks for the query term
- *CustomAndQuery(String query)*:
 - searches the dictionary for the terms in the query and performs the intersection
- *CustomORQuery(String query)*:
 - searches the dictionary for the terms in the query and performs the union
- *intersection(List<String> list1, List<String> list2)*:
 - method for intersection used in AND query
- *union(List<String> result, LinkedHashSet<String> linkedHashSet)*:
 - method for intersection used in OR query
- *printQueries()*:
 - prints the custom queries

Spimi_merge (pkg): This package has a java class with the same name that includes the following:

- *Merge(int nbrOfBlocks)*:
 - merge all 42 blocks into final blocks of 25,000 terms.

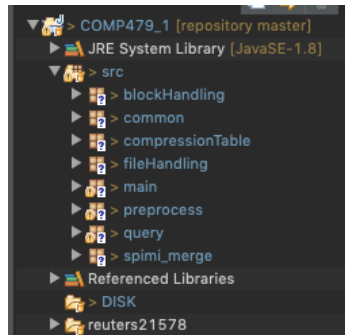
Main (pkg): This package has a java class with the same name that includes the following:

- The main.java runs the program by calling all previous functions. It starts by running the the reuters files and parsing them, tokenizing them and sending them to blocks. Then these blocks are merged, sorted and sent to final files. A query search term is done at the end and a compression table is printed.

DISK (Folder): This folder contains all of the different blocks created when storing the terms in the dictionary. There are currently 42 blocks created, in which each block has terms from 500 different

documents. This folder also includes the FinalBlock file, which has all of the terms merged from all 42 blocks, which are sorted by their keys in alphabetical order.

Reuters21578(Folder): This folder contains the reuters files from which the SPIMI inverted index is created



How the project works

To run the program, only the Main.java needs to be used. The system will output each time it runs through a file to start indexing the terms into a dictionary and then use the compression techniques while preprocessing the files. Once this is done, each document ID is saved in a term-docID posting list to be used later for query search. There is a tracker for the document ID, when it reaches 500, the posting lists are then stored into a block and the tracker goes back to 0. After all the blocks are generated, we merge them into a final block sorted alphabetically. Each 25,000 terms are written to a block.

For the dictionary, a LinkedHashSet was used for the values of the posting list because it removes duplicates, compared to an ArrayList.

When merging into the final block, we need the keys to be sorted alphabetically, so a TreeMap is used.

Block Example

Example of Block1.txt

```
f:466
offerings:154 247
hours:159 290 432 446
opposing:316
oilseedstats:313
buying:343 478
ramon:316
introduce:32 55
niamit:335 334 467
pumping:236 353
news:28
president:52 318
micro:361
reducing:200
finance:7 26 137 175 190 214 225 281 282 295 340 350 359 384 419 458
federax:50 62 74 80 81 103 109 111 141 162 172 175 190 203 204 215 230 240 331 347 348 365 403 452 458 465 475 492
trust:470
painewebber:148
downward:341
measured:371
scherer:149
amenab:488
chicago:13 18 56 200 447 454 471 479 480
plan:3 47
conference:367
required:116 176 179 195 319 391 454
measures:209 237 241 311 317 319 335 356
astor:469
pover:238
savings:225 319 335 342 365
manipu:248
heint:467
convincing:458
suit:156
domestic:326
deemed:144
warrant:381 485
newswr:1387
avia:341
creek:368
bankers:26
swiss:179 364 394 415 416 427 468
given:54
manue:137
soviets:286
begin:342 386
licensing:10 28 32 55 195
wab:45
uae:247 273 349
credi:19 57 97 115 177 300 325 326 382
gains:64 168 364 441
candida:178 222 308
midcon:391
survive:241
uac:375
corporate:295 317 382
ube:471
cans:28 28 51 61 177 178 347 386
```

Final Block Example

```
abzi:467 6430 6435 6436 6437 6439 6450 6461
aba:1679 3829 4785 4866 5846 5855 5107 5102 5239 6321 6868 7375
ababa:5570
abama:370 482 1022 1031 2927 3110 3374 3948 4016 7128 7144
aban:1007
abandon:342 862 1312 1904 2117 2971 3028 3246 4849 5482 5961 6392 6646 6896
abandoned:318 376 341 1000 1036 2115 2656 3440 3885 4655 5473 6090 6337 7254 7029
abandonmen:2273 4495
abank:1916
abankn:2052 2739
abas:4289 8400
abase:763 1663 2139
abases:4169 4339
abbed:2967
abbett:3758
abbey:852 7564 7777
abbrevia:5593
abc:386 7203
abde:6177 6208 8405
abdu:246 1893 3280 3464 3509 3525 5171 5184 6137 6334 7350 8096 8239 8405
abduc:6009
abe:10 215 424 474 1597 2100 3367 3749 3781 4812 4906 4917 5952 6095 6079 7094 7368 7608
aberr:2363
aberrcrombie:6113
aberra:4165
aberr:4496 6206 6217
abex:1789 5748 8293
abeyance:1879
abfi:5904 6784
abhis:5171 5182 5184 5599
abi:139 110 138 144 179 181 207 223 232 236 248 270 273 352 385 390 568 682 684 742 775 802 842 843 854 872 875 891 982 926 942 965 983 1083 1098 1148 1152 1193 1210 12
1501 1535 1542 1553 1560 1597 1626 1815 1836 1893 1897 1906 1916 1963 1967 2061 2077 2110 2115 2195 2231 2242 2278 2286 2290 2420 2448 2449 2514 2544 2685 2694 2709 27
2903 2913 2956 2968 2975 2979 2986 2988 3002 3010 3020 3027 3029 3031 3052 3066 3068 3100 3209 3282 3240 3351 3352 3359 3380 3396 3411 3415 3432 3440 3455 3490 3512 35
3613 3645 3657 3665 3726 3759 3767 3841 3864 3949 3979 4038 4075 4130 4139 4143 4147 4165 4174 4232 4297 4338 4425 4496 4522 4546 4555 4598 4616 4633 4662 4692 4709 47
4878 4902 4906 4969 5004 5032 5070 5107 5169 5176 5181 5193 5199 5206 5210 5214 5255 5258 5271 5273 5279 5283 5290 5298 5323 5377 5394 5934 6025 6072 6108 6150 6168 61
6472 6490 6521 6605 6637 6722 6778 6791 6807 6906 6988 6922 6923 6928 6934 7030 7056 7100 7163 7218 7217 7221 7304 7307 7312 7316 7322 7366 7408 7427 7450 7489 7493 75
7787 7815 7871 7873 7874 7892 7950
abide:951 2376 3446 4665 5167 5273 6348 8427
abiding:2383 4632 4665 5167 5273
abidjan:1889 2324 5168 5192
abilene:4510
abio:4766
abir:1938 7498
abis:2045 3431 3560 5308 5890 5948
abms:1184 3207 3211
abni:2742
abnn:324 1960 2742 5419 6890
abnorma:1981 2679 4897
abo:209 225 256 274 278 255 318 893 991 1275 1574 2195 2975 2988 3364 3564 3589 4670 4785 4902 5087 5190 5217 5318 5338 5432 6401 6646 6962 7851 7336 7561
aboard:2795 2958 5695
abolish:7336
abolishing:7561
aboo:1937
abou:200
abor:188 360 432 668 737 1456 1626 1695 1817 1840 1888 2370 2495 2544 3026 3062 3347 3645 3955 3970 4503 4514 4577 4907 5107 5298 5376 5425 5561 6215 6784 7135 7315 79
aborg:230 311 335 403 748 813 873 952 1010 1391 1619 1809 1947 2036 2209 2320 2352 2411 2571 2631 2818 3277 3284 3338 3564 3589 3668 3809 4028 4402 4429 4444 4651 4664
```

Test Queries

- *1 query term*

```
<terminated> Main (3) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java (Oct. 27, 2019, 4:15:39 p.m.)
Constructing index from reuter file 7...
Preprocess for file 7
Constructing index from reuter file 8...
Preprocess for file 8
Constructing index from reuter file 9...
Preprocess for file 9
Constructing index from reuter file 10...
Preprocess for file 10
Constructing index from reuter file 11...
Preprocess for file 11
Constructing index from reuter file 12...
Preprocess for file 12
Constructing index from reuter file 13...
Preprocess for file 13
Constructing index from reuter file 14...
Preprocess for file 14
Constructing index from reuter file 15...
Preprocess for file 15
Constructing index from reuter file 16...
Preprocess for file 16
Constructing index from reuter file 17...
Preprocess for file 17
Constructing index from reuter file 18...
Preprocess for file 18
Constructing index from reuter file 19...
Preprocess for file 19
Constructing index from reuter file 20...
Preprocess for file 20

Merging blocks to final file....
Sorted list and sent to FinalBlock0
Sorted list and sent to FinalBlock1

# distinct terms      Δ % from unfiltered      Δ % from previous
Unfiltered            5687694
Numbers               5687208                  -1.4412519379558786%
StopWords             4888999                  -12.928241153678744%
Case folding          4878281                  -16.445772678912657%

Would you like to test a query? (Y/N)y
Please enter your words to search:
abama
abama: 370 482 1822 1831 2927 3110 3374 3948 4816 7128 7144 8785 9567 10487 12193 13618 13782 14156
```

As shown above, in the FinalBlock0.txt, all doc ID's are found in this query search. The only bug in my code is that when the 25,000 terms are separated into more than 1 Final block, they are not grouped by terms. Meaning that the same term can be found in both my final blocks even though they are still sorted alphabetically. But we can still manage to see that in order to search for a query, all final blocks are searched, and the correct result is outputted.

- *AND queries*

While comparing with my colleagues, I found out that for the second query, mine was empty. I thought at first that my AND query method was faulty, and this is why I have added other queries not offered by the teacher. Since my new queries came back with a result, I came to the conclusion that maybe I had done a parsing and compression different than my colleagues, which might give different results.

- *Jimmy AND Carter: []*
- *Green AND Party: []*
- *Innovation AND in AND telecommunication: []*
- *Abusive AND Added: [1477]*
- *Advances AND age: [2601, 1694, 8137, 12741, 3185, 296]*

```
=====Starting with AND queries=====
jimmy AND carter: []
green AND party: []
abusive AND added: [1477]
advances AND age: [2601, 1694, 8137, 12741, 3185, 296]
innovation AND in AND telecommunication: []
```

- *OR queries (I added extra queries to make sure the OR query works since the one provided was empty)*

I did not order them

- *Environmentalism OR ecologist: []*
- *Jimmy AND Carter: [13540, 12136]*
- *Green AND Party: [8630, 5124, 6698, 8193, 2404, 13629, 12878, 1753, 7415, 10230, 16115, 15209, 2627, 3714, 3397, 2162, 13190, 1249, 4719, 345, 10682, 9328, 5107, 12300, 14427, 707]*

```
=====Starting with OR queries=====
environmentalist OR ecologist: []
jimmy OR carter: [13540, 12136]
green OR party: [8630, 5124, 6698, 8193, 2404, 13629, 12878, 1753, 7415, 10230, 16115, 15209, 2627, 3714, 3397, 2162, 13190, 1249, 4719, 345, 10682, 9328, 5107, 12300, 14427, 707]
```

Conclusion

While completing this project, I firstly tried to do it using python. I figured that with all the libraries available for NLP it would be easier. But learning a new language in a short period of time was not the most efficient way to do this. I then turned to Java, even though I knew the level of difficulty would be much higher because of the lack of libraries and frameworks. That being said, I did learn a lot. I found that the pre-processing part of the assignment was one of the toughest. It took me a while. To figure out how to split the html document and read the document ID and then the body of it. After that, the tokenization was not a big problem, neither was the compression techniques.

I started the project with one main class, as a monolithic approach. Towards the end, I started splitting classes into packages that made more sense for them to be together. This makes the code look cleaner than having one class. This way it would be easier to implement more features in Project 2 without having to redo all the code.

Moreover, I did find this assignment to be quite hard. It took a lot of thinking and analyzing different approaches possible. But at the end, I learned the true meaning of efficiency and optimization.