COMP479 Project 1
Report

Project 1

Submitted by

Student ID	Name of Student
40058103	Guillaume Rochefort-Mathieu

Department of Computer Science and Engineering
Concordia University
Montreal

Fall 2019

Contents

List of Figures

Chapter 1: Implementation

The implementation of the project was made keeping in mind the usage of the least amount of memory possible within the constraint of the project.

1.1 Preprocessing

The first part of the project was the preprocessing. The preprocessing includes two part the first is the processing of the document from our corpus and the second is the processing of the token and the lossy compression techniques that were used.

1.1.1 Document Preprocessing

The document preprocessing uses a stream of the files from a given directory. The corpus for this project is spread out in multiple files with the ".sgm" extension. Hence, if the document ends with ".sgm" then it is a file that contains document from our corpus.

From these given file we change the line feed to space and split the document on the closing tag </REUTERS>. By the description file in the corpus directory, every document is mark by "<RETUERS .*>(.*) </REUTERS>". Hence, splitting on the closing tags will make strings for each document in the given file.

Only the text found in the body tag is indexed. Tried including others filed from the document but it ended up including numbers and noises which brought very little to the index.

Note that documents are passed on to the token preprocessing step through as a Python generator.

1.1.2 Token Preprocessing

The token preprocessing uses the document stream that was disscussed in 2.1.1. Once the document in generated from the token preprocessing step the document is tokenized using the NLTK library. NLTK is used since the algorithm for tokenization are out of the scope of the this project.

Once the document has been tokenize it is then compress by case-folding and removing any tokens that contains a digit.

Note that tokens are passed on to the SPIMI by the means of a Python generator by yielding a tuple of (term, docID).

1.1.3 Index Compression

Here are the results for some lossy dictionary techniques. Note that stemming was not implemented.

	(distinct) terms			nonpositional postings			positional postings		
	number	Δ%	T%	number	Δ%	T%	number	Δ%	T%
unfiltered	76 899			1 658 033			2 873 759		
no numbers	51 456	-33	-33	1 538 146	-7	-7	2 740 108	-5	-5
case folding	44 699	-13	-46	1 492 345	-3	-10	2 740 103	-0	-5
30 stop words	44 676	-0	-46	1 492 345	-2	-12	2 694 286	-2	-7
150 stop words	44 565	-0	-46	1 155 947	-21	-33	1 917 530	-29	-36

Figure 1.1: Effect of preprocessing on the number of terms

The only discrepency from the table presented above is the lossy compression technique "no numbers". Our guiding table shows no real impact of "no number" compression. Yet, this corpus had alot of terms containing numbers. If we were to investigate the same compression for the postings we can see that these number follows the ones from guide table [1].

The compressing techniques that were used in the construction of the index were: no numbers, case folding. As previous demonstratated in table 1.1 the compression of no numbers and case folding reduced the dictionary to 44 699 terms fitting the description of merging the final index in two final blocks.

1.2 Indexer

For the project we were task into implementing a indexer. The indexing algorithm in question is SPIMI (Single-pass in-memory indexing) [2].

1.2.1 Indexing Algortihm

From the provided pseudocode, slight implementation modification were made to satisfy the Python language. Since the algorithm calls for the use of token stream we had to include the proper ettiquette of using token stream with try catch blocks. The algorithm keeps a dictionary in memory until 500 documents were processed, then the terms are sorted and the dictionary is written in such order in a block.

1.2.2 Final Block Merging

To simplify the merging of the blocks to the final blocks the merge uses a k-way merge algorithm with a min-heap. Since the blocks a writen to disk sorted, by the properties of the SPIMI algorithm, we can abstract the files as sorted arrays and use a k-way sorted array merge algorithm. The concept of the min-heap algorithm is that we put in the heap the first entry of each block in the using the term as the key. This way the first element of the min-heap is the next alphabetically smallest term to add to the final block. To handle identical term we merge the posting lists until the mininum term from the heap is different.

The k-way merged sorted block algorithm space complexity is at most $O(n + l)$ where n is the maximum number of terms that our memory may hold for a final block and where k are the smallest entry from each block (ie the first entry). After insert in the dictionary the next smallest term we must update the heap with the following entries from the respective blocks.

Finally, either when the in-memory dictionary contains 25 000 terms or that all blocks were merge we write the terms to a final block. That block is then added to a list of final blocks which are returns which are used to build a simple boolean search engine.

1.3 Search Engine

In this section the implementation of the simple boolean search engine will be explained.

1.3.1 Query

Queries are used to preprocess the infromation enter by the user and compress it according to the compression techniques used in the construction of the index. Through the use of queries a query language was introduced to make the retrieval more intuitive.

The language that was conceived for this search was the following "(AND—OR):(.*)". This makes is possible to easily distinguish AND and OR queries. An attempt to combine was attempted by quickly left behind has this introduced a concept of precedence and query expansion which we have not covered yet.

1.3.2 Simple Search Engine

The search engine is initialized by building a BST from the blocks generated by the merging of the SPIMI blocks. The keys used in the BST are the first term of each block. This makes it possible to serch in which blocks a query terms belongs in logarithmic time (ie $O(log\ n)$). Note that the BST search does not tell if tell if the terms exists in the index only in which blocks the terms should belong if it would be in the index. Once the block of the term has been determine it is loaded in memory and the terms postings list is retrieve if the terms exists in the dictionary. The step is done for each term of the query.

Once all the postings list are retrived the query operation is executed. If the query is an AND, then the postings list is first sorted by ascending length and intersect the with the result starting with the smallest listi s the result. If the operation is an OR then the postings list are inversely map that is the docID retuns the number of terms that the document have. The inverse map is then return sorted in descending order of number of terms a document contains.

Note that single keyword may either use AND or OR as the search engine checks if only one postings list is return then it returns it as the results.

Chapter 2: Queries Output

In this chapter the results of the queries given by Dr. Sabine Bergler and other students.

2.1 Test Queries

In this section the results of my test queries will be discussed

2.1.1 Single Keyword

The and query "university" resulted in the following documents.

['144', '153', '308', '340', '635', '1833', '1959', '2036', '2129', '2218', '2502', '2514', '2676', '2840', '2854', '2894', '3203', '3234', '3531', '4190', '4491', '4625', '4707', '4727', '5486', '6278', '6492', '7144', '7163', '7232', '8574', '8729', '8763', '8876', '9110', '9342', '9479', '9602', '9708', '9718', '9762', '9801', '10134', '10138', '10179', '10195', '10199', '10230', '10346', '10366', '10606', '11175', '11328', '11485', '11823', '11914', '12233', '12393', '12431', '12907', '13080', '14165', '14359', '14478', '14613', '14618', '14635', '14674', '14687', '14711', '14712', '14713', '14749', '15075', '15232', '15364', '15372', '15373', '15795', '16032', '16777', '16964', '17070', '17085', '17305', '17369', '17428', '17477', '17497', '18126', '18300', '18448', '18684', '18923', '18984', '19292', '19355', '19372', '19432', '19505', '19736', '19974', '20184', '20870', '8008', '21069', '21367']

2.1.2 And

lack of investment

The and query "lack of investment" resulted in the following documents.

[247, 916, 2725, 4016, 4049, 4642, 5171, 5879, 6980, 7051, 8627, 9180, 10697, 11769, 13629, 15743, 16769, 17452, 17891, 18443, 19353, 21508, 21525]

2.1.3 Or

haunted building

The or query "haunted building" resulted in the following documents.

['28', '302', '434', '718', '809', '860', '927', '974', '1023', '1275', '1276', '1282', '1288', '1695', '2011', '2059', '2091', '2114', '2229', '2280', '2282', '2444', '2469', '2775', '2835', '3026', '3030', '3105', '3221', '3281', '3284', '3355', '3390', '3457', '3543', '3607', '3684', '3762', '3825', '3828', '3837', '3956', '4049', '4152', '4168', '4260', '4272', '4290', '4608', '4621', '4640', '4764', '4967', '4983', '5070', '5071', '5086', '5234', '5258', '5308', '5388', '5507', '5514', '5780', '5816', '5985', '6091', '6143', '6197', '6217', '6228', '6309', '6374', '6541', '6590', '6722', '6727', '6949', '6973', '7035', '7251', '7254', '7588', '7777', '7816', '7852', '7855', '7986', '8417', '8604', '8618', '8685', '8743', '8867', '8975', '9055', '9168', '9320', '9321', '9672', '9784', '9820', '9848', '10006', '10131', '10205', '10317', '10345', '10462', '10553', '10614', '10753', '10794', '10934', '11032', '11102', '11170', '11177', '11178', '11213', '11231', '11236', '11237', '11261', '11448', '11504', '11533', '11654', '11768', '11819', '12143', '12188', '12314', '12590', '12598', '12616', '12876', '12973', '12994', '13012', '13135', '13178', '13256', '13266', '13629', '14389', '14419', '14582', '14689', '14727', '14840', '14850', '14863', '14869', '14870', '14900', '14955', '15045', '15048', '15063', '15182', '15213', '15253', '15298', '15310', '15390', '15402', '15407', '15818', '16076', '16139', '16292', '16353', '16577', '16600', '16860', '16868', '16897', '16925', '16930', '16931', '2001', '17042', '17212', '17309', '17335', '17401', '17450', '17475', '17754', '17756', '17781', '17872', '17876', '17883', '17901', '17938', '17964', '17976', '17986', '18144', '18217', '18444', '18479', '18510', '18551', '18577', '18672', '18869', '18892', '18928', '18939', '19078', '19191', '19247', '19296', '19314', '19340', '19663', '19670', '19942', '19995', '20732', '20790', '16864']

2.2 Challenge Queries

2.2.1 And

In this section the results of the AND queries that were given will be disscussed.

Jimmy Carter

The query "Jimmy Carter" resulted in the following documents.

[12136, 13540, 17023, 18005, 19432, 20614]

Note that the lossy compression techniques used in the construction of this index does not discard any relevant information for this query particularly as "Jimmy Carter" is a name and its case to be folded would result in the same list of document ids.

Green Party

The and query "Green Party" results in the following documents.

[21577]

The document retrieve in this case may not be relevant to the information need that the query "Green Party" was constructed. As the query is capitalized then they their information need must be about a political party. Since the query parser reflects the lossy compression techniques used in the index the document might only contain the words "green" and "party".

Innovations in telecommunication

The query "Innovations in telecommunication" resulted in no documents being returned.

[]

After a quick observation we can see that if stemming was used we might have had a better recall but not necessarily the precision of the search engine.

Furthermore, there seem to be multiple instances where telecommunication was spelled incorrectly and was indexed as a different term in the dictionary. This could also have result in a positive hit.

2.2.2 Or

In this section the results of the AND queries that were given will be disscussed.

environmentalist ecologist

The or query "environmentalist ecologist" resulted in retrieving these documents.

['5774']

In this case lossy compressions used in the index would have not change the result of the query. Yet, stemming could have also improve the recall but at the cost of the precsision.

2.3 Daniel Stroppolo's Queries

2.3.1 And

luxury cars

The and query "luxury cars" resulted in retrieving these documents. From Daniel's index:

[1979 2018 4986 9139 9713 9780 9911 10131 17474 17806 19539 20245]

From my index:

[1979, 2018, 4986, 9139, 9713, 9780, 9911, 10131, 17474, 17806, 19539, 20245]

Daniel and I have obtained identical results for this query.

2.3.2 Or

keyboard manufacturers

The or query "keyboard manufacturers" resulted in retrieving these documents. From Daniel's index:

[294 389 635 727 974 1045 1214 1227 1319 1337 1697 1854 2253 2305 2362 2906 3030 3201 3267 3581 3613 3795 4049 4050 4055 4158 4352 4381 4395 4632 4728 4944 4982 5258 5542 6253 6277 6281 6346 6915 7135 7393 7429 7540 7960 8004 8085 8090 8092 8114 8136 8185 8250 8482 8585 8661 8706 8766 8840 9149 9184 9213 9559 9628 9657 9663 9717 9784 9907 10287 10608 10642 10674 10743 10760 10779 10996 11037 11215 11261 11391 11598 12655 13080 13951 14199 14748 14826 15003 15063 15422 16028 16032 16093 16195 16423 16557 16558 16782 16856 16860 16948 17094 17195 17367 17401 17474 17497 17499 17843 18217 18292 18395 18421 18435 18490 18991 19016 19043 19089 19253 19392 19479 19861 19974 20351]

From my index:

['1319', '294', '389', '451', '635', '727', '914', '974', '1045', '1214', '1227', '1337', '1459', '1697', '1787', '1854', '2253', '2305', '2362', '2906', '3030', '3201', '3267', '3298', '3397', '3489', '3555', '3581', '3613', '3795', '4005', '4049', '4050', '4055', '4158', '4352', '4381', '4395', '4632', '4811', '4863', '4871', '4944', '4969', '4982', '5004', '5131', '5186', '5210', '5258', '5294', '5311', '5542', '5869', '5887', '6253', '6277', '6281', '6346', '6915', '7135', '7246', '7263', '7393', '7429', '7447', '7540', '7960', '8085', '8090', '8092', '8114', '8127', '8136', '8185', '8250', '8585', '8661', '8669', '8706', '8766', '8840', '8841', '8932', '9149', '9184', '9213', '9559', '9628', '9636', '9657', '9663', '9717', '9784', '9907', '10118', '10287', '10535', '10568', '10608', '10642', '10680', '10743', '10760', '10779', '10996', '11037', '11215', '11261', '11331', '11351', '11391', '11598', '11603', '11907', '11913', '12091', '12093', '12235', '12333', '12389', '12468', '12536', '12655', '13080', '13629', '13951', '14199', '14331', '14582', '14748', '14826', '14982', '15003', '15063', '15347', '15393', '15422', '15447', '16028', '16031', '16032', '16093', '16146', '16195', '16423', '16557', '16558', '16636', '16782', '16856', '16860', '16899', '16904', '16948', '17094', '17195', '17367', '17401', '17474', '17497', '17499', '17843', '17858', '17931', '18217', '18292', '18360', '18395', '18421', '18435', '18475', '18490', '18555', '18905', '18991', '19016', '19039', '19043', '19086', '19089', '19149', '19157', '19253', '19301', '19337', '19392', '19479', '19861', '19971', '19974', '20351', '20394', '20538', '20704', '20989', '8004', '21031', '21145', '21473', '21572']

Daniel and I are both implemented the index with no numbers and case folding. What varies is the tokenization of the document. Daniel implemented his own tokenization while I used Python NLTK library.

References

[1] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008, p. 87.

[2] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008, p. 73.