Report for Project 2 of COMP 479 by Silas Kalinowski ID: 40256077

The samples from moodle can be found in the file sample_queries.pdf, the demo can be found in the file demo.pdf.

Design and summary of approach:

My project consist of 6 modules:

The first module named 'main.py' calls the needed functions for the creation and compression of the index and prints the compression table. Also, the query is started after printing the compression table.

The module 'module1.py' contains the read_extract function. This function is nearly identical to the read_extract() function of Project 1 except that this function now also extracts the NEWID of every document with its body. It returns the NEWID combined with the article's body in a dictionary.

The module 'Subproject1.py' contains three function that implement the naive indexer. The naive_indexer function processes Reuter files and returns the sorted list F. For that it uses the parser function which returns a list of token,documentID tuples when given a List of tokens and a documentID. The create_index function creates as the name says an index based on a sorted list F.

The module 'Subproject2.py' implements the single term query. It consists of three functions. One function is the implementation of the query. The second function enables the user to search for terms in the console and start the sample and test queries. And the third function runs all the test and sample queries when called.

The module 'Subproject3.py' implements the compression of the index. Every compression step is implemented by a single function which returns the compressed index as a txt. Also, every function returns the size of vocabulary, and the size of the postings lists so that the compression table can be printed.

The module 'Helper_functions.py' contains five function that are used to read an index from a txt file, print the compression table, write an index into a txt file, compute the difference between two values in percent and write the list F in a file.

For my design I ensured that every part of the project is implemented in separate modules. The modules then again consist of function that implement a specific part of the assignment.

I decided to implement every compression separately so that every compression step is traceable and verifiable. For that I also implemented that every step of the compression returns a new index that is saved in a txt file. I chose a format for the txt file that is easily understandable and readable so that I could verify if every step worked as desired.

For sorting the F List for the Subproject1 as already said in the assumptions I sorted the list based on term and documentIDs. That is not as efficient as just sorting it based on term but with this sorting, I do not have to sort the postings lists when creating the index, which would be way more expensive.

The indices are stored in a dictionary when they are being compressed or created. I chose dictionary because lookup and inserting elements is very efficient.

The query processor can be used with any indices as long as the index was saved in a txt file with a specific format. This design enables me to search even the index of the first compression step and not only the fully compressed and uncompressed index.

I also implemented a function that lets the user search for single term in the console and run the sample and test queries. This function is automatically started when running the main file.

Assumptions:

Assumptions for Compression:

When filtering all the numbers for the compression, I assumed that every term that does not include an alphabetic letter and contains at least one number is a number. So, dates and time specifications of any sorts are also filtered out. But numbers that include a unit like 100kg or 100-kg are not filtered out.

For the stemming compression I used NLTK's Porter Stemmer.

To filter out the first 150 stop words I assumed that the stop words should be based on the most frequent terms in the index. Therefore, filtering out 150 stop words means removing the 150 most frequent terms in the index.

Other Assumptions:

When sorting the List F, I sorted alphabetically by term and by the documentID. That means that the list [(a,104), (b, 99), (a, 105)] would be sorted like this: [(a,104), (a,105), (b,99)].

For the token lists which I used to construct the index I assumed that tokens that only consist of non alphanumeric characters are not meaningful for the index and are therefore filtered out. Therefore punctations, etc. are not included in the index.

The list F is a List of tuples in which every tuple contains two elements: the term and the documentID.

The index is a dictionary in which the term is the key and the values is the posting list. The DocumentIDs in the posting list are strings. I did not save the frequency for every term because, in this case, calculating the length of the postings lists is not too costly.

When reading and extracting the tokens of the articles I extracted the text in the BODY-tags of the articles.

For the compression table I calculated the values the same way as in the textbook. I especially made sure that the difference from the removing 150 stop words is in regard to the number of the index after case folding and not in regard to the removal of 30 stop words.

Comparing my and the textbooks compression:

table textbook:

performed stemming with the Porter stemmer (Chapter 2, page 33).

	(distinct) terms			nonpositional postings		
	number	Δ%	T%	number	Δ%	T%
unfiltered	484,494			109,971,179		
no numbers	473,723	−2	−2	100,680,242	−8	−8
case folding	391,523	−17	−19	96,969,056	−3	−12
30 stop words	391,493	−0	−19	83,390,443	−14	−24
150 stop words	391,373	−0	−19	67,001,847	−30	−39
stemming	322,383	−17	−33	63,812,300	−4	−42

My table:

compression	dictionary size	difference in percent	cml	postings lists size	difference in percent	cml
unfiltered	76793	0	0	1582940	0	0
no numbers	53329	30.56	30.56	1467499	7.29	7.29
case folding	46544	12.72	39.39	1422115	3.09	10.16
30 stopw's	46514	0.06	39.43	1142682	19.65	27.81
150 stopw's	46394	0.32	39.59	870265	38.8	45.02
stemming	36017	22.37	53.1	831166	4.49	47.49

Comparing the compression of the dictionary:

When comparing the compressions from the textbook and my compression, I especially noticed that removing numbers from my index compressed my index much better. When talking in relative terms my index's dictionary size decreased by 30.56% and the textbook's index only by 2%. That major difference could be explained by a different definition of what a number is. I assumed that every term that has no alphabetic letter and has at least one number is a number. Maybe the textbook assumed that only terms that only consist of numbers are filtered out.

When looking at the compression numbers for casefolding the numbers are similar. My index was compressed by 12.72% the textbook's index was compressed by 17%. When factoring in that my index was already more compressed by removing numbers the slightly higher compression rate makes sense for the textbook.

Regarding the impact of removing stop words the dictionary size obviously only decreased by 150 in both cases.

Stemming the terms had a higher impact on the dictionary size of my index.

Comparing the size of the non-positional postings lists:

Removing numbers had roughly the same effect on the size of the postings lists on both indices (7.29 <-> 8). That is interesting since the numbers were greatly different when comparing the dictionary size. The fact that most of the numbers do not have a high frequency could explain that.

Case folding has nearly an identical effect on the size of postings lists.

Removing stop words compressed both postings lists by a high percentage (14+30 <-> 19.65 + 38.8). I would have expected that the compression rate in the textbook is slightly higher. Because the corpus of the textbook is larger and has a lot of more documents/articles the postings list of the stop words are longer and therefore the compression rate should be higher. That surprise could be explained by the fact that the textbooks posting lists is way bigger than my postings lists. So a smaller compression rate still is way bigger in absolute terms.

Stemming had a very similar effect on both sizes of postings lists.

Aborted design ideas:

When planning the project, I thought about not saving each index of the compression steps. But then I realised as already mentioned above especially for traceability and testing, saving the indices is very helpful. The same thought applies to saving the F file. Also, with saving the indices the query can be used independently from the other functions as long as the index was created and is stored in the local directory. In my first plan I always had to run the creation and compression when using the query.

For the data structure which I wanted to use for the indices I first planned to store the index in a list, because List can be easily sorted. But the long look up time and the fact that dictionaries in python can also be sorted made me chose dictionaries over lists.

When sorting F and removing duplicated in my first implementation I only sorted F alphabetically based on the terms. But then I realised when creating the index, I need to sort the postings lists because F was only sorted by term and not also by documentID. That is why in my final implementation I sorted F based on term and documentID.

At first, I designed the query so that one has to call the function in the code with the terms you are looking for. Because the user should not edit the code, I wrote a function that enables the user to search for term in the console.

Results of my three sample queries

See full output in console by typing t while the query processor is started.

Query: in

Uncrompressed: 12785 documents contain the term 'in': ['1', '2', '3', '4', '6', '8', '9', '12', '15', '16', '17', '18', '19', '20', '23', …]

Compressed: No document contains the term 'in'.

Explanation: Because ‘in’ is very frequent in the corpus and therefore is in the 150 stop words that are removed the query in the compressed index returns no documents. The query in the uncompressed index returns nearly every article.

Query: article

Uncompressed: 41 documents contain the term 'article': ['925', '982', '1022', '1552', '1904', '2036', '2796', '4868', '5917', '6112', '6382', '6384', '7260', '8118', '8195', '8203', '8441', '8756', '9327', '10036', '10504', '10623', '11110', '11768', '11918', '12431', '12750', '12879', '13949', '15040', '15369', '16607', '16649', '17119', '17776', '17806', '18700', '18823', '18923', '19263', '21123']

Compressed: No document contains the term 'article'.

Explanation: Stemming article or articles leaves articl. That is why article is not found in the compressed index but returns 41 articles in the uncompressed index.

Query: zone

Uncompressed: 44 documents contain the term 'zone': ['1', '273', '626', '630', '2264', '2411', '4246', '4442', '5057', '5143', '5268', '5526', '5564', '5784', '5812', '5863', '6411', '6426', '6660', '7067', '7629', '7645', '7823', '8070', '8563', '8645', '8948', '9754', '9764', '10214', '10216', '10331', '10539', '10662', '10811', '10936', '10995', '12806', '12963', '13320', '13535', '13542', '15932', '17054']

Compressed: 80 documents contain the term 'zone': ['1', '273', '276', '626', '630', '1088', '1323', '1990', '2264', '2411', '2522', '2782', '2880', '3442', '4246', '4442', '4734', '5057', '5143', '5167', '5244', '5268', '5273', '5526', '5564', '5784', '5812', '5863', '6108', '6166', '6400', '6411', '6426', '6660', '7067', '7304', '7629', '7645', '7823', '8070', '8563', '8645', '8948', '9153', '9209', '9754', '9764', '9864', '10214', '10216', '10331', '10539', '10662', '10811', '10868', '10936', '10995', '12502', '12641', '12650', '12651', '12690', '12806', '12963', '12983', '13259', '13320', '13535', '13542', '14757', '14773', '15470', '15549', '15694', '15932', '16966', '17054', '19110', '19559', '20087']

Explanation: For the compression zones was stemmed to zone and maybe even Zone was put in one equivalent class as zone because of casefolding.

Additonal Testing

Testing the removal of numbers:

Query: 275

Uncompressed: Expected: Several documents. Return: 25 documents contain the term '275': ['635', '933', '2326', '2542', '2705', '2828', '2971', '3028', '3329', '3500', '3792', '4774', '6550', '8932', '10056', '11233', '12746', '13586', '14119', '14852', '15659', '16660', '17506', '18061', '19948']

Compressed: Expected: No document. Return: No document.

Testing casefolding:

Query: 'Wednesday'

Uncompressed: Expected: Several Documents. Return: 169 documents contain the term 'Wednesday': ['81', '109', '175', '181', '190', '203', '236', '343', '362', ..]

Compressed: Expected: No document. Return: No document.

What I learned in this project:

With this project I learned that separating modules and functions properly makes a major difference. Especially for a better overview over the project a good structure and a sophisticated separation of the modules and functions is needed. It also makes it easier to implement changes and fix bugs.

Furthermore, as already mentioned in aborted designs saving results can be very helpful. With this project I also realised how much time one can save by planning before starting to code. Even a rough plan how you want to implement the project saves much time when coding and prevents bugs later.

The comparison of my compression table to the compression table in the textbook helped my understand both tables better. Also, I realised that different assumptions regarding the compression can have a major influence on the values in the table. For example, the assumptions which word is a number can change the compression rate from 10% to 30%.

Next, implementing the single term query showed me that only searching for single terms is very efficient and easy to implement. When using a dictionary, the look up is efficient and easy. But when thinking about Boolean retrieval or even phrase queries with ranking, I realised how complex other forms of retrieval are.

Lastly, as already mentioned in aborted design ideas using a dictionary to store the index reminded me how important it is to use proper data structures.

I certify that this submission is my original work and meets the Faculty's Expectations of Originality.

Signature ID: 40256077 Date 2022-10-08