Fast Arabic Query Matching for Compressed Arabic Inverted Indices

Ameen A. Al-Jedady[1], Mohammed N. Al-Kabi[2] and Izzat M. Alsmadi[1]

[1]*Dept. of Computer Information Systems, IT & CS Faculty, Yarmouk University, Irbid, Jordan*

[2]*Faculty of Sciences and IT, Zarqa University, Zarqa, Jordan*

ameen_on@yahoo.com, ialsmadi@yu.edu.jo, mohammedk@zpu.edu.jo

Abstract

Information retrieval systems and Web search engines apply highly optimized techniques for compressing inverted indices. These techniques reduce index sizes and improve the performance of query processing that uses compressed indices to find relevant documents for the users' queries.

In this paper, we proposed a novel technique for querying compressed Arabic inverted indices in search engines. The technique depends on encoding Arabic terms stored in the inverted indices of Web search engines and information retrieval systems. This minimizes the storage space required for those terms in the index and decrease the number of comparisons needed for query matching in the query processing stage. The number of comparisons is decreased by minimizing the number of bytes that represent the terms in the index and applying the same encoding technique to the keywords of the query. The results showed a 38% reduction in total size of the index. The average number of comparisons to find a word is also decreased in the new index. Both sequential and binary searches were decreased by: 13.58%, and 38.63% respectively relative to the total number of comparisons of each keyword in the query.

Keywords: *Querying compressed Arabic terms, index compression, index querying, integer coding, fast query matching*

1. Introduction

The amount of information the web is providing to users is tremendous and it is continuously getting larger which demands the need to have a robust and reliable information retrieval system. There are several important factors affecting the operation of any information retrieval system, including the amount of resources that the system has, and the way it uses them to satisfy its users' needs. This is accomplished by either an easy-to-use interface with high quality results it produces, or the fast response to the user's queries. Web Search engines need to process thousands of queries per second and return the results in less than one second, out of tens of millions of documents. Web Search engines are facing huge performance challenges, especially with the continuous growth of information on the Web. This means that the need for optimized systems is growing and there is a lot of researches around space optimization [1, 2, 4] performance optimization [7, 25] and query processing optimization [19].

Data compression generally aims to create a compact form of the original stream of data. Most compression techniques are based on looking for repetitive items to be substituted with a compact form. There are two types of data compression: lossless

compression and lossy compression [19, 21]. This study is based on Huffman encoding which is one of the famous lossless compression techniques, where a stream of bits is generated for each letter [11, 21].

Compression techniques have their pros and cons. The pros are focused on reducing the space needed to store the compressed materials. This is beneficial in the case of scarce disk space, besides improving caching, and as a consequence of reducing the needed size of storage, the seek time and transfer costs are decreased. While the main disadvantages of using these techniques are summarized by the need for decoding before we can use them, a considerable time is needed to decode the compressed materials, beside the need to encode them in the cases of updates [19, 22, 26]. However, we would not face the problem of decoding in our study since we use Huffman code to convert the index terms as well as the words of the query into integers to reduce their size and to do the matching process as well.

The inverted index records consists of two components the vocabulary and the inverted list [19]. The vocabulary is the word from the document collection and the inverted list is the information about the location of that word in the documents collection, its position, frequency, etc. Each term or word in the index has an inverted list containing index postings that contains information about the occurrences of the term in particular documents (e.g. through providing their IDs). It also includes other information such as the location of the word within that document. For example {207, 2, 26, 33} might mean that the word occurs twice in document 207 at positions 26 and 33 from the beginning of the document [23].

Some studies proposed solutions such as caching index [25] and compression of index files [1, 2, 19, 26]. In this paper, we focus on the space optimization of inverted indices and their impact on the performance of the query matching process.

The searchers are usually interested in the result lists generated by the search engines, not the way their requests were processed. So there is no need to store the keywords as the searchers understand if there are other better ways to do that such as encoding the vocabularies of the index as well as the user's queries and then perform the matching process.

Our study focuses on encoding the terms of the index file, in particular the Arabic terms to minimize the number of bytes to save each term. This leads to minimizing the index file while minimizing the overall index size. Eventually, the number of comparisons in the query matching will be minimized. We will analyze the encoding technique and its effect in improving the space reduction and query processing optimization.

This study presents a new encoding technique for Arabic terms in the inverted index, assuming that we have an optimized document ordering so that any other compression technique can also be used for compressing the other elements of the index file besides the compression of the terms in the index.

2. Related Work

The continuous daily accumulation of vast amount of information on different Web sites, motivate Web search engines to access and collect the largest possible number of these Web pages in order to achieve one of the main key components adopted by Web search engines which is called comprehensiveness. It affects directly the quality of the results of the Web search engine pages to different queries, since these pages may contain references to Web pages not already crawled by the spiders of Web search engines. Therefore Web search engine spiders are continuously visiting the different

Web sites and collecting a vast amount of information to enable their users to search over this gigantic amount of information. Consequently, Web search engines have to adapt optimized compression techniques to minimize the space and to speed up the process of retrieving information. There are many studies that present different techniques to compress the inverted index of Web search engines such as: [7, 19, 22, 23, 26].

[16] presented a comprehensive survey to the new direction of compressing indices, where the study started by showing that full text indices provide a speedy search over a giant poll of textual documents. The disadvantage of this approach is definitely the space overhead. The new emerged trend in recent years focuses on designing compressed indices exploiting the full-text compression [15, 16].

Compressed text indexing as a new technique is addressed by [9] study, where the researchers presented the design, implementation and an evaluation of this new technology. The base ideas that these indexes are depend on are presented. Also Pizza & Chili Web site is used because many of the compression techniques are already implemented and ready for use as final software. So the tests are conducted to reveal the best of these techniques. The main target of this extensive study is to discover the practical relevancy of this new technology.

Several researchers have discussed Word-based compression techniques. [4] study is one of the pioneering researches that proposed a new local adaptive data compression scheme. It presented a word-based compression technique that is based on "Move to the Front" (MTF) compression technique, which yields good compression ratios. They proved that their scheme always performs better than Huffman coding method. This study was followed by a study, which was conducted by [13] to compress semi-static words and separators as well, and based on Prediction by Partial Matching (PPM) technique. PPM technique is a context based technique, which yields good compression ratios.

[14] presented a fast technique to compress and decompress words and separators. The study claims a significant speed of decompressing arbitrary portions of the text, beside the ability to search within a compressed text without the need to decompression.

Compression of Arabic text was studied by several researchers such as [12] and [17]. [12] studied the compression of Arabic text by mapping the Arabic text to binary format then applying the file splitting technique. This technique showed a considerable reduction in the size of large Arabic text. [17] continued the research to use the previous techniques with hybrid dynamic coding to compress the small Arabic text while minimizing memory overhead. The results showed a reasonable compression ratio for short Arabic text.

[3] presented a new multilayer approach for compressing Arabic text; where features of Arabic language being exploited to effectively minimize the storage space. Three layers are used by researcher according to the category of the Arabic word: derivative layer, non-derivative layer, functional words layer, where each of these layers will adopt a different compression technique. The fourth layer which is called Mask layer is used to decompress the Arabic text back to its original state depending on layer. Using multilayer approach helps to maximize compression ratio relative to typical techniques.

[10] study used Dynamic Huffman coding to compress Arabic and English text, where the compression ratios for Arabic text was better (or higher) than those of English text. Their test results revealed also that an increase in file size will lead to an

increase in the compression ratio, and definitely the frequency of terms within text affects compression ratios positively.

[26] presented in their tutorial the main techniques used to design and implement high-performance text indexing algorithm. This technique is also mentioned in 1, 2, 5, 8, 20, 22, 23]. [26] mentioned that inverted file indices can be compressed through the use of compact storage of integer techniques.

[6] described a new document representation model, where a tree data structure is used for related documents, to index shared contents once, instead of indexing each document separately. The researchers showed that this representation model can be encoded within the inverted index, in a way that reduces the size of the index significantly.

[15] study proved that combining three known techniques (i.e. inverted index compression, block addressing and compressed text which permits direct search) outperforms using these techniques separately. The tests on the hybrid technique showed that it is 7 times faster than Glimpse which is a known system that adopts block addressing. The percentage of text compression is less than 40% of the original size of text, while there are many compression techniques with compression ratio below 30% of the text size. They showed that the optimal block size for 1GByte of text is 4,000 words.

Some studies focused on the cashing of index files [25], and other papers focused on the compression of the index with various techniques such as replacing document IDs in the posting list with the difference between it and the preceding document ID. This difference is called d-gap and it applies integer compression algorithms. Since the resulting compression ratio depends on the properties of the sequences of the integers which depend on the way IDs are assigned to the documents. Studies in this field was conducted to enhance the d-gap technique through document IDs renumbering [5, 24]. The main idea of document IDs renumbering is to group similar documents together to decrease the average distance between documents that relates to a particular term, which will decrease the d-gap. Later on, compress the inverted index using integer compression algorithms, which improve the compression of inverted index [1]. Some studies presented a new compression technique [7], and recent studies focused on the optimization of existing compression techniques to minimize index size while improving the speed of query processing [24].

[23] study targeted towards the compression of term positions within the indices of Web Search engines. An exploitation to the most efficient way to access position data for ranking functions during query execution is also proposed in the paper. Two effective techniques are proposed: Remaining Page-Adaptive Rice Coding with Smoothing (RPA-RC-S) and Remaining Page-Adaptive BASC with Smoothing (RPA-BASC-S), besides statistical methods (Optb-4D, Huff-4D, and LLRUN-4D). The authors indicated moderate improvements in compression ratios.

2 Methodology and Approaches

This section demonstrates our methodology toward improving the compression process of Arabic indices of Search engines. Figure 1 shows the high level steps followed in this approach. Those steps can be described as follows:

Figure 1. The Overall Methodology Workflow

I. Determine the codeword for each Arabic letter according to its frequency.

The general steps are shown in Figure 2. Choosing a random document is one option to get a general idea of the frequency of each Arabic letter; although it is not the most efficient way. We analyzed two documents; one from the Internet and another one from an old Arabic literature. The analysis result shows that the percentage of the occurrences of all the letters is almost the same; so we selected the second document to be used as a reference.
Selecting the document to determine the exact percentage of the occurrences of each character is vital due to its impact on the length of the code of each character. However, this is not the main concern or scope of this paper. We recommend using the current search engine index as a starting point to calculate the frequency of Arabic letters. Since the index contains each word only once, each letter is counted as much as it appears in the index, where repetitive words are eliminated.

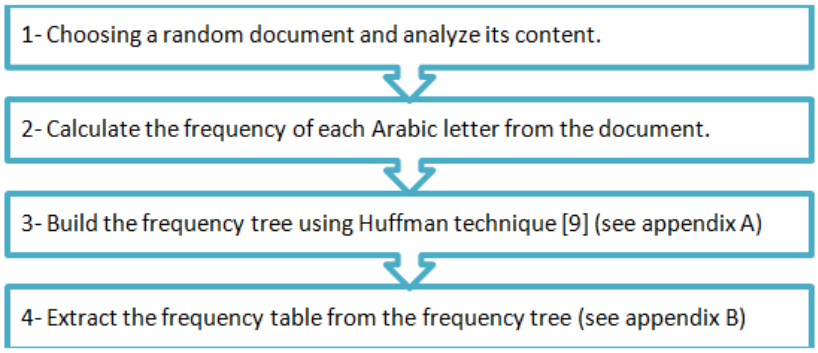


Figure 2. The Steps for Determining the Codeword of each Character

After the frequency of each Arabic letter was calculated, the frequency tree was built according to Huffman coding technique [11, 22] (as shown in appendix A). Then the frequency table was obtained by traversing the frequency tree from the root to the leaves that contains the Arabic characters.

II. Modify the indexer to encode the vocabularies.

The search engine spiders are continuously crawling the World Wide Web (WWW) to collect new and updated Web pages and send them to the indexer for processing. Our proposed model will alter the way the indexer stores Arabic vocabularies in the index file. The Query module will also use the same proposed model to alter the Arabic words to retrieve their locations. It can be used beside any other compression algorithm for further improvement of the search engines indices.

The indexer will neither store the exact Arabic document word in the inverted file, nor its stem. Rather it will store the numbers represented by the code of the letters composing the word using the frequency table (Appendix B). Although the stem of the term is not stored in the index, it can be sent to the stemmer before encoding it; in case that the stemmer is required in the system. In this case, the query terms will be sent to the stemmer before encoding each one of them to search for them in the new index.

As shown in Figure 3, each document was tokenized into words. Characters of each word were then traversed one by one and each was replaced by its code from the frequency table [Appendix B]. After completing word processing, the search was started for codeword in the index. If it exists; a search for the ID of the current document among its posting list in the index is started. If it exists, next word is selected, and the process is repeated to the end of the document. For each word, in case the codeword is in the index while the ID of the current document is not among its posting list, then its posting list is updated to contain the current document ID. If the codeword is not in the index then we need to do two things: first we add the codeword of the current word to the index, second we add the current document ID to the posting list of the newly added codeword. The pseudo code for this step is given in Algorithm 2 that contain calls for Algorithm 1.

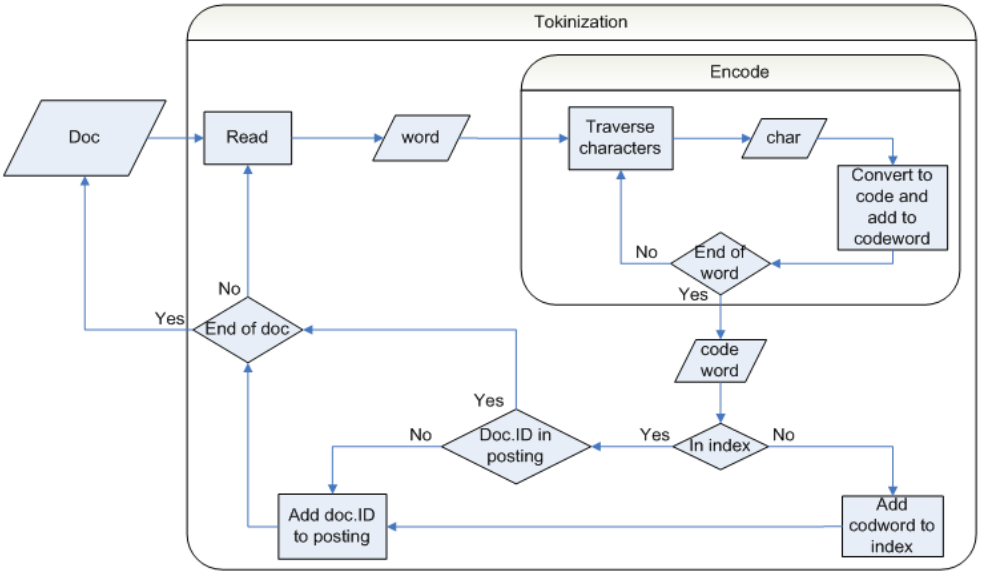


Figure 3. The Proposed Process of Indexing the Collection

Algorithm 1	Encode (W, FrequencyTable).
Input:	W: An Arabic word. FrequencyTable: Table that contains the frequency of each Arabic letter and its code see Appendix B.
Output:	Array-of-bytes: An Array of bytes where each byte represent 8 bits of the codeword of W after the encoding.

```
BEGIN
   FOR EACH character C in W
        word-code += code-of(C in frequencyTable)
   ENDFOREACH
   FOR EACH 8-bit segment in word-code //i.e. divide word-code into segments
of 8 bits
        Array-of-bytes += convert(segment to one Byte)
   ENDFOREACH
   RETURN the overall array-of-bytes
END
```

Algorithm 2	Indexer (ArbClct).
Input:	ArbClct: Collection of Arabic documents.
Output:	currentIndex: either create new of append an existent index file consist of records of the form (Array-of-bytes → posting list).

```
Begin
   FOR EACH Document Doc in ArbClct
        ID = identification number of Doc
        FOR EACH Word W in current document Doc
                array-of-bytes = Encode(W)
                IF (array-of-bytes not in currentIndex) THEN
                     Add array-of-bytes to currentIndex
                          Add current Doc.ID to array-of-bytes posting list
                ELSE
                          IF (current Doc.ID  not in array-of-bytes posting list)
                          THEN
                              Add current Doc.ID to array-of-bytes posting list
                          ENDIF
                ENDIF
        ENDFOREACH
   ENDFOREACH
   Return currentIndex
End
```

III. Modify the query processor to encode the query before searching the index.

The word(s) of the query must be encoded in the same manner as the index terms, then we search the index for the codeword of each word in the query and retrieve the posting list for each one. In case the system pre-processes the words of the documents before indexing them we will need to pre-process the query in

the same manner. This means that each word in the query is pre-processed, stemmed for example before we use algorithm 1 to encode it.

IV. Test the system and analyze the results.

This section provides an initial case study to evaluate the proposed approach and algorithm. Two Arabic documents named Doc1 and Doc2 are used to evaluate this novel approach.

First, each document is processed as described in the methodology section. Table 1 shows the data produced from the processing of the two documents.

Table 1. Summary of the Case Study Results

Encoded term	Number of bytes (encoded)	Number of char's	Arabic Index term	Term code word
235 89 147	3	5	البحث Search	111010110101100110010011
235 89 177 48	4	6	البحوث researches	11101011010110011011000010011
234 251 212 20	4	7	العربية Arabic	11101010111110111110101000001010
56 75 32	3	3	ضغط compression	001110000100101100100
192 253 128	3	5	فهارس Indices	1100000011111101100000
163 250	2	4	مجال Field	1010001111111010
172 246 111 96	4	6	محركات Engines	10101100111101100110111110110

In the next step, the new index and the old index are evaluated to compare their sizes as well as the performance of querying their content. Table 2 shows the new index vs. the old one, taking in consideration that the new index reorders its entries in an ascending order (as integers).

Table 2. The New Index vs. the Old Index

Old index:	New index:
البحث→1, 2	56 75 32→1
البحوث→1, 2	163 250→1, 2
العربية→1, 2	172 246 111 96→1, 2
ضغط →1	192 253 128→1
فهارس→1	234 251 212 20→1, 2
مجال→1, 2	235 89 147→1, 2
محركات→1, 2	235 89 177 48→1, 2

Assuming that each Arabic character needs only one byte, the size of the vocabulary part of the new and old indices is: 23, and 36 bytes respectively. This means that the reduction in the index size was (1 – 23/36) = 36% of the total size of the terms in the index. This reduction is corresponding to the decrease in the number of bytes to represent 6 out of 7 words which is 85% of the terms of the index. The average number

of comparisons to find a word was also decreased in the new index as shown in Figures 4 and 5. For example to find the word (indices فهارس) using the old index needs 9, and 7 comparisons, while using the new index it needs only 6, and 3 comparisons, using sequential search and binary search respectively.

الاستعلام query	Old sequential	New sequential
البحث search	5	8
البحوث researches	11	12
العربية Arabic	13	8
ضغط compression	6	3
فهارس indices	9	6
مجالfield	9	3
محركات engines	1	6
Average # of comparisons	66	46

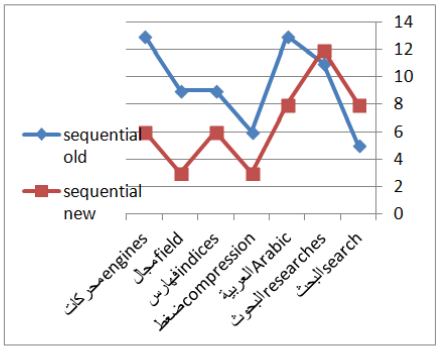


Figure 4. Number of Comparisons Needed to Find Each Word in the New and Old Indices (using sequential search)

الاستعلام Query	Old binary search	New binary search
البحث Search	1	4
البحوث researches	7	8
العربية Arabic	11	6
ضغط compression	3	5
فهارس Indices	7	3
مجال Field	5	3
محركات engines	9	6
Average # of comparisons	53	35

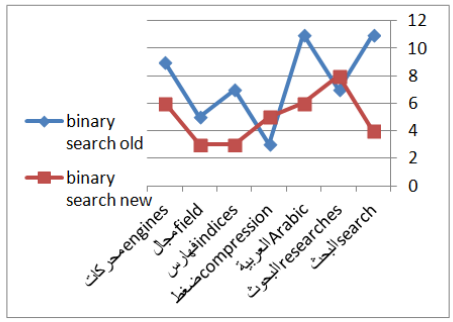


Figure 5. Number of Comparisons Needed to Find Each Word in the New and Old Indices (using binary search)

The decrement in the average number of comparisons for each term was reduced by (1 – 46/66) = 30.3% for sequential search as shown in figure 4. Also the decrement in the average number of comparisons for each term using binary search was decreased by (1 – 35/53) = 34% as shown in figure 5.

3. Experimental Results

The experiment was conducted on 10,100 Arabic text files which were collected manually from the Wikipedia and some other news websites. We applied the proposed encoding on the vocabulary of the index which had approximately 152,200 terms. Then

we analyzed the results, which showed a considerable reduction on the space required for storing the terms of the index and on the number of bytes representing the word, which will yield to a faster matching process for the query.

Assuming that each Arabic character is stored in only one Byte (which is not always true), the reduction of the size of the terms part (not the entire index size) was 38%. However, it should be considered that 99.17% of the index terms had compression gain of the proposed encoding method.

The main contribution of this novel technique relies mainly on the decrement of the number of comparisons, while the reduction in the size of the index is slightly small. Our investigation to discover the reasons behind the slight reduction in the index size shows that, the average size of the original word was 6.3 bytes and reduced after encoding to 3.9 bytes. That makes the percentage of the reduction around 38%.

The investigation also shows that each word in the index is repeated on average in 11 documents, which means that the average posting list size is 11 × 4 bytes which is equal to 44 bytes. So each record in the old index occupies 6.3 + 44 which is equal to 50.3. While the record of the new index will occupies 3.9 + 44 which is equal to 47.9. This means that the reduction of size for the entire index will be limited to 4.76%.

On the other hand, each word in the index was enquired from the old index and the new one, and number of comparisons for each query was observed. It is noticed that the average number of comparisons to find the index term using binary search method was decreased by 38.63% while the average number of comparisons to find the index term sequentially was decreased by only 13.58%. However, it should be considered that 96%, and 53% of the index terms had comparison gain of the proposed encoding method, using binary search and sequential search respectively. The sequential search performance was not significant because the new index reorders the terms by their codeword rather than sorting them alphabetically.

4. Conclusions and Future Work

In this paper we studied the compression of indices in search engines. Relevant previous work projects are focused on DocID reassignment and documents reordering to group similar documents close together. Other papers focused on the compression of integers within the posting list. In our approach, we focused on the compression of the terms of the index to reduce the size of the index and to minimize number of comparisons needed to match the user's query using sequential search and binary search.

The sequential search results can also be improved through building a Huffman tree that takes into consideration putting most of the first characters of the alphabets at the left hand side of the tree to make most of their code representations as zeros.

We have noticed that the proposed method produced significant improvements that can be used to improve earlier approaches. This work motivates a further research on the proposed model and its improvements to the performance of the information retrieval systems using methods such as the B-tree. Further studies should also evaluate the impact in the stemming process and the ability to use fixed codeword for the characters to convert the stemming process to formal mathematical formulas. This paper also motivates researching the importance of using old indices to build the frequency table and the frequency tree, and see how that process can improve the search and query performance. Our future work will cover the difference between applying Huffman coding to English and applying Huffman coding to Arabic.

Acknowledgement

The authors thank Dr. Zakaria Zaatreh and Mr. Amer O. ELMughrabi for their careful reading and comments.

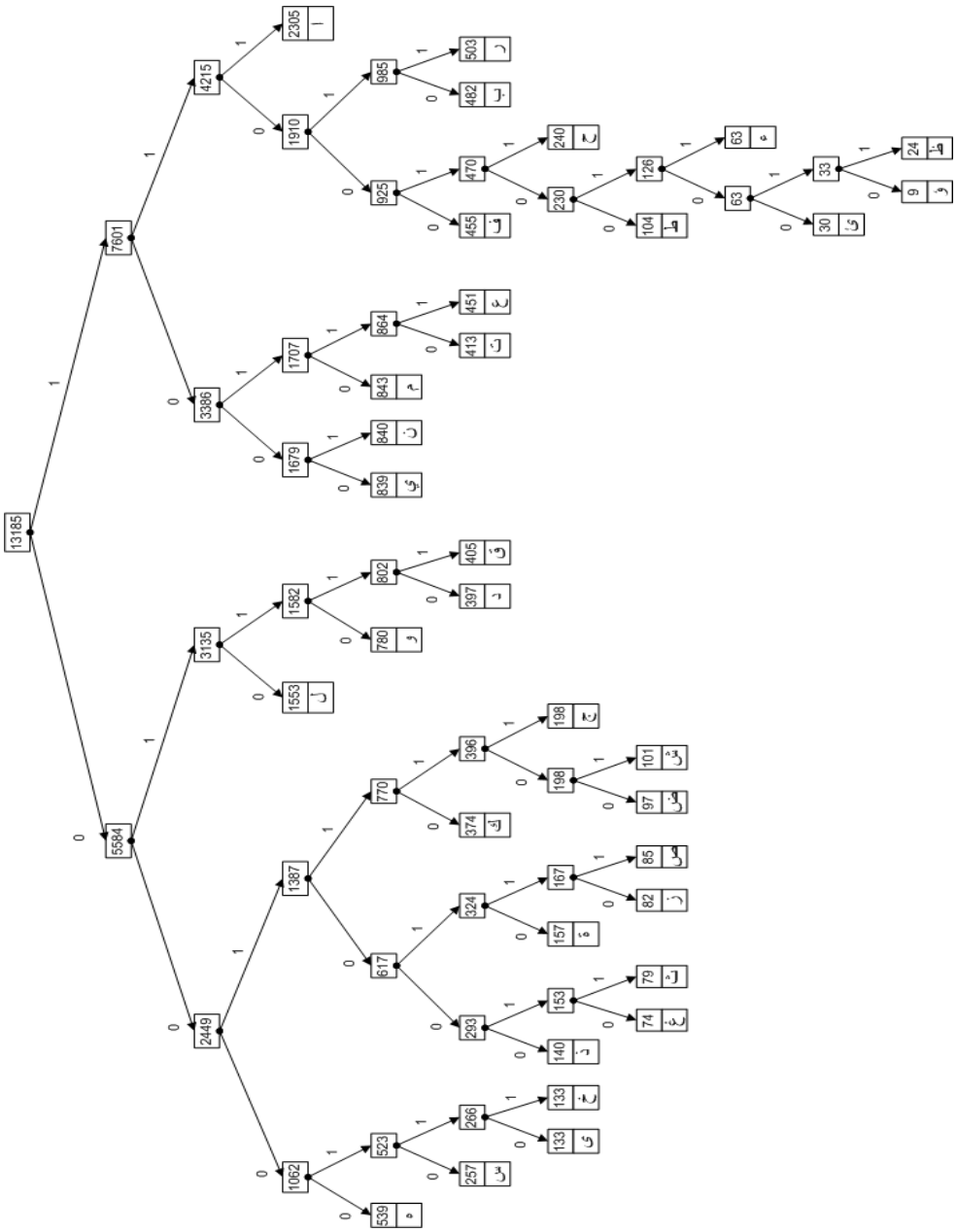
References

[1] V. Anh and A. Moffat, "Index Compression Using Fixed Binary Codewords", In: Proceedings of the 15th Australasian database conference (ADC '04), **(2004)**, pp. 61 – 67.
[2] V. Anh and A Moffat, "Inverted Index Compression Using Word-Aligned Binary Codes", Journal of Information Retrieval, vol. 8, no. 1, **(2005)**, pp. 151 – 166.
[3] A. Awajan, "Multilayer Model for Arabic Text Compression", The International Arab Journal of Information Technology, vol. 8, no. 2, **(2011)**, pp. 188-196.
[4] J. Bentley, D. Sleator, R. Tarjan and V. Wei, "A Locally Adaptive Data Compression Scheme", Communications of the ACM, vol. 29, no. 4, **(1986)**, pp. 320 – 330.
[5] R. Blanco and A. Barreiro, "Document Identifier Reassignment Through Dimensionality Reduction", In: Proceedings of the 27th European Conference on Information Retrieval Research (ECIR 2005), **(2005)**, pp. 375 – 387.
[6] A. Broder, N. Eiron, M. Fontoura, M. Herscovici, R. Lempel, J. McPherson, R. Qi and E. Shekita, "Indexing Shared Content in Information Retrieval Systems", In: Proceedings of the 10th International Conference on Extending Database Technology (EDBT'06), **(2006)**, pp. 313 – 330.
[7] S. Büttcher and C. L. A. Clarke, "Index Compression is Good, Especially for Random Access", In: Proceedings of the 16th ACM Conference on Information and Knowledge Management (CIKM 2007), **(2007)**, pp. 761-770.
[8] J. Chen, P. Zhong and T. Cook, "A Mixed Coding Scheme for Inverted File Index Compression", First IEEE Workshop on Hot Topics in Web Systems and Technologies, **(2006)**, pp. 1-8.
[9] P. Ferragina, R. González, G. Navarro and R. Venturini, "Compressed Text Indexes: From Theory to Practice", ACM Journal of Experimental Algorithmics (JEA), vol. 13, **(2009)**.
[10] S. Ghwanmeh, R. Al-Shalabi and G. Kanaan, "Efficient Data Compression Scheme using Dynamic Huffman Code Applied on Arabic Language", Journal of Computer Science, vol. 2, no. 12, **(2006)**, pp. 887-890.
[11] D. A. Huffman, "A method for the construction of minimum redundancy codes", In: Proceedings of the IRE, **(1952)**, pp. 1098–1102.
[12] A. R. M. Jardat, M. I. Irshid and T. T. Nassar, "Entropy Reduction of Arabic Text Files", Asian Journal of Information Technology, vol. 5, no. 6, **(2006)**, pp. 578-583.
[13] A. Moffat, "Word-Based Text Compression", Software—Practice and Experience, vol. 19, no. 2, **(1989)**, pp. 185-198.
[14] E. Moura, G. Navarro, N. Ziviani and R. Baeza-Yates, "Fast and flexible word searching on compressed text", ACM Transactions on Information Systems, vol. 18, no. 2, **(2000)**, pp. 113–139.
[15] G. Navarro, E. S. De Moura, M. Neubert, N. Ziviani and R. Baeza-yates, "Adding Compression to Block Addressing Inverted Indexes", Journal of Information Retrieval, vol. 3, no. 1, **(2000)**, pp. 49-77.
[16] G. Navarro and V. Mäkinen, "Compressed Full-Text Indexes", ACM Computing Surveys (CSUR), vol. 39, no. 1, **(2007)**.
[17] E. Omer and K. Khatatneh, "Arabic Short Text Compression", Journal of Computer Science, vol. 6, no. 1, **(2010)**, pp. 24-28.
[18] K. Sayood, "Introduction to Data Compression", third edition, Elsevier Inc, PA, **(2006)**.
[19] F. Scholer, H. Williams, J. Yiannis and J. Zobel, "Compression of Inverted Indexes for Fast Query Evaluation", In: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR 2002), **(2002)**, pp. 222 – 229.
[20] F. Silvestri, R. Perego and S. Orlando, "Assigning Document Identifiers to Enhance Compressibility of Web Search Engines Indexes", In: Proceedings of the 2004 ACM symposium on Applied computing (SAC '04), **(2004)**, pp. 600-605.
[21] P. Wayner, "Compression Algorithms for Real Programmers", Morgan Kaufmann, Los Altos-CA, **(2000)**.
[22] I. H. Witten, A. Moffat and T. C. Bell, "Managing Gigabytes: Compressing and Indexing Documents and Images", Morgan Kaufmann, second edition, Los Altos-CA, **(1999)**.
[23] H. Yan, S. Ding and T. Suel, "Compressing Term Positions in Web Indexes", In: The 32nd Annual International ACM SIGIR Conference (SIGIR'09), **(2009a)**, pp. 147-154.
[24] H. Yan, S. Ding and T. Suel, "Inverted Index Compression and Query Processing with Optimized Document Ordering", In: Proceedings of the 18th international conference on World Wide Web (WWW '09), **(2009b)**, pp. 401 – 410.

[25] J. Zhang, X. Long and T. Suel, "Performance of Compressed Inverted List Caching in Search Engines", In: Proceedings of the 17th international conference on World Wide Web (WWW '08), **(2008)**, pp. 387 – 396.
[26] J. Zobel and A. Moffat, "Inverted Files for Text Search Engines", ACM Computing Surveys (CSUR), vol. 38, no. 2, **(2006)**, pp. 6, doi:10.1145/1132956.1132959.

Appendices

Appendix A: The Frequency Tree



Appendix B:

This appendix shows table3 of the frequency of Arabic letters from a sample Arabic text from a story (about 13,185 characters). Huffman code algorithm was run to encode these letters. The encoding will give the common used letters a shorter code as shown in the following table which was extracted from the frequency tree [Appendix A]

Table 3. The Frequency Table of Arabic Letters

i	Arabic Alphabet	Alphabet Frequency	Percentage	Huffman Code
1	(alif, "أ")	2305	17.48%	111
2	(laam, "ل")	1553	11.78%	010
3	(miim, "م")	843	6.39%	1010
4	(nuun, "ن")	840	6.37%	1001
5	(yaa', "ي")	839	6.36%	1000
6	(waaw, "و")	780	5.92%	0110
7	(haa', "هـ")	539	4.09%	0001
8	(raa', "ر")	503	3.81%	11011
9	(baa', "ب")	482	3.66%	11010
10	(faa', "ف")	455	3.45%	11000
11	(‘ayn, "ع")	451	3.42%	10111
12	(taa', "ت")	413	3.13%	10110
13	(qaaf, "ق")	405	3.07%	01111
14	(daal, "د")	397	3.01%	01110
15	(kaaf, "ك")	374	2.84%	00110
16	(siin, "س")	257	1.95%	00000
17	(haa', "ح")	240	1.82%	110011
18	(jiim, "ج")	198	1.50%	001111
19	(taa' marbuuta, "ة")	157	1.19%	001010
20	(dhaal, "ذ")	140	1.06%	001000
21	(alif maqSuura, "ى")	133	1.01%	000010
22	(xaa', "خ")	133	1.01%	000011
23	(taa', "ط")	104	0.79%	1100100
24	(shiin, "ش")	101	0.77%	0011101
25	(daad, "ض")	97	0.74%	0011100
26	(saad, "ص")	85	0.64%	0010111
27	(zaay, "ز")	82	0.62%	0010110
28	(thaa', "ث")	79	0.60%	0010011
29	(ghayn, "غ")	74	0.56%	0010010
30	(hamza, "ء")	63	0.48%	11001011
31	(yaa' seat, "ئ")	30	0.23%	110010100
32	(zaa', "ظ")	24	0.18%	1100101011
33	(waaw seat, "ؤ")	9	0.07%	1100101010

Authors

Ameen A. Al-Jedady, is a master graduate from Yarmouk University in Jordan. Originally from Yemen, Mr Ameen had both his B.sc and master degrees in Computer Information Systems from CIS department, IT faculty at Yarmouk University in Irbid, Jordan. His main research focus is in Information retrieval.

Mohammed Al-Kabi, born in Baghdad/Iraq in 1959. He obtained his Ph.D. degree in Mathematics from the University of Lodz/Poland (2001), his masters degree in Computer Science from the University of Baghdad/Iraq (1989), and his bachelor degree in statistics from the University of Baghdad/Iraq(1981). Mohammed Naji AL-Kabi is an assistant Professor in the Faculty of Sciences and IT, at Zarqa University. Prior to joining Zarqa University, he worked many years at Yarmouk University in Jordan, the Nahrain University and Mustanserya University in Iraq. AL-Kabi's research interests include Information Retrieval, Web search engines, Data Mining, Software Engineering & Natural Language Processing. He is the author of several publications on these topics. His teaching interests focus on information retrieval, Web programming, data mining, DBMS (ORACLE & MS Access).

Izzat Alsmadi is an associate professor in the CIS department at Yarmouk University, Irbid, Jordan. Born in Jordan 1972, Izzat Alsmadi had his master and phd in software engineering from North Dakota State University (NDSU), Fargo , USA in the years 2006 and 2008 respectively. His main areas of research include: software engineering, testing, metrics, and information retrieval.