UDC 004.021 DOI https://doi.org/10.35546/kntu2078-4481.2024.2.17

A. V. VASHCHENKO
Student
Kherson National Technical University
ORCID: 0009-0007-9941-7947

IE. A. DROZDOVA
Senior Lecturer at the Department of Computer Systems and Networks
Kherson National Technical University
ORCID: 0000-0003-0276-6387

O. O. PRYKHODKO
Senior Lecturer at the Department of Specialized Translation
and Foreign Languages
Kherson National Technical University
ORCID: 0000-0002-8732-3659

DETERMINING OPTIMAL COMPRESSION ALGORITHM FOR FILES OF DIFFERENT FORMATS

The aim of the article is to highlight possible areas of the use of compression algorithms in various fields. The article investigates the efficiency of the compression of files of different formats and provides recommendations on the optimal compression algorithms for specific scenarios.

The research was carried out empirically: compression algorithms were implemented at the software level, the programs were used to compress files of various formats, the size of the resulting files was determined, and a comparison of the efficiency of the methods was made.

Research results. The article presents a table of file sizes after compression using different compression algorithms, calculates compression ratios for each case, determines the average compression ratios for each compression algorithm, analyses the efficiency of compression algorithms, and identifies the optimal compression algorithms for files of different formats.

The scientific novelty of this work is an integrated approach to comparing compression algorithms by compression ratios on different file formats and the study of the combination of Huffman and LZ78 algorithms, which has not been widely studied before. This allows us to gain a deeper understanding of the process of compressing files of different formats and identify effective algorithms for specific data types. The analysis can contribute to the development and improvement of file compression methods and have practical applications in various fields, such as data storage and transmission, file compression, and improving the performance of information processing systems.

The practical significance of the work lies in its potential usefulness for various fields. It provides recommendations and conclusions on the selection of efficient file compression algorithms for different file formats. This can have a positive impact on data storage and transmission, data processing speed, software development, and multimedia data. Using optimal compression algorithms helps reduce file size, saves resources, and improves user experience.

Key words: *algorithm, compression, encoding, Shannon-Fano, Huffman, RLE, LZ78.*

А. В. ВАЩЕНКО
студент
Херсонський національний технічний університет
ORCID: 0009-0007-9941-7947

Є. А. ДРОЗДОВА
старший викладач кафедри комп'ютерних систем та мереж
Херсонський національний технічний університет
ORCID: 0000-0003-0276-6387

О. О. ПРИХОДЬКО
старший викладач кафедри галузевого перекладу та іноземних мов
Херсонський національний технічний університет
ORCID: 0000-0002-8732-3659

ВИЗНАЧЕННЯ ОПТИМАЛЬНОГО АЛГОРИТМУ СТИСНЕННЯ ДЛЯ ФАЙЛІВ РІЗНОГО ФОРМАТУ

Метою статті є висвітлення можливих напрямків використання алгоритмів стиснення у різних сферах. У статті описується дослідження ефективності стиснення файлів різного формату та надаються рекомендації щодо оптимальних алгоритмів стиснення для конкретних сценаріїв.

Дослідження проводилися емпіричним методом: було на програмному рівні реалізовано алгоритми стиснення, стиснуто файли, визначено розмір отриманих файлів і зроблено порівняння щодо ефективності методів.

Основні результати досліджень: у статті представлено таблицю розмірів файлів після стиснення за різними алгоритмами стиснення, підраховано коефіцієнти стиснення для кожного випадку, було визначено середні коефіцієнти стиснення для кожного алгоритму стиснення, проведено аналіз ефективності алгоритмів стиснення та визначено оптимальні алгоритми стиснення для файлів різного формату.

Наукова новизна даної роботи полягає у комплексному підході до порівняння алгоритмів стиснення за коефіцієнтами стиснення на різних форматах файлів та дослідження поєднання алгоритмів Хаффмана та LZ78, що раніше не було широко досліджено. Це дозволяє отримати більш глибоке розуміння процесу стиснення файлів різного формату та виявити ефективні алгоритми для конкретних типів даних. Наукова робота може сприяти розвитку та вдосконаленню методів стиснення файлів і мати практичне застосування у різних областях, таких як зберігання та передача даних, стиснення файлів і покращення продуктивності систем обробки інформації.

Практична значимість цієї роботи полягає у її потенційній користі для різних сфер. Вона надає рекомендації та висновки щодо вибору ефективних алгоритмів стиснення файлів різного формату. Це може мати позитивний вплив на зберігання та передачу даних, швидкість обробки даних, розробку програмного забезпечення та роботу з мультимедійними даними. Застосування оптимальних алгоритмів стиснення допомагає зменшити обсяг файлів, економить ресурси та поліпшує користувацький досвід.

Ключові слова: *алгоритми, стиснення, кодування, Шеннон-Фано, Хаффман, RLE, LZ78.*

Problem statement

Every year, the amount of digital information generated and processed is growing. This leads to the need to increase the capacity of data storage and increase the bandwidth in computer networks.

Although the modernization of computer networks and storage systems is inevitable, it is possible to use these systems rationally to avoid unnecessary modernization costs.

One solution to this problem is data compression. Data compression is the process of converting input data into a smaller volume that can be more conveniently stored and transmitted.

Data compression is an important element in the storage and transmission of information in the modern world. By compressing data, you can reduce the amount of information that needs to be stored or transmitted, thereby reducing storage costs and increasing data transfer speeds. Data compression is essential in industries where data processing, storage, and transmission are important. It helps to reduce the amount of data, save disk space, and reduce the cost of data storage and transmission. Today, data compression is used in all industries and in everyday life, although the end user may not think about it.

Existing data compression methods can be divided into two groups: lossless compression methods and lossy compression methods [1].

Lossless compression methods are compression methods where the encoded data can be recovered with bit accuracy. These methods can be used to compress any kind of data: video, audio, text, programs, etc.

Lossless compression methods are versatile because they allow you to compress both important documents that must be identical to the original version after decompression and multimedia. Although complex methods that combine lossy and lossless compression are usually used to compress multimedia data, lossless compression alone guarantees the maximum quality of this data after decompression, so this approach is also common and used.

The peculiarity of lossy compression methods is the impossibility of full data recovery. These methods are used when it is acceptable to have a difference between the data before compression and after decompression. Usually, the scope of lossy compression methods is limited to the compression of photo, video, and audio data.

It should be noted that it is impossible to compress data indefinitely. In the case of using lossy compression methods, information will be lost and its amount will tend to be zero. When using lossless compression methods, in the best-case scenario, the average code length can reach the value of the source entropy. But in practice, after several iterations of compression, the file may start to "grow", because, for example, in the case of Shannon-Fano encoding, the file must store not only the encoded sequence but also the data for reconstructing the binary tree that will be needed for decoding, as well as additional bits that are not used at all. Since the smallest unit of addressing is a byte, the sequence must be supplemented with bits to write to disk so that the total number of bits is a multiple of 8 and can be written as bytes.

Research publications

The analysis of research and publications comparing the efficiency of Shannon-Fano, Huffman, RLE, and LZ78 compression methods gives us important conclusions about their properties and applications.

Data compression is an important stage in the processing and transmission of information, especially in conditions of limited resources or limited bandwidth. The optimal data compression method depends on the specific context, such as the type of data, its properties, and size [2].

Studies show that the Huffman method is effective for compressing text data with uneven character distribution. It uses variable length codes, where frequent characters are represented by shortcodes and rare characters by long codes [3].

The Shannon-Fano method also uses variable-length codes but is based on recursively dividing characters in a set into subsets with close probabilities. This method shows good results but may be less efficient than the Huffman method.

RLE is a simple compression method that is based on replacing repeated sequences of characters with a single character followed by the number of repetitions. This method is well suited for data with a large number of repetitions, such as images with rich geometric structures. It can achieve high compression efficiency for such data types but has limited suitability for compressing other data types that do not have repeating sequences.

LZ78 is a compression method that is based on building a dictionary of repeated phrases during compression. It uses a combination of character code and references to previous phrases to create new phrases. LZ78 performs well for data with many repetitions and repeated phrases, such as text data and some types of images.

To summarise, studies show that the effectiveness of Shannon-Fano, Huffman, RLE, and LZ78 compression methods depends on the type of data to be compressed. Each of these methods has its own advantages and limitations. The optimal choice of compression method depends on the specific context and data compression requirements.

The purpose of the study is to investigate compression algorithms, and their software implementation, compare the efficiency of compressing files of different formats, and determine the feasibility and uses.

The main material

The Shannon-Fano algorithm is one of the earliest data compression algorithms developed by Claude Shannon and Robert Fano in 1948.

The Shannon-Fano algorithm counts the number of symbols in the original text to determine their frequency of occurrence and then builds a binary tree. This encoding method is prefix-based. This means that there is no code for which another code would be a prefix, which allows for decoding the encoded text without error. However, the disadvantage of the Shannon-Fano algorithm is that it does not always provide optimal text compression [4].

The Shannon-Fano algorithm is based on the idea of dividing the lists of symbols in the original text into two groups with a minimum difference in their total weight, which is then encoded with different prefixes. This process is repeated recursively for each group of symbols until each symbol is assigned its own code [5].

The Shannon-Fano coding algorithm:

Step 1: calculating the frequency of symbol occurrence;

Step 2: sorting the list of symbols according to frequency;

Step 3: divide the list of symbols into two parts, with the total frequency counts of the symbols included in them being as close to each other as possible. Add 1 to the code of one part and 0 to the other;

Step 4: consider each part separately. If there is more than one element in the resulting sublist, then perform Step 3.

It should be noted that many works mention sorting the list of symbols by descending frequency, but this does not matter. Also, when creating programs, the absolute frequency is counted, because counting the relative frequency requires additional calculations that are unreasonable in this context.

If the message was previously encoded by the ASCII encoding table, which encodes each symbol with one byte, the size of the message itself will be significantly reduced. but there is a need to store the information that will be necessary to restore the original message.

Since the bit length of a coded message is usually not a multiple of 8, it is necessary to supplement the message with bits to enable the last part of the message to be recorded on a storage device or transmitted over a computer network.

Since, for technical reasons, the message is supplemented with bits at the end, it is necessary to determine where the created message has ended. If this is not done, nothing will prevent the program from reading a few more bits and writing a couple more bytes to the file that is being restored after compression. This is unacceptable. In the proposed implementation, a special code sequence is created to indicate the end of the file. It is encoded along with all bytes, so it meets the prefix requirement and is the last to be written to the file.

The main problem with Shannon-Fano coding is that there is no guarantee of optimal coding, so the Huffman algorithm is used.

The Huffman algorithm was patented in 1952 by David Huffman. It is similar to the Shannon-Fano algorithm, but ensures optimal encoding of the message by guaranteeing that frequently occurring symbols will not have a code length longer than symbols occurring less frequently [6].

The algorithm builds a binary tree where the leaves are the nodes that represent the symbols. The tree is built from the leaves to the root.

Huffman coding algorithm:

Step 1: calculating the frequency of occurrence of symbols;

Step 2: finding the two nodes with the lowest frequency of occurrence;

Step 3: creating a new node with a frequency equal to the total frequencies of the found nodes. The resulting node is the parent of the found nodes. Adding 1 to the code of one of the nodes and 0 to the code of the other. Removing the found nodes from the search area;

Step 4: as long as there is more than one node, step 3.

The main difference between the Huffman algorithm and the Shannon-Fano algorithm is that in the Shannon-Fano algorithm, the encoding is from the root to the leaf, while in the Huffman algorithm, it is from the leaf to the root.

The sequence of bits in the codes in both cases is determined from the root to the leaves.

Similarly, as in the case of Shannon-Fano encoding, the encoded message will be a sequence of bits that is formed by writing the corresponding codes instead of symbols.

As with Shannon-Fano file compression, Huffman file compression requires writing to the file the data that will be needed to reconstruct the binary tree, adding an additional node during encoding, and, if necessary, writing additional bits at the end of the file.

The data for binary tree recovery is implementation-specific, so it may differ in different formats [7].

Another feature of the algorithm is that the codes and their lengths generated by the algorithm may also vary depending on the implementation but always have the same average codeword length and the same message length.

The Huffman algorithm provides an average codeword length close to the source entropy and a fairly high encoding/decoding rate, so it is still used, but usually in combination with other algorithms. In lossless compression methods, it is usually used in combination with sequence compression algorithms. For example, the Deflate algorithm is a combination of the LZ77 and Huffman algorithms. In lossy compression methods, it is used with lossy compression algorithms. For example, in JPEG, lossy compression is performed first, and then the blocks resulting from lossy coding are encoded with the Huffman algorithm.

The RLE algorithm was formulated in 1967 by an engineer and programmer Ivan Sutton. The idea of RLE encoding is to encode series lengths and write a symbol and a number to the output file, which indicates the number of consecutive repetitions of that symbol [8].

The RLE encoding algorithm:

Step 1: read the symbol, and identify it as a base symbol;

Step 2: set the iterator to 1;

Step 3: read the symbol. If it is identical to the base symbol, step 4, otherwise, step 5;

Step 4: increment the iterator, step 3;

Step 5: write the base symbol and counter;

Step 6: write the new c symbol as the base symbol, step 2.

This method is fast and uses minimal RAM, but it can only compress files where byte duplication is common. If bytes are not repeated in a row, the method can increase the file at least twice, depending on the size of the counter [9].

LZ78 is a data compression algorithm proposed by Abraham Lempel and Jakob Ziv in 1978 [10].

This algorithm works on replacing repeated text fragments with pointers to previous text fragments that have already been encountered.

The main idea of the algorithm is to create a dictionary from unique text fragments and their indexes in the dictionary. As you go through the text, the algorithm builds new dictionary elements by adding new unique fragments and also remembers each new text fragment and its index in the dictionary [11].

For data compression, the LZ78 algorithm uses index-symbol pairs, which can be encoded in fewer bits than individual symbols. Thus, during compression, repetitive text fragments can be encoded using replacement with the previous fragments with a pointer to their index in the dictionary.

Encoding algorithm using the LZ78 method:
Step 1: creating an empty dictionary;
Step 2: setting the index to zero;
Step 3: reading the symbol;
Step 4: creating an index-symbol pair;
Step 5: search for the index- symbols pair in the dictionary. If it is found, step 6, otherwise, step 7;
Step 6: set the index equal to the index of the found pair in the dictionary, step 3;
Step 7: if the dictionary is not full, add the index-symbol pair to the dictionary and save it to a file. Step 2.

An important nuance is that the size of the decoder dictionary must be at least as large as the encoder dictionary. If you encounter a reference to a dictionary item that exceeds the size of the dictionary, the behavior of the program may be unpredictable [12].

It is also important to note that the size of the dictionary should be limited to a specific value. In the case of using the list of symbols with automatic expansion, the program can quickly occupy RAM in large quantities. You should also consider the size of variables that store indexes because the size of the block that is written to the file greatly affects the file size and the variable should not overflow during operation [13].

To determine the compression efficiency, we selected 3 files of different sizes with the formats bmp, avi, exe, svg, txt, docx, wav, and mp3, which were compressed using the created programs. The results are shown in Table 1.

Table 1

Compression results

File type	Size, b	Size after compression, b			
		Shannon-Fano's	Huffman's	RLE	LZ78
bmp	66614	21279	21251	24326	15483
	224878	128175	127892	86338	63282
	2305078	1603667	1601713	2498194	1590789
avi	742478	349530	348123	437722	206271
	1480958	1191936	1186316	1908096	1073700
	2279794	2050837	2041972	3496664	1990515
exe	58188	37602	36323	77650	34476
	90624	69197	69017	154496	63831
	3422558	2337648	2304403	4807708	2553375
svg	22491	10972	10972	43468	15882
	31150	16426	16292	59122	22857
	1315818	652497	648407	2461306	720525
txt	429364	266866	265570	851266	241755
	959175	516991	515434	1911692	397134
	4205781	2545690	2540770	8343772	2216709
docx	15372	14974	14923	26330	19335
	75409	73919	73741	145008	89175
	812853	814900	812969	1608446	936618
wav	1073218	1017328	1015185	2137522	1152963
	5226766	4958190	4946632	10412130	5673471
	10406738	9835206	9822602	20730756	11207190
mp3	764176	767031	764660	1511636	880290
	2113939	2119861	2113266	4185290	2403408
	5289384	5300723	5285340	10477126	5980500

Based on the table of compression results, the average compression ratios were calculated for each compression method and a table was created based on this data (Table 2).

As can be seen from Tables 1 and 2, the LZ78 method is better suited for compressing bmp, avi, exe, and txt data formats, but when used with data formats that are already compressed, such as docx, wav, and mp3, the resulting file is larger than the original one.

At the same time, the Huffman algorithm proved to be better for compressing svg files and, when compressing previously compressed files, does not increase the resulting file by more than 0.1%.

Table 2

Table of average compression ratios

File type	**Average compression ratio**			
	Shannon-Fano's	**Huffman's**	**RLE**	**LZ78**
bmp	2.11	2.11	2.09	3.10
avi	1.49	1.50	1.04	2.04
exe	1.44	1.47	0.68	1.48
svg	1.99	2.00	0.53	1.54
txt	1.71	1.71	0.50	2.03
docx	1.01	1.02	0.54	0.84
wav	1.06	1.06	0.50	0.93
mp3	1.00	1.00	0.51	0.88

Taking into account the efficiency of LZ78 and the fact that the files do not become significantly larger when compressed with the Huffman algorithm, we compressed the files first with the LZ78 algorithm and then compressed the result with the Huffman method. The results are shown in Table 3.

Table 3

Results of compression by the LZ78 and Huffman algorithms

File type	**Size, b**	**LZ78 + Huffman's**	**Compression ratio**	**Average compression ratio**
bmp	66614	13806	4.83	3.39
	224878	59970	3.75	
	2305078	1437104	1.60	
avi	742478	193076	3.85	2.15
	1480958	1036176	1.43	
	2279794	1930033	1.18	
exe	58188	31153	1.87	1.62
	90624	59276	1.53	
	3422558	2334263	1.47	
svg	22491	13771	1.63	1.73
	31150	20037	1.55	
	1315818	654688	2.01	
txt	429364	229818	1.87	2.14
	959175	374366	2.56	
	4205781	2113515	1.99	
docx	15372	16902	0.91	0.92
	75409	79635	0.95	
	812853	894140	0.91	
wav	1073218	1123355	0.96	0.95
	5226766	5535567	0.94	
	10406738	10949817	0.95	
mp3	764176	841273	0.91	0.91
	2113939	2311717	0.91	
	5289384	5767211	0.92	

As can be seen from Table 3, data compression first by the LZ78 algorithm and then by the Huffman method gives a positive result – the average compression ratio has increased for all compressed files compared to compression by the LZ78 algorithm alone.

This means that it is appropriate to use the Huffman algorithm in combination with other data compression methods, and in this case, with sequence compression algorithms.

It is worth noting that when encoding with the Huffman algorithm after LZ78 encoding, the index-symbol pair was divided into 3 parts: two parts of the index and the symbol. If the index was encoded separately and the symbol separately, or if the indexes were read in full and encoded together with the symbols, the compression results would be different.

Also, the size of the dictionary and the size of the variable that stores the index when encoding with the LZ78 method are of great importance. In the proposed implementation, we used a dictionary of 65535 index-symbol pairs and a variable that takes up two bytes and can take values from 0 to 65535.

Increasing the size of the dictionary may result in fewer index-character pairs in the resulting file since the encoded sequences can be longer, but the indexes will require more memory.

Conclusions

RLE often leads to larger files and has compression rates that are significantly lower than those of LZ78, so its use is not advisable.

The Shannon-Fano algorithm has the same complexity but lower performance than the Huffman algorithm, so its use is also inappropriate.

LZ78 is more appropriate to use in combination with the Huffman algorithm to compress a lot of uncompressed data.

The Huffman algorithm compresses .svg files better than LZ78, does not lead to a significant increase in the resulting file size when compressing encoded or encrypted data, and provides opportunities for creating methods based on combining other algorithms.

Bibliography

1. Ващенко А. В., Дроздова Є. А. Вирішення проблеми зберігання даних за допомогою програми стиснення файлів. *Інформаційні системи та комп'ютерно-інтегровані технології: ідеї, проблеми, рішення – 2021*. 2021. С. 13–15.

2. Prudvi C., Muchahary D., Raghuvanshi A. S. Analysis of image compression techniques for iot applications. *2022 international conference on intelligent technologies (CONIT)*, м. Hubli, India, 24–26 черв. 2022 р. 2022. DOI:10.1109/conit55038.2022.9848206

3. Image compression using huffman coding scheme with partial/piecewise color selection / A. H. M. Z. Karim та ін. *2021 IEEE 4th international conference on computing, power and communication technologies (GUCON)*, м. Kuala Lumpur, Malaysia, 24–26 верес. 2021 р. 2021. DOI:10.1109/gucon50781.2021.9573863

4. Dharma Walidaniy W., Yuliana M., Briantoro H. Improvement of PSNR by Using Shannon-Fano Compression Technique in AES-LSB StegoCrypto. *2022 international electronics symposium (IES)*, м. Surabaya, Indonesia, 9–11 серп. 2022 р. 2022. DOI:10.1109/ies55876.2022.9888656

5. Pic X., Antonini M. A constrained Shannon-Fano entropy coder for image storage in synthetic DNA. *2022 30th european signal processing conference (EUSIPCO)*, м. Belgrade, Serbia, 29 серп. – 2 верес. 2022 р. 2022. DOI:10.23919/eusipco55093.2022.9909833

6. Gupta M. Alternatives to huffman coding by comparison to other algorithms. *2021 innovations in power and advanced computing technologies (i-pact)*, м. Kuala Lumpur, Malaysia, 27–29 листоп. 2021 р. 2021. DOI:10.1109/i-pact52855.2021.9696497

7. Compressed DNA coding using minimum variance huffman tree / P. Mishra та ін. *IEEE communications letters*. 2020. Т. 24, № 8. С. 1602–1606. DOI:10.1109/lcomm.2020.2991461

8. Image compression using run length encoding and its optimisation / A. Birajdar та ін. *2019 global conference for advancement in technology (GCAT)*, м. BANGALURU, India, 18–20 жовт. 2019 р. 2019. DOI:10.1109/gcat47503.2019.8978464

9. Saidani A., Xiang J., Mansouri D. A new lossless compression scheme for wsns using RLE algorithm. *2019 20th asia-pacific network operations and management symposium (APNOMS)*, м. Matsue, Japan, 18–20 верес. 2019 р. 2019. DOI:10.23919/apnoms.2019.8893093

10. Koppl D., Navarro G., Prezza N. HOLZ: high-order entropy encoding of lempel-ziv factor distances. *2022 data compression conference (DCC)*, м. Snowbird, UT, USA, 22–25 берез. 2022 р. 2022. DOI:10.1109/dcc52660.2022.00016

11. Puglisi S. J., Rossi M. On lempel-ziv decompression in small space. *2019 data compression conference (DCC)*, м. Snowbird, UT, USA, 26–29 берез. 2019 р. 2019. DOI:10.1109/dcc.2019.00030

12. Muller R. Linear computation coding inspired by the lempel-ziv algorithm. *2022 IEEE information theory workshop (ITW)*, м. Mumbai, India, 1–9 листоп. 2022 р. 2022. DOI:10.1109/itw54588.2022.9965875

13. Decompressing lempel-ziv compressed text / P. Bille та ін. *2020 data compression conference (DCC)*, м. Snowbird, UT, USA, 24–27 берез. 2020 р. 2020. DOI:10.1109/dcc47342.2020.00022

References

1. Vashchenko, A. V., & Drozdova, Ie. A. (2021). Vyrishennia problemy zberihannia danykh za dopomohoiu prohramy stysnennia failiv [Solving data storage problems with a file compression program]. *1st International Scientific and Practical Conference of IS and ICT – 2021, Information systems and computer-integrated technologies: ideas, problems, solutions*, 13–15 (in Ukr.).

2. Prudvi, C., Muchahary, D., & Raghuvanshi, A. S. (2022). Analysis of image compression techniques for iot applications. У *2022 international conference on intelligent technologies (CONIT)*. IEEE. https://doi.org/10.1109/conit55038.2022.9848206

3. Karim, A. H. M. Z., Miah, M. S., Al Mahmud, M. A., & Rahman, M. T. (2021). Image compression using huffman coding scheme with partial/piecewise color selection. У *2021 IEEE 4th international conference on computing, power and communication technologies (GUCON)*. IEEE. https://doi.org/10.1109/gucon50781.2021.9573863

4. Dharma Walidaniy, W., Yuliana, M., & Briantoro, H. (2022). Improvement of PSNR by Using Shannon-Fano Compression Technique in AES-LSB StegoCrypto. У *2022 international electronics symposium (IES)*. IEEE. https://doi.org/10.1109/ies55876.2022.9888656

5. Pic, X., & Antonini, M. (2022). A constrained Shannon-Fano entropy coder for image storage in synthetic DNA. У *2022 30th european signal processing conference (EUSIPCO)*. IEEE. https://doi.org/10.23919/eusipco55093.2022.9909833

6. Gupta, M. (2021). Alternatives to huffman coding by comparison to other algorithms. У *2021 innovations in power and advanced computing technologies (i-pact)*. IEEE. https://doi.org/10.1109/i-pact52855.2021.9696497

7. Mishra, P., Bhaya, C., Pal, A. K., & Singh, A. K. (2020). Compressed DNA coding using minimum variance huffman tree. *IEEE Communications Letters*, *24*(8), 1602–1606. https://doi.org/10.1109/lcomm.2020.2991461

8. Birajdar, A., Agarwal, H., Bolia, M., & Gupte, V. (2019). Image compression using run length encoding and its optimisation. У *2019 global conference for advancement in technology (GCAT)*. IEEE. https://doi.org/10.1109/gcat47503.2019.8978464

9. Saidani, A., Xiang, J., & Mansouri, D. (2019). A new lossless compression scheme for wsns using RLE algorithm. У *2019 20th asia-pacific network operations and management symposium (APNOMS)*. IEEE. https://doi.org/10.23919/apnoms.2019.8893093

10. Koppl, D., Navarro, G., & Prezza, N. (2022). HOLZ: High-order entropy encoding of lempel-ziv factor distances. У *2022 data compression conference (DCC)*. IEEE. https://doi.org/10.1109/dcc52660.2022.00016

11. Puglisi, S. J., & Rossi, M. (2019). On lempel-ziv decompression in small space. У *2019 data compression conference (DCC)*. IEEE. https://doi.org/10.1109/dcc.2019.00030

12. Muller, R. (2022). Linear computation coding inspired by the lempel-ziv algorithm. У *2022 IEEE information theory workshop (ITW)*. IEEE. https://doi.org/10.1109/itw54588.2022.9965875

13. Bille, P., Berggren Ettienne, M., Gagie, T., Li Gortz, I., & Prezza, N. (2020). Decompressing lempel-ziv compressed text. У *2020 data compression conference (DCC)*. IEEE. https://doi.org/10.1109/dcc47342.2020.00022