UDC 004.627
DOI https://doi.org/10.32782/IT/2025-1-22

Sviatoslav MYCHKA
Postgraduate Student at the Department of Software Engineering, Kharkiv National University of Radio Electronics, 14, Nauky Ave., Kharkiv, Ukraine, 61166, sviatoslav.mychka@nure.ua
***ORCID:** 0009-0005-0292-6522*

Nataliya GOLYAN
Candidate of Technical Sciences, Associate Professor at the Department of Software Engineering, Kharkiv National University of Radio Electronics, 14, Nauky Ave., Kharkiv, Ukraine, 61166, nataliia.golian@nure.ua
***ORCID:** 0000-0002-1390-3116*
***Scopus Author ID:** 56007783500*

To cite this article: Mychka, S., Golyan, N. (2025). Metody stysnennia binarnykh danykh bez vtrat dlia pidvyshchennia productyvnosti prohramnykh system [Lossless binary data compression methods for increasing the productivity of software systems]. *Information Technology: Computer Science, Software Engineering and Cyber Security*, 162–168, doi: https://doi.org/10.32782/IT/2025-1-22

LOSSLESS BINARY DATA COMPRESSION METHODS FOR INCREASING THE PRODUCTIVITY OF SOFTWARE SYSTEMS

The article describes methods used for data compression without considering formats or types. Data compression is the process of encoding data to reduce its size; lossless data compression means that the reverse decoding process restores the data in its original form. There are limitations to lossless data compression that depend on the information entropy of the message: the lower it is, the greater the potential compression ratio of this data. Data with high entropy, for example, random or previously compressed with a sufficiently optimal encoding, cannot be compressed.

*The work **aims** to investigate using data compression algorithms with different types of information, formats or information entropy. Lossless data compression algorithms are divided into subcategories, in particular, dictionary and entropy, which differ in the principle of operation. Dictionary and entropy methods can also be combined to increase compression efficiency.*

*The **scientific novelty** lies in finding patterns between the algorithms used and the data compression ratios of specific formats. For the first time, data on many different data compression algorithms, both independent and those consisting of others, were processed and systematized. As a result, data were obtained on compression methods best suited for compressing images, Internet pages, source code, and other widely used formats.*

*The research **methodology** is based on measuring several characteristics of the original and compressed files and the operation of algorithms with subsequent comparison of this data. Therefore, files can be distinguished by format and information entropy before compression. After compression, a compression ratio can be found that characterizes the efficiency of the algorithms. The study involves universal algorithms that perceive information as a specific sequence of bytes. Thus, they can be applied to various file formats, including those most often used in distributed data storage systems.*

*The **conclusion** contains practical recommendations for the application of data compression algorithms. The data obtained during the study can be used for integration into other software products or further analysis.*

***Key words:** compression, algorithm, entropy, software system, C#, ASP.NET.*

Святослав МИЧКА
аспірант кафедри програмної інженерії, Харківський національний університет радіоелектроніки, просп. Науки, 14, м. Харків, Україна, 61166
***ORCID:** 0009-0005-0292-6522*

Наталія ГОЛЯН
кандидат технічних наук, доцент кафедри програмної інженерії, Харківський національний університет радіоелектроніки, просп. Науки, 14, м. Харків, Україна, 61166
***ORCID:** 0000-0002-1390-3116*
***Scopus Author ID:** 56007783500*

Бібліографічний опис статті: Мичка, С., Голян, Н. (2025). Методи стиснення бінарних даних без втрат для підвищення продуктивності програмних систем. *Information Technology: Computer Science, Software Engineering and Cyber Security*, 162–168, doi: https://doi.org/10.32782/IT/2025-1-22

МЕТОДИ СТИСНЕННЯ БІНАРНИХ ДАНИХ БЕЗ ВТРАТ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ПРОГРАМНИХ СИСТЕМ

У статті описані методи, що застосовуються для стискання даних без урахування форматів, типів тощо. Стискання даних – це процес їхнього кодування, що має на меті зменшення їхнього обсягу; стискання даних без втрат означає, що обернений процес декодування відновлює дані в початковому вигляді. Для стискання даних без втрат існують обмеження, що залежать від інформаційної ентропії повідомлення: чим вона нижча, тим більшим є потенційний коефіцієнт стиснення цих даних. Дані з високою ентропією, наприклад, випадкові або попередньо стиснуті достатньо оптимальним кодуванням, не можна стиснути.

***Метою роботи** є дослідити використання алгоритмів стиснення даних із різними видами інформації, форматами, інформаційною ентропією тощо. Алгоритми стиснення даних без поділяються на підкатегорії, зокрема на словникові та ентропійні, що розрізняються за принципом дії. Також словникові та ентропійні методи можуть бути об'єднані для підвищення ефективності стискання.*

***Наукова новизна** полягає в знаходженні закономірностей між алгоритмами, що застосовуються, та коефіцієнтами стиснення даних певного формату. Вперше оброблено та систематизовано дані про велику кількість різних алгоритмів стиснення даних, як самостійних, так і таких, що складаються з інших. У результаті отримано дані про методи стиснення даних, що найкраще підходять для стискання зображень, інтернет-сторінок, вихідного коду тощо.*

***Методологія** дослідження базується на вимірюванні кількох характеристик оригінальних та стиснених файлів, а також роботи алгоритмів із подальшим співставленням цих даних. Так, файли можуть бути розрізнені за форматом та інформаційною ентропією, а після стиснення можна знайти коефіцієнт стиснення, що характеризує ефективність роботи алгоритмів. В дослідженні беруть участь універсальні алгоритми, що сприймають будь-яку інформацію як певну послідовність байтів. Таким чином, вони можуть бути застосовані до різних форматів файлів, зокрема таких, що найчастіше використовуються в розподілених системах зберігання даних.*

***Висновок** містить практичні рекомендації по застосуванню алгоритмів стиснення даних. Дані, отримані в ході дослідження, можуть бути використані для інтегрування в інші програмні продукти, або подальшого аналізу.*

***Ключові слова:** стиснення даних, алгоритм, ентропія, програмна система, C#, ASP.NET.*

Introduction. In the modern world, the amount of new data is constantly growing. Accordingly, the costs of storing and transporting such volumes of data are also growing. Modern data compression algorithms include entropy (frequency), dictionary, and context mixing. Often, algorithms complement each other, significantly improving compression ratios. One of the well-known examples of this is Deflate, which includes two others – LZ77 and the optimal Huffman code.

A data compression algorithm is a data encoding that reduces their occupied volume. Usually, such transformations are reversible; that is, they have an inverse action called recovery or decompression, which consists of partial or complete recovery of the original data. Data compression is based on reducing redundancy, so entropy coding encodes symbols that occur more often in the data into shorter sequences, usually less than 8 bits. The relevance of developing and improving data compression algorithms is that more substantial data compression with the ability to accurately (using lossless compression algorithms) restore the original information allows for reducing the costs of their storage, transportation and processing.

Topicality of the research. The main characteristic of data compression algorithms is the compression ratio – the ratio of the original data's size to the data's size after the algorithm has done its work. If the algorithm works with a ratio of 1, it does not perform valuable work since the data volume does not decrease; that is, compression is not performed. Such property is native to algorithms such as the Burroughs-Wheeler transform, which is not a compression algorithm but can be used as an intermediate stage in data preparation for other algorithms. If the algorithm works with a coefficient less than 1, it performs a disservice; instead of compressing, it actually increases the volume of input data.

The topicality of developing and improving data compression algorithms lies, first of all, in the fact that stronger data compression with the ability to accurately (using lossless compression algorithms) restore the original information allows reducing the costs of their storage, transportation and processing. Therefore, there are competitions like Hutter Prize or Large Text Compression Benchmark.

Literature review. To determine the effectiveness of lossless data compression

algorithms, the compression ratio is determined – the ratio of the initial size of uncompressed data to the size of the data at the output of the algorithm (1).

$$k = \frac{V_0}{V_{compr}}. \quad (1)$$

The mean entropy of the message is calculated using Shannon entropy.

$$H(X) = -K\sum_{i=1}^{n} p_i \cdot \log_2 p_i, \quad (2)$$

where $H(X)$ is the average entropy of the message; K is a constant coefficient that determines the units of measurement of entropy, and its change is equivalent to a change in the base of the logarithm; p_i is the frequency of occurrence of the i-th symbol in the message; n is the power of the dictionary, which is actually equal to the number of symbols in it (Shannon, 1948, p. 12).

N i.i.d. random variables each with entropy $H(X)$ can be compressed into more than N $H(X)$ bits with negligible risk of information loss, as $N \to \infty$; conversely if they are compressed into fewer than N $H(X)$ bits it is virtually certain that information will be lost (McKay, 2003, p. 92). This enables lossless compression algorithms, which reduce this redundancy and assign new codes to the symbols, according to the amount of information they carry.

Although lossless data compression cannot bypass boundaries set by information entropy, new ways of finding the best suitable encoding methods are discovered almost every year. Feature-based data compression concept is becoming more popular nowadays, where features means significant pieces of information, important for a human or a machine (Podgorelec, 2024). The methods that involve context modeling for symbols or larger entities prediction are gaining popularity, too, if compression ratio is more important than speed (Mohideen, 2021). The context modeling concept developed into using machine learning (Lopes, 2022).

For distributed platforms, where the data is anticipated to be accessed much more than uploaded, speed of decompression becomes much more important than the compression ratio. Several algorithms exist to achieve fast decompression (Collet, 2011).

The aim of the research. Based on the analysis of the subject area, the following tasks can be set for solving during the research:

– investigate lossless compression algorithms for various types of data, in particular text, video, static images, etc., and their combinations;

– determine the criteria for comparing the algorithms under study and use them to compare the effectiveness of applying compression methods to certain types of files;

– using the obtained data, develop an application that will select the necessary compression algorithm for the data provided by the user.

This set of tasks aims to improve understanding of data compression methods. During their solution and additional experiments, different algorithms and their combinations will be compared, which will allow assessing the usefulness of both newer and older data compression methods. Therefore, the work aims to investigate using data compression algorithms with different types of information, formats or information entropy.

Overview of data compression methods. The main requirement for lossless compression algorithms, unlike lossy ones, is the ability to fully recover the data that has been compressed. Accordingly, each algorithm has its own mechanism for ensuring full reversibility of compression operations.

The standard **LZ77** algorithm encodes repeated sequences as two numbers – offset and match length. For each sequence of symbols $x[i]..x[j]$, where i, j – indices of the symbols in a message, the algorithm checks whether such sequence is present in the buffer. If a match is found, then sequence $x[i]..x[j+1]$ is checked, until for some sequence $x[i]..x[j+1]$ the match is no longer found. If no match is found, the algorithm finds the offset from the last index of the sliding buffer to the first index of the sequence c, and match length l. After this, the sequence $x[i]..x[j+1]$ is written to the end of the buffer, and the three values c, l and $x[j]$ are written to the match (Ziv, Lempel, 1977).

The decoding of the values is performed backwards. For each (c, l, $x[j]$) write to the buffer the sequence that starts on the c index from the end and lasts l symbols, and write $x[j]$ to the end of the buffer. If $l > c$, the sequence is repeated cyclically.

Figures 1 depicts the processes of encoding of a sample text “abcabdabdabf”.

Huffman algorithm, unlike LZ77, is an entropy compression method. It consists of two parts: constructing an optimal code tree and generating the encoded sequence. In a non-adaptive algorithm for constructing an optimal code tree, it is necessary to calculate the frequency of each symbol and, using the obtained table (dictionary) of frequencies and taking the symbols as the leaves of the tree, apply the following algorithm:

1. select two nodes with the lowest frequencies, which are taken as their «weight» and create a parent node from them, the weight of which will

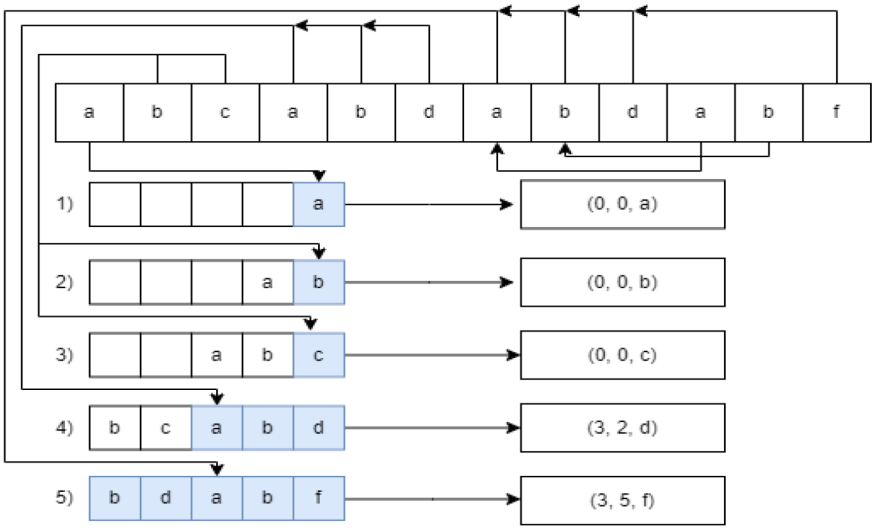


Fig. 1. LZ77 Encoding process

be the sum of the weights of the selected child elements;

2. for the left and right arcs from the parent element to the children, 0 and 1 are assigned;

3. steps 1–2 are repeated, and now the newly created node will be in the list of nodes free for selection, and its child elements become unavailable for selection;

4. when only one free node remains in the tree, the tree is considered complete, and this node is its root.

Thus, the tree is built from the leaves to the root. Figure 2 shows an example of building a tree for a certain symbol frequency table.

After building the tree, we get a scheme by which any symbol from the dictionary is encoded. To encode a message, the algorithm traverses the tree from the corresponding leaf to the root and, at each transition to the code, adds 1 or 0, respectively. Having reached the root, the algorithm reverses the sequence, obtaining the desired code.

For decoding using a non-adaptive algorithm, it is necessary to have data on symbol frequencies in order to build a tree similar to that presented in Figure 3. Then, having the compressed data and taking into account that all codes are prefixes, we can apply the following algorithm:

1. take the next bit from the sequence and add it to the buffer;

2. if the value in the buffer matches the symbol code, write this symbol;

3. repeat until the end of the encoded message is reached.

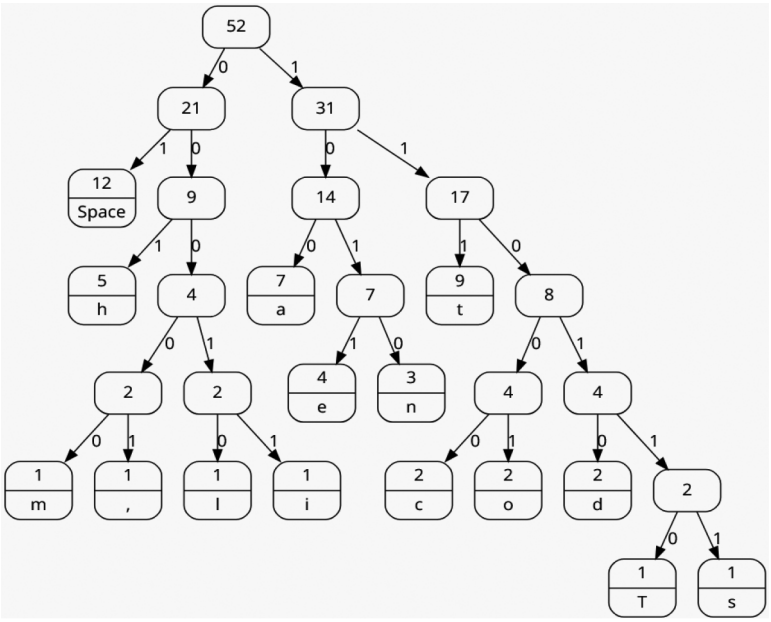
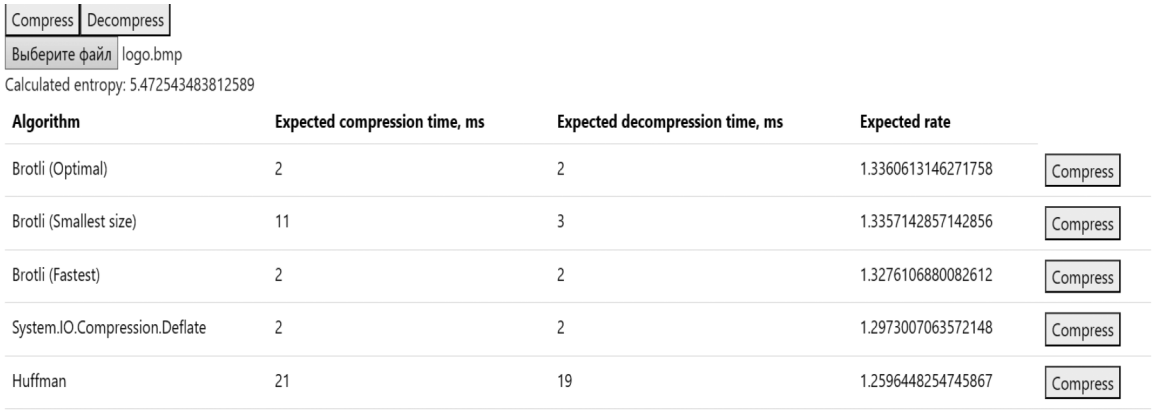


Fig. 2. Huffman tree


Compress | Decompress

Выберите файл logo.bmp

Calculated entropy: 5.472543483812589

Algorithm	Expected compression time, ms	Expected decompression time, ms	Expected rate	
Brotli (Optimal)	2	2	1.3360613146271758	Compress
Brotli (Smallest size)	11	3	1.3357142857142856	Compress
Brotli (Fastest)	2	2	1.3276106880082612	Compress
System.IO.Compression.Deflate	2	2	1.2973007063572148	Compress
Huffman	21	19	1.2596448254745867	Compress

Fig. 3. User interface with the demonstration of the research results

Deflate compression method combines LZ77 and Huffman compression to achieve higher compression ratios (Deutsch, 1996). A typical Deflate stream consists of blocks, each preceded by a three-bit header that specifies whether the block is the last one and the encoding type. Thus, the first bit, set to zero, indicates that the block is not the last one, and vice versa, a one indicates that the block in question is the last one in the stream. The next two bits of the block specify the type of Huffman encoding that was applied to the data in it.

Deflate allows to combine the dictionary LZ77 algorithm that removes the repetitions in blocks, and then compress the result with Huffman optimal encoding. It ensures that the data with nearly equal distribution of symbols will still be compressed.

Brotli comes as a program with a static dictionary containing over 10,000 words and terms from English and some other languages and their transformations, determined by prefixes and suffixes (Alakuijala, 2016). The algorithm uses them to recognize points in the code and does not read the entire code, instead it finds points of interest in text, determines a certain term from them and moves on.

By using the dictionary, Brotli has significantly improved Gzip compression, which does not use it. Brotli's work is also based on the use of LZ77 and optimal Huffman coding, but Brotli also uses a context modeling algorithm. In total, the algorithm has four context determination modes available to it, which determine its identifier. They are based on the last two bytes of the stream p1 and p2. At the beginning of the stream, these two bytes are initialized as zero. The context is determined by six least or most significant bits of p1 (LSB6, MSB6), or a composite function (UTF8, Signed).

Concerning context modelling method, the **PPM (Prediction by Partial Matching)** algorithm is a special case of it. Its basic idea is to find the context of the current symbol from some previous ones. The model does not perform any compression but only predicts the value of the symbol being compressed using entropy compression methods such as optimal Huffman or arithmetic coding.

Usually, the length of the context used is limited. It is denoted as n, and the n-order model is denoted as $PPM(n)$. If the context of n symbols cannot be predicted, then a context of $n-1$ symbols is used. The transitions are recursive until a context is found or until n becomes 0. PPM represents a variant of the shuffling strategy, where probability estimates made using contexts of different lengths are combined into a total probability (Cleary, 1997). Then, this estimate is encoded by some entropy encoder, usually optimal Huffman coding or arithmetic. Compression occurs at the stage of using entropy coding.

Testing of combined compression algorithms. The value of the average entropy is maximum for messages in which the frequencies of occurrence of all symbols from the dictionary are equal. By the formula (3),

$$\max(H) = \frac{-n \cdot 1}{n} \cdot \log_2 \frac{1}{n} = -\log_2 \frac{1}{n}, \qquad (3)$$

where $\max(H)$ is the maximum average message entropy for a dictionary of n symbols, which all occur with frequency of $\frac{1}{n}$.

For messages using more than one character, the minimum average entropy can be obtained when the difference between the character frequencies is the largest. Assuming that each character from the dictionary occurs at least once in the message and the message is of constant length, we have the highest possible frequency of a single character:

$$\max(p) = 1 - \frac{d-1}{n} = \frac{n-d+1}{n}, \quad n \geq d, \qquad (4)$$

where max(p) is the maximum possible symbol frequency, d is the dictionary power, n is the message length.

Formulas (3) and (4) are used for generating test data. Balancing the dictionary volume and frequencies of symbols, we can achieve the desired entropy for the test data. Also, real files with different formats can be added to compare the testing with real data.

A separate application was created for the research, which includes methods for data management and implementation of compression algorithms and libraries. In particular, for algorithms such as Deflate and Brotli, the System. IO.Compression library was used. For others, separate open-source implementations were used (PPM, LZ77). Huffman coding was implemented independently using.NET 8 and C# (Ding, 2024).

For each entropy value from 0 to 9, data of different sizes was generated, defined by categories from 100 bytes to 1 megabyte. The file size may differ from the size declared in the category by no more than 6 % for data of 100 bytes and no more than 2 % for others, except for the exceptions described below.

According to formula (3), the maximum average message entropy achieved using 100 bytes is $-\log_2 \frac{1}{100} \approx 6{,}64$ bits per symbol. Therefore, to achieve entropy close to 7 and 8, respectively, 110 and 250 bytes are used, which gives the average message entropy $-\log_2 \frac{1}{110} \approx 6.78$ and $-\log_2 \frac{1}{250} \approx 7{,}96$ bits per symbol, respectively. Files of other sizes are generated without changes using the same symbol rates.

Results. To demonstrate the results obtained from the experiments, an application was developed that allows the analysis of files provided by the user and the determination of the algorithms whose application will give the highest compression ratios. On the client side, the average entropy of the data he provided, the file size and its extension are calculated and sent to the server. The algorithm on the server side selects algorithms with the highest compression ratios for such a file, after which the client part offers the user to compress the provided file with one of these algorithms. The compressed file can also be restored using another interface.

To implement the server part ASP.NET and the C# programming language were used. Data is obtained from a previously created MS SQL database, communication occurs using the Entity Framework ORM. The client part was developed using React using the Axios library to process GET and POST requests. Figure 3 shows the example of the user interface of the demonstration program.

The system developed during the research will help users find the compression algorithm that is fast enough and gives the optimal compression ratio. It is also open for future enhancements like adding new properties or algorithms.

Conclusions. Despite limits set by fundamental theory of information, lossless compression remains topical. The methods that combine old and well-tried compression algorithms with state-of-the-art concepts like feature-based coding or prediction algorithms based on context modeling or machine learning gained popularity for combining high compression speed and ratios.

This paper represents the methods of data compression and the methods of their categorization. The results are used in the demonstration program to predict the compression ratio and time based on previous experiments. The predictions made by the algorithm are precise in most of the cases, which proves that mean entropy of a message is the main concept in its compression.

Future researches of this topic could be dedicated to the algorithms utilizing context modeling and machine learning. Near-lossless compression remains topical, too, especially combined with lossless compression methods and utilized on images, audio etc (Jeromel, 2019).

BIBLIOGRAPHY:

1. Shannon C. E. A mathematical theory of communication. *Bell System Technical Journal*. 1948. Vol. 27, no. 4. P. 623–656. https://doi.org/10.1002/j.1538-7305.1948.tb00917.x (date of access: 19.03.2025).

2. McKay D. J. C. Information theory, inference & learning algorithms. Cambridge, UK : Cambridge University Press, 2003. 640 p.

3. State-of-the-Art Trends in Data Compression: COMPROMISE Case Study / D. Podgorelec et al. *Entropy*. 2024. Vol. 26, no. 12. P. 1032. https://doi.org/10.3390/e26121032 (date of access: 20.03.2025).

4. Mohideen R. M. K., Peter P., Weickert J. A systematic evaluation of coding strategies for sparse binary images. *Signal Processing: Image Communication*. 2021. Vol. 99. P. 116424. https://doi.org/10.1016/j.image.2021.116424 (date of access: 22.03.2025).

5. Collet Y. LZ4 Block Format Description. GitHub. URL: https://github.com/lz4/lz4/blob/dev/doc/lz4_Block_format.md (date of access: 22.03.2025).

6. Ziv J., Lempel A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*. 1977. Vol. 23, no. 3. P. 337–343. https://doi.org/10.1109/tit.1977.1055714 (date of access: 22.03.2025).

7. Deutsch P. DEFLATE Compressed Data Format Specification version 1.3. RFC Editor, 1996. https://doi.org/10.17487/rfc1951 (date of access: 23.03.2025).

8. Alakuijala J., Szabadka Z. Brotli compressed data format. RFC Editor, 2016. https://doi.org/10.17487/rfc7932 (дата звернення: 23.03.2025).

9. Chapter 7: Collecting User Input with Forms. *Web Applications with ASP.NET Core Blazor*. 2024. P. 129–160. https://doi.org/10.1515/9781501519475-010 (date of access: 25.03.2025).

10. Jeromel A., Žalik B. An efficient lossy cartoon image compression method. *Multimedia Tools and Applications*. 2019. Vol. 79, no. 1–2. P. 433–451. https://doi.org/10.1007/s11042-019-08126-7 (date of access: 25.03.2025).

REFERENCES:

1. Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(4), 623–656. https://doi.org/10.1002/j.1538-7305.1948.tb00917.x

2. MACKAY, D. J. C. (2003). Information Theory, Inference & Learning Algorithms. Cambridge University Press.

3. Podgorelec, D., Strnad, D., Kolingerová, I., & Žalik, B. (2024). State-of-the-Art Trends in Data Compression: COMPROMISE Case Study. *Entropy*, 26(12), 1032. https://doi.org/10.3390/e26121032

4. Mohideen, R. M. K., Peter, P., & Weickert, J. (2021). A systematic evaluation of coding strategies for sparse binary images. *Signal Processing: Image Communication*, 99, 116424. https://doi.org/10.1016/j.image.2021.116424

5. Collet, Y. LZ4 Block Format Description GitHub. Retrieved from: https://github.com/lz4/lz4/blob/dev/doc/lz4_Block_format.md

6. Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23 (3), 337–343. https://doi.org/10.1109/tit.1977.1055714

7. Deutsch, P. (1996). DEFLATE Compressed Data Format Specification version 1.3. RFC Editor. https://doi.org/10.17487/rfc1951

8. Alakuijala, J., & Szabadka, Z. (2016). Brotli Compressed Data Format. RFC Editor. https://doi.org/10.17487/rfc7932

9. Chapter 7: Collecting User Input with Forms. (2024b). In Web Applications with ASP.NET Core Blazor (p. 129–160). De Gruyter. https://doi.org/10.1515/9781501519475-010

10. Jeromel, A., & Žalik, B. (2019). An efficient lossy cartoon image compression method. *Multimedia Tools and Applications*, 79 (1–2), 433–451. https://doi.org/10.1007/s11042-019-08126-7