ИНФОРМАТИКА, ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И УПРАВЛЕНИЕ
INFORMATION TECHNOLOGY, COMPUTER SCIENCE, AND MANAGEMENT

UDC 519.683.4 https://doi.org/10.23947/1992-5980-2019-19-1-86-92

Arithmetic coder optimization for compressing images obtained through remote probing of water bodies [***]

R. V. Arzumanyan[1**]
[1] Institute of Computer Technology and Information Security, Southern Federal University, Taganrog, Russian Federation

Оптимизация арифметического кодера для сжатия изображений, полученных при дистанционном зондировании водных объектов[*]

Р. В. Арзуманян[1**]
[1] Институт компьютерных технологий и информационной безопасности Южного федерального университета, г. Таганрог, Российская Федерация

Introduction. The fast program algorithm of arithmetic coding proposed in the paper is for the compression of digital images. It is shown how the complexity of the arithmetic coder algorithm depends on the complexity measures (the input size is not considered). In the course of work, the most computationally complex parts of the arithmetic coder algorithm are determined. Performance optimization of their software implementation is carried out. Codecs with the new algorithm compress photo and video records obtained through the remote probing of water bodies without frame-to-frame difference.
Materials and Methods. In the presented paper, a selection of satellite images of the Azov Sea area was used. At this, the software algorithm of the arithmetic coder was optimized; a theoretical study was conducted; and a computational experiment was performed.
Research Results. The performance of the software implementation of the arithmetic coder is increased by the example of the VP9 video codec. Numerous launches of reference and modified codecs were made to measure the runtime. Comparison of the average time of their execution showed that the modified codec performance is 5.21% higher. The overall performance improvement for arithmetic decoding was 7.33%.
Discussion and Conclusions. Increase in the speed of the latest digital photo and video image compression algorithms allows them to be used on mobile computing platforms, also as part of the onboard electronics of unmanned aerial vehicles. The theoretical results of this work extend tools of the average-case complexity analysis of the algorithm. They can be used in case where the number of algorithm steps depends not only on the input size, but also on non-measurable criteria (for example, on the common RAM access scheme from parallel processors).

Введение. Предложенный в статье быстрый программный алгоритм арифметического кодирования предназначен для сжатия цифровых изображений. Показано, каким образом сложность алгоритма арифметического кодера зависит от критериев сложности (при этом размер входа не учитывается). В процессе работы определены наиболее вычислительно сложные части алгоритма арифметического кодера. Выполнена оптимизация производительности их программной реализации.
Кодеки с новым алгоритмом сжимают без учета межкадровой разницы фото- и видеоматериалы, полученные при дистанционном зондировании водных объектов.
Материалы и методы. В представленной научной работе использована подборка спутниковых снимков акватории Азовского моря. При этом оптимизирован программный алгоритм арифметического кодера, проведено теоретическое исследование, выполнен вычислительный эксперимент.
Результаты исследования. Увеличена производительность программной реализации арифметического кодера на примере видеокодека VP9. Для измерения времени выполнения произведены многочисленные запуски эталонного и модифицированного кодеков. Сравнение среднего времени их исполнения показало, что производительность модифицированного кодека на 5,21 % выше. Прирост общей производительности для арифметического декодирования составил 7,33 %.
Обсуждение и заключения. Увеличение скорости работы новейших алгоритмов сжатия цифровых фото- и видеоизображений позволяет применять их на мобильных вычислительных платформах, в том числе в составе бортовой электроники беспилотных летательных аппаратов. Теоретические результаты данной работы расширяют методы анализа сложности алгоритма в среднем случае. Они могут использоваться в ситуации, когда количество шагов алгоритма зависит не только от размеров входа, но и от неизмеримых критериев (например, от схемы обращения к общей оперативной памяти со стороны параллельных процессоров).

[*] The research is done within the frame of RSF project no. 17-11-01286.
[**] E-mail: roman.arzum@gmail.com
[***] Работа выполнена в рамках проекта РНФ 17-11-01286.

Keywords: arithmetical coding, performance optimiza tion, image compression, average-case algorithm complexity, video codec.

Ключевые слова: арифметическое кодирование,оптимизация производительности, сжатие изображений, сложность алгоритма в среднем, видеокодек.

For citation: R.V. Arzumanyan. Arithmetic coder optimization for compressing images obtained through remote probing of water bodies. Vestnik of DSTU, 2019, vol. 19, no. 1, pp. 86–92. https://doi.org/10.23947/1992-5980-2019-19-1-86-92

***Образец для цитирования*:** Арзуманян, Р. В. Оптимизация арифметического кодера для сжатия изображений, полученных при дистанционном зондировании водных объектов / Р. В. Арзуманян // Вестник Донского гос. техн. ун-та. — 2019. — Т. 19, № 1. — С. 86–92. https://doi.org/10.23947/1992-5980-2019-19-1-86-92

Introduction. Monitoring of the condition of the water body is often carried out through unmanned aerial vehicles (UAV), conducting aerial photography in the visible and infrared bands. The performance capabilities of the mobile camera, which the UAV has, impose a number of restrictions on the equipment that processes and stores the footage until the UAV returns. Specifically, the following factors should be considered.

1. Energy efficiency of the equipment that encodes the footage since it depends on the off-line operation time of the UAV.

2. Coding gain of photo and video data during the flight. High resolution images occupy a significant amount of the software memory, and this limits the amount of information that a UAV can accumulate.

Virtually all equipment for photo and video hardware supports the most common codec for compressing JPEG images. However, it is inferior to the most modern HEVC and VP9 codecs, which not only support video sequences, but also allow for better compression of single images. Thus, GoogleVP9 shows similar JPEG visual quality by the SSIM metric (structural similarity) and at the same time compresses images by 25–34% stronger [1].

Compared to the same JPEG, HEVC codec enables to improve the compression rate by 10–44% in PSNR metric (peak signal-to-noise ratio) [2]. However, a high compression ratio with a smaller bitstream size results in greater computational complexity of the HEVC and VP9 codecs [3, 4]. The architecture of the JPEG codec is presented in general in Fig. 1.

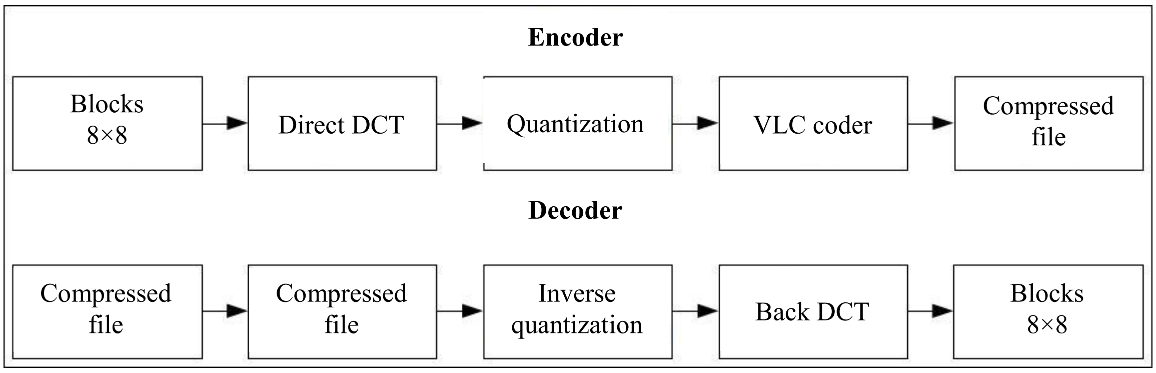

Fig. 1. JPEG codec flow chart

Here, VLC means compression through variable-length codes (variable length coding). The input frame is divided into blocks of fixed size (8×8). Each of them is subjected to direct discrete cosine transform (DCT), quantization of transform coefficients and subsequent entropic compression using the Huffman algorithm [4]. Discrete cosine transform is performed in the integer form [5]. Since the adoption of the JPEG standard in 1992, a number of fast algorithms have been developed. They allow the conversion to be carried out entirely in the CPU registers. Huffman's entropy compression is also not a complicated problem, so, even mobile processors in program mode can compress and decode JPEG images [6].

HEVC [6] and VP9 are fundamentally similar, and they are hybrid block codecs with splitting a frame into blocks of indeterminate length, intraframe prediction, discrete transform, and subsequent filtering to eliminate blocking artifacts. The blockdiagram of the HEVC codec is shown in Fig. 2.

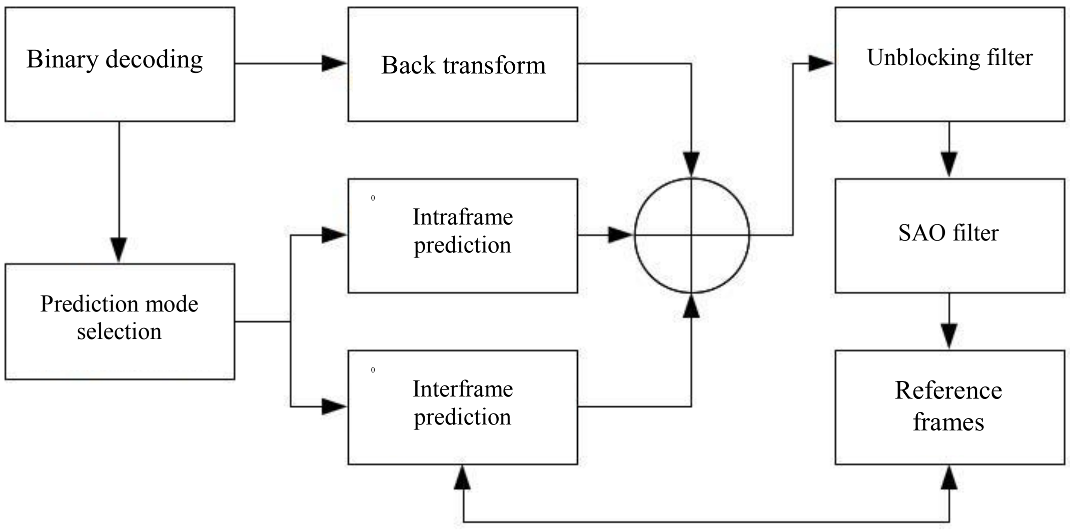

Fig. 2. HEVC codec flowchart

Here, the abbreviation SAO denotes the sample adaptive offset. In addition to the above image reconstruction algorithms, both codecs use context-adaptive binary arithmetic entropy coding, which is much more complicated than Huffman compression. With a high level of visual quality, it is arithmetic coding that occupies a significant part of the total decoder operation time. In general, the flowchart of the arithmetic coder is shown in Fig. 3.

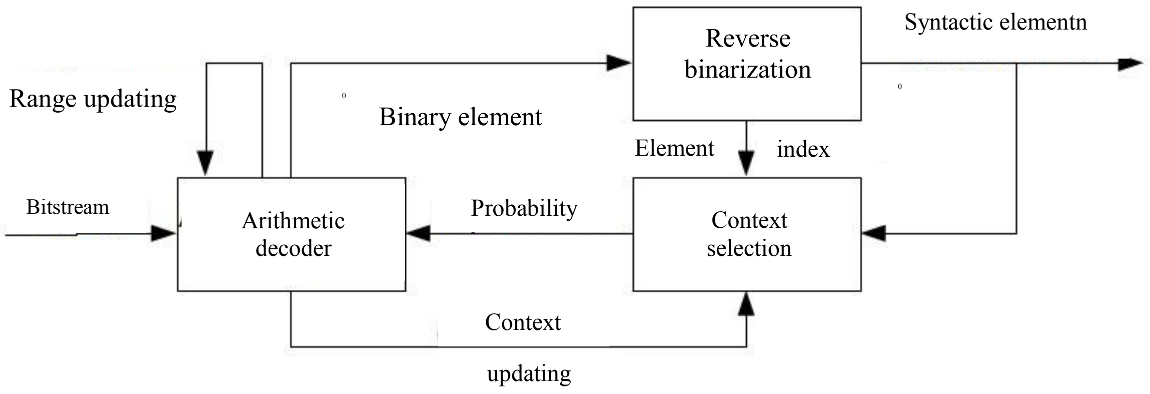

Fig. 3. Arithmetic coder flowchart

Codecs of a new generation are supposed to be used, among other things, for compressing images obtained through remote sensing of natural objects. In this case, it is necessary to optimize the operation of hybrid block codecs to ensure the processing of photo and video UAV data in real time. It is necessary to put more focus on the optimization of binary context-adaptive arithmetic coding, due to its essential complexity.

The objective of this study is to speed up the operation of the arithmetic decoder on mobile processors of the ARM architecture. This will improve the performance of Google VP9 video codecs and will allow for their application in the UAV on-board electronics for remote sensing of water bodies. The use of enhanced tools for compressing photos and videos obtained from aerial photography enables to increase the amount of stored data, improve their visual quality and resolution.

Main Part. Core components of a modern video codec include the binarization of syntactic elements of the bitstream and the adaptive binary encoding of these elements into the bitstream. This stage is not safe for the vectorization and parallelization, but it can be optimized through the statistical analysis of the input data. The main way to predict the running time of a program is to analyze the complexity of the corresponding algorithm. Distinction is provided between complexity at the best, worst and average case. $\sqsupset x$ is input data of A algorithm which is used to calculate the output of y algorithm. The time function of the algorithm is denoted by $C_A^T(x)$, and the memory cost function – by $C_A^S(x)$. In the worst case, we will call $T_A(n) = \max_{\|x\|=n} C_A^T(x)$ functions of numeric argument the time and space complexity A

$$S_A(n) = \max_{\|x\|=n} C_A^S(x).$$

Consider a finite set of n size inputs:

$$X_n = \{x: \|x\| = n\}.$$

$\forall x \in X_n$ corresponds to the probability:

$$P_n(x) \in [0,1]: \sum_{x \in X_n} P_n(x) = 1.$$

The following expectation is called the average-case complexity:

$$T_A = \sum_{x \in X_n} P_n(x) C_A^T(x),$$

$$S_A = \sum_{x \in X_n} P_n(x) C_A^S(x).$$

The described approach is classical for analyzing average-case complexity of the algorithm, and it is described in detail in [7, 8]. Note the reasons why the application of this method in practice may be difficult or impractical.

1. Difference between the number of steps of the algorithm in theory and the number of processor cycles required to perform the step in practice. Thus, the majority of modern central processors produce addition and multiplication per cycle, while the remainder of the division is calculated for dozens of cycles.

2. Hardware features of the memory system. Modern computers use a multi-level memory hierarchy. Its components operate at different speeds. Memory calls take considerably longer than register operations.

3. Optimizing compilers and hardware planners. When building executable files, optimizing compilers transform dramatically the code without changing the program state engine. CPU hardware schedulers change the procedure of executing instructions for greater performance and predict conditional transitions, while cache controllers read from memory in blocks.

4. In case of the software algorithms implementation on the general-purpose processors, software components interfere. For example, the task scheduler shares processor time, and parallel processes that have multiple threads can run on a variable number of processor cores.

The proposed modification of the algorithm complexity serves as a theoretical addition to practical tools for measuring performance, such as, for example, profiling and instrumentation of the program code. For the first time, the method of splitting the algorithm inputs into complexity classes was presented in [9]. Consider A algorithm and a set of all possible inputs:

$$G: \{g_1, g_2 \dots\},$$

and also all possible samples from G, different in size and composition:

$$g_i: \{g_i^1, g_i^2 \dots\}.$$

A set of criteria for the complexity measure of the algorithm implementation (for example, the number of processor cycles, run-time, etc.):

$$\propto_i: g_i \to \mathbb{R}.$$

A set of complexity measures:

$$A: \{\propto_1, \propto_2 \dots\}.$$

This set has the following properties.

1. $\forall \propto_1, \propto_2 \in A: \propto_1 \neq \propto_2$ — all A elements are different.
2. $\forall \propto_i \in A$ splits G into the set of equivalence complexity classes

$$G(\alpha_i) = \{g_i^1 \cap g_i^2 \dots\}.$$

3. All samples from $G(\alpha_i)$ are equally complex:

$$\exists\, \alpha_i \in A, g_i^k \in G(\alpha_i), \alpha_i: g_i^k \to r_i, k \in [0, \|G(\alpha_i)\|], r_i \in \mathbb{R}.$$

Thus, all elements of A are different, and they can be reordered so that the complexity function of the criterion is nondecreasing throughout the whole set of criteria. The expected complexity is similar to the estimates of the average-case complexity of the algorithm for the discrete and continuous probability of complexity. For the discrete case:

$$R(A) = \sum_{\alpha_i \in A} r_i p_i,$$

for the continuous case:

$$R(A) = \int_A r\, dF(r).$$

The method in question is applicable for analyzing the complexity of an arithmetic codec. The process of entropy compression [10, 11] can be divided into component parts.

1. Binarization or conversion of the coded character (syntactic element of the compressed bitstream) into a string made up of zeros and ones (bit string).

2. Context modeling for compressing syntactic elements in the normal mode. This step is not performed for syntactic elements whose statistical distribution is close to normal, and they are coded in the bypass mode.

3. Arithmetic coding of a bit string.

Consider in more detail the arithmetic decoding scheme of the Google VP9 codec, namely the part that is associated with the subexponential coding of syntactic elements.

Imagine an algorithm of the subexponential coding in general [12]. The first step includes the calculation of the variables:

$$b = \begin{cases} k: n < 2^k \\ \lfloor \log_2 n \rfloor: n \geq 2^k, \end{cases}$$

$$u = \begin{cases} 0: n < 2^k \\ b - k + 1: n \geq 2^k, \end{cases}$$

where k is parametrical value, it is 4 for the Google VP9 codec.

In the second step, the unary code $u(u+1)$ bit is complemented by n low-order bits. The code length is equal to:

$$u + 1 + n = \begin{cases} k + 1: n < 2^k \\ 2\lfloor \log_2 n \rfloor - k + 2: n \geq 2^k. \end{cases}$$

Hence, literal decoding is reduced to decoding the bits that compose it in the loop. To optimize the performance of this algorithm, it is important to know the probability distribution of literal lengths. Literals occupying the largest number of bits in a compressed bitstream (such as inverse transform coefficients and motion vectors) are coded in series, so there is a high probability that the distribution of literal lengths in a compressed bitstream will be constant with multiple repeats of elements of the same value. To test this hypothesis, experimental data on the distribution of literal lengths in a set of satellite images of the Sea of Azov is acquired (Table 1).

Table 1

Literal length, bit	1	2	3	4	5	6
Probability, %	0.94	0	67.35	18.25	0	13.46

The literals of 3, 4, and 6 bits in length are most probable. The maximum possible literal length for this sequence is just 6 bits. This fact is essential for software optimization of the function of subexponential decoding of a literal. As part of optimizing a real codec, the run-time criterion is of prime interest. To obtain the set of difficulties: $\{r_0, \ldots, r_4\}$, we will profile the program performance.

A set of unique elements (R) will make up a set of criteria of the run-time complexity (A).

The following approaches that are based on the data obtained are applied for the optimization.

1. Memoization of calculating the literal length to decode a series of literals of equal length.
2. Unwinding of a cycle of subexponential decoding of a literal.
3. More efficient algorithm for calculating the number of bits in a literal.
4. More efficient use of the processor registers immediately within the arithmetic decoding function.

The implementation of points 1 and 4 is fairly obvious; therefore, we consider in more detail points 2 and 3. Compiling of a decoded literal by bits decoded from a compressed bitstream occurs within the subexponential decoding function. In this case, the bottleneck is a loop with varying number of iterations. It can be replaced with a switch-case set without break at the end. This technique is known as the Duff's device method. It allows replacing several loop iterations through sequentially execution of the instructions without the need for conditional transitions. The bit shift amount is a constant that does not need to be read from the register - loop counter.

Code Listing 1: The original literal decoding function

```
static int vp9_read_literal(vp9_reader *br, int bits)
{
int z = 0, bit;
 for (bit = bits – 1; bit >= 0; bit –)
   z |= vp9_read_bit(br) << bit;
return z;
}
```

Code Listing 2: Modified literal decoding function

```
static int vp9_read_literal(vp9_reader *br, int bits) {
 register int z = 0;
 switch(bits -1){
   case 6: z |= vp9_read(br, 128) << 6;
   case 5: z |= vp9_read(br, 128) << 5;
   case 4: z |= vp9_read(br, 128) << 4;
```

```
  case 3: z |= vp9_read(br, 128) << 3;
  case 2: z |= vp9_read(br, 128) << 2;
  case 1: z |= vp9_read(br, 128) << 1;
  case 0: z |= vp9_read(br, 128);
break;
}
return z;
}
```

Another bottleneck is the calculation of the number of literal bits in the while loop [13]. This is a worse solution because the number of loop iterations is unpredictable. Instead, a fast bit-counting algorithm was used [14, 15], which performs the calculation for a fixed number of steps without conditional transitions.

Code Listing 3: Fast counting of the number of bits in a literal

```
Unsig ned intv; // 32-bit argument
Register unsig ned intr; // variable for the number of bits
register unsigned int shift;
r = (v > 0xFFFF) << 4;
v >>= r;
shift = (v > 0xFF) << 3;
v >>= shift;
r |= shift;
shift = (v > 0xF) << 2;
v >>= shift;
r |= shift;
shift = (v > 0x3) << 1;
v >>= shift;
r |= shift;
r |= (v >> 1);
```

Numerous starts of the reference and modified codecs were made to measure the runtime. In this case, their average run-time was compared. It has been found that the performance of the modified codec is 5.21% higher. The increase in overall performance for arithmetic decoding was 7.33%.

Conclusions. The operation of the arithmetic coder as a component of the video codec has been optimized using the example of the Google VP9 standard. To solve this problem, a modification of the method for analyzing the average-case complexity of algorithm has been proposed. The approach is based on the partitioning of the set of inputs into equivalence complexity classes. The considered method enables to predict the average-case complexity of the algorithm when the number of steps of the algorithm and the time of its execution depend on difficult-to-measure parameters, which is typical of the context-adaptive arithmetic coding. The proposed method has been applied to optimize the speed of the arithmetic binary coder (using the example of the Google VP9 codec) for the image compression problems obtained under remote sensing of water bodies. The research results make it possible to apply advanced methods of compressing photo and video data obtained through the aerial photography of water bodies. Thus, it is possible to increase an amount of the accumulated data, to improve the visual quality and resolution of the footage by 25–34% (according to the SSIM visual quality metric) and to increase the speed of the arithmetic coder by 7%.

References

1. WebP Compression Study. Google Developers. Available at: https://developers. google.com/ speed/webp/docs/webp_study (accessed 01.02.19).

2. Nguyenand, T., Marpe, D. Objective Performance Evaluation of the HEVC Main Still Picture Profile. *IEEE Transactions on Circuits and Systems for Video Technology*, 2015, vol. 25, no. 5, pp. 790–797.

3. Blahut, R. Bystrye algoritmy tsifrovoy obrabotki signalov. [Fast algorithms for signal processing.] Moscow: Mir, 1989, 448 p. (in Russian).

4. Wallace, G.K. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 1992, vol. 38, no. 1, pp. XVIII–XXXIV.

5. Dvorkovich, A.V., Dvorkovich, V.P. Tsifrovye videoinformatsionnye sistemy (teoriya i praktika). [Digital video information systems (theory and practice).] Moscow: Tekhnosfera, 2012, 1009 p. (in Russian).

6. Asaduzzaman, A., Suryanarayana, V. R., Rahman, M. Performance-power analysis of H.265/HEVC and H.264/AVC running on multicore cache systems. Intelligent Signal Processing and Communications Systems. Available at: https://ieeexplore.ieee.org/document/6704542 (accessed 01.02.19).

7. Sedgewick, R., Wayne, K. Algorithms. Fourth edition. Upper Saddle River: Addison-Wesley, 2016, 960 p.

8. Cormen, T.H., et al. Introduction to Algorithms. 3rd edition. Cambridge; London: The MIT Press, 2009, 1296 p.

9. Welch, W.J. Algorithmic complexity: three NP — hard problems in computation all statistics. Journal of Statistical Computation and Simulation, 1982, vol. 15, no. 1, pp. 17–25.

10. High efficiency video coding. Fraunhofer Heinrich Hertz Institute. Available at: http://hevc.info/ (accessed: 01.02.19).

11. Sze, V., Budagavi, M. Parallelization of CABAC transform coefficient coding for HEVC. Semantic Scholar. Allen Institute for Artificial Intelligence Logo. Available at: https://www.semanticscholar.org/paper/Parallelization-of-CABAC-transform-coefficient-for-Sze-Budagavi/0653a22ff7b82bdd0130cea8b597a7024ab46882 (accessed: 01.02.19).

12. Salomon, D., Motta, G. Handbook of data compression. London; Dordrecht; Heidelberg; New York: Springer-Verlag, 2010, 1360 p.

13. Anderson, S. E. Bit Twiddling Hacks. Available at: https://graphics.stanford.edu/~seander/bithacks.html (accessed 01.02.19).

14. Gervich, L.R., Shteinberg, B.Y., Yurushkin, M.V. Programmirovanie ekzaflopsnykh sistem. [Exaflops systems programming.] Open Systems. DBMS. 2013, vol. 8, pp. 26–29 (in Russian).

15. Warren, Jr., G.S. Algoritmicheskie tryuki dlya programmistov. [Algorithm programming tricks.] 2nd ed. Moscow: Williams, 2013, 512 p. (in Russian).

Received 02.11.2018
Submitted 03.11.2018
Scheduled in the issue 15.01.2019

Author:

Arzumanyan, Roman V.,
postgraduate of the Intelligent Multiprocessor Systems Department, Institute of Computer Technology and Information Security, Southern Federal University (22, ul. Chekhova, Taganrog, Rostov Region, 347922, RF),
ORCID: https://orcid.org/0000-0003-3370-5093
roman.arzum@gmail.com