Belarusian State Technological University
Department of Information Systems and Technology

Pavel Urbanovich

INFORMATION PROTECTION

Part 9: **DATA COMPRESSION**

pav.urb@yandex.by, p.urbanovich@belstu.by

Definition and Main Goals

- Data compression has a history that predates physical computing.
- Data compression can be considered as one of the classes of information encryption methods.
- **Morse code**, for example, compresses information by assigning shorter codes to characters that are statistically common in the English language (such as the letters “e” and “t”).

Data compression is a reduction in the number of bits (characters) needed to represent data.

Def. Data compression is the process of modifying, encoding or converting the bits structure of data in such a way that it consumes less space on disk.

- It enables reducing the storage size of one or more data instances or elements.

Data compression is also known as source coding or bit-rate reduction.

•Data compression enables sending a data object or file quickly over a network or the Internet and in optimizing physical storage resources.

•Data compression has wide implementation in computing services and solutions, specifically data (information) communications.

•Data compression has important application in the areas of **data (information) transmission** and **data (information) storage**.

Compressing data can be a lossless or lossy process.

Lossless compression enables the restoration of a file to its original state, without the loss of a single bit of data, when the file is uncompressed.

Lossless compression is the typical approach with executables, as well as text and spreadsheet files, where the loss of words or numbers would change the information.

Lossy compression permanently eliminates bits of data that are redundant, unimportant or imperceptible.

Lossy compression is useful with graphics, audio, video and images, where the removal of some data bits has little or no discernible effect on the representation of the content.

Fundamental Concepts

•A simple characterization of data compression is that it involves transforming a string of characters in some representation (such as **ASCII**) into a new string (of bits, for example) which contains the same information but whose length is as small as possible.

Example 1. The following string of characters is used to illustrate the concepts defined:

aa bbb cccc ddddd eeeeee fffffffgggggggg.

•A code is a mapping of source messages (words from the source alphabet ***alpha***) into codewords (words of the code alphabet ***beta***).

• The source messages are the basic units into which the string to be represented is partitioned.

•These basic units may be single symbols from the source alphabet, or they may be strings of symbols.

Example 2. For string, ***alpha*** = { a, b, c, d, e, f, g, *space*}. For purposes of explanation, ***beta*** will be taken to be { 0, 1 }.

source message	codeword
a	000
b	001
c	010
d	011
e	100
f	101
g	110
space	111

Fig.1

source message	codeword
aa	0
bbb	1
cccc	10
ddddd	11
eeeeee	100
fffffff	101
gggggggg	110
space	111

Fig.2

If the string ***EXAMPLE 1*** were coded using the Figure 1 code, the length of the coded message would be **120**; using Figure 2 the length would be **30**.

- The oldest and most widely used codes, ASCII, is example of codes, mapping an alphabet of 64 (or 256) single characters onto 6-bit (or 8-bit) codewords. It do not provide compression.

- When source messages of variable length are allowed, the question of how a message ensemble (sequence of messages) is parsed into individual messages arises.

- Many of the algorithms described here are **defined-word schemes**.

- A distinct code is uniquely decodable if every codeword is identifiable in a sequence of codewords.

- The codes of Figure 1 and Figure 2 are both distinct, but the code of Figure 2 is not **uniquely decodable**.

For example, the coded message **11** could be decoded as either **ddddd** or **bbbbbb**.

- A uniquely decodable code is a **prefix code** (or prefix-free code) if it has the prefix property, which requires that **no codeword is a proper prefix of any other codeword**.

Methods Classification

I. a) **without loss of information - *Lossless*** (text documents, program codes, databases),

   b) **with loss of information** (graphic, sound, audio - multimedia files),

II. a) **block** or **block-sorting** or **character-based** methods (examples: **Run-length encoding, RLE; Burrows-Wheeler transform, BWT**),

   b) **probabilistic** methods (examples: **Huffman** m., **Shannon-Fano** m.),

   c) **dictionary** methods (examples: **LZ**xx m.),

   d) **arithmetic** methods,

   e) **combined** methods.

Efficiency Evaluation

$R_1 = V_{ac}/V_{bc}$;

$R_2 = (V_{bc} - V_{ac})/V_{bc} = 1 - R_1$

V_{bc} - the file size before compression,

V_{ac} - the file size after compression;

ratio R_1 shows what volume part of the file before the compression takes the file after compression;

factor R_2 shows the degree of file compression, i.e. the ratio of the "exhausted" volume of the source file to this source file (before compression).

RLE Method

Run-length encoding (**RLE**) is a very simple form of lossless data compression

Idea: when you compress, a **string of identical characters** constituting a series **is replaced by a string that contains the repeating character itself and the number of its repetitions.**

Example. Black text (black pixel - B) on a white background (white pixel - W).

Input:
WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW → (67)

Output:
12W1B12W3B24W1B14W → (18)

BWT Method

M. Burrows and D. Wheeler proposed (1994) a new **lossless** compression algorithm.

It is based on a permutation of the input sequence - the Burrows-Wheeler Transformation (BWT), also called Block Sorting -, which groups symbols with a similar context close together.

In the original version, this permutation was followed by a move to front (MTF) transformation and a final entropy coding (EC) stage.

Later versions used different algorithms which come after the Burrows-Wheeler transform, since the stages after the Burrows-Wheeler transform have a significant influence on the compression rate too.

The most effective application of BWT-archivers for texts and any data with stable contexts.

- The heart of the algorithm is a reversible block sort which increases the compressibility of the input data.
- The Burrows Wheeler Transform is an algorithm that takes **a block of data and rearranges it using a sorting algorithm**. The resulting output block is extremely well-suited for compression.
- The resulting **output block contains exactly the same data elements as the input block, differing only in their ordering**.
- The **transformation is reversible**, meaning the original ordering of the data elements can be restored with no loss of fidelity.

BWT: Direct Transformation (compression)

The transformation is performed in **three steps**:

1. A **table** (**w1**: (n x n) **of all cyclic shifts** of the input string (**s_n**) of **n** symbols is compiled.

2. The **lexicographic (in alphabetical order) sorting** of the table rows is performed: **$w_1 \rightarrow w_2$**.

3. As the output string (**BWT(s_n)**), the **last column** (**n**) **of the conversion table (w_2) and the line number (#) that is the same as the original are selected**.

Example. Let s_n ="ABACABA“, n = 7.

s_n	w1	w2	$BWT(s_n)$, #
ABACABA	ABACABA BACABAA ACABAAB CABAABA ABAABAC BAABACA AABACAB	AABACAB ABAABAC ABACABA ACABAAB BAABACA BACABAA CABAABA	BCABAAA, 3

#

BWT: Inverse transformation

Input: **BWT(s_n),#**

The transformation is performed in **two steps** for **two operations in each** for the **construction of the matrix** w_2:

- inscribing **BWT(s_n)** into the free right-most column of the created matrix,
- sorting the resulting fragment of the created matrix:

inscribing	sorting	inscribing	sorting	inscribing	sorting	inscribing
B	A	BA	AA	BAA	AAB	BAAB
C	A	CA	AB	CAB	ABA	CABA
A	A	AA	AB	AAB	ABA	AABA
B	A	BA	AC	BAC	ACA	BACA
A	B	AB	BA	ABA	BAA	ABAA
A	B	AB	BA	ABA	BAC	ABAC
A	C	AC	CA	ACA	CAB	ACAB
Step1		Step2		Step3		Step4

sorting	inscribing	sorting	inscribing	sorting	inscribing	sorting
AABA	BAABA	AABAC	BAABAC	AABACA	BAABACA	AABACAB
ABAA	CABAA	ABAAB	CABAAB	ABAABA	CABAABA	ABAABAC
ABAC	AABAC	ABACA	AABACA	ABACAB	AABACAB	ABACABA
ACAB	BACAB	ACABA	BACABA	ACABAA	BACABAA	ACABAAB
BAAB	ABAAB	BAABA	ABAABA	BAABAC	ABAABAC	BAABACA
BACA	ABACA	BACAB	ABACAB	BACABA	ABACABA	BACABAA
CABA	ACABA	CABAA	ACABAA	CABAAB	ACABAAB	CABAABA

Step4 Step5 Step6 Step7

= 3

- The time complexity of this algorithm is ***O*(n³ log n)**.
- The spatial complexity is ***O*(n²)**.
- The sorting operations needed at the front end of the BWT will usually have ***O*(n log n)**.
- The BWT is apparently **not covered by any software patents**.
- It is used in the **bzip2** archiver.
- It is typically **used in conjunction with other archiver** (eg. **RLE**).
- The **main problem** in implementing BWT **is the choice of a fast data sort algorithm with a long n**.

Task: Input: 101010 Output: ?

Probabilistic Methods (Shannof-Fano, Huffman)

- **Idea:** In comparison with ASCII codes, having the same length, the binary codes in the **S-F** and **H** have different lengths: alphabet symbols with a higher probability of appearing in the texts correspond to codes of smaller length and vice versa.

- It is the **idea of constructing a "tree"**, the position of the symbol on the "branches" of which is determined by the frequency of its appearance. Each character is assigned a code whose length is inversely proportional to the frequency of the occurrence of this symbol.

- The **Shannon-Fano** and **Huffman** algorithm yields a **prefix code**.

There are two types of probabilistic methods that distinguish by the way of determining the probability of occurrence of each symbol:

- **static** methods using a fixed symbol frequency table, calculated before the beginning of the compression process; static methods are characterized by good speed and do not require significant memory resources; they have found wide application in numerous archiver programs, for example **ARC**, **PKZIP**,

- **dynamic** or adaptive methods, in which the frequency of the appearance of symbols is constantly changing and as the new data block is read, the initial values of the frequencies are recalculated.

Shannon-Fano Method

Shannon-Fano method (coding), named after Claude Shannon and Robert Fano.

In Shannon–Fano coding, the symbols are arranged in order from most probable to least probable, and then divided into two sets whose total probabilities are as close as possible to being equal.

All symbols then have the first digits of their codes assigned; symbols in the first set receive "0" and symbols in the second set receive "1".

As long as any sets with more than one member remain, the same process is repeated on those sets, to determine successive digits of their codes.

When a set has been reduced to one symbol this means the symbol's code is complete and will not form the prefix of any other symbol's code.

In this way:

the code with **Shannon-Fano method** is constructed as follows:

- the source messages or symbols ***a(i)*** and their probabilities ***p(a(i))*** are listed in order of no increasing probability,

- this list is then **divided in such a way as to form two groups of as nearly equal total probabilities as possible**,

- each symbol in the first group **receives 0** as the first digit of its codeword; the symbols in the second half have code words **beginning with 1**,

- each of these groups is then divided according to the same criterion and additional code digits are appended,

- the process is continued until **each subset contains only one message**.

Example. Let we have the probability distribution:

p(*a*) =1/2, p(*b*)= 1/4, p(*c*)= 1/8, p(*d*)= 1/16, p(*e*)= 1/32, p(*f*)= 1/32

$$\Sigma p_i (') = 1$$

Using the algorithm described above, we obtain a code table:

a	1/2	0
b	1/4	10
c	1/8	110
d	1/16	1110
e	1/32	11110
f	1/32	11111

When compressed, each document symbol should be replaced by the corresponding binary code, and vice versa.

Example. Let the probability distribution is formed on the basis of the following message (see slide 3):
aa bbb cccc ddddd eeeeee fffffffgggggggg

A Shannon-Fano Codes:

g	8/40	00
f	7/40	010
e	6/40	011
d	5/40	100
space	5/40	101
c	4/40	110
b	3/40	1110
a	2/40	1111

It is clear, that the compression of the message consists in replacing the letters with the corresponding codes.

The length of the compressed message is **117** bit.

Huffman Method

- Huffman coding came about as the result of a class project at MIT by its student, David Huffman.
- In 1951, **D.Huffman** was taking a class under **Robert Fano**, (who invented an efficiency scheme known as Shannon-Fano coding).
- When Fano gave his class the opportunity to either write a term paper or take a final exam, **Huffman chose the term paper, which sought to find an efficient binary coding method**.
- This resulted in Huffman coding, which by the 1970s had become a prominent digital encoding algorithm.
- Huffman coding is a **lossless** data encoding algorithm.

Algorithm of a Binary Tree Creation

- The process behind its scheme includes sorting numerical values from a set in order of their frequency.
- The least frequent numbers are gradually eliminated via the **Huffman tree**, which adds the two lowest frequencies from the sorted list in every new "branch".
- The sum is then positioned above the two eliminated lower frequency values, and replaces them in the **new sorted list**.
- Each time a new branch is created, it moves the general direction of the tree either to the right (for higher values) or the left (for lower values).
- When the sorted list is exhausted and the tree is complete, the final value is **zero** if the tree ended on a left number, or it is **one** if it ended on the right.

Example. Let we have the probability distribution:

a_i	1	2	3	4	5	6	7
$P(a_i)$	0,4	0,2	0,1	0,1	0,1	0,05	0,05

The input data is written in a column, the last two (least) probabilities are added together,
and the resulting sum becomes a new element of the table that occupies the corresponding place in the list of decreasing probabilities.
This procedure continues until there are only two elements left in the column.

1	0,4	0,4	0,4	0,4	0,4	0,6
2	0,2	0,2	0,2	0,2	0,4	0,4
3	0,1	0,1	0,2	0,2	0,2	
4	0,1	0,1	0,1	0,2		
5	0,1	0,1	0,1			
6	0,05	0,1				
7	0,05					

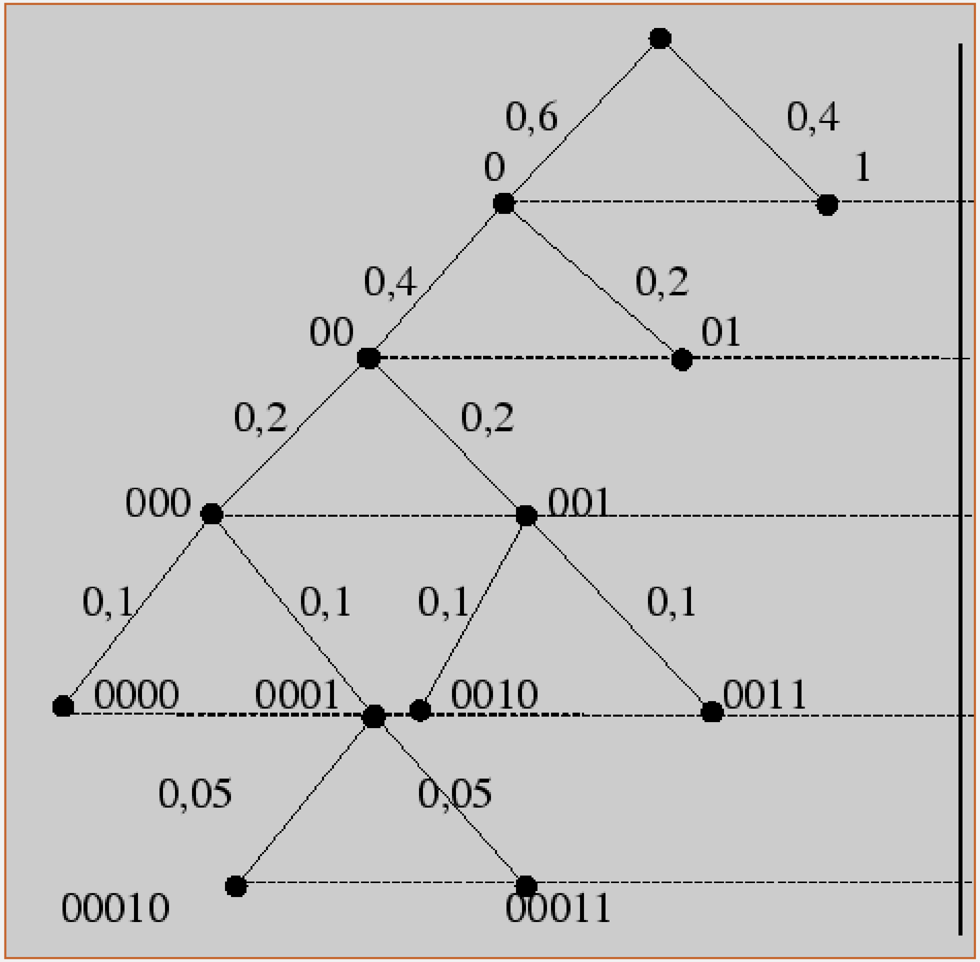
The second step is coding, "passing" the table (tree) from right (root) to left:

1	0,4	0,4	0,4	0,4	0,4	0,6 0
2	0,2	0,2	0,2	0,2	0,4 00	0,4 1
3	0,1	0,1	0,2	0,2 000	0,2 01	
4	0,1	0,1	0,1 0010	0,2 001		
5	0,1	0,1 0000	0,1 0011			
6	0,05 00010	0,1 0001				
7	0,05 00011					

As you can see, the minimum code length ($\mathbf{L}_{min}$) is 1 bit.

The construction of the code tree begins from the root.

- Two outgoing edges are assigned as weights the probabilities 0.6 and 0.4 in the last column. The code symbols **0** and **1** are assigned to the tree vertices thus formed.
- Then we "go" along the table from right to left. Since the probability of 0.6 is the result of the addition of two probabilities of 0.4 and 0.2, two edges with weights of 0.4 and 0.2, respectively, emanate from vertex **0**, which leads to the formation of two new vertices with code symbols **00** and **01**.
- The procedure continues as long as there are probabilities in the table that result from the summation.
- The construction of the code tree ends with the formation of seven leaves corresponding to these symbols with the codes assigned to them.
- The tree obtained as a result of Huffman coding has the following form (see next slide).



Codes	Symbols
1	1
01	2
0010 0011 0000	3 4 5
00010 00011	6 7

Table of codes

symbol	**1**	**2**	**3**	**4**	**5**	**6**	**7**
code	**1**	**01**	**0010**	**0011**	**0000**	**00010**	**00011**

It is understood that all code combinations must not contain prefixes.

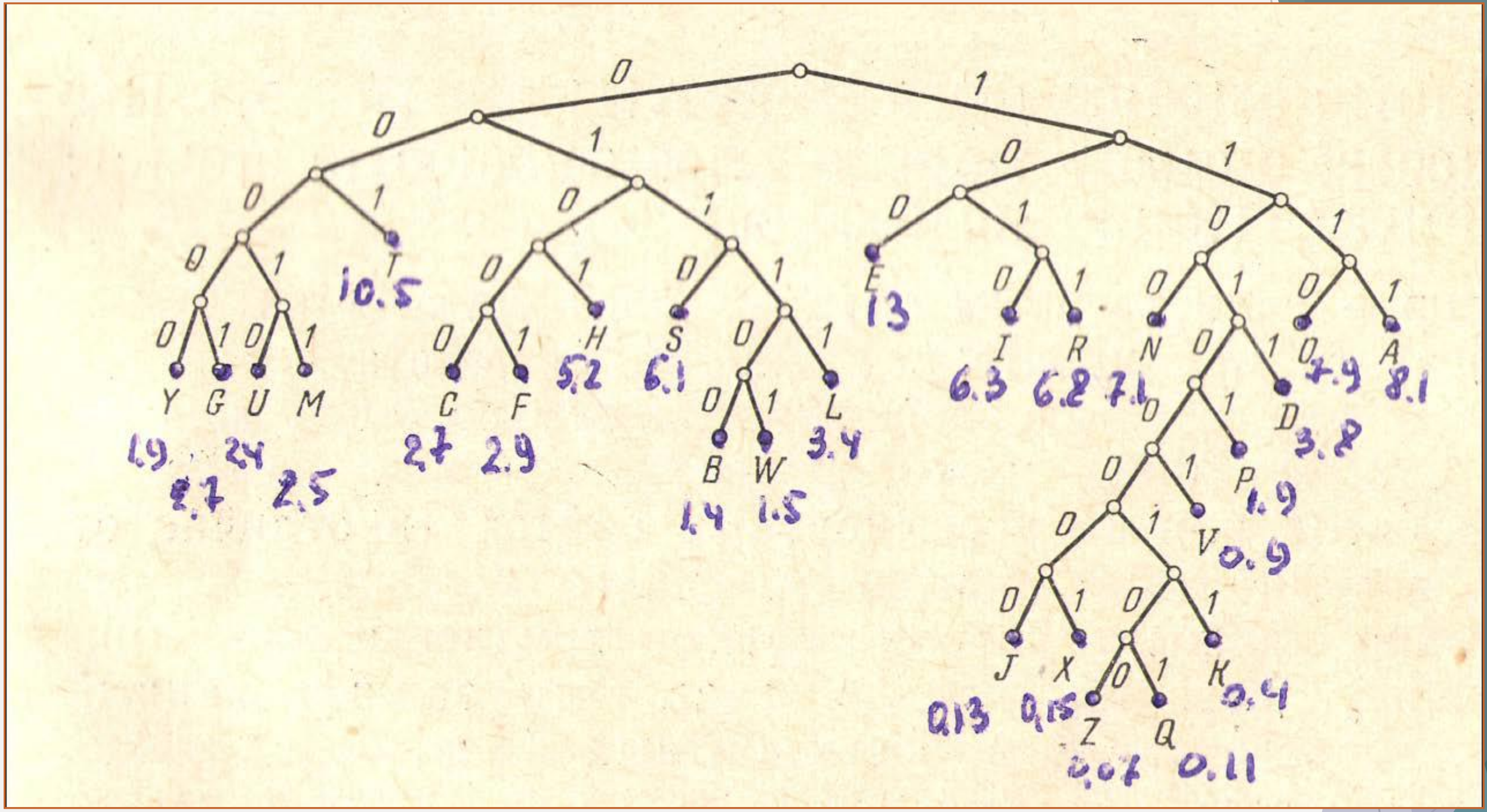
If the table contains character codes of some alphabet, then formally the procedure of message compression based on the symbols of this alphabet consists in replacing each message symbol with the appropriate code.

- As you can see, both in the **S-F** method and in the **H** method can be implemented different code tables for the same probability distribution.
- The smallest length of the compressed message is provided by the table that is characterized by the smallest coefficient:

$$C_i = \Sigma p(a_i) L_{i,}$$

L_i **-** code length in bits;
The encoding price (the average length of the code word) **C** is the criterion for the degree of **coding optimality**.
C corresponds to the average number of bits per alphabet character (message).

Let us calculate it in our case:
C = 1 * 0.4 + 2 * 0.2 + 4 * (0.1 * 0.3) + 5 * (0.05 * 2) = 2.5 bits



Huffman's binary tree for the English alphabet

E	100	M	00011
T	001	U	00010
A	1111	G	00001
O	1110	Y	00000
N	1100	P	110101
R	1011	W	011101
I	1010	B	011100
S	0110	V	1101001
H	0101	K	110100011
D	11011	X	110100001
L	01111	J	110100000
F	01001	Q	1101000101
C	01000	Z	1101000100

Code table based on Huffman's binary tree for the English alphabet
(Source: L.J. Hoffman, Modern methods for computer security and privacy, Prentice-Hall, 1977)

As you can see, the minimum code length ($\mathbf{L}_{min}$) is 3 bit.

Inverse transformation (decompression)

Both for the **S-F** method and for the **H** method the inverse transformation is carried out according to the identical algorithm:

1. The initial $\mathbf{L_{min}}$ bits of the sequence are initiated for analysis: L:=$\mathbf{L_{min}}$.
2. The analysis by comparing them with the codes in the table is performed.
   If a accordance is found, the corresponding symbol ($\mathbf{a_i}$) of the alphabet is formed on the output of the decompressor and the move to the analysis of the next $\mathbf{L_{min}}$ bits is carried out. If there is no - move to step 3.
3. L :=**L+1**. Move to step 2.

References:

1. M. Burrows and D.J. Wheeler. A Block-sorting Lossless Data Compression Algorithm, *Digital Systems Research Center Research Report 124*, 1994, URL:http://gatekeeper.dec.com/pub/DEC/SRC/researchreports/abstracts/src-rr-124.html
2. Data compression, [Electronic Resource], URL: https://searchstorage.techtarget.com/definition/compression
3. Source Coding for Compression, [Electronic Resource], URL: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-661-receivers-antennas-and-signals-spring-2003/lecture-notes/lecture16.pdf
4. Huffman D. A Method for the Construction of Minimum-Redundancy Codes, Proceedings of the IRE, (1952),no. 40 (9),p. 1098–1101
5. Коды Фано и Хаффмана, [Electronic Resource], URL: http://cito-web.yspu.org/link1/metod/theory/node36.html
6. Урбанович,П.П. Лабораторный практикум по дисциплинам «Защита информации и надежность информационных систем» и «Криптографические методы защиты информации». Ч.1: Кодирование информации: учебно-метод.пос./П.П. Урбанович, Д.В.Шиман, Н.П. Шутько. – Минск: БГТУ,2019. – 95 с.
7. Урбанович, П. П. Защита информации и надежность информационных систем : пос. для студ. вузов спец. 1-40 05 01-03 «Информационные системы и технологии (издательско-полиграфический комплекс)» / П. П. Урбанович, Д. В. Шиман.- Минск : БГТУ, 2014. - 91 с. (URL: https://elib.belstu.by/handle/123456789/23761)
8. Сжатие и архивирование данных : учебно-методическое пособие для аспирантов / [сост.: Н. В. Пацей, П. П. Урбанович]. - Минск : БГТУ, 2004. - 32 с. (URL: https://elib.belstu.by/handle/123456789/25876)