**Lab work #3**

**Topic: Encoding information using the Run method Length Encoding .**

**The purpose of the work** : to study the method of encoding the lengths of the Run series Length Encoding .

**Theoretical Part**

**RLE Algorithm**

**Run Length Encoding (RLE)** is a data encoding method designed to compress data containing a series of repeating elements. This method is especially effective in cases where the same character or color pixel is repeated many times in a row, which is often the case in text files and images with large areas of uniform color.

**The principle of RLE** is to replace a sequence of repeating characters with a pair consisting of the number of repetitions and the character itself. For example, the string "AAAABBBCCDAAA" can be converted to "4A3B2C1D3A". This allows for a significant reduction in the amount of data when there are long sequences of identical characters.

**Advantages and disadvantages of RLE** :

1. **Advantages** :
   * Ease of implementation.
   * Efficient compression of data with large sequences of identical elements.
   * No loss of information during data compression and recovery.
2. **Flaws** :
   * Low efficiency when encoding data with frequent character changes.
   * Possible increase in data volume when compressing files that do not contain long runs of identical characters.

**Application of RLE**

Although RLE is not an optimal choice for all data types, it can be successfully applied in specialized areas such as archiving black and white images, documents with limited character sets, and other media where repetitions of the same element occur frequently.

In this lab, students will implement the RLE algorithm, examine its effectiveness on various types of data, and evaluate the advantages and disadvantages of the method in practice.

When implementing this algorithm, two options are possible:

**Encoding with ESCAPE character:**

Let us have some symbol, which we will call the ESCAPE symbol. As this symbol we can take such a symbol, which does not occur in our data, or, if there are none, then any other. Further we will denote the ESCAPE symbol as #.

Coding rules:

In this table ( n ) means that we consider that this byte does not store a symbol, but a number (we can consider that this is a symbol with code n ).

|  |  |  |
| --- | --- | --- |
| Rule number | Description | Example |
| 1 | n consecutive identical letters in the form of 3 symbols, #, the number of symbols and the symbol itself. The rule can be applied starting with n = 4 . | AA …. A −> # nA |
| 2 | Less than four consecutive characters, with the exception of #, are encoded as they are, since there is no benefit in encoding them differently. | A −> A  AA −> AA  AAA −> AAA |
| 3 | One or more # are always encoded with three symbols | # −> #1#  ## −> #2#  ### −> #3# |
| 4 | n consecutive different symbols are not encoded. | ABC –> ABC |

For example:

AAAAABCDCDBBBB −> #5 ABCDCD #4 B

The decoding process is simple: we process the characters of the encoded text in a row, if the current character is not #, then we rewrite it as is, if it is # − then we always look at the next 2 characters, from one of them we extract the character counter, from the second the character that we need number of times to repeat.

Accordingly, rule 3 follows, because otherwise, if we were to encode # as it is, i.e. #→#, then, when decoding, the next symbol would be perceived as the number of repetitions.

**JUMP Method**

In the traditional RLE algorithm ( Run Length Encoding ), which is used to encode sequences of repeating symbols, there may be situations where it is necessary to efficiently handle transitions between series of identical and different symbols. Here the JUMP method comes to the rescue, which allows you to optimize this process, improving the overall efficiency of encoding in the presence of variable sequences in the data.

The JUMP method is especially useful in scenarios where the data contains frequent and short runs of different symbols between long blocks of repetitions. This method avoids the excessive increase in data size that would occur if each such run were encoded separately using the traditional RLE rule. Using a special feature in the quantity byte allows the decoder to quickly determine which rule was applied and correctly reconstruct the original data.

|  |  |  |
| --- | --- | --- |
| Rule number | Description | Example |
| 1 | n consecutive identical letters in the form of 2 symbols, the number of symbols and the symbol itself. The rule can be applied starting with n=2. | AA …. A −> nA |
| 2 | n consecutive different symbols are coded as the number of symbols and the symbols themselves. Only in this case, the byte in which the number of symbols is stored also stores the sign that rule 2 is used. | ABCDAC … −> n ' ABCDAC |

Encoding the counter byte: for example, you can select one of the bits in this byte and write 0 to it if we use rule 1 and, accordingly, 1 if we use rule 2. The remaining 7 bits are left to store the number of characters.

Example:

Here the counter byte is specified in the form ( x , y ) where x is the rule flag, y is the fragment length.

AAAAABCDCDBBBB −> 5 A 133 BCDCD 4 B

5 consecutive A symbols are encoded as binary number 00000101

The 5 different BCDCD symbols are also encoded by the number 00000101, in which the most significant bit is then set to 1 – 10000101 , which corresponds to the decimal number 133.

When decoding, the most significant bit of the read symbol is checked. If it is 0, the next symbol is repeated the corresponding number of times. If the bit is 1, it is cleared. The resulting number indicates the number of symbols that follow it. These symbols are read and written to the output stream.

**Lab assignment:**

Write a program that implements *the escape* or *jump* methods of encoding information **( *escape is* option 1, *jump is* option 2)** .

The program must accept text either from the input window on the main form or read from a file as input information. Calculate and output the compression ratio.