A logo of a university

Description automatically generated

International University of Information Technology

Department of Computer Engineering

**Laborotoy Work №3**

Complete by a student of the group: Ospan Ramazan it2-2310

Checked by senior lector of the Department of Computer Engineering:

Daurenbaeva Nurkamilya

Almaty 2024

**Run Length Encoding (RLE) Method:**

Run Length Encoding (RLE) is a data compression technique designed for the efficient storage or transmission of data containing long sequences of identical elements. This method replaces sequences of repeating characters with a pair consisting of the number of repetitions and the character itself.

**Principle of Operation:**

RLE works by replacing sequences of repeating characters with their compressed representation. For example, the string "AAAABBBCCDAAA" is transformed into "4A3B2C1D3A," where each sequence of repeating characters is replaced by a number and the character.

**Encoding Examples:**

The string "AAAAABCDCDBBBB" is encoded as "#5ABCDCDB#4B". Here:

* "AAAAA" is replaced with "#5A"
* "B" is not encoded, as it is less than 4 repetitions
* "C" is not encoded
* "D" is not encoded
* "BBBB" is replaced with "#4B"

**Advantages of RLE:**

1. Simplicity of implementation.
2. Effective compression of data containing long sequences of identical elements.
3. No loss of information during compression and decompression.

**Disadvantages of RLE:**

1. Low efficiency when encoding data with frequent changes in characters.
2. Potential increase in data size when encoding files that do not contain long sequences of repeating characters.

**Algorithm**

A screen shot of a computer

Description automatically generated

def escapeEncode(text):

    result = []  # Хранение результата

    index = 0

    while index < len(text):

        count = 1

        # Подсчёт количества повторяющихся символов

        while index + 1 < len(text) and text[index] == text[index + 1]:

            count += 1

            index += 1

        # Если символ '#', кодируем с #1# #2# и т.д

        if text[index] == '#':

            result.append(f"#1#{count}")

        elif count >= 4:

            result.append(f"#{count}{text[index]}")

        else:

            result.append(text[index] \* count)  # Если повторов меньше 4, оставляем как есть

        index += 1

    return ''.join(result)

def compressRatio(original, encoded):

    # Проверка на случай пустого текста

    if not encoded:

        return 0

    # Возвращаем отношение длины исходного текста к длине сжатого

    return len(original) / len(encoded)

# Ввод текста и кодирование

inputText = input("Ввод текста: ")

encodedText = escapeEncode(inputText)

# Вывод закодированного текста и коэффициента сжатия

print("Кодировка:", encodedText)

print(f"Коэффициент сжатия: {compressRatio(inputText, encodedText):.2f}")

A black background with green and yellow text

Description automatically generated

* result is used to store the encoded parts of the text..
* index is used to iterate over the characters in the text.

A screen shot of a computer

Description automatically generated

This block counts the number of repeating characters starting from the current index. count is incremented if the next character is the same, and index moves to the next character.

A computer screen shot of text

Description automatically generated

* If the current character is #, it is encoded as #1#, where count indicates the number of occurrences of the # character.
* If the number of repeating characters is 4 or more, it is encoded as #countSymbol.
* If the number of repetitions is less than 4, the characters are added to result unchanged.

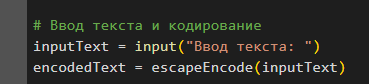
A black background with white text

Description automatically generated

Move to the next character and convert the result list to a string and return it



Calculate the ratio of the length of the original text to the length of the encoded text



The user inputs text, which is then encoded using the escapeEncode function.