A logo of a university

Description automatically generated

International University of Information Technology

Department of Computer Engineering

**Laborotoy Work №4**

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

**1. LZW Algorithm**

The **LZW (Lempel-Ziv-Welch)** algorithm is a data compression method that builds a dictionary of repeating strings of characters. The algorithm allows data to be encoded without analyzing the entire text first, which makes it efficient for compressing data in a streaming process.

#### 2. ****How the LZW Algorithm Works****

* **Initialization**: The dictionary starts with all possible single characters (for example, ASCII characters).
* **Reading the Text**: The algorithm reads the characters one by one and forms strings from them.
  + If the string is already in the dictionary, it is extended with the next character.
  + If the string is not in the dictionary, the algorithm adds it and assigns a unique code to it.
* **Compression**: Instead of storing or sending the original characters, LZW stores or sends the codes for the strings, which reduces the size of the data.

#### 3. ****LZW Decompression****

To decompress, we don’t need to store the dictionary in the compressed file. The decoding process is similar to compression and can rebuild the dictionary from the codes received. This allows the data to be compressed and then restored without any loss.

#### 4. ****Advantages of LZW****

* **Efficiency**: LZW works well when there are many repeating strings.
* **Versatility**: The algorithm can compress different types of data, such as text, images, and more.
* **Streaming**: Compression happens while the data is being read, making the algorithm useful for online applications.

#### 5. ****Example of LZW****

For example, if we want to compress the string "ABABABA", the algorithm processes the characters, adds new strings to the dictionary, and replaces them with codes:

* The string "AB" is added to the dictionary when it appears the first time, and its repetitions are replaced by a code, reducing the size of the data.

#### 

#### 6. ****Compression Ratio****

The compression ratio shows how much smaller the data becomes after compression. It is calculated by comparing the original size of the data to the size of the compressed file.

**Algorithm**

# Функция для сжатия данных (кодирования)

def lzwCompress(thisUncompressed):

    # Инициализация словаря (ASCII символы)

    thisDictionary = {chr(i): i for i in range(256)}

    thisDictSize = 256

    thisString = ""

    thisCompressed = []

    for thisChar in thisUncompressed:

        thisCombined = thisString + thisChar

        if thisCombined in thisDictionary:

            thisString = thisCombined

        else:

            thisCompressed.append(thisDictionary[thisString])

            # Добавляем новую строку в словарь

            thisDictionary[thisCombined] = thisDictSize

            thisDictSize += 1

            thisString = thisChar

    # Выводим код для последней строки

    if thisString:

        thisCompressed.append(thisDictionary[thisString])

    return thisCompressed

# Функция для распаковки данных (декодирования)

def lzwDecompress(thisCompressed):

    # Инициализация словаря (ASCII символы)

    thisDictionary = {i: chr(i) for i in range(256)}

    thisDictSize = 256

    thisString = chr(thisCompressed.pop(0))

    thisDecompressed = [thisString]

    for thisCode in thisCompressed:

        if thisCode in thisDictionary:

            thisEntry = thisDictionary[thisCode]

        elif thisCode == thisDictSize:

            thisEntry = thisString + thisString[0]

        else:

            raise ValueError("Некорректный код: %s" % thisCode)

        thisDecompressed.append(thisEntry)

        # Добавляем новую строку в словарь

        thisDictionary[thisDictSize] = thisString + thisEntry[0]

        thisDictSize += 1

        thisString = thisEntry

    return ''.join(thisDecompressed)

# Функция для расчета коэффициента сжатия

def calculateCompressionRatio(thisOriginal, thisCompressed):

    thisOriginalSize = len(thisOriginal) \* 8  # Оригинальный размер в битах

    thisCompressedSize = sum(len(bin(thisCode)[2:]) for thisCode in thisCompressed)  # Размер сжатого файла в битах

    return thisOriginalSize / thisCompressedSize

# Основная функция для ввода данных и работы с файлами

def main():

    # Выбор способа ввода: с экрана или из файла

    thisChoice = input("Введите 1 для ввода текста вручную, 2 для чтения из файла: ")

    if thisChoice == "1":

        thisText = input("Введите текст для сжатия: ")

    elif thisChoice == "2":

        thisFileName = input("Введите имя файла: ")

        try:

            with open(thisFileName, 'r') as thisFile:

                thisText = thisFile.read()

        except FileNotFoundError:

            print("Файл не найден.")

            return

    else:

        print("Неверный выбор.")

        return

    # Сжатие текста

    thisCompressed = lzwCompress(thisText)

    print(f"Сжатый текст: {thisCompressed}")

    # Распаковка текста

    thisDecompressed = lzwDecompress(thisCompressed)

    print(f"Распакованный текст: {thisDecompressed}")

    # Проверка, что распаковка даёт исходный текст

    if thisText == thisDecompressed:

        print("Распаковка успешна, текст совпадает.")

    else:

        print("Ошибка: распакованный текст не совпадает с исходным.")

    # Расчет коэффициента сжатия

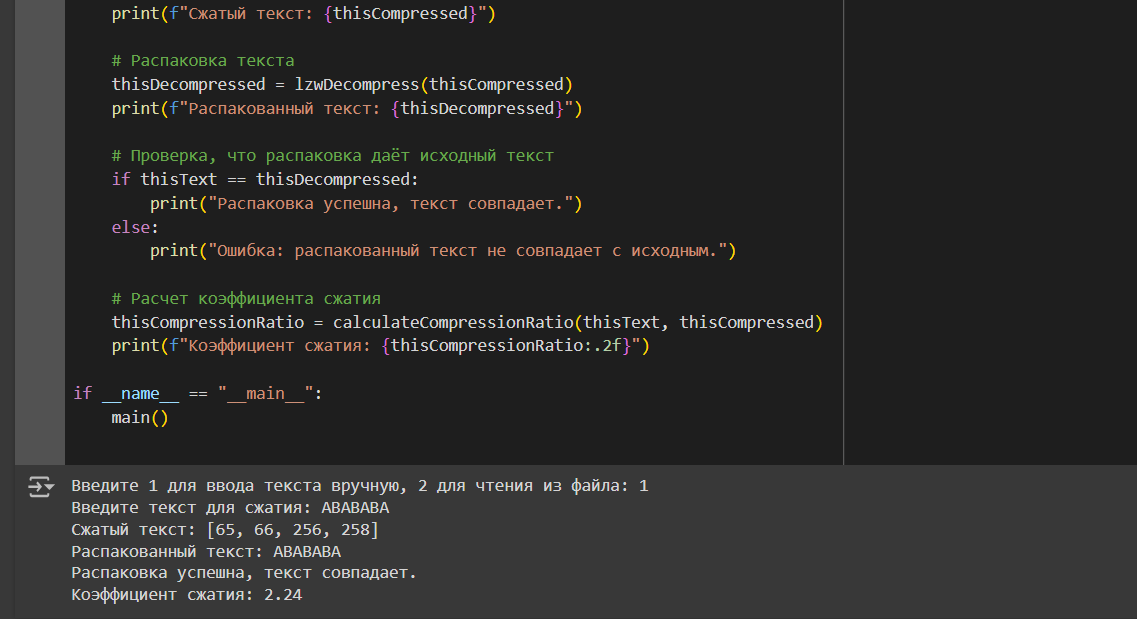
    thisCompressionRatio = calculateCompressionRatio(thisText, thisCompressed)

    print(f"Коэффициент сжатия: {thisCompressionRatio:.2f}")

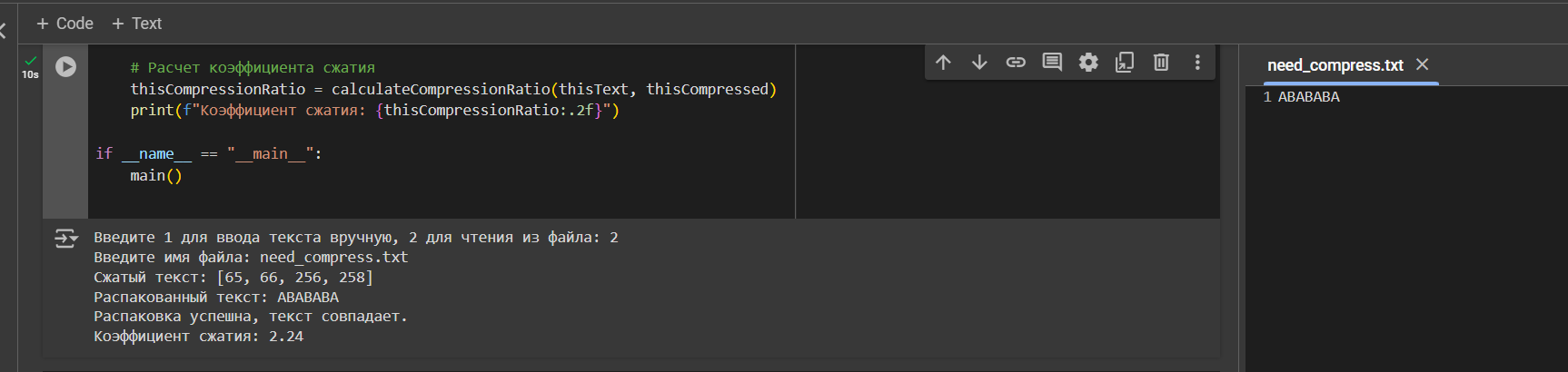
if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Test:**

****

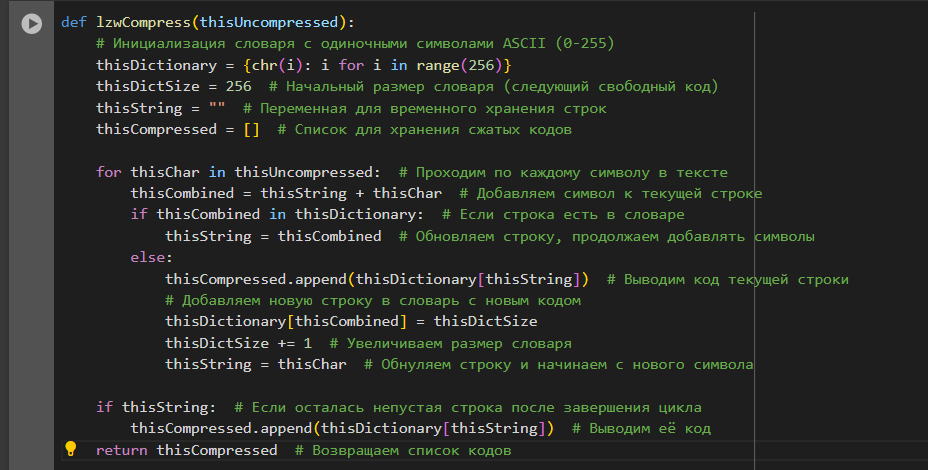
Test with input from keyboard



With import .txt file from directory

**Explanation:**

Compression (LZW Compression)

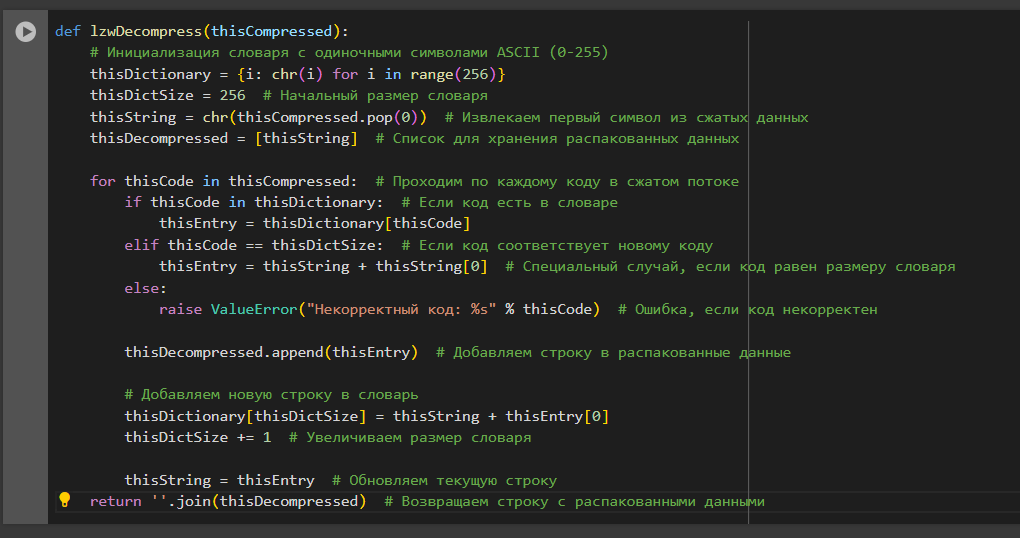
****

1. **Dictionary Initialization**: The dictionary is filled with all possible single characters (ASCII). Each character gets its own code.
2. **Main Loop**: The algorithm combines the current character with previous ones into strings.

* If this string is already in the dictionary, it keeps adding the next character.
* If the string is not in the dictionary, the code for the current string is output, the new string is added to the dictionary, and the process repeats with the next character.

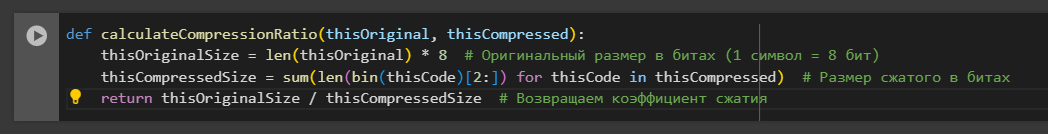
1. **End of Process**: If there is a string left in the buffer thisString, its code is added to the compressed data list.

**Decompression (LZW Decompression)**

****

* 1. **Dictionary Initialization**: The dictionary is initialized with single characters (ASCII).
  2. **First Code Processing**: The first code is taken, and it is directly converted to a character. This character becomes the starting string.
  3. **Main Loop**: For each code in the compressed data:
* If the code is in the dictionary, the corresponding string is found.
* If the code equals the current size of the dictionary, a new string is created (this can happen when adding new strings during decompression).
  1. **Updating the Dictionary**: Each time, a new string is added to the dictionary based on the current and previous strings.
  2. **End of Process**: All the decompressed data is joined into a final string.

**Additional Functions**

****

1. **Data Input**: The user chooses whether to input text manually or read it from a file.
2. **Compression and Decompression**: The text is compressed, then decompressed to check if it matches the original text.
3. **Checking the Result**: The program checks if the decompressed text matches the original.
4. **Output**: The program prints the compressed data, decompressed data, and the compression ratio.

#### In concludion

The LZW algorithm is a powerful tool for lossless data compression, used in file formats like GIF and TIFF. Its ability to dynamically build a dictionary while compressing makes it efficient and flexible for many applications.