A logo of a university

Description automatically generated

International University of Information Technology

Department of Computer Engineering

**Laborotoy Work №7**

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

**Objective of the Work**

The purpose of this laboratory work is to explore the operating principles of the Shannon-Fano algorithm, a method used for lossless data compression. The lab aims to develop a comprehensive understanding of encoding and decoding messages based on symbol probability and to implement the Shannon-Fano algorithm programmatically to encode a given string.

**Theoretical Background**

**Shannon-Fano Algorithm Overview:** The Shannon-Fano algorithm is an entropy coding technique devised for efficient data compression. The algorithm categorizes symbols based on their frequency of occurrence and assigns shorter binary codes to more frequent symbols, while less frequent symbols receive longer codes. This method reduces the total amount of data required for storing or transmitting information, leveraging the probabilities of symbol occurrence.

**Algorithm Principles:**

1. **Probability Calculation:** Calculate the occurrence probability for each symbol in the message.
2. **Sorting Symbols:** Arrange the symbols in descending order based on their probabilities.
3. **Division into Groups:** Split the sorted list of symbols into two groups such that the sum of probabilities in each group is as close to equal as possible.
4. **Binary Code Assignment:** Assign '0' to the left group and '1' to the right group.
5. **Recursive Division:** Continue this division process recursively for each group until every symbol is assigned a unique binary code.

**Practical Part**

**Objectives:**

1. Generate Shannon-Fano codes for the symbols of the input message.
2. Implement the algorithm in a programming language (e.g., Python) to encode a specific string.
3. Evaluate the compression efficiency based on the lengths of the generated binary codes compared to the original data size.

**Steps to Implementation:**

1. **Determination of Symbol Frequencies:**
   * Input the string for encoding.
   * Count the frequency of each character in the string.
   * Calculate the probability of occurrence for each symbol (probability = symbol frequency / total number of symbols).
2. **Construction of Shannon-Fano Codes:**
   * Sort the symbols by their probabilities in descending order.
   * Divide the symbols into two groups with approximately equal total probabilities.
   * Assign binary codes ('0' for the first group and '1' for the second).
   * Repeat the division process recursively for each group until all symbols have unique codes.

Python Algorithm:

# Функция для сортировки символов по их частоте, с учетом порядка появления символов

def sort\_by\_frequency(frequencies, text):

    # Добавляем порядок символов, чтобы при одинаковой частоте они оставались в исходном порядке

    symbols\_with\_order = [(char, freq, text.index(char)) for char, freq in frequencies.items()]

    return sorted(symbols\_with\_order, key=lambda x: (-x[1], x[2]))

# Рекурсивная функция для построения кодов Шеннона-Фаннона

def shannon\_fano\_code(symbols, code\_dict, prefix=""):

    if len(symbols) == 1:

        code\_dict[symbols[0][0]] = prefix

        return

    # Находим точку разделения

    total = sum([symbol[1] for symbol in symbols])

    running\_total = 0

    split\_index = 0

    for i, symbol in enumerate(symbols):

        running\_total += symbol[1]

        if running\_total >= total / 2:

            split\_index = i + 1

            break

    # Рекурсивно обрабатываем левую и правую части

    shannon\_fano\_code(symbols[:split\_index], code\_dict, prefix + "0")

    shannon\_fano\_code(symbols[split\_index:], code\_dict, prefix + "1")

# Функция для подсчета частот символов

def calculate\_frequencies(text):

    frequencies = {}

    for char in text:

        if char in frequencies:

            frequencies[char] += 1

        else:

            frequencies[char] = 1

    return frequencies

# Функция для кодирования текста

def encode\_text(text, code\_dict):

    encoded\_text = ""

    for char in text:

        encoded\_text += code\_dict[char]

    return encoded\_text

# Функция для вычисления коэффициента сжатия

def calculate\_compression\_ratio(original\_text, encoded\_text):

    original\_size = len(original\_text) \* 8  # исходный размер

    encoded\_size = len(encoded\_text)  # закодированный размер в битах

    return original\_size / encoded\_size

# Основная функция

def main():

    # Запрос текста для шифровки

    text = input("Введите текст для шифрования: ")

    # Подсчет частот символов

    frequencies = calculate\_frequencies(text)

    # Сортировка символов по частоте, с учетом порядка появления символов

    sorted\_symbols = sort\_by\_frequency(frequencies, text)

    # Генерация кодов Шеннона-Фаннона

    code\_dict = {}

    shannon\_fano\_code(sorted\_symbols, code\_dict)

    # Кодирование текста

    encoded\_text = encode\_text(text, code\_dict)

    # Вывод кодов для каждого символа

    print("Коды для каждого символа:")

    for char, code in code\_dict.items():

        print(f"'{char}': {code}")

    # Вывод закодированного текста

    print("\nЗакодированный текст:")

    print(encoded\_text)

    # Коэффициент сжатия

    compression\_ratio = calculate\_compression\_ratio(text, encoded\_text)

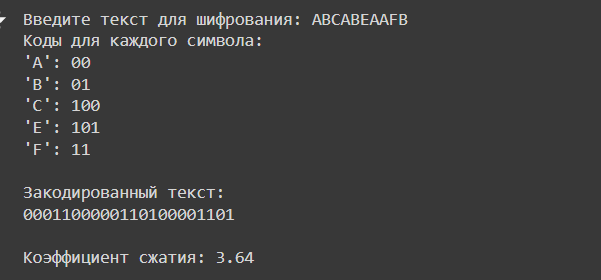
    print(f"\nКоэффициент сжатия: {compression\_ratio:.2f}")

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

    main()

Test:

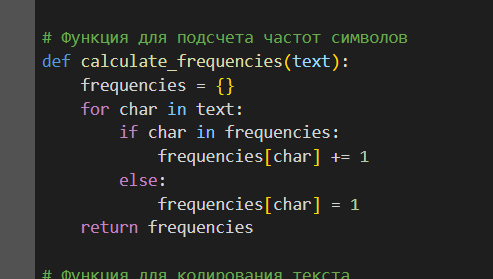
Word: ‘***ABCABEAAFB***’

Result: 

Explanation:

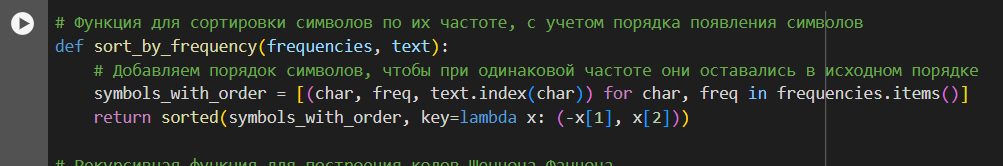
#### 1. ****Counting Symbol Frequencies****

The function calculate\_frequencies(text) iterates through each character in the string and counts its occurrences, storing this in a dictionary called frequencies. Each key in the dictionary is a character, and the value is its count in the text.



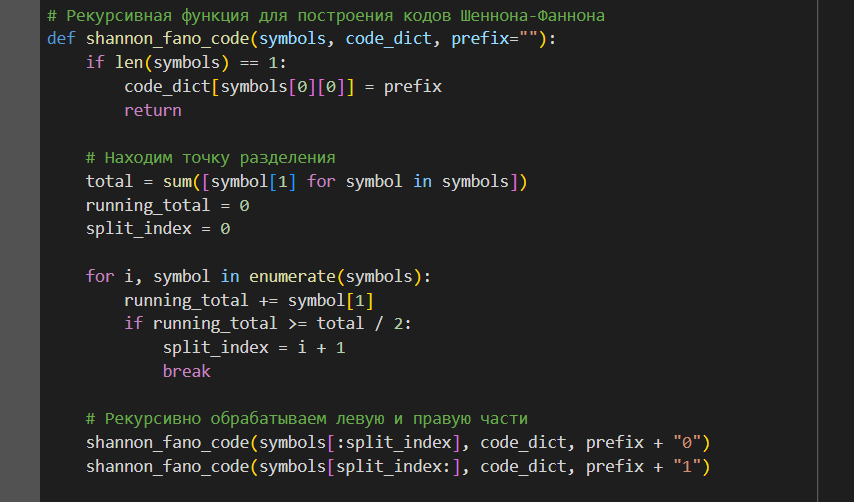
#### 2. ****Sorting Symbols by Frequency****

The sort\_by\_frequency(frequencies, text) function sorts characters by their frequency of occurrence. To maintain the original order of characters in cases of frequency ties, it also considers the first appearance index of each character in the text.



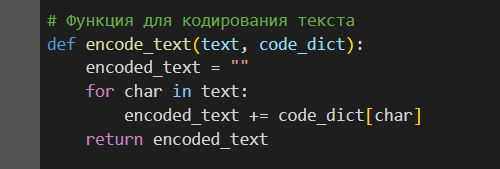
#### 3. ****Constructing Shannon-Fano Codes****

The shannon\_fano\_code(symbols, code\_dict, prefix="") function recursively divides the list of symbols into two groups that approximately equalize the sum of their frequencies. It assigns '0' to the left group and '1' to the right, continuing until each symbol has a unique code.



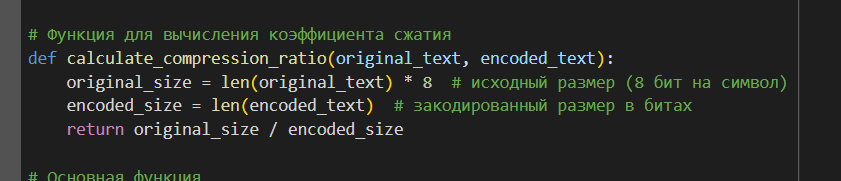
#### 4. ****Encoding the Text****

The encode\_text(text, code\_dict) function creates an encoded string using the code\_dict, which contains mappings from characters to their corresponding codes.



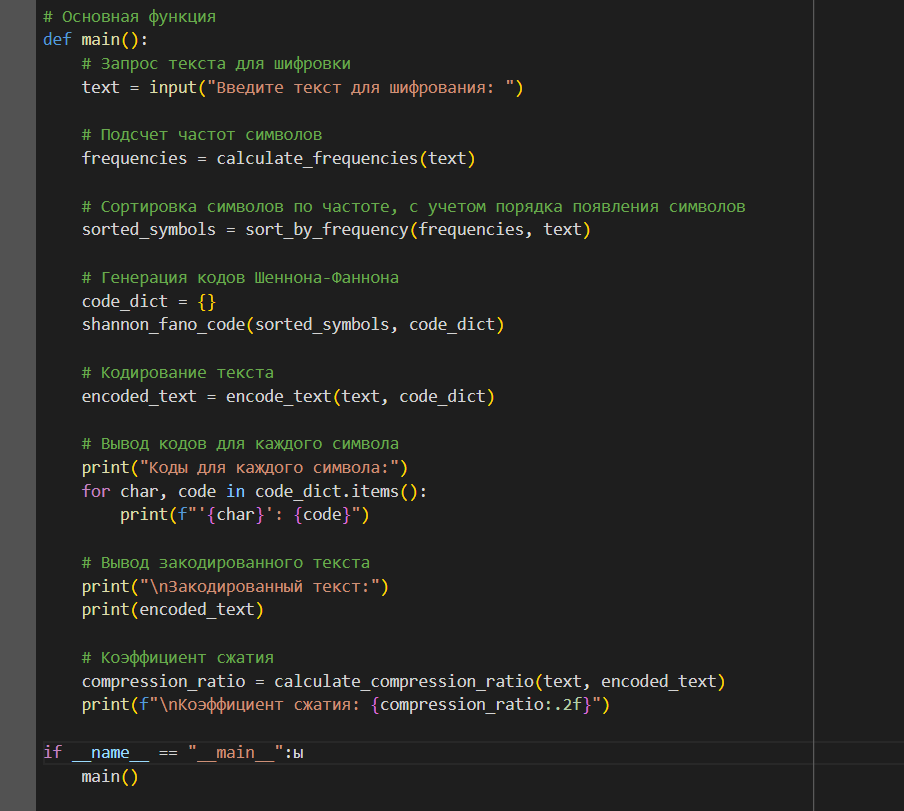
#### 5. ****Calculating Compression Ratio****

The calculate\_compression\_ratio(original\_text, encoded\_text) function calculates the compression ratio by comparing the size of the original text (in bits) to the encoded text size.



#### ****Main Function****

The main() function orchestrates the entire process: soliciting text input, generating frequencies, sorting, encoding, and outputting results.



**Conclusion**

The Shannon-Fano algorithm provides a systematic approach to data compression by effectively using the frequency of symbol occurrence to reduce the average code length needed to represent each symbol. This lab demonstrates the practical application and efficiency of the algorithm through both theoretical understanding and software implementation.