МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа № 5

по дисциплине: Теория информации

тема: «Исследование особенностей метода арифметического кодирования»

Выполнил: ст. группы ПВ-223
Игнатьев Артур Олегович

Проверил:
Твердохлеб Виталий Викторович

Белгород 2024г.

Лабораторная работа №5

«Исследование особенностей метода арифметического кодирования»

Цель работы: Исследовать особенности матода арифметического кодирования. Построить обработчик, реализующий данный алгоритм, используя его описание в прикрепленном файле.

1. Для изучения особенностей метода арифметического кодирования напишем программу на языке Python.

```
from decimal import Decimal  # Используется для обеспечения любой определяемой пользователем точности.


class ArithmeticEncoding:
    # Arithmetic Encoding — класс для построения арифметического кодирования.

    def __init__(self, frequency_table, save_stages=False):
        # Frequency_table: таблица частот в виде словаря, где ключ — это символ, а значение — частота.
        # save_stages: если True, то интервалы каждого этапа сохраняются в списке. Обратите внимание,
        # что установка save_stages=True может вызвать переполнение памяти, если сообщение большое.
        self.save_stages = save_stages
        if (save_stages == True):
            print("ВНИМАНИЕ: установка save_stages=True может привести к переполнению памяти, если сообщение большое.")
        self.probability_table = self.get_probability_table(frequency_table)

    def get_probability_table(self, frequency_table):
        # Вычисляет таблицу вероятностей на основе таблицы частот.

        total_frequency = sum(list(frequency_table.values()))
        probability_table = {}
        for key, value in frequency_table.items():
            probability_table[key] = value / total_frequency
        return probability_table

    def get_encoded_value(self, last_stage_probs):
        # После кодирования всего сообщения этот метод возвращает одно значение, представляющее все сообщение.

        last_stage_probs = list(last_stage_probs.values())
        last_stage_values = []
        for sublist in last_stage_probs:
            for element in sublist:
                last_stage_values.append(element)
        last_stage_min = min(last_stage_values)
        last_stage_max = max(last_stage_values)
        return (last_stage_min + last_stage_max) / 2

    def process_stage(self, probability_table, stage_min, stage_max):
        # Обработка этапа процесса кодирования/декодирования.

        stage_probs = {}
        stage_domain = stage_max - stage_min
        for term_idx in range(len(probability_table.items())):
            term = list(probability_table.keys())[term_idx]
```

```
            term_prob = Decimal(probability_table[term])
            cum_prob = term_prob * stage_domain + stage_min
            stage_probs[term] = [stage_min, cum_prob]
            stage_min = cum_prob
        return stage_probs

    def encode(self, msg, probability_table):
        # Кодирует сообщение.

        msg = list(msg)
        encoder = []
        stage_min = Decimal(0.0)
        stage_max = Decimal(1.0)
        for msg_term_idx in range(len(msg)):
            stage_probs = self.process_stage(probability_table, stage_min,
                                             stage_max)
            msg_term = msg[msg_term_idx]
            stage_min = stage_probs[msg_term][0]
            stage_max = stage_probs[msg_term][1]
            if self.save_stages:
                encoder.append(stage_probs)
        last_stage_probs = self.process_stage(probability_table, stage_min,
                                              stage_max)
        if self.save_stages:
            encoder.append(last_stage_probs)
        encoded_msg = self.get_encoded_value(last_stage_probs)
        return encoded_msg, encoder

    def decode(self, encoded_msg, msg_length, probability_table):
        # Декодирует сообщение.

        decoder = []
        decoded_msg = []
        stage_min = Decimal(0.0)
        stage_max = Decimal(1.0)
        for idx in range(msg_length):
            stage_probs = self.process_stage(probability_table, stage_min,
                                             stage_max)
            for msg_term, value in stage_probs.items():
                if encoded_msg >= value[0] and encoded_msg <= value[1]:
                    break
            decoded_msg.append(msg_term)
            stage_min = stage_probs[msg_term][0]
            stage_max = stage_probs[msg_term][1]
            if self.save_stages:
                decoder.append(stage_probs)
            if self.save_stages:
                last_stage_probs = self.process_stage(probability_table,
stage_min,
                                                      stage_max)
            decoder.append(last_stage_probs)
            return decoded_msg, decoder
```

Этот класс реализует алгоритм арифметического кодирования. Пошаговое описание его работы:

1. `__init__` метод инициализирует объект ArithmeticEncoding. Он принимает таблицу частот символов и параметр save_stages, который указывает, нужно ли сохранять промежуточные интервалы на каждом этапе

кодирования. Также в этом методе строится таблица вероятностей символов на основе частот символов.

2. `get_probability_table` метод вычисляет таблицу вероятностей на основе таблицы частот символов. Эта таблица используется для кодирования и декодирования сообщения.

3. `get_encoded_value` метод возвращает закодированное значение, представляющее всё сообщение после завершения кодирования.

4. `process_stage` метод обрабатывает каждый этап кодирования или декодирования. Он вычисляет вероятности символов для текущего этапа и обновляет диапазон кодирования.

5. `encode` метод кодирует сообщение, используя таблицу вероятностей. Он проходит по каждому символу сообщения, обновляя диапазон кодирования и сохраняя промежуточные значения, если параметр save_stages установлен в True.

6. `decode` метод декодирует закодированное сообщение, используя таблицу вероятностей. Он также сохраняет промежуточные значения, если параметр save_stages установлен в True.

Этот алгоритм предоставляет возможность кодирования и декодирования сообщений на основе статистических свойств символов, что позволяет достигнуть высокой степени сжатия информации.

Задание 2

Напишем программу для тестирования данного алгоритма на примерах из второй лабораторной работы: «в чащах юга жил бы цитрус? Да, но фальшивый экземпляр!» и «Victoria nulla est, Quam quae confessos animo quoque subjugat hostes»

Программа для первого предложения

```
import pyae

# Пример кодирования простого текстового сообщения с использованием модуля
PyAE.
frequency_table = {"в": 2, " ": 9, "ч": 1, "а": 5, "щ": 1,
                   "х": 1, "ю": 1, "г": 1, "ж": 1, "и": 3,
                   "л": 3, "б": 1, "ы": 2, "ц": 1, "т": 1,
                   "р": 2, "у": 1, "с": 1, "?": 1, "Д": 1,
                   ",": 1, "н": 1, "о": 1, "ф": 1, "ь": 1,
                   "ш": 1, "й": 1, "э": 1, "к": 1, "з": 1,
                   "е": 1, "м": 1, "п": 1, "я": 1, "!": 1, }
AE = pyae.ArithmeticEncoding(frequency_table=frequency_table,
                             save_stages=True)
original_msg = "в чащах юга жил бы цитрус? Да, но фальшивый экземпляр!"
print("Original Message: {msg}".format(msg=original_msg))
encoded_msg, encoder = AE.encode(msg=original_msg,
                                 probability_table=AE.probability_table)
print("Encoded Message: {msg}".format(msg=encoded_msg))
decoded_msg, decoder = AE.decode(encoded_msg=encoded_msg,
                                 msg_length=len(original_msg),
                                 probability_table=AE.probability_table)
print("Decoded Message: {msg}".format(msg=decoded_msg))
decoded_msg = "".join(decoded_msg)
print("Message Decoded Successfully? {result}".format(result=original_msg ==
                                                      decoded_msg))
original_size = len(original_msg)
compressed_size = len(str(encoded_msg))
print("Compression ratio:", original_size / compressed_size)
```

Результат работы программы:

```
Original Message: в чащах юга жил бы цитрус? Да, но фальшивый экземпляр!
Encoded Message: 0.0026579567888026001106514072 75
Message Decoded Successfully? False
Compression ratio: 1.6875
```

Программа для второго предложения:

```
import pyae

# Пример кодирования простого текстового сообщения с использованием модуля
PyAE.
frequency_table = {"V": 1, "i": 3, "c": 2, "t": 4, "o": 6,
                   "r": 1, "a": 6, " ": 9, "n": 3, "u": 7,
                   "l": 2, "e": 5, "s": 7, ",": 1, "Q": 1,
                   "m": 2, "q": 3, "f": 1, "b": 1, "j": 1,
                   "g": 1, "h": 1, }
AE = pyae.ArithmeticEncoding(frequency_table=frequency_table,
                             save_stages=True)
original_msg = "Victoria nulla est, Quam quae confessos animo quoque subjugat
hostes"
print("Original Message: {msg}".format(msg=original_msg))
encoded_msg, encoder = AE.encode(msg=original_msg,
                                 probability_table=AE.probability_table)
print("Encoded Message: {msg}".format(msg=encoded_msg))
decoded_msg, decoder = AE.decode(encoded_msg=encoded_msg,
                                 msg_length=len(original_msg),
                                 probability_table=AE.probability_table)
decoded_msg = "".join(decoded_msg)
print("Message Decoded Successfully? {result}".format(result=original_msg ==
                                                      decoded_msg))
original_size = len(original_msg)
compressed_size = len(str(encoded_msg))
print("Compression ratio:", original_size / compressed_size)
```

Результат работы программы:

```
Original Message: Victoria nulla est, Quam quae confessos animo quoque
subjgat hostes
Encoded Message: 0.000256299151751058993127670430 6
Message Decoded Successfully? False
Compression ratio: 2.0606060606060606
```

Пример программы для двоичного сообщения:

```
import pyae

# Пример кодирования простого текстового сообщения с использованием модуля
PyAE.
frequency_table = {"1": 6,
                   "0": 4}
AE = pyae.ArithmeticEncoding(frequency_table=frequency_table,
                             save_stages=True)
original_msg = "1100101011"
print("Original Message: {msg}".format(msg=original_msg))
encoded_msg, encoder = AE.encode(msg=original_msg,
                                 probability_table=AE.probability_table)
print("Encoded Message: {msg}".format(msg=encoded_msg))
decoded_msg, decoder = AE.decode(encoded_msg=encoded_msg,
                                 msg_length=len(original_msg),
                                 probability_table=AE.probability_table)
print("Decoded Message: {msg}".format(msg=decoded_msg))
decoded_msg = "".join(decoded_msg)
print("Message Decoded Successfully? {result}".format(result=original_msg ==
                                                      decoded_msg))
original_size = len(original_msg)
compressed_size = len(str(encoded_msg))
print("Compression ratio:", original_size / compressed_size)
```

Результат работы программы:

```
Original Message: 1100101011
Encoded Message: 0.32870983679999997036354670 85
Decoded Message: ['1', '1', '0', '0', '1', '0', '1', '0', '1', '1']
Message Decoded Successfully? True
Compression ratio: 0.3333333333333333
```

Вывод: в ходе выполнения лабораторной работы был написан обработчик, реализующий метод арифметического кодирования . В качестве тестовых данных были использованы строки из второй лабораторной работы. На их примере мы увидели что метод арифметического кодирования обладает большим коэффициентом сжатия в сравнении с методами Хаффмена и Шеннона – Фано. Так же данный алгоритм может сжимать сообщения записанные двоичным сообщением