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: 0.002657956788802600110651407275  
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: 0.0002562991517510589931276704306  
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: 0.3287098367999999703635467085  
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