ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____

ПРЕПОДАВАТЕЛЬ

Доцент				Суетина Т. А.
должность, уч. степень, звание		подпись, дата		инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

Энтропийные алгоритмы сжатия информации

Вариант 5

по курсу: Техника аудиовизуальных средств информации

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4128				Воробьев В. А.
			подпись, дата		инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1 Введение

1.1 Цель лабораторной работы

Освоить алгоритмы для сжатия информации.

1.2 Задание

Выполнить сжатие текста 4 способами:

- Метод Хаффмана;
- Метод Шенона-Фано;
- Арифметическим кодированием;
- Алгоритмом LZW.

Для каждого метода рассчитать коэффициент сжатия текста.

Вариант 5: ШОРОХ ОТ ДУБКА КАК БУДТО ХОРОШ

2 Выполнение работы

2.1 Теоретические сведения

$$K = \frac{V_{вх}}{V_{вых}}, \tag{1}$$

где K - степень сжатия.

2.2 Анализ исходного текста

Для начала проанализируем текст.

Таблица 2.1 - Количество вхождений символов.

Буква	Ш	О	Р	Х	space	Д	У	Б	К	А	Т
Кол-во	2	6	2	2	5	2	2	2	3	2	2

Всего букв: 30

2.3 Метод Шеннона-Фано

Таблица 2.2 - Решение методом Шеннона-Фано

Буква	О	space	К	Ш	Р	Х	Д	У	Б	А	Т
Частота	6	5	3	2	2	2	2	2	2	2	2
	1			0							
	1	0		1				0			
		1	0	1		0		1		0	
				1	0	1	0	1	0	1	0
ИТОГ	11	101	100	0111	0110	0101	0100	0011	0010	0001	0000

Итоговый код:

[0111 11 0110 11 0101]101[11 0000]101[0100 0011 0010 100 0001]101 [100 0001 100]101[0010 0011 0100 0000 11]101[0101 11 0110 11 0111]

Коэффициент сжатия по формуле 2.1: $K = 120/100 = 1.2$

2.4 Метод Хаффмана

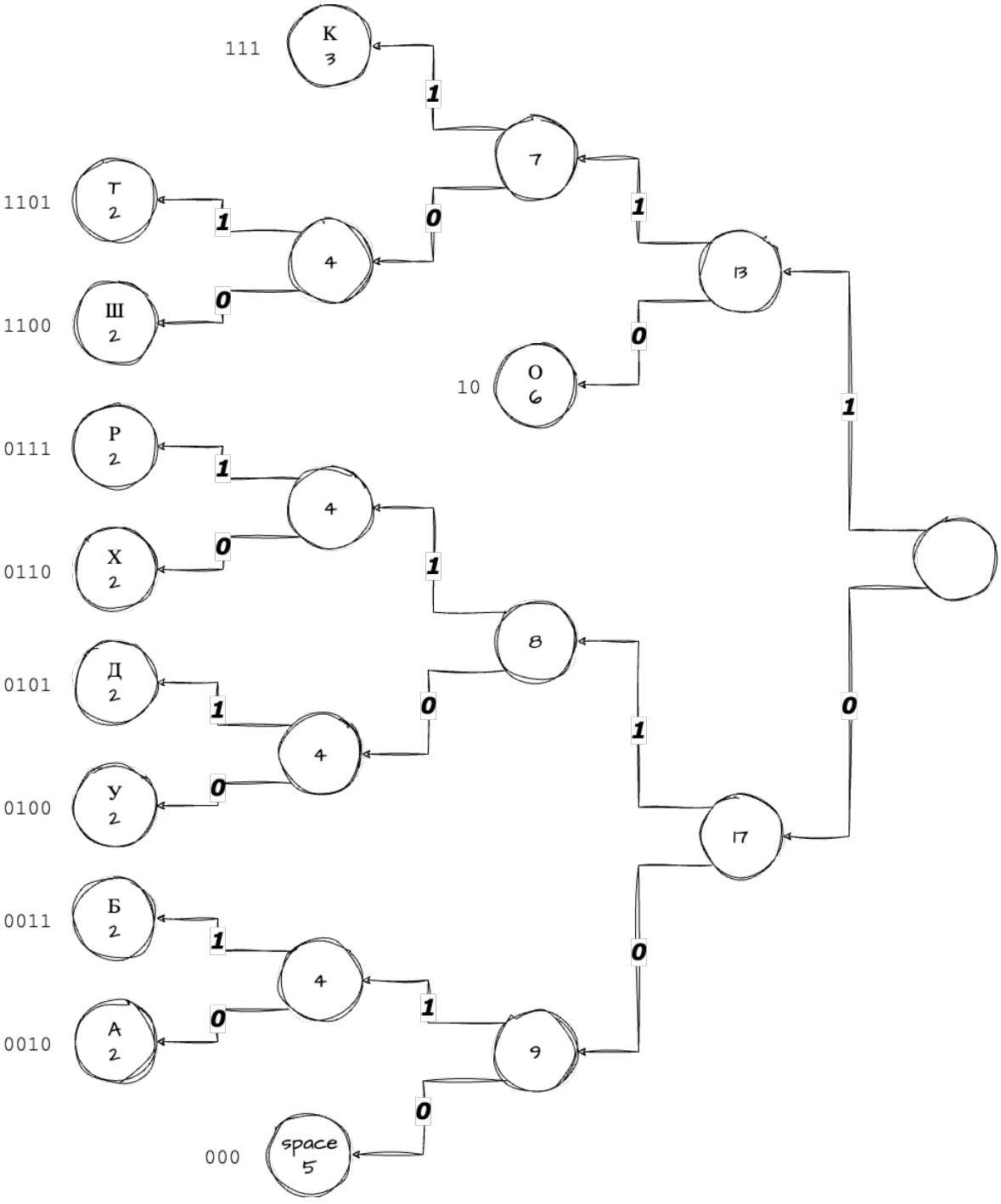

Рисунок 2.1 - Граф для метода Хаффмана

Итоговый код:

$[1100\ 10\ 0111\ 10\ 0110]000[10\ 1101]000[0101\ 0100\ 0011\ 111\ 0010]000$
$[111\ 0010\ 111]000[0011\ 0100\ 0101\ 1101\ 10]000[0110\ 10\ 0111\ 10\ 1100]$

Коэффициент сжатия по формуле 2.1: $K = 120/100 = 1.2$

2.5 Арифметическое кодирование

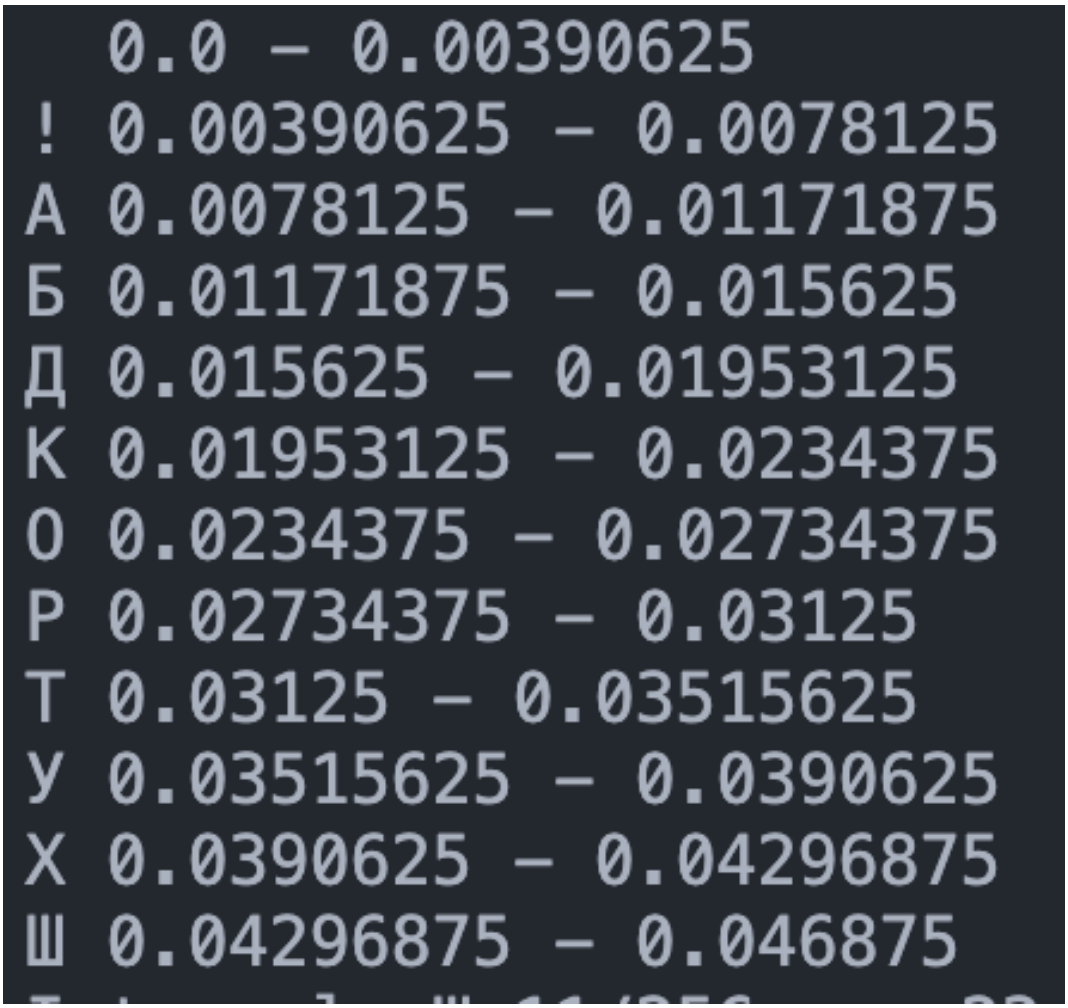

Рисунок 2.2 - Интервалы

Скрипт на Python представлен в Приложении, результат его работы изображен на 2.3.

Рисунок 2.3 - Результат арифметического кодирования

Видно, что получившийся полуинтервал имеет начало 0.044103411903879560948890382045153082129074456674419288651682950491687880565550757024937994599930336315264724006438913035930622779485280596693959854514512542866652599934821195911365615484247409971985121046600397676229476928710937 5 и конец 0.044103411903879560948890382045153082129074456674419292305513631962144485174502693941607376970436481928183263610316823751893910023646447565464431148008852525204896914136037521480319257953257167104155200831883121281862258911132 8125.

Исходя из рисунка 2.3, можно сделать вывод, что сообщение можно закодировать количеством бит равным = 171.

Коэффициент сжатия по формуле 2.1: $K = 248/171 = 1.45$

2.6 Алгоритм LZW

Скрипт на Python представлен в Приложении, результат его работы изображен на 2.4.

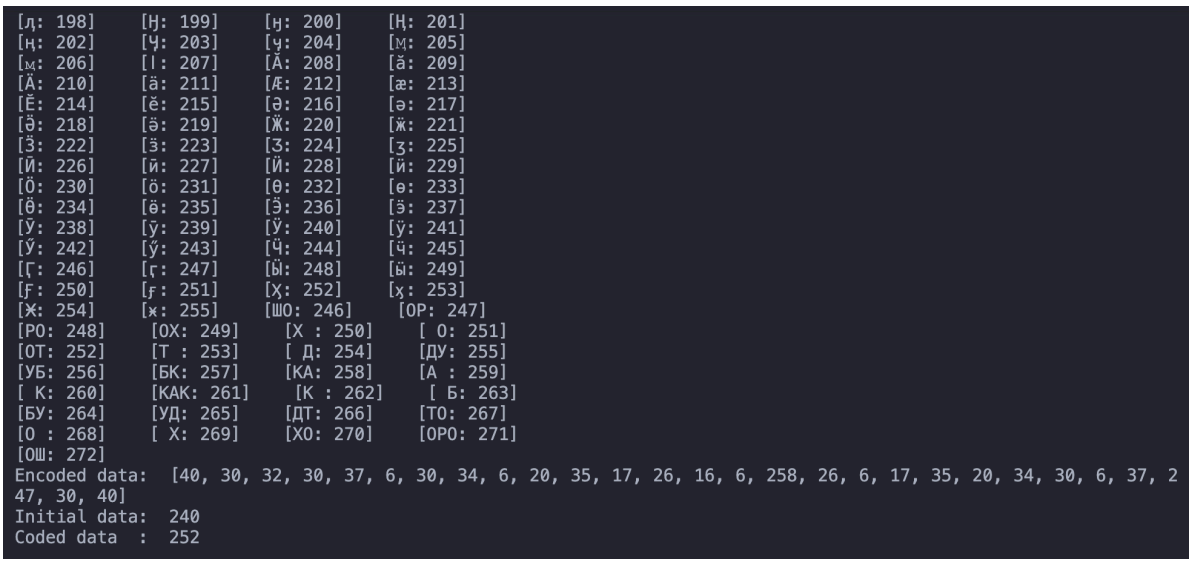

Рисунок 2.4 - Результат работы LZW

Коэффициент сжатия по формуле 2.1: $K = 252/240 = 0.95$

3 Вывод

В ходе выполнения лабораторной мы сжали исходную строку “ШОРОХ ОТ ДУБКА КАК БУДТО ХОРОШ” 4 разными способами. Для каждого способа мы посчитали коэффициент сжатия текста, и получили следующие значения:

1. Арифметическое кодирование = 1.45
2. Метод Хаффмана = 1.2
3. Метод Шенона-Фано = 1.2
4. Алгоритм LZW = 0.95

Как мы видим, арифмитическое кодирование имеет самую высокую степень сжатия, но тем не требует значительно большую мощность вычислительных ресурсов.

Метод Хаффмана и метод Шенона-Фано имеет одинаковую степень сжатия. Эти алгоритмы являются простыми в реализации, поэтому для некоторых задач могут быть весьма эффективными.

Алгоритм LZW имеет степень сжатия меньше единицы. Так получилось, потому что мы не учитывали то, что для предыдущих алгоритмов нужно передавать таблицу кодировок. Для алгоритма LZW этого не требуется, что является ощутимым плюсом.

Полученные навыки пригодятся нам при создании ПО чувствительного к размеру информации.

ПРИЛОЖЕНИЕ

Листинг арифметического кодирования:

```
import decimal as decimal
from fractions import Fraction
from collections import Counter
import math
import os


def arithmetic_encoding(input, bounds):
    lower, upper = Fraction(0), Fraction(1)

    for char in input:
        length = upper - lower
        lower += length * Fraction(bounds[char][0])
        upper = lower + length * Fraction(bounds[char][1])
        print(f"Interval: {char}", lower, " - ", upper)
    return [lower, upper]


def decimal_from_fraction(frac):
    return frac.numerator / decimal.Decimal(frac.denominator)


def get_accuracy(num1, num2):
    length = len(num1)
    match = False
    for i in range(length):
        if num1[i] == num2[i]:
            continue
        if (match):
            return min(length, i+1)
        match = True
    raise ValueError("Increase decimal.getcontext().prec")


input = "ШОРОХ ОТ ДУБКА КАК БУДТО ХОРОШ!"
decimal.getcontext().prec = 500
coding = 256

bounds = dict()
```

```
chars = sorted(Counter(input).keys())
for i, char in enumerate(chars):
    bounds[char] = (Fraction(i, coding), Fraction(i + 1,
       coding))

intervals = [f"{chars[i]} {float(bounds[chars[i]][0])} - {
   float(bounds[chars[i]][1])}" for i in range(len(chars))]
for interval in intervals:
    print(interval)

result = arithmetic_encoding(input, bounds)
lower_bound = str(decimal_from_fraction(result[0]))
upper_bound = str(decimal_from_fraction(result[1]))
accuracy = get_accuracy(lower_bound, upper_bound)
prefix = os.path.commonprefix([lower_bound, upper_bound])
print()
print(f'Final interval: {lower_bound}, {upper_bound}')
print(f'Interval cut  : {lower_bound[:accuracy]}, {
   upper_bound[:accuracy]}')
print(f'Prefix        : {prefix}')
coded_num = decimal.Decimal(prefix[-1:1:-1])
print("Initial data: ", len(input) * math.ceil(math.log2(
   coding)))
print("Coded data  : ", math.ceil(math.log2(coded_num)))
```

Листинг LZW кодирования:

```
def lzw_encode(input_string, dictionary):
    code = []
    s = ""
    for c in input_string:
        sc = s + c
        if sc in dictionary:
            s = sc
        else:
            code.append(dictionary[s])
            dictionary[sc] = len(dictionary)
            s = c
    if s:
        code.append(dictionary[s])

    for i, (key, value) in enumerate(dictionary.items()):
        print(f"[{key}: {value}]", end="    ")
```

```
        if (i + 1) % 4 == 0:
            print()
    print()
    return code, max(code).bit_length() * len(code)


input = "ШОРОХ ОТ ДУБКА КАК БУДТО ХОРОШ"
alphabet = list(set(input))

dictionary = {alphabet[i]: i for i in range(len(alphabet))}
for i in range(len(dictionary), 256):
    dictionary[chr(i + 1024)] = i

print(dictionary)

code, size_in_bits = lzw_encode(input, dictionary)

print("Encoded data: ", code)
print("Initial data: ", len(input) * 8)
print("Coded data  : ", size_in_bits)
```

Листинг кодирования по словарю:

```
word = "ШОРОХ ОТ ДУБКА КАК БУДТО ХОРОШ"


def fill_word(word, dict):
    result = "["
    bits = 0
    for i in word:
        bits += len(dict[i])
        if (i == " "):
            result += f"]\\allowbreak{dict[i]}\\allowbreak["
        else:
            result += f"\,{dict[i]}\,"
    result += "]"
    print(result)
    print(f"Bits: {bits}")


shenon = {
    "К": "111",
    "Т": "1101",
```

```
21      "Ш": "1100",
22      "Р": "0111",
23      "Х": "0110",
24      "Д": "0101",
25      "У": "0100",
26      "Б": "0011",
27      "А": "0010",
28      " ": "000",
29      "О": "10",
30  }
31
32  xaphan = {
33      "О": "11",
34      " ": "101",
35      "К": "100",
36      "Ш": "0111",
37      "Р": "0110",
38      "Х": "0101",
39      "Д": "0100",
40      "У": "0011",
41      "Б": "0010",
42      "А": "0001",
43      "Т": "0000",
44  }
45
46  fill_word(word, shenon)
47  print()
48  fill_word(word, xaphan)
```