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

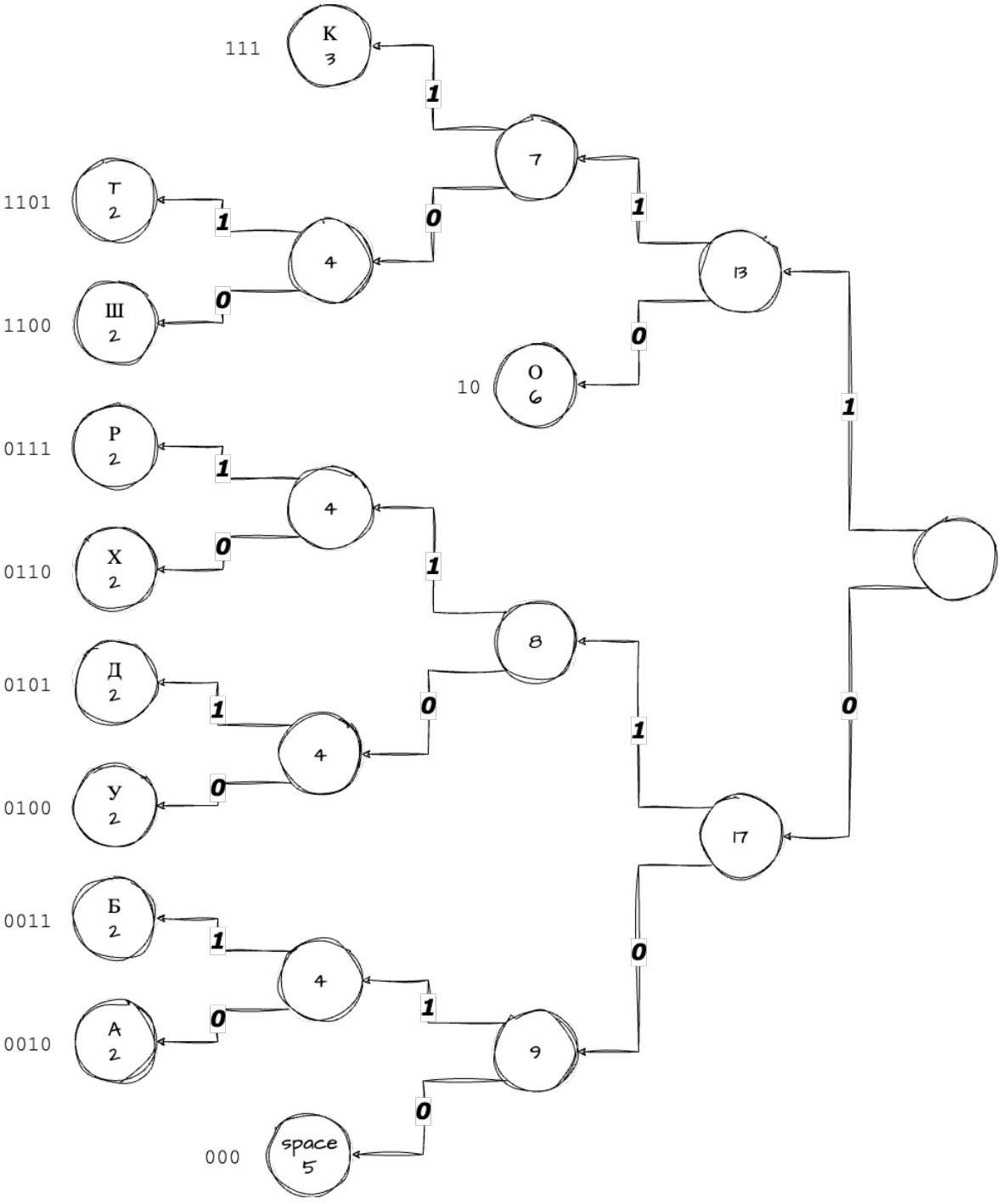

Рисунок 2.1 - Граф для метода Хаффмана

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

Скрипт на Python представлен в Приложении, результат его работы изображен на 2.2.

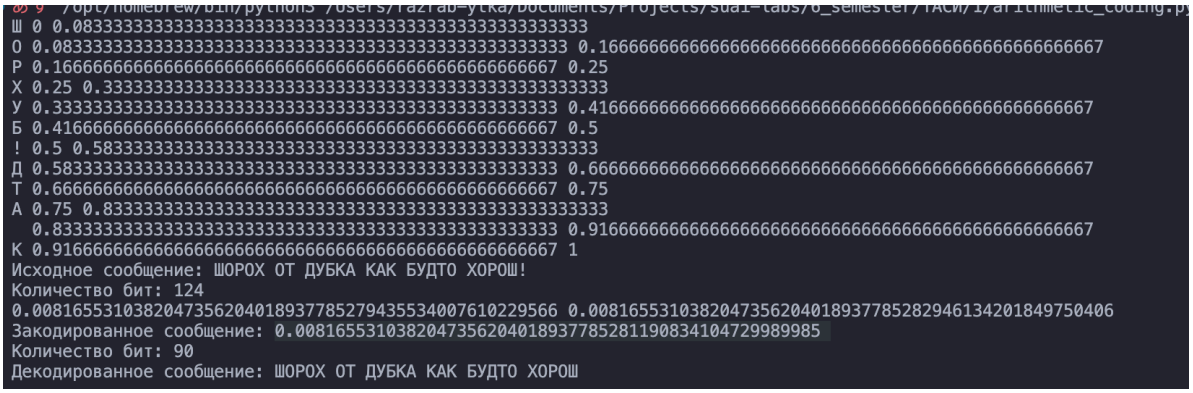

Рисунок 2.2 - Результат арифметического кодирования

Видно, что сообщение можно закодировать $0.689963272214418381846575040112511608865095537019 45$.

Исходя из рисунка 2.2, можно сделать вывод, что сообщение можно закодировать количеством бит равным = 90.

Коэффициент сжатия по формуле 2.1: $K = 124/90 = 1.34$

2.6 Алгоритм LZW

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

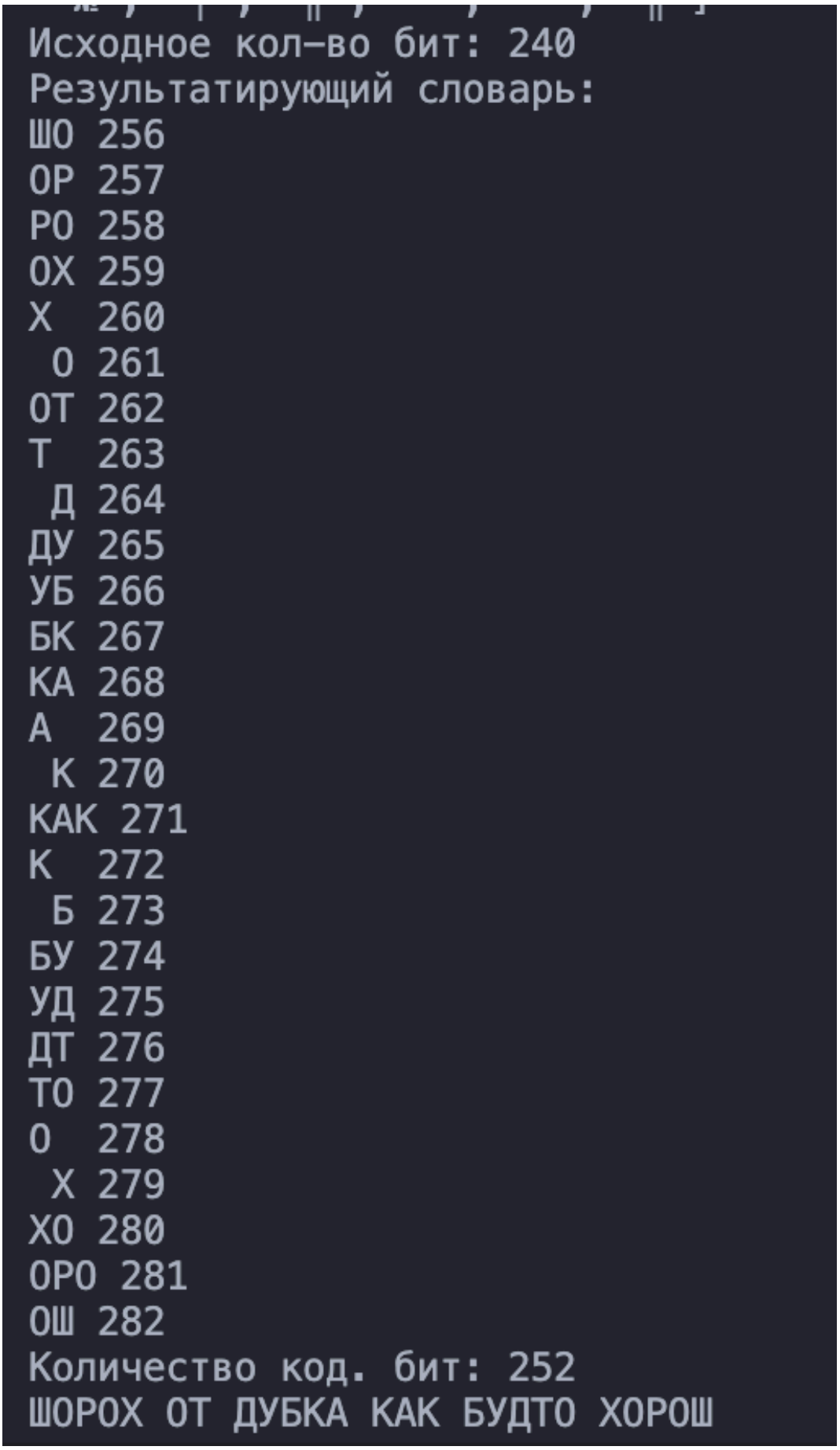

Рисунок 2.3 - Результат работы LZW

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

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 = 1.34
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. Тем не менее если подать строку больше длинны, то мы получим более высокие результаты.

ПРИЛОЖЕНИЕ

Листинг арифметического кодирования:

```
import math
import help as calc
from decimal import Decimal, getcontext

getcontext().prec = 50

input_string = "ШОРОХ ОТ ДУБКА КАК БУДТО ХОРОШ!"
dict_words = set(input_string)
dict_len = len(dict_words)

intervals = {
    ch: (Decimal(i) / Decimal(dict_len), Decimal(i + 1) /
       Decimal(dict_len))
    for i, ch in enumerate(dict_words)
}

for key, value in intervals.items():
    print(key, value[0], value[1])


def encode(message):
    left = Decimal(0.0)
    right = Decimal(1.0)
    for ch in message:
        new_right = left + (right - left) * intervals[ch][1]
        new_left = left + (right - left) * intervals[ch][0]
        left = new_left
        right = new_right
    print(left, right)
    return (left + right) / Decimal(2)


def decode(code):
    result = ''
    while True:
        for ch, (start, end) in intervals.items():
            if start <= code < end:
                if ch == '!':
                    return result
```

```
                result += ch
                code = (code - start) / (end - start)
                break


print(f"Исходное сообщение: {input_string}")
print(f"Количество бит: {len(input_string) * math.ceil(math.
    log2(dict_len))}")
code = encode(input_string)
print("Закодированное сообщение:", code)
print(f"Количество бит: {calc.bits(code)}")
decoded_message = decode(code)
print("Декодированное сообщение:", decoded_message)
```

Листинг LZW кодирования:

```
import string
from math import ceil, log2


def lzw_encode(input_string, dict_symbols):
    code = []
    current_string = input_string[0]
    for next_symbol in input_string[1:]:
        new_string = current_string + next_symbol
        if new_string in dict_symbols:
            current_string = new_string
        else:
            code.append(dict_symbols[current_string])
            dict_symbols[new_string] = len(dict_symbols)
            current_string = next_symbol
    code.append(dict_symbols[current_string])
    print("Результатирующий словарь:")
    for (i , (k, v)) in enumerate(dict_symbols.items(), ):
        if(i < 256):
            continue
        print(k, v)
    return code


def lzw_decode(encoded_sequence, dict_symbols):
    reverse_dict = {value: key for key, value in dict_symbols
        .items()}
```

```
27      decoded_string = reverse_dict[encoded_sequence[0]]
28      current_string = reverse_dict[encoded_sequence[0]]
29      for code in encoded_sequence[1:]:
30          if code in reverse_dict:
31              next_string = reverse_dict[code]
32          else:
33              next_string = current_string + current_string[0]
34          decoded_string += next_string
35          reverse_dict[len(reverse_dict)] = current_string +
              next_string[0]
36          current_string = next_string
37      return decoded_string
38
39
40  input_string = "ШОРОХ ОТ ДУБКА КАК БУДТО ХОРОШ"
41  dict = sorted(set("!\"#$%&'()*+,-. /0123456789:;<=>?
      @ABCDEFGHIJKLMNOPQRSTUVWXYZ[]^_`abcdefghijklmnopqrstuvwxyz
      {|}~
      №╚╘╒╬АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзийклмнопрсту
      "))
42  print(dict)
43  dict_symbols = {symbol: index for index,
44                  symbol in enumerate(dict)}
45
46  print(f"Исходное кол-во бит: {len(input_string) * ceil(log2(
      len(dict)))}")
47  encoded_sequence = lzw_encode(input_string, dict_symbols)
48  encoded_length = len(encoded_sequence)
49  print(f"Количество код. бит: {encoded_length * ceil(log2(max(
      encoded_sequence)))}")
50  decoded_string = lzw_decode(encoded_sequence, dict_symbols)
51  print(decoded_string)
```

Листинг кодирования по словарю:

```
1   word = "ШОРОХ ОТ ДУБКА КАК БУДТО ХОРОШ"
2
3
4   def fill_word(word, dict):
5       result = "["
6       bits = 0
7       for i in word:
8           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",
    "Ш": "1100",
    "Р": "0111",
    "Х": "0110",
    "Д": "0101",
    "У": "0100",
    "Б": "0011",
    "А": "0010",
    " ": "000",
    "О": "10",
}

xaphan = {
    "О": "11",
    " ": "101",
    "К": "100",
    "Ш": "0111",
    "Р": "0110",
    "Х": "0101",
    "Д": "0100",
    "У": "0011",
    "Б": "0010",
    "А": "0001",
    "Т": "0000",
}

fill_word(word, shenon)
print()
fill_word(word, xaphan)
```