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

$$K = \frac{V_{вх}}{V_{вых}}, \tag{2.1}$$

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

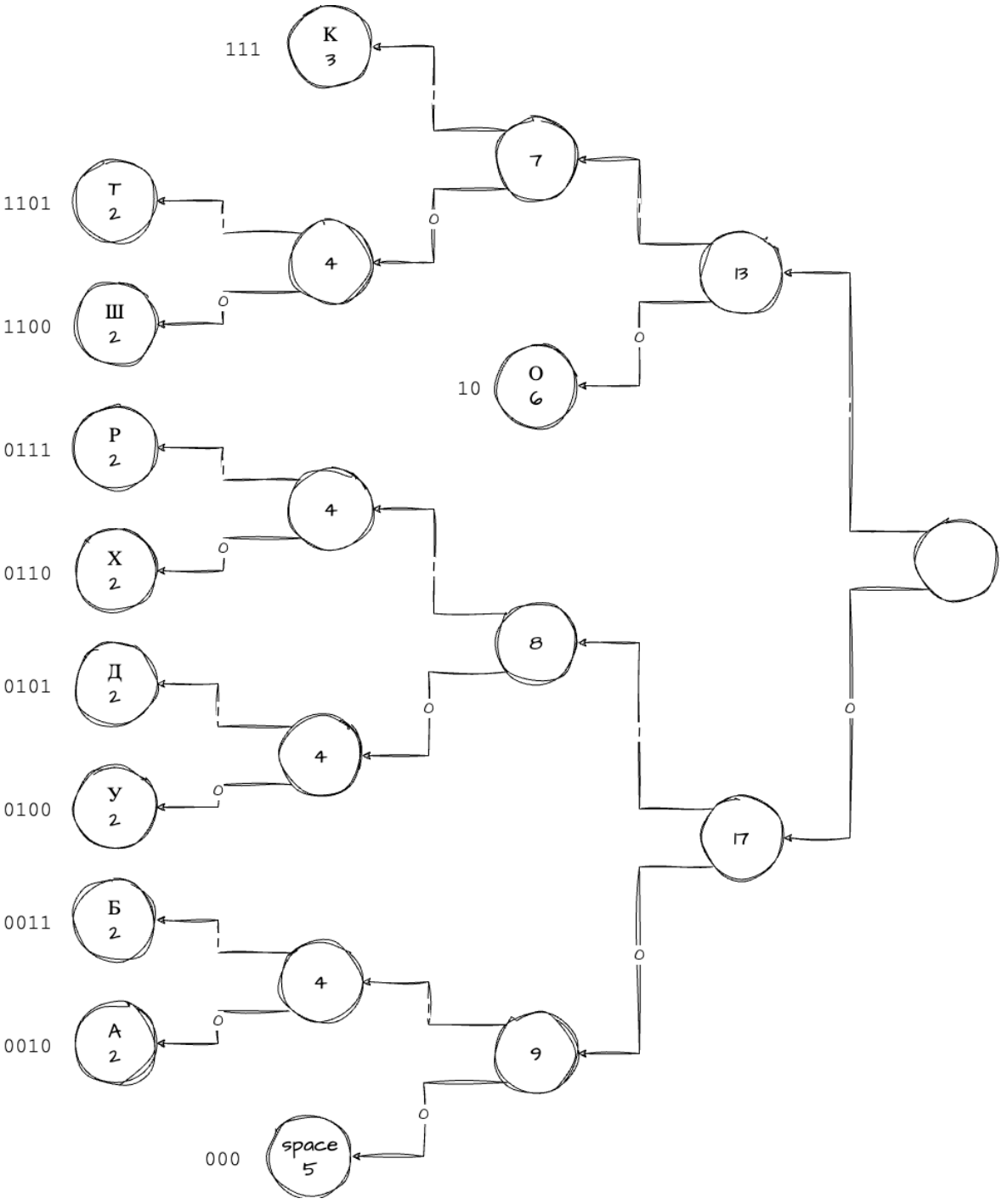

Рисунок 2.1 - Граф для метода Хаффмана

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

Таблица 2.3 - Таблица интервалов

Ш	О	Р	Х	space	Д	У	Б	К	А	Т
$\frac{2}{30}$	$\frac{8}{30}$	$\frac{10}{30}$	$\frac{12}{30}$	$\frac{17}{30}$	$\frac{19}{30}$	$\frac{21}{30}$	$\frac{23}{30}$	$\frac{26}{30}$	$\frac{28}{30}$	$\frac{30}{30}$
$\frac{0}{30}$	$\frac{2}{30}$	$\frac{8}{30}$	$\frac{10}{30}$	$\frac{12}{30}$	$\frac{17}{30}$	$\frac{19}{30}$	$\frac{21}{30}$	$\frac{23}{30}$	$\frac{26}{30}$	$\frac{28}{30}$

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

Рисунок 2.2 - Результат арифметического кодирования

Видно, что получившийся полуинтервал имеет начало $0.007491234978280553441395982224376491768148032361088748495779293270670468164890615017316007369538876504$ и конец $0.007491234978280553441395982224391606091525613885464659445273216987363890074114548014936179948312863938$.

Исходя из рисунка 2.2, можно сделать вывод, что сообщение можно закодировать числом $0.007491234978280553441395982224 38 = 0.0000000111$

01100011000011011011011011000011001000110010110100111001111001010101010110010010101101010011, то есть 000000011101100011000011011011011100001100100011001011010011100111100101010101011001001010110101011.

Коэффициент сжатия по формуле 2.1: $K = 120/99 = 1.21$

2.6 Алгоритм LZW

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

```
Ш: 0    Д: 1     : 2    Р: 3
О: 4    Б: 5    А: 6    К: 7
Х: 8    Т: 9    У: 10    ШО: 11
ОР: 12    РО: 13    ОХ: 14    Х : 15
 О: 16    ОТ: 17    Т : 18     Д: 19
ДУ: 20    УБ: 21    БК: 22    КА: 23
А : 24     К: 25    КАК: 26    К : 27
 Б: 28    БУ: 29    УД: 30    ДТ: 31
ТО: 32    О : 33     Х: 34    ХО: 35
ОРО: 36    ОШ: 37    Encoded data:  [0, 4, 3, 4, 8, 2, 4, 9, 2, 1, 10, 5, 7,
6, 2, 23, 7, 2, 5, 10, 1, 9, 4, 2, 8, 12, 4, 0]
Size of the encoded data in bits:  140
```

Рисунок 2.3 - Результат работы LZW

Коэффициент сжатия по формуле 2.1: $K = 120/140 = 0.85$

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 = 6
2. Метод Хаффмана = 1.2
3. Метод Шенона-Фано = 1.2
4. Алгоритм LZW = 1.21

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:

```
# Для начала pip install fractions
# Или pip3 install decimal
import decimal as decimal
from fractions import Fraction
from collections import Counter
import os


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

    for char, l, u in bounds:
        range = upper - lower
        upper = lower + range * Fraction(u)
        lower = lower + range * Fraction(l)

        print(f'Character: {char}, Interval: ({lower}, {upper
            })')

    return [lower, upper]


def create_bound_map(input):
    char_counts = Counter(input)
    total_chars = len(input)

    bound_map = {}
    lower_bound = Fraction(0)

    for char, count in char_counts.items():
        upper_bound = lower_bound + Fraction(count,
            total_chars)
        bound_map[char] = (lower_bound, upper_bound)
        lower_bound = upper_bound

    return bound_map


def decimal_from_fraction(frac):
```

```
38      return frac.numerator / decimal.Decimal(frac.denominator)
39
40
41  def get_accuracy(num1, num2):
42      length = len(num1)
43      match = False
44      for i in range(length):
45          if num1[i] == num2[i]:
46              continue
47          if (match):
48              return min(length, i+1)
49          match = True
50      print("Increase decimal.getcontext().prec")
51
52  input = "ШОРОХ ОТ ДУБКА КАК БУДТО ХОРОШ!"
53  bound_map = create_bound_map(input)
54
55  bounds = [(char, bound_map[char][0], bound_map[char][1]) for
        char in input]
56  result = arithmetic_encoding(bounds)
57  decimal.getcontext().prec = 100
58  lower_bound = str(decimal_from_fraction(result[0]))
59  upper_bound = str(decimal_from_fraction(result[1]))
60  split_i = get_accuracy(lower_bound, upper_bound)
61  print()
62  print(f'Final interval: {lower_bound}, {upper_bound}')
63  print(f'Interval cut  : {lower_bound[:split_i]}, {upper_bound
        [:split_i]}')
64  print(f'Prefix        : {os.path.commonprefix([lower_bound,
        upper_bound])}')
```

Листинг LZW кодирования:

```
1   def lzw_encode(input_string, dictionary):
2       code = []
3       s = ""
4       for c in input_string:
5           sc = s + c
6           if sc in dictionary:
7               s = sc
8           else:
9               code.append(dictionary[s])
10              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"
dictionary = {element: i for i, element in enumerate(set(
    input))}

code, size_in_bits = lzw_encode(input, dictionary)

print("Encoded data: ", code)
print("Size of the encoded data in bits: ", 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)
```