

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

<b>1</b>	<b>Введение</b>	<b>3</b>
1.1	Цель лабораторной работы	3
1.2	Задание	3
<b>2</b>	<b>Выполнение работы</b>	<b>4</b>
2.1	Теоретические сведения	4
2.2	Анализ исходного текста	4
2.3	Метод Шеннона-Фано	4
2.4	Метод Хаффмана	5
2.5	Арифметическое кодирование	5
2.6	Алгоритм LZW	7
<b>3</b>	<b>Вывод</b>	<b>8</b>
	<b>Приложение</b>	<b>9</b>

## **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

$$K = \frac{V_{вх}}{V_{вых}}, \quad (2.1)$$

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

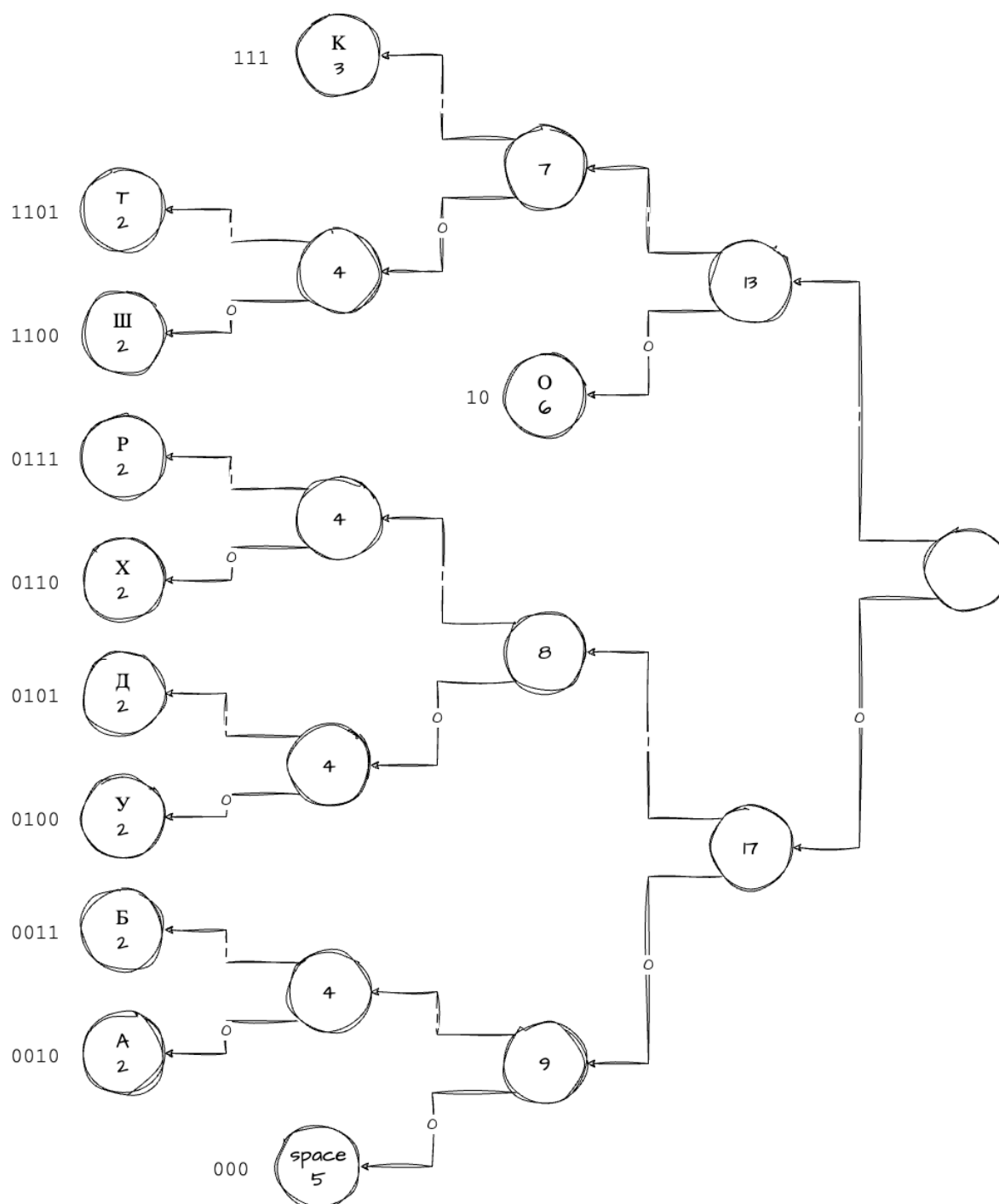


Рисунок 2.1 - Граф для метода Хаффмана

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.

```

Character: Ш, Interval: (0, 2/31)
Character: О, Interval: (4/961, 16/961)
Character: Р, Interval: (220/29791, 244/29791)
Character: О, Interval: (6868/923521, 7012/923521)
Character: Х, Interval: (214348/28629151, 214636/28629151)
Character: , Interval: (6648244/887503681, 6649684/887503681)
Character: , Interval: (206098444/27512614111, 206107084/27512614111)
Character: Т, Interval: (6389198644/852891037441, 6389215924/852891037441)
Character: , Interval: (198065365324/26439622160671, 198065451724/26439622160671)
Character: Д, Interval: (6140027966644/819628286980801, 6140028139444/819628286980801)
Character: У, Interval: (190340870594764/25408476896404831, 190340870940364/25408476896404831)
Character: Б, Interval: (5900566996386484/787662783788549761, 5900566997077684/787662783788549761)
Character: К, Interval: (182917576905261004/24417546297445042591, 182917576907334604/24417546297445042591)
Character: А, Interval: (5670444884121151924/756943935220796320321, 5670444884125299124/756943935220796320321)
Character: , Interval: (175783791407805476044/23465261991844685929951, 175783791407826212044/23465261991844685929951)
Character: К, Interval: (5449297533642488157364/727423121747185263828481, 5449297533642550365364/727423121747185263828481)
Character: А, Interval: (168928223542918874702284/22550116774162743178682911, 168928223542918999118284/22550116774162743178682911)
Character: К, Interval: (5236774929830488226170804/699053619999045038539170241, 5236774929830488599418804/699053619999045038539170241)
Character: , Interval: (162340022824745139490270924/21670662219970396194714277471, 162340022824745141356510924/21670662219970396194714277471)
Character: Б, Interval: (5032540707567099367121918644/671790528819082282036142601601, 5032540707567099370854398644/671790528819082282036142601601)
Character: У, Interval: (156008761934580080459161557964/20825506393931550743120420649631, 156008761934580080466626517964/20825506393931550743120420649631)
Character: Д, Interval: (4836271619971982494375842536884/645590698195138073036733040138561, 4836271619971982494390772456884/645590698195138073036733040138561)
Character: Т, Interval: (149924420219131457325904927283404/20013311644049280264138724244295391, 149924420219131457325934787123404/20013311644049280264138724244295391)
Character: О, Interval: (4647657026793075177103112465465524/620412660965527688188300451573157121, 464765702679307517710329162450524/620412660965527688188300451573157121)
Character: , Interval: (144077367830585330490198636337911244/19232792489931358333837313998767870751, 14407736783058533049019953213311244/19232792489931358333837313998767870751)
Character: Х, Interval: (4466398402748145245196166684427248564/596216567187872108348956733961803993281, 4466398402748145245196168476017648564/596216567187872108348956733961803993281)
Character: О, Interval: (138458350485192502601081170800425505484/18482713582824035358817658752815923791711, 138458350485192502601081181549967905484/18482713582824035358817658752815923791711)
Character: Р, Interval: (4292208865040967580633516380809529870004/572964121067545096123347421337293637543041, 4292208865040967580633516402308614670004/572964121067545096123347421337293637543041)
Character: О, Interval: (133058474816269994999639007848093595570124/17761887753093897979823770061456102763834271, 13305847481626994999639007977088104370124/17761887753093897979823770061456102763834271)
Character: Ш, Interval: (133058474816269994999639007848093595570124/17761887753093897979823770061456102763834271, 4124812719304369844988809243548890480273844/550618520345910837374536871905139185678862401)
Character: !, Interval: (127869194298435465194653086549757615870889164/17069174130723235958610643029059314756044734431, 4124812719304369844988809243548890480273844/550618520345910837374536871905139185678862401)
Final interval: 0.007491234978280553441395982224376491768148032361088748495779293270670468164890615017316007369538876504, 0.007491234978280553441395982224391606091525613885464659445273216987363890074114548014936179948312863938
Interval cut : 0.007491234978280553441395982224376, 0.007491234978280553441395982224391
Prefix : 0.0074912349782805534413959822243

```

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  $0.00749123497828055344139598222438 = 0.0000000111$

011000110000110110110110110000110010001100101101001110011110010101  
 01011001001010110101011, то есть 000000011101100011000011011011011011  
 000011001000110010110100111001111001010101011001001010110101011.

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, арифметическое 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:

```
1  # Для начала pip install fractions
2  # Или pip3 install decimal
3  import decimal as decimal
4  from fractions import Fraction
5  from collections import Counter
6  import os
7
8
9  def arithmetic_encoding(bounds):
10     lower, upper = Fraction(0), Fraction(1)
11
12     for char, l, u in bounds:
13         range = upper - lower
14         upper = lower + range * Fraction(u)
15         lower = lower + range * Fraction(l)
16
17         print(f'Character: {char}, Interval: ({lower}, {upper}
18             ))')
19
20     return [lower, upper]
21
22 def create_bound_map(input):
23     char_counts = Counter(input)
24     total_chars = len(input)
25
26     bound_map = {}
27     lower_bound = Fraction(0)
28
29     for char, count in char_counts.items():
30         upper_bound = lower_bound + Fraction(count,
31             total_chars)
32         bound_map[char] = (lower_bound, upper_bound)
33         lower_bound = upper_bound
34
35     return bound_map
36
37 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 = "ЦЮРОХ 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
56           char in input]
57 result = arithmetic_encoding(bounds)
58 decimal.getcontext().prec = 100
59 lower_bound = str(decimal_from_fraction(result[0]))
60 upper_bound = str(decimal_from_fraction(result[1]))
61 split_i = get_accuracy(lower_bound, upper_bound)
62 print()
63 print(f'Final interval: {lower_bound}, {upper_bound}')
64 print(f'Interval cut  : {lower_bound[:split_i]}, {upper_bound
65       [:split_i]}')
66 print(f'Prefix          : {os.path.commonprefix([lower_bound,
67       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)

```

```

11         s = c
12     if s:
13         code.append(dictionary[s])
14
15     for i, (key, value) in enumerate(dictionary.items()):
16         print(f"{key}: {value}", end="    ")
17         if (i + 1) % 4 == 0:
18             print()
19     print()
20     return code, max(code).bit_length() * len(code)
21
22
23 input = "ШОРОХ ОТ ДУБКА КАК БУДТО ХОРОШ"
24 dictionary = {element: i for i, element in enumerate(set(
25     input))}
26
27 code, size_in_bits = lzw_encode(input, dictionary)
28
29 print("Encoded data: ", code)
30 print("Size of the encoded data in bits: ", size_in_bits)

```

### **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])
9         if (i == " "):
10             result += f"]\\allowbreak{dict[i]}\\allowbreak["
11         else:
12             result += f"\,{dict[i]}\,"
13     result += "]"
14     print(result)
15     print(f"Bits: {bits}")
16
17
18 shenon = {
19     "K": "111",
20     "T": "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)

```