

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 | 3 |
| 1.1 | Цель лабораторной работы | 3 |
| 1.2 | Задание | 3 |
| 2 | Выполнение работы | 4 |
| 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 | 6 |
| 2.6 | Алгоритм LZW | 7 |
| 3 | Вывод | 9 |
| | ПРИЛОЖЕНИЕ | 10 |

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

$$K = \frac{V_{вх}}{V_{вых}}, \quad (1)$$

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

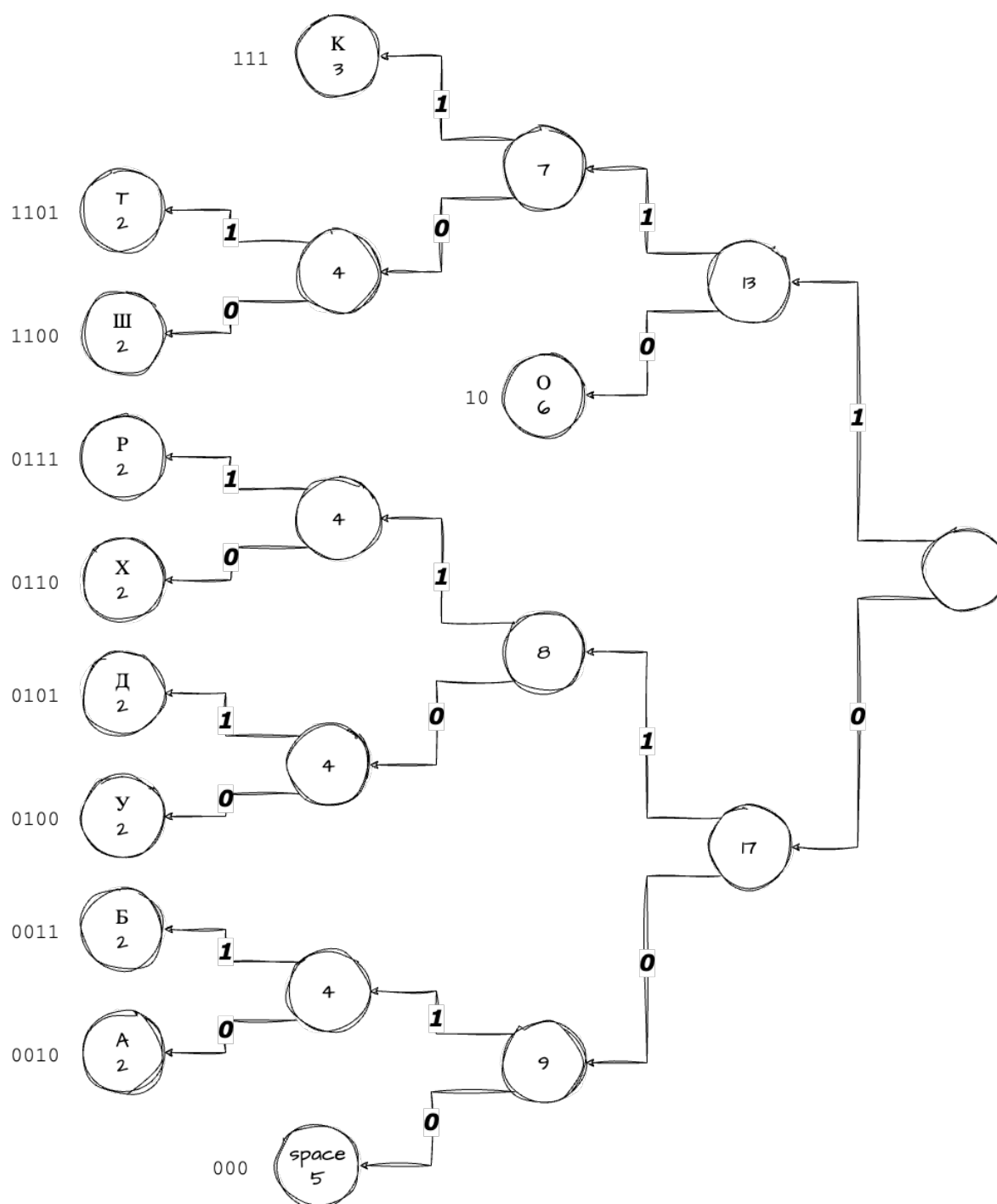


Рисунок 2.1 - Граф для метода Хаффмана

ИТОГОВЫЙ КОД:

$$\begin{array}{l} [1100\ 10\ 0111\ 10\ 0110]000[10\ 1101]000[0101\ 0100\ 0011\ 111\ 0010]000 \\ [111\ 0010\ 111]000[0011\ 0100\ 0101\ 1101\ 10]000[0110\ 10\ 0111\ 10\ 1100] \end{array}$$

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

2.5 Арифметическое кодирование

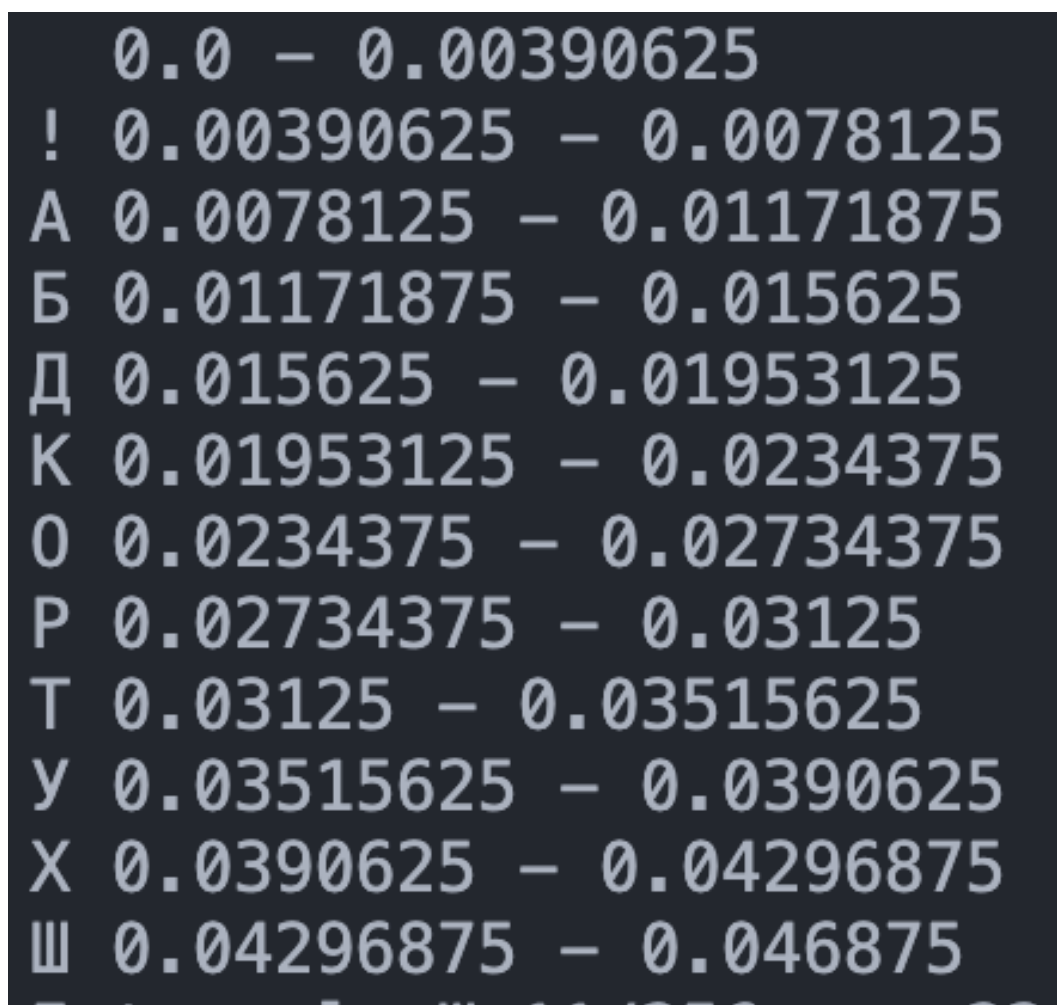


Рисунок 2.2 - Интервалы

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

```

Interval: Д 64457208709399212679377445006952347303738417289/1461501637330902918203684832716283019655932542976 -
257828834837596850717509780027809389230425327281/5846006549323611672814739330865132078623730171904
Interval: Т 8250522714803099222960312960889900454893989071117/187072209578355573530071658587684226515959365500928
- 66004181718424793783682503687119203639291157492061/1496577676626844588240573268701473812127674924007424
Interval: О 8448535259958373604311360471951258065811862543593183/191561942608236107294793378393788647952342390272
950272 - 16897070519916747208622720943902516131624699801648241/383123885216472214589586756787577295904684780545
900544
Interval: 8448535259958373604311360471951258065811862543593183/191561942608236107294793378393788647952342390272
950272 - 4325650053098687285407416561639044129695674597034171571/9807971461541688693493420973761978775159930381
9750539264
Interval: X 553683206796631972532149319889797648601046228530495150463/1255420347077336152767157884641533283220471
0888928069025792 - 1107366413593263945064298639779595297202092467782849381551/251084069415467230553431576928306
6566440942177856138051584
Interval: О 141742900939937784968230225891788198041867834535972335760403/3213876088517980551083924184682325205044
405987565585670602752 - 283485801879875569936460451783576396083735669146997685085181/64277521770359611021678483
69364650410088811975131171341205504
Interval: Р 72572365281248145903733875656595557397436331282943207004276961/16455045573212060421549691825573505049
82735865633579863348609024 - 72572365281248145903733875656595557397436331283543631112791961/1645504557321206042
154969182557350504982735865633579863348609024
Interval: 0 1161157844499970334459742010505528918358981300527316471109124501/263280729171392966744795069209176080
7972377385013727781357744384 - 2322315688999940668919484021011057836717962601055158313313199627/52656145834278
593348959013841835216159447547700274555627155488768
Interval: Ш 594512816383984811243387909378830806199798425869991812289916201387/1347997333357531989733350754350981
5336818572211270286240551805124608 - 594512816383984811243387909378830806199798425869998116743055608887/1347997
3333575319897333507543509815336818572211270286240551805124608
Interval: ! 38048820248575027919576826200245171596787099255679477562667921740643/86271829334882047342934448278462
8181556388621521298319395315527974912 - 38048820248575027919576826200245171596787099255679480714894491444393/86
2718293348820473429344482784628181556388621521298319395315527974912

Final interval: 0.04410341190387956094889038204515308212907445667441928865168295049168788056555075702493799459993
03363152647240064389130359306227794852805966939598545145125428666525999348211959113656154842474099719851210466003
976762294769287109375, 0.0441034119038795609488903820451530821290744566744192923055136319621444851745026939416073
76970436481928183263610316823751893910023646447565464431148008852525204896914136037521480319257953257167104155200
8318831212818622589111328125
Interval cut : 0.044103411903879560948890382045153082129074456674419288, 0.0441034119038795609488903820451530821
29074456674419292
Prefix : 0.0441034119038795609488903820451530821290744566744192
Initial data: 248
Coded data : 171

```

Рисунок 2.3 - Результат арифметического кодирования

Видно, что получившийся полуинтервал имеет начало 0.0441034119038795609488903820451530821290744566744192886516829504916878805655507570249379945999303363152647240064389130359306227794852805966939598545145125428666525999348211959113656154842474099719851210466003976762294769287109375 и конец 0.0441034119038795609488903820451530821290744566744192923055136319621444851745026939416073769704364819281832636103168237518939100236464475654644311480088525252048969141360375214803192579532571671041552008318831212818622589111328125.

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.

```

[Л: 198] [Н: 199] [Х: 200] [Н: 201]
[н: 202] [ч: 203] [ч: 204] [м: 205]
[м: 206] [л: 207] [А: 208] [а: 209]
[А: 210] [ä: 211] [Æ: 212] [æ: 213]
[Ė: 214] [ė: 215] [Ə: 216] [ə: 217]
[ë: 218] [ë: 219] [Ж: 220] [ж: 221]
[Э: 222] [э: 223] [З: 224] [з: 225]
[И: 226] [й: 227] [Й: 228] [й: 229]
[О: 230] [о: 231] [Ө: 232] [ө: 233]
[ö: 234] [ö: 235] [Э: 236] [э: 237]
[У: 238] [у: 239] [Ў: 240] [ў: 241]
[У: 242] [ў: 243] [Ч: 244] [ч: 245]
[Г: 246] [г: 247] [Ы: 248] [ы: 249]
[Ғ: 250] [ғ: 251] [Х: 252] [х: 253]
[Ж: 254] [ж: 255] [ШО: 246] [ОР: 247]
[РО: 248] [ОХ: 249] [X : 250] [ O: 251]
[ОТ: 252] [Т : 253] [ Д: 254] [ДУ: 255]
[УБ: 256] [БК: 257] [КА: 258] [А : 259]
[ К: 260] [КАК: 261] [К : 262] [ Б: 263]
[БУ: 264] [УД: 265] [ДТ: 266] [ТО: 267]
[О : 268] [ X: 269] [ХО: 270] [ОРО: 271]
[ОШ: 272]
Encoded data: [40, 30, 32, 30, 37, 6, 30, 34, 6, 20, 35, 17, 26, 16, 6, 258, 26, 6, 17, 35, 20, 34, 30, 6, 37, 2
47, 30, 40]
Initial data: 240
Coded data : 252

```

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, 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 import decimal as decimal
2 from fractions import Fraction
3 from collections import Counter
4 import math
5 import os
6
7
8 def arithmetic_encoding(input, bounds):
9     lower, upper = Fraction(0), Fraction(1)
10
11     for char in input:
12         length = upper - lower
13         lower += length * Fraction(bounds[char][0])
14         upper = lower + length * Fraction(bounds[char][1])
15         print(f"Interval: {char}", lower, " - ", upper)
16     return [lower, upper]
17
18
19 def decimal_from_fraction(frac):
20     return frac.numerator / decimal.Decimal(frac.denominator)
21
22
23 def get_accuracy(num1, num2):
24     length = len(num1)
25     match = False
26     for i in range(length):
27         if num1[i] == num2[i]:
28             continue
29         if (match):
30             return min(length, i+1)
31         match = True
32     raise ValueError("Increase decimal.getcontext().prec")
33
34
35 input = "ШОРОХ ОТ ДУБКА КАК БУДТО ХОРОШ!"
36 decimal.getcontext().prec = 500
37 coding = 256
38
39 bounds = dict()
```

```

40 chars = sorted(Counter(input).keys())
41 for i, char in enumerate(chars):
42     bounds[char] = (Fraction(i, coding), Fraction(i + 1,
        coding))
43
44 intervals = [f"{chars[i]} {float(bounds[chars[i]][0])} - {
    float(bounds[chars[i]][1])}" for i in range(len(chars))]
45 for interval in intervals:
46     print(interval)
47
48 result = arithmetic_encoding(input, bounds)
49 lower_bound = str(decimal_from_fraction(result[0]))
50 upper_bound = str(decimal_from_fraction(result[1]))
51 accuracy = get_accuracy(lower_bound, upper_bound)
52 prefix = os.path.commonprefix([lower_bound, upper_bound])
53 print()
54 print(f'Final interval: {lower_bound}, {upper_bound}')
55 print(f'Interval cut : {lower_bound[:accuracy]}, {
    upper_bound[:accuracy]}')
56 print(f'Prefix : {prefix}')
57 coded_num = decimal.Decimal(prefix[-1:1:-1])
58 print("Initial data: ", len(input) * math.ceil(math.log2(
    coding)))
59 print("Coded data : ", math.ceil(math.log2(coded_num)))

```

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 = "ШЮРОХ ОТ ДУБКА КАК БУДТО ХОРОШ"
24 alphabet = list(set(input))
25
26 dictionary = {alphabet[i]: i for i in range(len(alphabet))}
27 for i in range(len(dictionary), 256):
28     dictionary[chr(i + 1024)] = i
29
30 print(dictionary)
31
32 code, size_in_bits = lzw_encode(input, dictionary)
33
34 print("Encoded data: ", code)
35 print("Initial data: ", len(input) * 8)
36 print("Coded data : ", size_in_bits)

```

Листинг кодирования по словарю:

```

1 word = "ШЮРОХ 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)

```