Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
Национальный исследовательский университет ИТМО

МЕГАФАКУЛЬТЕТ ТРАНСЛЯЦИОННЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММИРОВАНИЯ

ЛАБОРАТОРНАЯ РАБОТА №2
По дисциплине «Введение в цифровую культуру и программирование»
Эффективное кодирование

Выполнил Фадеев Артём Владимирович
(Фамилия Имя Отчество)
Проверила ___________________________
(Фамилия Имя Отчество)

Санкт-Петербург, 2020 г.

ИЗОБРАЖЕНИЕ ДО/ПОСЛЕ ЗАДАННОГО ФОРМАТА

ЦИФРОВАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ

Цифровая последовательность 64 строки изображения до/после изменения

```
Before:
192  192  202  210  207  203  200  193  206  198  192  191
191  195  190  176  176  182  184  185  187  186  190  201
194  189  186  183  179  182  182  173  179  191  198  193
185  182  180  178  181  181  184  187  196  153  173  253
250  251  253  253  255  242  252  255  249  253  250  255
253  210  217  254  240  193  233  255  208  186  185  249
250  248  249  250  254  222  192  247  250  254  207  137
189  213  187   74  142  156   44   38   44  124  162  184
184  196  191  204  211  198  198  198  203  214  211  207
213  213  210  216  227  219  209  206  213  220  220  215
205  204  196  204  222  218  213  228

After:
180  180  200  200  200  200  200  180  200  180  180  180
180  180  180  160  160  180  180  180  180  180  180  200
180  180  180  180  160  180  180  160  160  180  180  180
180  180  180  160  180  180  180  180  180  140  160  240
240  240  240  240  240  240  240  240  240  240  240  240
240  200  200  240  240  180  220  240  200  180  180  240
240  240  240  240  240  220  180  240  240  240  200  120
180  200  180   60  140  140   40   20   40  120  160  180
180  180  180  200  200  180  180  180  200  200  200  200
200  200  200  200  220  200  200  200  200  220  220  200
200  200  180  200  220  200  200  220
```

ПРОЦЕСС ПОЛУЧЕНИЯ

Для получения цифровой последовательность пикселей 64 строки моего изображения я написал программу на языке Python. Для начала необходимо подключить библиотеки:

1. numpy – позволяет работать с массивами
2. matplotlib – визуализация изображений
3. scikit-image – работа с изображениями

Далее приведу код программы:

```
# to see my image
from matplotlib import pyplot as plt
# to download, show, save my image
from skimage.io import imread, imshow, imsave
# my image
img = imread('kek_black.jpg')
# show me my image
plt.imshow(img)
plt.show()

# pixels from 64 line
p = []

for i in range(63, 64):
    for j in range(0, 128):
        p.append(img[i][j])

print('\n', "Before: ")

for i in range(0, len(p)):
    print("%4d" % p[i], end=' ')
    if i % 12 == 11:
        print()

# here we will write the probably 0...240
total = []
for i in (range(0, 256 // 20 + 1)):
    total.append(0)

print('\n\n', "After: ")

for i in range(0, len(p)):
    p[i] = round(p[i] // 20) * 20
    print("%4d" % p[i], end=' ')
    total[p[i] // 20] += 1
    if i % 12 == 11 :
        print()

print('\n')
```

РАСЧЕТ ЧАСТОТЫ ВСТРЕЧАЕМОСТИ И ЭНТРОПИИ

Аналогично воспользуемся программой:

```
# here we will save count of numbers and round pixel
pairs = []
for i in range(0, len(total)):
    pairs.append((total[i], 20 * i))

print('\n', '|', end="")
# sort
pairs.sort(reverse=True)
# print
for i in range(0, len(pairs)):
    print("%4d" % pairs[i][1], '|', end=' ')
print('\n', '|', end="")

for i in range(0, len(pairs)):
    print("%4d" % pairs[i][0], '|', end=' ')
print('\n', '|', end="")

for i in range(0, len(pairs)):
    print("%4.2f" % float(pairs[i][0] / 128), '|', end=' ')
print()
```

Получим результат:

180	200	240	160	220	140	120	40	60	20	100	80	0
46	32	26	8	7	3	2	2	1	1	0	0	0
0.36	0.25	0.20	0.06	0.05	0.02	0.02	0.02	0.01	0.01	0.00	0.00	0.00

В первой строке находятся наши округлённые пиксели, во второй количество встречи пикселя в строке и его вероятность соответственно. Пиксели со значениями 100, 80, 0 не являются уникальными, поэтому **количество символов алфавита** = **10**. Соответственно **минимальная длина двоичного кода** = log2(10) = **4**.

Рассчитаем энтропию:

```
# ans - my entropy = -∑(i...n) = p[i]*log2(p[i])
ans = 0.00
# import math in the top of the code
for i in range(0, len(pairs)):
    if pairs[i][0] != 0:
        ans -= float(pairs[i][0] / 128) * (math.log2(float(pairs[i][0] / 128)))
print('\n', "%.4f" % ans)
```

Итоговое значение энтропии:

```
2.4008
```

ДВОИЧНЫЙ РАВНОМЕРНЫЙ КОД

180	200	240	160	220	140	120	40	60	20
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Полученная в результате двоичного равномерного кодирования последовательность:

```
0000 0000 0001 0001 0001 0001 0001 0000 0001 0000 0000 0000
0000 0000 0000 0011 0011 0000 0000 0000 0000 0000 0000 0001
0000 0000 0000 0000 0011 0000 0000 0011 0011 0000 0000 0000
0000 0000 0000 0011 0000 0000 0000 0000 0000 0101 0011 0010
0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010
0010 0001 0001 0010 0010 0000 0100 0010 0001 0000 0000 0010
0010 0010 0010 0010 0010 0100 0000 0010 0010 0010 0001 0110
0000 0001 0000 1000 0101 0101 0111 1001 0111 0110 0011 0000
0000 0000 0000 0001 0001 0000 0000 0000 0001 0001 0001 0001
0001 0001 0001 0001 0100 0001 0001 0001 0001 0100 0100 0001
0001 0001 0000 0001 0100 0001 0001 0100
```

Код:

```python
# bu has binary codes for numbers
bu = ['----', '1001', '0111', '1000', '----', '----', '0110', '0101', '0011', '0000', '0001', '0100', '0010']
for i in range(0, len(p)):
    print("%4s" % bu[p[i] // 20], end=' ')
    if i % 12 == 11:
        print()
```

Длина двоичного кода = 4 бит

Количество переданной информации = 4 * 128 = 512 бит

КОДЫ ШЕННОНА-ФАНО

180	200	240	160	220	140	120	40	60	20
46	32	26	8	7	3	2	2	1	1
0.36	0.25	0.20	0.06	0.05	0.02	0.02	0.02	0.01	0.01

Присвоим вероятностям буквы от A до J по убыванию вероятности встречи элементов в множестве. И будем делить множество на две части, суммарные вероятности которых максимально близки.

<table>
<tr><td>A</td><td rowspan="2">1</td><td colspan="5">11</td></tr>
<tr><td>B</td><td colspan="5">10</td></tr>
<tr><td>C</td><td rowspan="8">0</td><td colspan="5">01</td></tr>
<tr><td>D</td><td rowspan="7">00</td><td rowspan="2">001</td><td colspan="3">0010</td></tr>
<tr><td>E</td><td colspan="3">0011</td></tr>
<tr><td>F</td><td rowspan="5">000</td><td rowspan="2">0001</td><td colspan="2">00011</td></tr>
<tr><td>G</td><td colspan="2">00010</td></tr>
<tr><td>H</td><td rowspan="3">0000</td><td colspan="2">00001</td></tr>
<tr><td>I</td><td rowspan="2">00000</td><td>000001</td></tr>
<tr><td>J</td><td>000000</td></tr>
</table>

Закодированная кодом Шеннона-Фано последовательность:

11	11	10	10	10	10	10	11	10	11	11	11
11	11	11	0010	0010	11	11	11	11	11	11	10
11	11	11	11	0010	11	11	0010	0010	11	11	11
11	11	11	0010	11	11	11	11	11	00011	0010	01
01	01	01	01	01	01	01	01	01	01	01	01
01	10	10	01	01	11	0011	01	10	11	11	01
01	01	01	01	01	0011	11	01	01	01	10	00010
11	10	11	000001	00011	00011	00001	000000	00001	00010	0010	11
11	11	11	10	10	11	11	11	10	10	10	10
10	10	10	10	0011	10	10	10	10	0011	0011	10
10	10	11	10	0011	10	10	0011				

Код:

```
# CODE SHEN-FANO
fano = ['----', '000000', '00001', '000001', '----', '----', '00010', '00011', '0010', '11', '10', '0011', '01']
for i in range(0, len(p)):
    print("%7s" % fano[p[i] // 20], end=' ')
    if i % 12 == 11:
        print()
print()
```

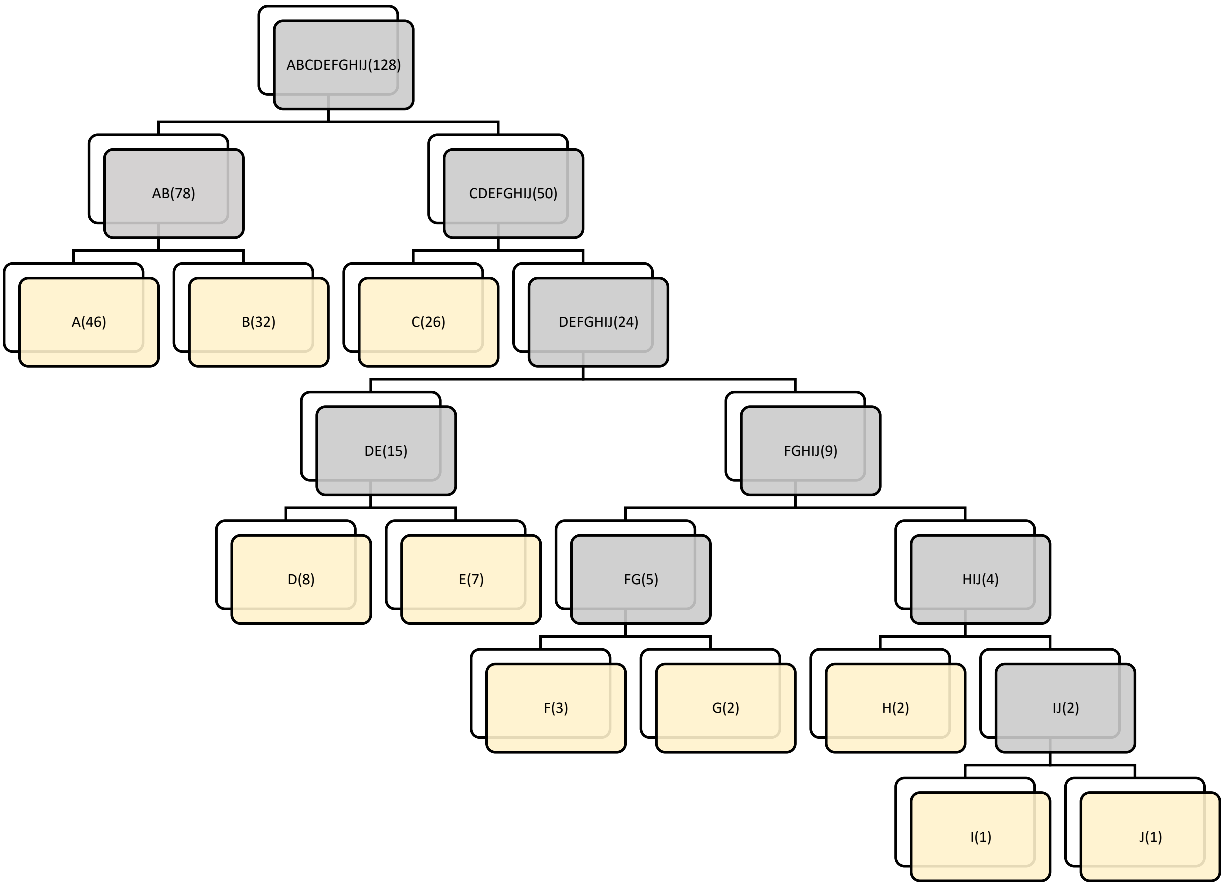
Средняя длина кодовой комбинации:

$$I_{\text{ср}} = \frac{2*(46+32+26)+4*(8+7)+5*(3+2+2)+6*(1+1)}{128} = \frac{315}{128} \approx 2.4609$$

Относительная избыточность кода: $Q = 1 - \frac{2.4008}{2.4609} \approx 0.024$

Количество переданной информации = 315 бит

КОД ХАФФМАНА



A	B	C	D	E	F	G	H	I	J
11	10	01	0011	0010	00011	00010	00001	000001	000000

Закодированная кодом Хаффмана последовательность:

11	11	10	10	10	10	10	11	10	11	11	11
11	11	11	0011	0011	11	11	11	11	11	11	10
11	11	11	11	0011	11	11	0011	0011	11	11	11
11	11	11	0011	11	11	11	11	11	00011	0011	01
01	01	01	01	01	01	01	01	01	01	01	01
01	10	10	01	01	11	0010	01	10	11	11	01
01	01	01	01	01	0010	11	01	01	01	10	00010
11	10	11	000001	00011	00011	00001	000000	00001	00010	0011	11
11	11	11	10	10	11	11	11	10	10	10	10
10	10	10	10	0010	10	10	10	10	0010	0010	10
10	10	11	10	0010	10	10	0010				

Эффективное кодирование

Код:

```
# CODE HAFF
haf = ['----', '000000', '00001', '000001', '----', '----', '00010', '00011', '0011', '11', '10', '0010', '01']
for i in range(0, len(p)):
    print("%8s" % haf[p[i] // 20], end=' ')
    if i % 12 == 11:
        print()
print()
```

Средняя длина кодовой комбинации:

$$I_{\text{ср}} = \frac{2*(46+32+26)+4*(8+7)+5*(3+2+2)+6*(1+1)}{128} = \frac{315}{128} \approx 2.4609$$

Относительная избыточность кода:

$$Q = 1 - \frac{2.4008}{2.4609} \approx 0.024$$

Количество переданной информации = 315 бит

ОЦЕНКА СТЕПЕНИ СЖАТИЯ И ВЫВОД

При равномерном двоичном кодировании количество переданной информации - 512 бит.

При использовании метода кодирования Шеннона-Фано – 315 бит

При использовании метода кодирования Хаффмана – 315 бит

Степени сжатия соответственно:

$$k = \frac{315}{512} = 0.6152$$

$$k = \frac{315}{512} = 0.6152$$

Оба метода оказались наиболее эффективными, чем метод равномерного кодирования информации. Из-за того, что алгоритмы используют переменную длину при кодировании: часто встречающиеся символы получают кодовое слово меньшей длины, а редко встречающийся – наибольшей.