МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

**Лабораторная работа №1**

**По дисциплине «Введение в цифровую культуру и программирование»**

**Эффективное кодирование**

Выполнил студент группы: №М3110

Юрченко Владислав Витальевич

Проверил:

Хлопотов Максим Валерьевич

***САНКТ-ПЕТЕРБУРГ***

***2019***

Задание № 2

Изображение выглядит как человек, носит, одежда, женщина

Автоматически созданное описаниеИсходное изображение

Исходная строка:

160 160 160 160 180 160 100 80 60 80 100 120 140 140 140 140 140 120 120 120 160 180 180 180 160 160 160 160 160 180 180 180 200 200 200 200 200 200 200 200 200 220 220 220 220 220 220 220 200 200 200 200 200 160 140 120 120 120 120 140 140 160 160 160 160 180 180 180 180 180 180 180 180 180 180 180 180 200 200 200 200 200 200 200 120 20 20 20 20 20 20 20 20 20 60 80 80 100 120 140 160 180 200 180 180 180 180 180 160 160 160 140 160 140 140 120 160 180 180 180 180 180 180 180 180 180 180 180

Задание № 3

from PIL import ImageDraw, Image  
import math  
  
image = Image.open('test.bmp') # Открываем изображение  
draw = ImageDraw.Draw(image) # Объявляем переменную для создания нового изображения  
width = image.size[0] # Ширина  
height = image.size[1] # Высота  
pix = image.load() # Значения пикселей  
  
# Округляем среднюю строку  
unique = set() # Задаем множество уникальных значений  
count = [0]\*14 # Счетчик округленных значений  
middle\_w = width//2  
for x in range(width):  
 grey = pix[x, middle\_w][0]  
 grey = round(grey / 20)  
 count[grey] += 1  
 grey \*= 20  
 unique.add(grey)  
 draw.point((x, middle\_w), (grey, grey, grey)) # Изменяем пиксель для финального изображения  
 print(grey, end = ‘ ‘) # Выводим значение

# Вычисляем энтропию  
entropy = 0  
for i in range(0, 14):  
 if count[i] != 0:  
 p = count[i] / 128  
 entropy += p \* math.log(p, 2)  
entropy \*= -1  
  
image.save("result.bmp", "bmp") # Сохраняем изображение

**Задание № 4**

|  |  |  |
| --- | --- | --- |
| Значение | количество | вероятность |
| 180 | 36 | 0,281 |
| 200 | 22 | 0,172 |
| 160 | 22 | 0,172 |
| 140 | 12 | 0,094 |
| 120 | 11 | 0,086 |
| 20 | 9 | 0,07 |
| 220 | 7 | 0,055 |
| 80 | 4 | 0,031 |
| 100 | 3 | 0,023 |
| 60 | 2 | 0,016 |
| 260 | 0 | 0 |
| 240 | 0 | 0 |
| 40 | 0 | 0 |
| 0 | 0 | 0 |

Количество символов в алфавите: 10

Энтропия: 2,887958702693938

Расчетная длина двоичного кода: 4

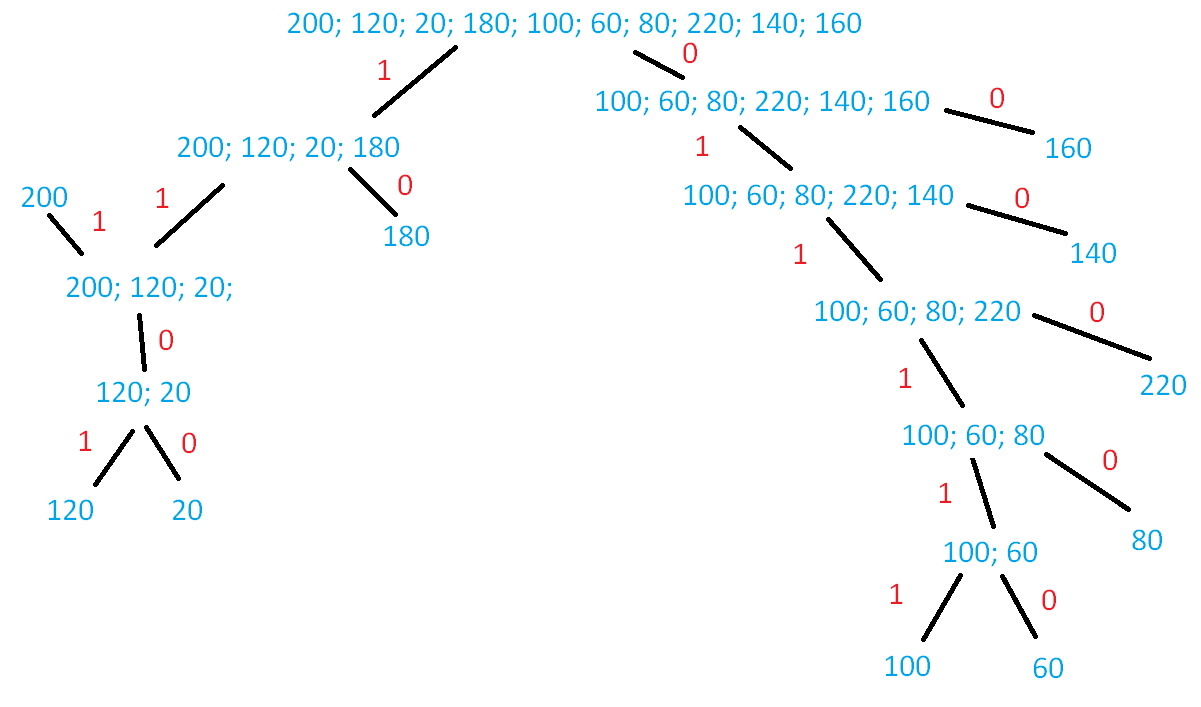
**Задание № 5**

Коды Шеннона–Фано

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Число** | **20** | **60** | **80** | **100** | **120** | **140** | **160** | **180** | **200** | **220** |
| **Код** | **1010** | **11010** | **11011** | **1011** | **1100** | **1110** | **100** | **01** | **00** | **1111** |

**Задание № 6**

Дерево Хаффмана



**Задание № 7**

Коды Хаффмана

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Число** | **20** | **60** | **80** | **100** | **120** | **140** | **160** | **180** | **200** | **220** |
| **Код** | **0011** | **011110** | **01110** | **111110** | **1011** | **010** | **00** | **01** | **111** | **0110** |

**Задание № 8**

Равномерное кодирование

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Число** | **20** | **60** | **80** | **100** | **120** | **140** | **160** | **180** | **200** | **220** |
| **Код** | **0000** | **0001** | **0010** | **0011** | **0100** | **0101** | **0110** | **0111** | **1000** | **1001** |

Закодированное сообщение равномерным кодированием:

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

Длина кодового слова: 4 бит

Количество переданной информации при передаче сообщения: 512 бит

**Задание № 9**

Закодированное сообщение кодом Шеннона–Фано:

100 100 100 100 01 100 1011 11011 11010 11011 1011 1100 1110 1110 1110 1110 1110 1100 1100 1100 100 01 01 01 100 100 100 100 100 01 01 01 00 00 00 00 00 00 00 00 00 1111 1111 1111 1111 1111 1111 1111 00 00 00 00 00 100 1110 1100 1100 1100 1100 1110 1110 100 100 100 100 01 01 01 01 01 01 01 01 01 01 01 01 00 00 00 00 00 00 00 1100 1010 1010 1010 1010 1010 1010 1010 1010 1010 11010 11011 11011 1011 1100 1110 100 01 00 01 01 01 01 01 100 100 100 1110 100 1110 1110 1100 100 01 01 01 01 01 01 01 01 01 01 01 Средняя длина кодового слова: 3,2890625 бит

Количество переданной информации при передаче сообщения: 380 бит

**Задание № 10**

Закодированное сообщение кодом Хаффмана:

00 00 00 00 01 00 111110 01110 011110 01110 111110 1011 010 010 010 010 010 1011 1011 1011 00 01 01 01 00 00 00 00 00 01 01 01 111 111 111 111 111 111 111 111 111 0110 0110 0110 0110 0110 0110 0110 111 111 111 111 111 00 010 1011 1011 1011 1011 010 010 00 00 00 00 01 01 01 01 01 01 01 01 01 01 01 01 111 111 111 111 111 111 111 1011 0011 0011 0011 0011 0011 0011 0011 0011 0011 011110 01110 01110 111110 1011 010 00 01 111 01 01 01 01 01 00 00 00 010 00 010 010 1011 00 01 01 01 01 01 01 01 01 01 01 01

Средняя длина кодового слова: 2,8828125 бит

Количество переданной информации при передаче сообщения: 376 бит

**Задание № 11**

**Кодировка Шеннона-Фано:**

Степень сжатия: 512/421 = 1,216152019002375

Избыточность: 1- 2,887958702693938 / 3,2890625 = 0,1219507982308217

**Кодировка Хаффмана:**

Степень сжатия: 512/376 = 1,361702127659574

Избыточность: 1- 2,887958702693938 / 2.9375= 0,01686512248717

**Задание № 12**

**Вывод: Код Хаффмана сжимает информацию эффективнее или хотя бы так же эффективно как код Шеннона-Фано.**