МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ 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 №2**

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

**Эффективное кодирование**

Выполнил студент группы М3103:

Гаврилов Алексей Олегович

Проверил:

Хлопотов Максим Валерьевич

САНКТ-ПЕТЕРБУРГ

2019

**Исходная картинка Картинка в оттенках серого**



**Код квантования и перевода в градиент черного на Python:**

**import** numpy **as** np  
**import** cv2  
img = cv2.imread(**"key-flat-128x128.png"**,0) # чтение в оттенках серого  
**for** i **in** range(128):  
 **for** j **in** range(128):  
 img[i][j] = ( img[i][j] // 20 ) \* 20 # квантование строк  
cv2.imwrite(**'key-flag-black.png'**, img)

**Код подсчета частоты символов:**

line = img[64]  
chars = set()  
chars\_count={}  
**for** i **in** range(len(line)):  
 **if**(line[i] **in** chars):  
 chars\_count[line[i]]+=1  
 **else**:  
 chars\_count[line[i]] = 1  
 chars.add(line[i])

**Первичный алфавит**

|  |  |  |
| --- | --- | --- |
| **Символ** | **Количество символов** | **Вероятность появления** |
| **200** | **1** | **0.0078125** |
| **100** | **1** | **0.0078125** |
| **160** | **1** | **0.0078125** |
| **220** | **4** | **0.03125** |
| **180** | **38** | **0.296875** |
| **80** | **83** | **0.6484375** |

**Информационная энтропия:**

H=, где p(i) – вероятность i-ого символа

Энтропия: 1.245706888182414

Количество элементов: 6

**Коды Шеннона-Фано:**

|  |  |
| --- | --- |
| **Символ** | **Код** |
| **80** | **0** |
| **180** | **11** |
| **220** | **101** |
| **100** | **1001** |
| **200** | **10000** |
| **160** | **10001** |

**Код алгоритма Шеннона-Фано на Python:**

result=[]  
  
**def** sf(a,code):  
 p1=[]  
 p1\_probability=0  
 p2=[]  
 p2\_probability=0  
 a2=[]  
 a2.extend(a)  
 **while**(len(a2)!=0):  
 **if**(p1\_probability<=p2\_probability):  
 p1.append(a2.pop(0))  
 p1\_probability+=p1[len(p1)-1][1]  
 **else**:  
 p2.append(a2.pop())  
 p2\_probability+=p2[len(p2)-1][1]  
 **if**(len(p1)>1):  
 sf(p1,code+**'0'**)  
 **else**:  
 result.append([p1[0][0],code+**'0'**])  
 **if**(len(p2)>1):  
 sf(p2,code+**'1'**)  
 **else**:  
 result.append([p2[0][0],code+**'1'**])  
array\_chars=[]  
probability = 0  
  
**for** char **in** chars\_count.keys():  
 probability+=chars\_count[char]  
**for** char **in** chars\_count.keys():  
 chars\_count[char]/=probability  
**for** k,i **in** chars\_count.items():  
 array\_chars.append([k,i])  
array\_chars.sort(key = **lambda** char:char[1], reverse =**True** )  
sf(array\_chars,**''**)  
  
print(result)

**Коды Хаффмана для символов:**

|  |  |
| --- | --- |
| **Символ** | **Код** |
| **80** | **0** |
| **180** | **10** |
| **220** | **110** |
| **160** | **11100** |
| **100** | **11101** |
| **200** | **1111** |

**Кодовое дерево Хаффмана:**

**Код алгоритма Хаффмана на Python:**

**class** charcode(object):*#Класс для каждого различного символа, в котором хранится инфа о нем* frequency=0 *#Частота встречаемости в коде* char\_=**''***#Сам символ* char\_byte=**''***#Его двоичный код* left=**''***#Левый лист при наличии* right=**''***#Правый лист при наличии*objs={}*#Словарь символов  
#---------------------------Записываем элементы из массива в массив объектов-символов***for** xkey,xvalue **in** chars\_count.items():  
 tmp=charcode()  
 tmp.frequency = int(xvalue)  
 tmp.char=xkey  
 objs[xkey] = tmp  
  
*#-----------Дублируем словарь, чтобы был массив для восстановления кода в обратном порядке*objtmp={}  
objtmp = objs.copy()  
*#Идем по графу от листьев к вершинам, записываем значение вершинам***while**(len(objs)>1):  
 *#Находим правый лист* min1=**''  
 for** x **in** objs:  
 **if**(min1 ==**''**): min1=x  
 **elif**(objs[x].frequency<objs[min1].frequency): min1=x  
 r=objs.pop(min1)  
  
 *#Находим левый лист* min2=**''  
 for** x **in** objs:  
 **if**(min2 ==**''**): min2=x  
 **elif**(objs[x].frequency<objs[min2].frequency): min2=x  
 l=objs.pop(min2)  
 *#Добавляем вершину в массивы* z=charcode()  
  
 z.frequency = int(l.frequency)+int(r.frequency)  
 z.left=l.char  
 z.right = r.char  
 z.char=str(l.char)+str(r.char)  
  
 objs[z.char] = z  
 objtmp[z.char] = z  
**for** x **in** objs:  
 print(objs[x].char,objs[x].frequency,objs[x].left,objs[x].right)  
 answers[xk]= xv  
**for** xv **in** answers.keys():  
 print(answers[xv].char, answers[xv].char\_byte,answers[xv].frequency)

*#--------------------Восстановление листьев с кодами-------------------*answers ={}*#Массив конечных листьев***while**(len(objs)>0):  
 xk,xv=objs.popitem()  
 **if**(xv.left!=**''**):  
 tmp=charcode()  
 tmp.char\_byte=xv.char\_byte+**'0'** tmp.char = xv.left  
 tmp.left = objtmp[xv.left].left  
 tmp.right = objtmp[xv.left].right  
 tmp.frequency = objtmp[xv.left].frequency  
 objs[tmp.char] = tmp  
 *# print(tmp.)* **if**(xv.right!=**''**):  
 tmp=charcode()  
 tmp.char\_byte=xv.char\_byte+**'1'** tmp.char = xv.right  
 tmp.left = objtmp[xv.right].left  
 tmp.right = objtmp[xv.right].right  
 tmp.frequency = objtmp[xv.right].frequency  
 objs[tmp.char] =tmp  
 **if**(xv.right==**'' and** xv.left ==**''**):

**Закодированное равномерным кодом сообщение, длина кодового слова, количество переданной информации при передаче сообщения:**

00110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100010000010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010000100101010101010101010000110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011001100110011

Длина закодированного слова: 4.0

Размер переданного сообщения: 64 байта

**Закодированная полученным кодом Шеннона–Фано последовательность, длина кодового слова, количество переданной информации при передаче сообщения:**

00000000000000000000000000000010011000111111111111111111111111111111111111111111111111111111111111111111111111111100001011011011011100000000000000000000000000000000000000000000000000000

Средняя длина закодированного слова: 1.4453125

Размер переданного сообщения: 24 байта

**Закодированная полученным кодом Хаффмана последовательность, длина кодового слова, количество переданной информации при передаче сообщения:**

00000000000000000000000000000011101111001010101010101010101010101010101010101010101010101010101010101010101010101011111101101101101000000000000000000000000000000000000000000000000000000

Средняя длина закодированного слова: 1.4453125

Размер переданного сообщения: 24 байта

**Расчеты по коду Шенона-Фано и Хаффмана:**

Бинарных символов: 185

Символов в строке: 128

Средняя длина закодированного слова: 185/128 = 1.4453125

**Вывод:**   
  
Степень сжатия сообщения(Хаффман): Q1 = 0.36132813  
Q1 = m1/n, где n - число знаков в исходном сообщении, которое закодировано равномерным кодом, m1 - число знаков в исходном сообщении, которое закодировано кодом Хаффмана  
  
Степень сжатия сообщения(Шеннон): Q2 =0.36132813  
Q2 = m2/n, где n - число знаков в исходном сообщении, которое закодировано равномерным кодом, m2 - число знаков в исходном сообщении, которое закодировано кодом Шеннона-Фано  
  
Средняя длина кода Хаффмана: 1.4453125  
Средняя длина кода Шеннона: 1.4453125