Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных средств

**СИСТЕМЫ ОБРАБОТКИ МЕДИАДАННЫХ**

Лабораторная работа № 6

Коды Хаффмана

Вариант № 14

группа № 850702

| Выполнил: | Турко В.Д. |
| --- | --- |
| Проверил | Рыбенков Е.В. |

Минск 2021

**1 ЦЕЛЬ РАБОТЫ**

Использование кодов Хаффмана для кодирования с минимальной избыточностью.

**2 ЗАДАНИЕ**

1. Построить и вывести гистограмму изображения.
2. Рассчитать энтропию источника.
3. Выполнить кодирование изображения с помощью алгоритма Хаффмана (реализация MATLAB и собственная реализация).
4. Рассчитать среднюю длину кода, выполнить декодирование и рассчитать коэффициент сжатия для обеих реализаций.

**3 ХОД РАБОТЫ**

Для выполнения лабораторной работы был использован язык программирования Python (v3.10) и изображение, показанное на рисунке 1.

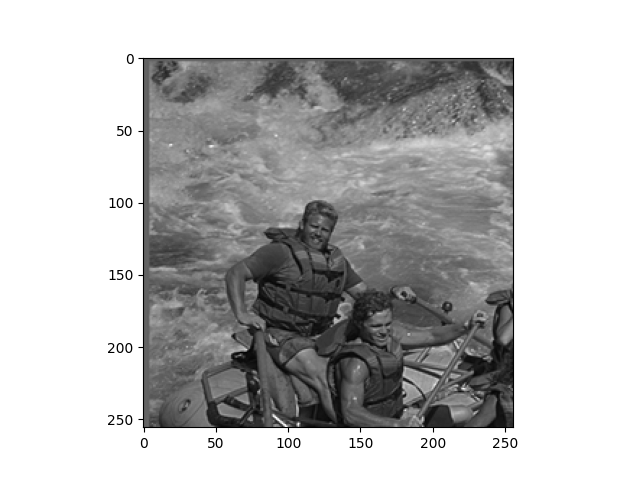


Рисунок 1 - Исходное изображение

**3.1 Вспомогательные функции**

Функции для расчета гистограммы, энтропии, средней длины кода и коэффициента сжатия.

def count\_variances(img):

variances = {}

counts\_dict = {}

values, counts = np.unique(img, return\_counts=True)

for index, value in np.ndenumerate(values):

counts\_dict[value] = counts[index]

variances[value] = counts[index] / img.size

return variances, counts\_dict

def histogram(variances):

hist = np.zeros(256)

for k, v in variances.items():

hist[k] = v

return hist

def entropy(variances: list):

ent = 0

for val in variances:

ent += val \* log2(val)

return -ent

def avg\_length(variances: dict, huffman: dict):

avg\_len = 0

for k, v in variances.items():

avg\_len += v \* len(huffman[k])

return avg\_len

def compression(encoded\_img: np.ndarray, huff\_dict: dict):

encoded = 0

for \_, val in np.ndenumerate(encoded\_img):

encoded += len(val)

for k, v in huff\_dict.items():

encoded += 8 + len(v)

return (encoded\_img.size \* 8 / encoded) \* 100

**3.2 Функции, реализующие алгоритм Хаффмана**

def huffmandict(counts\_dict: dict):

from heapq import heappush, heappop, heapify

from collections import defaultdict

code = defaultdict(list)

heap = [(count, [value]) for value, count in counts\_dict.items()]

while len(heap) > 1:

freq0, letters0 = heappop(heap)

for ltr in letters0:

code[ltr].insert(0, '0')

freq1, letters1 = heappop(heap)

for ltr in letters1:

code[ltr].insert(0, '1')

heappush(heap, (freq0+freq1, letters0+letters1))

for k, v in code.items():

code[k] = ''.join(v)

return dict(sorted(code.items(), key=lambda item: int(item[1], 2)))

def huffmanenco(img, huff\_dict):

result = np.empty(img.shape, dtype=object)

for coord, val in np.ndenumerate(img):

result[coord[0]][coord[1]] = huff\_dict[val]

return result

def huffmandeco(encoded\_img: np.ndarray, huff\_dict: dict):

result = np.zeros(encoded\_img.shape, dtype='int')

reverse\_huff = dict((v,k) for k,v in huff\_dict.items())

for coord, val in np.ndenumerate(encoded\_img):

result[coord[0]][coord[1]] = reverse\_huff[val]

return result

**3.3 Результат работы программы**

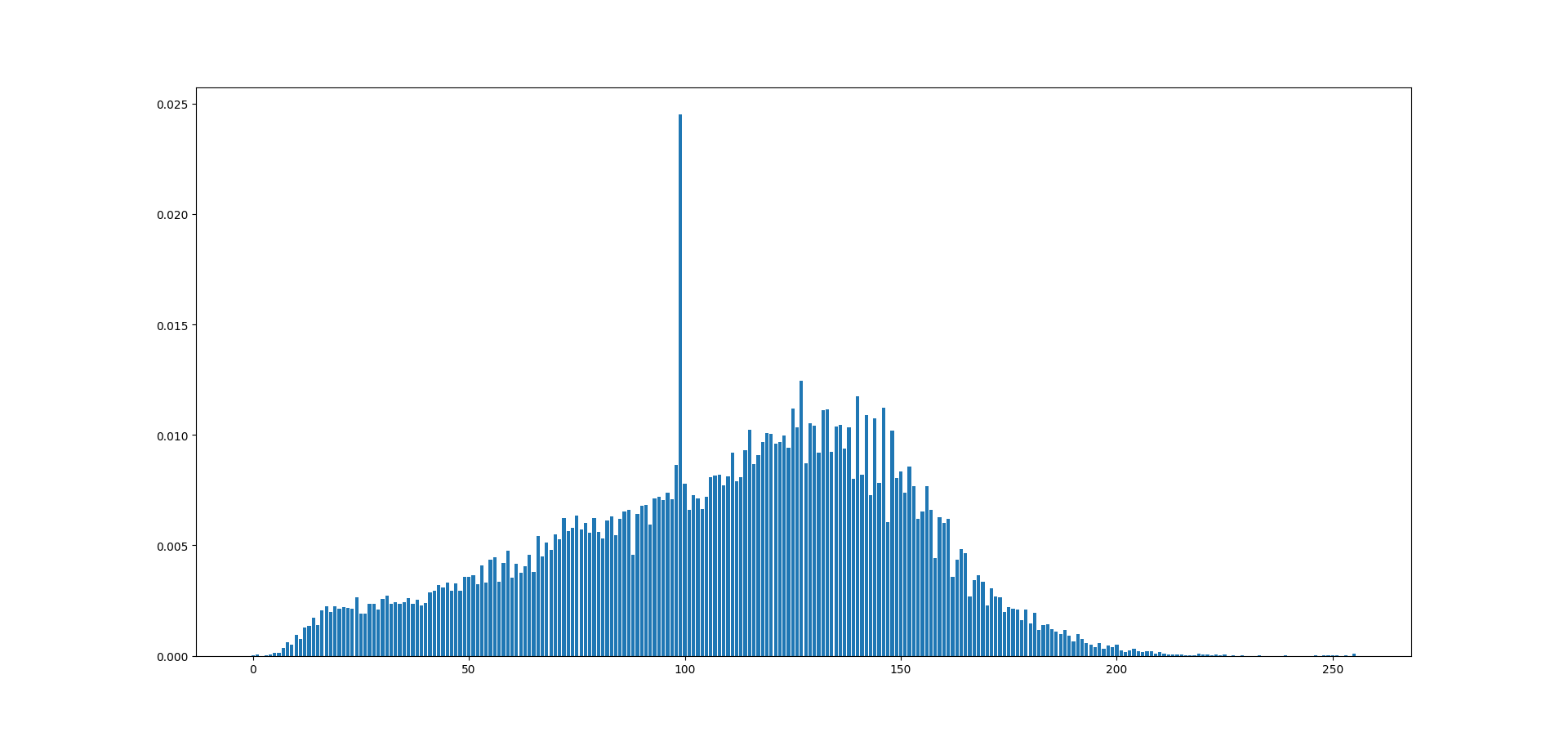


Рисунок 2 - Гистограмма

Entropy: 7.300272174244419

Self-writtem huffdict

Avg length: 7.3528900146484375

Compressed: 107.89994690287341%

Package huffdict

Avg length: 7.3315887451171875

Compressed: 108.20704073903613%

**4 ВЫВОД:**

В результате выполнения лабораторной был реализован алгоритм кодирования Хаффмана, который дал схожий результат с реализацией из пакета huffman для Python.