Содержание

1 Постановка задачи

Цель исследования: изучение способов представления изображений, ознакомление со структурой формата BMP, анализ статистических свойств изображений, а также получение практических навыков обработки изображений.

2 Структура формата RGB24

Структура BMP файла:

1. Заголовок.
2. Палитра — таблица цветов (отсутствует для RGB24).
3. Данные по пикселям.

Заголовок состоит из двух последовательно расположенных частей, которые описываются структурами:

```
typedef struct tagBITMAPFILEHEADER {
    WORD    bfType;
    DWORD   bfSize;
    WORD    bfReserved1;
    WORD    bfReserved2;
    DWORD   bfOffBits;
} BITMAPFILEHEADER, *PBITMAPFILEHEADER;

typedef struct tagBITMAPINFOHEADER {
    DWORD   biSize;
    LONG    biWidth;
    LONG    biHeight;
    WORD    biPlanes;
    WORD    biBitCount;
    DWORD   biCompression;
    DWORD   biSizeImage;
    LONG    biXPelsPerMeter;
    LONG    biYPelsPerMeter;
    DWORD   biClrUsed;
    DWORD   biClrImportant;
} BITMAPINFOHEADER, *PBITMAPINFOHEADER;
```

При выполнении исследования использовались поля:

- `bfOffBits` — смещение от начала `BITMAPFILEHEADER` до массива пикселей.
- `biWidth` — ширина изображения в пикселях.
- `biHeight` — высота изображения в пикселях. Если отрицательна, то строки изображения идут сверху вниз.

Особенности формата RGB24:

- Каждый пиксель представляется тремя байтами — B, G, и R соответственно.
- Данные представляются в виде одномерного массива, в котором значения, относящиеся к отдельным пикселям, записаны строка за строкой.
- Строки могут идти как снизу вверх, так и сверху вниз.
- Ширина строки в *байтах* должна быть выравнена по границе двойного слова (32 бита), т.е. должна быть кратна четырем. При необходимости в конце каждой

строки добавляются дополнительные байты (от одного до трех), значения которых несущественны (dummy bytes).

3 Алгоритм чтения/записи файла RGB24

При работе с файлами RGB24 использовалась следующая структура:

```
type Image struct {
    FileHeader BitmapFileHeader
    InfoHeader BitmapInfoHeader
    PixelData  []byte
}
```

Данная структура объединяет в себе заголовки и данные о пикселях изображения. При чтении изображения считываются заголовки, затем данные о пикселях в виде одномерного массива байт, при этом учитываются *dummy bytes*. Запись происходит аналогично.

4 Описание реализованных алгоритмов

1. Пиковое отношение сигнал шум:

$$PSNR = 10 \lg \frac{WH(2^L - 1)^2}{\sum_{i=1}^{H} \sum_{j=1}^{W} \left(I_{i,j}^{(A)} - \hat{I}_{i,j}^{(A)} \right)^2}, \tag{4.1}$$

где W — ширина изображения, H — высота изображения, L — количество бит в используемой разрядной сетке, $I_{y,x}^{(A)}$ — интенсивность пикселя на позиции (y, x), $\hat{I}_{y,x}^{(A)}$ — интенсивность пикселя восстановленного изображения на той же позиции.

2. Оценка энтропии при поэлементном независимом сжатии:

$$\hat{H}(X) = -\sum_{x} \hat{p}(x) \log_2 \hat{p}(x), \tag{4.2}$$

где $\hat{p}(x) = \frac{n_x}{n}$ — оценка вероятности сообщения x (n — общее число пикселей).

3. Преобразование цветового пространства:

   3.1. $RGB \to YC_bC_r$:

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B; \\ C_b &= 0.5643(B - Y) + 128; \\ C_r &= 0.7132(R - Y) + 128. \end{aligned} \tag{4.3}$$

   3.2. $YC_bC_r \to RGB$:

$$\begin{aligned} G &= Y - 0.714(C_r - 128) - 0.334(C_b - 128); \\ R &= Y + 1.402(C_r - 128); \\ B &= Y + 1.772(C_b - 128). \end{aligned} \tag{4.4}$$

4. Насыщение:

$$Sat(x, x_{min}, x_{max}) = \begin{cases} x_{min} = \text{если } x < x_{min}; \\ x_{max} = \text{если } x > x_{max}; \\ x \quad\quad = \text{во всех остальных случаях.} \end{cases} \tag{4.5}$$

5 Результаты выполнения исследования

Задание 1

Согласовано следующее изображение:

Рис. 5.1: Исходное изображение.

Задание 3

Выделили R, G и B компоненты изображения и записали их в отдельные файлы:

Рис. 5.2: Выделенные компоненты R, G, B.

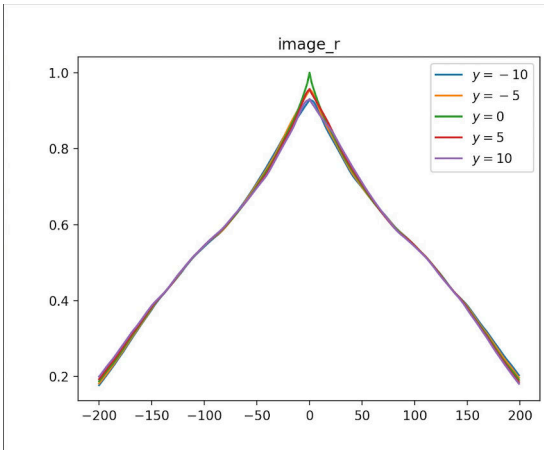
Задание 4

Оценки коэффициентов корреляции между каждой парой компонент:

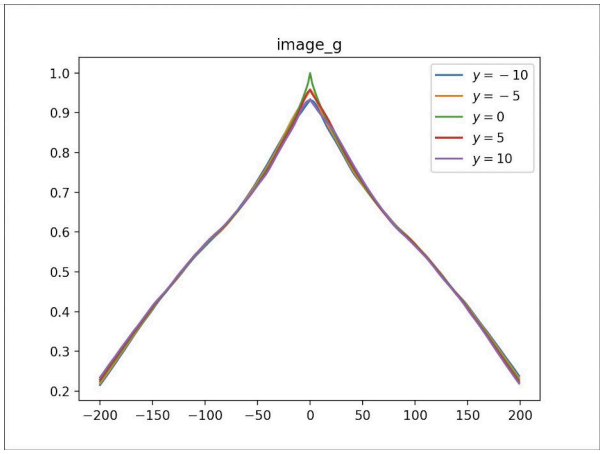
$r_{R,G}$	0.996
$r_{R,B}$	0.958
$r_{B,G}$	0.976

Таблица 5.1: Оценки коэффициентов корреляции.

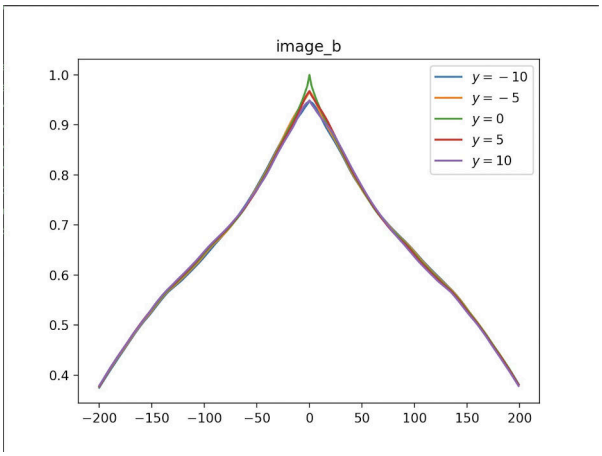
Сечения трехмерного графика оценки нормированной автокорреляционной функции:



(a) Автокорреляционная функция для компоненты R.



(b) Автокорреляционная функция для компоненты G.



(c) Автокорреляционная функция для компоненты B.

Рис. 5.3: Автокорреляционные функции компонент R, G, B.

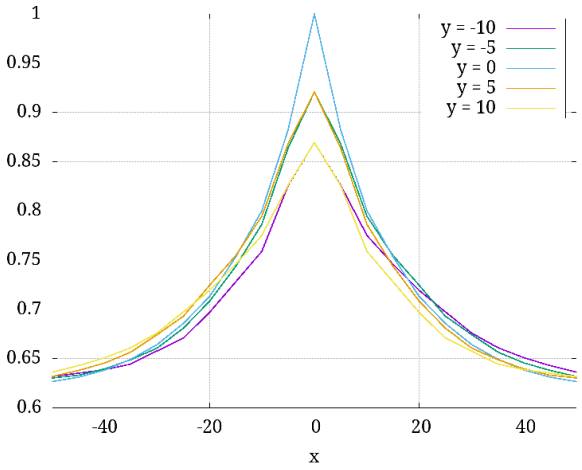
Задание 5

Оценки коэффициентов корреляции между каждой парой компонент:

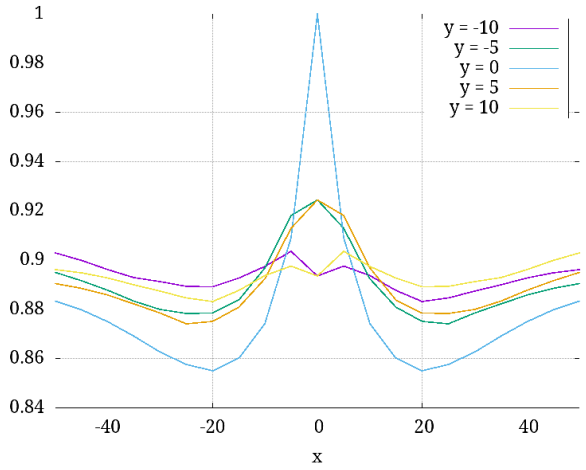
r_{Y,C_b}	0.083
r_{Y,C_r}	-0.111
r_{C_b,C_r}	-0.087

Таблица 5.2: Оценки коэффициентов корреляции.

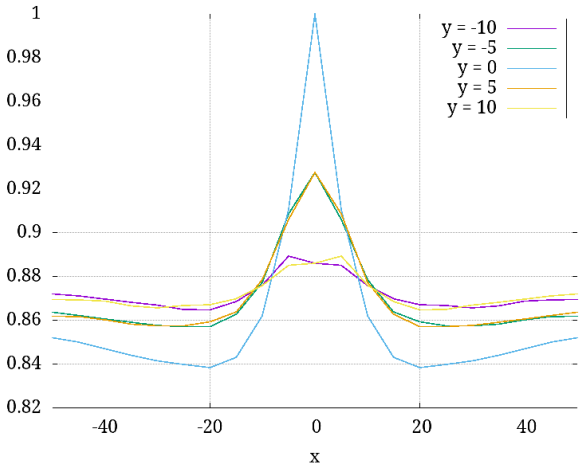
Сечения трехмерного графика оценки нормированной автокорреляционной функции:



(a) Автокорреляционная функция для компоненты Y.



(b) Автокорреляционная функция для компоненты C_b.



(c) Автокорреляционная функция для компоненты C_r.

Рис. 5.4: Автокорреляционные функции компонент Y, C_b, C_r.

Задание 6

Выделили Y, C_b и C_r компоненты изображения и записали их в отдельные файлы:

Рис. 5.5: Выделенные компоненты Y, C_b, C_r.

Задание 7

Преобразовали компоненты R, G, B в Y, C_b, C_r и обратно.

Пиковые значения отношения сигнал/шум для восстановленных компонент:

Компонента	$PSNR$, дБ
R	48.864
G	48.796
B	66.988

Таблица 5.3: $PSNR$.

Задание 9

При децимации компонент C_b, C_r в 2 раза по ширине и 2 раза по высоте с последующим восстановлением получили:

(a) исключение строк и столбцов.

(b) среднее арифметическое соседей.

Рис. 5.6: Результаты восстановления изображения после децимации.

Задание 10

Значения $PSNR$ по исходным и восстановленным данным:

Компонента	$PSNR$, дБ
C_b	42.327
C_r	46.386
R	43.960
G	47.694
B	38.157

(a) удаление строк и столбцов.

Компонента	$PSNR$, дБ
C_b	44.856
C_r	48.556
R	44.855
G	50.904
B	39.958

(b) среднее арифметическое соседей.

Таблица 5.4: $PSNR$.

Задание 11

При децимации компонент C_b, C_r в 4 раза по ширине и 4 раза по высоте с последующим восстановлением получили:

(a) исключение строк и столбцов. (b) среднее арифметическое соседей.

Рис. 5.7: Результаты восстановления изображения после децимации.

Значения $PSNR$ по исходным и восстановленным данным:

Компонента	$PSNR$, дБ
C_b	37.523
C_r	41.643
R	39.284
G	45.741
B	33.074

(a) удаление строк и столбцов.

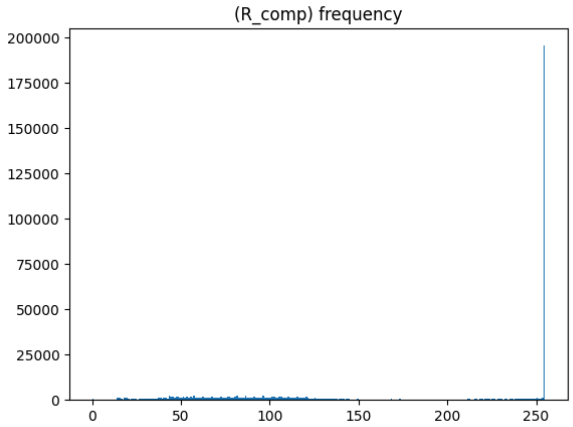
Компонента	$PSNR$, дБ
C_b	40.774
C_r	44.619
R	41.251
G	49.760
B	36.107

(b) среднее арифметическое соседей.

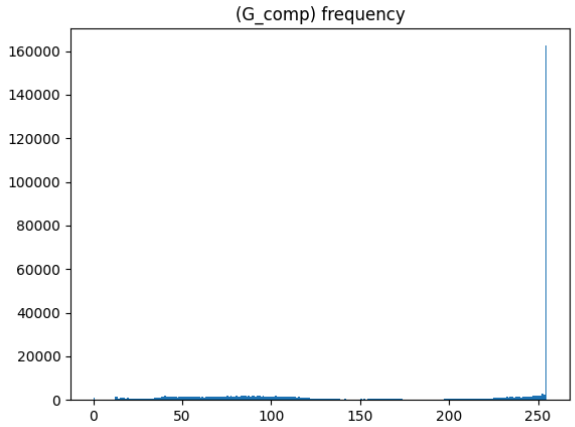
Таблица 5.5: $PSNR$.

Задание 12

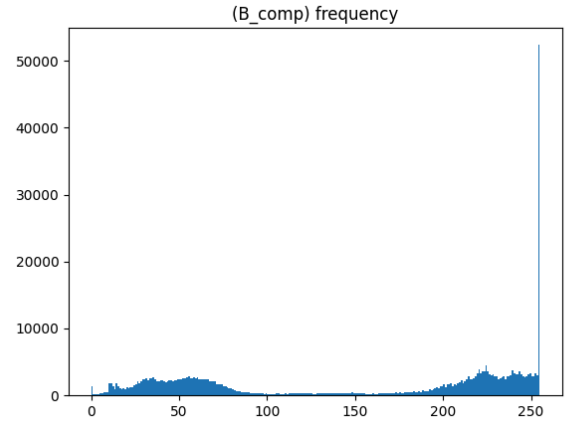
Построили гистограммы частот для компонент R, G, B, Y, C_b, C_r:



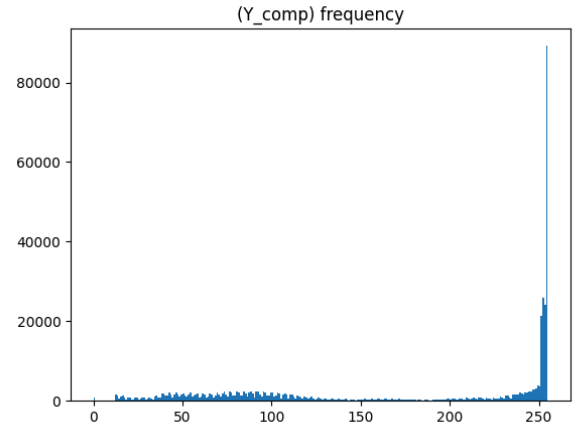
(a) R.



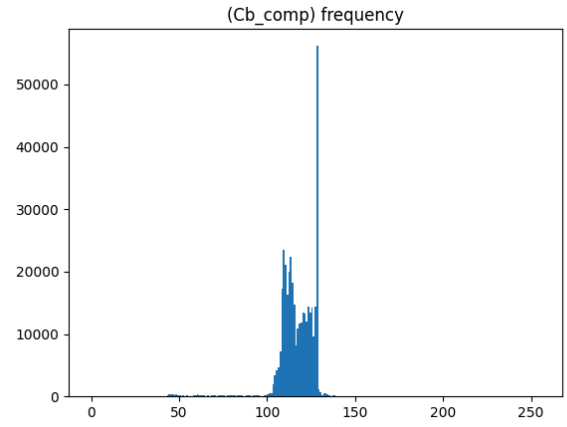
(b) G.



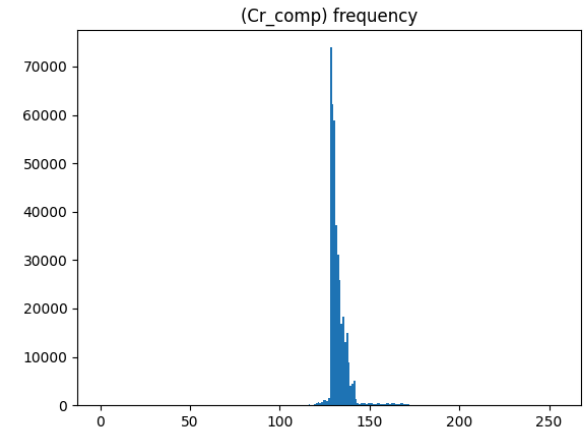
(c) B.



(d) Y.



(e) C_b.



(f) C_r.

Рис. 5.8: Гистограммы частот компонент.

Задание 13

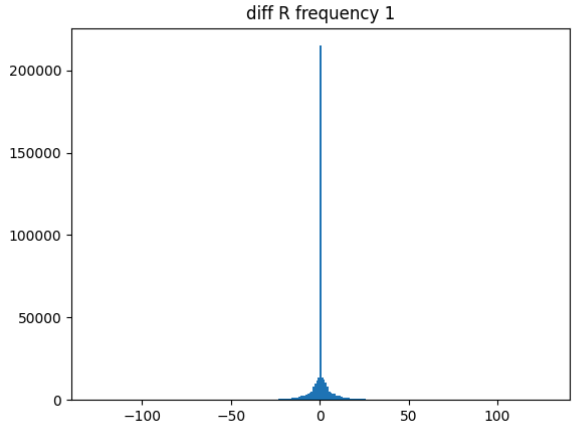
Оценка числа бит, затрачиваемых при поэлементном независимом сжатии компонент:

Компонента	H, бит
R	4.854
G	5.526
B	7.107
Y	6.336
C_b	4.707
C_r	3.692

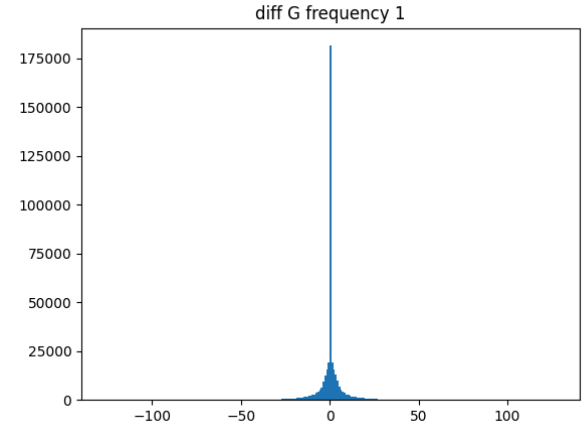
Таблица 5.6: Энтропия.

Задание 15

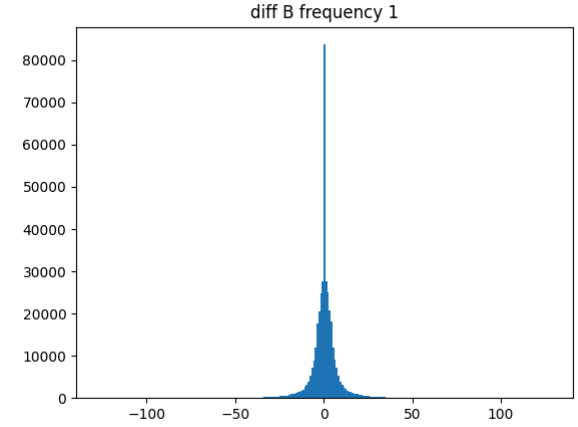
1. Гистограммы частот для D^1_R, D^1_G, D^1_B, D^1_Y, $D^1_{C_b}$, $D^1_{C_r}$:



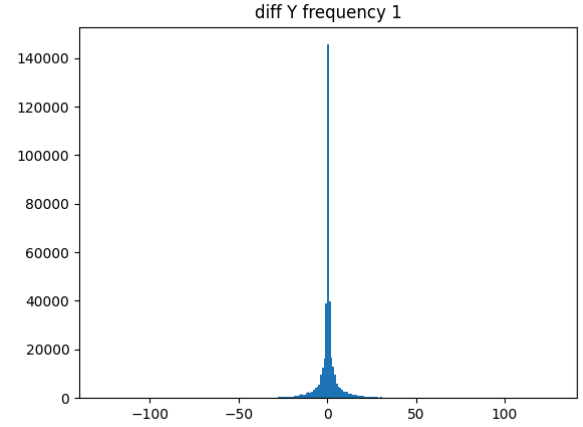
(a) D^1_R.



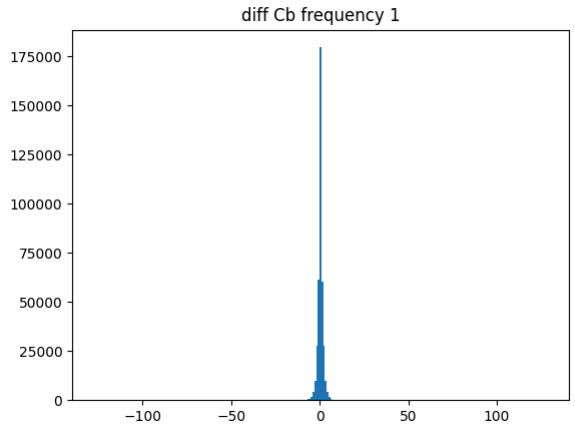
(b) D^1_G.



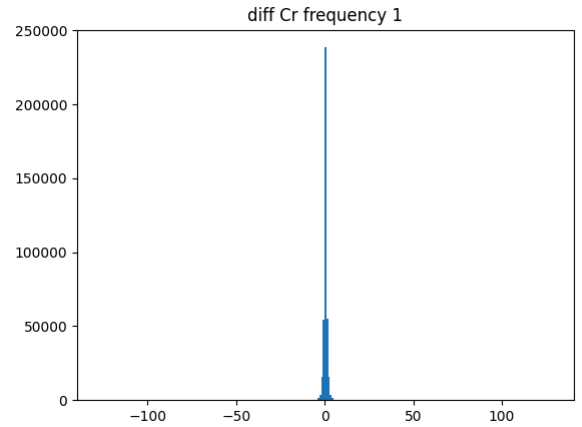
(c) D^1_B.



(d) D^1_Y.



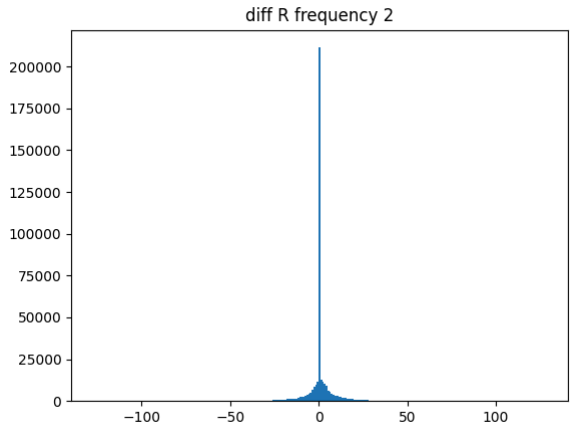
(e) $D^1_{C_b}$.



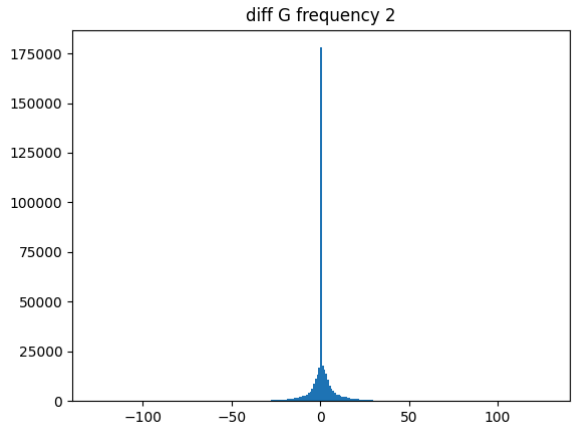
(f) $D^1_{C_r}$.

Рис. 5.9: Гистограммы частот компонент.

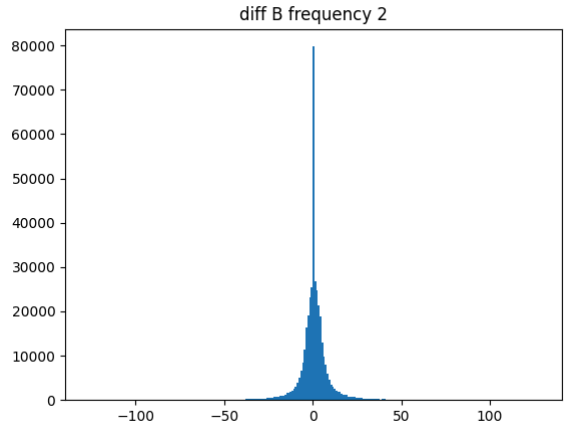
2. Гистограммы частот для D_R^2, D_G^2, D_B^2, D_Y^2, $D_{C_b}^2$, $D_{C_r}^2$:



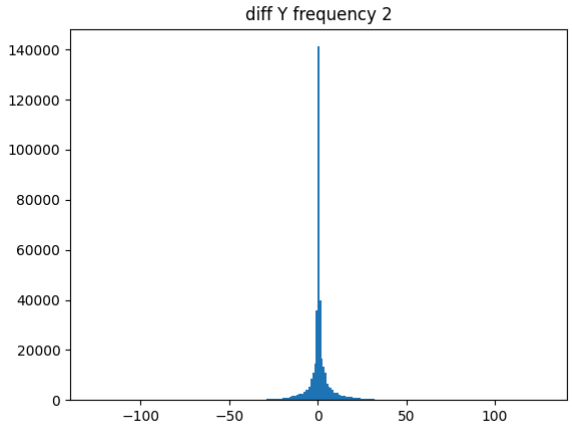
(a) D_R^2.



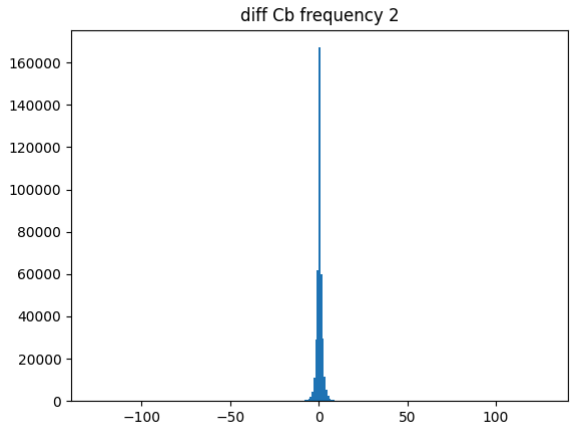
(b) D_G^2.



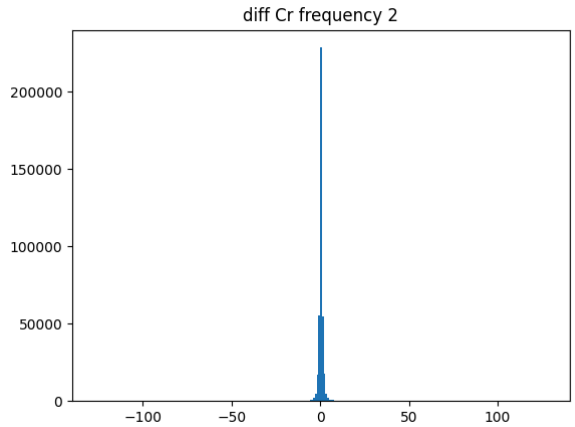
(c) D_B^2.



(d) D_Y^2.



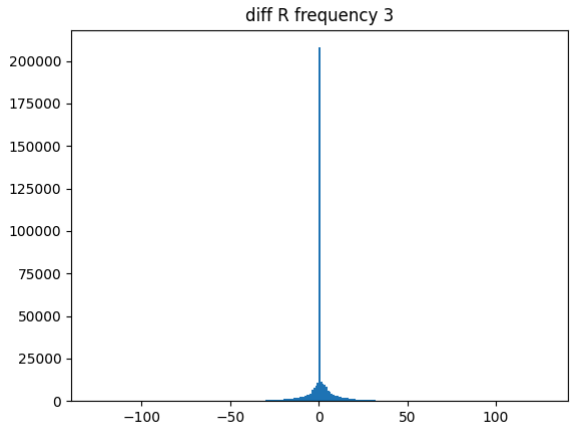
(e) $D_{C_b}^2$.



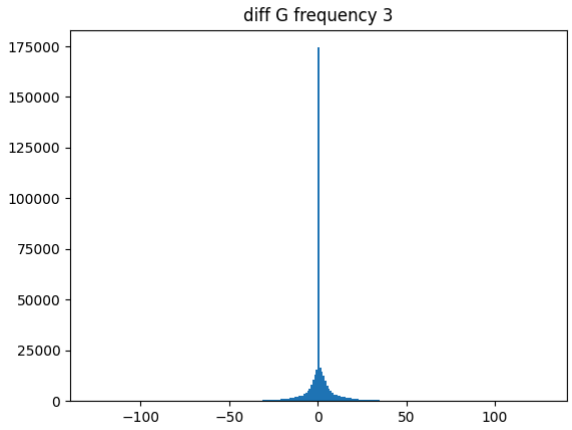
(f) $D_{C_r}^2$.

Рис. 5.10: Гистограммы частот компонент.

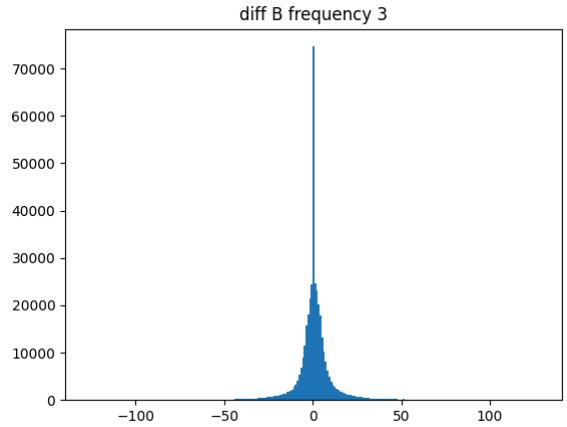
3. Гистограммы частот для D^3_R, D^3_G, D^3_B, D^3_Y, $D^3_{C_b}$, $D^3_{C_r}$:



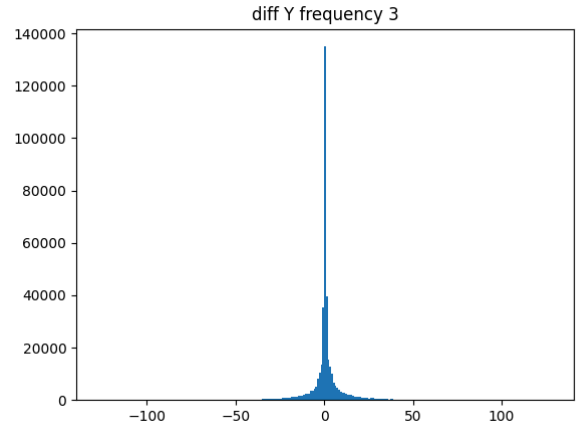
(a) D^3_R.



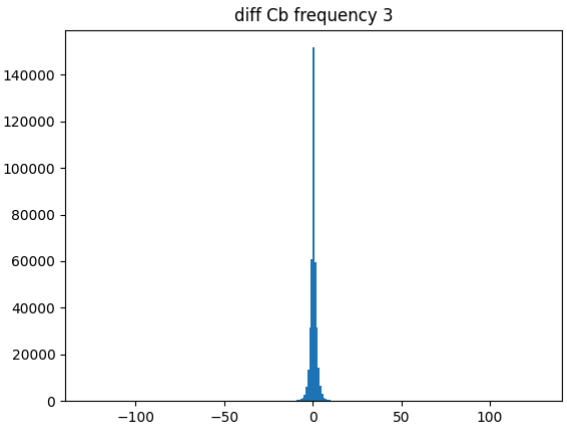
(b) D^3_G.



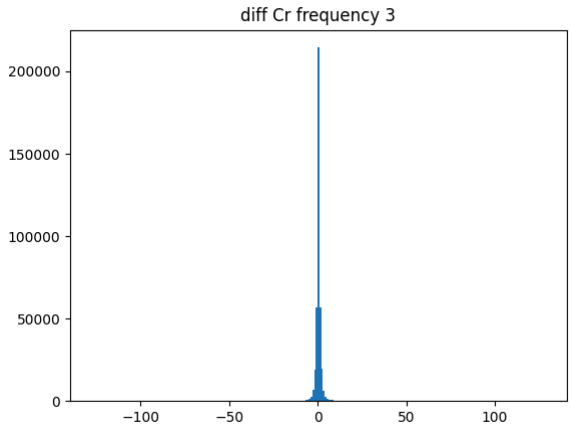
(c) D^3_B.



(d) D^3_Y.



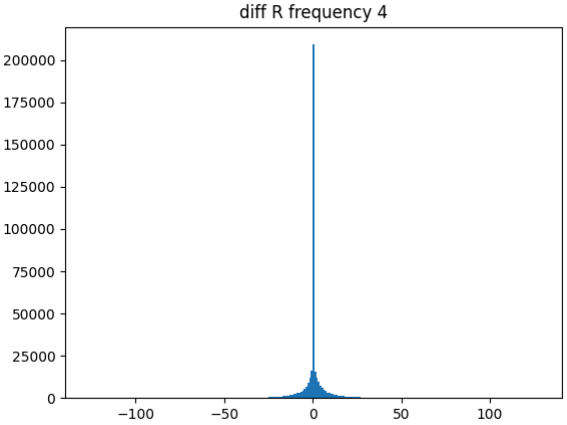
(e) $D^3_{C_b}$.



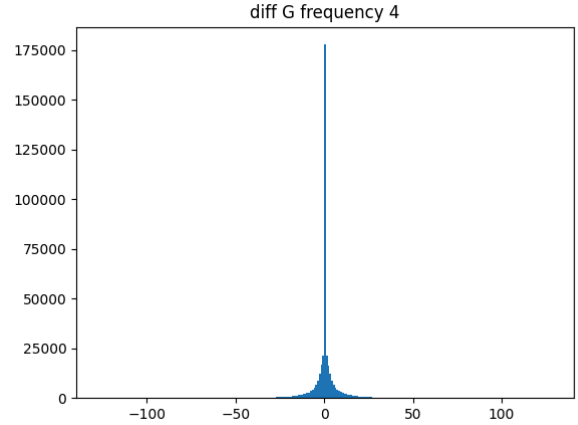
(f) $D^3_{C_r}$.

Рис. 5.11: Гистограммы частот компонент.

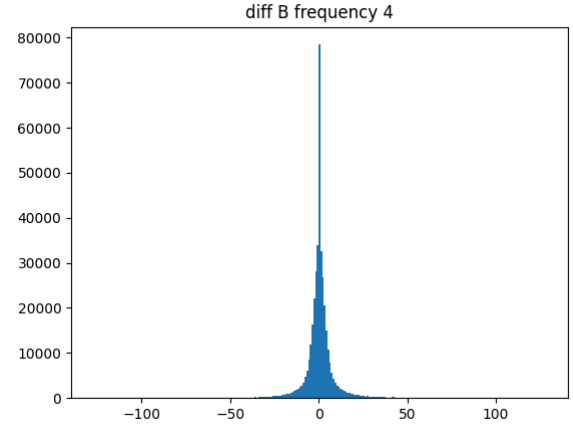
4. Гистограммы частот для D_R^4, D_G^4, D_B^4, D_Y^4, $D_{C_b}^4$, $D_{C_r}^4$:



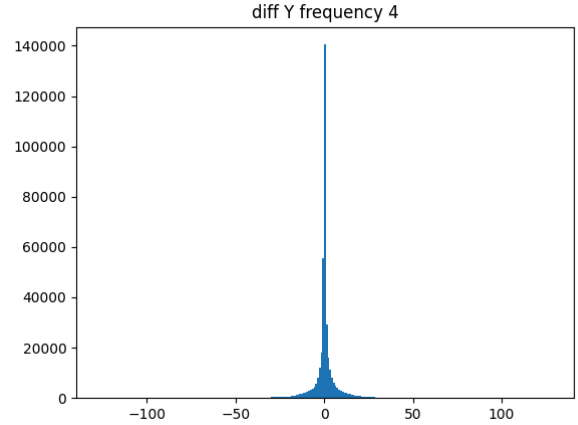
(a) D_R^4.



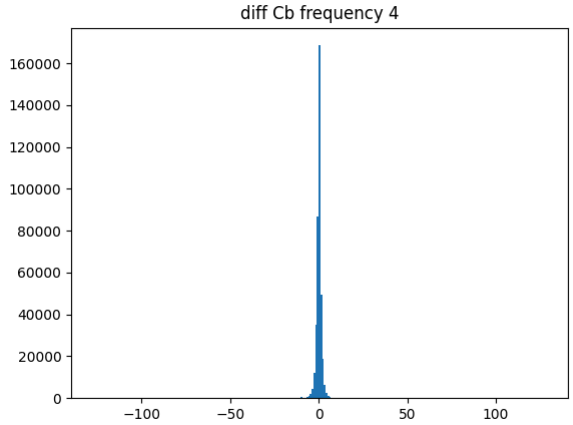
(b) D_G^4.



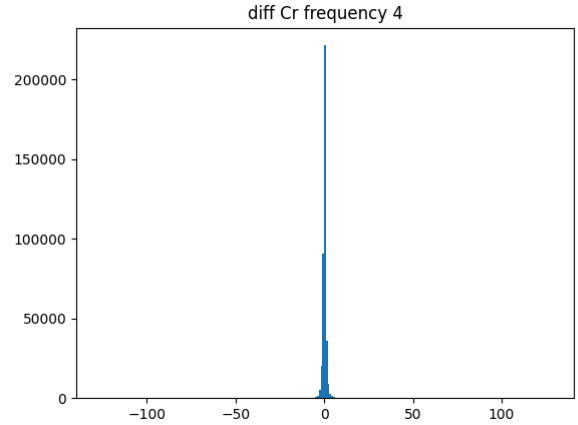
(c) D_B^4.



(d) D_Y^4.



(e) $D_{C_b}^4$.



(f) $D_{C_r}^4$.

Рис. 5.12: Гистограммы частот компонент.

Задание 16

Оценка числа бит, затрачиваемых при поэлементном независимом сжатии компонент:

Компонента	H, бит
D_R^1	3.456
D_G^1	3.809
D_B^1	4.625
D_Y^1	4.001
$D_{C_b}^1$	2.476
$D_{C_r}^1$	1.859

(a) D^1.

Компонента	H, бит
D_R^2	3.609
D_G^2	3.967
D_B^2	4.775
D_Y^2	4.171
$D_{C_b}^2$	2.654
$D_{C_r}^2$	2.012

(b) D^2.

Компонента	H, бит
D_R^3	3.799
D_G^3	4.163
D_B^3	4.998
D_Y^3	4.385
$D_{C_b}^3$	2.846
$D_{C_r}^3$	2.185

(c) D^3.

Компонента	H, бит
D_R^4	3.396
D_G^4	3.692
D_B^4	4.439
D_Y^4	3.730
$D_{C_b}^4$	2.071
$D_{C_r}^4$	1.409

(d) D^4.

Таблица 5.7: Энтропия.

6 Индивидуальное задание

Постановка задачи: изменение иркости на N градаций.

Для получения результата было искользовано два подхода:

1. прибавление константного значения к каждой компоненте RGB;
2. Прибавление константного значения к компоненте Y;

Результаты работы алгоритма:

(a) Исходное изображение.

(b) Преобразованное изображение.

7 Выводы

В ходе проведенного исследования были сделаны следующие выводы:

- **задание 4:**
  - компоненты R, G, B изображения сильно коррелируют между собой;
  - автокорреляционная функция для компонент максимально при нулевых сдвигах по x и по y;
- **задание 5:**
  - компоненты Y, C_b, и Y, C_r изображения почти не коррелируют;
- **задание 6:**
  - компонента Y представляет собой черно-белое представление исходного изображения;
  - компоненты C_b, C_r являются инверсией друг друга;
- **задание 9:**
  - при децимации компонент C_b, C_r качество изображения "на глаз" сильно не меняется;
- **задание 10:**
  - значение $PSNR$ после децимации восстановления компонент C_b, C_r довольно высоки (> 35), причём при децимации с вычислением среднего арифметического соседних $PSNR$ выше;
- **задание 11:**
  - значение $PSNR$ остаётся высоким (> 30) даже после децимации C_b, C_r в 4 раза;
- **задание 12:**
  - по гистограммам частот компонент видно, что значения компонент C_b, C_r в основном принимают средние значения;
- **задание 13:**
  - компоненты C_b, C_r несут в себе наименьшее количество информации ($5 - 6$ бит на пиксель);
- **задание 15:**
  - разница между интенсивностями соседних пикселей компонент очень мала;
- **задание 16:**
  - разностное кодирование позволяет сильно сократить количество бит, затрачиваемых на представление пикселя изображения;
- **индивидуальное задание:**
  - контрастность изображения можно менять несколькими способами.

8. Листинг программы

```
import math
import matplotlib.pyplot as plt
import numpy as np
import struct
from PIL import Image


tmp = 0

def saveInFileAndGetComponents(image, filename, position, startPos, width,
height):
    step = 0
    storeImage = []
    for i in range(startPos, len(image)):
        if step != position:
            image[i] = 0
        else:
            storeImage.append(image[i])
        step = (step + 1) % 3
    imageFile = open(filename, 'wb')
    imageFile.write(bytes(image))
    imageFile.close()

    return storeImage

with
open('/Users/vladislavkacanovskij/Documents/Programs/Python/multimedia/kodim20.bm
p', 'rb') as f:
    header = f.read(54)
    pixel_offset = int.from_bytes(header[10:14], byteorder='little')
    width = int.from_bytes(header[18:22], byteorder='little')
    height = int.from_bytes(header[22:26], byteorder='little')

    # f_bitcount = struct.unpack(format,)
    image =
Image.open('/Users/vladislavkacanovskij/Documents/Programs/Python/multimedia/kodi
m20.bmp')

    f.seek(0)
    fullImage = bytearray(f.read())

    imageR = saveInFileAndGetComponents(fullImage.copy(), 'red.bmp', 2,
pixel_offset, width, height)
    imageG = saveInFileAndGetComponents(fullImage.copy(), 'green.bmp', 1,
pixel_offset, width, height)
    imageB = saveInFileAndGetComponents(fullImage.copy(), 'blue.bmp', 0,
pixel_offset, width, height)

def getR(image1, image2):
    mean1 = sum(image1) / (width * height)
    mean2 = sum(image2) / (width * height)
    cov = sum([(image1[i] - mean1) * (image2[i] - mean2) for i in
range(len(image1))]) / (width * height)
    std1 = math.sqrt(sum([(image1[i] - mean1) ** 2 for i in range(len(image1))])
/ (width * height - 1))
    std2 = math.sqrt(sum([(image2[i] - mean2) ** 2 for i in range(len(image2))])
/ (width * height - 1))
    corr = cov / (std1 * std2)
```

```
    return corr

def cadr(value):
    global tmp
    if value < 0:
        tmp += 1
        return 0
    elif value > 255:
        tmp += 1
        return 255
    return value

def calculateEntropy(arrayValues, label):
    valueToAmount = {}
    for i in arrayValues:
        value = int(i)
        temp = valueToAmount.get(value)
        valueToAmount[value] = 1 if temp == None else valueToAmount[value] + 1
    entropy = 0
    all = 0
    for key in valueToAmount.keys():
        all += valueToAmount[key]
    for key in valueToAmount.keys():
        px = valueToAmount[key] / all
        entropy = entropy - px * math.log2(px)
    print(label, entropy)

def createBarChart(componentArray, label, rangeValues):
    fig = plt.figure()
    plt.title(label)
    plt.hist(componentArray, bins=255, range=rangeValues)
    plt.savefig(label + '_plot.png')
    # plt.show()

def calculateDifferenceModulation(componentArray, width, height, cmplabel):
    diffModulation1 = []
    diffModulation2 = []
    diffModulation3 = []
    diffModulation4 = []
    for i in range(1, height):
        for j in range(1, width):
            diffModulation1.append(componentArray[i * width + j] -
componentArray[i * width + j - 1])
            diffModulation2.append(componentArray[i * width + j] -
componentArray[(i - 1) * width + j])
            diffModulation3.append(componentArray[i * width + j] -
componentArray[(i - 1) * width + j - 1])
            average = (componentArray[i * width + j - 1] + componentArray[(i - 1)
* width + j] + componentArray[(i - 1) * width + j - 1]) / 3
            diffModulation4.append(componentArray[i * width + j] - average)

    createBarChart(diffModulation1, "diff " + cmplabel + " frequency 1", (-127,
128))
    createBarChart(diffModulation2, "diff " + cmplabel + " frequency 2", (-127,
128))
    createBarChart(diffModulation3, "diff " + cmplabel + " frequency 3", (-127,
128))
    createBarChart(diffModulation4, "diff " + cmplabel + " frequency 4", (-127,
128))

    calculateEntropy(diffModulation1, 'H(d' + cmplabel + '^1) = ')
    calculateEntropy(diffModulation2, 'H(d' + cmplabel + '^2) = ')
```

```
    calculateEntropy(diffModulation3, 'H(d' + cmplabel + '^3) = ')
    calculateEntropy(diffModulation4, 'H(d' + cmplabel + '^4) = ')

def splitIntoSubframes(array, i_count, j_count):
    h_downsampled, w_downsampled = height // 2, width // 2
    A_downsampled = []
    for i in range(0, height, 2):
        for j in range(0, width, 2):
            A_downsampled.append(array[(i + i_count) * w_downsampled + (j +
j_count)])

    A_restored = [[0 for j in range(width)] for i in range(height)]
    for i in range(h_downsampled):
        for j in range(w_downsampled):
            i_new = i * 2 + 1
            j_new = j * 2 + 1

            A_restored[i_new][j_new] = A_downsampled[i * w_downsampled + j]
            if j_new > 0:
                A_restored[i_new][j_new - 1] = A_downsampled[i * w_downsampled +
j]
            if i_new > 0:
                A_restored[i_new - 1][j_new] = A_downsampled[i * w_downsampled +
j]
            if i_new > 0 and j_new > 0:
                A_restored[i_new - 1][j_new - 1] = A_downsampled[i *
w_downsampled + j]
    result = []
    for sublist in A_restored:
        for item in sublist:
            result.append(item)
    return result

def createSubframes(yArray):
    yFile = bytearray(fullImage[:pixel_offset])
    j = 0
    for i in range(pixel_offset, len(fullImage), 3):
        yFile.extend(bytearray([int(yArray[j]), int(yArray[j]), int(yArray[j])]))
        j += 1
    return yFile

def decimationEvenNumbered(array, times=1): # четная нумерация
    h_downsampled, w_downsampled = height, width
    for time in range(times):
        A_downsampled = []
        for i in range(1, h_downsampled + 1, 2):
            for j in range(1, w_downsampled + 1, 2):
                A_downsampled.append(array[i * w_downsampled + j])
        array = A_downsampled
        h_downsampled, w_downsampled = h_downsampled // 2, w_downsampled // 2

    A_restored = []
    for time in range(times):
        A_restored = [[0 for j in range(w_downsampled * 2)] for i in
range(h_downsampled * 2)]
        for i in range(h_downsampled):
            for j in range(w_downsampled):
                i_new = i * 2 + 1
                j_new = j * 2 + 1

                A_restored[i_new][j_new] = A_downsampled[i * w_downsampled + j]
                if j_new > 0:
```

```
                    A_restored[i_new][j_new - 1] = A_downsampled[i *
w_downsampled + j]
                if i_new > 0:
                    A_restored[i_new - 1][j_new] = A_downsampled[i *
w_downsampled + j]
                if i_new > 0 and j_new > 0:
                    A_restored[i_new - 1][j_new - 1] = A_downsampled[i *
w_downsampled + j]
        A_downsampled = [] # A_downsampled = A_restored
        for sublist in A_restored:
            for item in sublist:
                A_downsampled.append(item)
        h_downsampled *= 2
        w_downsampled *= 2
    return A_restored

def decimationAriphmeticMean(array, times=1): #ср арифм
    h_downsampled, w_downsampled = height, width
    for time in range(times):
        A_downsampled = []
        for i in range(1, h_downsampled + 1, 2):
            for j in range(1, w_downsampled + 1, 2):
                A_downsampled.append((array[i * w_downsampled + j] + array[(i -
1) * w_downsampled + j] + array[i * w_downsampled + j - 1] + array[(i - 1) *
w_downsampled + j - 1]) // 4)
        array = A_downsampled
        h_downsampled, w_downsampled = h_downsampled // 2, w_downsampled // 2

    A_restored = []
    for time in range(times):
        A_restored = [[0 for j in range(w_downsampled * 2)] for i in
range(h_downsampled * 2)]
        for i in range(h_downsampled):
            for j in range(w_downsampled):
                i_new = i * 2 + 1
                j_new = j * 2 + 1

                A_restored[i_new][j_new] = A_downsampled[i * w_downsampled + j]
                if j_new > 0:
                    A_restored[i_new][j_new - 1] = A_downsampled[i *
w_downsampled + j]
                if i_new > 0:
                    A_restored[i_new - 1][j_new] = A_downsampled[i *
w_downsampled + j]
                if i_new > 0 and j_new > 0:
                    A_restored[i_new - 1][j_new - 1] = A_downsampled[i *
w_downsampled + j]
        A_downsampled = [] # A_downsampled = A_restored
        for sublist in A_restored:
            for item in sublist:
                A_downsampled.append(item)
        h_downsampled *= 2
        w_downsampled *= 2
    return A_restored

def convertForDecimation(yArray, Cr_restored, Cb_restored, label): #
Конвертировать для децимации
    global tmp
    tmp = 0
    yFile = bytearray(fullImage[:pixel_offset])
    sumR = 0
    sumG = 0
```

```
    sumB = 0
    sumCr = 0
    sumCb = 0
    schet = 0
    for i in range(height):
        for j in range(width):
            g = cadr((yArray[i * width + j] - 0.714 * (Cr_restored[i][j] - 128) -
0.334 * (Cb_restored[i][j] - 128)))
            r = cadr((yArray[i * width + j] + 1.402 * (Cr_restored[i][j] - 128)))
            b = cadr((yArray[i * width + j] + 1.772 * (Cb_restored[i][j] - 128)))
            sumR = sumR + (imageR[i * width + j] - int(r)) ** 2
            sumG = sumG + (imageG[i * width + j] - int(g)) ** 2
            sumB = sumB + (imageB[i * width + j] - int(b)) ** 2
            sumCr = sumCr + (int(Cr_restored[i][j]) - int(crArray[i * width +
j])) ** 2
            sumCb = sumCb + (int(Cb_restored[i][j]) - int(cbArray[i * width +
j])) ** 2
            yFile.append(int(b))
            yFile.append(int(g))
            yFile.append(int(r))
    print('')
    print(label)
    print('PSNR( Blue ) = ', 10 * math.log10((width * height * (255 ** 2)) /
sumB))
    print('PSNR( Green ) = ', 10 * math.log10((width * height * (255 ** 2)) /
sumG))
    print('PSNR( Red ) = ', 10 * math.log10((width * height * (255 ** 2)) /
sumR))
    print('PSNR( Cb ) = ', 10 * math.log10((width * height * (255 ** 2)) /
sumCb))
    print('PSNR( Cr ) = ', 10 * math.log10((width * height * (255 ** 2)) /
sumCr))
    print ('clip_count = ' + str(tmp))
    return yFile
print('-------------------------------------------------')

print('biBitCount = ' + (str)(image.BITFIELDS * 8))

print('--------------4---correlation-------------------')

print('r(red, green) = ', getR(imageR, imageG))
print('r(red, blue) = ', getR(imageR, imageB))
print('r(blue, green) = ', getR(imageB, imageG))
print('')


yFile = bytearray(fullImage[:pixel_offset])
cbFile = bytearray(fullImage[:pixel_offset])
crFile = bytearray(fullImage[:pixel_offset])

yArray = []
cbArray = []
crArray = []
for i in range(pixel_offset, len(fullImage), 3):
    y = 0.299 * fullImage[i + 2] + 0.587 * fullImage[i + 1] + 0.114 *
fullImage[i]
    cb = 0.5643 * (fullImage[i] - y) + 128
    cr = 0.7132 * (fullImage[i + 2] - y) + 128
    yArray.append(y)
    cbArray.append(cb)
    crArray.append(cr)
```

```
    yFile.extend(bytearray([int(y),int(y),int(y)]))
    crFile.extend(bytearray([int(cr),int(cr),int(cr)]))
    cbFile.extend(bytearray([int(cb),int(cb),int(cb)]))

print('-------------5--correlation------------------')
print('R(y, Cb) = ', getR(yArray, cbArray))
print('R(y, Cr) = ', getR(yArray, crArray))
print('R(Cb, Cr) = ', getR(cbArray, crArray))
print('')


imageFile = open('y.bmp', 'wb')
imageFile.write(bytes(yFile))
imageFile.close()

imageFile = open('Cb.bmp', 'wb')
imageFile.write(bytes(cbFile))
imageFile.close()

imageFile = open('Cr.bmp', 'wb')
imageFile.write(bytes(crFile))
imageFile.close()

tmp = 0
print('----------------------7----------------------')
yFile = bytearray(fullImage[:pixel_offset])
sumR = 0
sumG = 0
sumB = 0
for i in range(len(yArray)):
    g = cadr((yArray[i] - 0.714 * (crArray[i] - 128) - 0.334 * (cbArray[i] -
128)))
    r = cadr((yArray[i] + 1.402 * (crArray[i] - 128)))
    b = cadr((yArray[i] + 1.772 * (cbArray[i] - 128)))
    sumR = sumR + (imageR[i] - int(r)) ** 2
    sumG = sumG + (imageG[i] - int(g)) ** 2
    sumB = sumB + (imageB[i] - int(b)) ** 2
    yFile.append(int(b))
    yFile.append(int(g))
    yFile.append(int(r))
print('psnr(blue) = ', 10 * math.log10((width  * height * (255 ** 2)) / sumB))
print('psnr(green) = ', 10 * math.log10((width  * height * (255 ** 2)) / sumG))
print('psnr(red) = ', 10 * math.log10((width  * height * (255 ** 2)) / sumR))
print ('clip_count = ' + str(tmp))

print('')

imageFile = open('after.bmp', 'wb')
imageFile.write(bytes(yFile))
imageFile.close()

print('----------------------dop----------------------')
y_new_File = bytearray(fullImage[:pixel_offset])

brightness = 50

sumR = 0
sumG = 0
sumB = 0
for i in range(len(yArray)):
    g = cadr(yArray[i] - 0.714 * (crArray[i] - 128) - 0.334 * (cbArray[i] - 128))
    r = cadr(yArray[i] + 1.402 * (crArray[i] - 128))
```

```
    b = cadr(yArray[i] + 1.772 * (cbArray[i] - 128))
    sumR = sumR + (imageR[i] - int(r)) ** 2
    sumG = sumG + (imageG[i] - int(g)) ** 2
    sumB = sumB + (imageB[i] - int(b)) ** 2

    new_b = (int(b) + brightness)
    new_g = (int(g) + brightness)
    new_r = (int(r) + brightness)


    y_new_File.append(cadr(new_b))
    y_new_File.append(cadr(new_g))
    y_new_File.append(cadr(new_r))


# print('psnr(blue) = ', 10 * math.log10((width  * height * (255 ** 2)) / sumB))
# print('psnr(green) = ', 10 * math.log10((width  * height * (255 ** 2)) / sumG))
# print('psnr(red) = ', 10 * math.log10((width  * height * (255 ** 2)) / sumR))
# print('')

imageFile = open('brightness_DOP.bmp', 'wb')
imageFile.write(bytes(y_new_File))
imageFile.close()
print('Complite!\n')

sumR = 0
sumG = 0
sumB = 0
for i in range(len(yArray)):
    g = cadr(yArray[i] + brightness - 0.714 * (crArray[i] - 128) - 0.334 *
(cbArray[i] - 128))
    r = cadr(yArray[i] + 1.402 * (crArray[i] - 128))
    b = cadr(yArray[i] + 1.772 * (cbArray[i] - 128))
    sumR = sumR + (imageR[i] - int(r)) ** 2
    sumG = sumG + (imageG[i] - int(g)) ** 2
    sumB = sumB + (imageB[i] - int(b)) ** 2

    new_b = (int(b))
    new_g = (int(g))
    new_r = (int(r))


    y_new_File.append(cadr(new_b))
    y_new_File.append(cadr(new_g))
    y_new_File.append(cadr(new_r))


imageFile = open('brightness_DOP_ycbcr.bmp', 'wb')
imageFile.write(bytes(y_new_File))
imageFile.close()
print('Complite!\n')

print('Complite!\n')

yArray = [round(yArray[i]) for i in range(len(yArray))]
yArray = np.array(yArray)
cbArray = [round(cbArray[i]) for i in range(len(cbArray))]
cbArray = np.array(cbArray)
crArray = [round(crArray[i]) for i in range(len(crArray))]
crArray = np.array(crArray)

imageR = [round(imageR[i]) for i in range(len(imageR))]
```

```
imageR = np.array(imageR)
imageG = [round(imageG[i]) for i in range(len(imageG))]
imageG = np.array(imageG)
imageB = [round(imageB[i]) for i in range(len(imageB))]
imageB = np.array(imageB)

# print('------------------12----------------\n')

createBarChart(yArray, "(Y_comp) frequency", (0, 255))
createBarChart(cbArray, "(Cb_comp) frequency", (0, 255))
createBarChart(crArray, "(Cr_comp) frequency", (0, 255))
createBarChart(imageR, "(R_comp) frequency", (0, 255))
createBarChart(imageG, "(G_comp) frequency", (0, 255))
createBarChart(imageB, "(B_comp) frequency", (0, 255))

Cb_restored = decimationEvenNumbered(cbArray)
Cr_restored = decimationEvenNumbered(crArray)

print('---------------------8-11---------------------')

yFile = convertForDecimation(yArray,Cr_restored,Cb_restored, "even_numbered
decimation x2")

imageFile = open('after_EvenNumb_Decim_x2.bmp', 'wb')
imageFile.write(bytes(yFile))
imageFile.close()

Cb_restored = decimationAriphmeticMean(cbArray)
Cr_restored = decimationAriphmeticMean(crArray)

yFile = convertForDecimation(yArray,Cr_restored,Cb_restored, "ariphmetic mean
decimation x2")

imageFile = open('after_AriphmMean_Decim_x2.bmp', 'wb')
imageFile.write(bytes(yFile))
imageFile.close()


Cb_restored2 = decimationEvenNumbered(cbArray, times=2)
Cr_restored2 = decimationEvenNumbered(crArray, times=2)

yFile = convertForDecimation(yArray, Cr_restored2, Cb_restored2, "even_numbered
decimation x4")

imageFile = open('after_EvenNumb_Decim_x4.bmp', 'wb')
imageFile.write(bytes(yFile))
imageFile.close()

Cb_restored2 = decimationAriphmeticMean(cbArray, times=2)
Cr_restored2 = decimationAriphmeticMean(crArray, times=2)

yFile = convertForDecimation(yArray, Cr_restored2, Cb_restored2, "ariphmetic mean
decimation x4")

imageFile = open('after_AriphmMean_Decim_x4.bmp', 'wb')
imageFile.write(bytes(yFile))
imageFile.close()

print('--------------------13---------------------')

calculateEntropy(imageB, 'H(B) = ')
calculateEntropy(imageG, 'H(G) = ')
```

```
calculateEntropy(imageR, 'H(R) = ')
calculateEntropy(yArray, 'H(Y) = ')
calculateEntropy(cbArray, 'H(Cb) = ')
calculateEntropy(crArray, 'H(Cr) = ')

print('--------------------16----------------------')
calculateDifferenceModulation(imageB, width, height, 'B')
print('')
calculateDifferenceModulation(imageG, width, height, 'G')
print('')
calculateDifferenceModulation(imageR, width, height, 'R')
print('')
calculateDifferenceModulation(yArray, width, height, 'Y')
print('')
calculateDifferenceModulation(cbArray, width, height, 'Cb')
print('')
calculateDifferenceModulation(crArray, width, height, 'Cr')
print('------------------------------------------')

def get_components(image, filename, position, start_pos, width, height):
    step = 0
    store_image = [[]]
    component_pos = 0
    for i in range(start_pos, len(image)):
        cur_height = len(store_image) - 1
        if position != step:
            image[i] = 0
        else:
            store_image[cur_height].append(image[i])
            component_pos = component_pos + 1
        if (component_pos == width and cur_height + 1 != height):
            store_image.append([])
            component_pos = 0
        step = (step + 1) % 3
    imageFile = open(filename, 'wb')
    imageFile.write(bytes(image))
    imageFile.close()

    return store_image

def get_r(image1, image2):
    mean1 = sum(sum(image1, [])) / (len(image1) * len(image1[0]))
    mean2 = sum(sum(image2, [])) / (len(image2) * len(image2[0]))
    cov = sum([(image1[i][j] - mean1) * (image2[i][j] - mean2) for i in
range(len(image1)) for j in range(len(image1[0]))]) / (len(image2) *
len(image2[0]))
    std1 = math.sqrt(sum([(image1[i][j] - mean1) ** 2 for i in range(len(image1))
for j in range(len(image1[0]))]) / (len(image2) * len(image2[0]) - 1))
    std2 = math.sqrt(sum([(image2[i][j] - mean2) ** 2 for i in range(len(image2))
for j in range(len(image2[0]))]) / (len(image2) * len(image2[0]) - 1))
    corr = cov / (std1 * std2)

    return corr

def get_r_auto(full_image, delta_x, delta_y):
    image1 = [[]]
    image2 = [[]]
    current_row = 0

    def correct_x(x):
        res = x
        if x < 0:
```

```
            res = 0
        elif x > len(full_image):
            res = len(full_image)
        return res

    def correct_y(y):
        res = y
        if y < 0:
            res = 0
        elif y > len(full_image[0]):
            res = len(full_image[0])
        return res

    first_start_row = correct_x(0 + delta_x)
    first_start_column = correct_y(0 + delta_y)
    first_end_row = correct_x(len(full_image) + delta_x)
    first_end_column = correct_y(len(full_image[0]) + delta_y)

    second_start_row = correct_x(0 - delta_x)
    second_start_column = correct_y(0 - delta_y)
    second_end_row = correct_x(len(full_image) - delta_x)
    second_end_column = correct_y(len(full_image[0]) - delta_y)


    for i in range(first_start_row, first_end_row):
        current_column = 0
        image1.insert(current_row, [])
        for j in range(first_start_column, first_end_column):
            image1[current_row].insert(current_column, full_image[i][j])
            current_column += 1
        current_row += 1
    if len(image1) > 0:
        image1.pop(len(image1) - 1)

    current_row = 0
    for i in range(second_start_row, second_end_row):
        current_column = 0
        image2.insert(current_row, [])
        for j in range(second_start_column, second_end_column):
            image2[current_row].insert(current_column, full_image[i][j])
            current_column += 1
        current_row += 1
    if len(image2) > 0:
        image2.pop(len(image2) - 1)


    return get_r(image1, image2)

def show_graphics(image, title):
    all_x = []
    max_check_points = 400
    all_x.insert(0,math.floor((-0.5) * max_check_points))

    for i in range(1, max_check_points):
        all_x.insert(i, i - (max_check_points / 2))
        all_x[i] = int(math.floor(all_x[i]))
    all_y = [-10, -5, 0, 5, 10]

    def auto_correlation(all_x, current_y):
        result = []
        for current_x in all_x:
            result.append(get_r_auto(image, current_x, current_y))
```

```
        return result

    plt.figure()
    for y in all_y:
        plt.title(title)
        plt.plot(all_x, auto_correlation(all_x, y), label='$y = %i$'%y)
        plt.legend()

    plt.show()

with
open('/Users/vladislavkacanovskij/Documents/Programs/Python/multimedia/kodim20.bm
p', 'rb') as f:
    header = f.read(54)
    pixel_offset = int.from_bytes(header[10:14], byteorder='little')
    width = int.from_bytes(header[18:22], byteorder='little')
    height = int.from_bytes(header[22:26], byteorder='little')

    f.seek(0)
    full_image = bytearray(f.read())

    image_r = get_components(full_image.copy(), 'red.bmp', 2, pixel_offset,
width, height)
    image_g = get_components(full_image.copy(), 'green.bmp', 1, pixel_offset,
width, height)
    image_b = get_components(full_image.copy(), 'blue.bmp', 0, pixel_offset,
width, height)

print('--------------4---correlation-------------------')

print(get_r(image_r, image_g))
print(get_r(image_r, image_b))
print(get_r(image_b, image_g))
print('-----------------------------------------------')
```