Цель работы:

Приобретение навыков встраивания и извлечения информации из коэффициентов дискретного косинусного преобразования с помощью метода наименее значимых битов.

Задача работы:

Программно реализовать метод встраивания и извлечения сообщения из коэффициентов дискретного косинусного преобразования изображения формата «.bmp» методом LSB. Рассчитать параметры PSNR и RMSE для исходного изображения и изображения со встроенной информации. Построить графики зависимости PSNR и RMSE от количества встроенной информации. Провести оценку целесообразности встраивания.

Теоретическая часть

ДКП или дискретно косинусное преобразование являющееся частным случаем двумерного преобразования Фурье. Как известно, преобразование Фурье — это метод обработки, который, анализируя изменения сигнала во времени, выражает их в виде частотного спектра. Любой сигнал можно разложить на частотные гармонические составляющие, и затем по известным значениям амплитуды в фазы этих составляющих их линейным суммировав нем восстановить исходный сигнал. Последнюю операцию называют обратным преобразованием Фурье. В цифровых системах сигнал выражается последовательностью дискретных отсчетов. При использовании преобразования Фурье для фрагмента цифрового сигнала из некоторого ограниченного числа отсчетов последний можно разложить на такое же число дискретных частот. Это преобразование называют дискретным преобразованием Фурье [1].

Поскольку любое изображение или его фрагмент можно рассматривать как функцию изменения яркости (цветности) как по оси X, так и по оси Y, то дискретное ортогональное преобразование Фурье будет представлять собой замену массива отсчетов изображения соответствующего фрагмента на массив коэффициентов, соответствующих амплитудам частотных составляющих Фурье [1].

Объем расчетов для нахождения этих коэффициентов весьма значителен. Поэтому преобразования осуществляются над небольшими по размеру фрагментами, обычно 8×8 элементов. Дискретно-косинусное преобразование Фурье в определенной степени минимизирует объем этих вычислений использованием в качестве набора преобразующих (базисных) функций только косинусных составляющих. В результате массиву исходных значений сигнала соответствует массив из такого же числа коэффициентов, представляющих собой амплитуды этих косинусных составляющих [1].

Метод LSB заключается в выделении наименее значимых бит изображения-контейнера с последующей их заменой на биты сообщения. Поскольку замене подвергаются лишь наименее значимые биты, разница между исходным изображением-контейнером и контейнером, содержащим скрытые данные невелика и обычно незаметна для человеческого глаза. Метод LSB применим лишь к изображениям в форматах без сжатия (например, BMP), так как для хранения скрытого сообщения используются наименее значимые биты значений пикселей, при сжатии с потерями эта информация может быть утеряна [2].

Преобразования, имеющие высокие значения выигрыша от кодирования, такое как ДКП характеризуется резко неравномерным распределением дисперсий коэффициентов субполос. Стегосообщение добавляется к субполосам изображения. Низкочастотные субполосы содержат подавляющую часть энергии изображения и, следовательно, носят шумовой характер. Высокочастотные субполосы наиболее подвержены воздействию со стороны различных алгоритмов обработки, будь то сжатие или НЧ фильтрация. Высокочастотные субполосы не подходят для вложения из-за большого шума обработки, а низкочастотные — из-за высокого шума изображения. Поэтому приходится ограничиваться среднечастотными полосами, в которых шум изображения равен шуму обработки [2].

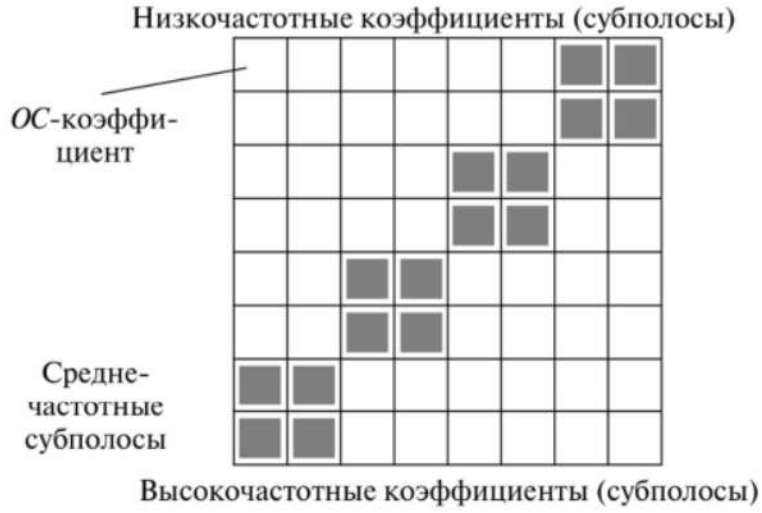

Рисунок 1. Субполосы коэффициентов ДКП.

Двумерное дискретно-косинусное преобразование матрицы с размерами M на N реализуется согласно следующему выражению [2]:

$$B_{pq} = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N},$$

где $0 \le p \le M-1$ и $0 \le q \le N-1$;

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{M}}, & \text{если } p = 0; \\ \sqrt{\frac{2}{M}}, & \text{если } 1 \le p \le M-1. \end{cases} \qquad \alpha_q = \begin{cases} \frac{1}{\sqrt{N}}, & \text{если } q = 0; \\ \sqrt{\frac{2}{N}}, & \text{если } 1 \le q \le N-1. \end{cases}$$

Выражение обратного дискретно-косинусного преобразования есть представление матрицы размерами M на N виде суммы следующих функций [2]:

$$\alpha_p \alpha_q \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N}, \text{ где } 0 \le p \le M-1 \text{ и } 0 \le q \le N-1$$

Практическая часть

Согласно задачам данной лабораторной работы, необходимо программно реализовать механизм встраивания и извлечения сообщения в коэффициенты дискретно косинусного преобразования изображения методом LSB. Для выполнения этих задач был выбрал язык программирования Python 3.8.

Имеется цветное изображение размера 320x200 пикселей, состоящее из пикселей, представленными сочетанием трех цветовых компонентов (субпикселей) RGB (красный, зеленый, синий). Каждый субпиксель имеет интенсивность, выраженную восьмибитным числом, в десятичном виде от 0 до 255 включительно, где 255 – максимальная интенсивность.

Задача программного модуля – разобрать имеющееся изображение на пиксели, разбить изображение на блоки размером 8x8 пикселей, провести побитовое встраивание сообщения, заведомо переведенного в двоичную строку, в наименее значимые биты коэффициентов дискретно косинусного преобразования блоков, затем собрать изображение обратно в формат «.bmp».

Преобразование сообщения, состоящего из кириллических букв и пробелов в двоичный код, осуществляется со стандартизированной размерностью символа в 7 бит.

Дискретное косинусное преобразование для блоков 8x8 субпикселей проводится по формулам, указанным в теоретической части .

Встраивание происходит в определенные коэффициенты, встраивание в которые не сильно повлияет на искажение исходного изображения. Эти определенные коэффициенты будут выявлены в результате исследования.

Демонстрация работы программы.

Рисунок 2. Исходное изображение.

```
Для встраивания введите 1, для извлечения введите 2: 1
Введите сообщение для встраивания в текст: Стеганография встраивание в коэффициенты ДКП
Встроено сообщение "Стеганография встраивание в коэффициенты ДКП" в изображение "img_encode.bmp"
Параметр RMSE равен 0.036
Параметр PSNR равен 77.09 dB
```

Рисунок 3. Демонстрация вывода программы при встраивании сообщения.

Рисунок 4. Изображение со встроенным сообщением.

```
Для встраивания введите 1, для извлечения введите 2: 2
Извлечено сообщение "Стегбногргзия встСаивамие в тоэффициенты ДКП" из изображения
```

Рисунок 5. Демонстрация вывода программы при извлечении сообщения.

Видно, что извлеченное сообщение не в полной мере соответствует тому, которое было встроено в коэффициенты дискретно косинусного преобразование. Это связано с тем, что после ДКП значения коэффициентов округляются для встраивания бита. А также из-за того, что после встраивания сообщения в наименее значимые биты происходит обратное ДКП, чтобы получить значения пикселей, и из-за того, что полученные значения пикселей должны быть целыми числами, то происходит повторное округление чисел. Данные округления значений пикселей сказываются на коэффициентах при повторном ДКП для извлечения сообщения.

Выбор коэффициентов для встраивания.

Проведем исследование, которое покажет нам, в какие коэффициенты ДКП лучше всего встраивать информацию. Показателем для сравнения будем считать процент ошибок в извлеченном сообщении.

Для этого проведем последовательное встраивание сообщений, состоящих из 100, 200 и 300 символов в различные наборы коэффициентов.

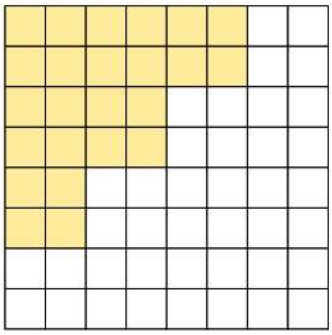

Рисунок 6. Низкочастотные коэффициенты.
Проценты ошибок: 10%, 12%, 11%

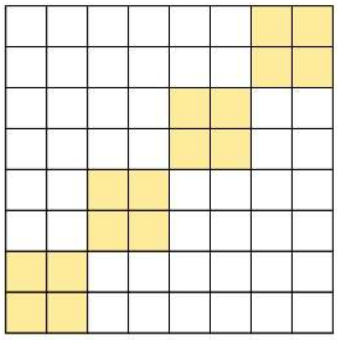

Рисунок 7. Среднечастотные коэффициенты.
Проценты ошибок: 8%, 7%, 9%

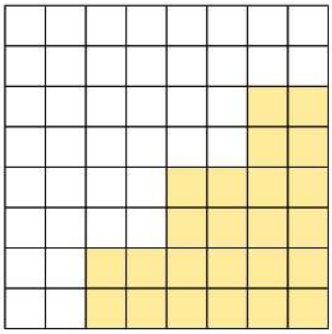

Рисунок 8. Высокочастотные коэффициенты.
Проценты ошибок: 9%, 12%, 10%

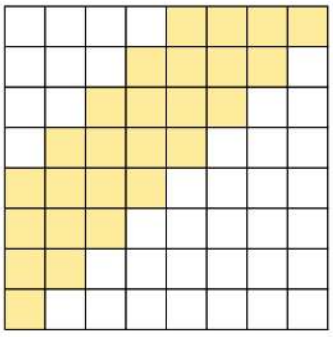

Рисунок 9. Комбинация коэффициентов.
Проценты ошибок: 4%, 6%, 6%

Как видно из результатов исследования, лучшим набором коэффициентов для встраивания будем считать комбинированный набор, состоящий из коэффициентов средней частоты и некоторых коэффициентов низкой частоты, выдающий до 6% ошибок в извлеченном сообщении.

Округление коэффициентов ДКП.

Проведем исследование, которое покажет нам, что будет происходить при округлении всех коэффициентов ДКП после преобразования, а не округления только тех коэффициентов, в которые встраивается сообщение. Будем сравнивать процент ошибок, PSNR и RMSE.

Для этого проведем встраивание сообщения длиной 200 символов.

Результаты при частичном округлении коэффициентов ДКП:

Ошибки	6%
RMSE	0.068
PSNR	71.485 dB

Результаты при округлении всех коэффициентов ДКП:

Ошибки	35%
RMSE	0.294
PSNR	58.777 dB

Как видно из результатов исследования, округление всех коэффициентов ДКП не приводит к лучшим результатам, так как изображение искажается полностью, вне зависимости от размера встраиваемого сообщения.

Параметр PSNR.

Был программно реализован расчет параметра PSNR.
Встроим сообщение «стеганография» в изображение. Параметр PSNR равен: 82.7 dB

Встроим по нарастающей 1-80 слов «стеганография» в изображение.
Построим график зависимости параметра PSNR от количества встраиваемых слов.

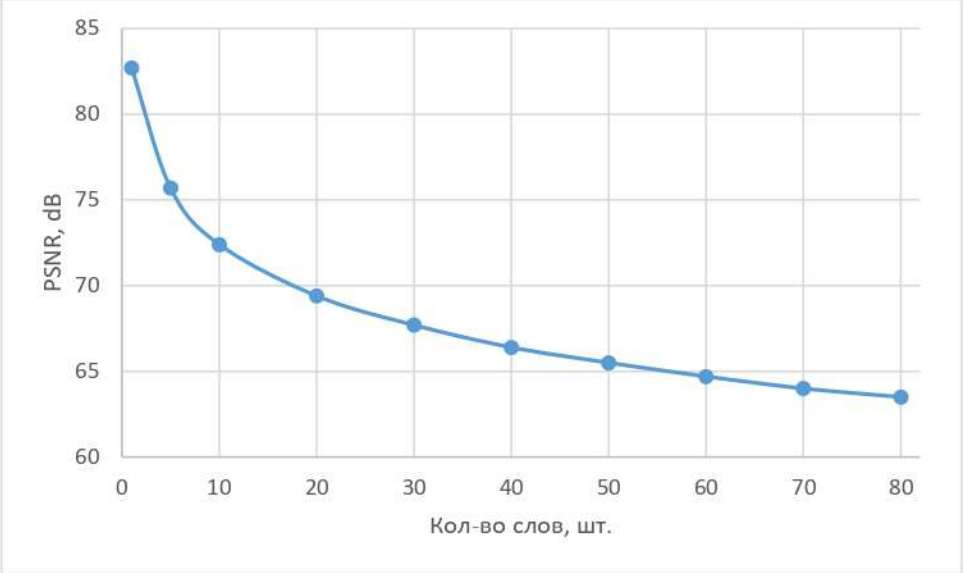

Рисунок 10. График зависимости параметра PSNR от количества встраиваемых слов.

График зависимости параметра PSNR от количества встраиваемых слов получился постоянно убывающим, напоминающим убывающий экспоненциальный график.

Это говорит о том, что чем больше длина встраиваемого сообщения в текст, тем больше шума наблюдается на изображении, так как параметр PSNR есть отношение максимально возможного сигнала к уровню шума.

Параметр RMSE.

Был программно реализован расчет параметра RMSE.
Встроим сообщение «стеганография» в изображение. Параметр RMSE равен: 0.020

Встроим по нарастающей 1-80 слов «стеганография» в изображение.
Построим график зависимости параметра RMSE от количества встраиваемых слов.

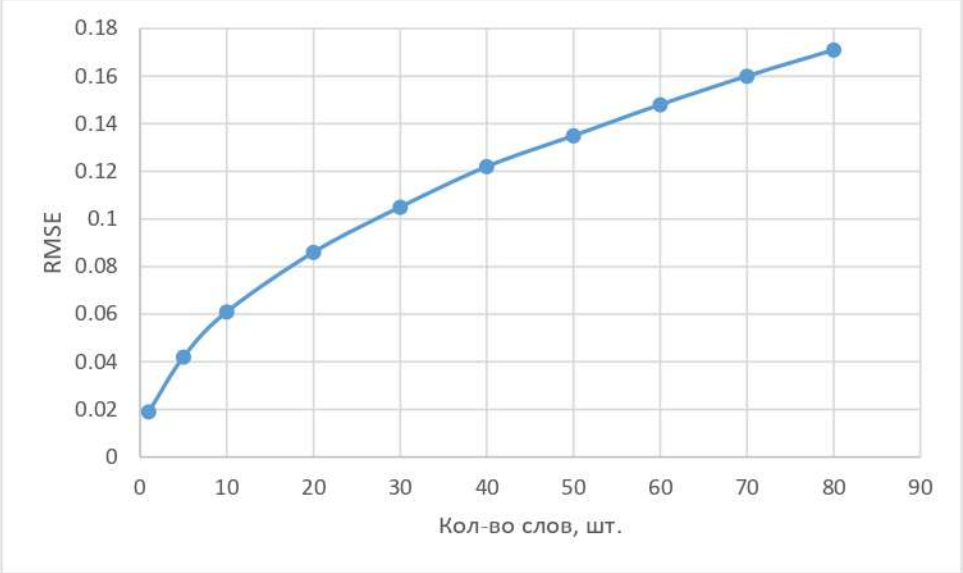

Рисунок 11. График зависимости параметра RMSE от количества встраиваемых слов.

.

График зависимости параметра RMSE от количества встраиваемых слов получился возрастающим, что свидетельствует о увеличении среднеквадратичного отклонения.

Оценка целесообразности.

Так как встраивание производилось только в 26 коэффициентов среднечастотной субполосы из 64 коэффициентов и коэффициент изменятся максимум на 1 единицу, то визуально данные шумы незаметны. При таком подходе в изображение с размером 320x200 пикселей возможно встроить сообщение длиной до 11142 символов. Так как этот метод не подразумевает прямую замену значения пикселей, а изменяет коэффициенты преобразования, то данный метод встраивания является довольно эффективным в плане объема встраивания, а также процент ошибочных символов до 6% при встраивании в определенные коэффициенты делает метод встраивания целесообразным.

Вывод:

В результате работы над программной реализацией метода встраивания и извлечения были приобретены навыки сокрытия информации в цифровых изображениях с помощью метода LSB в коэффициентах дискретного косинусного преобразования блоков изображения.

Встраивание сообщения в коэффициенты дискретно косинусного преобразования методом LSB является целесообразным для использования благодаря: вменяемой плотности встраивания (1 символ на 6 пикселей изображения), сложности обнаружения так как визуально искажения изображения незаметны, а также процент ошибочных символов до 6% при встраивании в определенные коэффициенты делает метод встраивания целесообразным. Однако, стоит учесть, то данный метод не гарантирует 100% совпадения исходного и извлеченного сообщений, что в некоторых случаях бывает необходимо.

Список использованной литературы:

1. Конспект лекций по дисциплине «Системы и сети телевидения» // Слёзкин В.Г., Гимпилевич Ю.Б. // Севастопольский национальный технический университет 2012г. // 50-55 с.
2. «Цифровая обработка изображений» // Р. Гонсалес, Р. Вудс // М.: Техносфера, 2005. –1072 с.
3. «Теория и применение цифровой обработки сигналов» // Рабинер Л., Гоулд Б. // М.: Мир.-1978.-848с.

Приложение 1. Листинг программного кода.

```
from PIL import Image
import math

img_in = img_out = 0
img_plain_path = r'C:\Users\stepa\Documents\ИТМО\Основы стеганографии\Лаба 3\img_plain.bmp'
img_encode_path = r'C:\Users\stepa\Documents\ИТМО\Основы стеганографии\Лаба 3\img_encode.bmp'
block_size = 8

# Проверка результата выполнения функции
def check_result(res, msg):
    if res == -1:
        print(msg)
        exit1()

# Завершеине работы при ошибке
def exit1():
    if img_in != 0:
        img_in.close()
    if img_out != 0:
        img_out.close()
    print('exit(-1)')
    exit(-1)

# Проверка возможности встраивания
def can_encode(msg_plain, block_size, img_plain_path):
    img, img_res = getPixels(img_plain_path)
    if len(msg_plain)*7+7 > (img_res[0]*img_res[1]/(block_size**2))*26:
        return -1
    return 0

# Создание списка пикселей
def getPixels(img_path):
    img = Image.open(img_path)
    img_res = img.size
    pixels = list(img.getdata())
    img.close()
    return pixels, img_res

# Создание изображения
def putPixels(img_path, pixels, img_res):
    img = Image.new('RGB', img_res)
    img.putdata(pixels)
    img.save(img_path)
    img.close()
    return 0

# Кодирование двоичной строки
def strEncode(msg_plain):
    msg_cypher = ''
    for pix_num in msg_plain:
        if pix_num == ' ':
            msg_cypher += '1111111'
        else:
            msg_cypher += format(ord(pix_num)-1024, '07b')
    msg_cypher += '0000000'
    return msg_cypher

# Декодирование двоичной строки
def strDecode(msg_cypher):
    str_temp = ''
    plain_word = ''
    for bit in range(len(msg_cypher)):
        str_temp += msg_cypher[bit]
        if len(str_temp) == 7:
            if str_temp == '1111111':
                plain_word += ' '
            else:
                plain_word += chr(int(str_temp, 2)+1024)
            str_temp = ''
    return plain_word
```

```
# Создание списков квадратных блоков их списка RGB пикселей
def tupleToBlocks(pixels, img_res, block_size):
    blocks_r = []
    blocks_g = []
    blocks_b = []
    color_id = 0
    block_temp = []
    for blocks_list in [blocks_r, blocks_g, blocks_b]:
        for column_num in range(0, img_res[0], block_size):
            for row_num in range(0, img_res[0]*img_res[1], img_res[0]):
                row_temp = []
                for pix_num in range(block_size):
                    row_temp.append(
                        pixels[column_num+row_num+pix_num][color_id])
                block_temp.append(row_temp)
                if len(block_temp) == block_size:
                    blocks_list.append(block_temp)
                    block_temp = []
        color_id += 1
    return blocks_b+blocks_g+blocks_r

# Создание списка RGB пикселей из списков квадратных блоков
def blocksToTuple(blocks, img_res, block_size):
    pixels = []
    blocks_b = blocks[0:int(len(blocks)/3)]
    blocks_g = blocks[int(len(blocks)/3):int(len(blocks)/3)*2]
    blocks_r = blocks[int(len(blocks)/3)*2:int(len(blocks))]
    pix_c = 0
    for column_num in range(int(img_res[1]/block_size)):
        for row_num in range(block_size):
            for block_num in range(0, len(blocks_r), int(img_res[1]/block_size)):
                for pix_num in range(block_size):
                    pix_c += 1
                    row_temp = (blocks_r[column_num+block_num][row_num][pix_num],
blocks_g[column_num + block_num][row_num][pix_num],
blocks_b[column_num+block_num][row_num][pix_num])
                    pixels.append(row_temp)
    pixels = tuple(pixels)
    return pixels

# Дискретно косинусное преобразование(ДКП)
def enDCT(block_plain):
    block_enDCTed = []
    for k in range(len(block_plain)):
        row_temp = []
        for l in range(len(block_plain)):
            block_sum = 0
            for m in range(len(block_plain)):
                for n in range(len(block_plain)):
                    block_sum += block_plain[m][n] *
math.cos(math.pi*k*((2*m+1)/(2*len(block_plain)))) *
math.cos(math.pi*l*((2*n+1)/(2*len(block_plain))))
            if k == 0 and l == 0:
                alpha = 1
            elif k == 0 or l == 0:
                alpha = math.sqrt(2)
            else:
                alpha = 2
            block_sum = (alpha*block_sum)/len(block_plain)
            row_temp.append(block_sum)
        block_enDCTed.append(row_temp)
    return block_enDCTed

# Обратное ДКП
def deDCT(block_enDCTed):
    block_deDCTed = []
    for m in range(len(block_enDCTed)):
        row_temp = []
        for n in range(len(block_enDCTed)):
            block_sum = 0
            for k in range(len(block_enDCTed)):
                for l in range(len(block_enDCTed)):
                    if k == 0 and l == 0:
                        alpha = 1
                    elif k == 0 or l == 0:
                        alpha = math.sqrt(2)
                    else:
                        alpha = 2
                    block_sum += (alpha/len(block_enDCTed)) * block_enDCTed[k][l] *
math.cos(math.pi*k*((2*m+1)/(2*len(block_enDCTed)))) *
math.cos(math.pi*l*((2*n+1)/(2*len(block_enDCTed))))
            row_temp.append(round(block_sum))
        block_deDCTed.append(row_temp)
    return block_deDCTed
```

```
# Рассчет параметра PSNR
def psnr(img_path_1, img_path_2):
    img_a = getPixels(img_path_1)[0]
    img_b = getPixels(img_path_2)[0]
    mse_r = mse_g = mse_b = mse_t = psnr = 0
    for y in range(len(img_a)):
        for x in range(3):
            if x == 0:
                mse_r += (int(img_b[y][x])-int(img_a[y][x]))**2
            elif x == 1:
                mse_g += (int(img_b[y][x])-int(img_a[y][x]))**2
            elif x == 2:
                mse_b += (int(img_b[y][x])-int(img_a[y][x]))**2
    mse_t = (mse_r+mse_g+mse_b)/(3*len(img_a))
    rmse = round(math.sqrt(mse_t), 3)
    psnr = round(10*math.log10((255**2)/mse_t), 3)
    return psnr, rmse

# Встраивание сообщения в коэффициенты ДКП
def encode(msg_plain, block_size, img_plain_path, img_encode_path):
    # Кодирование двоичной строки
    msg_cypher = strEncode(msg_plain)
    # Создание списка пикселей
    pixels, img_res = getPixels(img_plain_path)
    # Получение списков квадратных блоков
    blocks_plain = tupleToBlocks(pixels, img_res, block_size)
    # ДКП блоков
    blocks_enDCTed = []
    [blocks_enDCTed.append(enDCT(block)) for block in blocks_plain]
    # Встраивание
    block_row = [0,0,1,1,2,2,3,3,4,4,5,5,6,6,7,7]
    block_elem = [7,6,7,6,5,4,5,4,3,2,3,2,1,0,1,0]
    bit_num = 0
    for block_num in range(len(blocks_enDCTed)):
        for adr in range(16):
            if bit_num == len(msg_cypher):
                break
            if msg_cypher[bit_num] == '0':
                if round(blocks_enDCTed[block_num][block_row[adr]][block_elem[adr]]) % 2 == 1:
                    blocks_enDCTed[block_num][block_row[adr]][block_elem[adr]] -= 1
            elif msg_cypher[bit_num] == '1':
                if round(blocks_enDCTed[block_num][block_row[adr]][block_elem[adr]]) % 2 == 0:
                    blocks_enDCTed[block_num][block_row[adr]][block_elem[adr]] += 1
            bit_num += 1
    # Обратное ДКП
    blocks_deDCTed = []
    [blocks_deDCTed.append(deDCT(block)) for block in blocks_enDCTed]
    # Создание изображения
    pixels = blocksToTuple(blocks_deDCTed, img_res, block_size)
    putPixels(img_encode_path, pixels, img_res)
    return 0

# Извлечение сообщения из коэффициентов ДКП
def decode(img_encode_path, block_size):
    # Создание списка пикселей
    pixels, img_res = getPixels(img_encode_path)
    # Получение списков квадратных блоков
    blocks_encode = tupleToBlocks(pixels, img_res, block_size)
    # ДКП блоков
    blocks_enDCTed = []
    [blocks_enDCTed.append(enDCT(block)) for block in blocks_encode]
    # Извлечение двоичной строки
    msg_cypher = ''
    bit_count = zero_count = 0
    block_row = [0,0,1,1,2,2,3,3,4,4,5,5,6,6,7,7]
    block_elem = [7,6,7,6,5,4,5,4,3,2,3,2,1,0,1,0]
    for block_num in range(len(blocks_enDCTed)):
        for adr in range(16):
            if round(blocks_enDCTed[block_num][block_row[adr]][block_elem[adr]]) % 2 == 0:
                msg_cypher += '0'
                zero_count += 1
                bit_count += 1
            if round(blocks_enDCTed[block_num][block_row[adr]][block_elem[adr]]) % 2 == 1:
                msg_cypher += '1'
                bit_count += 1
            if bit_count == 7 and zero_count != 7:
                bit_count = 0
                zero_count = 0
            elif bit_count == 7 and zero_count == 7:
                msg_cypher = msg_cypher[:-7]
                break
    # Декодирование двоичной строки
    plain_word = strDecode(msg_cypher)
    return plain_word
```

```
# ------------------------------------------------------------------------------------------------
# Выбор встраивания/извлечения
option = input('Для встраивания введите 1, для извлечения введите 2: ')
if option not in '12':
    print('Неверный символ: ', option)
    exit1()
# Встраивание
if option == '1':
    # Ввод сообщения
    msg_plain = input('Введите сообщение для встраивания в текст: ')
    # Проверка возможности встраивания
    res = can_encode(msg_plain, block_size, img_plain_path)
    check_result(res, 'Нехватка блоков для встраивания')
    # Встраивание сообщения в коэффициенты ДКП
    res = encode(msg_plain, block_size, img_plain_path, img_encode_path)
    check_result(res, 'Не удалось выполнить встраивание')
    # Рассчет параметра PSNR
    psnrrmse = psnr(img_plain_path, img_encode_path)
    print('Встроено сообщение \"', msg_plain,
          '\" в изображение \"', img_encode_path, '\"', sep='')
    print('Параметр RMSE равен', psnrrmse[1])
    print('Параметр PSNR равен', psnrrmse[0], 'dB')
# Извлечение
elif option == '2':
    # Извлечение сообщения из коэффициентов ДКП
    res = decode(img_encode_path, block_size)
    check_result(res, 'Не удалось выполнить извлечение')
    print('Извлечено сообщение \"', res[:13],
          '\" из изображения \"', img_encode_path, '\"', sep='')
exit(0)
# ------------------------------------------------------------------------------------------------
```