Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ № 4
по дисциплине «Методы защиты мультимедиа-данных»
ТЕМА РАБОТЫ

Студент гр. БПИ196
_____ Е.Н. Мосолков
«___» _____ 2022 г.

Руководитель
МНС кафедры информационной безопасности киберфизических систем
_____А.С. Мельман
«___» _____ 2022 г.

Москва 2022

СОДЕРЖАНИЕ

1 Задание на практическую работу

Целью работы является приобретение навыков программной реализации встраивания информации в цифровые изображения, сжатые по методу JPEG.

В рамках практической работы необходимо выполнить следующее:

1. Написать программную реализацию рассмотренных методов встраивания информации в JPEG-изображения (всех четырёх);
2. Провести вычислительные эксперименты с полученной программной реализацией и сделать выводы об эффективности рассмотренных методов встраивания;
3. Подготовить отчёт о выполнении работы.

2 Краткая теоретическая часть

Из-за большой популярности JPEG [1, 2], встраивание информации в изображения этого формата широко распространено. Это может быть как встраивание водяного знака для подтверждения авторства или защиты интеллектуальной собственности, так и стеганографическое встраивание информации с целью сокрытия факта ее передачи и/или хранения. Некоторые алгоритмы предусматривают встраивание информации на этапе сжатия, однако большинство схем позволяет использовать уже сжатые JPEG-изображения как входные данные. Чаще всего при встраивании изменению подвергаются квантованные коэффициенты ДКП. На этапе квантования каждый из 64 вещественных коэффициентов ДКП в блоке размером 8×8 делится на соответствующее число матрицы квантования, а после округляется до целого. При этом высокочастотные коэффициенты подвергаются более сильному квантованию, чем низкочастотные. В результате квантования в блоке коэффициентов ДКП остаётся всего несколько ненулевых коэффициентов, которые сконцентрированы в левом верхнем углу матрицы (рисунок 1).

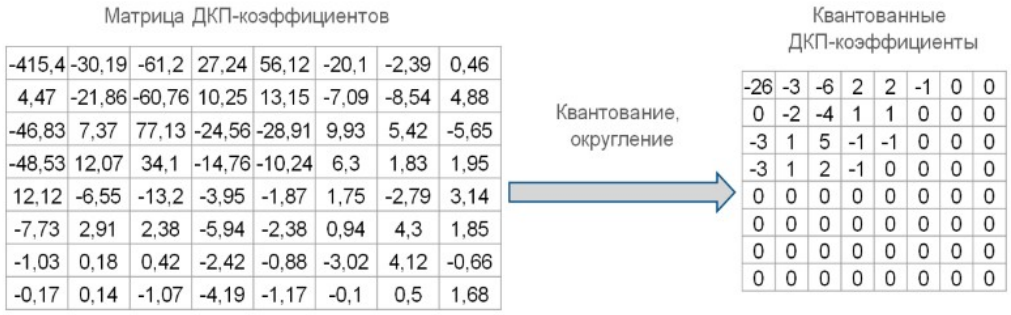

Рисунок 1 – Квантование

Изменение DC-коэффициента (верхний левый угол) приводит к значительной деградации изображения, поэтому встраивание информации осуществляется только в AC-коэффициенты. Стоит отметить, что для реализации алгоритма сокрытия данных в JPEG-изображениях нет необходимости в собственноручной реализации кодера JPEG. Программные библиотеки позволяют изменять нужные коэффициенты и записывать их в JPEG-файл средствами готовых функций. Рассмотрим примеры известных методов встраивания дополнительной информации в JPEG-изображения [3].

2.1 JSTEG

JSteg – это аналог метода LSB для частотной области. Он заключается в замене наименее значимых битов квантованных коэффициентов ДКП на биты секретного сообщения. DC-коэффициенты и коэффициенты, равные 0 и, для встраивания информации не используются, т.к. их изменение повлечёт за собой существенные искажения стегоизображения, а главное – возникнет неоднозначность при извлечении

данных. Получатель сообщения не может знать наверняка, было ли данное значение коэффициента нулевым изначально или приобрело это значение в результате встраивания информации в коэффициент со значением , поэтому данные значения пропускаются как на этапе встраивания, так и на этапе извлечения. Таким образом, встраивание осуществляется по следующей формуле:

$$C = (c_7 c_6 \dots c_0)_2;\ C' = \text{sign}\ C \times (c_7 c_6 \dots c_0')_2;\ c_0' = \begin{cases} 0, & \text{если } b_i = 0; \\ 1, & \text{если } b_i = 1, \end{cases}$$

где C – коэффициент до встраивания, C – коэффициент после встраивания, – бит сообщения. Обратите внимание, что знак коэффициента ДКП не изменяется при встраивании информации. Процедура извлечения совпадает с таковой в методе LSB, рассмотренном ранее для пространственной области.

2.2 F3

Алгоритм F3 также встраивает информацию в квантованные ненулевые AC-коэффициенты ДКП изображений JPEG. Основной принцип, напоминает идею метода PM1 для пространственной области, однако в данном случае изменяемый коэффициент во всех случаях подвергается уменьшению:

$$C' = \begin{cases} sign\ C \times (|C| - 1), \text{ если } |C| \bmod 2 \neq 0 \text{ и } b_i = 0 \text{ или} |C| \bmod 2 = 0 \text{ и } b_i = 1; \\ C, \text{ иначе.} \end{cases}$$

Для борьбы с ошибками извлечения, возникающими по причине неоднозначной интерпретации нулевого значения, алгоритм F3 использует процедуру «сокращения». Если в результате встраивания нулевого бита коэффициент ДКП приобрёл значение 0, то необходимо повторно выполнить встраивание того же бита в ближайший ненулевой коэффициент. Если в результате встраивания вновь возникает значение 0, процедура повторяется. Таким образом, в результирующем стегоизображении все «проблемные» коэффициенты принимают нулевые значения. При извлечении чётные коэффициенты (по модулю) соответствуют нулевому биту, нечётные – единичному.

2.3 F4

Ещё одна разновидность подобных методов – алгоритм F4. В отличие от F3, F4 дублирует принцип метода PM1, однако, кроме проверки коэффициента на чётность, он требует проверки на знак изменяемого значения:

$$C' = \begin{cases} C + 1, \text{ если} C < 0 \text{ и } (|C| \bmod 2 \neq 0 \text{ и } b_i = 1) \text{ или} (|C| \bmod 2 = 0 \text{ и } b_i = 0); \\ C - 1, \text{ если} C > 0 \text{ и } (|C| \bmod 2 \neq 0 \text{ и } b_i = 0) \text{ или} (|C| \bmod 2 = 0 \text{ и } b_i = 1); \\ C, \text{ иначе.} \end{cases}$$

Также как и предшественник, F4 применяет процедуру «сокращения» для борьбы с неоднозначностью извлечения. Если в результате встраивания бита значение AC-коэффициента становится равно 0, необходимо выполнить повторное встраивание этого

бита в следующий ненулевой коэффициент. Извлечение выполняется по следующей формуле:

$$b_i = \begin{cases} 0, & \text{если } ((C' > 0) \text{ и } (|C| \bmod 2 = 0)) \text{ или } ((C' < 0) \text{ и } (|C| \bmod 2 \neq 0)); \\ 1, & \text{если } ((C' > 0) \text{ и } (|C| \bmod 2 \neq 0)) \text{ или } ((C' < 0) \text{ и } (|C| \bmod 2 = 0)). \end{cases}$$

2.4 F5

Согласно F5, перед началом встраивания необходимо выполнить перестановку коэффициентов ДКП в соответствии с псевдослучайной последовательностью. Эта последовательность является секретным ключом, зная который, впоследствии её можно будет повторить для извлечения секретных данных. В рамках настоящей работы данный пункт может быть опущен. Сообщение встраивается в ненулевые AC-коэффициенты с помощью матрицы вложения, которая минимизирует число модификаций, вносимых в изображения. В процессе встраивания k битов сообщения вставляются в одну группу коэффициентов путем уменьшения абсолютного значения не более чем одного коэффициента из каждой группы. Для примера рассмотрим 2 частных случая.

Пример 1. Необходимо встроить 2 бита (x_1, x_2) в 3 коэффициента (a_1, a_2, a_3). Для этого нужно сопоставить значения битов сообщения с младшими битами коэффициентов (по модулю, после встраивания знак сохраняется):

1) Если $x_1 = a_1 \oplus a_3$ и $x_2 = a_2 \oplus a_3$, то изменения коэффициентов не требуются.
2) Если $x_1 \neq a_1 \oplus a_3$ и $x_2 = a_2 \oplus a_3$, то необходимо изменить младший бит a_1.
3) Если $x_1 = a_1 \oplus a_3$ и $x_2 \neq a_2 \oplus a_3$, то необходимо изменить младший бит a_2.
4) Если $x_1 \neq a_1 \oplus a_3$ и $x_2 \neq a_2 \oplus a_3$, то необходимо изменить младший бит a_3.

Пример 2. Необходимо встроить 3 бита (x_1, x_2, x_3) в 7 коэффициентов $(a_1, a_2, a_3, a_4, a_5, a_6, a_7)$:

1) Если $x_1 = a_1 \oplus a_3 \oplus a_5 \oplus a_7$, $x_2 = a_2 \oplus a_3 \oplus a_6 \oplus a_7$ и $x_3 = a_4 \oplus a_5 \oplus a_6 \oplus a_7$, то изменения коэффициентов не требуются.
2) Если $x_1 \neq a_1 \oplus a_3 \oplus a_5 \oplus a_7$, $x_2 = a_2 \oplus a_3 \oplus a_6 \oplus a_7$ и $x_3 = a_4 \oplus a_5 \oplus a_6 \oplus a_7$, то необходимо изменить младший бит a_1.
3) Если $x_1 = a_1 \oplus a_3 \oplus a_5 \oplus a_7$, $x_2 \neq a_2 \oplus a_3 \oplus a_6 \oplus a_7$ и $x_3 = a_4 \oplus a_5 \oplus a_6 \oplus a_7$, то необходимо изменить младший бит a_2.
4) Если $x_1 \neq a_1 \oplus a_3 \oplus a_5 \oplus a_7$, $x_2 \neq a_2 \oplus a_3 \oplus a_6 \oplus a_7$ и $x_3 = a_4 \oplus a_5 \oplus a_6 \oplus a_7$, то необходимо изменить младший бит a_3.
5) Если $x_1 = a_1 \oplus a_3 \oplus a_5 \oplus a_7$, $x_2 = a_2 \oplus a_3 \oplus a_6 \oplus a_7$ и $x_3 \neq a_4 \oplus a_5 \oplus a_6 \oplus a_7$, то необходимо изменить младший бит a_4.
6) Если $x_1 \neq a_1 \oplus a_3 \oplus a_5 \oplus a_7$, $x_2 = a_2 \oplus a_3 \oplus a_6 \oplus a_7$ и $x_3 \neq a_4 \oplus a_5 \oplus a_6 \oplus a_7$, то необходимо изменить младший бит a_5.
7) Если $x_1 = a_1 \oplus a_3 \oplus a_5 \oplus a_7$, $x_2 \neq a_2 \oplus a_3 \oplus a_6 \oplus a_7$ и $x_3 \neq a_4 \oplus a_5 \oplus a_6 \oplus a_7$, то необходимо изменить младший бит a_6.
8) Если $x_1 \neq a_1 \oplus a_3 \oplus a_5 \oplus a_7$, $x_2 \neq a_2 \oplus a_3 \oplus a_6 \oplus a_7$ и $x_3 \neq a_4 \oplus a_5 \oplus a_6 \oplus a_7$, то необходимо изменить младший бит a_7.

Обратите внимание, что данный алгоритм предусматривает процедуру «сокращения», как и у его предшественников.

Для извлечения информации, встроенной по алгоритму F5, необходимо вычислить значения секретных битов x, по тем же формулам, которые использовались для проверки соответствия при встраивании.

3 Программная реализация

Программа реализована как восемь ключевых функций (по 2 на каждый метод встраивания изображения) и главной функцией, которая ведет диалог с пользователем, выбирает функцию, которую нужно запустить и показывает информацию о процедуре.

Весь код написан с помощью Python в окружении Jupyter notebook.

1. JSTEG

```
def embed_JSTEG(jpeg, secret=''):
    global embed_capacity, capacity
    cnt = -1
    bits = ''
    for i in range(0, jpeg.coef_arrays[0].shape[0]):
        for j in range(0, jpeg.coef_arrays[0].shape[1]):
            if i % 8 == j % 8 == 0 and abs(jpeg.coef_arrays[0][i][j]) <= 1:
                continue
            curr = None
            cnt += 1
            if secret == '':
                curr = str(random.randint(0, 1))
                bits += curr
            else:
                if len(secret) <= cnt:
                    continue
                curr = secret[cnt]
            if int(curr) != (abs(jpeg.coef_arrays[0][i][j]) % 2):
                if jpeg.coef_arrays[0][i][j] > 0:
                    jpeg.coef_arrays[0][i][j] ^= 1
                else:
                    jpeg.coef_arrays[0][i][j] = -abs(jpeg.coef_arrays[0][i][j])^1
    if secret == '':
        embed_capacity = 1
        print(f'{cnt + 1} bit\n{bits[:50]}')
    else:
        print(f'Applied {len(secret)} bit.')
        embed_capacity = len(secret) / (cnt + 1)
    capacity = cnt + 1
```

Рисунок 1. Код функции встраивания изображения в jpeg методом JSTEG

```
def extract_JSTEG(jpeg, size):
    cnt = -1
    res = ''
    for i in range(0, jpeg.coef_arrays[0].shape[0]):
        for j in range(0, jpeg.coef_arrays[0].shape[1]):
            if i % 8 == j % 8 == 0 and abs(jpeg.coef_arrays[0][i][j]) <= 1:
                continue
            cnt += 1
            if cnt == size:
                return res
            if abs(jpeg.coef_arrays[0][i][j]) & 1:
                res += '1'
            else:
                res += '0'
    return res
```

Рисунок 2. Код функции извлечения изображения из jpeg методом JSTEG

2. F3

```
def embed_F3(jpeg, secret=''):
    global embed_capacity, capacity
    cnt = -1
    bits = ''
    fail = False
    for i in range(0, jpeg.coef_arrays[0].shape[0]):
        for j in range(0, jpeg.coef_arrays[0].shape[1]):
            if i % 8 == j % 8 == 0 and jpeg.coef_arrays[0][i][j] == 0:
                continue
            if not fail:
                cnt += 1
                curr = None
                if secret == '':
                    curr = str(random.randint(0, 1))
                    bits += curr
                else:
                    if cnt >= len(secret):
                        continue
                    curr = secret[cnt]
                if int(curr) != abs(jpeg.coef_arrays[0][i][j]) % 2:
                    if jpeg.coef_arrays[0][i][j] > 0:
                        jpeg.coef_arrays[0][i][j] -= 1
                    else:
                        jpeg.coef_arrays[0][i][j] = -abs(jpeg.coef_arrays[0][i][j]) - 1
                fail = jpeg.coef_arrays[0][i][j] == 0
    if secret == '':
        embed_capacity = 1
        print(f'{cnt + 1 - int(fail)} bit:\n{bits[:50]}')
    else:
        print(f'{len(secret)} bit')
        embed_capacity = len(secret) / (cnt + 1)
    capacity = cnt + 1 - int(fail)
```

Рисунок 3. Код функции встраивания изображения в jpeg методом F3

```
def extract_F3(jpeg, size):
    cnt = -1
    res = ''
    for i in range(0, jpeg.coef_arrays[0].shape[0]):
        for j in range(0, jpeg.coef_arrays[0].shape[1]):
            if i % 8 == j % 8 == 0 and jpeg.coef_arrays[0][i][j] == 0:
                continue
            cnt += 1
            if cnt == size:
                return res
            if abs(jpeg.coef_arrays[0][i][j]) & 1:
                res += '1'
            else:
                res += '0'
    return res
```

Рисунок 4. Код функции извлечения изображения из jpeg методом F3

3. F4

```
def embed_F4(jpeg, secret=''):
    global embed_capacity, capacity
    cnt = -1
    bits = ''
    fail = False
    for i in range(0, jpeg.coef_arrays[0].shape[0]):
        for j in range(0, jpeg.coef_arrays[0].shape[1]):
            if i % 8 == j % 8 == 0 and jpeg.coef_arrays[0][i][j] == 0:
                continue
            if not fail:
                curr = None
                cnt += 1
                if secret == '':
                    curr = str(random.randint(0, 1))
                    bits += curr
                else:
                    if cnt >= len(secret):
                        continue
                    curr = secret[cnt]

            c1 = jpeg.coef_arrays[0][i][j] < 0 and abs(jpeg.coef_arrays[0][i][j]) % 2 != 0 and curr == '1'
            c2 = abs(jpeg.coef_arrays[0][i][j]) % 2 == 0 and curr == '0'
            c3 = jpeg.coef_arrays[0][i][j] > 0 and abs(jpeg.coef_arrays[0][i][j]) % 2 != 0 and curr == '0'
            c4 = abs(jpeg.coef_arrays[0][i][j]) % 2 == 0 and curr == '1'
            if c1 or c2:
                jpeg.coef_arrays[0][i][j] += 1
            elif c3 or c4:
                jpeg.coef_arrays[0][i][j] -= 1
            fail = jpeg.coef_arrays[0][i][j] == 0
    if secret == '':
        embed_capacity = 1
        print(f'{cnt + 1 - int(fail)} bit\n{bits[:50]}')
    else:
        print(f'{len(secret)} bit')
        embed_capacity = len(secret) / (cnt + 1)
    capacity = cnt + 1 - int(fail)
```

Рисунок 5. Код функции встраивания изображения в jpeg методом F4

```
def extract_F4(jpeg, size):
    cnt = -1
    res = ''
    for i in range(0, jpeg.coef_arrays[0].shape[0]):
        for j in range(0, jpeg.coef_arrays[0].shape[1]):
            if i % 8 == j % 8 == 0 and jpeg.coef_arrays[0][i][j] == 0:
                continue
            cnt += 1
            if cnt == size:
                return res
            c1 = jpeg.coef_arrays[0][i][j] < 0 and abs(jpeg.coef_arrays[0][i][j]) % 2 == 0
            c2 = jpeg.coef_arrays[0][i][j] > 0 and abs(jpeg.coef_arrays[0][i][j]) % 2 != 0
            c3 = jpeg.coef_arrays[0][i][j] < 0 and abs(jpeg.coef_arrays[0][i][j]) % 2 != 0
            c4 = jpeg.coef_arrays[0][i][j] > 0 and abs(jpeg.coef_arrays[0][i][j]) % 2 == 0
            if c1 or c2:
                res += '1'
            elif c3 or c4:
                res += '0'
    return res
```

Рисунок 6. Код функции извлечения изображения из jpeg методом F4

4. F5

```
def embed_F5(jpeg, secret=''):
    global embed_capacity, capacity
    cnt = -1
    bits = ''
    fail = False
    buffer = []
    for i in range(0, jpeg.coef_arrays[0].shape[0]):
        for j in range(0, jpeg.coef_arrays[0].shape[1]):
            if i % 8 == j % 8 == 0 and jpeg.coef_arrays[0][i][j] == 0:
                continue
            if not fail and len(buffer) == 0:
                curr = next = None
                if bits == '':
                    curr = str(random.randint(0, 1))
                    next = str(random.randint(0, 1))
                    bits += curr
                    bits += next
                    cnt += 2
                else:
                    cnt += 1
                    if cnt >= len(bits):
                        continue
                    if cnt == len(bits) - 1:
                        cnt += 1
                    else:
                        curr = bits[cnt]
                        cnt += 1
                        next = bits[cnt]
            buffer.append((i, j))
            if len(buffer) == 3:
                if curr is None:
                    fail = True
                    buffer = []
                    continue
                c1 = int(curr) != (get_abs(0, buffer, jpeg) ^ get_abs(1, buffer, jpeg)) & 1
                c2 = int(next) == (get_abs(1, buffer, jpeg) ^ get_abs(2, buffer, jpeg)) & 1
                if c1 and c2:
                    jpeg.coef_arrays[0][buffer[0][0]][buffer[0][1]] = xor(jpeg.coef_arrays[0][buffer[0][0]][buffer[0][1]])
                elif not c1 and not c2:
                    jpeg.coef_arrays[0][buffer[1][0]][buffer[1][1]] = xor(jpeg.coef_arrays[0][buffer[1][0]][buffer[1][1]])
                elif c1 and not c2:
                    jpeg.coef_arrays[0][buffer[2][0]][buffer[2][1]] = xor(jpeg.coef_arrays[0][buffer[2][0]][buffer[2][1]])

                fail = False
                for k in buffer:
                    if jpeg.coef_arrays[0][k[0]][k[1]] == 0:
                        fail = True
                if fail:
                    for k in buffer:
                        jpeg.coef_arrays[0][k[0]][k[1]] = 0
                buffer = []
    if secret == '':
        embed_capacity = 1
        print(f'{cnt + 1 - int(fail) * 2} bit\n{bits[:50]}')
    else:
        print(f'{len(secret) - (len(secret) % 2)} bit.')
        embed_capacity = len(secret) / (cnt + 1)
    capacity = cnt + 1 - int(fail) * 2
```

Рисунок 7. Код функции встраивания изображения в jpeg методом F5

```python
def extract_F5(jpeg, size):
    buffer = []
    cnt = -1
    res = ''
    for i in range(0, jpeg.coef_arrays[0].shape[0]):
        for j in range(0, jpeg.coef_arrays[0].shape[1]):
            if i % 8 == j % 8 == 0 and jpeg.coef_arrays[0][i][j] == 0:
                continue
            if len(buffer) == 0:
                cnt += 2
            if cnt >= size:
                return res
            buffer.append((i, j))
            if len(buffer) == 3:
                res += res_xor(get_abs(0, buffer, jpeg), get_abs(2, buffer, jpeg))
                res += res_xor(get_abs(1, buffer, jpeg), get_abs(2, buffer, jpeg))
                buffer = []
    return res
```

Рисунок 8. Код функции извлечения изображения из jpeg методом F5

5. Main

```
def main():
    path = 'image.jpg'
    watermarked_path = 'watermarked_image.jpg'
    choice = input('Type 1 to apply watermark or 2 to extract watermark: ')
    alghoritm = input('Choose alghoritm:\n\t1. JSTEG\n\t2. F3\n\t3. F4\n\t4. F5\nYour choice: ')
    if choice == '1':
        data = []
        mse = 0
        secret = input("Input watermark as a binary number: ")
        jpeg = jio.read(path)
        for i in range(jpeg.coef_arrays[0].shape[0]):
            for j in range(jpeg.coef_arrays[0].shape[1]):
                if i % 8 == j % 8 == 0: continue
                data.append(jpeg.coef_arrays[0][i][j])
        if alghoritm == '1':
            embed_JSTEG(jpeg, secret)
        elif alghoritm == '2':
            embed_F3(jpeg, secret)
        elif alghoritm == '3':
            embed_F4(jpeg, secret)
        elif alghoritm == '4':
            embed_F5(jpeg, secret)
        jio.write(jpeg, watermarked_path)
        watermarked_jpeg = jio.read(watermarked_path)
        image = np.array(Image.open(path))
        watermarked_image = np.array(Image.open(watermarked_path))
        for i in range(image.shape[0]):
           for j in range(image.shape[1]):
               for k in range(image.shape[2]):
                   mse += (image[i][j][k] - watermarked_image[i][j][k])**2
        mse /= (image.shape[0] * image.shape[1] * image.shape[2])
        psnr = 10 * math.log10(255**2 / mse)
        print(f'Capacity: {capacity}\nMSE: {mse}\nPSNR: {psnr}\nEC: {embed_capacity}')
    elif choice == '2':
        jpeg = jio.read(path)
        secret = ''
        size = int(input('Input size: '))
        dwm = ''.join([str(random.randint(0, 1)) for i in range(size)])
        b = 0
        if alghoritm == '1':
            secret = extract_JSTEG(jpeg, size)
        elif alghoritm == '2':
            secret = extract_F3(jpeg, size)
        elif alghoritm == '3':
            secret = extract_F4(jpeg, size)
        elif alghoritm == '4':
            secret = extract_F5(jpeg, size)
        ncc1 = ncc2 = ncc3 = 0
        for i in range(min(len(secret), size)):
            if secret[i] != dwm[i]:
                b += 1
            x = ord(secret[i]) - ord('0')
            y = ord(dwm[i]) - ord('0')
            ncc1 += x * y
            ncc2 += x**2
            ncc3 += y**2
        print(f'NCC: {ncc1 / (ncc2**0.5 * ncc3**0.5)}\nBER: {b / size}')
        print(secret) if len(secret) <= 50 else print(secret[:50])
    else:
        print('Invalid argument')
```

Рисунок 9. Код основной функции

4 Результаты экспериментов

Проведем эксперименты со следующими вводными данными на каждый метод встраивания водяного знака в изображение.

Как изображение возьмем следующую последовательность бит:

1111111110101010010101001010100101010010010101001010100101010100101010010100000111100010010101101010101001010101001010100101010

Получаем следующие выводы работы программы с различными методами встраивания:

```
Type 1 to apply watermark or 2 to extract watermark:  1
Choose alghoritm:
        1. JSTEG
        2. F3
        3. F4
        4. F5
Your choice:  1
Input watermark as a binary number:  1111111110101010010101001010100101010010010101001(
Applied 122 bit.
Capacity: 4879949
MSE: 0.001300452116728319
PSNR: 76.9898599497309
EC: 2.5000261273222324e-05
```

Рисунок 10. Результат встраивания JSTEG

```
Type 1 to apply watermark or 2 to extract watermark:  1
Choose alghoritm:
        1. JSTEG
        2. F3
        3. F4
        4. F5
Your choice:  2
Input watermark as a binary number:  1111111110101010010101001010100101010010010101001(
122 bit
Capacity: 4879568
MSE: 0.001282367447595561
PSNR: 77.05067875780286
EC: 2.500221331068652e-05
```

Рисунок 11. Результат встраивания F3

```
Type 1 to apply watermark or 2 to extract watermark:  1
Choose alghoritm:
        1. JSTEG
        2. F3
        3. F4
        4. F5
Your choice:  3
Input watermark as a binary number:  1111111110101010010101001010100101010010010101001(
122 bit
Capacity: 4879967
MSE: 0.0026781750924784217
PSNR: 73.85241394162257
EC: 2.5000169058520273e-05
```

Рисунок 12. Результат встраивания F4

```
Type 1 to apply watermark or 2 to extract watermark:  1
Choose alghoritm:
        1. JSTEG
        2. F3
        3. F4
        4. F5
Your choice:  4
Input watermark as a binary number:  1111111110101010010101001010100101010010010101001(
122 bit.
Capacity: 4223101
MSE: 2197.424753596383
PSNR: 14.711663484110122
EC: 2.8888724186326587e-05
```

Рисунок 13. Результат встраивания F5

Заметим, что наибольшая средняя квадратичная ошибка у метода F5, при этом остальные методы по данному значению близятся к нулю. Зато F5 достигает минимального пикового соотношения шум сигнал. Также данные говорят о том, что метод F5 занимает больше всего места.

Посмотрим теперь на результаты извлечения изображения

```
Type 1 to apply watermark or 2 to extract watermark:  2
Choose alghoritm:
        1. JSTEG
        2. F3
        3. F4
        4. F5
Your choice:  1
Input size:  121
NCC: 0.3550234734023465
BER: 0.4462809917355372
1000000010000000100000001000000010000000100000010
```

Рисунок 14. Результат извлечения JSTEG

```
Type 1 to apply watermark or 2 to extract watermark:  2
Choose alghoritm:
        1. JSTEG
        2. F3
        3. F4
        4. F5
Your choice:  2
Input size:  121
NCC: 0.22978625750451445
BER: 0.49586776859504134
10000000100000001000000010000000100000001000000010
```

Рисунок 15. Результат извлечения F3

```
Type 1 to apply watermark or 2 to extract watermark:  2
Choose alghoritm:
        1. JSTEG
        2. F3
        3. F4
        4. F5
Your choice:  3
Input size:  121
NCC: 0.75
BER: 0.05785123966942149
1111111111111111
```

Рисунок 16. Результат извлечения F4

```
Type 1 to apply watermark or 2 to extract watermark:  2
Choose alghoritm:
        1. JSTEG
        2. F3
        3. F4
        4. F5
Your choice:  4
Input size:  121
NCC: -0.7246046307102528
BER: 0.9917355371900827
□□□□□□□□□□□□□□□
```

Рисунок 17. Результат извлечения F5

5 Выводы о проделанной работе

В рамках данной лабораторной работы я изучил и применил на практике навыки встраивания информации в цифровые изображения, сжатые по методу JPEG.