МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ

ТЕХНОЛОГИЙ

Основы стеганографии

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

«Встраивание информации в картинки»

Работу выполнил студент

группы N3352, Нгуен Тхай Хынг

Работу проверил: ассистент ФБИТ,

Университет ИТМО,

Давыдов Вадим Валерьевич

Санкт-Петербург

2020

I. Цель работы

Встраивание текстовой информации в картинки, используя метода LSB (наименьший значимый бит). Файл картики в формате .bmp, цвет представлен сочетанием трех цветовых компонент RGB.

Извлечение тектовой информации из стегоконтейнера.

Построение графика значений PSNR исходного изображения и изображения с информацией, встроенные по наростающей один символ, 5, 10, 20, 30, 40,50.

Проведение простого атака.

Оценка целесообразности метода с реальными примерами.

II. Теория

Наименьший значимый бит или НЗБ (Least Significant Bit - LSB) - это позиция бита в двоичном целом числе, определяющая, является ли число четным или нечетным, когда это двоичное число преобразуется в десятичное [1].

Метод наименьшего значимого бита (LSB) является одним из стеганографических методов, в котором наименьший значащий бит изображения заменяется битом данных [2].

В этом работе, используется метод замены наименьшего значащего бита с картинками в формате ".bmp". Формат файла BMP, также известный как файл растрового изображения или формат файла растрового изображения, не зависящий от устройства, или просто растровое изображение - это формат файла растрового графического изображения, используемый для хранения растровых цифровых изображений независимо от устройства отображения, особенно в Microsoft Windows операционные системы [3].

RGB (Red, Green, Blue – красный, зеленый, синий) – аддитивная цветовая модель, описывающая способ кодирования цвета для цветовоспроизведения с помощью трех цветов, которые принято называть основными [4].

Для оценки качества изображения RGB (три канала в одной пиксели) используют меру среднеквадратического искажения (Mean Squared Error - MSE) [5].

$$\text{MSE} = \frac{1}{mnt} \sum_{i=0}^{i=m-1} \sum_{j=0}^{j=n-1} \sum_{k=0}^{k=t-1} [I(i,j,k) - K(i,j,k)]^2$$

Где:

m,n – размер изображения.

k – количество каналов в одной писели.

I, K – значение каналов пикселей исходного и выходного изображения.

Пиковое отношение сигнала к шуму обозначается аббревиатурой PSNR и является инженерным термином, означающим соотношение между максимумом возможного значения сигнала и мощностью шума, искажающего значения сигнала [5].

$$\text{PSNR} = 10\log_{10}(\frac{MAX_I^2}{MSE})$$

Где:

MAX_I - это максимальное значение, принимаемое пикселем изображения. Когда пиксели имеют разрядность 8 бит, MAXI = 255.

III. Практика

Для выполнения этой лабораторной работы, я написал 3 программы на языке Python 3.8 с помощью IDE Visual Studio Code. В программах я использовал библиотеку PIL для обработки файла изображения.

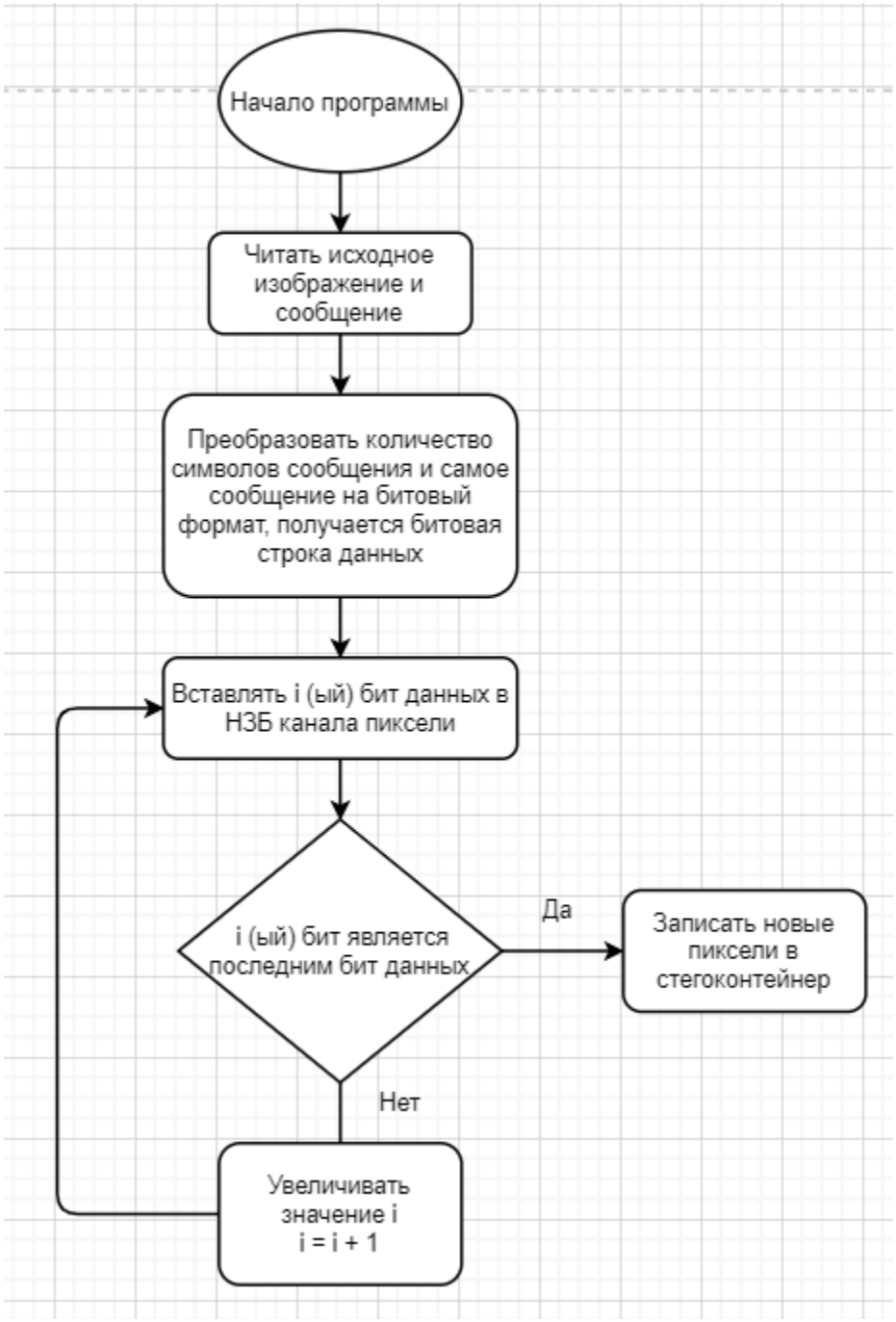
- Программа “image_stegano.py” для встраивания и извлечения сообщения

- Программа “psnr.py” для расчета значений PSNR

- Программа “detect.py” для провеления простого атака

Часть 1:

1. Блок-схема алгоритма встраивания:



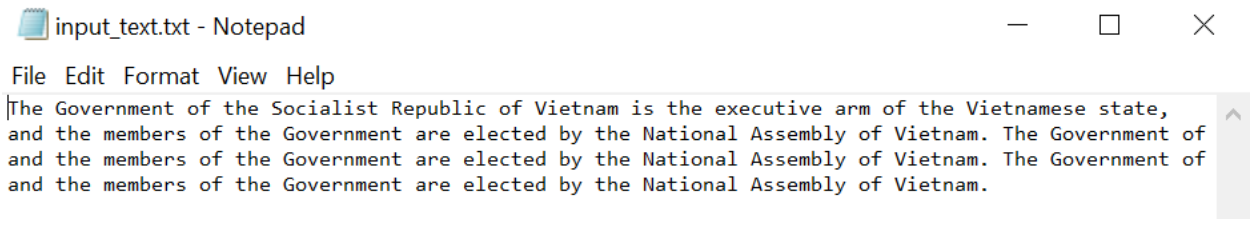
Рисунка 1. Блок-схема алгоритма встраивания

2. Блок-схема алгоритма извлечения:

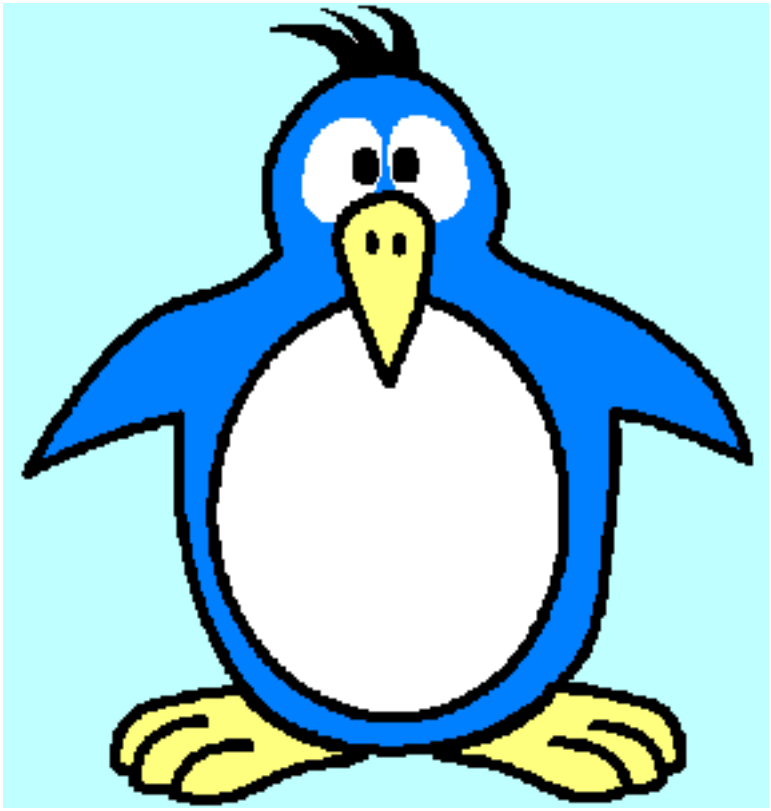
Рисунка 2. Блок-схема алгоритма извлечения

3. Встраивание и извлечение:

- Секретное сообщение, которое мы хотим встраивать в файле “input_text.txt”:



- Исходное изображение “input_image.bmp”:



Рисунка 3. Исходное изображение

- Запуск программы “image_stegano.py”:

```
PS E:\Documents\ITMO\steganography\Lab 2> py .\image_stegano.py
1. Encode, 2. Decode: 1
ENCODING ...
Input text file (input_text.txt): input_text.txt
Source image file (input_image.bmp): input_image.bmp
Destination image file (output_image.bmp): output_image.bmp
Encoded succesfully
PS E:\Documents\ITMO\steganography\Lab 2>
```

Рисунка 4. Прогамма встроила сообщение успешно

```
PS E:\Documents\ITMO\steganography\Lab 2> py .\image_stegano.py
1. Encode, 2. Decode: 2
DECODING ...
Input image file (output_image.bmp): output_image.bmp
Output text file (output_text.txt): output_text.txt
Decoded successfully
PS E:\Documents\ITMO\steganography\Lab 2> cat .\output_text.txt
The Government of the Socialist Republic of Vietnam is the executive arm of the Vietnamese state, and the members of the Governme
nt are elected by the National Assembly of Vietnam.
PS E:\Documents\ITMO\steganography\Lab 2>
```

Рисунка 5. Прогамма извлекла сообщение успешно

Часть 2:

1. Построение графика PSNR:

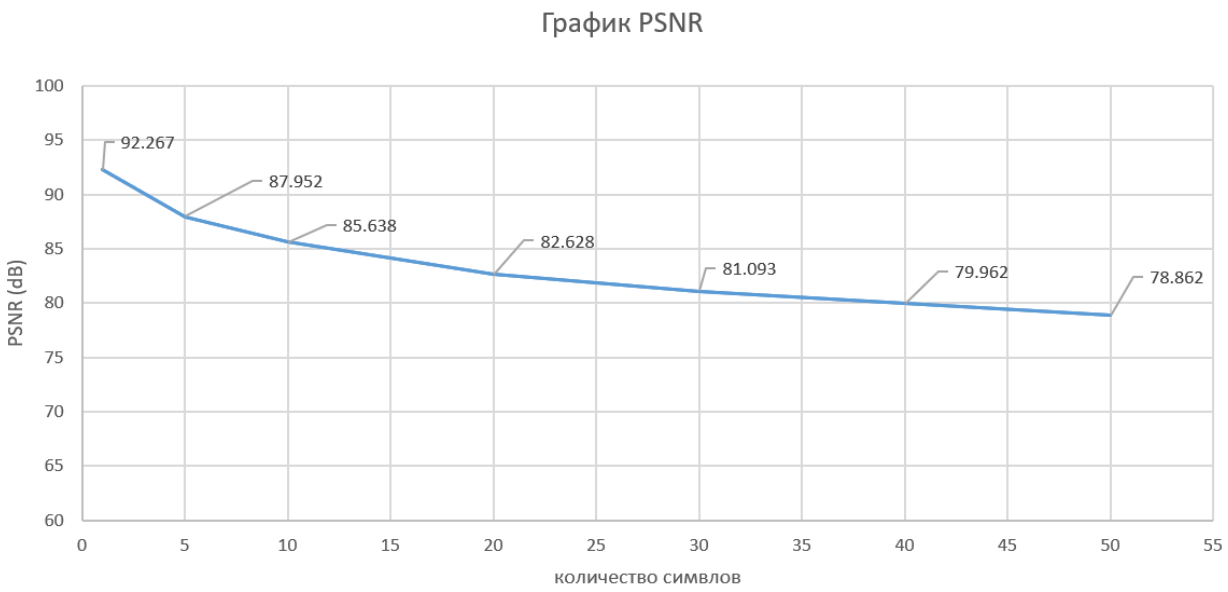
Мы построим графику значений PSNR исходного изображения и изображения с информацией, встроенные по наростающей одно слово, 5, 10, 20, 30, 40,50 до конца текстового файла.

```
PS E:\Documents\ITMO\steganography\Lab 2> & python "e:/Documents/ITMO/steganography/Lab 2/psnr.py"
92.26650007915408
PS E:\Documents\ITMO\steganography\Lab 2> & python "e:/Documents/ITMO/steganography/Lab 2/psnr.py"
87.9528624375642
PS E:\Documents\ITMO\steganography\Lab 2> & python "e:/Documents/ITMO/steganography/Lab 2/psnr.py"
85.63892176233834
PS E:\Documents\ITMO\steganography\Lab 2> & python "e:/Documents/ITMO/steganography/Lab 2/psnr.py"
82.62862180569853
PS E:\Documents\ITMO\steganography\Lab 2> & python "e:/Documents/ITMO/steganography/Lab 2/psnr.py"
81.09378712259642
PS E:\Documents\ITMO\steganography\Lab 2> & python "e:/Documents/ITMO/steganography/Lab 2/psnr.py"
79.96201086537134
PS E:\Documents\ITMO\steganography\Lab 2> & python "e:/Documents/ITMO/steganography/Lab 2/psnr.py"
78.86205893075288
```

Рисунка 6. Запуск программы “psnr.py” для 7 разных случаи количества символов в картинке

Количество симвлов	PSNR (in dB)
1	92.267
5	87.952
10	85.638
20	82.628
30	81.093
40	79.962
50	78.862

Таблица 1. Количества симвлов и значения PSNR



Рисунка 7. График значения PSNR и количества символов

2. Зависимость значения PSNR от размера встроенного текста:

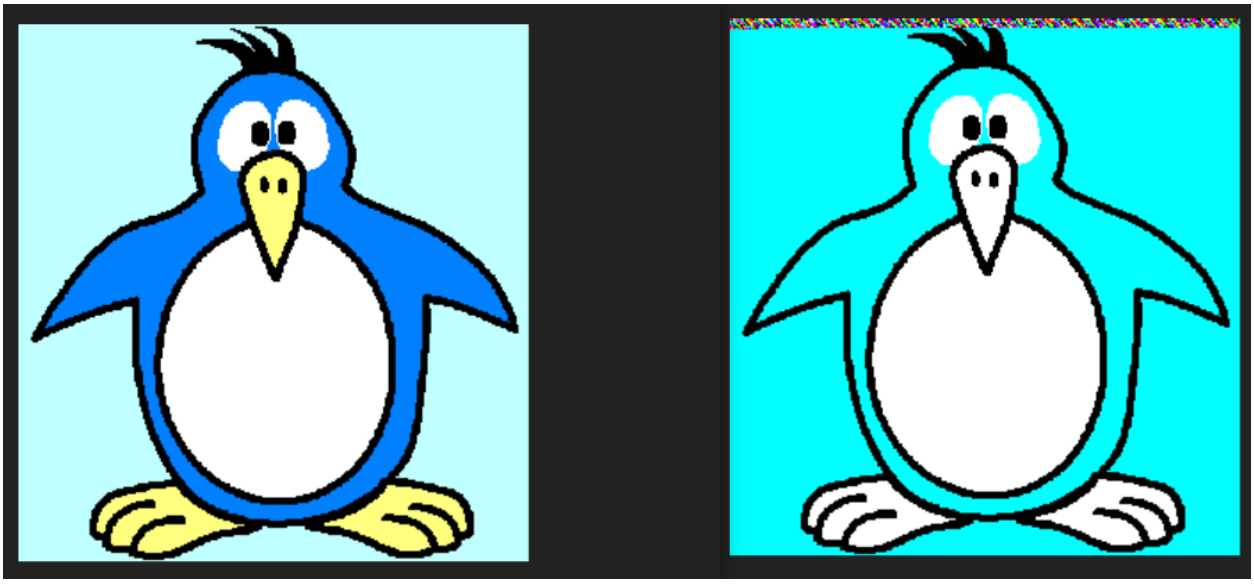
Можно видеть, что значение PSNR снижается, когда мы вставляем больше символа.

3. Проведение простого атака:

Для проведения атака, используется программа “detect.py”, если значение НЗБ канала равно 1 программа устанавливает значение канала в 255, если значение НЗБ канала равно 0 устанавливает значение канала в 1. После этого программа сохраняет новое изображение.

```
PS E:\Documents\ITMO\steganography\Lab 2> & python "e:/Documents/ITMO/steganography/Lab 2/detect.py"
Image file name that you want to analyze: output_image.bmp
New image file name: detected_image.bmp
```

Рисунка 8. Запуск программы “detect.py”



Рисунка 9. Разница между “output_image.bmp” и “detected_image.bmp”

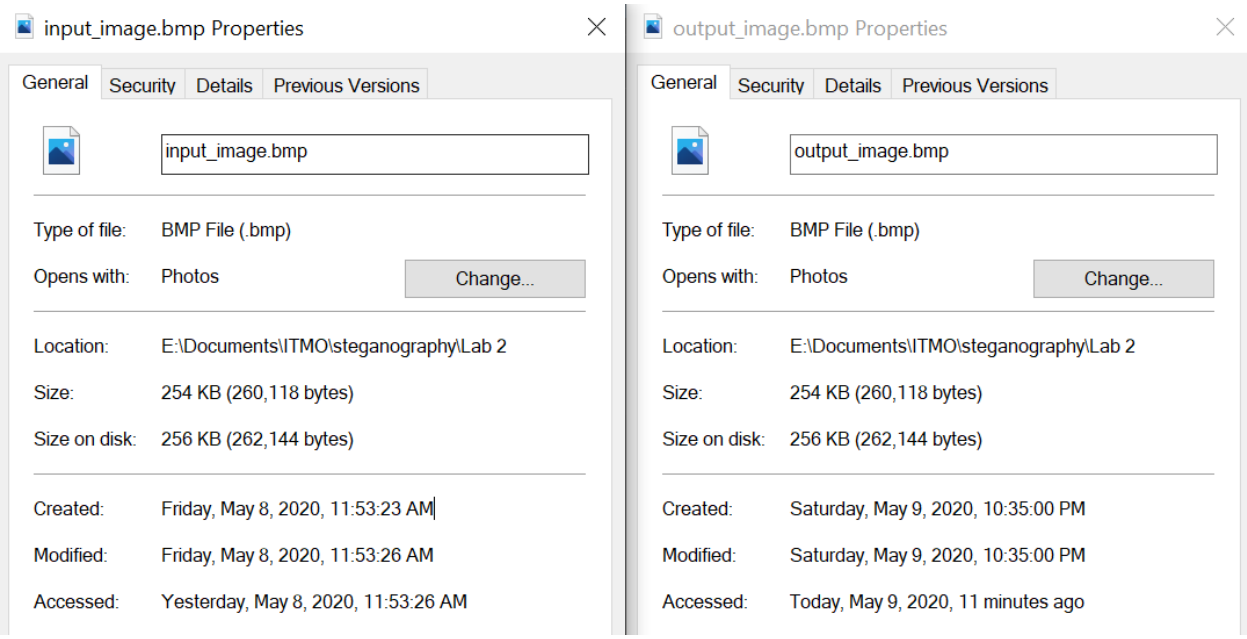
На основании разницы между 2 изображения. Можно делать вывод, что “output_image.bmp” содержает сообщение. Потому что буквы a, b, c, d,… , o имеют повторение “0110”, буквы p, q, r, s … , z имеют повторение “0111”. Такие повторения призвели к повторению значения каналов RGB в изображении.

ASCII Code	HTML Symbol	HTML Color Names	HTTP sta
95	137	5F	01011111	_
96	140	60	01100000	`
97	141	61	01100001	a
98	142	62	01100010	b
99	143	63	01100011	c
100	144	64	01100100	d
101	145	65	01100101	e
102	146	66	01100110	f
103	147	67	01100111	g
104	150	68	01101000	h
105	151	69	01101001	i
106	152	6A	01101010	j
107	153	6B	01101011	k
108	154	6C	01101100	l
109	155	6D	01101101	m
110	156	6E	01101110	n
111	157	6F	01101111	o

ASCII Code	HTML Symbol	HTML Color Names	HTTP sta
112	160	70	01110000	p
113	161	71	01110001	q
114	162	72	01110010	r
115	163	73	01110011	s
116	164	74	01110100	t
117	165	75	01110101	u
118	166	76	01110110	v
119	167	77	01110111	w
120	170	78	01111000	x
121	171	79	01111001	y
122	172	7A	01111010	z
123	173	7B	01111011	{
124	174	7C	01111100	|
125	175	7D	01111101	}

Таблица 2. Значения букв на ASCII таблице

4. Оценка целесообразности:



Рисунка 10. Размер изображение до и после встраивания не изменяется

Можно сделать вывод, что метод замены наименьшего значащего бита целесообразен по следующим причинам:

• Можно встраивать большое сообщение (до 1/8 размера исходного изображения).

• Размер стегоконтейнера остается таким же, как размер контейнера.

• Стегоконтейнер неразличимым невооруженным глазом.

Вывод

При выполнении данной лабораторной работы мною был изучен метод НЗБ стеганографии. Я научился применять его и проводить последующую оценку их применению. По результатам работы были сделаны следующие выводы:

Обнаружение метода НЗБ изображения невооруженным глазом или сравнением размера является невозможным, даже с специальным средствам при отсутствии исходного контейнера довольно сложно. Если я захочу в будущем поместить стего в изображении, то, скорее всего, воспользуюсь методом НЗБ, потому что у него высокая пропускная способность и довольная надежность.

Список использованной литературы

1. https://en.wikipedia.org/wiki/Bit_numbering - Bit numbering

2. https://www.ijltet.org/wp-content/uploads/2015/02/60.pdf - Steganography in Images Using LSB Technique

3. https://en.wikipedia.org/wiki/BMP_file_format - BMP file format

4. https://ru.wikipedia.org/wiki/RGB - RGB

5. https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio - PSNR (Peak signal-to-noise ratio)

Приложение

Программа image_stegano.py:

```
# Python program implementing image-based steganography
from PIL import Image

def str_2_bin(secret_message):
    binary_message = ''
    for character in secret_message:
        binary_message += bin(ord(character))[2:].zfill(8)
    return binary_message

def bin_2_str(binary_message):
    secret_message = ''
    for i in range(0, len(binary_message), 8):
        secret_message += chr(int(binary_message[i:i+8], 2))
    return secret_message

def encode(filename_in, secret_message, filename_out):
    binary_message = str_2_bin(secret_message)
    binary_length = bin(len(secret_message))[2:].zfill(8)
    total_data = binary_length + binary_message
    #print(total_data)
```

```
    image = Image.open(fp=filename_in, mode='r')
    width, heigth = image.size
    data_index = 0
    total_data_length = len(total_data)
    #while(data_index < total_data_length):
    for y in range(heigth): # y
        for x in range(width): # x
            pixel = list(image.getpixel((x, y))) # x, y
            for k in range(len(pixel)):
                if data_index < total_data_length:
                    # Replace the LSB value
                    new_pixel = int(bin(pixel[k])[2:].zfill(8)[:-1] + total_data[data_index], 2)
                    pixel[k] = new_pixel
                    data_index += 1
                else:
                    break
            image.putpixel((x, y), tuple(pixel))
    image.save(filename_out)
    image.close()

def decode(filename_in):
    image_in = Image.open(filename_in, mode='r')
    width, height = image_in.size
    # Decode the first 8 bits to get the length (a number of characters) of the secret message
    binary_length = ''
    binary_length_index = 0
    for y in range(height):
```

```
    for x in range(width):
        pixel = image_in.getpixel((x, y))
        for i in range(len(pixel)):
            if binary_length_index < 8:
                if pixel[i] % 2 == 0:
                    binary_length += '0'
                    binary_length_index += 1
                elif pixel[i] % 2 == 1:
                    binary_length += '1'
                    binary_length_index += 1
            else:
                break
length = int(binary_length, 2)
# Continue decode {length} characters of the secret message, skip the first 8 bits
binary_message = ''
binary_message_index = 0
for y in range(height):
    for x in range(width):
        pixel = image_in.getpixel((x, y))
        for i in range(len(pixel)):
            if binary_message_index < 8:
                binary_message_index += 1
                continue
            elif binary_message_index >= 8 and len(binary_message) < length * 8:
                if pixel[i] % 2 == 0:
                    binary_message += '0'
                elif pixel[i] % 2 == 1:
                    binary_message += '1'
```

```
            else:
                break
    secret_message = bin_2_str(binary_message)
    return secret_message

# The main program
if __name__ == '__main__':
    operation = input('1. Encode, 2. Decode: ')
    if operation == '1':
        print('ENCODING ...')
        input_text = input('Input text file (input_text.txt): ')
        src_image = input('Source image file (input_image.bmp): ')
        dst_image = input('Destination image file (output_image.bmp): ')
        f = open(file=input_text, mode='r')
        secret_message = f.read()
        f.close()
        encode(src_image, secret_message, dst_image)
        print("Encoded successfully")
    elif operation == '2':
        print('DECODING ...')
        encoded_image = input('Input image file (output_image.bmp): ')
        output_text = input('Output text file (output_text.txt): ')
        secret_message = decode(encoded_image)
        f = open(file=output_text, mode='w')
        f.write(secret_message)
        f.close()
        print("Decoded successfully")
    else:
```

```
    exit()
```

Программа psnr.py:

```
# Calculate PSNR of the original image and the stegcontainer image
from PIL import Image
import math


def psnr(input_image, output_image):
    output_image = Image.open(output_image, mode='r')
    width, height = output_image.size
    # Decode the first 8 bits to get the length (a number of characters) of the secret message
    binary_length = ''
    binary_length_index = 0
    for y in range(height):
        for x in range(width):
            pixel = output_image.getpixel((x, y))
            for i in range(len(pixel)):
                if binary_length_index < 8:
                    if pixel[i] % 2 == 0:
                        binary_length += '0'
                        binary_length_index += 1
                    elif pixel[i] % 2 == 1:
                        binary_length += '1'
                        binary_length_index += 1
                else:
                    break
    # Number of channels need to be calculated
    channel_number = int(binary_length, 2) * 8 + 8
    # Calculate PSNR of the {length * 8} bits, skip the first 8 bits
```

```
    input_image = Image.open(input_image, mode='r')
    current_channel_number = 0
    I_K_2 = 0
    for y in range(height):
        for x in range(width):
            output_pixel = output_image.getpixel((x, y))
            input_pixel = input_image.getpixel((x, y))
            for i in range(len(pixel)):
                if current_channel_number < channel_number:
                    subtraction = input_pixel[i] - output_pixel[i]
                    if (subtraction != 0):
                        I_K_2 += 1
                    current_channel_number += 1
                else:
                    break
    mse = I_K_2 / (width * height * 3)
    psnr = 10 * math.log(pow(255, 2) / mse, 10)
    return psnr

# The main program
if __name__ == '__main__':
    # Change the parameters
    print(psnr('input_image.bmp', 'output_image50.bmp'))
```

Программа detect.py:

```
# Detect image-based steganography
from PIL import Image
```

```
def detect(encoded_image, detected_image):
    image = Image.open(fp=encoded_image, mode='r')
    width, heigth = image.size
    for y in range(heigth):
        for x in range(width):
            pixel = list(image.getpixel((x, y)))
            for k in range(len(pixel)):
                if pixel[k] % 2 == 0:
                    pixel[k] = 1
                elif pixel[k] % 2 == 1:
                    pixel[k] = 255
            image.putpixel((x, y), tuple(pixel))

    image.save(detected_image)
    image.close()
# The main program
if __name__ == '__main__':
    sample_image = input('Image file name that you want to analyze: ')
    detected_image = input('New image file name: ')
    detect(sample_image, detected_image)
```