

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)**

Факультет Информатика и вычислительная техника

(наименование факультета)

Кафедра Кибербезопасность информационных систем

(наименование кафедры)

**Лабораторная работа № 9**

По дисциплине «Математические методы сокрытия и маскирования информации»

Выполнил обучающийся гр. ВКБ53

Донеров Юрий Андреевич

Проверил:

доц. Сафарьян Ольга Александровна

Ростов-на-Дону

2024

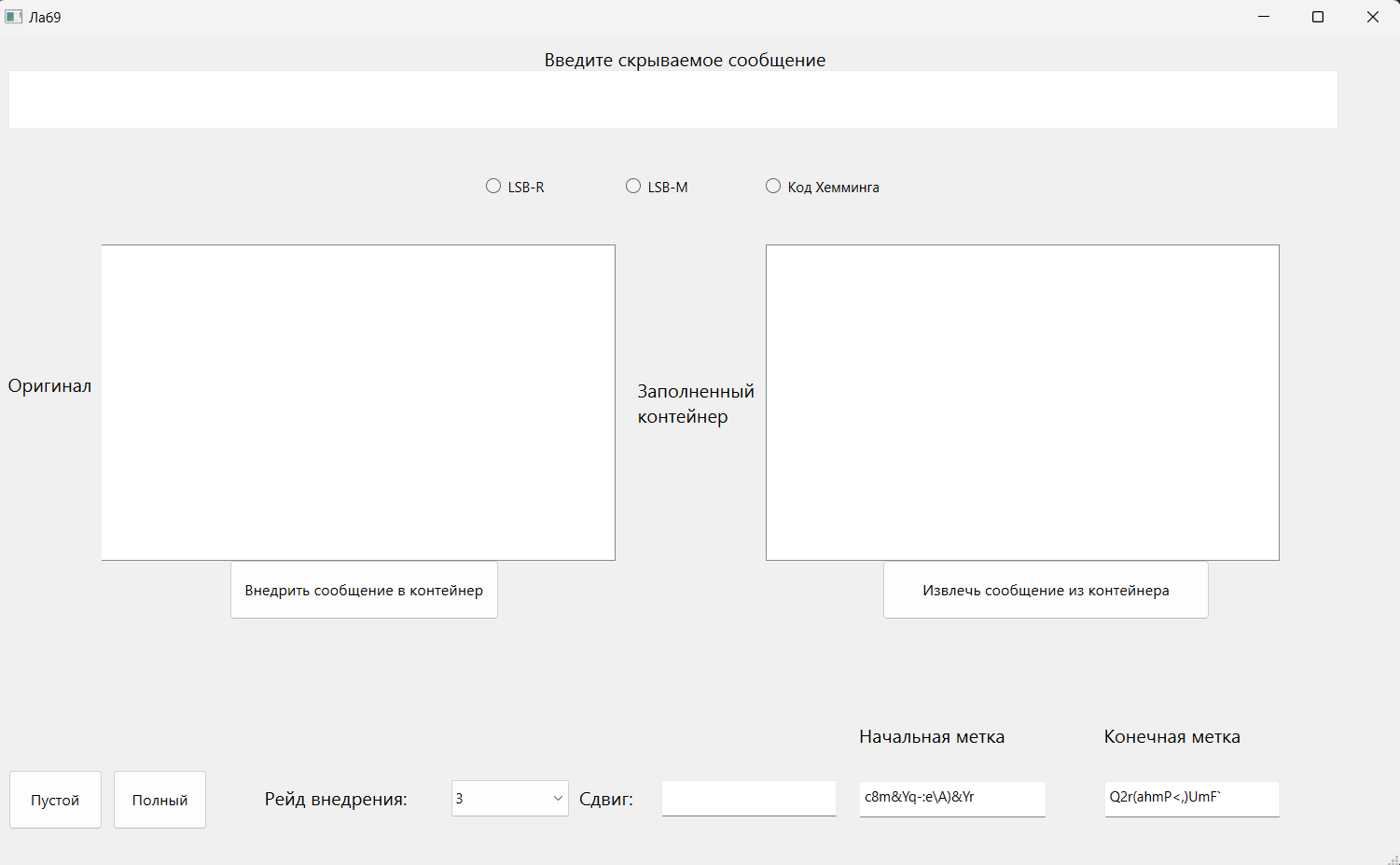
**Цель работы:** познакомиться с одним из наиболее распространенных методов сокрытия информации в неподвижных изображениях.

**Задание:** Написать программы внедрения (и извлечения) скрытой информации в BMP-файлы, используя методы LSB-R и LSB-M, с задаваемым рейтом внедрения. Для заданных файлов (PU24-\*.bmp) исследовать зависимости между рейтом внедрения и степенью сжатия, изменение степени сжатия при повторном внедрении при использовании различных архиваторов или графических конверторов. Затем написать программу внедрения на основе кода Хемминга. Провести для нее аналогичные исследования, сопоставить результаты с обычными методами при (почти) том же рейте внедрения. Сделать выводы о стеганографической стойкости исследованных методов

Ссылка на Github данной работы:

<https://github.com/iTitanick/Donerov_Steg/>

Внешний вид интерфейса программы:



Код программы:

main\_programm.py:

import sys

import LSB

import hashlib

import steg\_on\_hamming

from addit\_functs import read\_color, get\_size, save\_color, generate\_alphanum\_crypt\_string

from PyQt6.QtWidgets import QApplication

from PyQt6.QtWidgets import QMainWindow, QFileDialog

from PyQt6.QtWidgets import QGraphicsPixmapItem, QGraphicsItem, QGraphicsScene

from PyQt6.QtGui import QPixmap

from main\_window import Ui\_MainWindow

class MainWindow(QMainWindow):

    def \_\_init\_\_(self):

        super(MainWindow, self).\_\_init\_\_()

        self.ui = Ui\_MainWindow()

        self.ui.setupUi(self)

        self.new\_container\_color = 0

        self.full\_container\_color = 0

        self.full\_container\_name = 0

        self.status = "hide"

        self.ui.exit.triggered.connect(QApplication.instance().quit)

        self.ui.button\_hide\_message.clicked.connect(self.hideMassageClicked)

        self.ui.button\_extract\_message.clicked.connect(self.extractMassageClicked)

        self.ui.button\_hide\_message\_2.clicked.connect(self.openNewcontainer)

        self.ui.button\_hide\_message\_3.clicked.connect(self.openFullcontainer)

*# self.ui.textEdit\_m\_start.setText(generate\_alphanum\_crypt\_string(15))*

*# self.ui.textEdit\_m\_end.setText(generate\_alphanum\_crypt\_string(15))*

        self.ui.textEdit\_m\_start.setText('c8m&Yq-:e\A)&Yr')

        self.ui.textEdit\_m\_end.setText('Q2r(ahmP<,)UmF`')

        self.ui.rd.toggled.connect(self.updateComboBox)

        self.ui.rd2.toggled.connect(self.updateComboBox)

        self.ui.rd3.toggled.connect(self.updateComboBox)

    def updateComboBox(self):

*# Очистка комбобокса*

        self.ui.stego\_reid.clear()

*# Обновление комбобокса в зависимости от выбранной радиокнопки*

        if self.ui.rd.isChecked():

            self.ui.stego\_reid.addItems(["3", "1.5", "1", "0.75", "0.5", "0.25", "0.2", "0.1", "0.05"])

        elif self.ui.rd2.isChecked():

            self.ui.stego\_reid.addItems(["3", "1.5", "1", "0.75", "0.5", "0.25", "0.2", "0.1", "0.05"])

        elif self.ui.rd3.isChecked():

            self.ui.stego\_reid.addItems(["0.25"])

    def hideMassageClicked(self):

        if self.new\_container\_color:

            self.ui.statusbar.showMessage("Начинаем процесс заполнения контейнера")

            text = self.ui.textEdit\_message.toPlainText()

            print(self.ui.stego\_reid.currentText())

            try:

                reid = float(self.ui.stego\_reid.currentText())

                sdvig = int(self.ui.textEdit\_sdvig.toPlainText())

                m\_start = self.ui.textEdit\_m\_start.toPlainText().encode("utf-8")

                m\_end = self.ui.textEdit\_m\_end.toPlainText().encode("utf-8")

*# text = AES\_cript.encrypt(text, key)*

                m\_start = hashlib.sha256(m\_start).hexdigest()

                m\_end = hashlib.sha256(m\_end).hexdigest()

                new\_image = ""

                if self.ui.rd.isChecked():

                    new\_image = LSB.LSB\_R\_enc(self.new\_container\_color, text, m\_start, m\_end, sdvig, reid)

                elif self.ui.rd2.isChecked():

                    new\_image = LSB.LSB\_M\_enc(self.new\_container\_color, text, m\_start, m\_end, sdvig, reid)

                elif self.ui.rd3.isChecked():

                    new\_image = steg\_on\_hamming.SoH\_enc(self.new\_container\_color, text, m\_start, m\_end, sdvig)

                name\_file = QFileDialog.getSaveFileName(self, "Сохранить файл", "./"

                                                             , filter="Изображения (\*.bmp)")[0]

                save\_color(self.new\_container\_name, name\_file, new\_image, 'pixels')

                self.full\_container\_name = name\_file

                self.openFullcontainer()

                self.ui.statusbar.showMessage("Сообщение скрыто")

            except Exception as err:

                print(err)

        else:

            self.ui.statusbar.showMessage("Ожидание открытия файла")

    def extractMassageClicked(self):

        if self.full\_container\_name:

            text = self.ui.textEdit\_message.toPlainText()

            reid = float(self.ui.stego\_reid.currentText())

            m\_start = self.ui.textEdit\_m\_start.toPlainText().encode("utf-8")

            m\_end = self.ui.textEdit\_m\_end.toPlainText().encode("utf-8")

            m\_start = hashlib.sha256(m\_start).hexdigest()

            m\_end = hashlib.sha256(m\_end).hexdigest()

            try:

                private\_text = ""

                if self.ui.rd.isChecked() or self.ui.rd2.isChecked():

                    private\_text = LSB.LSB\_dec(self.full\_container\_color, m\_start, m\_end, reid)

                elif self.ui.rd3.isChecked():

                    private\_text = steg\_on\_hamming.SoH\_dec(self.full\_container\_color, m\_start, m\_end)

*# text = AES\_cript.decrypt(private\_text, key)*

                text = private\_text

                self.ui.textEdit\_message.setText(text)

                self.ui.statusbar.showMessage("Сообщение извлечено")

            except Exception as err:

                print(err)

                self.ui.statusbar.showMessage("Ошибка!!!")

        else:

            self.ui.statusbar.showMessage("Ожидание открытия файла")

        return 0

    def openNewcontainer(self):

        self.new\_container\_name = QFileDialog.getOpenFileName(self, "Открыть файл", "./"

                                                         , filter="Изображения (\*.png \*.jpg \*.bmp)")[0]

        width, height = get\_size(self.new\_container\_name)

        k = 0

        if width > height:

            k = width / 468

        else:

            k = height / 298

        new\_width = round(width / k)

        new\_height = round(height / k)

        self.new\_container\_color = read\_color(self.new\_container\_name, 'pixels')

        self.scene\_new\_container = QGraphicsScene()

        pic = QGraphicsPixmapItem()

        pic.setPixmap(QPixmap(self.new\_container\_name).scaled(new\_width, new\_height))

        self.scene\_new\_container.addItem(pic)

        self.ui.original\_image.setScene(self.scene\_new\_container)

        self.ui.statusbar.showMessage("Файл " + self.new\_container\_name + " открыт")

    def openFullcontainer(self):

        if self.full\_container\_name == 0:

            self.full\_container\_name = QFileDialog.getOpenFileName(self, "Открыть файл", "./"

                                                            , filter="Изображение без сжатия (\*.bmp)")[0]

            self.status = "show"

        width, height = get\_size(self.full\_container\_name)

        k = 0

        if width > height:

            k = width / 468

        else:

            k = height / 298

        new\_width = round(width / k)

        new\_height = round(height / k)

        self.full\_container\_color = read\_color(self.full\_container\_name, 'pixels')

        self.scene\_full\_container = QGraphicsScene()

        pic = QGraphicsPixmapItem()

        pic.setPixmap(QPixmap(self.full\_container\_name).scaled(new\_width, new\_height))

        self.scene\_full\_container.addItem(pic)

        self.ui.stego\_image.setScene(self.scene\_full\_container)

        if self.status == "show":

            self.ui.statusbar.showMessage("Файл " + self.full\_container\_name + " открыт")

def main():

    app = QApplication([])

    application = MainWindow()

    application.show()

    sys.exit(app.exec())

if \_\_name\_\_ == '\_\_main\_\_':

    main()

dop\_func:

import secrets

import string

from PIL import Image, ImageDraw

import numpy as np

*# Функция извлекает цвета RGB из изображение*

def read\_color(img\_path, color\_metod):

    img = Image.open(img\_path)

    img\_pix = img.load()

    width = img.size[0]

    height = img.size[1]

    red = []

    green = []

    blue = []

    for x in range(width):

        for y in range(height):

            r, g, b = img\_pix[x, y]

            red.append(r)

            green.append(g)

            blue.append(b)

    img.close()

    if color\_metod == 'red':

        return red

    elif color\_metod == 'green':

        return green

    elif color\_metod == 'blue':

        return blue

    elif color\_metod == 'pixels':

        data = []

        for i in range(len(red)):

            data.append(red[i])

            data.append(green[i])

            data.append(blue[i])

        return data

    else:

        return red + green + blue

*# Сохранение изменённых пикселей в изображении*

def save\_color(img\_old\_path, img\_new\_path, img\_pix\_new, color\_metod):

    img = Image.open(img\_old\_path)

    img\_draw = ImageDraw.Draw(img)

    img\_pix = img.load()

    width = img.size[0]

    height = img.size[1]

    index = 0

    for x in range(width):

        for y in range(height):

            r, g, b = img\_pix[x, y]

            if color\_metod == 'red':

                img\_draw.point((x, y), (img\_pix\_new, g, b))

            elif color\_metod == 'green':

                img\_draw.point((x, y), (r, img\_pix\_new, b))

            elif color\_metod == 'blue':

                img\_draw.point((x, y), (r, g, img\_pix\_new))

            elif color\_metod == 'pixels':

                red = img\_pix\_new[index]

                green = img\_pix\_new[index+1]

                blue = img\_pix\_new[index+2]

                img\_draw.point((x, y), (red, green, blue))

                index += 3

            else:

                red = img\_pix\_new[0:len(r)]

                green = img\_pix\_new[len(r):len(g)]

                blue = img\_pix\_new[len(g):]

                img\_draw.point((x, y), (red, green, blue))

    img.save(img\_new\_path, "BMP")

    img.close()

*# Получает размеры изображения*

def get\_size(img\_path):

    img = Image.open(img\_path)

    width = img.size[0]

    height = img.size[1]

    img.close()

    return width, height

*# Преобразует текст в массив байтов utf-8*

def text\_to\_binary(text):

    text = text.encode("utf-8")

    data = []

    for c in text:

        data.append(c)

    return data

*# Преобразует массив байт в текст utf-8*

def binary\_to\_text(data):

*# text = ""*

*# for byte in data:*

*#     try:*

*#         c = bytes([byte]).decode("utf-8")[0]*

*#         text += c*

*#     except:*

*#         continue*

    return bytes(data).decode("utf-8")

*# Переводи байт в массив длины 8 бит*

def number\_to\_bin\_arr(data):

    return [1 if data & (1 << (7 - n)) else 0 for n in range(8)]

*# Обратное преобразование массива бит в байт*

def bin\_arr\_to\_number(data):

    out = 0

    for bit in data:

        out = (out << 1) | bit

    return out

*# заменяет указанный бит в числе на единицу*

def set\_bit(value, bit):

    return value | (1<<bit)

*# заменяет указанный бит в числе на ноль*

def clear\_bit(value, bit):

    return value & ~(1<<bit)

*# Поиск первого вхождения подмассива в массиве*

def find\_SubarrayStartIndex(array, subArray):

    index = -1

    for i in range(len(array) - len(subArray) + 1):

        index = i

        for j in range(len(subArray)):

            if array[i + j] != subArray[j]:

                index = -1

                break

        if index >= 0:

            return index

    return -1

*# Извлечение подмассива данных, окруженных двумя дургими подмассивами*

def subarr\_extract(bit\_start, arr, bit\_end):

    index\_bs = find\_SubarrayStartIndex(arr, bit\_start)

    index\_be = find\_SubarrayStartIndex(arr, bit\_end)

    return arr[index\_bs+len(bit\_start):index\_be]

*# Получение бита числа на определённой позиции*

def retn\_bit(num, pos):

    array\_bit = number\_to\_bin\_arr(num)

    return array\_bit[pos]

*# Деление вектора по модулю 2*

def mod\_on\_2(vector):

    div\_vector = vector.copy()

    div\_vector.fill(2)

    return np.remainder(vector, div\_vector)

*# Переворачивание бита в векторе*

def reverse\_bit(vector, index):

    if index == -1:

        return vector

    if vector[index] == 1:

        vector[index] = 0

    else:

        vector[index] = 1

    return vector

def generate\_alphanum\_crypt\_string(length):

    letters\_and\_digits = string.ascii\_letters + string.digits + string.punctuation

    crypt\_rand\_string = ''.join(secrets.choice(

        letters\_and\_digits) for i in range(length))

    return crypt\_rand\_string

LSB.py:

from dop\_func import \*

import random

TB\_old = []

TB\_new = []

def LSB\_R\_enc(img, text, start, end, sdvig, raid=0.25):

*# Преобразование текстовых маркеров в бинарное представление*

    binary\_text = text\_to\_binary(start) + text\_to\_binary(text) + text\_to\_binary(end)

*# Вычисление нового значения raid (частоты изменения пикселей)*

    raid = round(3 / raid)

    print(raid)

*# Вычисление начального индекса для скрытия битов*

    index = round(sdvig \* raid)

*# Инициализация списка для хранения младших битов*

    array\_LSB = []

*# Цикл по каждому символу в бинарном тексте*

    for ch in binary\_text:

*# Конвертация символа в массив битов*

        arr\_bit\_ch = number\_to\_bin\_arr(ch)

*# Цикл по каждому биту в символе*

        for bit in arr\_bit\_ch:

*# Установка младшего бита в пикселе изображения в зависимости от значения бита*

            if bit == 0:

                img[index] = clear\_bit(img[index], 0)

            else:

                img[index] = set\_bit(img[index], 0)

*# Получение младшего бита из обновленного пикселя и добавление его в список*

            bits = number\_to\_bin\_arr(img[index])

            array\_LSB.append(bits[7])

*# Обновление индекса с учетом raid*

            index += raid

*# Формирование байтов из младших битов и добавление их в список TB\_old*

    TB\_old = []

    for i in range(0, len(array\_LSB), 8):

        byte = bin\_arr\_to\_number(array\_LSB[i:i + 8])

        TB\_old.append(byte)

*# Возвращение измененного изображения*

    return img

import random

def LSB\_M\_enc(img, text, start, end, sdvig, raid=0.25):

*# Преобразуем текст в бинарное представление и объединяем с маркерами начала и конца*

    binary\_text = text\_to\_binary(start) + text\_to\_binary(text) + text\_to\_binary(end)

*# Вычисляем значение шага для изменения пикселей в изображении*

    raid = round(3 / raid)

*# Инициализируем индекс для отслеживания текущего положения в изображении*

    index = round(sdvig \* raid)

*# Перебираем каждый бит скрытого текста*

    for ch in binary\_text:

*# Преобразуем символ (бит) в массив битов*

        arr\_bit\_ch = number\_to\_bin\_arr(ch)

*# Перебираем каждый бит в массиве битов символа*

        for bit in arr\_bit\_ch:

*# Получаем текущий младший бит пикселя изображения*

            bit\_img = retn\_bit(img[index], 7)

*# Если бит текста и младший бит пикселя равны 0*

            if bit == 0 and bit\_img == 0:

*# Ничего не делаем и смещаемся к следующему пикселю*

                index += raid

                continue

*# Если бит текста 0, а младший бит пикселя 1*

            elif bit == 0 and bit\_img == 1:

*# Изменяем младший бит пикселя на случайное значение, чтобы сделать изменения менее заметными*

                if img[index] == 255:

                    img[index] = img[index] - 1

                else:

                    rand\_bit = random.randrange(-1, 2, 2)

                    img[index] = img[index] + rand\_bit

*# Если бит текста 1, а младший бит пикселя 0*

            elif bit == 1 and bit\_img == 0:

*# Изменяем младший бит пикселя на случайное значение*

                if img[index] == 0:

                    img[index] = img[index] + 1

                else:

                    rand\_bit = random.randrange(-1, 2, 2)

                    img[index] = img[index] + rand\_bit

*# Если бит текста и младший бит пикселя равны 1*

            elif bit == 1 and bit\_img == 1:

*# Ничего не делаем и смещаемся к следующему пикселю*

                index += raid

                continue

*# Смещаемся к следующему пикселю*

            index += raid

*# Возвращаем измененное изображение*

    return img

def LSB\_dec(img, start, end, raid=0.25):

*# Округляем 3 / raid и присваиваем результат переменной raid*

    raid = round(3 / raid)

*# Создаем пустой список для хранения LSB каждого байта изображения*

    array\_LSB = []

*# Перебираем каждый байт в изображении с шагом, определенным переменной raid*

    for i in range(0, len(img), raid):

*# Получаем текущий байт изображения*

        byte = img[i]

*# Преобразуем текущий байт в массив битов*

        bits = number\_to\_bin\_arr(byte)

*# Добавляем последний (младший) бит текущего байта в список array\_LSB*

        array\_LSB.append(bits[7])

*# Создаем пустые списки для хранения битов начальной и конечной строки*

    bit\_start = []

    bit\_end = []

*# Преобразуем начальную строку в бинарный формат и добавляем ее биты в bit\_start*

    for e in text\_to\_binary(start):

        bit\_start = bit\_start + number\_to\_bin\_arr(e)

*# Преобразуем конечную строку в бинарный формат и добавляем ее биты в bit\_end*

    for e in text\_to\_binary(end):

        bit\_end = bit\_end + number\_to\_bin\_arr(e)

*# Извлекаем массив битов, соответствующий скрытому тексту, используя биты начальной и конечной строки*

    text\_bit = subarr\_extract(bit\_start, array\_LSB, bit\_end)

*# Создаем пустой список для хранения байтов скрытого текста*

    text\_byte = []

*# Преобразуем массив битов скрытого текста обратно в байты*

    for i in range(0, len(text\_bit), 8):

        byte = bin\_arr\_to\_number(text\_bit[i:i + 8])

        text\_byte.append(byte)

    print(TB\_old)

*# Выводим извлеченный скрытый текст в виде байтов*

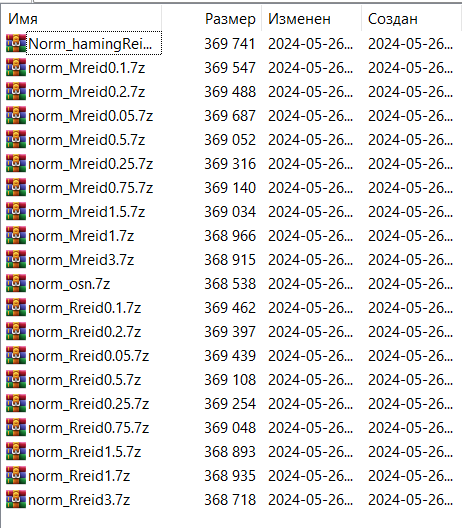
    print(text\_byte)

*# Преобразуем байты скрытого текста в строку и возвращаем результат*

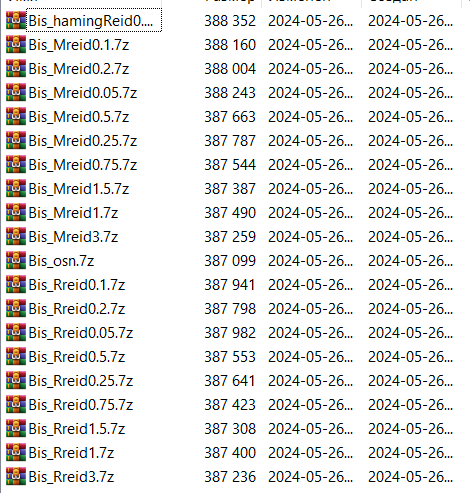
    return binary\_to\_text(text\_byte)

**Анализ:**

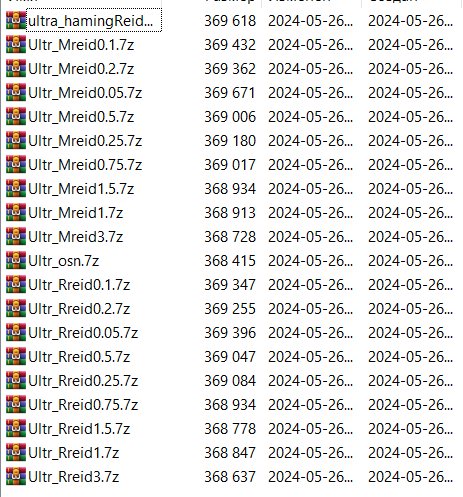
Нормальный режим + оригинал:



Быстрый режим + оригинал:



Ультра режим + оригинал:



**Анализ:**

\*Размер указан в байтах.

При сжатии 7zip нормальный режим

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Рейд 0.1 | | Рейд 0.25 | | Рейд 0.75 | | Рейд 1.5 | | Рейд 3 | |
| Сжат | Не сжат | Сжат | Не сжат | Сжат | Не сжат | Сжат | Не сжат | Сжат | Не сжат |
| LSB\_r | 369462 | 553030 | 369254 | 553030 | 369048 | 553030 | 368893 | 553030 | 368718 | 553030 |
| LSB\_m | 369547 | 553030 | 369316 | 553030 | 369140 | 553030 | 369034 | 553030 | 368915 | 553030 |
| Hamming | - | - | 369741 | 553030 | - | - | - | - | - | - |
| Оригинал | 368538 | 553030 | 368538 | 553030 | 368538 | 553030 | 368538 | 553030 | 368538 | 553030 |

При сжатии 7zip быстрый режим

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Рейд 0.1 | | Рейд 0.25 | | Рейд 0.75 | | Рейд 1.5 | | Рейд 3 | |
| Сжат | Не сжат | Сжат | Не сжат | Сжат | Не сжат | Сжат | Не сжат | Сжат | Не сжат |
| LSB\_r | 387941 | 553030 | 387641 | 553030 | 387423 | 553030 | 387308 | 553030 | 387236 | 553030 |
| LSB\_m | 388160 | 553030 | 387787 | 553030 | 387544 | 553030 | 387387 | 553030 | 387259 | 553030 |
| Hamming | - | - | 388352 | 553030 | - | - | - | - | - | - |
| Оригинал | 387099 | 553030 | 387099 | 553030 | 387099 | 553030 | 387099 | 553030 | 387099 | 553030 |

При сжатии 7zip ультра режим

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Рейд 0.1 | | Рейд 0.25 | | Рейд 0.75 | | Рейд 1.5 | | Рейд 3 | |
| Сжат | Не сжат | Сжат | Не сжат | Сжат | Не сжат | Сжат | Не сжат | Сжат | Не сжат |
| LSB\_r | 369347 | 553030 | 369084 | 553030 | 368934 | 553030 | 368778 | 553030 | 368637 | 553030 |
| LSB\_m | 369432 | 553030 | 369180 | 553030 | 369017 | 553030 | 368934 | 553030 | 368728 | 553030 |
| Hamming | - | - | 369618 | 553030 | - | - | - | - | - | - |
| Оригинал | 368415 | 553030 | 368415 | 553030 | 368415 | 553030 | 368415 | 553030 | 368415 | 553030 |

Как можно заметить из анализа на сжатие, чем выше указан рейд внедрения, тем лучше файл поддается сжатию. Рейтинг по способности к сжатию среди алгоритмов выглядит следующим образом:

1. Оригинал
2. LSB\_R
3. Hamming
4. LSB\_M

**Вывод:** Данная лабораторная работа позволила на практике познакомиться с одним из наиболее распространенных методов сокрытия информации в неподвижных изображениях – методом замены наименее значащего бита.