

ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ  
ФЕДЕРАЛЬНОЕ  
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО  
ОБРАЗОВАНИЯ  
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра прикладной математики и кибернетики

Лабораторная работа №7

Выполнили:  
Студенты 4 курса ИВТ,  
группы ИП-013  
Копытина Татьяна, Семилетко Максим

Работу проверил: доцент кафедры ПМиК  
Перцев И.В.

Новосибирск 2024 г.

## **Оглавление**

Задание .....	3
Листинг программы .....	3
Результат работы программы.....	12

## Задание

Стеганография. Записать в младшие биты растра True Color BMP файла (контейнера) текстовый файл. Размер текстового файла взять 25%, 50% и 75% от размера графических данных (использовать для внедрения младшие 2, 4 или 6 бит в каждом байте графических данных). Извлечь файл из контейнера. Сравнить на идентичность с исходным. Сравнить графические файлы до внедрения информации и после.

## Листинг программы

```
import random

import math

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image


BMP_HEADER_BSIZE = 14

BMP_INFO_HEADER_BSIZE = 40


class BmpFile:

    def init(self, name):

        self.name = name

        self.fileObj = None

        self.header = None

        self.infoHeader = None

        self.palette = None

        self.paletteSize = None

        self.colorCount = None

        self.bpp = None

        self.padding = None


        self.type = None

        self.size = None
```

```
self.reserved = None
self.offset = None

self.infoHeaderSize = None
self.width = None
self.height = None
self.planes = None
self.depthColor = None
self.compression = None
self.compressedSize = None
self.xPixPM = None
self.yPixPM = None
self.usedColors = None
self.importantColors = None
```

```
def PrintInfo(self):
    print("-----HEADER-----")
    print(f"TYPE: {self.type}")
    print(f"FILE SIZE: {self.size}")
    print(f"RESERVED: {self.reserved}")
    print(f"DATA OFFSET: {self.offset}")
    print("-----INFO HEADER-----")
    print(f"HEADER SIZE: {self.infoHeaderSize}")
    print(f"WIDTH: {self.width}")
    print(f"HEIGHT: {self.height}")
    print(f"PLANES: {self.planes}")
    print(f"DEPTH: {self.depthColor}")
    print(f"COMPRESSION: {self.compression}")
    print(f"COMPRESSED SIZE: {self.compressedSize}")
    print(f"X RESOLUTION: {self.xPixPM}")
```

```
print(f"Y RESOLUTION: {self.yPixPM}")
print(f"USED COLORS: {self.usedColors}")
print(f"IMPORTANT COLORS: {self.importantColors}")
print()
```

```
class BmpFileReader:
```

```
    def init(self, fileName):
```

```
        self.bmpObj = BmpFile(fileName)
```

```
    def Read(self):
```

```
        self.bmpObj.fileObj = open(self.bmpObj.name, 'rb')
```

```
        self.bmpObj.header =
```

```
self.bmpObj.fileObj.read(BMP_HEADER_BSIZE)
```

```
        # HEADER
```

```
        self.bmpObj.type = self.bmpObj.header[:2].decode('utf-8')
```

```
        self.bmpObj.size = int.from_bytes(self.bmpObj.header[2:6],
'little')
```

```
        self.bmpObj.reserved =
int.from_bytes(self.bmpObj.header[6:10], 'little')
```

```
        self.bmpObj.offset = int.from_bytes(self.bmpObj.header[10:14],
'little')
```

```
        # HEADER #
```

```
        self.bmpObj.infoHeader =
```

```
self.bmpObj.fileObj.read(BMP_INFO_HEADER_BSIZE)
```

```
        # INFO HEADER
```

```
        self.bmpObj.infoHeaderSize =
int.from_bytes(self.bmpObj.infoHeader[:4], 'little')
```

```
        self.bmpObj.width =
int.from_bytes(self.bmpObj.infoHeader[4:8], 'little')
```

```
        self.bmpObj.height =
int.from_bytes(self.bmpObj.infoHeader[8:12], 'little')
```

```
        self.bmpObj.planes =
int.from_bytes(self.bmpObj.infoHeader[12:14], 'little')
```

```

        self.bmpObj.depthColor =
int.from_bytes(self.bmpObj.infoHeader[14:16], 'little')

        self.bmpObj.compression =
int.from_bytes(self.bmpObj.infoHeader[16:20], 'little')

        self.bmpObj.compressedSize =
int.from_bytes(self.bmpObj.infoHeader[20:24], 'little')

        self.bmpObj.xPixPM =
int.from_bytes(self.bmpObj.infoHeader[24:28], 'little')

        self.bmpObj.yPixPM =
int.from_bytes(self.bmpObj.infoHeader[28:32], 'little')

        self.bmpObj.usedColors =
int.from_bytes(self.bmpObj.infoHeader[32:36], 'little')

        self.bmpObj.importantColors =
int.from_bytes(self.bmpObj.infoHeader[36:40], 'little')

        # INFO HEADER #

        self.bmpObj.colorCount = pow(2, self.bmpObj.depthColor)

        self.bmpObj.paletteSize = self.bmpObj.colorCount * 4

        #self.bmpObj.palette =
self.bmpObj.fileObj.read(self.bmpObj.paletteSize)

        self.bmpObj.bpp = self.bmpObj.depthColor // 8

        self.bmpObj.padding = (4 - (self.bmpObj.width *
self.bmpObj.bpp) % 4) % 4

        return self.bmpObj.fileObj

def GenerateNewPalette(self, pixels, width, height):
    colors = {}
    for y in range(height):
        for x in range(width):
            flattenColor = (pixels[y, x][0] >> 4 << 4, pixels[y,
x][1] >> 4 << 4, pixels[y, x][2] >> 4 << 4)

            colors[flattenColor] = colors[flattenColor] + 1 if
flattenColor in colors else 1

```

```

colors = list(colors.items())
colors.sort(key=lambda x: x[1], reverse=False)

newPalette = []
newPalette.append(colors.pop()[0])
newColorCount = 1

while newColorCount < self.outputColorNum:
    newColor = colors.pop()[0]
    for color in newPalette:
        if 128*128*3 < self.CountDelta(color, newColor):
            newPalette.append(newColor)
            newColorCount += 1
            break
    return newPalette

row = originalFile.read((self.width + self.padding) * self.bpp)
    newFile.write(row)

    for _ in range(borderWidth):
        newFile.write(random.randint(0,
colorNum).to_bytes(self.bpp, 'little'))

        newFile.write(b'\x00' * (padding - self.padding))

    for _ in range(borderWidth):
        for _ in range(newWidth):
            newFile.write(random.randint(0,
colorNum).to_bytes(self.bpp, 'little'))
            newFile.write(b'\x00' * padding)
def EncodeText(self, sizePercent):
    self.encodeOffset = math.ceil(8 * sizePercent)

```

```

with open('text.txt', 'r') as textFile:
    textBits = ""
    readBitsCount = int(self.compressedSize * sizePercent)
    readBitsCount = (readBitsCount - readBitsCount % 24) * 8
    text = textFile.read()
    for i in text:
        textBits = textBits + bin(ord(i))[2:].zfill(8)
        if len(textBits) >= readBitsCount:
            break

with open(self.name, 'rb') as originalFile:
    header = originalFile.read(self.offset)
    graphImg = np.zeros((self.height, self.width, 3),
dtype=np.uint8)

    for y in range(self.height - 1, -1, -1):
        for x in range(self.width):
            blue = int.from_bytes(originalFile.read(1),
'little')

            green = int.from_bytes(originalFile.read(1),
'little')

            red = int.from_bytes(originalFile.read(1),
'little')

            graphImg[y, x] = [red, green, blue]

    originalFile.read(self.padding)

    bitCounter = 0
    textBitsCount = len(textBits)
    self.textBitsCount = textBitsCount

    for y in range(self.height - 1, -1, -1):

```



```

        for x in range(self.width):
            if bitCounter < textBitsCount:
                for i in range(3):
                    botRowOffset = bitCounter +
(self.encodeOffset * i)
                    graphImg[y, x][i] = ((graphImg[y, x][i] >>
self.encodeOffset) << self.encodeOffset) |
int(textBits[botRowOffset:botRowOffset+self.encodeOffset], 2)

                    bitCounter += self.encodeOffset * 3

with open('encoded_' + self.name, 'wb') as newFile:
    newHeader = bytearray(header)
    newFile.write(newHeader)

    newPixels = bytearray()
    for y in range(self.height - 1, -1, -1):
        for x in range(self.width):
            blue = int(graphImg[y, x][2]).to_bytes(1,
'little')
            green = int(graphImg[y, x][1]).to_bytes(1,
'little')
            red = int(graphImg[y, x][0]).to_bytes(1,
'little')

            newPixels.extend(blue)
            newPixels.extend(green)
            newPixels.extend(red)

            newPixels.extend(b'\x00' * self.padding)

    newFile.write(newPixels)

plt.figure()

```

```

        plt.imshow(graphImg)
        plt.axis('off')
        return graphImg

def DecodeText(self):
    bits = ""
    decodedText = ""
    bitsStr = ""
    bitCounter = 0

    with open('encoded_' + self.name, 'rb') as originalFile:
        header = originalFile.read(self.offset)

        graphImg = np.zeros((self.height, self.width, 3),
dtype=np.uint8)

        for y in range(self.height - 1, -1, -1):
            for x in range(self.width):
                blue = int.from_bytes(originalFile.read(1),
'little')
                green = int.from_bytes(originalFile.read(1),
'little')
                red = int.from_bytes(originalFile.read(1),
'little')
                graphImg[y, x] = [red, green, blue]

            originalFile.read(self.padding)

        textBitsCount = self.textBitsCount

        for y in range(self.height - 1, -1, -1):
            for x in range(self.width):

```

```

        if bitCounter < textBitsCount:
            binPowOffset = (pow(2, self.encodeOffset) - 1)

            bits = bits + bin(graphImg[y, x][0] &
binPowOffset)[2:].zfill(self.encodeOffset)

            bits = bits + bin(graphImg[y, x][1] &
binPowOffset)[2:].zfill(self.encodeOffset)

            bits = bits + bin(graphImg[y, x][2] &
binPowOffset)[2:].zfill(self.encodeOffset)

            bitCounter += self.encodeOffset * 3

    with open('decoded_text.txt', 'w') as decodedTxt:
        for i in range(0, len(bits), 8):
            decodedText += chr(int(bits[i:i+8], 2))
            a = 0

        decodedTxt.write(decodedText)

def EncodeTextScript():
    bmpReader = BmpFileReader('fish.BMP')
    bmpReader.Read()
    bmpReader.bmpObj.PrintInfo()
    bmpReader.bmpObj.EncodeText(0.6)
    bmpReader.bmpObj.DecodeText()
    plt.show()

if name == 'main':
    EncodeTextScript()

```

## Результат работы программы

