Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчёт

по лабораторной работе №8

**Стенографические методы**

|  |  |
| --- | --- |
| Выполнил:  студент группы 653502  Куликов А.Д. | Проверил:  Артемьев В.С. |

Минск 2019

# Постановка задачи

Создать программу, которая может реализовать сокрытия (извлечения) текстового сообщения в (из) JPEG изображение(я) на основе метода в частотной области изображения.

# Теоретическая справка

Стеганогра́фия (от [греч.](https://ru.wikipedia.org/wiki/Греческий_язык) στεγανός «скрытый»+ γράφω «пишу»; букв.«тайнопись») — способ передачи или хранения [информации](https://ru.wikipedia.org/wiki/Информация) с учётом сохранения в тайне самого факта такой передачи (хранения).

Все алгоритмы встраивания скрытой информации можно разделить на несколько подгрупп:

* Работающие с самим цифровым сигналом. Например, метод LSB.
* «Впаивание» скрытой информации. В данном случае происходит наложение скрываемого изображения (звука, иногда текста) поверх оригинала. Часто используется для встраивания цифровых водяных знаков (ЦВЗ).
* Использование особенностей форматов [файлов](https://ru.wikipedia.org/wiki/Файл). Сюда можно отнести запись информации в [метаданные](https://ru.wikipedia.org/wiki/Метаданные) или в различные другие не используемые зарезервированные поля файла.

По способу встраивания информации стегоалгоритмы можно разделить на линейные (аддитивные), нелинейные и другие. Алгоритмы аддитивного внедрения информации заключаются в линейной модификации исходного изображения, а её извлечение в декодере производится корреляционными методами. При этом ЦВЗ обычно складывается с изображением-контейнером либо «вплавляется» (fusion) в него. В нелинейных методах встраивания информации используется скалярное либо векторное квантование. Среди других методов определенный интерес представляют методы, использующие идеи фрактального кодирования изображений. К [аддитивным](https://ru.wikipedia.org/wiki/Аддитивность) алгоритмам можно отнести:

* [А17 (Cox)](https://ru.wikipedia.org/w/index.php?title=А17_(Cox)&action=edit&redlink=1);
* А18 (Barni);
* [L18D (Lange)](https://ru.wikipedia.org/w/index.php?title=L18D_(Lange)&action=edit&redlink=1);
* А21 (J. Kim);
* [А25 (С. Podilchuk)](https://ru.wikipedia.org/w/index.php?title=А25_(С._Podilchuk)&action=edit&redlink=1).

## Дискретное косинусное преобразование

Дискретное косинусное преобразование (англ. Discrete Cosine Transform, DCT) — одно из ортогональных преобразований. Вариант косинусного преобразования для вектора действительных чисел. Применяется в алгоритмах сжатия информации с потерями, например, MPEG и JPEG. Это преобразование тесно связано с дискретным преобразованием Фурье и является гомоморфизмом его векторного пространства.

Математически преобразование можно осуществить умножением вектора на матрицу преобразования. При этом матрица обратного преобразования с точностью до множителя равна транспонированной матрице. В математике матрицы выбирают так, чтобы преобразование было ортонормированным, а постоянный множитель равен единице. В компьютерных приложениях это не всегда так.

Различные периодические продолжения сигнала ведут к различным типам дискретного косинусного преобразования. Ниже приводятся матрицы для первых четырёх типов дискретного косинусного преобразования:

**A screenshot of a cell phone

Description automatically generated**

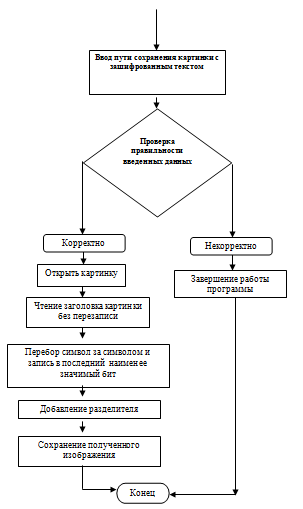
Именно DCT2 чаще всего встречается в практических приложениях благодаря свойству «уплотнения энергии».

DCT для вектора из 8 чисел часто называют DCT2\_{8}. Наиболее распространён двумерный вариант преобразования для матриц 8x8, состоящий из последовательности DCT2\_8 сначала для каждой строки, а затем для каждого столбца матрицы.

Существуют алгоритмы быстрого DCT-преобразования, похожие на алгоритм быстрого преобразования Фурье. Для DCT2\_8 и других вариантов дискретного косинусного преобразования с фиксированной размерностью вектора существуют также алгоритмы, позволяющие свести количество операций умножения к минимуму.

Существуют аналоги дискретного косинусного преобразования, приближающие косинус числами, легко получающимися путём небольшого количества операций сдвига и сложения, что позволяет избежать операций умножения и тем самым повысить скорость вычислений.

# Блок-схема алгоритма

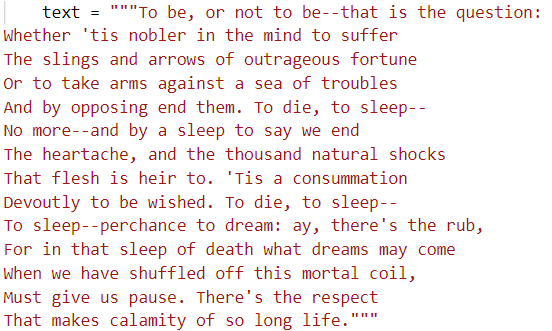


# Результат исполнения программы

****

Рисунок 1. Исходное изображение

Шифротекст:



****

Рисунок 2. Сообщение с зашифрованным текстом

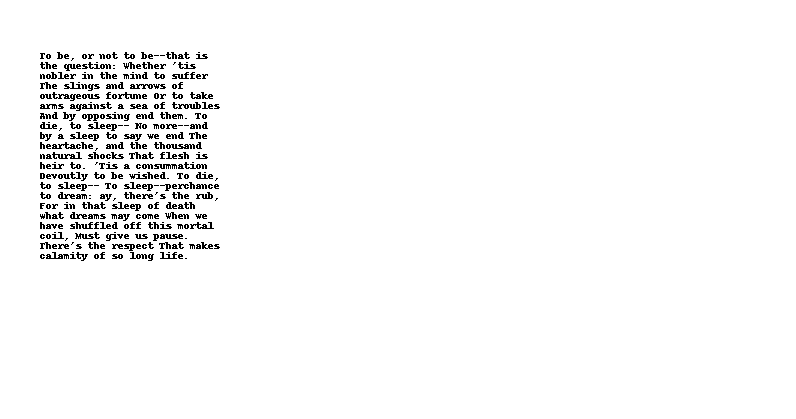


Рисунок 3. Расшифрованный текст

# Вывод

В ходе написания лабораторной работы были изучены алгоритмы сокрытия и извлечения текстового сообщения из JPEG изображений на основе метода в частотной области изображения, а также написаны их программные реализации. Были получены навыки усложнения и увеличения криптостойкости алгоритма сокрытия и извлечения текстового сообщения из JPEG изображений на основе метода в частотной области изображения, а также изучены модификации и режимы работы алгоритма сокрытия и извлечения текстового сообщения из JPEG изображений на основе метода в частотной области изображения.

# Приложение

from PIL import Image, ImageFont, ImageDraw

from textwrap import wrap

class Steganography:

@staticmethod

def \_\_write\_text(input\_text, img\_size):

img\_text = Image.new("RGB", img\_size)

drawer = ImageDraw.Draw(img\_text)

offset = 50

for line in wrap(input\_text, width=30):

drawer.text((40, offset), line, font=ImageFont.load\_default().font)

offset += 10

return img\_text

@staticmethod

def encode(input\_text, input\_image, enc\_img\_path):

rc, gc, bc, = input\_image.split()

image\_text = Steganography.\_\_write\_text(input\_text, input\_image.size)

bw\_encode = image\_text.convert('1')

enc\_img = Image.new('RGB', (input\_image.size[0], input\_image.size[1]))

pixels = enc\_img.load()

for i in range(input\_image.size[0]):

for j in range(input\_image.size[1]):

red\_template\_pix = bin(rc.getpixel((i, j)))

if bin(bw\_encode.getpixel((i, j)))[-1] == '1':

red\_template\_pix = red\_template\_pix[:-1] + '1'

else:

red\_template\_pix = red\_template\_pix[:-1] + '0'

pixels[i, j] = (int(red\_template\_pix, 2), gc.getpixel((i, j)), bc.getpixel((i, j)))

enc\_img.save(enc\_img\_path)

@staticmethod

def decode(encode\_img, img\_path):

rc = encode\_img.split()[0]

decode\_img = Image.new('RGB', encode\_img.size)

pixels = decode\_img.load()

for i in range(encode\_img.size[0]):

for j in range(encode\_img.size[1]):

if bin(rc.getpixel((i, j)))[-1] == '0':

pixels[i, j] = (255, 255, 255)

else:

pixels[i, j] = (0, 0, 0)

decode\_img.save(img\_path)

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

text = 'All the world is made of faith, and trust, and pixie dust'

encode\_img\_path = 'encode.png'

input\_img = Image.open('bsuir.png')

Steganography.encode(text, input\_img, encode\_img\_path)

encode\_image = Image.open(encode\_img\_path)

Steganography.decode(encode\_image, 'decode.png')