МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта
Направление 02.03.01 Математика и компьютерные науки

Отчет по лабораторной работе №3

по дисциплине «Функциональное программирование»

Вариант 4

Обучающийся: _____ Шихалев А.О.

Руководитель: _____ Моторин Д.Е.

«_____»_____20____г.

Санкт-Петербург, 2024

Содержание

Введение

Практическое задание №3 представляет собой реализацию следующего задания:

1. Найти портрет указанного человека:

   Ковалевская, Софья Васильевна

2. Перевести изображение в формат .bmp (24-разрядный), при необходимости изменить ширину и высоту изображения без искажений. Сохранить в файл формата .txt фрагмент биографии (не менее 1000 символов без пробелов, текст не должен обрываться на середине слова или предложения). Закодировать текст в изображение методом:

   Шифром Цезаря. Смещение задается пользователем

   Ключ к шифру записывается в имя файла. Написать функцию расшифровывающую текст из изображения используя ключ из имени файла и сохраняющую результат в отдельный текстовый файл.

3. Создать функции, шифрующие текст последний бит каждого байта, последние два бита каждого байта, ..., все биты в байте. В отчете привести примеры искажений изображения.

4. Написать второй вариант функции, шифрующую текст в изображение, а именно закодировать битовую последовательность зашифрованного текста следующим образом:

   - все чётные биты – в канал RED;
   - все нечётные биты – в канал GREEN;
   - исходную битовую последовательность канала BLUE инвертировать.

Лабораторная работа выполнена на языке Haskell в текстовом редакторе Visual Studio Code 1.95.3.

1 Математическое описание

1.1 Шифр Цезаря

Шифр Цезаря (лат. Notae Caesarianae), также известный как шифр сдвига или код Цезаря — разновидность шифра подстановки, в котором каждый символ в открытом тексте заменяется символом, находящимся на некотором постоянном числе позиций левее или правее него в алфавите (так, в шифре со сдвигом вправо на 3, А была бы заменена на Г, Б станет Д, и так далее). Шифр был назван в честь римского полководца Гая Юлия Цезаря, использовавшего его для секретной переписки со своими военачальниками.

Если сопоставить каждому символу алфавита его порядковый номер (нумеруя с 0), то шифрование и дешифрование можно выразить формулами модульной арифметики [1]:

$$y = (x + k) \mod n$$

$$x = (y - k) \mod n$$

где:
x — символ открытого текста,
y — символ шифрованного текста,
n — мощность алфавита,
k — ключ.

2 Особенности реализации

2.1 Исходное изображение и текст

Для выполнения лабораторной работы необходимо было найти изображение Ковалевской Софьи Васильевны (см. рис. 1). Изображение было переведено из формата `jpeg` в формат `bmp` с помощью сайта [3].

Рис. 1. Изображение Ковалевской С.В.

Отрывок биографии Ковалевской Софьи Васильевны длиною в 1442 символа без учёта пробелов представлен ниже.

Sofya Vasilyevna Kovalevskaya (born January 15, 1850, Moscow, Russia, died February 10, 1891, Stockholm ↪, Sweden) was a mathematician and writer who made a valuable contribution to the theory of partial ↪differential equations. She was the first woman in modern Europe to gain a doctorate in mathematics ↪, the first to join the editorial board of a scientific journal, and the first to be appointed professor ↪ of mathematics. In 1868 Kovalevskaya entered into a marriage of convenience with a young paleontologist ↪, Vladimir Kovalevsky, in order to leave Russia and continue her studies. The pair traveled together ↪ to Austria and then to Germany, where in 1869 she studied at the University of Heidelberg under the ↪ mathematicians Leo Konigsberger and Paul du Bois−Reymond and the physicist Hermann von Helmholtz ↪. The following year she moved to Berlin, where, having been refused admission to the university on account

↪ of her gender, she studied privately with the mathematician Karl Weierstrass. In 1874 she presented ↪three papers on partial differential equations, on Saturn's rings, and on elliptic integrals to the University ↪ of Gottingen as her doctoral dissertation and was awarded the degree summa cum laude in absentia ↪. Her paper on partial differential equations, the most important of the three papers, won her valuable ↪ recognition within the European mathematical community. It contains what is now commonly known ↪ as the Cauchy–Kovalevskaya theorem, which gives conditions for the existence of solutions to a certain ↪ class of partial differential equations. Having gained her degree, she returned to Russia, where her daughter ↪ was born in 1878. She separated permanently from her husband in 1881.

2.2 Кодирование и декодирование текста с помощью шифра Цезаря

Код функций для кодирования и декодирования текста с помощью шифра Цезаря представлен в листинге листинге 1. Функция `encryptCaesar` принимает алфавит в виде списка символов, смещение и сам текст, а возвращает зашифрованный текст. В её коде используется вспомогательная функция `indexOfSymbol`. Функция принимает список и элемент списка, а возвращает индекс этого элемента. Для создания алфавита используется функция `createAlphabetFromText`. Она принимает текст, а возвращает алфавит, который в нём используется, в виде списка символов. Для декодирования текста используется функция `decryptCaesar`, которая, по-сути, является лишь обёрткой над функцией `encryptCaesar`, так как процесс кодирования осуществляется почти так же как и декодирования. Функция `decryptCaesar` принимает на вход алфавит, смещение и закодированный текст, а возвращает декодированный текст.

Листинг 1. Функции для кодирования и декодирования текста с помощью шифра Цезаря.

```
1
2  createSpecialChars :: [Char]
3  createSpecialChars = [' ', '!', '(', ')', ',', '-','.', ':', ';', '?',
       ↪'\'']
4
5  createAlphabet :: [Char]
6  createAlphabet = ['A'..'Z'] ++ ['a'..'z'] ++ ['0'..'9'] ++
       ↪createSpecialChars
7
8  encryptCaesar :: [Char] -> Int -> String -> String
9  encryptCaesar alphabet shift text = map caesarChar text
10 where
11 caesarChar c = alphabet !! ((indexOf alphabet c + shift) `mod` length
       ↪alphabet)
12
13 indexOfSymbol :: [Char] -> Char -> Int
14 indexOfSymbol alphabet symbol =
15 case elemIndex symbol alphabet of
```

```haskell
Just idx -> idx
Nothing -> error "Character not found in alphabet!!!"


encryptCaesar :: [Char] -> Int -> String -> String
encryptCaesar alphabet shiftCaesar text = map caesarChar text
where
caesarChar c = alphabet !! ((indexOfSymbol alphabet c + shiftCaesar) `mod`
    ↪length alphabet)

decryptCaesar :: [Char] -> Int -> String -> String
decryptCaesar alphabet shift =
encryptCaesar alphabet (alphabetLength - (shift `mod` alphabetLength))
where
alphabetLength = length alphabet
```

Пример закодированного с помощью шифра Цезаря текста биографии Ковалевской Софьи Васильевны для смещения 5 представлен ниже.

Xtk3f,afxnq3j0sf,Pt0fqj0xpf3f,: gtws,Ofszfw3,6 ?,6(5?,Rtxht1?,Wzxxnf?,inji,Kjgwzfw3,65?,6()6?,Xythpmtqr
↪?,X1jijs;,1fx,f,rfymjrfynhnfs, fsi ,1wnyjw,1mt,rfij, f ,0 fqzfgqj , htsywngzynts,yt,ymj,ymjtw3,tk,ufwynfq,inkkjwjsynfq
↪,jvzfyntsxA,Xmj,1fx,ymj,knwxy,1trfs,ns,rtijws,Jzwtuj,yt,lfns,f , ithytwfyj , ns,rfymjrfynhx?,ymj,knwxy,yt
↪,otns,ymj,jinytwnfq,gtfwi,tk,f , xhnjsynknh,otzwsfq?,fsi, ymj,knwxy,yt,gj,fuutnsyji, uwtkjxxtw,tk,rfymjrfynhxA
↪,Ns,6(!(,Pt0fqj0xpf3f,jsyjwji , nsyt,f , rfwwnflj , tk, hts0jsnjshj ,1nym,f,3tzsl , ufqjtsytqtlnxy?, aqfinrnw,Pt0fqj0xp3
↪?,ns,twijw,yt, qjf0j , Wzxxnf,fsi, htsynszj , mjw,xyzinjxA,Ymj,ufnw,ywf0jqji,ytljymjw,yt,Fzxywnf,fsi,ymjs
↪,yt,Ljwrfs3?,1mjwj,ns,6(!),xmj,xyzinji , fy , ymj,Zsn0jwxny3,tk,Mjnijqgjwl,zsijw,ymj,rfymjrfynhnfsx,Qjt,
↪Ptsnlxgjwljw,fsi,Ufzq,iz , Gtnx'Wj3rtsi,fsi , ymj,um3xnhnxy,Mjwrfss,0ts,Mjqrmtqy4A,Ymj,ktqqt1nsl,3jfw
↪,xmj,rt0ji,yt,Gjwqns?,1mjwj?,mf0nsl,gjjs,wjkzxji,firnxxnts,yt,ymj,zsn0jwxny3,ts,fhhtzsy,tk,mjw,ljsijw?,
↪xmj,xyzinji,uwn0fyjq3,1nym,ymj,rfymjrfynhnfs,Pfwq,bjnjwxywfxxA,Ns,6("9,xmj,uwjxjsyji,ymwjj,ufujwx
↪,ts,ufwynfq,inkkjwjsynfq,jvzfyntsx?,ts,XfyzwsEx,wnslx?,fsi,ts,jqqnuynh,nsyjlwfqx,yt,ymj,Zsn0jwxny3,tk
↪,Ltyynsljs,fx,mjw,ithytwfq,inxxjwyfynts,fsi,1fx,f1fwiji,ymj,ijlwjj, xzrrf , hzr, qfzij , ns,fgxjsynfA,Mjw,ufujw
↪,ts,ufwynfq,inkkjwjsynfq,jvzfyntsx?,ymj,rtxy,nrutwyfsy,tk,ymj,ymwjj,ufujwx?,1ts,mjw,0fqzfgqj,wjhtlsnynts
↪,1nymns,ymj,Jzwtujfs,rfymjrfynhfq,htrrzsny3A,Ny,htsyfnsx,1mfy,nx,st1,htrrtsq3,pst1s,fx,ymj,Hfzhm3'
↪Pt0fqj0xpf3f,ymjtwjr?,1mnhm,ln0jx,htsinyntsx,ktw,ymj,j2nxyjshj,tk,xtqzyntsx,yt,f,hjwyfns,hqfxx,tk,ufwynfq
↪,inkkjwjsynfq,jvzfyntsxA,Mf0nsl,lfnsji,mjw,ijlwjj ?,xmj,wjyzwsji,yt,Wzxxnf?,1mjwj,mjw,ifzlmyjw,1fx,gtws
↪,ns,6("(A,Xmj,xjufwfyji,ujwrfsjsyq3,kwtr,mjw,mzxgfsi,ns,6((6A

2.3 Представление текста в виде последовательности бит

Код функций для преобразования текста в последовательность бит и обратно представлен в листинге 2. Функция `textToBits` принимает текст в виде строки и возвращает его представление в виде вектора бит. Она использует вспомогательную функцию `charToBits`,

которая преобразует символ в список бит, представляющих его код ASCII в двоичном виде. Для преобразования последовательности бит обратно в текст используется функция `bitsToText`. Она рекурсивно делит вектор бит на блоки по 8 бит, преобразует каждый блок в символ ASCII и объединяет их в строку. В процессе этого преобразования используется функция `bitsToInt`, которая преобразует вектор бит в целое число, интерпретируя их как двоичное представление этого числа.

Листинг 2. Функции для конвертации текста в последовательность бит и обратно.

```
textToBits :: String -> VU.Vector Int
textToBits text = VU.fromList $ concatMap charToBits text

charToBits :: Char -> [Int]
charToBits c = [if testBit (ord c) i then 1 else 0 | i <- [7,6..0]]

bitsToText :: VU.Vector Int -> String
bitsToText bits
| VU.null bits = []
| otherwise = (chr $ bitsToInt (VU.take 8 bits)) : bitsToText (VU.drop 8
    ↪bits)

bitsToInt :: VU.Vector Int -> Int
bitsToInt bits =
sum [bit * (2 ^ index) | (bit, index) <- zip (VU.toList bits) [len,(len -
    ↪1)..0]]
where
len = VU.length bits - 1
```

2.4 Работа с файлами

Для работы с текстовыми файлами использовались базовые функции Haskell – `readFile` (читает содержимое файла и возвращает его как строку) и `writeFile` (записывает строку в файл, заменяя его содержимое).

Для работы с изображениями использовалась библиотека `JuicyPixels` [4]. С её помощью можно как прочитать изображение в любом популярном формате, так и сохранить его. В частности в работе использовались функции: `readImage` – для чтения изображения из указанного файла, `saveBmpImage` – для сохранения изображения в формате bmp.

2.5 Сохранение зашифрованных данных в изображении

2.5.1 Первый вариант функции

Код функций для создания изображения с закодированными данными представлен в листинге 3. Функция `encodePixel` отвечает за кодирование последовательности бит в опре-

делённый пиксель изображения. Она принимает количество бит данных, которое будет сохранено в каждый байт изображения, исходное изображение, вектор бит зашифрованных данных, координаты пикселя (x, y) и возвращает новый пиксель с закодированными данными. Для этого функция вычисляет индекс пикселя в изображении, извлекает соответствующую часть вектора бит данных, преобразует её в целые числа, накладывает битовую маску, которая соответствует количество изменяемых бит в байте, и записывает закодированные данные. Для создания маски используется вспомогательная функция `createMask`.

Функция `encodePixel` затем используется вместе с функцией `generateImage` из библиотеки JuicyPixels для генерации нового изображения.

При сохранении изображения в файл, в его названии сохраняется смещение шифра Цезаря и количество бит в байте, отведённых для хранения зашифрованных данных. Например, название изображения `sofya_2_10.bmp` означает, что при кодировании использовался код Цезаря со смещением 10, а для хранения закодированных данных в каждом байте изображения использовалось 2 бита.

Листинг 3. Функции для создания изображения с закодированными данными.

```haskell
createMask :: Int -> Int
createMask shift = shiftL (complement 0) shift .&. complement 0

encodePixel :: Int -> Image PixelRGB8 -> VU.Vector Int -> Int -> Int ->
    ↪PixelRGB8
encodePixel bitsPerByte img bits x y = PixelRGB8 newR newG newB
where
width = imageWidth img

index = x + y * width
startPos = index * 3 * bitsPerByte
pixelBits = VU.slice startPos (3 * bitsPerByte) bits

bitsIntR = bitsToInt $ VU.slice 0 bitsPerByte pixelBits
bitsIntG = bitsToInt $ VU.slice bitsPerByte bitsPerByte pixelBits
bitsIntB = bitsToInt $ VU.slice (2 * bitsPerByte) bitsPerByte pixelBits

mask = createMask bitsPerByte

PixelRGB8 r g b = pixelAt img x y
newR = intToWord8 $ ((word8ToInt r) .&. mask) .|. bitsIntR
newG = intToWord8 $ ((word8ToInt g) .&. mask) .|. bitsIntG
newB = intToWord8 $ ((word8ToInt b) .&. mask) .|. bitsIntB
```

2.5.2 Второй вариант функции

Второй вариант функции `encodePixel` аналогичен первому, но с некоторыми изменениями в логике извлечения бит и обработки пикселей. Эта функция принимает такие же параметры, как и в первом варианте: количество бит данных, которые будут сохранены в каждый байт изображения, исходное изображение, вектор бит зашифрованных данных, координаты пикселя (x, y) и возвращает новый пиксель с закодированными данными.

Особенностью второго варианта является использование функции `VectorU.ifilter`, которая позволяет извлекать биты для каждого канала (красного и зелёного) по отдельности. Для канала красного (`R`) извлекаются биты с чётными индексами, а для зелёного (`G`) — с нечётными индексами. Это достигается с помощью фильтрации индексов, где для каждого индекса проверяется, является ли он чётным или нечётным. Кроме того, для синего канала (`B`) применяется операция дополнения (комплемента) значения цвета.

После извлечения битов для каждого канала они преобразуются в целые числа с помощью функции `bitsToInt`, и затем эти значения накладываются на исходные значения пикселя с использованием битовой маски, создаваемой функцией `createMask`.

Функция `encodePixel` затем используется в сочетании с функцией `generateImage` из библиотеки JuicyPixels для генерации нового изображения.

Листинг 4. Второй вариант функции сохранения зашифрованных данных в изображение

```
encodePixel :: Int -> Image PixelRGB8 -> VectorU.Vector Int -> Int -> Int
    ↪-> PixelRGB8
encodePixel bitsPerByte img bits x y = PixelRGB8 newR newG newB
where
width = imageWidth img

index = x + y * width
startPos = index * 2 * bitsPerByte
pixelBits = VectorU.slice startPos (2 * bitsPerByte) bits

bitsIntR = bitsToInt $ VectorU.ifilter (\i _ -> i `mod` 2 == 0) pixelBits
bitsIntG = bitsToInt $ VectorU.ifilter (\i _ -> i `mod` 2 == 1) pixelBits

mask = createMask bitsPerByte

PixelRGB8 r g b = pixelAt img x y
newR = intToWord8 $ ((word8ToInt r) .&. mask) .|. bitsIntR
newG = intToWord8 $ ((word8ToInt g) .&. mask) .|. bitsIntG
newB = complement b
```

2.6 Чтение зашифрованных данных из изображения

2.6.1 Первый вариант

Код функций для чтения зашифрованных данных из изображения представлен в листинге 5. Функция `extractBits` извлекает заданное количество бит из одного байта пикселя. Она принимает число бит на байт и байт пикселя, возвращая список бит. Функция `extractBitsFromPixel` предназначена для извлечения бит из всех трёх цветовых каналов (R, G, B) пикселя. Она объединяет списки бит из каждого канала в один общий список. Для извлечения бит из всего изображения используется функция `extractBitsFromImage`. Она последовательно обрабатывает все пиксели изображения, извлекая биты с помощью `extractBitsFromPixel`, и объединяет их в общий список.

Функция `extractShift` извлекает смещения для шифра Цезаря из названия файла изображения.

Листинг 5. Функции для чтения зашифрованных данных из изображения.

```
1  extractBits :: Int -> Pixel8 -> [Int]
2  extractBits bitsPerByte pixelByte =
3  [ if testBit pixelByte i then 1 else 0 | i <- [bitsPerByte-1, bitsPerByte
       ↪-2..0] ]
4
5  extractBitsFromPixel :: Int -> PixelRGB8 -> [Int]
6  extractBitsFromPixel bitsPerByte (PixelRGB8 r g b) =
7  let bitsR = extractBits bitsPerByte r
8      bitsG = extractBits bitsPerByte g
9      bitsB = extractBits bitsPerByte b
10 in bitsR ++ bitsG ++ bitsB
11
12 extractBitsFromImage :: Int -> Image PixelRGB8 -> [Int]
13 extractBitsFromImage bitsPerByte img =
14 let width = imageWidth img
15     height = imageHeight img
16     pixels = [ pixelAt img x y | y <- [0..height - 1], x <- [0..width - 1]
       ↪]
17 in concatMap (extractBitsFromPixel bitsPerByte) pixels
18
19 extractShift :: String -> Maybe Int
20 extractShift path =
21 let shift = takeWhile (`elem` ['0'..'9']) (reverse $ takeWhile (/= '_') (
       ↪reverse path))
22 in readMaybe shift
```

2.6.2 Второй вариант

Основное отличие второго варианта заключается в способе обработки бит из цветовых каналов. В данном подходе функция `extractBitsFromPixel` чередует биты из красного (R) и зелёного (G) каналов, создавая общий список бит в порядке их чередования. Для реализации этого используется вспомогательная функция `interleave`, которая принимает два списка и чередует их элементы.

Функция `extractBitsFromPixel` принимает количество бит на байт и пиксель (`PixelRGB8`), извлекая биты из каналов R и G с использованием функции `extractBits`. Затем с помощью функции `interleave` списки бит из каналов R и G объединяются в один список с чередованием элементов.

Функции `extractBits` и `extractBitsFromImage` остаются такими же, как в первом варианте, но теперь используют модифицированную `extractBitsFromPixel`. Использование функции `interleave` позволяет упорядочить биты из двух списков в чередующемся порядке. Второй вариант функции textttextractBitsFromPixel извлечение битов из пикселей изображения представлен в листинге 6.

Листинг 6. Функции для чтения зашифрованных данных из изображения (второй вариант).

```
1
2  extractBitsFromPixel :: Int -> PixelRGB8 -> [Int]
3  extractBitsFromPixel bitsPerByte (PixelRGB8 r g b) =
4  let bitsR = extractBits bitsPerByte r
5      bitsG = extractBits bitsPerByte g
6      interleave [] [] = []
7      interleave (x:xs) [] = x : interleave xs []
8      interleave [] (y:ys) = y : interleave [] ys
9      interleave (x:xs) (y:ys) = x : y : interleave xs ys
10 in interleave bitsR bitsG
```

3 Результаты работы программы

3.1 Первый вариант

При успешном завершении программа создаёт три файла: файл изображения с закодированных текстом, текстовый файл с закодированным текстом и текстовый файл с декодированным текстом.

```
PS E:\Haskell-labs\lab3> stack exec lab3-exe
Введите значение сдвига для шифра Цезаря:
5
Введите количество бит для кодирования:
1

Читаем текст из файла по пути: "src/bio.txt"
Успешно считали текст!

------------------------Шифрование текста--------------------------

Размер алфавита равен 74
Шифр в двоичном представлении: "[0,1,0,1,1,0,0,0,0,1,1,1,0,1,0,0,0,1,1,0,1,0,1,1,0,0,1,1,0,0]"

-------------------Кодирование текста в изображение--------------------

Ширина: 640 Высота: 779 пикселей
Изображение сохранено по пути: "src/sofya-1-5.bmp"

-------------------Декодирование текста из изображения--------------------

Из названия файла извлечен сдвиг: 5
Шифр в двоичном представлении: "[0,1,0,1,1,0,0,0,0,1,1,1,0,1,0,0,0,1,1,0,1,0,1,1,0,0,1,1,0,0]"
Успешно дешифровали!  Первые 20 символов: "Sofya Vasilyevna Kov"
Текст сохранён по пути: "src/decoded-bio.txt"
```

Рис. 2. Результаты работы программы в консоли.

На Рис. 2 представлены результаты работы программы в консоли.

(a) 1 бит. (b) 2 бита. (c) 3 бита.

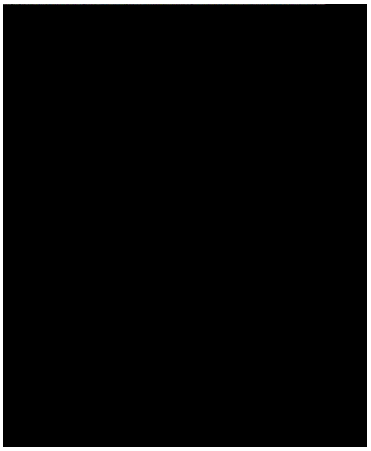
Рис. 3. Изображения с зашифрованными данными.

(a) 4 бита.

(b) 5 бит.

(c) 6 бит.

(d) 7 бит.



(e) 8 бит.

Рис. 4. Изображения с зашифрованными данными.

На рисунках 3a–3c и 4a–4e представлены результирующие изображения с разным количеством бит, отведённых под зашифрованные данные.

3.2 Второй вариант

Аналогично первому случаю, программа при успешном завершении создаст три файла с выводом в консоль путей сохранения файлов и изображения. Результаты работы программмы в консоли представлены на рис. 5.

```
PS E:\Haskell-labs\lab3> stack exec lab3-exe
Введите значение сдвига для шифра Цезаря:
5
Введите количество бит для кодирования:
4

Читаем текст из файла по пути: "src/bio.txt"
Успешно считали текст!

-------------------------Шифрование текста-------------------------

Размер алфавита равен 74
Шифр в двоичном представлении: "[0,1,0,1,1,0,0,0,0,1,1,1,0,1,0,0,0,1,1,0,1,0,1,1,0,0,1,1,0,0]"

-------------------Кодирование текста в изображение-------------------

Ширина: 640 Высота: 779 пикселей
Изображение сохранено по пути: "src/sofya-4-5.bmp"

-------------------Декодирование текста из изображения-------------------

Из названия файла извлечен сдвиг: 5
Шифр в двоичном представлении: "[0,1,0,1,1,0,0,0,0,1,1,1,0,1,0,0,0,1,1,0,1,0,1,1,0,0,1,1,0,0]"
Успешно дешифровали!  Первые 20 символов: "Sofya Vasilyevna Kov"
Текст сохранён по пути: "src/decoded-bio.txt"
```

Рис. 5. Результаты работы программы в консоли.

(a) 1 бит.

(b) 2 бита.

(c) 3 бита.

(d) 4 бита.

(e) 5 бит.

(f) 6 бит.

Рис. 6. Изображения с зашифрованными данными.

На рисунках 6a–6f и 7a–7b представлены результирующие изображения с разным ко-

(a) 7 бит.

(b) 8 бит.

Рис. 7. Изображения с зашифрованными данными.

личеством бит, отведённых под зашифрованные данные.

Заключение

В результате выполнения лабораторной работы была создана программа на языке Haskell, которая способна кодировать текстовые данных из указанного файла с помощью шифра Цезаря и сохранять эти данные внутрь изображения. Причём программа позволяет выбрать как смещение для шифра Цезаря, так и количество бит, которое будет использовано в каждом байте изображения для хранения данных.

Список литературы

[1] Luciano, D., Prichett, G., Cryptology: From Caesar Ciphers to Public-Key Cryptosystems, The College Mathematics Journal, 1987.

[2] David Deutsch – personal website, URL: https://www.daviddeutsch.org.uk/, Дата обращения: 19.11.2024

[3] Convertio – BPM to JPG online converter, URL: https://convertio.co/ru/bmp-jpg/, Дата обращения: 19.11.2024

[4] Hackage – JuicyPixels: Picture loading/serialization, URL: https://hackage.haskell.org/package/JuicyPixels, Дата обращения: 19.11.2024