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 практическому заданию №3

по дисциплине «Функциональное программирование»

Обучающийся: _____ Гладков И.А.

Руководитель: _____ Моторин Д.Е.

«_____»_____20____г.

Санкт-Петербург, 2024

Содержание

Введение

В данном отчете описан результат выполнения практического задания №3. Целью которого было шифрование текста биографии предоставленного ученного в изображение и дешифрирование текста из изображения.

Шифрование текста происходит в два этапа: сначала текст шифруется шифром с использованием кодового слова (кодовое слово выбирает пользователь), затем зашифрованный текст записывается в один, два, ... , восемь последних битов каждого байта изображения, на усмотрение пользователя. Дешифрирование выполняется с помощью ключа и количество бит, записанных в названии изображения, в котором зашифрован текст.

Практическая работа выполняется на языке Haskell 9.4.8 в текстовом редакторе Visual Studio Code 1.95.3

1 Постановка задач

Для выполнения практического задания необходимо выполнить следующие задачи:

1. Создать проект в `stack`
2. Все чистые функции записать в библиотеку `Lib.hs` и ограничить доступ к вспомогательным функциям
3. Использовать do-нотацию для работы с внешними файлами.
4. Использовать портрет Алонзо Черча в формате `.bmp` и текст с его биографией, содержащим в себе не менее 1000 символов без пробелов.
5. Реализовать функцию кодирующую текст в изображение с помощью шифра с использование кодового слова, где кодовое слово задает пользователь. Далее ключ шифру записывается в имя файла.
6. Написать функцию расшифровывающую текст из изображения используя ключ из имени файла и сохраняющую результат в отдельный текстовый файл.
7. Создать функции шифрующие текст в последний бит каждого байта, последние два бита каждого байта, ..., все биты в байте изображения.

2 Математическое описание

2.1 Шифр с использованием кодового слова

Шифр с использованием кодового слова представляет собой модифицированную версию шифра подстановки, в котором каждый символ открытого текста шифруется с использованием соответствующего символа кодового слова. Кодовое слово повторяется, если его длина меньше длины текста. Такой подход обеспечивает псевдослучайность шифрования и делает метод более устойчивым к криптоанализу по сравнению с простыми шифрами сдвига.

Пусть P — открытый текст, C — шифротекст, а K — кодовое слово. Если каждому символу алфавита сопоставить его порядковый номер, начиная с 0, то шифрование и дешифрование можно описать следующими формулами модульной арифметики:

$$c_i = (p_i + k_{(i \mod m)}) \mod n$$

$$p_i = (c_i - k_{(i \mod m)}) \mod n$$

где:

p_i — i-й символ открытого текста,

c_i — i-й символ шифротекста,

k_j — j-й символ кодового слова,

n — мощность алфавита,

m — длина кодового слова.

Для расшифрования используется обратное преобразование, основанное на вычитании значения символа кодового слова.

3 Особенности реализации

3.1 Исходное изображение и текст

В работе использовались 24-разрядное изображение формата `.bmp` и текстовый файл содержаший 1085 символов без пробелов.

Портер и биография Алонзо Черча представлены ниже:

Alonzo Church's parents were Mildred Hannah Letterman Parker and Samuel Robbins Church. His father was a judge. He was a student at Princeton receiving his first degree, an A.B., in 1924, then his doctorate three years later. His doctoral work was supervised by Veblen, and he was awarded his doctorate in 1927 for his dissertation entitled Alternatives to Zermelo's Assumption. While he was still working for his doctorate he married Mary Julia Kuczinski at Princeton in 1926. They had three children, Alonzo Jr, Mary Ann and Mildred. Church spent two years as a National Research Fellow, one year at Harvard University then a year at Gottingen and Amsterdam. He returned to the United States becoming Assistant Professor of Mathematics at Princeton in 1929.He was promoted to Associate Professor in 1939 and to Professor in 1947, a post he held until 1961 when he became Professor of Mathematics and Philosophy. In 1967 he retired from Princeton and went to the University of California at Los Angeles as Kent Professor of Philosophy and Professor of Mathematics. He continued teaching and undertaking research at Los Angeles until 1990 when he retired again, twenty-three years after he first retired! In 1992 he moved from Los Angeles to Hudson, Ohio, where he lived out his final three years.

3.2 Основной модуль

Основной модуль программы, `Main`, реализует меню для выбора действий: шифрования и расшифровки текста, а также управления путями к файлам и изображениями. В нем определены функции для работы с изображениями и текстами, а также для шифрования и расшифровки данных.

```
1  module Main (main) where
2
3  import Lib
4  import Codec.Picture
5  import System.IO
6  import Control.DeepSeq (deepseq)
7  import Control.Exception (IOException, try)
```

```
8
9   main :: IO ()
10  main = do
11  menu
12
13  menu :: IO()
14  menu = do
15  putStrLn "Choose the action:"
16  putStrLn "[1] - encrypting text"
17  putStrLn "[2] - decrypting text"
18  putStrLn "[0] - Bye!"
19  input <- getLine
20  if null input  -- Проверка на пустой ввод
21  then do
22  putStrLn "You must choose a valid option. Please try again."
23  menu
24  else do
25  let n = read input :: Int
26  case n of
27  1 -> do
28  menuEncrypting
29  putStrLn "---------------------------------"
30  menu
31  2 -> do
32  menuDecrypting
33  putStrLn "---------------------------------"
34  menu
35  0 -> putStrLn "Exiting..."
36  _ -> do
37  putStrLn "Invalid choice. Try again."
38  putStrLn "---------------------------------"
39  menu
```

Листинг 1. Основной модуль программы

3.2.1 Шифрование текста

Функция `menuEncrypting` обрабатывает процесс шифрования текста. Пользователь может указать путь к изображению, текстовому файлу и ключ шифрования. После получения всех данных, программа шифрует текст и сохраняет изображение с зашифрованными данными.

```
1   menuEncrypting :: IO()
2   menuEncrypting = do
3   putStrLn "write path to the image (default \"files\\Alonzo_Church.bmp\")"
4   input <- getLine
5   let path = if null input then  "files\\Alonzo_Church.bmp" else input :: String
6   key = getKey path
7   eimg <- readImage path
```

```
  case eimg of
  Left err -> putStrLn $ "Failed to open image: " ++ err
  Right dynImg -> do
  let img = convertRGB8 dynImg
  binaryImage = imageToBinary img
  height = imageHeight img
  width = imageWidth img
  putStrLn $ "image size: " ++ (show width) ++ " x " ++ (show height)

  putStrLn "write path to .txt file (default \"files\\Alonzo Church's Biography.txt\")"
  inputF <- getLine
  let fileName = if null inputF then  "files\\Alonzo Church's Biography.txt" else inputF
    ↪:: String
  openning <- try (readFile fileName) :: IO (Either IOException String)
  case openning of
  Left ex -> putStrLn $ "Failed to open file: " ++ show ex
  Right text -> do
  let binaryEncodedText = binaryEncoding $ encoding key text
  putStrLn $ "Lenght of text in bits: " ++ show  (length binaryEncodedText)
  putStrLn "Write Key for encoding (default \"Haskell\"):"
  input1 <- getLine
  let key = if null input1 then "Haskell" else input1 :: String

  putStrLn "Write count of bits (default 4):"
  input2 <- getLine
  let inp = if null input2 then 4 else read input2 :: Int
  count <- check inp

  putStrLn "Coordinates for encrypting write coordinate x (default 0):"
  inputX <- getLine
  let coordX = if null inputX then 0 else read inputX :: Int
  putStrLn "Coordinates for encrypting write coordinate y (default 0):"
  inputY <- getLine
  let coordY = if null inputY then 0 else read inputY :: Int

  (x, y) <- coordinate (coordX, coordY, width, height, count, (length binaryEncodedText))

  let encryptedText = encrypting binaryImage binaryEncodedText count x y
  img' = binaryStringToImage encryptedText width height
  img'' = ImageRGB8 img'
  putStrLn "text is encrypting..."
  encryptedText `deepseq` putStrLn "text encrypted..."
  putStrLn "image is saving..."
  let filePath = "files/" ++ key ++ "_" ++ show count ++ ".bmp"
  saveBmpImage filePath img''
  putStrLn $ "Image saved to \"" ++ filePath ++ "\""
```

Листинг 2. Шифрование текста

3.2.2 Дешифровка текста

Функция `menuDecrypting` обрабатывает процесс дешифровки текста. Пользователь может указать путь к зашифрованному изображению и файл для сохранения расшифрованного текста. Программа извлекает данные из изображения, расшифровывает их и сохраняет в файл.

```
menuDecrypting :: IO()
menuDecrypting = do
putStrLn "write path to the image (default \"files\\Haskell_4.bmp\")"
input <- getLine
let path = if null input then  "files\\Haskell_4.bmp" else input :: String
key = getKey path
eimg <- readImage path
case eimg of
Left err -> putStrLn $ "Failed to open image: " ++ err
Right dynImg -> do
let img = convertRGB8 dynImg
binaryImage = imageToBinary img
(key, bits) = getKeyBits path
bText = decrypting binaryImage bits
binaryDecodedText = binaryDecoding bText
decodedText = decoding key binaryDecodedText
putStrLn $ "write path to save decrypted image (default \"files\\decrypdedImage.txt\")"
input1 <- getLine
let fileName = if null input1 then "files\\decrypdedImage.txt" else input1 :: String

putStrLn "image is decrypting..."
bText `deepseq` putStrLn "image decrypted..."
putStrLn "text is saving..."
file <- openFile fileName WriteMode
hPutStr file decodedText
hClose file
putStrLn $ "Decrypted image saved to \"" ++ fileName ++ "\""
```

Листинг 3. Дешифровка текста

3.3 Библиотека `Lib.hs`

В данном разделе приведены основные функции, реализованные в проекте на языке Haskell.

3.3.1 Функция `removeDublicates`

Удаляет дубликаты из строки. Функция использует вспомогательную рекурсивную функцию для обхода всех символов в строке.

```
removeDublicates :: String -> String
removeDublicates = removeDublicates' []
where
```

```haskell
removeDublicates' :: String -> String -> String
removeDublicates' _ [] = []
removeDublicates' seen (x:xs)
| x `elem` seen = removeDublicates' seen xs
| otherwise = x : removeDublicates' (x:seen) xs
```

Листинг 4. Функция удаления дубликатов

3.3.2 Функция `convertText`

Преобразует текст с одного алфавита в другой, используя таблицу преобразования (список пар символов).

```haskell
convertText :: String -> String -> String -> String
convertText [] _ _ = []
convertText (c:cs) alphabet newAlphabet =
let converted = case lookup c (zip alphabet newAlphabet) of
Just x -> x
Nothing -> c
in converted : convertText cs alphabet newAlphabet
```

Листинг 5. Функция конвертации текста

3.3.3 Функция `createAlphabet`

Создает алфавит на основе переданного ключа, добавляя в него уникальные символы.

```haskell
createAlphabet :: String -> String
createAlphabet [] = " !\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_
  ↪`abcdefghijklmnopqrstuvwxyz{|}"
createAlphabet str =
let newStr = str ++ " !\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_
  ↪`abcdefghijklmnopqrstuvwxyz{|}"
in  removeDublicates newStr
```

Листинг 6. Функция создания алфавита

3.3.4 Функция `encoding`

Шифрует текст с помощью кодировки, основанной на кодовом слове.

```haskell
encoding :: String -> String -> String
encoding key text =
let alphabet = createAlphabet []
newAlphabet = createAlphabet key
in convertText text alphabet newAlphabet
```

Листинг 7. Функция шифрования

3.3.5 Функция `decoding`

Декодирует текст с использованием кодового слова, возвращая исходный текст.

```
decoding :: String -> String -> String
decoding key text =
let alphabet = createAlphabet key
defaultAlphabet = createAlphabet []
in convertText text alphabet defaultAlphabet
```

Листинг 8. Функция декодирования

3.3.6 Функции для работы с двоичным кодом

Эти функции позволяют преобразовывать числа, символы и строки в двоичную форму и обратно.

```
toBinary :: Int -> Vector Char
toBinary n = V.fromList (replicate (8 - length bin) '0') V.++ V.fromList bin
where bin = showIntAtBase 2 intToDigit n ""

fromBinary :: Vector Char -> Int
fromBinary str
| V.null str = 0
|otherwise = num + fromBinary str'
where
num = if V.head str == '1' then 2 ^ (length str - 1) else 0
str' = V.tail str

charToBinary :: Char -> Vector Char
charToBinary c = toBinary $ ord c

charFromBinary :: Vector Char -> Char
charFromBinary c =  chr $ fromBinary c

binaryEncoding :: String -> Vector Char
binaryEncoding [] = V.empty
binaryEncoding (c:cs) = charToBinary c V.++ binaryEncoding cs

binaryDecoding :: Vector Char -> String
binaryDecoding str
| V.null str = []
| V.length str < 8 = []
| otherwise = charFromBinary (V.take 8 str) : binaryDecoding (V.drop 8 str)
```

Листинг 9. Функции для работы с двоичным кодом

3.3.7 Функции для работы с изображениями

Эти функции преобразуют пиксели и изображения в двоичный формат и обратно.

```
pixelToBinary :: PixelRGB8 -> Vector Char
pixelToBinary (PixelRGB8 r g b) = toBinary (fromIntegral r) V.++ toBinary (fromIntegral
  ↪ g) V.++ toBinary (fromIntegral b)

imageToBinary :: Image PixelRGB8 -> Vector Char
imageToBinary img = V.concat [pixelToBinary (pixelAt img x y) | y <- [0..height-1], x
  ↪<- [0..width-1]]
where
width = imageWidth img
height = imageHeight img
```

Листинг 10. Функции для работы с изображениями

3.3.8 Функции для работы с двоичной строкой и изображениями

Эти функции обрабатывают двоичные строки, преобразуя их в изображения и обратно.

```
binaryStringToColor :: Vector Char -> Word8
binaryStringToColor str = fromIntegral $ foldl (\acc x -> acc * 2 + digitToInt x) 0 str

binaryStringToImage :: Vector Char -> Int -> Int -> Image PixelRGB8
binaryStringToImage str width height =
let size = height * width * 3 * 8
newStr = takeCurrentSize str size
pixelRenderer x y =
let index = (y * width + x) * 3 * 8
r = binaryStringToColor (V.take 8 (V.drop index newStr))
g = binaryStringToColor (V.take 8 (V.drop (index + 8) newStr))
b = binaryStringToColor (V.take 8 (V.drop (index + 16) newStr))
in PixelRGB8 r g b
in generateImage pixelRenderer width height
```

Листинг 11. Функции для работы с двоичными строками и изображениями

3.3.9 Функция `getKey`

Извлекает ключ из пути к файлу.

```
getKey :: String -> String
getKey path =
takeWhile (/= '.') (reverse (takeWhile (\c -> c /= '\\' && c /= '/') (reverse path)))
```

Листинг 12. Функция получения ключа

3.3.10 Функция getKeyBits

Извлекает ключ и количество бит из пути к файлу.

```
getKeyBits :: String -> (String, Int)
getKeyBits path =
let fileName = takeWhile (/= '.') (reverse (takeWhile (\c -> c /= '\\' && c /= '/') (
  ↪reverse path)))
key = takeWhile (/= '_') fileName
bits = takeWhile (/= '.') (filter  (/= '_') (dropWhile (/= '_') fileName))
bits' = read bits::Int
in (key, bits')
```

Листинг 13. Функция получения ключа и битов

3.3.11 Функция encrypting

Шифрует изображение, используя двоичный код и заданное количество бит.

```
encrypting :: Vector Char -> Vector Char -> Int -> Int -> Int -> Vector Char
encrypting img str count x y = runST $ do
let indx = 0
len = length img
str' = takeCurrentSize str (len `div` 8 * count)
result <- MV.new (len)

let go indx
| indx < x * y + y = do
V.forM_ (V.fromList [0..7]) $ \i -> do
let val = if str' !! indx == '1' then 1 else 0
MV.write result indx val
go (indx +1)
go indx
return result
```

Листинг 14. Функция шифрования изображения

4 Результаты

Запускаем программу, для зашифровки текста выбираем `«encrypting text»` и пишем путь к изображению, путь к текстовому файлу, кодовое слово и количество бит. Для расшифровки текста выбираем `«decrypting text»`, пишем путь к зашифрованному тексту, то есть изображению, после пишем куда сохранить текст после расшифровки.

Ниже приведены результаты программы шифрования текста в изображение с количеством битов от 1 до 8 и кодовым словом `«Haskell»`. При расшифровки каждого изображения получался изначальный текст.

Рис. 1. Оригинальное изображение

Рис. 2. Шифрование текста в 1 бит байта

Рис. 3. Шифрование текста в 2 бита байта

Рис. 4. Шифрование текста в 3 бита байта

Рис. 5. Шифрование текста в 4 бита байта

Рис. 6. Шифрование текста в 5 битов байта

Рис. 7. Шифрование текста в 6 битов байта

Рис. 8. Шифрование текста в 7 битов байта

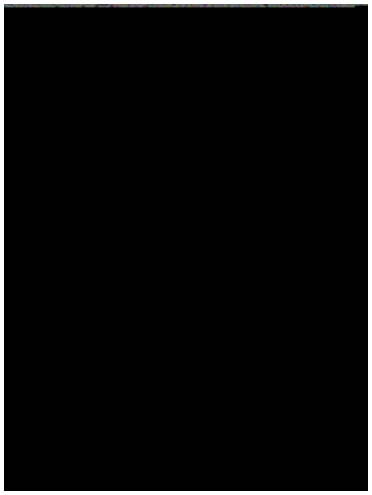

Рис. 9. Шифрование текста в 8 битов байта

Как можно заметить, чем большее количество битов в каждом байте используется для записи текста,тем сильнее изменяется изображение.

Проверим работу функции `encryptingCoordinates`. Запускаем программу, для зашифровки текста выбираем `«encrypting text with coordinates»` и пишем путь к изображению, путь к текстовому файлу, кодовое слово и количество бит, затем выбираем координаты по x и y.

Ниже приведены результаты программы шифрования текста в изображение с количеством битов от 1 до 8 и кодовым словом `«Haskell»`, начиная с координат 100 по x и 100 по y.

Рис. 10. Оригинальное изображение

Рис. 11. Шифрование текста в 1 бит байта

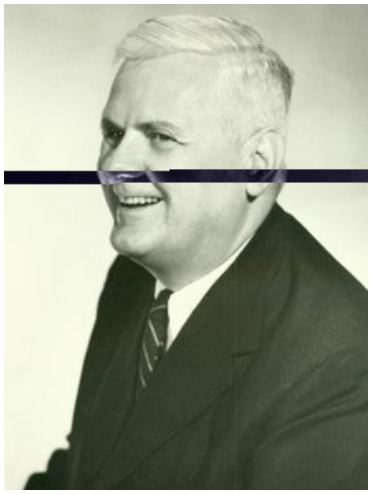

Рис. 12. Шифрование текста в 2 бита байта

Рис. 13. Шифрование текста в 3 бита байта

Рис. 14. Шифрование текста в 4 бита байта

Рис. 15. Шифрование текста в 5 битов байта

Рис. 16. Шифрование текста в 6 битов байта

Рис. 17. Шифрование текста в 7 битов байта

Рис. 18. Шифрование текста в 8 битов байта

Как можно заметить, чем большее количество битов в каждом байте используется для записи текста,тем сильнее изменяется изображение.

Заключение

В ходе выполнения практической работы был создан проект в `stack`, все чистые функции были записаны `Lib.hs` и был ограничен доступ к вспомогательным функциям. Использована do-нотация для работы с файлами. Были использованы портрет и биография Алонзо Черча. Была реализована функция шифрующая изображения в текст с помощью шифра с использованием кодового слова, а также реализована функция записывающая текст в последний бит, последние два бита, ..., все биты каждого байта изображения.

Список литературы

[1] Курт У. *Программируй на Haskell* / пер. с англ. С. Соловьева. — Москва: ДМК Пресс, 2019. — 384 с.

[2] Church A. Alan Mathison Church. Биография // MacTutor History of Mathematics Archive. URL: https://mathshistory.st-andrews.ac.uk/Biographies/Church/ (дата обращения: 01.12.2024).