Шифр Цезаря

**Цель работы**

Произвести шифрование кодом Цезаря, сжатого при помощи кодирования длин серий изображения, передаваемого по факсу. Ознакомиться на примере кода Цезаря с поточными алгоритмами шифрования.

**Задание**

1. Смоделировать процесс передачи документа по факсу со сжатием, реализовав следующий набор операций:

· открыть изображение (скриншот страницы книги или документа в формате А4);

· преобразовать его в бинарное изображение;

· представить результат в виде вектора;

· Обойти данный вектор и сформировать на его основе вектор в формате: [число белых пикселей, число черных пикселей, число белых пикселей, число черных пикселей...]

· сохранить содержимое вектора в бинарный файл.

2. Смоделировать процесс приема сжатого содержимого документа по факсу, реализовав следующий набор операций:

· прочитать бинарный файл в вектор;

· Обратить операцию сжатия их пункта 1;

· представить полученный вектор в виде матрицы-бинарного изображения;

· сохранить полученное изображение в файл.

3. Произвести шифрование кодом Цезаря:

· прочитать бинарный файл в вектор V;

· Для каждого значения Vi прибавлять к нему значение ключа k по модулю максимального числа для вашего типа данных max\_val: ( Vi + k ) mod max\_val;

· сохранить полученный вектор в бинарный файл.

4. Реализовать обратную операцию расшифровки с известным ключом.

Дешифровать, не зная ключа. Для этого:

5. произвести частотный анализ содержимого сжатого незашифрованного вектора.

· Взять 10 файлов-картинок с текстом с близкими свойствами (похожий размер шрифта и одинаковый формат страницы), обеспечить при этом чтобы они были одинаковой размерности.

· Сжать их в соответствии с заданием 1.

· Посчитать для сжатого их представления, сколько раз встречается там каждое значение.

· Таким образом найти первую десятку наиболее встречающихся значений в сжатых незашифрованных векторах.

6. Произвести анализ зашифрованного файла, воспользовавшись известным статистическими параметрами распределения значений в незашифрованных.

· Посчитать для зашифрованного файла, сколько раз встречается там каждое значение.

· Перебрать первые несколько наиболее встречающихся значений в сжатых незашифрованных векторах.

· Проверить каждое на соответствие наиболее частому значению из зашифрованного.

· Подсчитать значение ключа так, как если бы самое частое значение в зашифрованном файле соответствовало текущему из перебираемых.

· Попытаться расшифровать файл этим ключом. Если получается изображение заданной размерности, записать полученный файл как кандидат на правильный вариант.

· Проверить получившиеся файлы вручную.

**Код**

import cv2;

import numpy as np;

from numpy import append

from numpy import empty

#функция для запись в файл вектора

def writeFile(text, name):

    file=open(name, 'w')

    file.write(text)

    file.close()

#функция для преобразования бинарного изображения в виде вектора;

def convertFile(filename):

    image = cv2.imread(filename, 2);

    \_, bw\_img = cv2.threshold(image, 127, 255, cv2.THRESH\_BINARY);

    vector\_bw = tuple(np.reshape(bw\_img, -1)); #преобразуем в вектор

    #vector\_bw = vectorBinImage.tolist()

    first\_count, vector\_of\_counts\_black\_and\_white = compression\_to\_bw\_format(vector\_bw)

    return first\_count, vector\_of\_counts\_black\_and\_white, image

def compression\_to\_bw\_format(vector):

    vector\_black\_and\_white = []

    i = white\_index = black\_index = 0

    first\_count =  "BLACK" if not vector[i] else  "WHITE"

    while i < len(vector):

        if not vector[i]:

            try:

                black\_index = vector.index(255, i)

            except ValueError:

                black\_index = len(vector)

        else:

            try:

                white\_index = vector.index(0, i)

            except ValueError:

                white\_index = len(vector)

        count = abs(black\_index - white\_index)

        vector\_black\_and\_white.append(count)

        i += count

    return first\_count, vector\_black\_and\_white

def compression\_to\_back\_bw\_format(first\_count, bw\_vector\_black\_and\_white, img):

    bw\_vector = []

    if first\_count == "BLACK":

      for i in range(len(bw\_vector\_black\_and\_white)):

        bw\_vector.extend((255 if i & 1 else 0,) \* bw\_vector\_black\_and\_white[i])

    else:

        for i in range(len(bw\_vector\_black\_and\_white)):

          bw\_vector.extend((0 if i & 1 else 255,) \* bw\_vector\_black\_and\_white[i])

    matrix\_from\_vector = empty(shape=(0, 0))

    for j in range(len(img)):

        matrix\_from\_vector = append(matrix\_from\_vector, bw\_vector[len(img[0]) \* j:len(img[0]) \* (j + 1)])

    matrix\_from\_vector = matrix\_from\_vector.reshape(len(img), len(img[0]))

    return matrix\_from\_vector

def encrypt(vector, key):

    return tuple(int((item + key) % 65535) for item in vector)

def decrypt(encrvector, key):

    return tuple(int((item - key) % 65535) for item in encrvector)

#частотный анализ

def frequencyAnalysis(vector):

    freqDict={ }

    vectorSet = set(vector)

    for i in vectorSet:

      freqDict[i] = vector.count(i)

    return freqDict

Исходное изображение:



Код:

key = 1717

vector = convertFile('/content/drive/MyDrive/pyth/sibirskij-kotik3.jpg');

#print(vector);

writeFile(str(vector), '/content/drive/MyDrive/pyth/cat\_img.txt')

vector\_encrypt = encrypt(vector[1], key)

writeFile(str(vector\_encrypt), '/content/drive/MyDrive/pyth/cat\_bin\_encrypt.txt')

vector\_decrypt = decrypt(vector\_encrypt, key)

decrypted\_bw\_vector = compression\_to\_back\_bw\_format(vector[0], vector\_decrypt, vector[2])

print(decrypted\_bw\_vector)

#writeFile(str(vector\_decrypt), '/content/drive/MyDrive/pyth/cat\_bin\_decrypt.txt')

cv2.imwrite('/content/drive/MyDrive/pyth/cat\_negw\_file.jpg', decrypted\_bw\_vector)

Результат:



Произвести частотный анализ содержимого сжатого незашифрованного вектора.

result=[]

for i in range(1,11):

        #img = cv2.imread(f'/content/drive/MyDrive/pyth/text{i}.png', 2)

        #\_, bw\_img = cv2.threshold(img, 127, 255, cv2.THRESH\_BINARY)

        #bw\_vector = tuple(np.reshape(bw\_img, -1))

        first\_count, vector\_black\_and\_white, img = convertFile(f'/content/drive/MyDrive/pyth/text{i}.jpg')

        #print(vector\_black\_and\_white)

        freqdict =frequencyAnalysis(vector\_black\_and\_white)

        res = dict(sorted(freqdict.items(), key=lambda x: x[1], reverse=True)[:10])

        #print(res)

        result.append(res)

print(result)

**Результаты**

Произвести частотный анализ содержимого сжатого незашифрованного вектора.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Text1 | Text2 | Text3 | Text4 | Text5 | Text6 | Text7 | Text8 | Text9 | Text10 |
| {3: 5012, 2: 4722, 5: 2179, 6: 1867, 9: 1687, 8: 1607, 4: 1404, 7: 873, 10: 763, 11: 529} | {3: 5214, 2: 4914, 5: 2370, 8: 1915, 9: 1788, 6: 1735, 4: 1411, 7: 904, 10: 778, 11: 449} | {3: 4639, 2: 4284, 5: 1983, 8: 1671, 9: 1608, 6: 1537, 4: 1288, 7: 811, 10: 700, 11: 396} | {3: 3574, 2: 3556, 5: 1518, 9: 1472, 6: 1471, 8: 1195, 4: 1022, 7: 633, 10: 467, 11: 343} | {2: 2968, 3: 2763, 6: 1283, 5: 1264, 8: 994, 9: 976, 4: 723, 7: 484, 10: 316, 11: 285} | {3: 6369, 2: 6108, 5: 2752, 6: 2439, 8: 2203, 9: 2044, 4: 1771, 7: 1151, 10: 1097, 11: 599} | {2: 3171, 3: 3112, 5: 1449, 6: 1133, 8: 1129, 9: 1038, 4: 912, 7: 604, 10: 457, 11: 338} | {3: 1579, 2: 1476, 5: 639, 8: 595, 6: 508, 9: 474, 4: 442, 7: 289, 10: 285, 11: 173} | {2: 5116, 3: 4719, 5: 2023, 6: 1946, 8: 1786, 9: 1746, 4: 1297, 7: 897, 10: 827, 11: 493} | {2: 3126, 3: 2603, 5: 1263, 8: 1130, 6: 1013, 9: 967, 4: 748, 7: 530, 10: 432, 11: 330} |

Провести эксперименты по дешифровке без ключа таким образом сформированных трех файлов:

keys= (1002, 1111, 1234)

frequency\_dicts=[]

for i in range(1,4):

      first\_count, vector\_black\_and\_white, img = convertFile(f'/content/drive/MyDrive/pyth/texted{i}.jpg')

      ervector = encrypt(vector\_black\_and\_white, keys[i-1])

      dict\_vector\_encrypt = frequencyAnalysis(ervector)

      dict\_vector\_encrypt = dict(sorted(dict\_vector\_encrypt.items(), key=lambda item: item[1], reverse=True)[:10])

      frequency\_dicts.append(dict\_vector\_encrypt)

      print(dict\_vector\_encrypt)

      frequency\_dict  = frequencyAnalysis(vector\_black\_and\_white)

      frequency\_dict = dict(sorted(frequency\_dict.items(), key=lambda item: item[1], reverse=True)[:10])

      print(frequency\_dict)

      key = tuple(frequency\_dict.keys())[0]

      print(key)

      decrypted\_vector = None

      start = time.perf\_counter()

      for key\_encrypted in dict\_vector\_encrypt.keys():

        #print(key\_encrypted)

        decrypted\_vector = decrypt(ervector, key\_encrypted - key)

        #print(decrypted\_vector)

        if decrypted\_vector == tuple(vector\_black\_and\_white):

            print(key\_encrypted-key)

            stop = time.perf\_counter()

            print( "Text ", stop-start)

            break

      vector\_bw = compression\_to\_back\_bw\_format(first\_count, decrypted\_vector, img)

      cv2.imwrite(f'/content/drive/MyDrive/pyth/text\_for\_encr\_file{i}.jpg', vector\_bw)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название файла | Наиболее частое значение в зашифрованном виде | Наиболее частое значение в расшифрованном виде | Правильный вариант ключа (само значение ключа) | Номер подошедшего значения из таблицы с частотным анализом незашифрованных сжатых файлов | Время, затраченное на перебор ключей и расшифровку |
| Texted1 | 1005: 4468 | 3: 4468 | 1002 | 1 | 0.00411 |
| Texted2 | 1114: 5922 | 3: 5922 | 1111 | 1 | 0.0069 |
| Texted3 | 1237: 5077 | 3: 5077 | 1234 | 1 | 0.0046 |

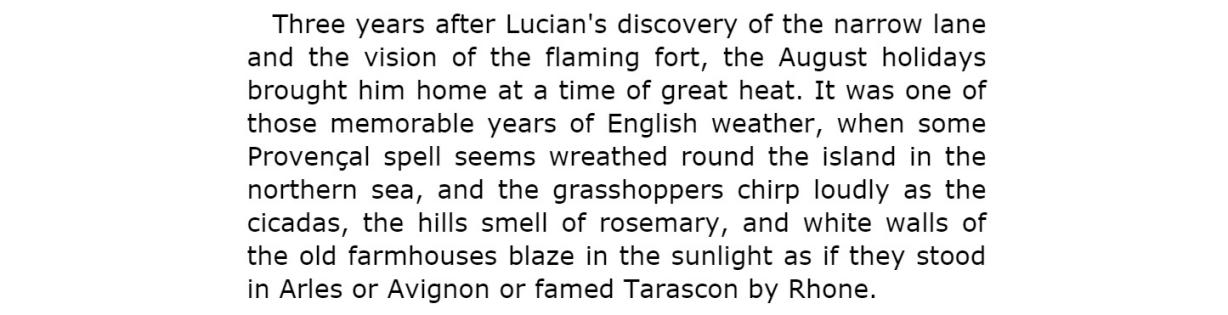
**Выводы**

В данной лабораторной работе мы реализовали шифрование Цезаря, сжатие изображения и работу факса. Для различных тестовых файлов мы подобрали ключи с помощью частотного анализа.

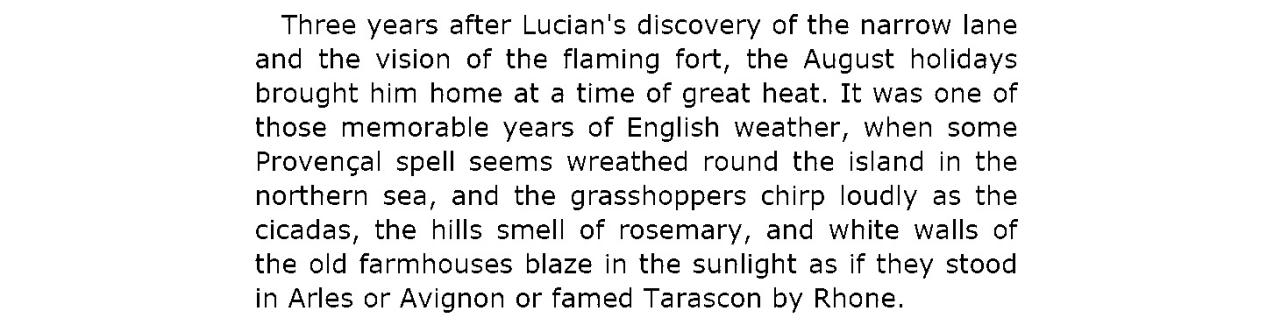
**Приложение**

Для таблицы 3 текста

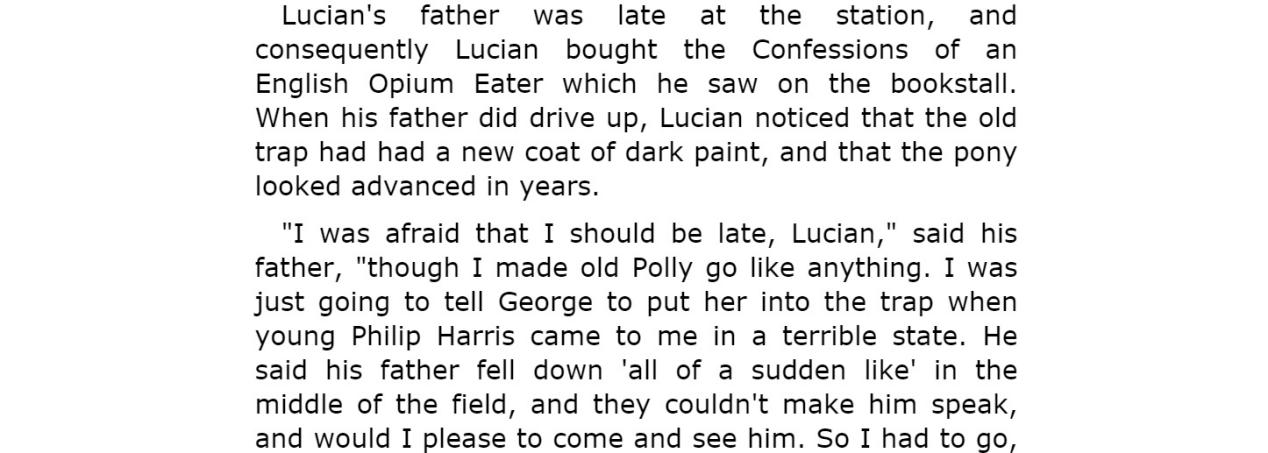
Изначальный Texted1.jpg



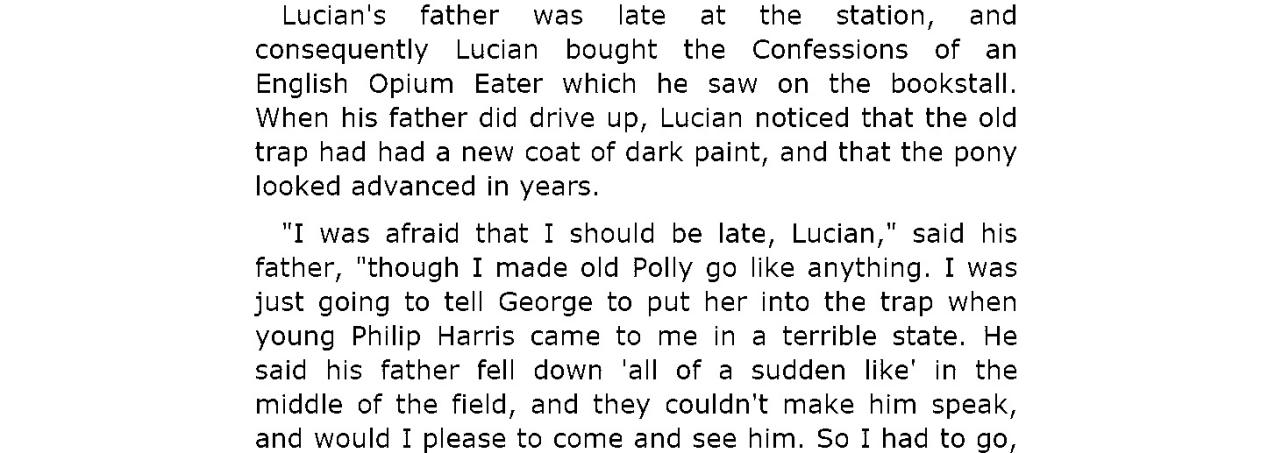
Расшифрованный



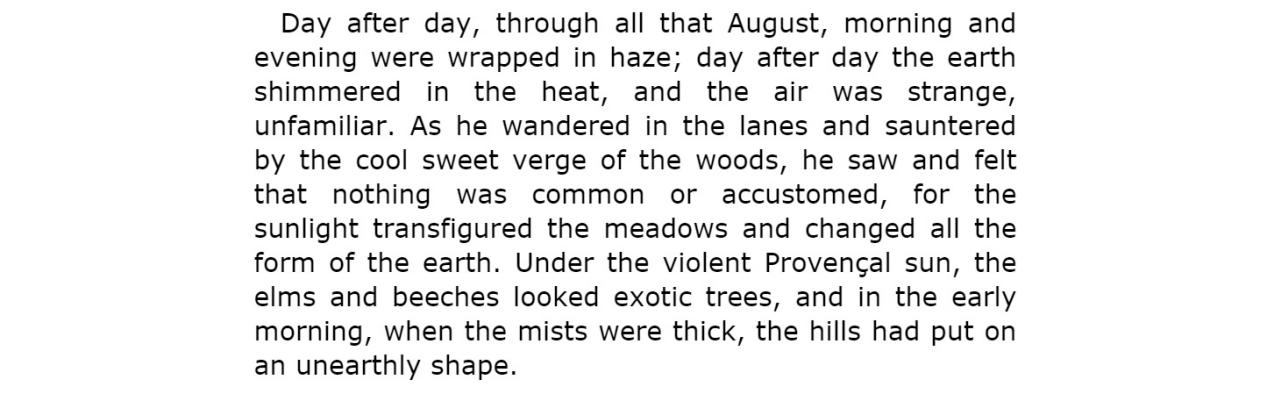
Изначальный Texted2.jpg



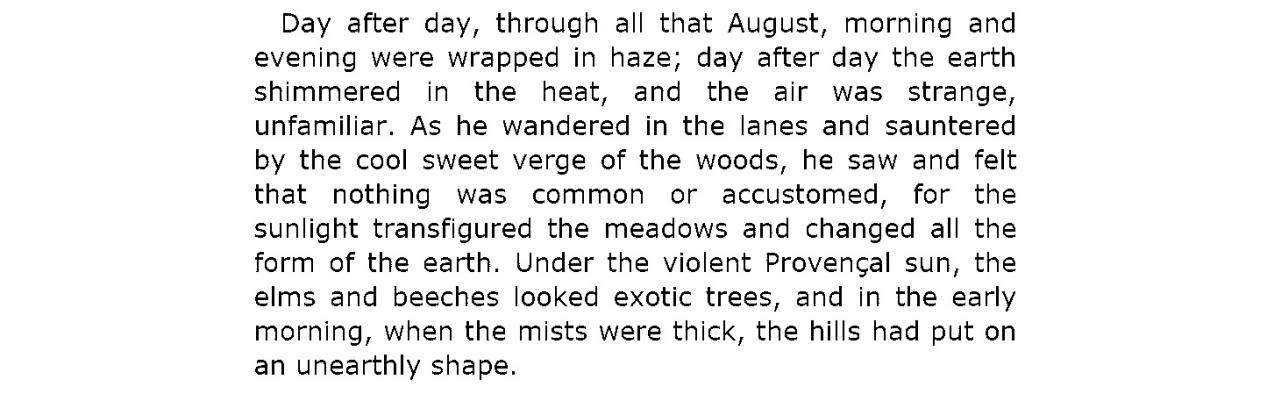
Расшифрованный



Изначальный Texted3.jpg



Расшифрованный



10 файлов для частотного анализа

Text1-text10

