Ковкель Никита, ФИТ 3-4

Информационная безопасность

Отчет по лабораторной работе № 5.

**Цель**: изучение и приобретение практических навыков разработки и использования приложений для реализации подстановочных шифров

**1. Практическая часть**

Маршрутная перестановка

Функция spiral\_encrypt шифрует открытый текст plaintext с помощью метода маршрутной перестановки, создается квадратная матрица размером spiral\_dimension x spiral\_dimension, где spiral\_dimension округляется вверх до ближайшего целого числа, чтобы обеспечить достаточное количество ячеек для всех символов. Исходный текст размещается в центре матрицы. Двигаемся по матрице в спиральном порядке, добавляя символы текста в ячейки. После заполнения матрицы выбираем символы с определенным шагом и формируем зашифрованный текст.

Функция spiral\_decrypt расшифровывает зашифрованный текст ciphertext с помощью метода маршрутной перестановки. Создается квадратная матрица размером spiral\_dimension x spiral\_dimension, где spiral\_dimension округляется вверх до ближайшего целого числа, чтобы обеспечить достаточное количество ячеек для всех символов. Зашифрованный текст размещается в матрице в спиральном порядке.Двигаемся по матрице в обратном спиральном порядке, извлекая символы и формируя исходный текст.

|  |
| --- |
| def spiral\_decrypt(cipher\_text, key):      plain\_text = []      spiral\_dimension = math.ceil(math.sqrt(len(cipher\_text) \* key))      grid = [[''] \* spiral\_dimension for \_ in range(spiral\_dimension)]      pos\_x = spiral\_dimension // 2      pos\_y = spiral\_dimension // 2      direction\_x = 0      direction\_y = -1        for character in cipher\_text:          grid[pos\_x][pos\_y] = character          pos\_x += direction\_x          pos\_y += direction\_y            if (pos\_x < 0 or pos\_x >= spiral\_dimension or pos\_y < 0 or pos\_y >= spiral\_dimension) or grid[pos\_x][pos\_y] != '':              pos\_x -= direction\_x              pos\_y -= direction\_y                direction\_x, direction\_y = -direction\_y, direction\_x                pos\_x += direction\_x              pos\_y += direction\_y        pos\_x = spiral\_dimension // 2      pos\_y = spiral\_dimension // 2      direction\_x = 0      direction\_y = -1        for \_ in range(len(cipher\_text) \* key):          if 0 <= pos\_x < spiral\_dimension and 0 <= pos\_y < spiral\_dimension and grid[pos\_x][pos\_y] != '':              plain\_text.append(grid[pos\_x][pos\_y])            pos\_x += direction\_x          pos\_y += direction\_y            if (pos\_x == pos\_y) or (pos\_x < key and pos\_x == -pos\_y) or (pos\_x > key and pos\_x == 1 - pos\_y):              direction\_x, direction\_y = -direction\_y, direction\_x        return ''.join(plain\_text) |

Листинг 1.1 – Функция для расшифрования

|  |
| --- |
| def spiral\_encrypt(plain\_text, key, step):      cipher\_text = []      spiral\_dimension = math.ceil(math.sqrt(len(plain\_text) \* key))      grid = [[''] \* spiral\_dimension for \_ in range(spiral\_dimension)]      pos\_x, pos\_y= spiral\_dimension // 2, spiral\_dimension // 2      direction\_x, direction\_y = 0, -1        for character in plain\_text:          char\_index = ALPHABET.find(character.lower())          if char\_index == -1:              continue            grid[pos\_x][pos\_y] = ALPHABET[char\_index]          pos\_x += direction\_x          pos\_y += direction\_y            if (pos\_x < 0 or pos\_x >= spiral\_dimension or pos\_y < 0 or pos\_y >= spiral\_dimension) or grid[pos\_x][pos\_y] != '':              pos\_x -= direction\_x              pos\_y -= direction\_y                direction\_x, direction\_y = -direction\_y, direction\_x                pos\_x += direction\_x              pos\_y += direction\_y        for i in range(0, spiral\_dimension, step):          for j in range(0, spiral\_dimension, step):              if i < spiral\_dimension and j < spiral\_dimension and grid[i][j] != '':                  cipher\_text.append(grid[i][j])        return ''.join(cipher\_text) |

Листинг 1.2 – Функция для расшифрования

Анализ гистограмм

Гистограммы зашифрованного текста и расшифрованного полностью совпадают и время выполнения кода.

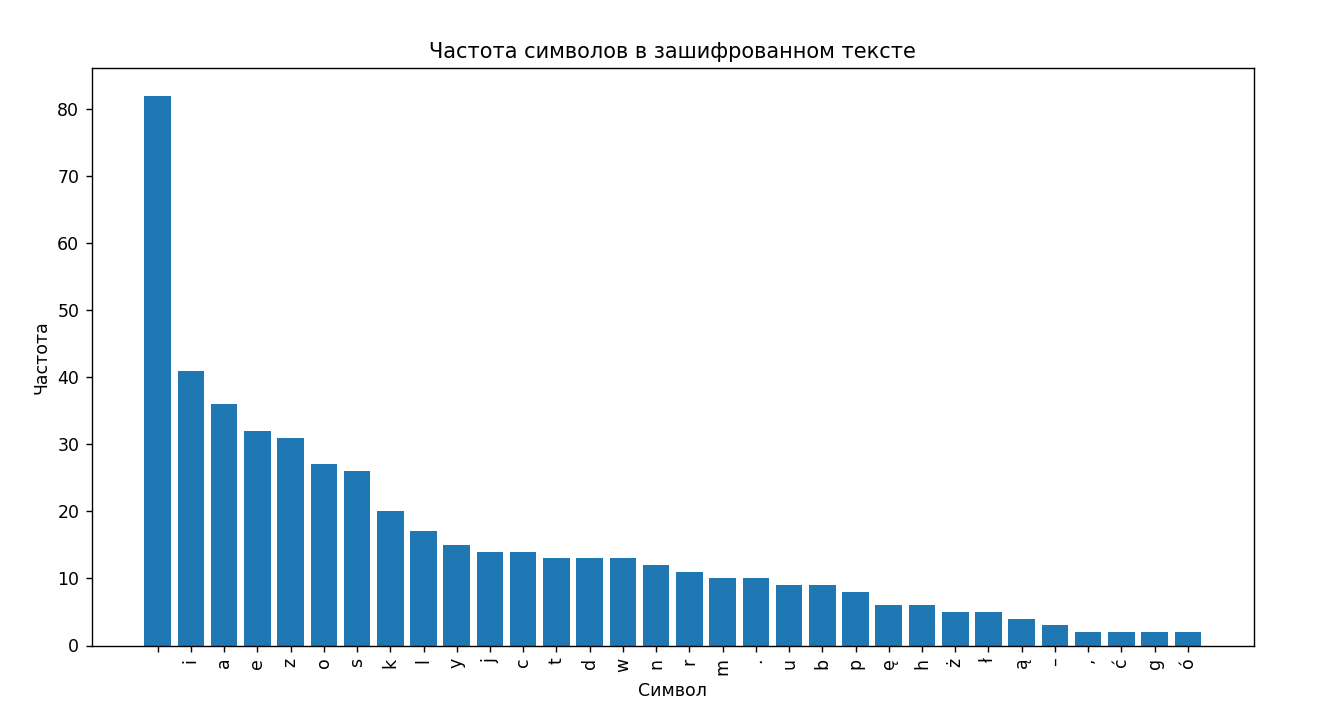


Рисунок 1.1 – Гистограмма зашифрованного текста

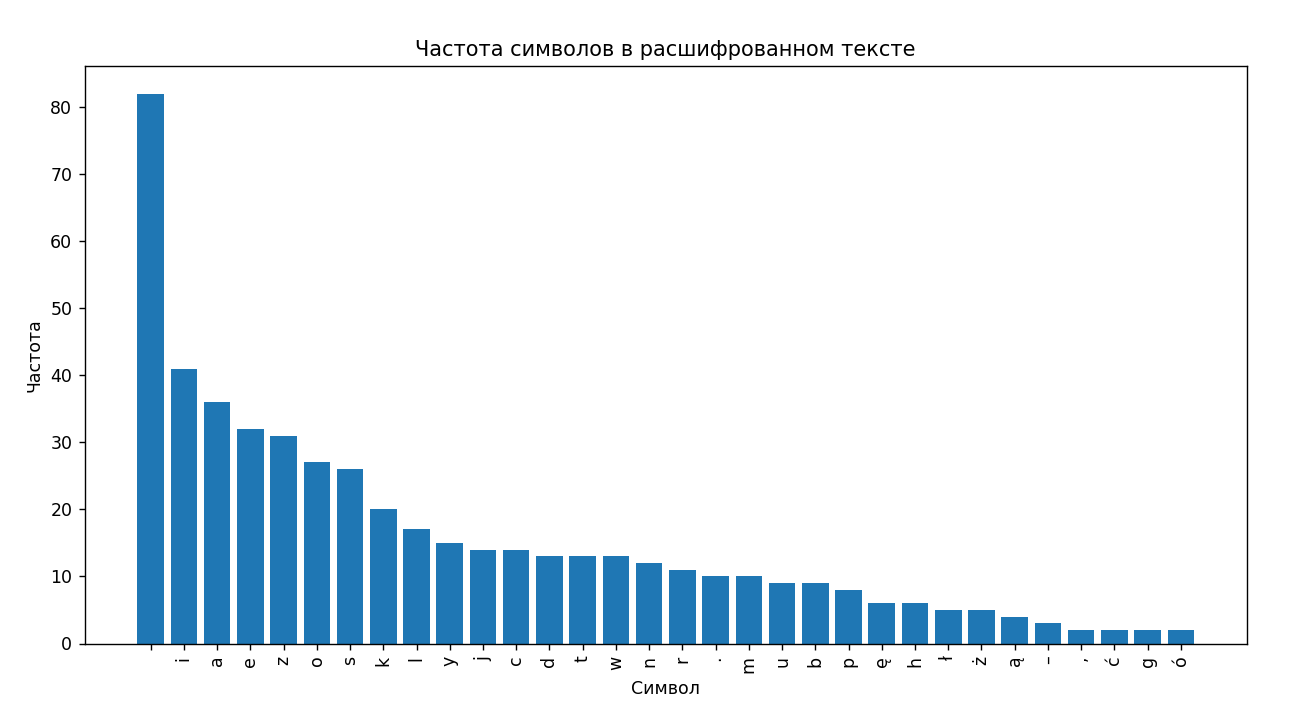


Рисунок 1.2 – Гистограмма расшифрованного текста

ъ

Рисунок 1.4 – Время выполнения кода

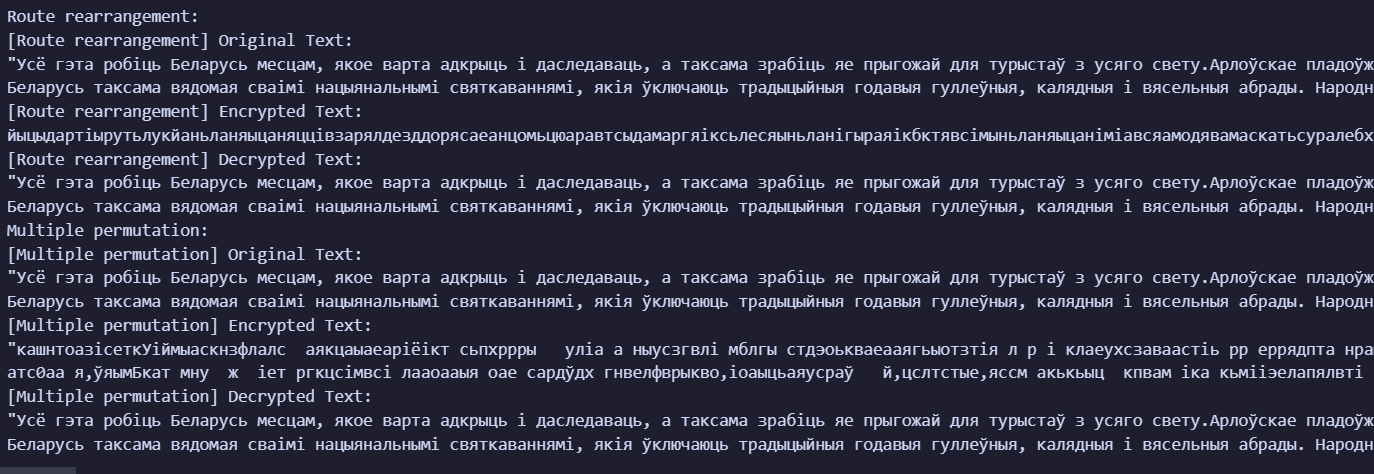


Рисунок 1.5 – Итог

1. Перестановочный шифр.

Сначала вызывается вспомогательная функция orderstring, которая возвращает порядок расположения букв в имени и фамилии. Затем определяется максимальный размер матрицы, в которую будет записываться открытый текст. Далее создается пустая матрица нужного размера, и в нее построчно записывается открытый текст, при этом позиции букв в матрице определяются порядком букв в имени и фамилии. После заполнения матрицы, построчно извлекаются символы из матрицы и объединяются в единую строку chipher\_text, которая и возвращается как результат шифрования.

|  |
| --- |
| def permute\_encrypt(plain\_text, key\_word):      cipher\_text = []      key\_length = len(key\_word)      permutation\_order = np.argsort(list(key\_word))        text\_length = len(plain\_text)      num\_of\_rows = math.ceil(text\_length / key\_length)      grid = np.zeros((num\_of\_rows, key\_length), dtype=str)        char\_index = 0      for column in permutation\_order:          for row in range(num\_of\_rows):              if char\_index < text\_length:                  grid[row][column] = plain\_text[char\_index]                  char\_index += 1        for row in range(num\_of\_rows):          for column in permutation\_order:              if grid[row][column] != '':                  cipher\_text.append(grid[row][column])        return ''.join(cipher\_text) |

Листинг 1.4 – Функция для шифрования

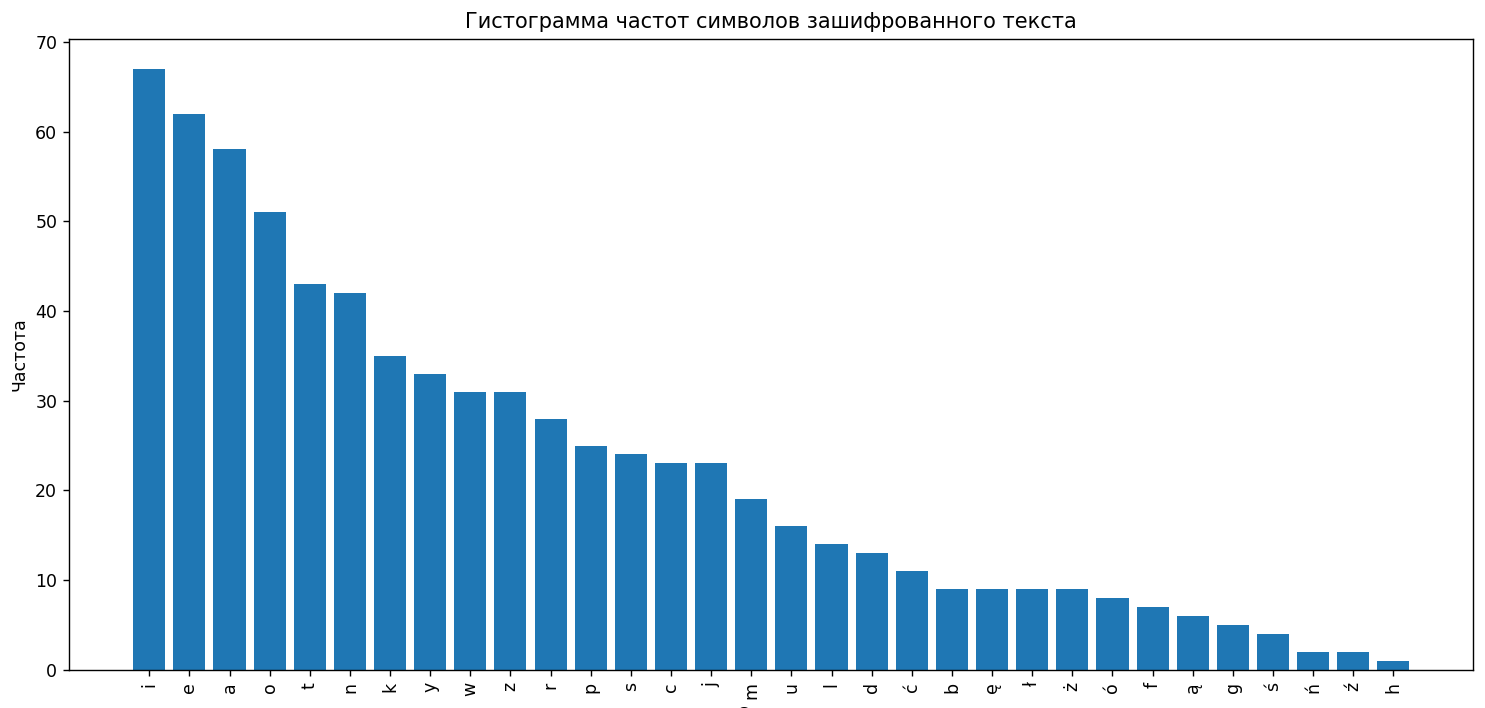
Аналогично шифрованию, Вычисляем длину ключевого слова (key\_length) Создаем порядок перестановки, сортируя символы ключевого слова (permutation\_order). Вычисляем количество строк в сетке (num\_of\_rows), исходя из длины зашифрованного текста и длины ключевого слова. Создаем сетку размером (num\_of\_rows, key\_length) и заполняем ее нулями (grid). Итерируем по порядку перестановки, для каждого столбца в сетке. Добавляем символы из зашифрованного текста в соответствующие ячейки сетки. Увеличиваем индекс символа (char\_index). Итерируем по столбцам и строкам сетки. Если ячейка не пуста (содержит символ). Добавляем символ в расшифрованный текст (plain\_text). Возвращаем объединенный расшифрованный текст.

|  |
| --- |
| def permute\_decrypt(cipher\_text, key\_word):      plain\_text = []      key\_length = len(key\_word)      permutation\_order = np.argsort(list(key\_word))        cipher\_text\_length = len(cipher\_text)      num\_of\_rows = math.ceil(cipher\_text\_length / key\_length)      grid = np.zeros((num\_of\_rows, key\_length), dtype=str)        char\_index = 0      for column in permutation\_order:          for row in range(num\_of\_rows):              if char\_index < cipher\_text\_length:                  grid[row][column] = cipher\_text[char\_index]                  char\_index += 1        for column in range(key\_length):          for row in range(num\_of\_rows):              if grid[row][column] != '':                  plain\_text.append(grid[row][column])        return ''.join(plain\_text) |

Листинг 1.5 – Функция для расшифрования

Анализ гистограмм

Гистограммы зашифрованного текста и расшифрованного полностью совпадают.



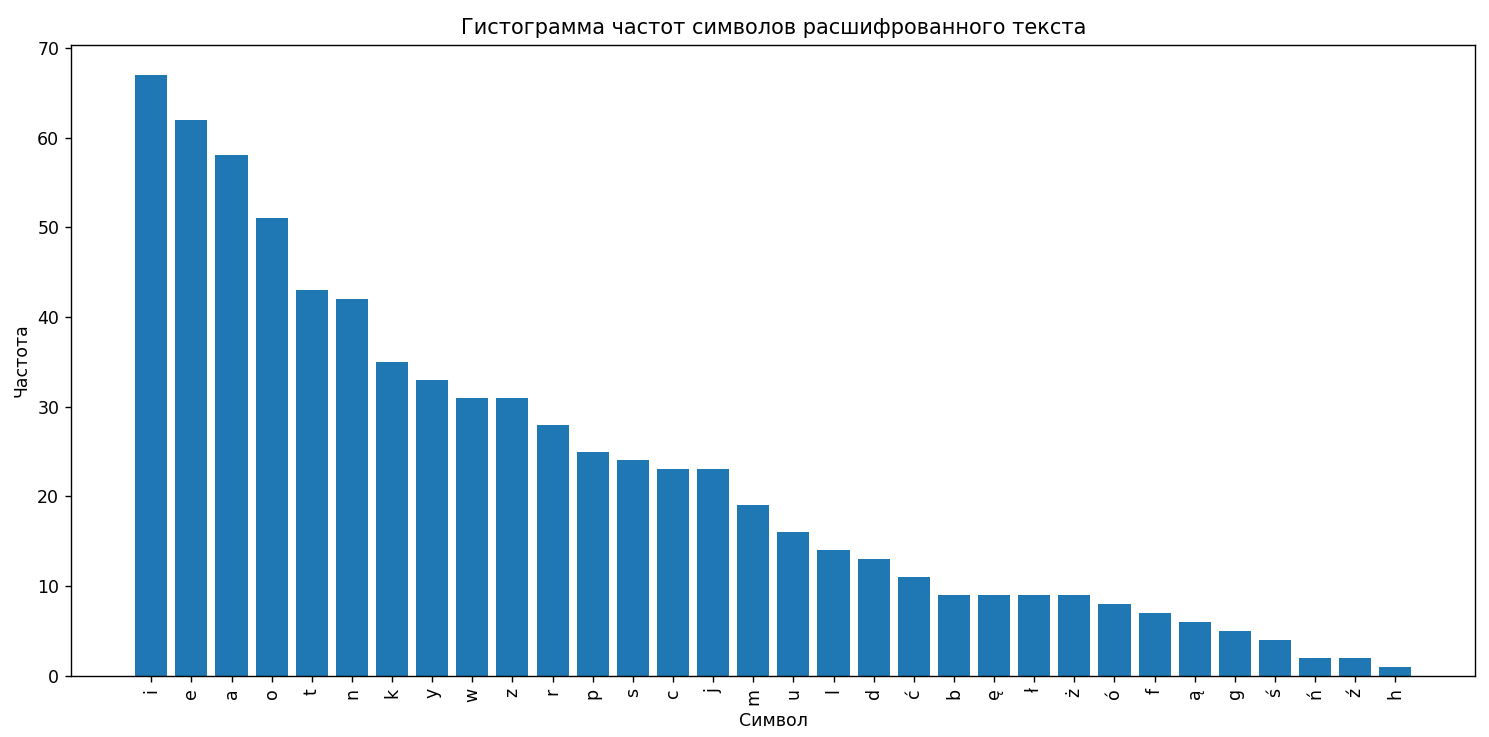


Рисунок 2.2 – Гистограммы зашифрованного и расшифрованного текстов

Вывод: на данной лабораторной работе мы изучили и приобрели практические навыки разработки и использования приложений для реализации перестановочных шифров (работа рассчитана на 4 часа аудиторных занятий).