Шифры престановки

Федюшина Ярослава Андреевна

Содержание

Список иллюстраций

Цель работы

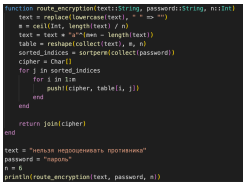
Целью данной лабораторной работы является изучение алгоритмов шифрования перестановки, принцип работы и реализация на языке Julia.

Задание

Реализовать все шифры программно.

Выполнение лабораторной работы

#Маршрутное шифрование



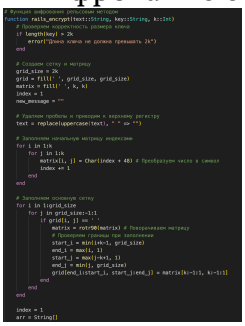
```
function route_encryption(text::String, password::String, n::Int)
    text = replace(lowercase(text), " " => "")
    m = ceil(Int, length(text) / n)
    text = text * "а"^(mn - length(text))
    table = reshape(collect(text), m, n)
    sorted_indices = sortperm(collect(password))
    cipher = Char[]
    for j in sorted_indices
        for i in 1:m
            push!(cipher, table[i, j])
        end
    end

    return join(cipher)
end

text = "нельзя недооценивать противника"
password = "пароль"
n = 6
println(route_encryption(text, password, n))
```

Рис. 1: Маршрут

#Шифрование с помощью решеток



```
# Функция шифрования с помощью решетки
function rails_encrypt(text::String, key::String, k::Int)
    # Проверяем корректность размера ключа
    if length(key) > 2k
        error("Длина ключа не должна превышать 2k")
    end

    # Создаем сетку и матрицу
    grid_size = 2k
    grid = fill(' ', grid_size, grid_size)
    matrix = fill(' ', k, k)
    index = 1
    new_message = ""

    # Удаляем пробелы и приводим к верхнему регистру
    text = replace(uppercase(text), " " => "")

    # Заполняем начальную матрицу индексами
    for i in 1:k
        for j in 1:k
            matrix[i, j] = Char(index + 48) # Преобразуем число в символ
            index += 1
        end
    end

    # Заполняем основную сетку
    for i in 1:grid_size
        for j in grid_size-1:1
            if grid[i, j] == ' '
                matrix = rotr90(matrix) # Поворачиваем матрицу
                # Проверяем границы при заполнении
                start_i = min(i+k-1, grid_size)
                end_i = max(i, 1)
                start_j = max(j-k+1, 1)
                end_j = min(j, grid_size)
                grid[end_i:start_i, start_j:end_j] = matrix[k-(i-1):k, k-(j-1):1]
            end
        end
    end

    index = 1
    arr = String[]
```

```
    # Вычисляем текст в сетке
    for r in text
        checker = false
        for i in 1:grid_size
            for j in 1:grid_size
                if grid[i, j] == Char(index + 48) && !checker
                    if !(string(i+1, j) in arr) && !(string(i-1, j) in arr) &&
                       !(string(i, j+1) in arr) && !(string(i, j-1) in arr)
                        grid[i, j] = r
                        push!(arr, string(i, j))
                        checker = true
                    end
                end
            end
        end
        if checker
            index += 1
            if index > k^2
                index = 1
                empty!(arr)
            end
            break
        end
    end
end

# Считываем зашифрованный текст по ключу
for j in sort(collect(key))
    for i in 1:grid_size
        # Проверяем существование позиции
        if find(first(j), key) = grid_size
            char = grid[i, find(first(j), key)]
            if isdigit(char)
                char = ' ' # Заменяем цифры на пробелы
            end
            new_message *= char
        end
    end
end

return new_message
end
```

#Таблица Виженера

```julia
# Функция для генерации ключа нужной длины
function generate_key(plaintext::String, key::String)
    key_length = length(key)
    plain_length = length(plaintext)

    new_key = ""
    key_chars = collect(key)  # Преобразуем ключ в массив символов
    for i in 1:plain_length
        new_key += key_chars[mod1(i, key_length)]
    end
    return new_key
end

# Функция шифрования
function vigenere_encrypt(plaintext::String, key::String)
    # Алфавит русский (только заглавные буквы)
    alphabet = ['А','Б','В','Г','Д','Е','Ё','Ж','З','И','Й','К','Л','М','Н','О','П',
                'Р','С','Т','У','Ф','Х','Ц','Ч','Ш','Щ','Ъ','Ы','Ь','Э','Ю','Я']

    plaintext = uppercase(plaintext)
    key = uppercase(generate_key(plaintext, key))

    ciphertext = ""
    for (p, k) in zip(plaintext, key)
        if isletter(p)
            # Находим позицию буквы в алфавите
            p_pos = findfirst(x -> x == p, alphabet)
            k_pos = findfirst(x -> x == k, alphabet)

            if p_pos !== nothing && k_pos !== nothing
                shift = (p_pos + k_pos - 2) % 33  # Корректируем индексацию
                ciphertext += alphabet[shift + 1]  # Возвращаем в диапазон 1-33
            else
                ciphertext += p
            end
        else
            ciphertext += p
        end
    end
    return ciphertext
end
```

```julia
# Функция дешифрования
function vigenere_decrypt(ciphertext::String, key::String)
    alphabet = ['А','Б','В','Г','Д','Е','Ё','Ж','З','И','Й','К','Л','М','Н','О','П',
                'Р','С','Т','У','Ф','Х','Ц','Ч','Ш','Щ','Ъ','Ы','Ь','Э','Ю','Я']

    ciphertext = uppercase(ciphertext)
    key = uppercase(generate_key(ciphertext, key))

    plaintext = ""
    for (c, k) in zip(ciphertext, key)
        if isletter(c)
            c_pos = findfirst(x -> x == c, alphabet)
            k_pos = findfirst(x -> x == k, alphabet)

            if c_pos !== nothing && k_pos !== nothing
                shift = (c_pos - k_pos + 33) % 33
                plaintext += alphabet[shift + 1]
            else
                plaintext += c
            end
        else
            plaintext += c
        end
    end
    return plaintext
end

# Пример использования
plaintext = "криптография серьезная наука"
key = "математика"

# Шифрование
encrypted = vigenere_encrypt(plaintext, key)
println("Зашифрованный текст: $encrypted")

# Дешифрование
decrypted = vigenere_decrypt(encrypted, key)
println("Расшифрованный текст: $decrypted")
```

Вывод

В ходе выполнения данной лабораторной работы были изучены три шифра перестановки, а так же были реализованы на языке Julia.