МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет»

Кафедра инфокоммуникаций

Отчет по лабораторной работе №7 по дисциплине «Алгоритмизация»

Выполнил студент группы ИВТ-б-о-22-1
Пушкин Н.С. « »_____20__г.
Подпись студента_____
Работа защищена « »_____20__г.
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2023

Порядок выполнения работы:

Реализация алгоритма Хаффмана:

```
C:\Users\Никита\Desktop\alg_lab_7\venv\Scripts\python.exe C:\Users\Никита\Desktop\alg_lab_7\main.py
результат : bitarray('11110110101110100111110001000011001000101011001')
декодирование: Nikita Pushkin

Process finished with exit code 0
```

Запуск программы

Предложение которое ввводит пользователь кодируется по спецальному алгоритму Хаффмана выдавая выходное значение 01.. строку в которой закодированы все буквы.

Код:

```
from queue import PriorityQueue
import bitarray


class Node:
    def __init__(self, symbol, frequency):
        self.symbol = symbol
        self.frequency = frequency
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.frequency < other.frequency

    def __eq__(self, other):
        if other is None or not isinstance(other, Node):
            return False
        return self.frequency == other.frequency


def huffman_tree(source):
    frequencies = {}
    for char in source:
        if char not in frequencies:
            frequencies[char] = 0
        frequencies[char] += 1

    priority_queue = PriorityQueue()
    for key, value in frequencies.items():
        priority_queue.put((value, key, Node(key, value)))

    while priority_queue.qsize() > 1:
        freq1, _, left = priority_queue.get()
        freq2, _, right = priority_queue.get()
        merged_node = Node('*', freq1 + freq2)
        merged_node.left = left
        merged_node.right = right
        priority_queue.put((freq1 + freq2, '', merged_node))

    return priority_queue.get()


def encode(source, root):
    encoded_source = bitarray.bitarray()
    for char in source:
```

```
        encoded_symbol = traverse_tree(root, char, [])
        encoded_source.extend(encoded_symbol)
    return encoded_source


def traverse_tree(current, char, encoding):
    if current is not None:
        if current.symbol == char:
            return encoding
        else:
            left_result = traverse_tree(current.left, char, encoding +
[False])
            if left_result:
                return left_result
            right_result = traverse_tree(current.right, char, encoding +
[True])
            return right_result


def decode(bits, root):
    decoded = ''
    current = root
    for bit in bits:
        if bit:
            current = current.right
        else:
            current = current.left

        if current.symbol != '*':
            decoded += current.symbol
            current = root
    return decoded


source = "Nikita Pushkin"
frequency, _, root = huffman_tree(source)
encoded = encode(source, root)
decoded = decode(encoded, root)
print("результат :", encoded)
print("декодирование:", decoded)
```