Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

**ОТЧЕТ**
**ПО ПРАКТИЧЕСКОЙ РАБОТЕ №7**
**дисциплины «Алгоритмизация»**

Выполнил:
Говоров Егор Юрьевич
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизирование систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А., канд. техн. наук, доцент, доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой_____Дата защиты_____

Ставрополь, 2023 г.

Тема: Алгоритм Хаффмана

Цель: Изучение, реализация и анализ эффективности алгоритма Хаффмана в контексте сжатия данных. Целью работы является освоение основных принципов построения кодов Хаффмана, их применение для сжатия информации, а также проведение экспериментов для оценки степени сжатия и времени выполнения алгоритма на различных типах данных

Порядок выполнения работы:

1. Написал алгоритм вычисления частоты символов, которые встречаются в тексте

```
char_count = {}
for char in sentence:
    char_count[char] = char_count.get(char, 0) + 1
return char_count
```

```
char_frequencies = count_characters(user_input)
print("Результат подсчета вхождений каждого символа:")
for char, count in char_frequencies.items():
    print(f"Символ '{char}': {count} раз")
```

Рисунок 1 – Фрагмент кода, в котором считается частота символа

2. Построение дерева в соответствии с процедурой хаффмана

```
def build_huffman_tree(frequencies):
    heap = []
    buffer_fs = set()

    for key in frequencies:
        heapq.heappush(heap, (frequencies[key], key))

    while len(heap) > 1:
        f1, i = heapq.heappop(heap)
        f2, j = heapq.heappop(heap)
        fs = f1 + f2
        ord_val = ord('a')
        fl = str(fs)

        while fl in buffer_fs:
            letter = chr(ord_val)
            fl = str(fs) + " " + letter
            ord_val += 1

        buffer_fs.add(fl)
        frequencies[fl] = {f"{x}": frequencies[x] for x in [i, j]}
        del frequencies[i], frequencies[j]
        heapq.heappush(heap, (fs, fl))

    return frequencies
```

```
huffman_tree = build_huffman_tree(char_frequencies)
print("\nДерево Хаффмана:")
print(huffman_tree)
```

Рисунок 2 – Функция построения дерева

3. Кодирование

```
def text_to_binary(text):
    binary_result = ' '.join(format(ord(char), '08b') for char in text)
    return binary_result
```

```
binary_text = text_to_binary(user_input)
print("\nБинарное представление текста:")
print(binary_text)
```

Рисунок 3 – Функция кодирования символов

4. Декодирование

```
def binary_to_text(binary_text):
    binary_values = binary_text.split()
    text_result = ''.join(chr(int(value, 2)) for value in binary_values)
    return text_result
```

```
decoded_text = binary_to_text(binary_text)
print("\nРаскодированный текст:")
print(decoded_text)
```

Рисунок 4 – Функция декодирования символов

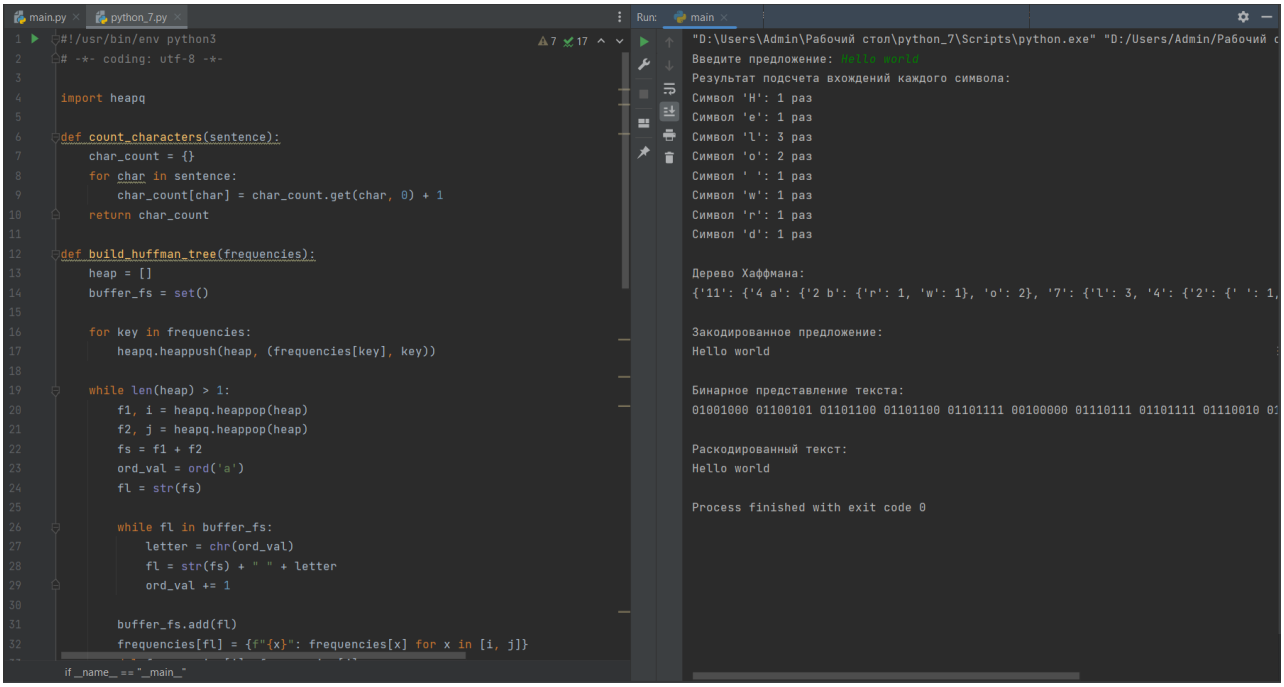


Рисунок 5 – Результат выполнения программы python_7.py

Вывод: В ходе выполнения данной практической работы был изучен и реализован алгоритм Хаффмана, используемый для сжатия данных. Эксперименты с различными типами данных позволили оценить эффективность алгоритма, выявив его способность достигать заметного уровня сжатия при минимальном времени выполнения. Результаты практической работы подтверждают применимость алгоритма Хаффмана в контексте оптимизации хранения и передачи данных