

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

```
main.py x python_7.py x  
1 #!/usr/bin/env python3  
2 # -*- coding: utf-8 -*-  
3  
4 import heapq  
5  
6 def count_characters(sentence):  
7     char_count = {}  
8     for char in sentence:  
9         char_count[char] = char_count.get(char, 0) + 1  
10    return char_count  
11  
12 def build_huffman_tree(frequencies):  
13     heap = []  
14     buffer_fs = set()  
15  
16     for key in frequencies:  
17         heapq.heappush(heap, (frequencies[key], key))  
18  
19     while len(heap) > 1:  
20         f1, i = heapq.heappop(heap)  
21         f2, j = heapq.heappop(heap)  
22         fs = f1 + f2  
23         ord_val = ord('a')  
24         fl = str(fs)  
25  
26         while fl in buffer_fs:  
27             letter = chr(ord_val)  
28             fl = str(fs) + " " + letter  
29             ord_val += 1  
30  
31         buffer_fs.add(fl)  
32         frequencies[fl] = {f"{x}": frequencies[x] for x in [i, j]}  
33  
34 if __name__ == "__main__":  
35     sentence = "Hello world"  
36     char_count = count_characters(sentence)  
37     frequencies = char_count  
38     huffman_tree = build_huffman_tree(frequencies)  
39     encoded_text = encode_text(sentence, huffman_tree)  
40     decoded_text = decode_text(encoded_text, huffman_tree)  
41     print("Введите предложение: ", sentence)  
42     print("Результат подсчета вхождений каждого символа:")  
43     for char, count in char_count.items():  
44         print(f"Символ '{char}': {count} раз")  
45     print("Дерево Хаффмана:")  
46     print(huffman_tree)  
47     print("Закодированное предложение:")  
48     print(encoded_text)  
49     print("Бинарное представление текста:")  
50     print(encoded_text)  
51     print("Раскодированный текст:")  
52     print(decoded_text)  
53     print("Process finished with exit code 0")
```

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