

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 ЛАБОРАТОРНОЙ РАБОТЕ №7
дисциплины «Алгоритмизация»

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

(подпись)

Руководитель практики:
Воронкин Р.А., доцент кафедры
инфокоммуникаций

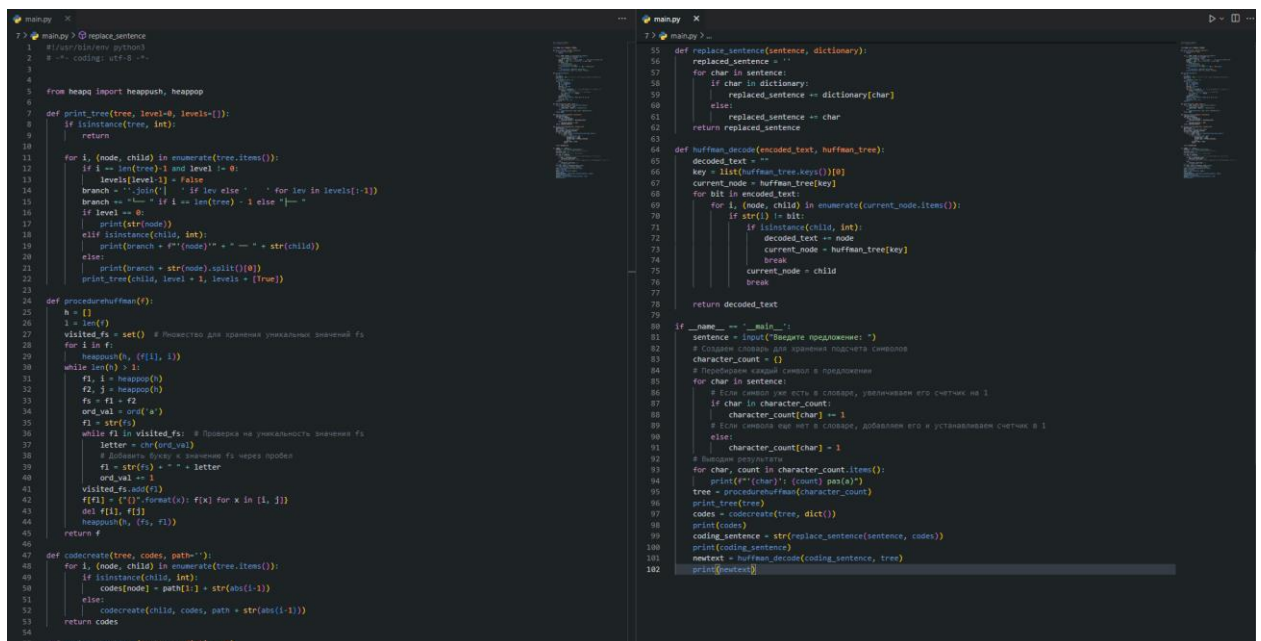
(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Порядок выполнения работы:

1. Написал программу для кодирования и декодирования текста при помощи кодировки Хаффмана. В коде есть следующие функции: `print tree()` отрисовывает дерево в терминале; `procedurehuffman()` создает дерево хаффмана на основе словаря, в котором каждому символу присвоена его частота использования в предложении; `codecreate()` создает каждому символу определенный код возвращает созданное в виде словаря; `replace sentence` заменяет в предложении каждый символ на ранее созданные коды; `huffman decode()` с помощью дерева Хаффмана декодирует последовательность 0 и 1:



```
1 # -*- coding: utf-8 -*-
2
3
4
5 from heapq import heappush, heappop
6
7 def print_tree(tree, level=0, levels=[]):
8     if isinstance(tree, int):
9         return
10
11     for i, (node, child) in enumerate(tree.items()):
12         if i == len(tree)-1 and level != 0:
13             levels[level-1] = False
14             branch = " " * (level-1)
15             if i == len(levels)-1:
16                 branch = " " * (level-1)
17             if level == 0:
18                 print(str(node))
19             elif isinstance(child, int):
20                 print(branch + str(node).split()[0])
21             else:
22                 print(branch + str(node).split()[0])
23                 print_tree(child, level+1, levels+[True])
24
25 def procedurehuffman(f):
26     n = []
27     i = len(f)
28     visited_fs = set() # Понесет для хранения уникальных значений fs
29     for i in f:
30         heappush(n, (f[i], i))
31         while len(n) > 1:
32             f1, i = heappop(n)
33             f2, j = heappop(n)
34             f = f1 + f2
35             ord_val = ord('a')
36             f1 = str(f1)
37             while f2 in visited_fs: # Проверка на уникальность значения fs
38                 letter = chr(ord_val)
39                 f1 = str(f1) + " " + letter
40                 ord_val += 1
41             visited_fs.add(f1)
42             f[f1] = f1
43             heappush(n, (f1, i))
44         return f
45
46 def codecreate(tree, codes, path=""):
47     for i, (node, child) in enumerate(tree.items()):
48         if isinstance(child, int):
49             codes[node] = path[i] + str(abs(i-1))
50         else:
51             codecreate(child, codes, path + str(abs(i-1)))
52     return codes
53
54 def replace_sentence(sentence, dictionary):
55     replaced_sentence = ""
56     for char in sentence:
57         if char in dictionary:
58             replaced_sentence += dictionary[char]
59         else:
60             replaced_sentence += char
61     return replaced_sentence
62
63 def huffman_decode(encoded_text, huffman_tree):
64     decoded_text = ""
65     key = list(huffman_tree.keys())[0]
66     current_node = huffman_tree[key]
67     for bit in encoded_text:
68         for i, (node, child) in enumerate(current_node.items()):
69             if str(i) != bit:
70                 if isinstance(child, int):
71                     decoded_text += node
72                     current_node = huffman_tree[key]
73                     break
74                 current_node = child
75                 break
76     return decoded_text
77
78 if __name__ == "__main__":
79     sentence = input("Введите предложение: ")
80     # Создание словаря для хранения подсчета символов
81     character_count = {}
82     for char in sentence:
83         # Если символ уже есть в словаре, увеличиваем его счетчик на 1
84         if char in character_count:
85             character_count[char] += 1
86         # Если символа нет в словаре, добавляем его и устанавливаем счетчик в 1
87         else:
88             character_count[char] = 1
89     # Выводим статистику
90     for char, count in character_count.items():
91         print(f"{char}: {count} раз")
92     tree = procedurehuffman(character_count)
93     print_tree(tree)
94     codes = codecreate(tree, dict())
95     print(codes)
96     coding_sentence = str(replace_sentence(sentence, codes))
97     print(coding_sentence)
98     new_text = huffman_decode(coding_sentence, tree)
99     print(new_text)
```

Рисунок 1. Код программы

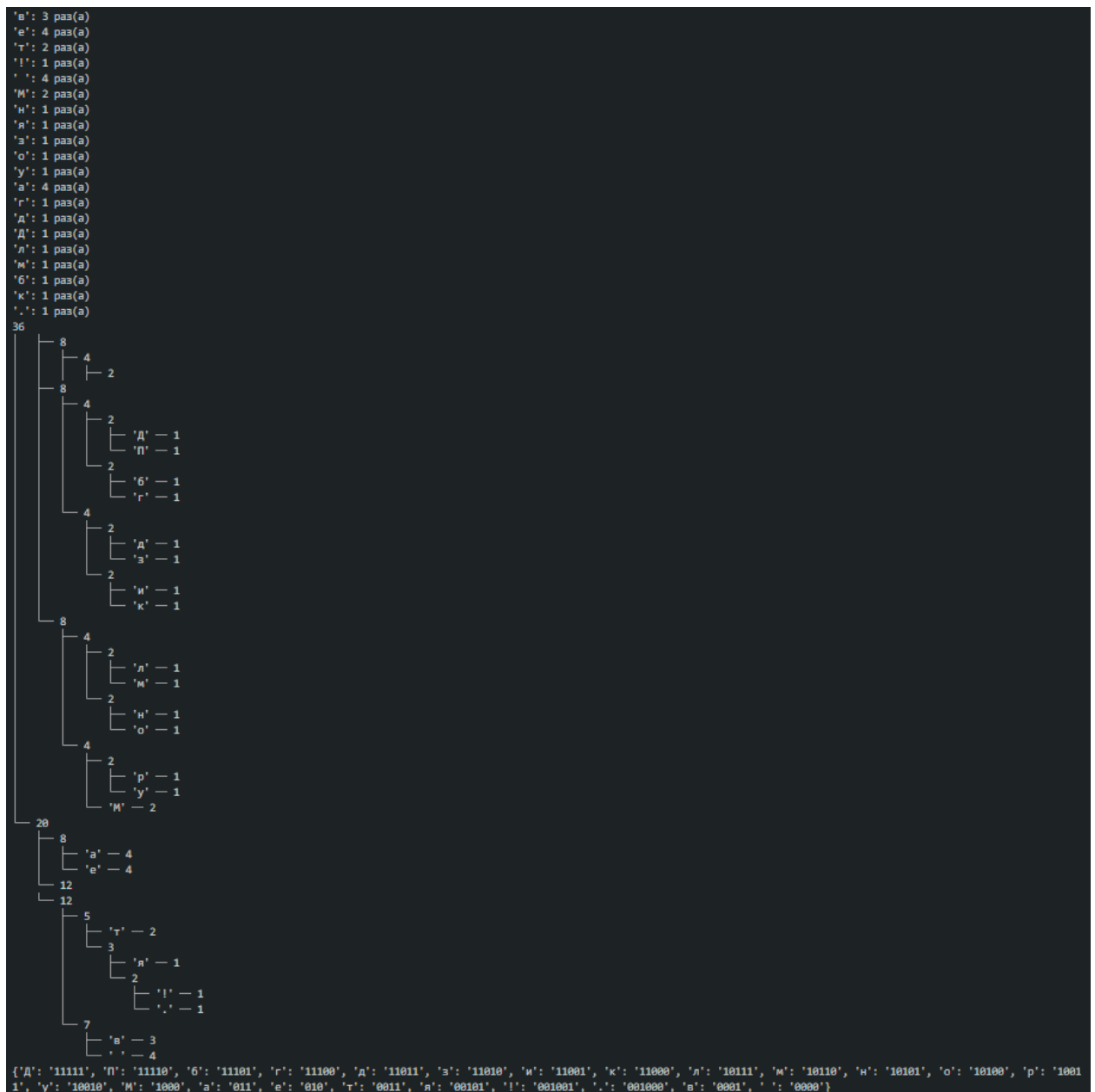


Рисунок 2. Результат выполнения программы

В процессе выполнения лабораторной работы был изучен алгоритм кодирования Хаффмана, включая создание дерева Хаффмана и кодирование и декодирование текста с его помощью. Из этого следует, что метод кодирования Хаффмана представляет собой эффективный способ сжатия данных, особенно текстовых, в которых некоторые символы встречаются чаще, чем другие.