

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:
Кенесбаев Хилол Куат улы
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

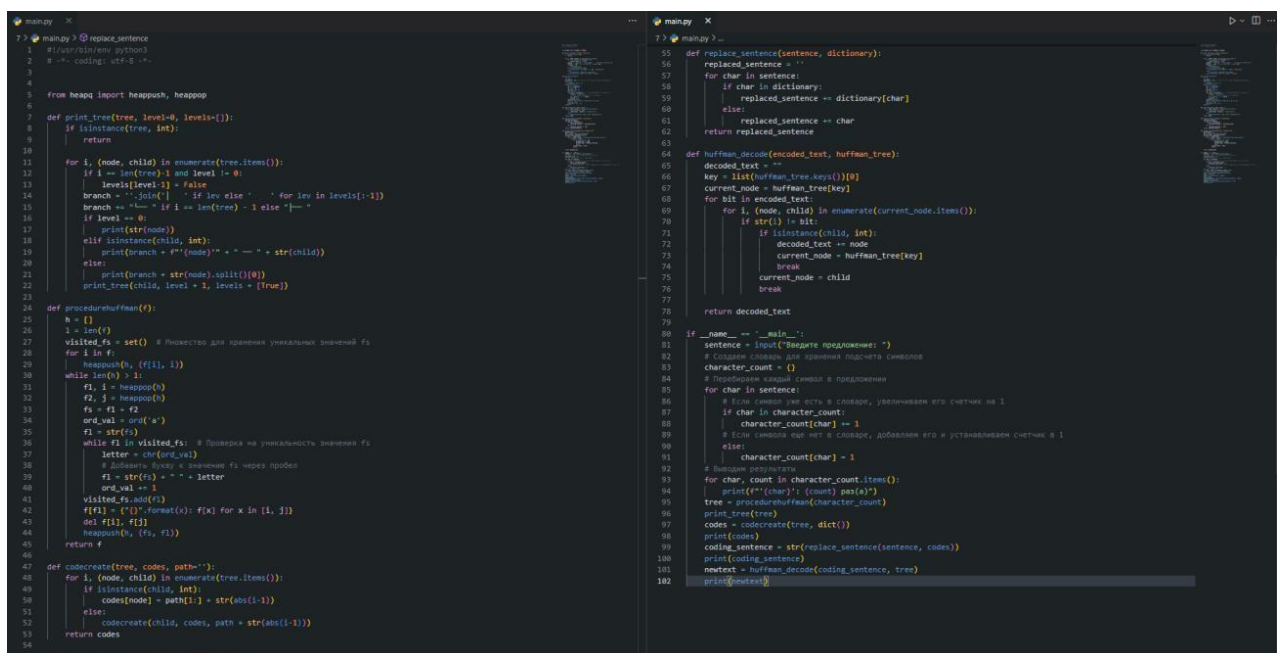
(подпись)

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

Рисунок 1 Код программы

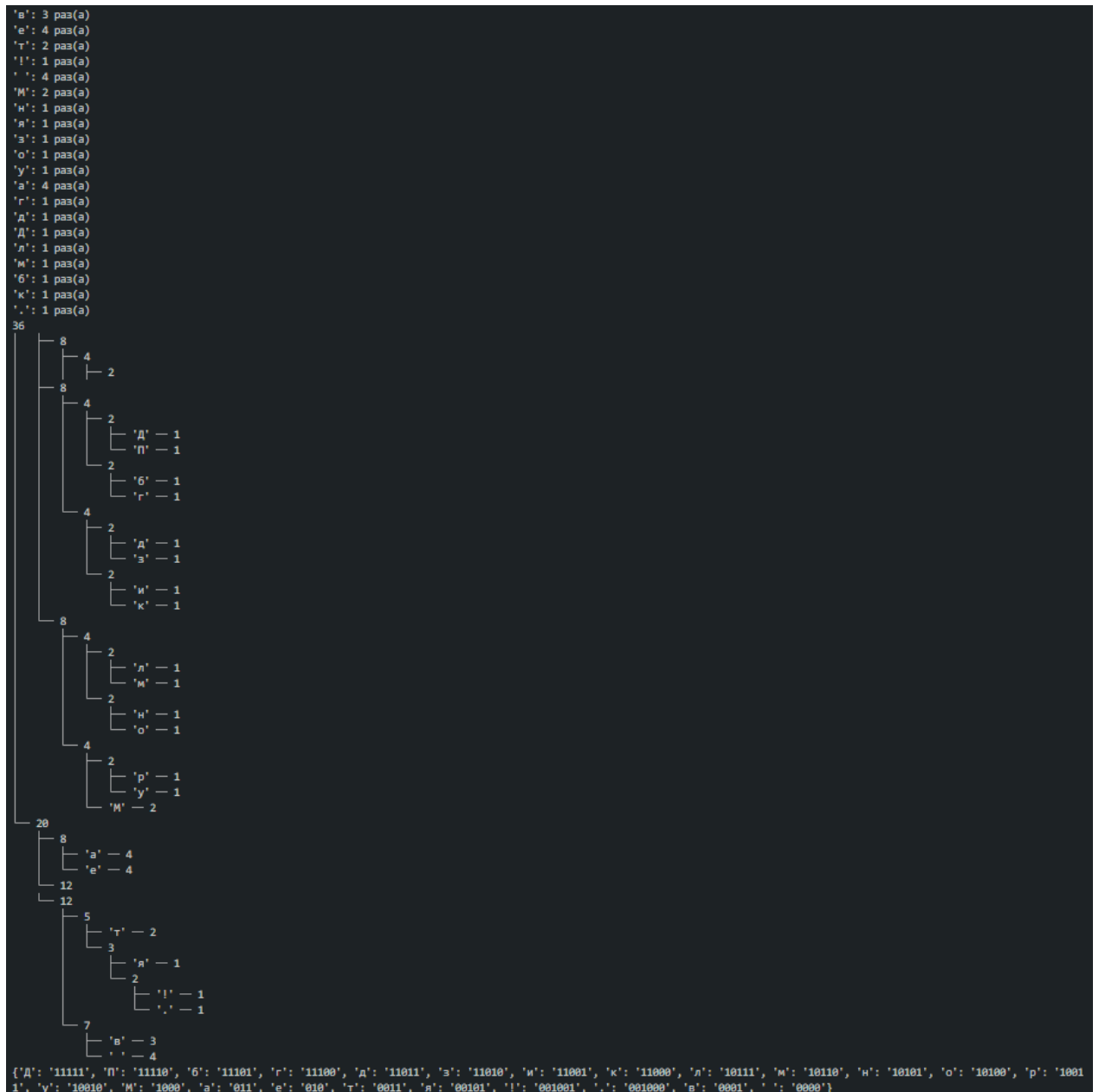


Рисунок 2 Результат выполнения программы

Вывод: 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.