

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:
Дзуев Альберт Мухаметович
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

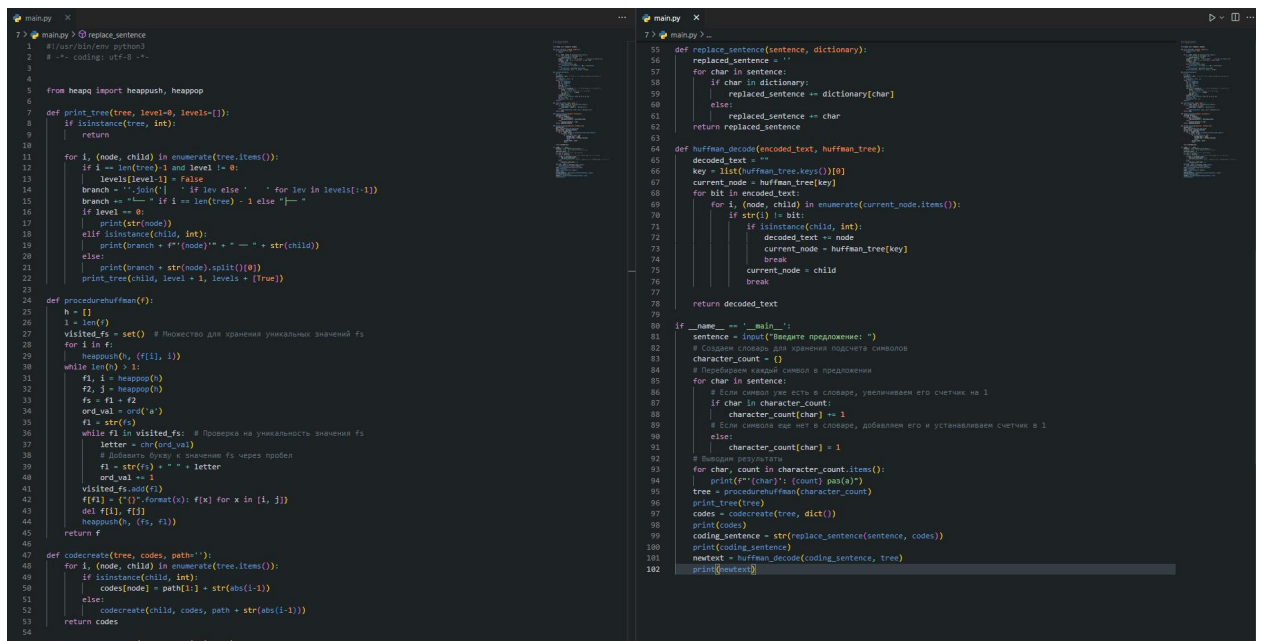
(подпись)

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:



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

Рисунок 1. Код программы

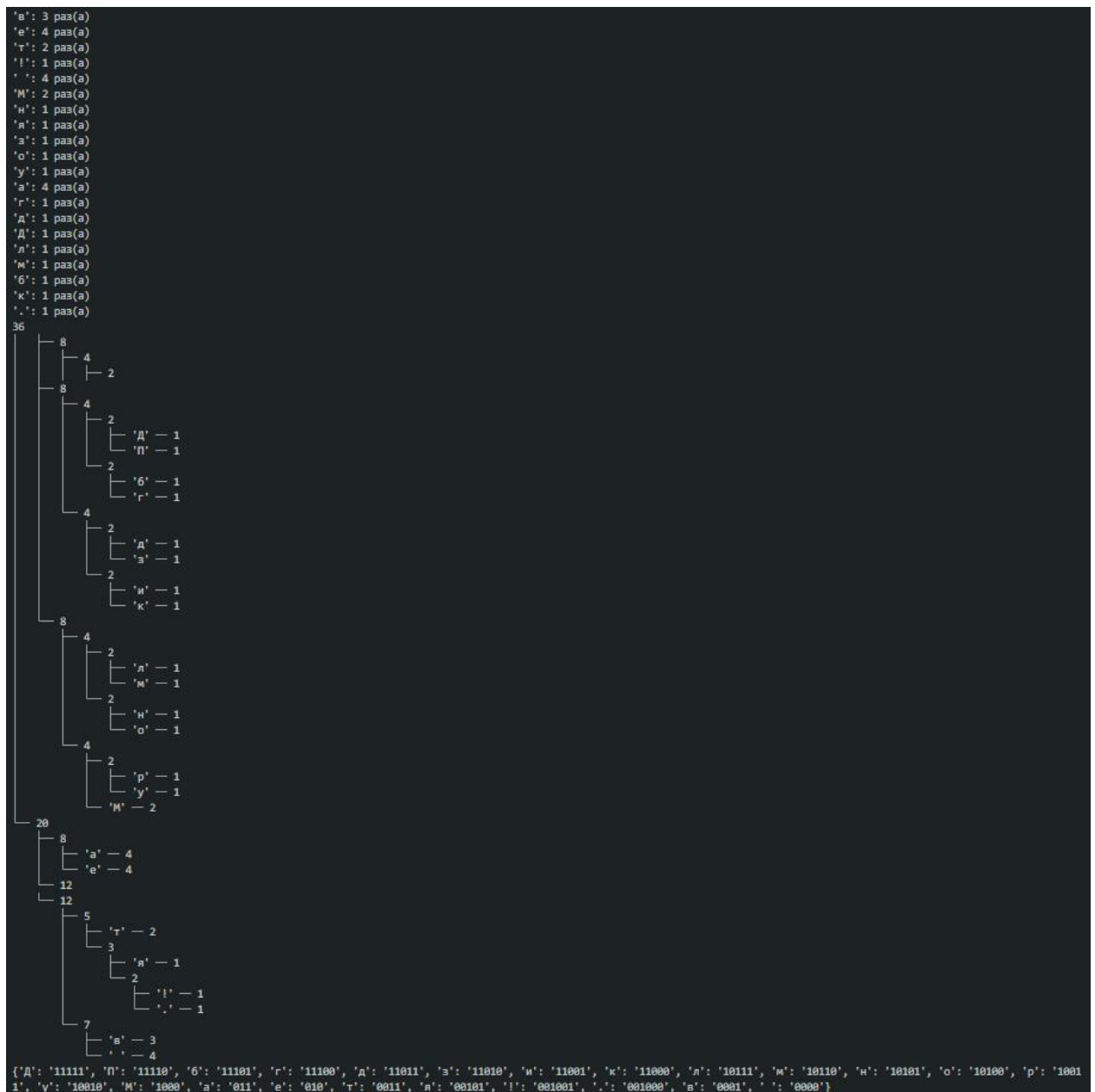


Рисунок 2. Результат выполнения программы

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.