

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:
Ибрагимов Муса Айнуудинович
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

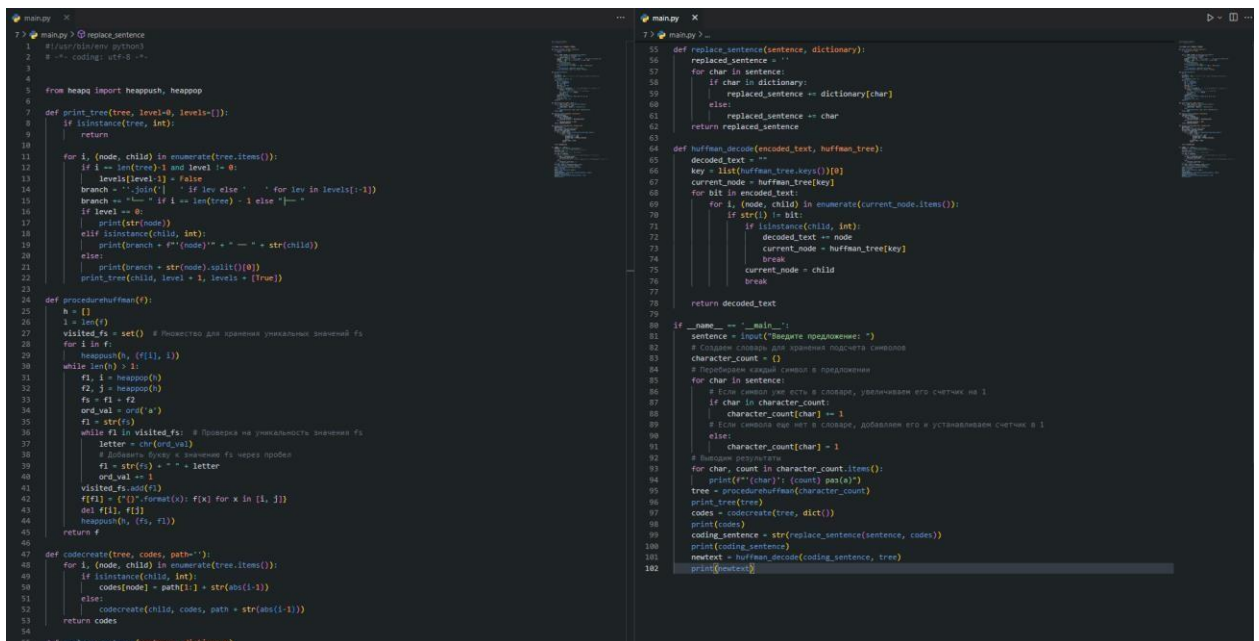
(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 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> main.py @ replace_sentence
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
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 = ""
15             if i % 2 == 0:
16                 branch = "0"
17             else:
18                 branch = "1"
19             if level == 0:
20                 print(str(node))
21             elif isinstance(child, int):
22                 print(branch + str(node) + " " + str(child))
23             else:
24                 print(branch + str(node).split()[0])
25                 print_tree(child, level+1, levels+[True])
26
27 def procedurehuffman():
28     h = []
29     s = len(f)
30     visited_fs = set() # Множество для хранения уникальных значений fs
31     for i in f:
32         heappush(h, (f[i], i))
33     while len(h) > 1:
34         f1, i = heappop(h)
35         f2, j = heappop(h)
36         fs = f1 + f2
37         ord_val = ord('a')
38         f1 = str(f1)
39         while f1 in visited_fs: # Проверка на уникальность значения fs
40             letter = chr(ord_val)
41             # Добавляем букву к значению fs через пробел
42             f1 = str(f1) + " " + letter
43             ord_val += 1
44         visited_fs.add(f1)
45         f[i] = f1
46         del f[i], f[j]
47         heappush(h, (fs, i))
48     return f
49
50 def codecreate(tree, codes, path=""):
51     for i, (node, child) in enumerate(tree.items()):
52         if isinstance(child, int):
53             codes[node] = path[i] + str(abs(i-1))
54         else:
55             codecreate(child, codes, path + str(abs(i-1)))
56     return codes
57
58 def replace_sentence(sentence, dictionary):
59     replaced_sentence = ""
60     for char in sentence:
61         if char in dictionary:
62             replaced_sentence += dictionary[char]
63         else:
64             replaced_sentence += char
65     return replaced_sentence
66
67 def huffman_decode(encoded_text, huffman_tree):
68     decoded_text = ""
69     key = list(huffman_tree.keys())[0]
70     current_node = huffman_tree[key]
71     for bit in encoded_text:
72         if str(bit) != bit:
73             if isinstance(child, int):
74                 decoded_text += node
75                 current_node = huffman_tree[key]
76                 break
77             current_node = child
78     return decoded_text
79
80 if __name__ == "__main__":
81     sentence = input("Введите предложение: ")
82     # Создание словаря для хранения подсчета символов
83     character_count = {}
84     # Пробежимся по каждому символу в предложении
85     for char in sentence:
86         # Если символ уже есть в словаре, увеличиваем его счетчик на 1
87         if char in character_count:
88             character_count[char] += 1
89         # Если символа еще нет в словаре, добавим его и установим счетчик в 1
90         else:
91             character_count[char] = 1
92     # Выводим статистику
93     for char, count in character_count.items():
94         print(f"{char}: {count} раз(a)")
95     tree = procedurehuffman(character_count)
96     print_tree(tree)
97     codes = codecreate(tree, dict())
98     print(codes)
99     coding_sentence = str(replace_sentence(sentence, codes))
100     print(coding_sentence)
101     newtext = huffman_decode(coding_sentence, tree)
102     print(newtext)
```

Рисунок 1. Код программы

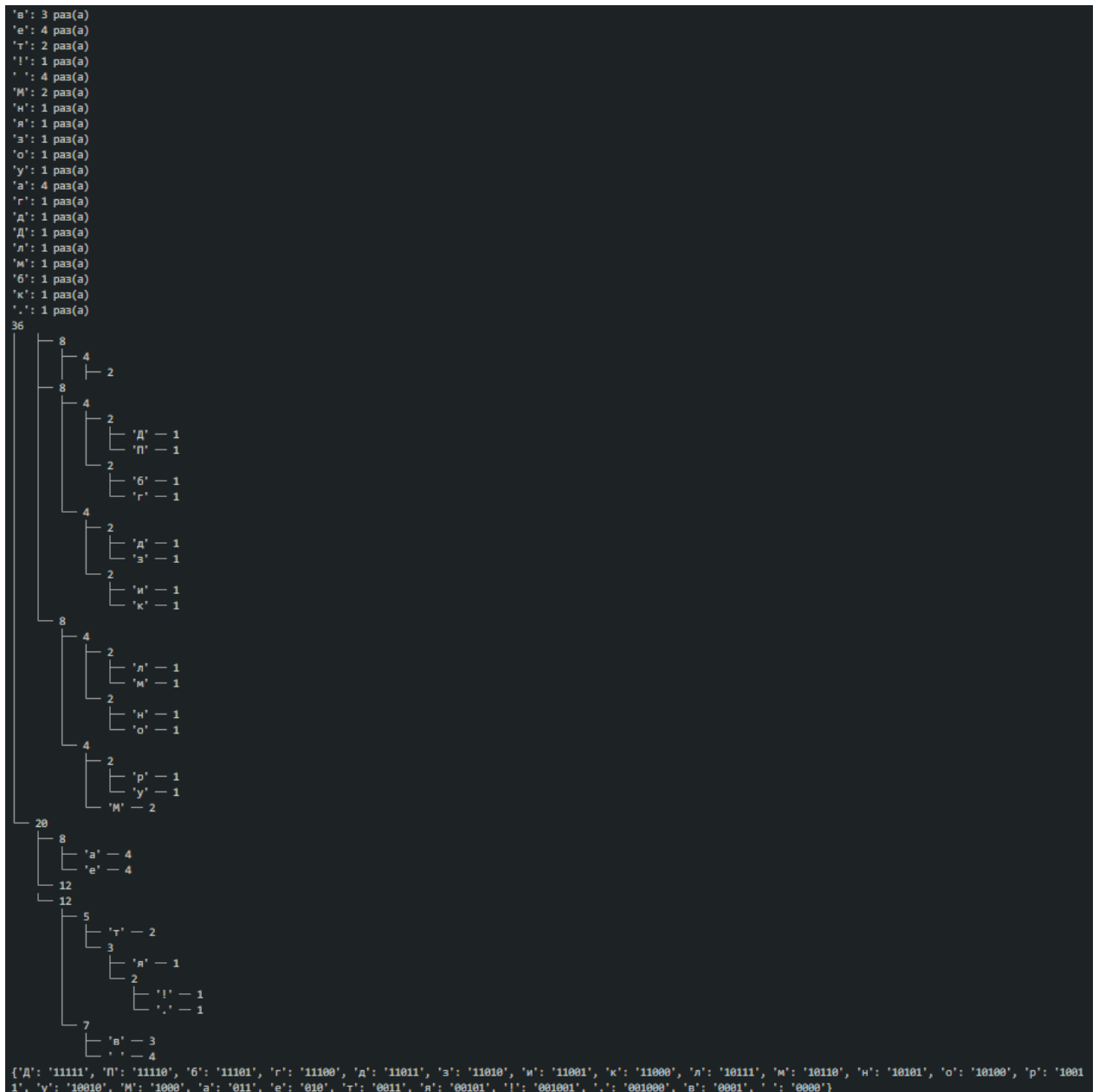


Рисунок 2. Результат выполнения программы

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.