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 №1
дисциплины «Алгоритмизация»

Выполнила:
Середа Кирилл Витальевич
1 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Ход выполнения заданий:

1) Написал программу для кодирования и декодирования текста при помощи кодировки Хаффмана

```
Введите текст: афцзалфзал фцлах фзх
Подсчет символов:
   'а': 4 раз
   'ф': 4 раз
   'ц': 2 раз
   'з': 3 раз
   'л': 3 раз
   ' ': 2 раз
   'х': 2 раз

Дерево Хаффмана:
20
├─ 8
│   ├─ 'а' — 4
│   └─ 'ф' — 4
└─ 12
    ├─ 5
    │   ├─ 'ц' — 2
    │   └─ 'з' — 3
    └─ 7
        ├─ 'л' — 3
        └─ 4
            ├─ ' ' — 2
            └─ 'х' — 2

Код Хаффмана:
   'а': 11
   'ф': 10
   'ц': 011
   'з': 010
   'л': 001
   ' ': 0001
   'х': 0000

Закодированный текст:
   11100110101100110010110010001100110011100000001100100000

Декодированный текст:
   афцзалфзал фцлах фзх
```

Рисунок 1 – Результат работы программы

Код программы:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from heapq import heappush, heappop


def display_tree(hierarchy, level=0, levels=None):
    if levels is None:
        levels = []
    if isinstance(hierarchy, int):
```

```
        return

    for i, (node, child) in enumerate(hierarchy.items()):
        if i == len(hierarchy) - 1 and level != 0:
            levels[level - 1] = False
        branch = ''.join('│   ' if lev else '    ' for lev in levels[:-1])
        branch += "└── " if i == len(hierarchy) - 1 else "├── "
        if level == 0:
            print(f"{node}")
        elif isinstance(child, int):
            print(f"{branch}'{node}' ── {child}")
        else:
            print(f"{branch}{str(node).split()[0]}")
        display_tree(child, level + 1, levels + [True])


def build_huffman_tree(frequencies):
    heap = []
    length = len(frequencies)
    visited_frequencies = set()

    for i in frequencies:
        heappush(heap, (frequencies[i], i))

    while len(heap) > 1:
        f1, i = heappop(heap)
        f2, j = heappop(heap)
        fs = f1 + f2
        ord_val = ord('a')
        fl = str(fs)

        while fl in visited_frequencies:
            letter = chr(ord_val)
            fl = str(fs) + " " + letter
            ord_val += 1

        visited_frequencies.add(fl)
        frequencies[fl] = {"{}".format(x): frequencies[x] for x in [i, j]}
        del frequencies[i], frequencies[j]
        heappush(heap, (fs, fl))

    return frequencies


def generate_huffman_codes(tree, codes, path=''):
    for i, (node, child) in enumerate(tree.items()):
        if isinstance(child, int):
            codes[node] = path[1:] + str(abs(i - 1))
        else:
            generate_huffman_codes(child, codes, path + str(abs(i - 1)))
    return codes


def substitute_text(text, code_dict):
    substituted_text = ''
    for char in text:
        if char in code_dict:
            substituted_text += code_dict[char]
        else:
            substituted_text += char
    return substituted_text


def huffman_decode(encoded_text, huffman_tree):
```

```
    decoded_text = ""
    key = list(huffman_tree.keys())[0]
    current_node = huffman_tree[key]

    for bit in encoded_text:
        for i, (node, child) in enumerate(current_node.items()):
            if str(i) != bit:
                if isinstance(child, int):
                    decoded_text += node
                    current_node = huffman_tree[key]
                    break
                current_node = child
                break

    return decoded_text


if __name__ == '__main__':
    input_sentence = input("Введите текст: ")
    char_count = {}

    for char in input_sentence:
        if char in char_count:
            char_count[char] += 1
        else:
            char_count[char] = 1

    print("Подсчет символов:")
    for char, count in char_count.items():
        print(f"   '{char}': {count} раз")

    huff_tree = build_huffman_tree(char_count)
    print("\nДерево Хаффмана:")
    display_tree(huff_tree)

    huff_codes = generate_huffman_codes(huff_tree, dict())
    print("\nКод Хаффмана:")
    for char, code in huff_codes.items():
        print(f"   '{char}': {code}")

    encoded_sentence = str(substitute_text(input_sentence, huff_codes))
    print("\nЗакодированный текст:")
    print(f"   {encoded_sentence}")

    decoded_text = huffman_decode(encoded_sentence, huff_tree)
    print("\nДекодированный текст:")
    print(f"   {decoded_text}")
```

Вывод: в ходе выполнения лабораторной работы был рассмотрен метод кодирования Хаффмана, был изучен алгоритм построения дерева Хаффмана. Исходя из изученного можно сделать вывод, что метод кодирования Хаффмана представляет собой эффективный способ сжатия данных.