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 ПРАКТИЧЕСКОЙ РАБОТЕ №7 дисциплины «Алгоритмизация» Вариант 7

Выполнил: Горбунов Данила Евгеньевич 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., канд. технических наук, доцент кафедры инфокоммуникаций (подпись) Отчет защищен с оценкой Дата защиты

Ход работы

1. Написал алгоритм, который рассчитывает частоту встречаемости каждого символа в тексте.

```
def main():
 sentence = input("Предложение: ")
 symbs = {}

 for char in sentence:
     if char in symbs:
         symbs[char] += 1
     else:
         symbs[char] = 1
     for char, count in symbs.items():
         print(f"'{char}': {count} pas")
```

Рисунок 1. Алгоритм подсчета количества символов

2. Написал алгоритм, который строит дерево в соответствии с процедурой Хаффмана

```
def huffman tree build(f):
h = []
buffer_fs = set()
     heappush(h, (f[i], i))
 while len(h) > 1:
     f1, i = heappop(h)
     f2, j = heappop(h)
     ord_val = ord('a')
     fl = str(fs)
     while fl in buffer fs:
          letter = chr(ord_val)
         fl = str(fs) + " " + letter
         ord val += 1
     buffer fs.add(fl)
     f[f1] = \{f''\{x\}'': f[x] \text{ for } x \text{ in } [i, j]\}
     del f[i], f[j]
     heappush(h, (fs, fl))
 return f
```

Рисунок 2. Алгоритм построения дерева

3. Написал алгоритм, который кодирует текст, используя результаты работы предыдущей функции.

```
def codingHuff(sentence, dictionary):
 replaced_sentence = ''
 for char in sentence:
     if char in dictionary:
         replaced_sentence += dictionary[char]
     else:
         replaced_sentence += char
 return replaced_sentence
```

Рисунок 3. Алгоритм кодирования

4. Написал алгоритм декодирования полученного результата

Рисунок 4. Алгоритм декодирования

5. Написал алгоритм, позволяющий наглядно увидеть код каждого символа.

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

Рисунок 9.Алгоритм code_dict

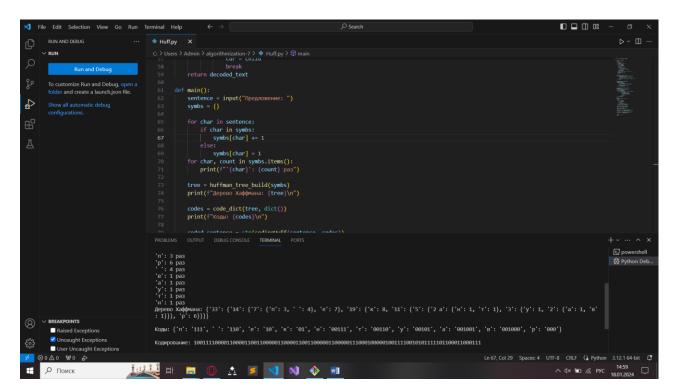


Рисунок 10. Результат работы программы

Вывод: В ходе выполнения данной практической работы был изучен и реализован алгоритм Хаффмана, используемый для сжатия данных. Эксперименты с различными входными данными позволили оценить эффективность алгоритма, выявив его достоинства. По результатам практической работы можно сказать, что алгоритм Хаффмана является крайне эффективным решением для оптимизации хранения и передачи данных.