МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №1

по дисциплине: Теория информации
тема: «Исследование кодирования по методу Хаффмана.
Оценка эффективности кода»

Выполнил: ст. группы ПВ-223
Дмитриев Андрей Александрович

Проверил:
Твердохлеб Виталий Викторович

Белгород 2024 г.

Цель работы: исследовать кодирования по методу Хаффмана. Научиться оценивать эффективность кода.

Задания лабораторной работы

Задание №1. Построить кодовое представление сообщения, вероятности появления символов в пределах алфавита которого приведены в табл.1.

Символ	s1	s2	s3	s4	s5	s6	s7	s8
Вероятность	0.23	0.19	0.16	0.16	0.10	0.10	0.05	0.01

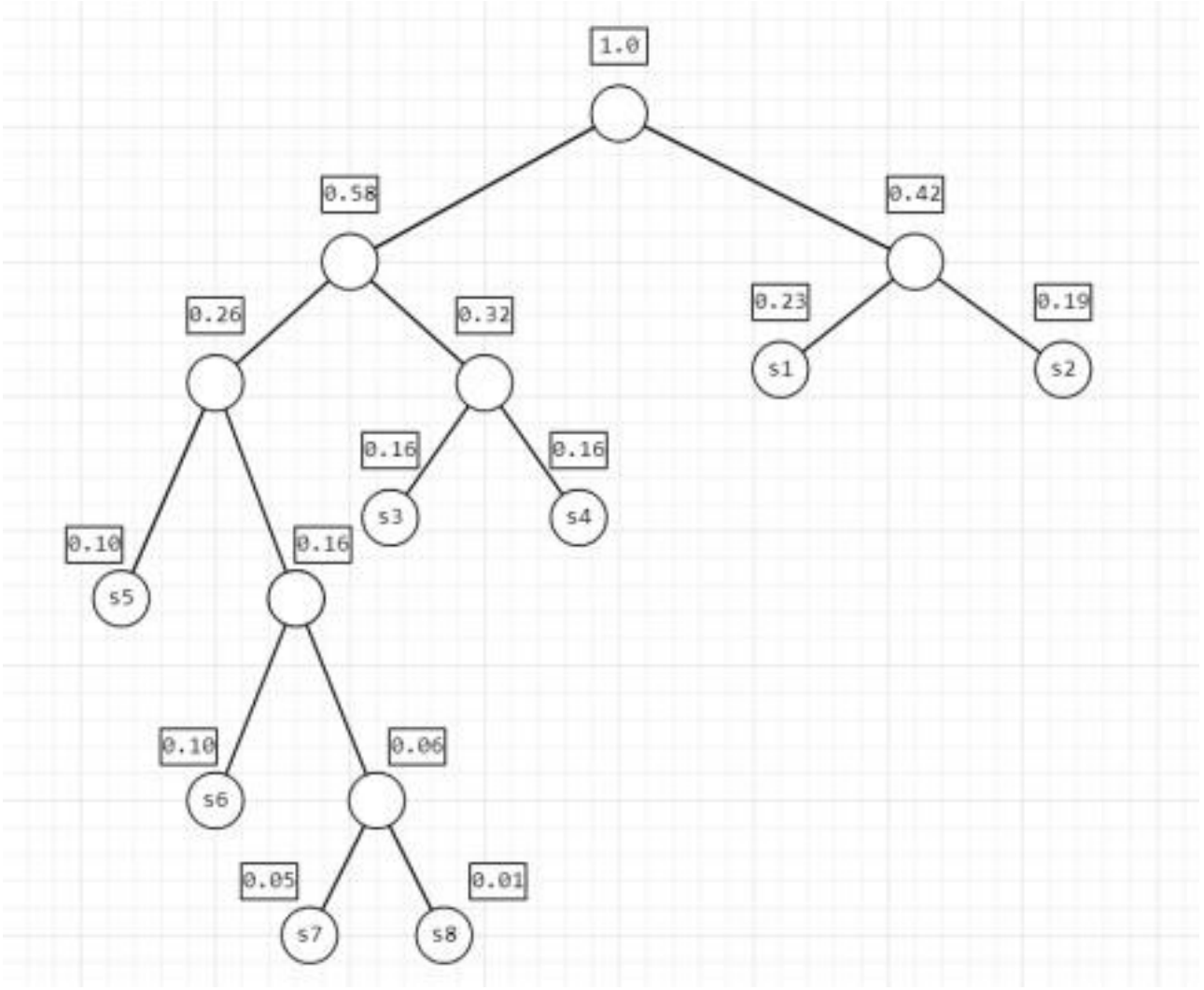
Кодовое представление сообщения мы можем построить благодаря методу деревьев Хаффмана.

1. Создать таблицу частотности символов в заданной строке или файле.
2. Создать лес узлов-деревьев, в котором каждый узел состоит из одного символа и его частоты в таблице.
3. Объединять два узла дерева с наименьшими частотами в новый узел, который будет иметь сумму их частот.
4. Удалить объединенные узлы из леса и добавить новый узел в лес. 5. Повторять шаги 3-4 до тех пор, пока не останется один узел в лесу - корень дерева.
6. Присвоить коды символам, используя левое направление (0) для листьев, а правое направление (1) для узлов-родителей.
7. Создать закодированную версию строки, используя полученные коды символов.
8. Создать таблицу декодирования символов, используя коды символов.
9. Декодировать закодированную строку, используя таблицу декодирования символов.

Ниже представлено полученное дерево Хаффмана.

В таблице есть символы с одинаковой вероятностью, значит, могут быть построены различные деревья, однако их эффективность останется неизменной.

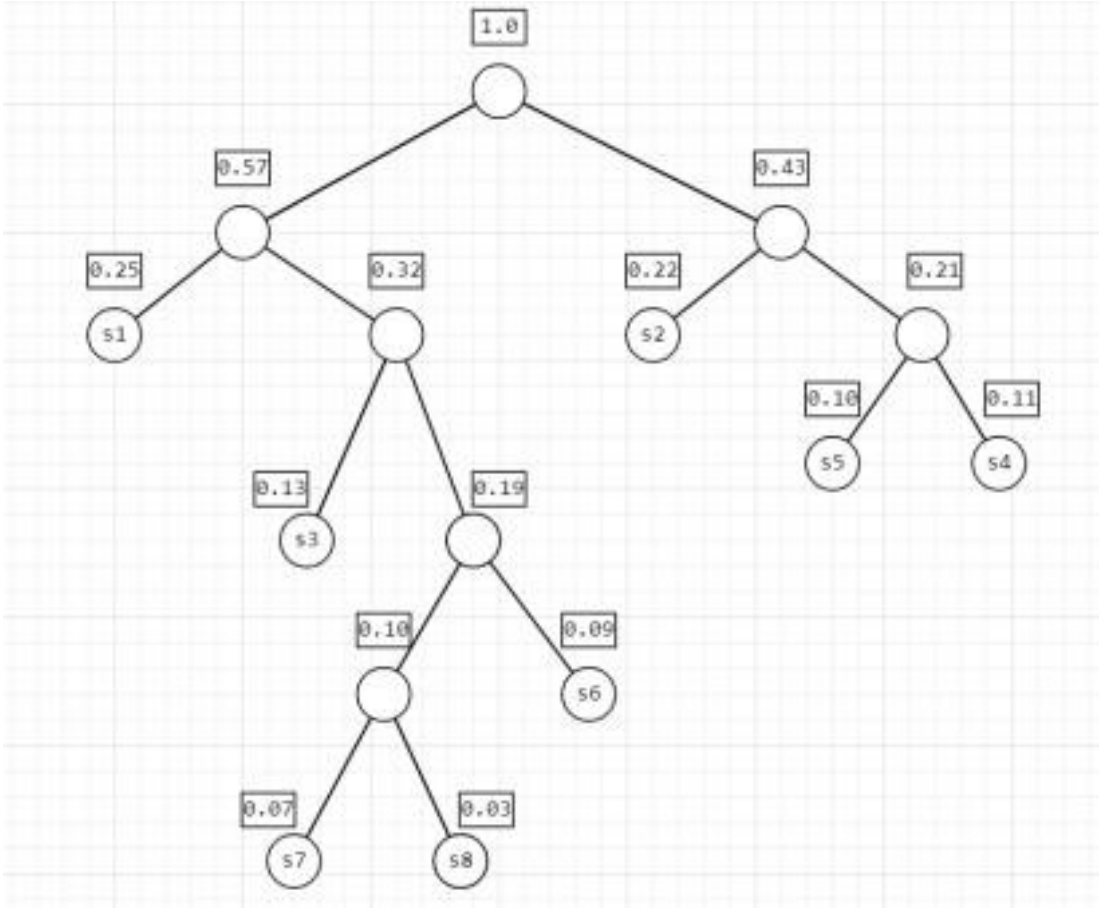
Символ	s1	s2	s3	s4	s5	s6	s7	s8
Вероятность	0.25	0.22	0.13	0.11	0.1	0.09	0.07	0.03



Коды символов

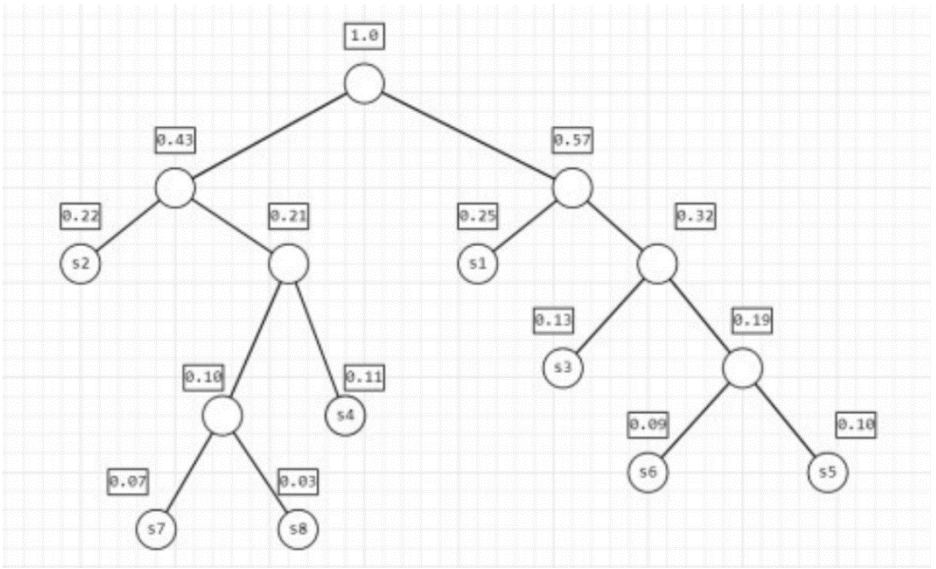
S1	10
S2	11
S3	010
S4	011
S5	000
S6	0010
S7	00110
S8	00111

Задание №2. Построить кодовое представление сообщения, вероятности появления символов в пределах алфавита которого приведены в табл.2.



S1	00
S2	10
S3	010
S4	111
S5	110
S6	0111
S7	01100
S8	01101

Пример альтернативного дерева (другая эффективность)



S1	10
S2	00
S3	110
S4	011
S5	1111
S6	1110
S7	0100
S8	0101

Задача №3. Построить кодовое представление сообщения:
оитомии о ими ооо итми о о о ооиимтомиимотоим оои тоо и и м оио и омтоо тоимо т и

Решение:
Построим таблицу появления символов в сообщении.

Символ	о	и	Т	м	‘ ‘
Кол-во появлений	25	19	8	10	18

Построим закодированное сообщение:
оитомии о ими ооо итми о о о ооиимтомиимотоим оои тоо и и м оио и омтоо тоимо т и =
11100101101110100011001001110001111111001001110001100110011001111101001101011011101001111010111001100111110000101111001000100001100111011001000110110101111000101110011111000100010

Задание №4.
Для условий, приведенных в заданиях 1 и 2 и 3, выявить возможность построения альтернативных кодовых моделей сообщения. В случае обнаружения таковых, выявить наиболее эффективные из них по критериям K_{comp} и δ.

Решение:

Возьмём для расчёта следующий вариант кодирования:
$s1 = 01$, $s2 = 00$, $s3 = 101$, $s4 = 110$, $s5 = 1111$, $s6 = 100$, $s7 = 11101$, $s8 = 11100$.

Вычислим среднюю длину кода (L) и энтропию (H) для кодирования:

$$L = \sum_{i=1}^{8} p_i l_i$$
$$= 0{,}23 * 2 + 0{,}19 * 2 + 0{,}16 * 3 + 0{,}16 * 3 + 0{,}10 * 4 + 0{,}10 * 3 + 0{,}05 * 5 + 0{,}01 * 5 = 2{,}8$$

$$H = -\sum_{i=1}^{8} \log_2 p_i$$
$$= -(0{,}23 * log2\ 0.23 + 0{,}19 * log2\ 0.19 + 0{,}16 * log2\ 0.16 + 0{,}16 * log2\ 0.16 + 0{,}10 * log2\ 0.10 + 0{,}10 * log2\ 0.10 + 0{,}05 * log2\ 0.05 + 0{,}01 * log2\ 0.01) \approx 2.74$$

Получим ответ:

$$K_{comp} = \frac{H}{L} = 0.98 \qquad \delta = 1 - K_{comp} = 0.02$$

Кодовая реализация.

```
import heapq
from collections import defaultdict


class Node:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.freq < other.freq


def build_huffman_tree(text):
    # Подсчет частоты символов в тексте
    freq_map = defaultdict(int)
    for char in text:
        freq_map[char] += 1

    # Создание очереди с приоритетом из узлов-листьев
    priority_queue = []
    for char, freq in freq_map.items():
        heapq.heappush(priority_queue, Node(char, freq))

    # Строим дерево Хаффмана
    while len(priority_queue) > 1:
        left_node = heapq.heappop(priority_queue)
        right_node = heapq.heappop(priority_queue)
        merged_node = Node(None, left_node.freq + right_node.freq)
        merged node.left = left node
        merged_node.right = right_node
        heapq.heappush(priority_queue, merged_node)

    return priority_queue[0]


def build_huffman_codes(node, current_code, huffman_codes):
    if node is None:
        return

    # Если узел - лист, то добавляем его код в словарь кодов Хаффмана
    if node.char is not None:
        huffman_codes[node.char] = current_code
        return

    # Рекурсивно строим коды для левого и правого поддеревьев
    build_huffman_codes(node.left, current_code + "0", huffman_codes)
    build_huffman_codes(node.right, current_code + "1", huffman_codes)


def huffman_encoding(text):
    root = build_huffman_tree(text)
    huffman_codes = {}
    build_huffman_codes(root, "", huffman_codes)

    encoded_text = ""
    for char in text:
        encoded_text += huffman_codes[char]

    return encoded_text, huffman_codes


def huffman_decoding(encoded_text, huffman_codes):
    decoded_text = ""
    shift = 0
```

```
    while shift < len(encoded_text):
        for val, code in huffman_codes.items():
            if code in encoded_text[shift:len(code)+shift]:
                decoded_text += val
                shift += len(code)

    return decoded_text


# Пример использования
text = input("Введите текст: ")
encoded_text, huffman_codes = huffman_encoding(text)
print("Закодированный текст:", encoded_text, "Длинна:", len(text), "->",
len(encoded_text))
print("Коды Хаффмана:", huffman_codes)
decoded_text = huffman_decoding(encoded_text, huffman_codes)
print("Раскодированный текст:", decoded_text)
```

Результаты работы алгоритма:

```
Введите текст: tttt 2 tttttttt2tttttttt ttttttttttt
Закодированный текст: 11110100011111111100111111110111111111111 Длинна: 296 -> 42
Коды Хаффмана: {'2': '00', ' ': '01', 't': '1'}
Раскодированный текст: tttt 2 tttttttt2tttttttt ttttttttttt
```

```
Введите текст: 1234567890qwertyuiopasdfghj
Закодированный текст:
 0101000000101110111000111001110100100011100010111101010010000100101101101111011000111111
 1111011110010101100011100010011010101100 Длинна: 216 -> 130
Коды Хаффмана: {'2': '0000', '5': '0001', '0': '0010', 'i': '0011', '8': '0100', '1':
 '01010', '3': '01011', 'j': '01100', 'h': '01101', '9': '01110', 'y': '01111', 'e':
 '10000', 'd': '10001', 'r': '10010', 'g': '10011', 'w': '10100', 's': '10101', 'u':
 '10110', '4': '10111', 'f': '11000', '6': '11001', '7': '11010', 't': '11011', 'a':
 '11100', 'p': '11101', 'q': '11110', 'o': '11111'}
Раскодированный текст: 1234567890qwertyuiopasdfghj
```

```
Введите текст: qwtyqweurquuqwurutqutruqwuruq
Закодированный текст: 011000010000011000001111010111110110011101110010111001101110110011101101
 Длинна: 232 -> 73
Коды Хаффмана: {'y': '0000', 'e': '0001', 't': '001', 'q': '01', 'w': '100', 'r': '101', 'u': '11'}
Раскодированный текст: qwtyqweurquuqwurutqutruqwuruq
```

Вывод: в ходе выполнения лабораторной работы были изучен алгоритм кодирования по методу Хаффмана, и применён метод оценки эффективности кода Хаффмана.