МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра МОиСИТ

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №7.2

«Алгоритмы кодирования и сжатия данных»
по дисциплине
«Структуры и алгоритмы обработки данных»

Выполнил студент группы ИКБО-10-24 Бикташев И. И.

Практическую работу выполнил «__»_____2025 г.

«Зачтено» «__»_____2025 г.

Москва 2025

Цель работы: Исследовать алгоритмы сжатия на примерах. Разработать программы сжатия и восстановления текста методами Хаффмана и Шеннона – Фано.

Шифрование методом Шеннона-Фано: строка для кодирования:

Прибавь к ослиной голове

Еще одну, получишь две.

Но сколько б ни было ослов,

Они и двух не свяжут слов.

Составим таблицу количества вхождений символов в строку.

Алфавит	Количество вхождений	Вероятность
Пробел	19	0,1863
о	14	0,1373
в	7	0,0686
л	7	0,0686
н	6	0,0588
и	6	0,0588
е	5	0,0490
с	5	0,0490
у	4	0,0392
к	3	0,0294
д	3	0,0294
б	3	0,0294
ь	3	0,0294
,	2	0,0196
п	2	0,0196
.	2	0,0196
г	1	0,0098
р	1	0,0098

щ	1	0,0098
ы	1	0,0098
ч	1	0,0098
ж	1	0,0098
т	1	0,0098
ш	1	0,0098
а	1	0,0098
я	1	0,0098
й	1	0,0098

Разделим символы на две группы с суммарной вероятностью ~0.5

1. Пробел, о, в, л, н, и – префикс «0»
2. Все остальные символы – префикс «1»

Составим коды для каждого символа в соответствии с его группой

Алфавит	1 цифра	2 цифра	3 цифра	4 цифра	5 цифра	6 цифра
Пробел	**0**	**0**	**0**			
о	**0**	**0**	**1**			
в	**0**	**1**	**0**	**0**		
л	**0**	**1**	**0**	**1**		
н	**0**	**1**	**1**	**0**		
и	**0**	**1**	**1**	**1**		
е	**1**	**0**	**0**	**0**		
с	**1**	**0**	**0**	**1**		
у	**1**	**0**	**1**	**0**	**1**	
к	**1**	**0**	**1**	**0**	**1**	
д	**1**	**0**	**1**	**1**	**0**	
б	**1**	**0**	**1**	**1**	**1**	
ь	**1**	**1**	**0**	**0**	**0**	**0**
,	**1**	**1**	**0**	**0**	**0**	**1**
п	**1**	**1**	**0**	**0**	**1**	**0**

.	**1**	**1**	**0**	**0**	**1**	**1**
г	**1**	**1**	**0**	**1**	**0**	**0**
р	**1**	**1**	**0**	**1**	**0**	**1**
щ	**1**	**1**	**0**	**1**	**1**	**0**
ы	**1**	**1**	**0**	**1**	**1**	**1**
ч	**1**	**1**	**1**	**0**	**0**	**0**
ж	**1**	**1**	**1**	**0**	**0**	**1**
т	**1**	**1**	**1**	**0**	**1**	**0**
ш	**1**	**1**	**1**	**0**	**1**	**1**
а	**1**	**1**	**1**	**1**	**0**	**0**
я	**1**	**1**	**1**	**1**	**0**	**1**
й	**1**	**1**	**1**	**1**	**1**	**0**

Построим дерево Шеннона-Фано для полученных кодов (рисунок 1)

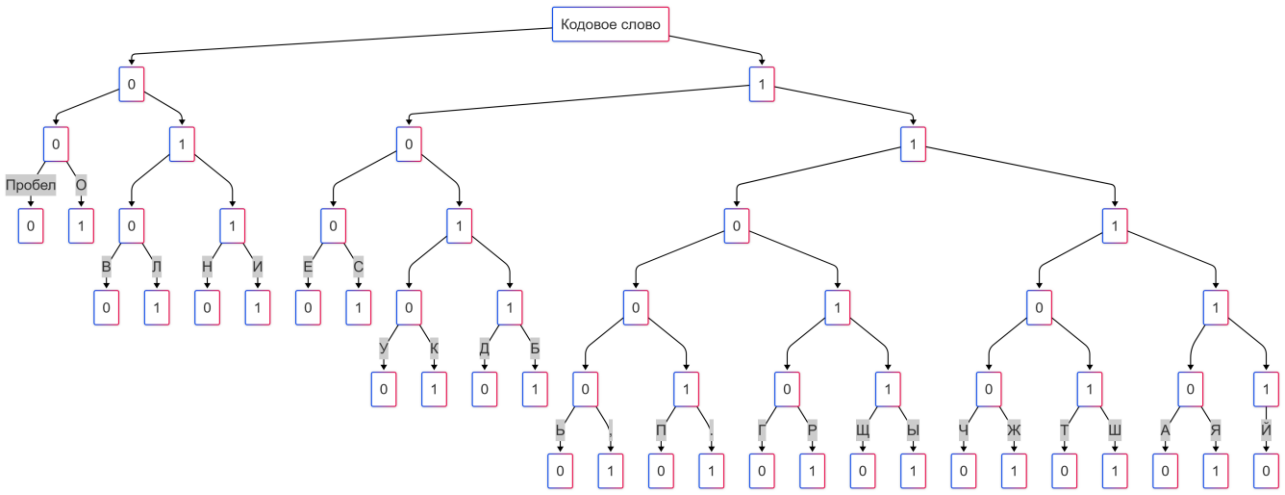

Рисунок 1 - Дерево Шеннона-Фано

Общий вес строки в битах в данной кодировке – 428 бит.

Вес строки в стандартной кодировке – 102 * 8 бит = 816 бит.

Шифрование методом LZ77: строка для кодирования 101000100101010001011

Основная идея алгоритма это замена повторного вхождения строки ссылкой на одну из предыдущих позиций вхождения. Для этого используют метод скользящего окна.

В алгоритме LZ77 совпадения строки кодируются тройкой:

1. i – Смещение

2. j – Длина совпадения

3. s – Первый несовпадающий элемент

Кодируемая пара трактуется именно как команда копирования символов из скользящего окна с определенной позиции, или дословно как: «Вернуться в словаре на значение смещения символов и скопировать значение длины символов, начиная с текущей позиции и добавить первый несовпадающий элемент».

Составим таблицу итераций.

N	Скользящее окно		Совпадающее значение	Закодированная тройка		
	Строка	Буфер		i	j	s
1	-	10100	-	0	0	1
2	1	01000	-	0	0	0
3	10	10001	10	2	2	0
4	10100	01001	0100	4	4	1
5	1010001001	01010	010	5	3	1
6	10100010010101	00010	00010	11	5	1
7	10100010010101000101	1	-	0	0	1
8	101000100101010001011	-	-	-1	-	-

Закодированная строка: (0, 0, 1), (0, 0, 0), (2, 2, 1), (4, 4, 1), (5, 3, 1), (11, 5, 1), (0, 0, 1), -1

Расшифруем получившуюся строку и сравним ее с исходной

Закодированная тройка			Результат
i	j	s	
0	0	1	1
0	0	0	10
2	2	0	10100
4	4	1	1010001001
5	3	1	10100010010101
11	5	1	10100010010101000101
0	0	1	101000100101010001011

Строки совпадают – кодировка выполнена успешно

Шифрование методом LZ78: строка для кодирования какатанекатанекатата

Метод LZ78 схож с LZ77, но он работает только с уже обработанными символами, т. е. не имеет буфера. Вместо этого LZ78 ищут максимальную подстроку уже присутствующую в словаре, из обработанных элементов. Если такой строки нет, то символ просто добавляется в словарь, иначе в словарь добавляется элемент, составленный из подстроки и нового символа.

Составим таблицу итераций.

N	Словарь	Считанная строка	Закодированная двойка	
			i	s
1	-	к	0	к
2	к	а	0	а
3	к, а	ка	1	а
4	к, а, ка	т	0	т
5	к, а, ка, т	ан	2	н
6	к, а, ка, т, ан	е	0	е
7	к, а, ка, т, ан, е,	кат	3	т
8	к, а, ка, т, ан, е, кат	ане	5	е
9	к, а, ка, т, ан, е, кат, ане	ката	7	а
10	к, а, ка, т, ан, е, кат, ане, ката	та	4	а
11	к, а, ка, т, ан, е, кат, ане, ката, та	-	-	-

Закодированная строка: (0, к), (0, а), (1, а), (0, т), (2, н), (0, е), (3, т), (5, е), (7, а), (4, а). Расшифруем получившуюся строку и сравним ее с исходной

Закодированная двойка		Результат
i	s	
0	к	к
0	а	к, а
1	а	к, а, ка
0	т	к, а, ка, т
2	н	к, а, ка, т, ан
0	е	к, а, ка, т, ан, е,
3	т	к, а, ка, т, ан, е, кат
5	е	к, а, ка, т, ан, е, кат, ане
7	а	к, а, ка, т, ан, е, кат, ане, ката
4	а	к, а, ка, т, ан, е, кат, ане, ката, та

Строки совпадают – кодировка выполнена успешно

Программа сжатия текста алгоритмом Шеннона – Фано:

В качестве узла дерева Шеннона – Фано используется структура ShannonFanoNode (листинг 1)

Листинг 1 - Узел дерева

```
struct ShannonFanoNode {
    char symbol;
    int frequency;
    string code;

    ShannonFanoNode(char s, int f) : symbol(s), frequency(f), code("") {}
};
```

Для начала нам надо для каждого символа(узла) надо вычислить количество его вхождений текста. Для этого напишем функцию calculateFrequencies (листинг 2)

Листинг 2 - Вычисление вхождений

```
unordered_map<char, int> calculateFrequencies(const string& text) {
    unordered_map<char, int> frequencies;
    for (char c : text) {
        frequencies[c]++;
    }
    return frequencies;
}
```

После этого вычисляем коды для каждого узла дерева. Для этого делим символы на две группы одинаковой суммарной встречаемости и назначаем им разные префиксы. После рекурсивно вызываем эту же функцию для этих двух групп, пока длина группы не станет 1 (листинг 3)

Листинг 3 - Вычисление кодов

```
void buildShannonFanoCodes(vector<ShannonFanoNode>& nodes, int start, int end)
{
    if (start >= end || start == end - 1)
        return;
    int total = 0;
    for (int i = start; i < end; i++) {
        total += nodes[i].frequency;
    }
    int sum = 0, splitIndex = start, minDiff = INT_MAX;
    for (int i = start; i < end; i++) {
        sum += nodes[i].frequency;
        int diff = abs(total - 2 * sum);
        if (diff < minDiff) {
            minDiff = diff;
            splitIndex = i + 1;
        }
    }
    for (int i = start; i < splitIndex; i++) {
        nodes[i].code += "0";
    }
    for (int i = splitIndex; i < end; i++) {
        nodes[i].code += "1";
    }
    buildShannonFanoCodes(nodes, start, splitIndex);
    buildShannonFanoCodes(nodes, splitIndex, end);
}
```

Результаты работы программы на строке, введённой с клавиатуры представлены на рисунке 2.

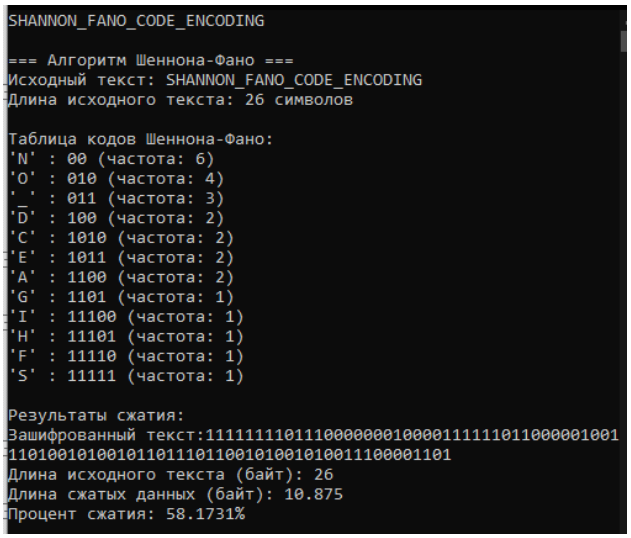

Рисунок 2 - Результат работы программы

Программа сжатия текста алгоритмом Хаффмана:

В качестве узла дерева Хаффмана используется структура HuffmanNode (листинг 4).

Листинг 4 – Узел дерева Хаффмана

```
struct HuffmanNode {
    char symbol;
    int frequency;
    HuffmanNode* left;
    HuffmanNode* right;

    HuffmanNode(char s, int f) : symbol(s), frequency(f), left(nullptr),
right(nullptr) {}
    HuffmanNode(int f, HuffmanNode* l, HuffmanNode* r)
        : symbol('\0'), frequency(f), left(l), right(r) {}
};
```

В дереве Хаффмана, только листья хранят в себе символ, который они шифруют, и его встречаемость; остальные же узлы дерева хранят суммарную встречаемость своих поддеревьев.

Для шифровки строки методом Хаффмана требуется для начала внести все уникальные символы строки в дерево. Для построения данного дерева используется очередь приоритетов, отсортированная по количеству вхождений символа. Уникальные символы строки будут являться лишь листьями дерева, для построения оставшихся узлов дерева требуется пройтись по всей очереди приоритетов и связать узлы между собой, создав каждому узлу дерева его родительский узел, вплоть до корневого (листинг 5)

Листинг 5 – Построение дерева

```
    string fullName = "бикташев искандер ирикович";
    unordered_map<char, int> frequencies = calculateFrequencies(fullName);
    priority_queue<HuffmanNode*,
                   vector<HuffmanNode*>,
                   CompareNodes> pq;
    for (auto& pair : frequencies) {
        pq.push(new HuffmanNode(pair.first, pair.second));
    }
    while (pq.size() > 1) {
        auto left = pq.top(); pq.pop();
        auto right = pq.top(); pq.pop();

        int combinedFreq = left->frequency + right->frequency;
        auto parent = new HuffmanNode(combinedFreq, left, right);
        pq.push(parent);
    }

    HuffmanNode* root = pq.top();
```

Для назначения каждому символу строки своего кодового слова пройдёмся по всем узлам дерева в глубину и на каждой глубине будем добавлять 0 или 1 в зависимости от того, в каком направлении пошла рекурсия (листинг 6)

Листинг 6 – Вычисление кодовых слов

```
void buildHuffmanCodes(HuffmanNode* node,
                       const string& code,
                       unordered_map<char, string>& codes) {
    if (!node)
        return;

    if (node->symbol != '\0') {
        codes[node->symbol] = code;
        return;
    }

    buildHuffmanCodes(node->left, code + "0", codes);
    buildHuffmanCodes(node->right, code + "1", codes);
}
```

Результат работы программы показан на рисунке 3

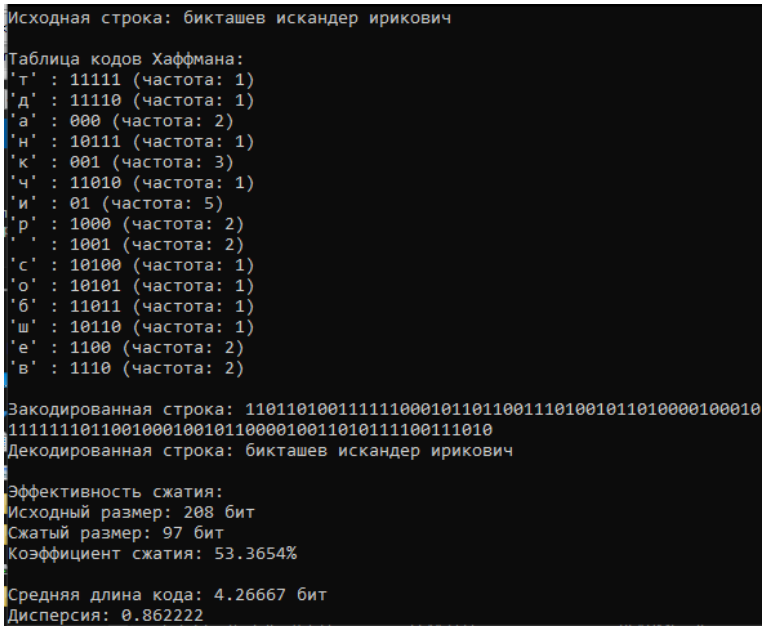

Рисунок 3 – Результат работы программы

На рисунке 4 представлено дерево Хаффмана для данной исходной строки

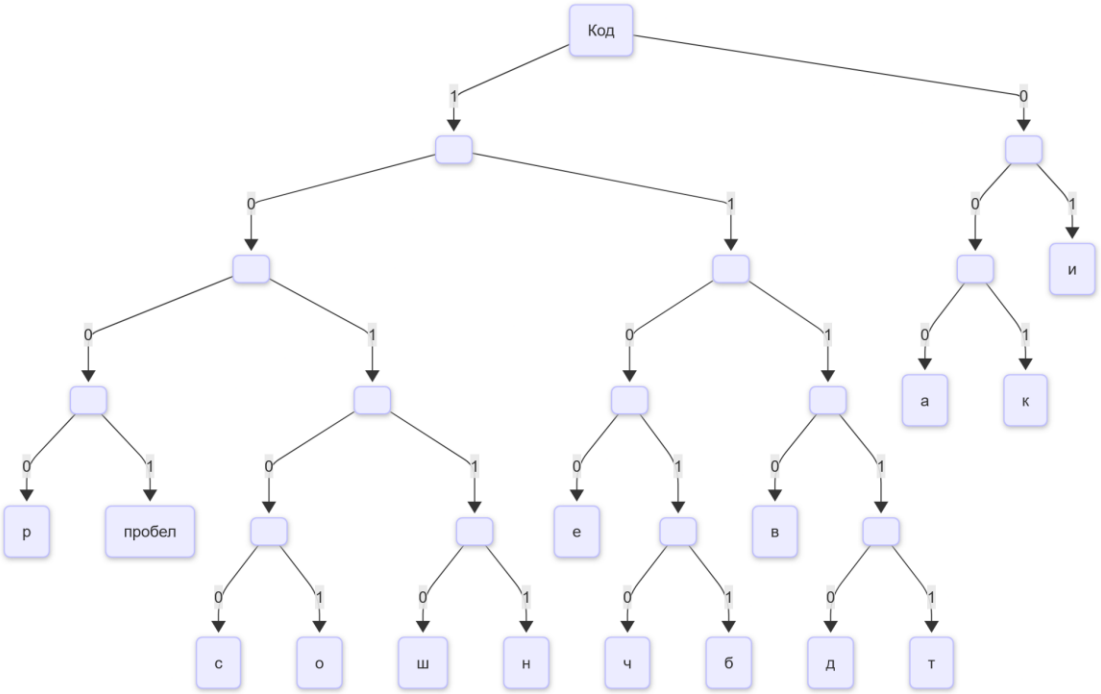

Рисунок 4 – Дерево Хаффмана

Вывод

Поставленные задачи выполнены: изучены и реализованы алгоритмы по сжатию данных. Реализован алгоритм сжатия с помощью методов Шеннона – Фано и Хаффмана.

Литература

Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/cpp/cpp/ (дата обращения 31.10.2025)

Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 31.10.2025)