МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

«Отдельные вопросы алгоритмизации»

по дисциплине «Структуры и алгоритмы обработки данных (часть 2/2)»

Выполнил студент группы ИКБО-41-23 Попов А.В.

Принял Рысин М.Л.
Ассистент

Практические работы выполнены «___»_____2024 г. (подпись студента)

«Зачтено» «___»_____2024 г. (подпись преподавателя)

Москва 2024

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Получение практических навыков и знаний по выполнению сжатия данных рассматриваемыми методами.

2 ПОСТАНОВКА ЗАДАЧИ

Задание 1 Исследование алгоритмов сжатия на примерах

1) Выполнить каждую задачу варианта, представив алгоритм решения в виде таблицы и указав результат сжатия. Примеры оформления решения представлены в Приложении1 этого документа.

2) Описать процесс восстановления сжатого текста.

3) Сформировать отчет, включив задание, вариант задания, результаты выполнения задания варианта.

Задание 2 Разработать программы сжатия и восстановления текста методами Хаффмана и Шеннона – Фано.

1) Реализовать и отладить программы.

2) Сформировать отчет по разработке каждой программы в соответствии с требованиями.

- По методу Шеннона-Фано привести: постановку задачи, описать алгоритм формирования префиксного дерева и алгоритм кодирования, декодирования, код и результаты тестирования. Рассчитать коэффициент сжатия. Сравнить с результат сжатия вашим алгоритмом с результатом любого архиватора.

- по методу Хаффмана выполнить и отобразить результаты выполнения всех требований, предъявленных в задании и оформить разработку программы: постановка, подход к решению, код, результаты тестирования.

Вариант №17. Условие задания:

Закодировать фразу методами Шеннона–Фано	Сжатие данных по методу Лемпеля–Зива LZ77 Используя двух символьный алфавит (0, 1) закодировать следующую фразу	Закодировать следующую фразу, используя код LZ78
Плыл по морю чемодан, В чемодане был диван, На диване ехал слон. Кто не верит – выйди вон!	00010000101010011101	webwerbweberweberweb

3 ЗАДАНИЕ 1.1

3.1 Алгоритм решения

Метод Шеннона-Фано — это алгоритм сжатия данных, который используется для кодирования символов на основе их частоты. Суть метода заключается в том, что каждый символ получает уникальный код, и чем чаще символ встречается в тексте, тем короче будет его код.

Процесс работает следующим образом: сначала для каждого символа вычисляется его частота появления в тексте, затем символы сортируются по убыванию частоты. После этого весь список символов делится на две группы таким образом, чтобы суммы частот в этих группах были как можно более равными. Этот процесс деления повторяется рекурсивно, пока не останется по одному символу в каждой группе. Символы из левой группы получают код с префиксом "0", а символы из правой группы — с префиксом "1".

После того как процесс завершен, получается кодировка для каждого символа, которая используется для сжатия данных.

Процесс восстановления сжатого текста методом Шеннона–Фано заключается в использовании ранее созданного словаря кодов для преобразования закодированной последовательности обратно в исходный текст. Каждый символ в словаре имеет уникальный префиксный код, поэтому декодирование выполняется посимвольно. Этот метод восстановления гарантирует точное восстановление текста, так как коды уникальны и не пересекаются.

Для варианта №17 кодировка фразы «Плыл по морю чемодан, В чемодане был диван, На диване ехал слон. Кто не верит – выйди вон!» представлена в Таблице 1.

Символ	Кол-во	1-я цифра	2-я цифра	3-я цифра	4-я цифра	5-я цифра	6-я цифра	7-я цифра	Код	Кол-во бит
пробел	**17**	0	0	0					000	51
о	**7**	0	0	1					001	21
ее	**7**	0	1	0					010	21
нн	**7**	0	1	1	0				0110	28
а	**6**	0	1	1	1				0111	24
лл	**5**	1	0	0	0				1000	20
дд	**5**	1	0	0	1				1001	20
в	**5**	1	0	1	0				1010	20
и	**4**	1	0	1	1	0			10110	20
ыы	**3**	1	0	1	1	1			10111	15
мм	**3**	1	1	0	0	0			11000	15
рр	**2**	1	1	0	0	1	0		110010	12
,	**2**	1	1	0	0	1	1		110011	12
чч	**2**	1	1	0	1	0			11010	10
тт	**2**	1	1	0	1	1	0		110110	12
пп	**1**	1	1	0	1	1	1		110111	6
ПП	**1**	1	1	1	0	0	0		111000	6
.	**1**	1	1	1	0	0	1	0	1110010	7
юю	**1**	1	1	1	0	0	1	1	1110011	7
В	**1**	1	1	1	0	1	0		111010	6
!	**1**	1	1	1	0	1	1	0	1110110	7
бб	**1**	1	1	1	0	1	1	1	1110111	7
НН	**1**	1	1	1	1	0	0		111100	6
хх	**1**	1	1	1	1	0	1	0	1111010	7

Символ	Кол-во	1-я цифра	2-я цифра	3-я цифра	4-я цифра	5-я цифра	6-я цифра	7-я цифра	Код	Кол-во бит
с	**1**	1	1	1	1	0	1	1	1111011	7
К	**1**	1	1	1	1	1	0		111110	6
-	**1**	1	1	1	1	1	1	0	1111110	7
йй	**1**	1	1	1	1	1	1	1	1111111	7
										387

Таблица 1 — Сжатие методом Шеннона-Фано

Не закодированная фраза — 91 * 8 = 728 бит

Закодированная фраза — 387 бит

Закодированная строка:

111000100010111100000011011100100011000001110010111001100011010010110000011001011101101100110
001110100001101001011000001100101110110010000111011110111100000010011011010100111011011001100
011110001110001001101101010011101100100000101111010011110000001111011100000101101110010000111
110110110001000011001000010100101100101011011011000011111100001010101111111111100110110000101
000101101110110

3.2 Реализация задачи и тестирование

```
struct Symbol {
    char character;
    int frequency;
    string code;
};

int splitSymbols(vector<Symbol>& symbols, int start, int end) {
    int total = 0;
    for (int i = start; i <= end; i++)
        total += symbols[i].frequency;

    int half = total / 2;
    int sum = 0;

    for (int i = start; i <= end; i++) {
        sum += symbols[i].frequency;
        if (sum >= half) {
            if (sum - half <= half - (sum - symbols[i].frequency)) {
                return i;
            }
            else {
                if (i - 1 >= start) {
                    return i - 1;
                }
                return i;
            }
        }
    }
    return start;
}
```

Рисунок 1 — Стукрура символа и функция разделения списка символов

```cpp
void shannonFano(vector<Symbol>& symbols, int start, int end) {
    if (start >= end)
        return;

    int split = splitSymbols(symbols, start, end);

    for (int i = start; i <= split; i++)
        symbols[i].code += "0";
    for (int i = split + 1; i <= end; i++)
        symbols[i].code += "1";

    shannonFano(symbols, start, split);
    shannonFano(symbols, split + 1, end);
}

unordered_map<char, string> encodeShannonFano(const string& text, vector<Symbol>& symbols) {
    unordered_map<char, int> frequencyMap;
    for (char ch : text)
        frequencyMap[ch]++;

    for (const auto& pair : frequencyMap)
        symbols.push_back({ pair.first, pair.second, "" });

    sort(symbols.begin(), symbols.end(), [](const Symbol & a, const Symbol & b) {return a.frequency > b.frequency;});

    shannonFano(symbols, 0, symbols.size() - 1);

    unordered_map<char, string> codeMap;
    for (const Symbol& symbol : symbols)
        codeMap[symbol.character] = symbol.code;

    return codeMap;
}
```

Рисунок 2 — Рекурсивная функция кодирования символов и основная функция сжатия методом Шеннона-Фано

```cpp
int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    string text = "Плыл по морю чемодан, В чемодане был диван, На диване ехал слон. Кто не верит – выйди вон!";
    vector<Symbol> symbols;
    unordered_map<char, string> codeMap = encodeShannonFano(text, symbols);

    cout << "Символы, их частота появления и коды:\n";
    for (const Symbol& symbol : symbols)
        cout << "'" << symbol.character << "'"
        << " -> Частота появления: " << symbol.frequency
        << ", Код: " << symbol.code << endl;

    cout << "\nЗакодированная строка: ";
    for (char ch : text)
        cout << codeMap[ch];
    cout << endl;

    return 0;
}
```

Рисунок 3 — Основная функция программы

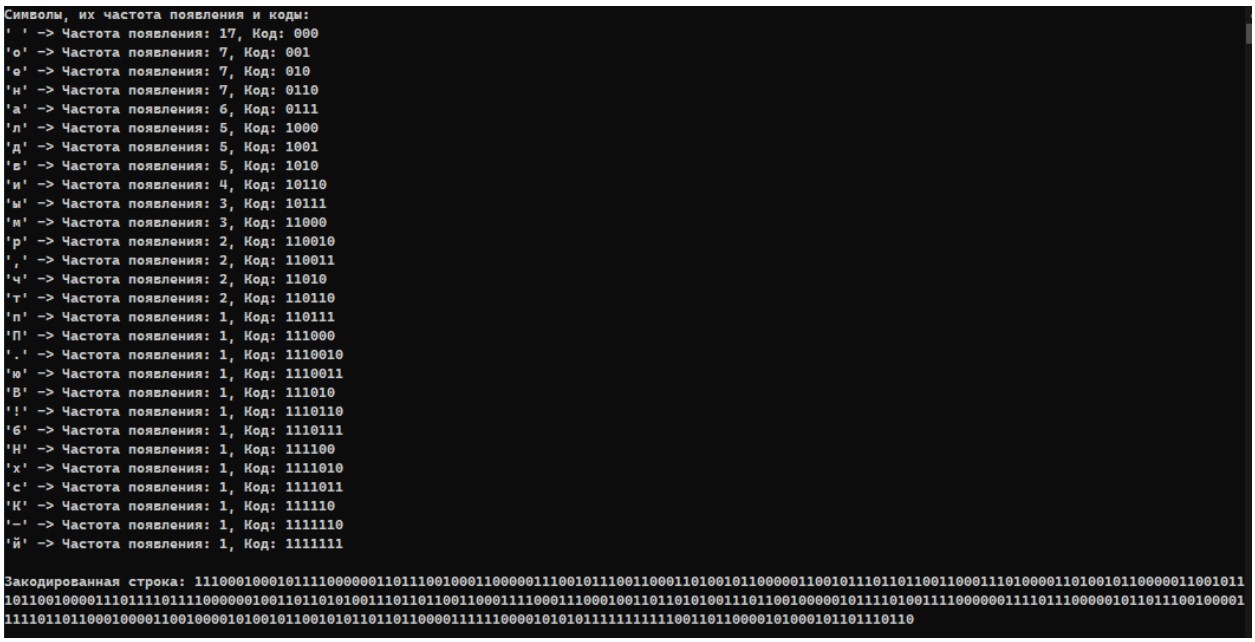


Рисунок 4 — Вывод программы для строки согласно варианту

4 ЗАДАНИЕ 1.2

4.1 Алгоритм решения

Метод Лемпеля-Зива (LZ77) — это алгоритм сжатия данных, который заменяет повторяющиеся фрагменты на пары (смещение, длина совпадения, следующий символ после совпадения). Для каждого символа алгоритм ищет ранее встречавшийся фрагмент в скользящем окне и заменяет его на ссылку на этот фрагмент. Если фрагмент не найден, символ записывается как есть. LZ77 эффективен для текстов с повторяющимися подстроками, но требует памяти для хранения буфера.

Принцип восстановления сжатой информации в LZ77 заключается в следующем: сжатые данные содержат пары (позиция, длина) и символ. При восстановлении, для каждой пары (позиция, длина), алгоритм копирует из уже восстановленных данных фрагмент длины, указанной в паре, начиная с позиции, и добавляет к этому следующий символ.

Для варианта №17 кодировка фразы состоящей из двух символьного алфавита (0, 1) представлена в Таблице 2.

Шаг	Скользящее окно		Совпадающая фраза	Закодированные данные		
	Окно поиска (5 символов)	Буфер (5 символов)		offset	length	nextChar
1	-	00010	-	0	0	0
2	0	00100	0	1	1	0
3	000	10000	-	0	0	1
4	0001	00001	000	4	3	0
5	10000	10101	10	5	2	1
6	00101	01001	010	4	3	0
7	10100	1101	1	5	1	1
8	100111	01	01	3	2	-

Таблица 2 - Сжатие методом Лемпеля –Зива LZ77

Сжатая фраза:

(0, 0, 0) (1, 1, 0) (0, 0, 1) (4, 3, 0) (5, 2, 1) (4, 3, 0) (5, 1, 1) (3, 2, -)

4.2 Реализация задачи и тестирование

```cpp
struct LZ77Triplet {
    int offset;
    int length;
    char nextChar;

    LZ77Triplet(int off, int len, char next) : offset(off), length(len), nextChar(next) {}
};

vector<LZ77Triplet> encodeLZ77(const string& input, int searchBufferSize, int lookaheadBufferSize) {
    vector<LZ77Triplet> encodedData;
    size_t i = 0;

    while (i < input.size()) {
        int matchOffset = 0;
        int matchLength = 0;
        char nextChar = '\0';

        int searchBufferStart = max(0, static_cast<int>(i) - searchBufferSize);
        string searchBuffer = input.substr(searchBufferStart, i - searchBufferStart);
        string lookaheadBuffer = input.substr(i, lookaheadBufferSize);

        for (size_t j = 0; j < searchBuffer.size(); ++j) {
            int length = 0;

            while (length < lookaheadBuffer.size() && searchBuffer[j + length] == lookaheadBuffer[length]) {
                ++length;

                if (j + length >= searchBuffer.size())
                    break;
            }

            if (length > matchLength) {
                matchLength = length;
                matchOffset = searchBuffer.size() - j;
            }
        }

        nextChar = (matchLength < lookaheadBuffer.size()) ? lookaheadBuffer[matchLength] : '\0';

        encodedData.emplace_back(matchOffset, matchLength, nextChar);

        i += matchLength + 1;
    }

    return encodedData;
}
```

Рисунок 5 — Структура триплета и функция сжатия методом Лемпеля –Зива LZ77

```cpp
int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    string input = "00010000101010011101";
    int searchBufferSize = 5;
    int lookaheadBufferSize = 5;

    vector<LZ77Triplet> compressed = encodeLZ77(input, searchBufferSize, lookaheadBufferSize);

    cout << "Результат сжатия информации (offset, length, nextChar):" << endl;
    for (const auto& triplet : compressed) {
        cout << "(" << triplet.offset << ", " << triplet.length << ", "
            << (triplet.nextChar ? triplet.nextChar : '-') << ") ";
    }


    return 0;
}
```

Рисунок 6 — Основная функция программы

```
Результат сжатия информации (offset, length, nextChar):
(0, 0, 0) (1, 1, 0) (0, 0, 1) (4, 3, 0) (5, 2, 1) (4, 3, 0) (5, 1, 1) (3, 2, -)
C:\Users\aleks\source\repos\ConsoleApplication23\x64\Debug\ConsoleApplication23.exe (п
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис
и остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно…
```

Рисунок 7 - Вывод программы для строки согласно варианту

5 ЗАДАНИЕ 1.3

5.1 Алгоритм решения

LZ78 — это один из алгоритмов сжатия данных. Он работает на основе построения словаря, в котором хранятся уникальные подстроки данных, встречающиеся в процессе их обработки.

Алгоритм работы LZ78 можно описать следующим образом. На этапе кодирования входная строка читается посимвольно. На каждом шаге алгоритм ищет самую длинную подстроку, которая уже есть в словаре. Если такая подстрока найдена, алгоритм добавляет в словарь новую запись, состоящую из индекса этой подстроки в словаре, дополненного следующим символом из входной строки. Если же подстрока отсутствует в словаре, она сразу добавляется в словарь, а для её кодирования записывается пара с индексом 0 и первым символом подстроки. Таким образом, LZ78 кодирует данные, динамически формируя словарь без необходимости полного анализа входного потока заранее.

Процесс декодирования является обратным. Декодер начинает с пустого словаря, идентичного тому, что создавался при сжатии. Для каждой пары (индекс, символ) из сжатых данных декодер извлекает подстроку из словаря по заданному индексу и добавляет к ней символ. Затем эта новая строка добавляется в словарь, а результат декодирования объединяется с ранее восстановленными данными.

Таким образом, алгоритм LZ78 эффективен для сжатия данных, содержащих повторяющиеся структуры, поскольку он позволяет сократить размер представления за счёт построения и использования компактного словаря.

Для варианта №17 кодировка фразы представлена в Таблице 3.

Словарь	Считываемое содержимое	Код
-	w	(o, w)
w = 1	e	(o, e)
w = 1, e = 2	b	(o, b)
w = 1, e = 2, b = 3	we	(1, e)
w = 1, e = 2, b = 3 we = 4	r	(0, r)
w = 1, e = 2, b = 3, r = 5 we = 4	bw	(3, w)
w = 1, e = 2, b = 3, r = 5 we = 4, bw = 6	eb	(2, b)
w = 1, e = 2, b = 3, r = 5 we = 4, bw = 6, eb = 7	er	(2, r)
w = 1, e = 2, b = 3, r = 5 we = 4, bw = 6, eb = 7, er = 8	web	(4, b)
w = 1, e = 2, b = 3, r = 5 we = 4, bw = 6, eb = 7, er = 8 web = 9	erw	(8, w)
w = 1, e = 2, b = 3, r = 5 we = 4, bw = 6, eb = 7, er = 8 web = 9, erw = 10		(7, -)

Таблица 3 - Сжатие методом LZ78

Сжатая фраза:

(0, 'w') (0, 'e') (0, 'b') (1, 'e') (0, 'r') (3, 'w') (2, 'b') (2, 'r') (4, 'b') (8, 'w') (7, '-')

5.2 Реализация задачи и тестирование

```
struct LZ78Pair {
    int index;
    char nextChar;

    LZ78Pair (int i, char next) : index(i), nextChar(next) {}
};

vector<LZ78Pair> encodeLZ78(const string& input) {
    unordered_map<string, int> dictionary;
    vector<LZ78Pair> compressedData;

    string current = "";
    int dictIndex = 1;

    for (char ch : input) {
        current += ch;

        if (dictionary.find(current) == dictionary.end()) {
            int prefixIndex = (current.size() > 1) ? dictionary[current.substr(0, current.size() - 1)] : 0;
            compressedData.emplace_back(prefixIndex, ch);

            dictionary[current] = dictIndex++;

            current = "";
        }
    }

    if (!current.empty()) {
        int prefixIndex = dictionary[current];
        compressedData.emplace_back(prefixIndex, '-');
    }

    return compressedData;
}
```

Рисунок 8 - Структура пары и функция сжатия методом LZ78

```
int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    string input = "webwerbweberweberweb";

    vector<LZ78Pair> encodedData = encodeLZ78(input);

    cout << "Результат сжатия информации: ";
    for (const auto& pair : encodedData) {
        cout << "(" << pair.index << ", '" << pair.nextChar << "') ";
    }
    cout << endl;

    return 0;
}
```

Рисунок 9 - Основная функция программы

```
Результат сжатия информации: (0, 'w') (0, 'e') (0, 'b') (1, 'e') (0, 'r') (3, 'w') (2, 'b') (2, 'r') (4, 'b') (8, 'w') (7, '-')

C:\Users\aleks\source\repos\ConsoleApplication24\x64\Debug\ConsoleApplication24.exe (процесс 23564) завершил работу с кодом 0 (0x0)
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически
и остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно…
```

Рисунок 10 - Вывод программы для строки согласно варианту

6 ЗАДАНИЕ 2.1

6.1 Требования к выполнению задания

Разработать алгоритм и реализовать программу сжатия текста алгоритмом Шеннона – Фано. Разработать алгоритм и программу восстановления сжатого текста. Выполнить тестирование программы на текстовом файле. Определить процент сжатия.

6.2 Реализация задачи и тестирование

Некоторые функции для реализации задания были взяты из задания 1.1, и их функциональность осталась без изменений. Однако в процессе работы были внесены следующие изменения:

```
string encodeShannonFano(const string& originalText, vector<Symbol>& symbols) {
    unordered_map<char, int> frequencyMap;
    string result;
    for (char ch : originalText)
        frequencyMap[ch]++;

    for (const auto& pair : frequencyMap)
        symbols.push_back({ pair.first, pair.second, "" });

    sort(symbols.begin(), symbols.end(), [](const Symbol& a, const Symbol& b) {return a.frequency > b.frequency; });

    shannonFano(symbols, 0, symbols.size() - 1);

    unordered_map<char, string> symbolMap;
    for (const Symbol& symbol : symbols) {
        symbolMap[symbol.character] = symbol.code;
    }

    for (char ch : originalText) {
        for (const Symbol& symbol : symbols) {
            if (symbol.character == ch) {
                result += symbol.code;
            }
        }
    }

    return result;
}
```

Рисунок 11 — Переработанная функция сжатия текстам

```
string decodeShannonFano(const string& encodedText, vector<Symbol>& symbols) {
    unordered_map<string, char> reverseCodeMap;
    for (const Symbol& symbol : symbols) {
        reverseCodeMap[symbol.code] = symbol.character;
    }

    string decodedText;
    string currentCode;

    for (char bit : encodedText) {
        currentCode += bit;

        if (reverseCodeMap.find(currentCode) != reverseCodeMap.end()) {
            decodedText += reverseCodeMap[currentCode];
            currentCode.clear();
        }
    }

    return decodedText;
}
```

Рисунок 12 — Функция восстановления сжатого файла

```
void printPrefixTree(vector<Symbol>& symbols) {
    cout << "Символы, их частота появления и коды:\n";
    char temp;
    for (const Symbol& symbol : symbols) {
        string temp;
        (symbol.character == ' ') ? temp = "Пробел" : ((symbol.character == '\n') ? temp = "Перенос строки" : temp = "");
        if (temp.empty()) {
            cout << "'" << symbol.character << "'" << " -> Частота появления: " << symbol.frequency << ", Код: " << symbol.code << endl;
        }
        else {
            cout << temp << " -> Частота появления: " << symbol.frequency << ", Код: " << symbol.code << endl;
        }
    }
    cout << endl;
}

string ParseFile(const string& name) {
    ifstream inputFile(name);
    if (!inputFile) {
        cerr << "Ошибка: не удалось открыть файл input.txt" << endl;
        return "";
    }

    string text((istreambuf_iterator<char>(inputFile)), istreambuf_iterator<char>());
    inputFile.close();
    return text;
}

void calculateCompressionRatio(const string& originalText, const string& encodedText) {
    int original_size = originalText.size() * 8;
    int encoded_size = encodedText.size();
    cout << "\nКоэффицент сжатия: " << endl;
    cout << double(original_size) / double(encoded_size) << endl;
}
```

Рисунок 13 — Функции вывода префиксного дерева, чтения файла с текстом и вычисления коэффициента сжатия

```
int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    string originalText = ParseFile("input.txt");
    vector<Symbol> symbols;

    string encodedText = encodeShannonFano(originalText, symbols);

    printPrefixTree(symbols);

    calculateCompressionRatio(originalText, encodedText);

    cout << decodeShannonFano(encodedText, symbols);

    return 0;
}
```

Рисунок 14 — Основная функция программы

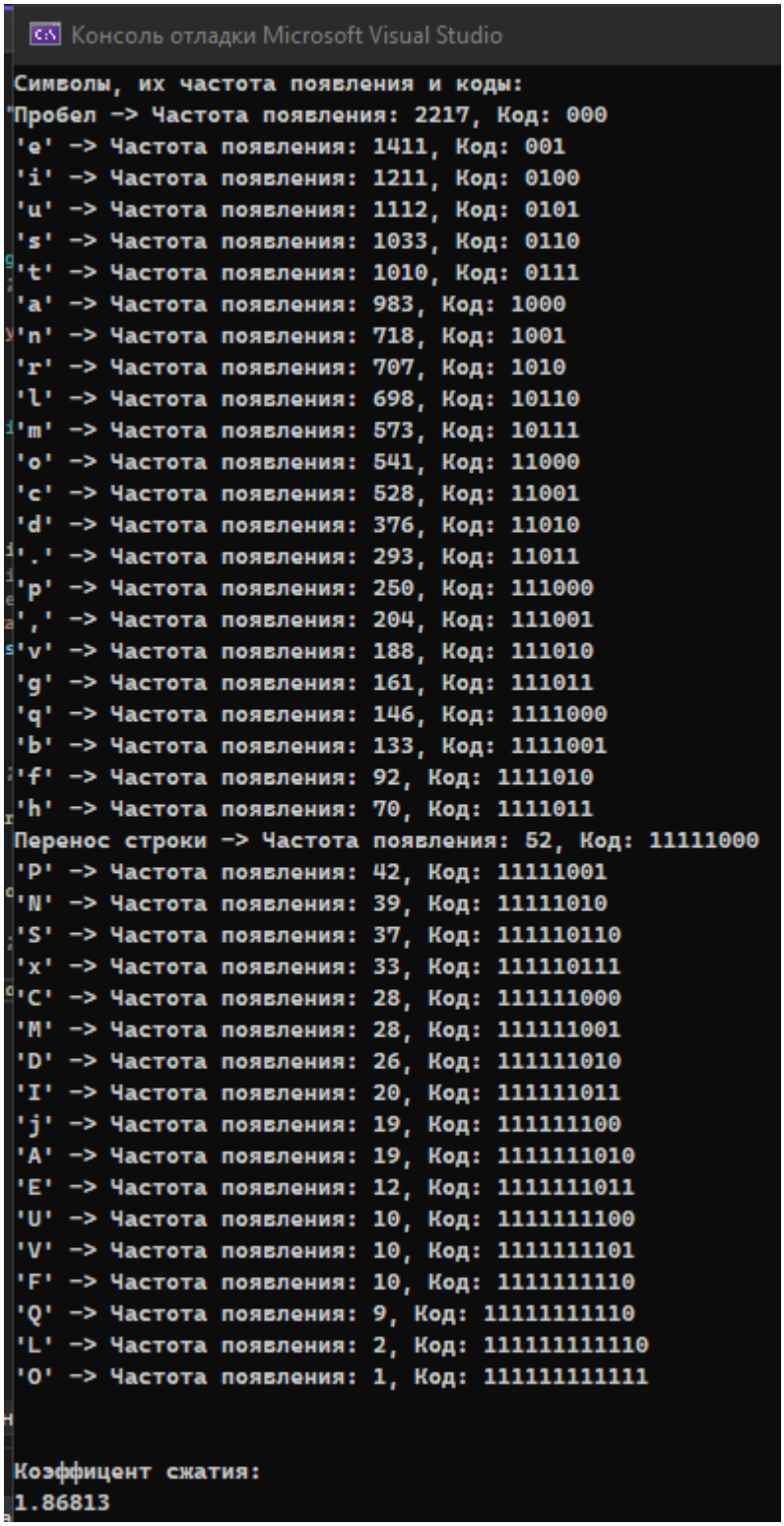


Рисунок 15 — Вывод дерева префиксов и коэффициента сжатия

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam
Suspendisse in facilisis mi. Aenean venenatis lobortis feugiat
elementum, dui orci scelerisque justo, id accumsan mauris lig
tellus, id sodales ante egestas at. Aenean sit amet felis nulla.
facilisis, faucibus metus. Quisque velit augue, elementum et

Integer tincidunt, libero id finibus porttitor, turpis augue lacinia
eu est eget leo lacinia ultrices et nec sem. Donec dapibus nib
enim felis. Vestibulum pellentesque lacus a velit viverra accu
cursus hendrerit. Proin pretium lorem eget magna volutpat, a
vel eros a bibendum.

Duis orci massa, mattis placerat lacus vitae, pulvinar congue
aliquet eget, elementum at magna. Curabitur vestibulum torto

Рисунок 16 — Содержание тестового файла

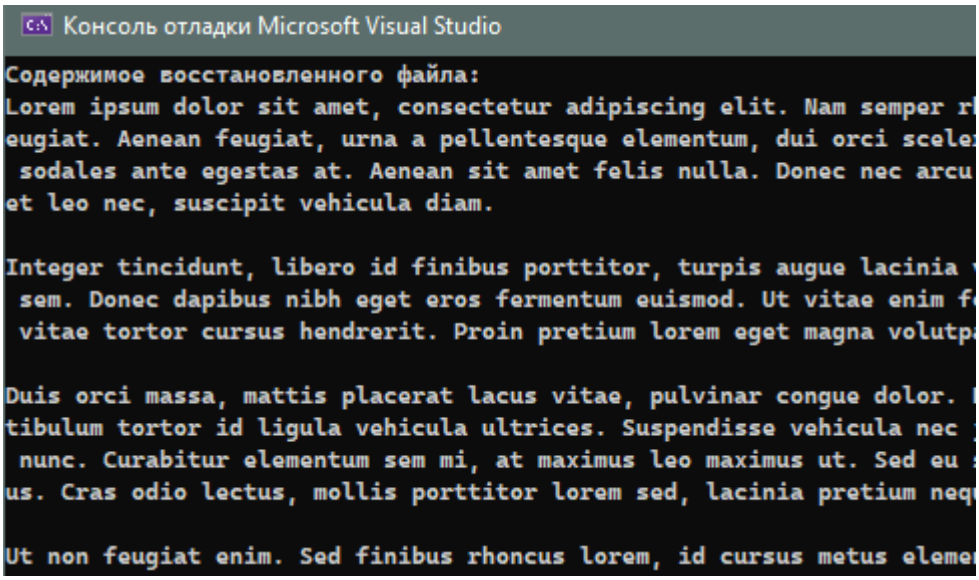


Рисунок 17 — Тестирование функции восстановления сжатого файла

6.3 Анализ коэффициента сжатия

Коэффициент сжатия методом Шеннона – Фано: 1.86813

Рисунок 18 — Вывод функции вычисления коэффициента сжатия

Коэффициент сжатия архиватором WinRAR: 3.22735

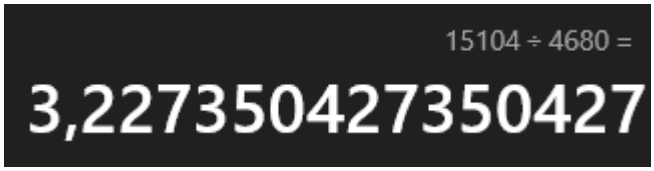


Рисунок 19 — Вычисленный коэффициент сжатия архиватором WinRAR

Из результатов сжатия видно, что полноценное приложение производит сжатие лучше, чем реализованная программа.

7 ЗАДАНИЕ 2.2

7.1 Требования к выполнению задания

Провести кодирование(сжатие) исходной строки символов «Фамилия Имя Отчество» с использованием алгоритма Хаффмана. Исходная строка символов, таким образом, определяет индивидуальный вариант задания для каждого студента.

7.2 Алгоритм решения

Алгоритм Хаффмана — это метод построения оптимального префиксного кода, который минимизирует среднюю длину кодируемого сообщения.

Работа алгоритма начинается с анализа входных данных: каждому символу присваивается его частота появления. Эти частоты используются для построения дерева Хаффмана, где каждый символ представляет собой лист, а его частота становится весом узла. На первом этапе все символы рассматриваются как отдельные узлы. Далее начинается процесс построения дерева. Алгоритм выбирает два узла с наименьшими весами и объединяет их в один новый узел, вес которого равен сумме весов объединяемых узлов. Этот новый узел становится родительским для выбранных узлов, а старые узлы удаляются из набора. Процесс продолжается до тех пор, пока не останется только один узел, представляющий корень дерева.

После построения дерева каждому символу присваивается уникальный бинарный код. Это делается путем прохождения по дереву от корня до листа: каждой ветви присваивается 0 или 1, в зависимости от направления. В итоге получается префиксный код, где ни один код не является началом другого.

Индивидуальный вариант (ФИО студента):

Попов Алексей Валерьевич

Символ	Частота	Вероятность
'е'	4	0.16
'в'	3	0.12
'п'	2	0.08
'о'	2	0.08 || ' '	2	0.08
'а'	2	0.08
'л'	2	0.08
'к'	1	0.04
'с'	1	0.04
'й'	1	0.04
'р'	1	0.04
'ь'	1	0.04
'и'	1	0.04
'ч'	1	0.04

Рисунок 20 - Таблица отсортированных частот встречаемости символов в исходной строке

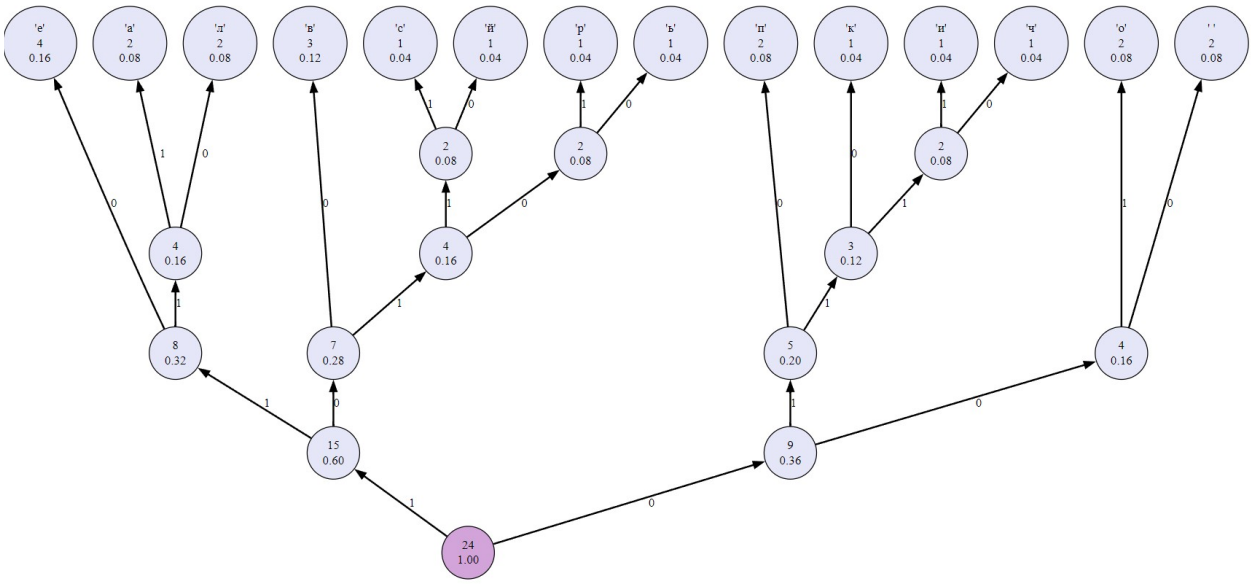


Рисунок 21 - Дерево кодирования Хаффмана

```
Сжатая строка: 010 001 010 001 100 000 1111 1110 110 0110 10111 110 10110 000 100 1111 1110 110 10101 10100 110 100 01111 01110
```

Рисунок 22 — Сжатая строка

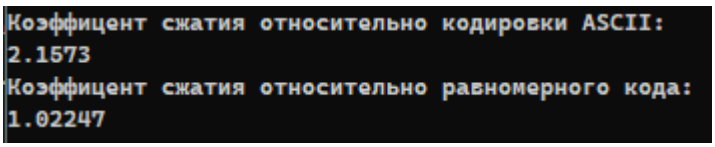
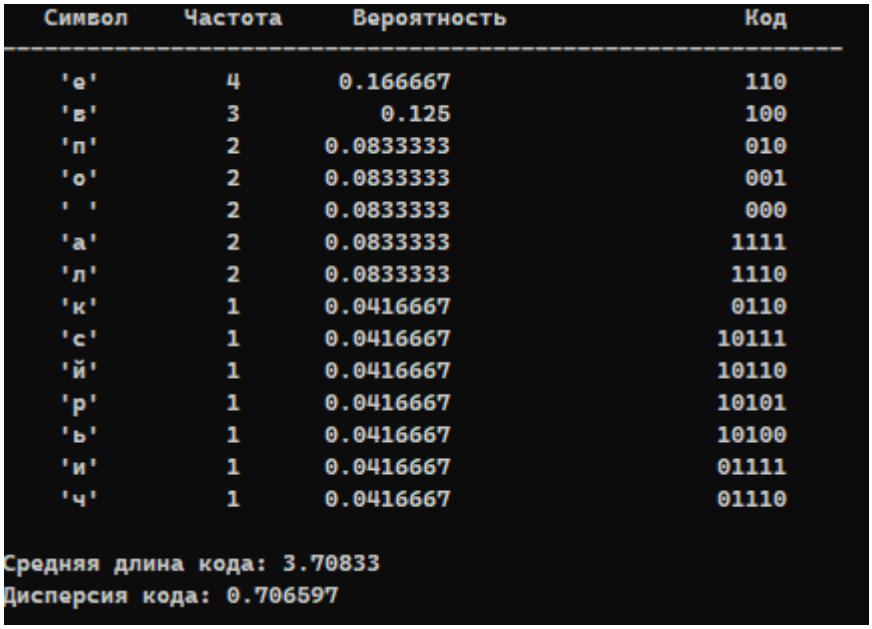


Рисунок 23 - Коэффициенты сжатия относительно кодировки ASCII и относительно равномерного кода

Коэффициент сжатия относительно ASCII больше, потому что ASCII кодирует каждый символ фиксированным количеством бит (8 бит), что избыточно для многих символов.

Коэффициент сжатия относительно равномерного кода всегда будет ближе к 1, так как равномерный код уже оптимален для равномерного распределения символов. Но если распределение сильно неравномерное, этот коэффициент тоже может значительно увеличиваться.



Символ	Частота	Вероятность	Код
'е'	4	0.166667	110
'в'	3	0.125	100
'п'	2	0.0833333	010
'о'	2	0.0833333	001 || ' '	2	0.0833333	000
'а'	2	0.0833333	1111
'л'	2	0.0833333	1110
'к'	1	0.0416667	0110
'с'	1	0.0416667	10111
'й'	1	0.0416667	10110
'р'	1	0.0416667	10101
'ь'	1	0.0416667	10100
'и'	1	0.0416667	01111
'ч'	1	0.0416667	01110

Средняя длина кода: 3.70833
Дисперсия кода: 0.706597

Рисунок 24 - Рассчет средней длины полученного кода и его дисперсии

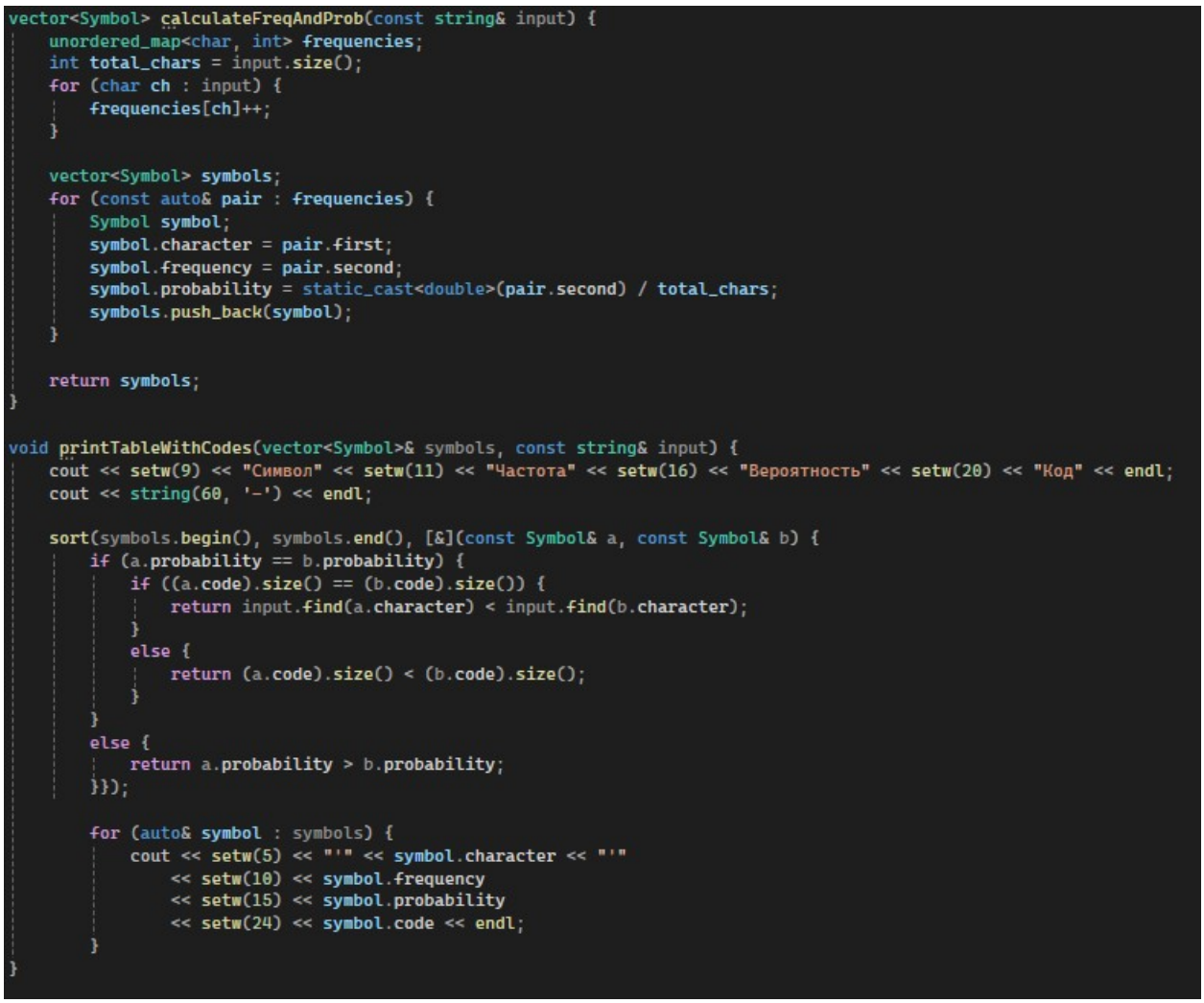
7.3 Реализация задачи

```
struct Symbol {
    char character;
    int frequency;
    double probability;
    string code;
};

struct Node {
    int pos;
    char character;
    int frequency;
    Node* left;
    Node* right;

    Node(char ch, int freq, int p) : character(ch), frequency(freq), left(nullptr), right(nullptr), pos(p) {}
};
```

Рисунок 25 — Структуры для записи символа и узла дерева

```
void assignCodes(Node* root, unordered_map<char, string>& huffmanCodes, string code = "") {
    if (!root) return;

    if (root->character != '\0') {
        huffmanCodes[root->character] = code;
    }

    assignCodes(root->left, huffmanCodes, code + "0");
    assignCodes(root->right, huffmanCodes, code + "1");
}

string compressString(const string& input, const unordered_map<char, string>& huffmanCodes) {
    string compressed;
    for (char ch : input) {
        compressed += huffmanCodes.at(ch);
    }
    return compressed;
}

string encodeHuffman(const string& input, vector<Symbol>& symbols) {
    vector<Node*> nodes;
    for (const auto& symbol : symbols) {
        nodes.push_back(new Node(symbol.character, symbol.frequency, 0));
    }

    for (auto& node : nodes) {
        node->pos = input.find(node->character);
    }

    while (nodes.size() > 1) {
        sort(nodes.begin(), nodes.end(), [&](Node* a, Node* b) {
            if (a->frequency == b->frequency) {
                return a->pos > b->pos;
            }
            else {
                return a->frequency < b->frequency;
            }});

            Node* left = nodes[0];
            Node* right = nodes[1];

            Node* combined = new Node('\0', left->frequency + right->frequency, 0);
            combined->left = left;
            combined->right = right;
            combined->pos = max(left->pos, right->pos);
            nodes.erase(nodes.begin(), nodes.begin() + 2);
            nodes.push_back(combined);
    }

    unordered_map<char, string> huffmanCodes;
    assignCodes(nodes[0], huffmanCodes);
    for (auto& symbol : symbols) {
        symbol.code = huffmanCodes.at(symbol.character);
    }
    return compressString(input, huffmanCodes);
}
```

Рисунок 26 — Функции реализации сжатия методом Хаффмана



```
vector<Symbol> calculateFreqAndProb(const string& input) {
    unordered_map<char, int> frequencies;
    int total_chars = input.size();
    for (char ch : input) {
        frequencies[ch]++;
    }

    vector<Symbol> symbols;
    for (const auto& pair : frequencies) {
        Symbol symbol;
        symbol.character = pair.first;
        symbol.frequency = pair.second;
        symbol.probability = static_cast<double>(pair.second) / total_chars;
        symbols.push_back(symbol);
    }

    return symbols;
}

void printTableWithCodes(vector<Symbol>& symbols, const string& input) {
    cout << setw(9) << "Символ" << setw(11) << "Частота" << setw(16) << "Вероятность" << setw(20) << "Код" << endl;
    cout << string(60, '-') << endl;

    sort(symbols.begin(), symbols.end(), [&](const Symbol& a, const Symbol& b) {
        if (a.probability == b.probability) {
            if ((a.code).size() == (b.code).size()) {
                return input.find(a.character) < input.find(b.character);
            }
            else {
                return (a.code).size() < (b.code).size();
            }
        }
        else {
            return a.probability > b.probability;
        }});

        for (auto& symbol : symbols) {
            cout << setw(5) << "'" << symbol.character << "'"
                << setw(10) << symbol.frequency
                << setw(15) << symbol.probability
                << setw(24) << symbol.code << endl;
        }
}
```

Рисунок 27 — Функция рассчета частот встречаемости символов и их вероятности появления, функция вывода таблицы кодов

```cpp
void calculateCompressionRation(const string& input, vector<Symbol>& symbols) {
    int ASCII_length = input.size() * 8;
    set<char> uniqueChars;
    for (char c : input)
        uniqueChars.insert(c);
    int Uniform_length = input.size() * (log(uniqueChars.size()) / log(2));
    int Encoded_length = encodeHuffman(input, symbols).size();
    cout << "Коэффицент сжатия относительно кодировки ASCII:\n";
    cout << double(ASCII_length) / Encoded_length << endl;
    cout << "Коэффицент сжатия относительно равномерного кода:\n";
    cout << double(Uniform_length) / Encoded_length << endl;
}

void calculateAvgLengthAndDispersion(vector<Symbol>& symbols) {
    double averageLength = 0;
    double dispersion = 0;
    for (auto& symbol : symbols) {
        averageLength += symbol.probability * (symbol.code).size();
    }
    for (auto& symbol : symbols) {
        dispersion += symbol.probability * pow((symbol.code).size() - averageLength, 2);
    }
    cout << "Средняя длина кода: " << averageLength << endl << "Дисперсия кода: " << dispersion << endl;
}
```

Рисунок 28 — Функция вычисления коэффициента сжатия относительно кодировки ASCII и равномерного кода, функция вычисления средней длины кода и его дисперсии

```cpp
int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    string input = "попов алексей валерьевич";
    vector<Symbol> symbols = calculateFreqAndProb(input);

    string encodedText = encodeHuffman(input, symbols);

    printTableWithCodes(symbols, input);

    cout << endl;

    calculateAvgLengthAndDispersion(symbols);

    return 0;
}
```

Рисунок 29 — Основная функция программы

8 ЗАДАНИЕ 2.3

8.1 Требования к выполнению задания

Применить алгоритм Хаффмана для архивации данных текстового файла. Выполнить практическую оценку сложности алгоритма Хаффмана. Провести архивацию этого же файла любым архиватором. Сравнить коэффициенты сжатия разработанного алгоритма и архиватора.

8.2 Реализация задачи

Большая часть функций для реализации задания были взяты из задания 2.2, и их функциональность осталась без изменений. Однако в процессе работы были внесены следующие изменения:

```
string parseFile(const string& name) {
    ifstream inputFile(name);
    if (!inputFile) {
        cerr << "Ошибка: не удалось открыть файл input.txt" << endl;
        return "";
    }

    string text((istreambuf_iterator<char>(inputFile)), istreambuf_iterator<char>());
    inputFile.close();
    return text;
}
```

Рисунок 30 — Функция чтения текста из файла

```
void calculateCompressionRation(const string& input, const string& output) {
    cout << "Коэффициент сжатия: " << double(input.size() * 8) / output.size();
}
```

Рисунок 31 — Функция вывода коэффициента сжатия алгоритмом Хаффмана

8.3 Анализ коэффициента сжатия

Символ	Частота	Вероятность	Код
' '	2217	0.147289	110
'e'	1411	0.0937417	001
'i'	1211	0.0804544	1111
'u'	1112	0.0738772	1011
's'	1033	0.0686288	1001
't'	1010	0.0671007	1000
'a'	983	0.0653069	0111
'n'	718	0.0477013	0100
'r'	707	0.0469705	0001
'l'	698	0.0463726	0000
'm'	573	0.038068	11100
'o'	541	0.0359421	10101
'c'	528	0.0350784	01101
'd'	376	0.0249801	01011
'.'	293	0.0194659	111010
'p'	250	0.0166091	101000
','	204	0.013553	011000
'v'	188	0.01249	010101
'g'	161	0.0106963	1110111
'q'	146	0.00969971	1010011
'b'	133	0.00883604	1010010
'f'	92	0.00611214	0101001
'h'	70	0.00465054	11101100
'\n'	52	0.00345469	01100100
'P'	42	0.00279033	01010001
'N'	39	0.00259102	111011011
'S'	37	0.00245815	011001111
'x'	33	0.0021924	011001110
'C'	28	0.00186022	011001100
'M'	28	0.00186022	011001011
'D'	26	0.00172735	011001010
'I'	20	0.00132873	010100000
'A'	19	0.00126229	1110110100
'j'	19	0.00126229	0110011011
'E'	12	0.000797236	0110011010
'U'	10	0.000664364	0101000010
'V'	10	0.000664364	11101101011
'F'	10	0.000664364	11101101010
'Q'	9	0.000597927	01010000111
'L'	2	0.000132873	010100001101
'O'	1	6.64364e-05	010100001100

Коэффициент сжатия: 1.87092

Рисунок 32 — Вывод таблицы кодов для текстового файла и коэффициента сжатия

Коэффициент сжатия алгоритмом Хаффмана: 1.87092

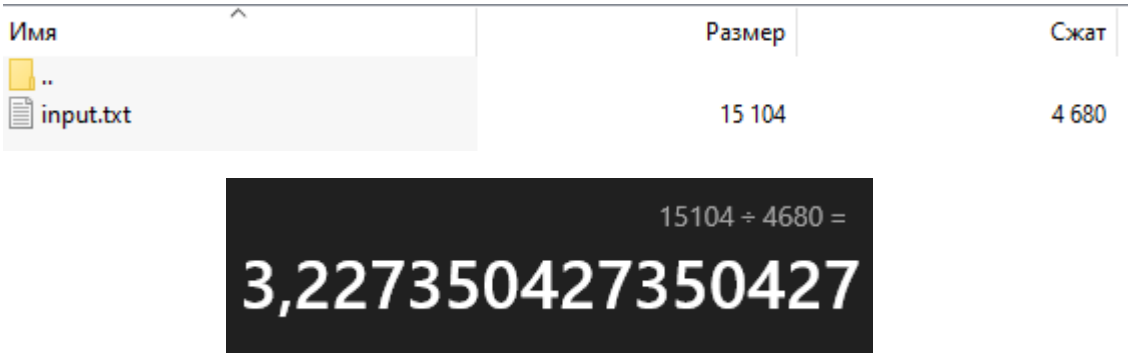


Рисунок 33 — Рассчет коэффициента сжатия архиватором WinRAR

Коэффициент сжатия архиватором WinRAR: 3.22735

Из результатов сжатия видно, что полноценное приложение производит сжатие лучше, чем реализованная программа.

9 ВЫВОД

В рамках работы были изучены и реализованы различные алгоритмы сжатия данных, включая LZ77, LZ78, Шеннона-Фано и Хаффмана. Выполнен расчет коэффициентов сжатия, построены таблицы частот и коды для префиксных деревьев, а также реализованы методы восстановления данных. Алгоритмы были протестированы на реальных данных, включая текстовые файлы, что позволило оценить их эффективность в зависимости от структуры данных. По итогам работы определены сильные и слабые стороны каждого метода, а также их применимость для разных типов данных.