МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания № 8, часть 1

Тема:

«Алгоритмы кодирования и сжатия данных»

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Лисовский И.В

Группа: ИКБО-21-23

Москва – 2024

СОДЕРЖАНИЕ

1 ЦЕЛЬ

Целью практической работы по теме "Алгоритмы кодирования и сжатия данных" может быть изучение и применение различных методов сжатия данных для повышения эффективности хранения и передачи информации. Студенты должны будут реализовать и проанализировать алгоритмы, такие как Хаффман или RLE, чтобы понять их преимущества и области применения.

2 ЗАДАНИЕ

2.1 Формулировка задачи

Задание 1 Исследование алгоритмов сжатия на примерах

- 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.

- 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 другого 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.

1. 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.

2. 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.

Вариант №19:

Закодировать фразу методами Шеннона–Фано	**Сжатие данных по методу Лемпеля–Зива LZ77 Используя двухсимвольный алфавит (0, 1) закодировать следующую фразу:**	**Закодировать следующую фразу, используя код LZ78**
Перводан, другодан, На колоде барабан; Свистель, коростель, Пятерка, шестерка, утюг.	00010000101010011101	comconcomconacom

2.2 Задание 1(1)

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

struct literal
{
	char liter;
	int count = 0;
	string code = "";
	literal(char liter, int count)
	{
		this->liter = liter;
		this->count = count;
```

```
	}
};

bool isFound(vector <literal> ex, char toFound)
{
	for (int i = 0; i < ex.size(); i++)
	{
		if (ex[i].liter == toFound)
		{
			return 1;
		}
	}
	return 0;
}

vector <literal> alph(string text)
{
	vector<literal> iter;
	for (int i = 0; i < text.length(); i++)
	{
		if (!isFound(iter, text[i]))	//если новая буква в алфавите
		{
			literal newLit(text[i], 1);
			iter.push_back(newLit);
		}
		else				//если уже есть подобная буква,
увеличиваем счетчик ее повторов
		{
			for (int j = 0; j < iter.size(); j++)
			{
				if (iter[j].liter == text[i])
				{
					iter[j].count++;
				}
			}
		}
	}
	return iter;
}

void sortingVector(vector <literal>& alphabet)
{
	int n = alphabet.size();
```

```
	for (int i = 0; i < n - 1; i++) {
		// Переменная для оптимизации (отслеживание обменов)
		bool swapped = false;
		// Последние i элементов уже стоят на своих местах
		for (int j = 0; j < n - i - 1; j++) {
			if (alphabet[j].count < alphabet[j + 1].count) {
				// Обмен элементов
				literal temp = alphabet[j];
				alphabet[j] = alphabet[j + 1];
				alphabet[j + 1] = temp;
				swapped = true;
			}
		}
		// Если не было обменов во внутреннем цикле, массив уже отсортирован
		if (!swapped)
			break;
	}
}

int findSplitPoint(int left, int right, const std::vector<literal>& symbols) {
	// Вычисляем общую сумму частот в текущей группе символов
	double total = 0.0;
	for (int i = left; i <= right; ++i) {
		total += symbols[i].count;
	}

	// Цель - найти точку, где накопленная сумма приближается к половине общей суммы
	double halfTotal = total / 2.0;
	double accum = 0.0;
	int split = left;

	for (int i = left; i <= right; ++i) {
		accum += symbols[i].count;
		if (accum >= halfTotal) {
			// Проверяем, было ли предыдущее значение накопленной суммы ближе к половине
			if (i == left) {
				// Если первый же символ превосходит половину, то точка разделения - первый символ
				split = i;
			}
			else {
```

```
                        double diff1 = std::abs(accum - halfTotal);
                        double diff2 = std::abs((accum - symbols[i].count) -
halfTotal);
                        if (diff2 < diff1) {
                                split = i - 1;
                        }
                        else {
                                split = i;
                        }
                }
                break;
        }
    }

    return split;
}


void shannonFano(int left, int right, std::vector<literal>& symbols) {
    if (left >= right)
        return;

    // Ищем точку разделения
    int split = findSplitPoint(left, right, symbols);

    // Присваиваем биты '0' и '1'
    for (int i = left; i <= split; ++i)
        symbols[i].code += "0";
    for (int i = split + 1; i <= right; ++i)
        symbols[i].code += "1";

    // Рекурсивные вызовы для подгрупп
    shannonFano(left, split, symbols);
    shannonFano(split + 1, right, symbols);
}

void codeText(string text)
{
    vector<literal>alphabet = alph(text);
    sortingVector(alphabet);
    cout << "alphabet: \n";
    cout << "----------------------------\n";
    for (int i = 0; i < alphabet.size(); i++)
```

```
	{
		cout << alphabet[i].liter << ": " << alphabet[i].count << "\n";
	}
	cout << "-----------------------------\n";
	cout << "Размер алфавита: " << alphabet.size() << "\n";
	shannonFano(0, alphabet.size() - 1, alphabet);
	int shrinkSize = 0;
	cout << "\n--------Таблица сжатия--------\n";
	for (int i = 0; i < alphabet.size(); i++)
	{
		shrinkSize += alphabet[i].code.length() * alphabet[i].count;
		cout << alphabet[i].liter << ": " << alphabet[i].code << "\n";
	}
	cout << "\nДо сжатия вес в битах: " << text.length() * 8 << "  (в байтах): "
<< text.length() << "\n";
	cout << "После сжатия вес в битах: " << shrinkSize << " (в байтах): " <<
shrinkSize / 8 << "\n";
	cout << "\n--------Закодированный текст--------\n";
	for (int i = 0; i < text.length(); i++)
	{
		if (i % 20 == 0 && i != 0)
		{
			cout << "<<\n";
		}
		for (int j = 0; j < alphabet.size(); j++)
		{
			if (text[i] == alphabet[j].liter)
			{
				cout << alphabet[j].code << " ";
			}
		}
	}
}


int main()
{
	setlocale(0, "");
	string text = "Перводан, другодан,\nНа колоде барабан;\nСвистель,
коростель,\nПятерка, шестерка, утюг.";
	codeText(text);
}
```

Таблица оформления закодированной фразы «Перводан, другодан, На колоде барабан; Свистель, коростель, Пятерка, шестерка, утюг.» методом Шеннона-Фано

Символ	Кол-во	1-я цифра	2-я цифра	3-я цифра	4-я цифра	5-я цифра	6-я	7-я	Код	Кол-во бит
а	8	0	0	0					000	24
е	7	0	0	1	0				0010	28
р	6	0	0	1	1				0011	24
о	6	0	1	0	0				0100	24
,	6	0	1	0	1				0101	24
пробел	6	0	1	1					011	18
т	5	1	0	0	0				1000	20
д	4	1	0	0	1	0			10010	20
к	4	1	0	0	1	1			10011	20
н	3	1	0	1	0	0			10100	15
л	3	1	1	1	0	1			11101	15
с	3	1	1	1	1				1111	12
П	2	1	1	0	0	0	0		110000	12
Символ	Кол-во	1-я цифра	2-я цифра	3-я цифра	4-я цифра	5-я цифра	6-я	7-я	Код	Кол-во бит
в	2	1	1	0	0	0	1		110001	12
у	2	1	1	0	0	1			11001	10
г	2	1	1	0	1	0			11010	10
б	2	1	1	0	1	1			11011	10
ь	2	1	1	1	0	0	0		111000	12
Н	1	1	1	1	0	0	1		111001	6
;	1	1	1	1	0	1	0		111010	6
С	1	1	1	1	0	1	1		111011	6
и	1	1	1	1	1	0	0	0	1111000	7
я	1	1	1	1	1	0	0	1	1111001	7
ш	1	1	1	1	1	0	1		111101	6
ю	1	1	1	1	1	1	0		111110	6
.	1	1	1	1	1	1	1		111111	6

До сжатия, вес: 648 бит – 81 байт

После сжатия, все: 360 бит – 45 байт

2.3 Задание 1(2)

Исходный текст	0.00.10.000.101.01.001.101
LZ-код	0.10.010.100.0111.0001.0001.101
R	2 3
Вводимые коды	- 10 11 100 101 110 111

2.4 Задание 1(3)

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;
int main() {
    string input = "comconcomconacom";
    unordered_map<string, int> dictionary;
    vector<pair<int, char>> output;
    setlocale(0, "");
    int dictSize = 1; // Начинаем индекс словаря с 1
    size_t i = 0;

    while (i < input.size()) {
        string currentSubstring;
        int index = 0;

        // Находим максимально длинную подстроку, которая уже есть в словаре
        while (i < input.size() && dictionary.find(currentSubstring + input[i]) !=
dictionary.end()) {
            currentSubstring += input[i];
            index = dictionary[currentSubstring];
            ++i;
        }
```

```
        // Выводим считываемую подстроку
        cout << "Считываемая подстрока: \"" << currentSubstring << "\"\n";

        // Если достигли конца строки
        if (i == input.size()) {
            output.emplace_back(index, '\0'); // Используем '\0' для обозначения
конца
            // Выводим текущее состояние словаря
            cout << "Текущий словарь:\n";
            for (const auto& entry : dictionary) {
                cout << "  \"" << entry.first << "\": " << entry.second << "\n";
            }
            cout << "----------------------------------------\n";
            break;
        }

        // Добавляем новую подстроку в словарь
        currentSubstring += input[i];
        dictionary[currentSubstring] = dictSize++;
        char nextChar = input[i];
        ++i;

        // Записываем пару (индекс, символ)
        output.emplace_back(index, nextChar);

        // Выводим текущее состояние словаря
        cout << "Текущий словарь:\n";
        for (const auto& entry : dictionary) {
            cout << "  \"" << entry.first << "\": " << entry.second << "\n";
        }
        cout << "----------------------------------------\n";
    }

    // Выводим результаты
    cout << "Сжатые данные (пары индекс-символ):\n";
    for (const auto& pair : output) {
        cout << "(" << pair.first << ", " << (pair.second == '\0' ? "EOF" :
string(1, pair.second)) << ")\n";
    }

    return 0;
}
```

Выходная таблица после сжатия методом LZ78:

Словарь	Считываемое содержимое	Код
	c	<0,c>
c=1	**o**	<0,o>
c=1, o=2	**m**	<0,m>
c=1, o=2,m=3	**co**	<1,o>
c=1, o=2,m=3, co=4	**n**	<0,n>
c=1, o=2,m=3, co=4, n=5	**com**	<4,m>
c=1, o=2,m=3, co=4, n=5, com=6	**con**	<4,n>
c=1, o=2,m=3, co=4, n=5, com=6, con=7	**a**	<0,a>
c=1, o=2,m=3, co=4, n=5, com=6, con=7, a=8		<6,EOF>

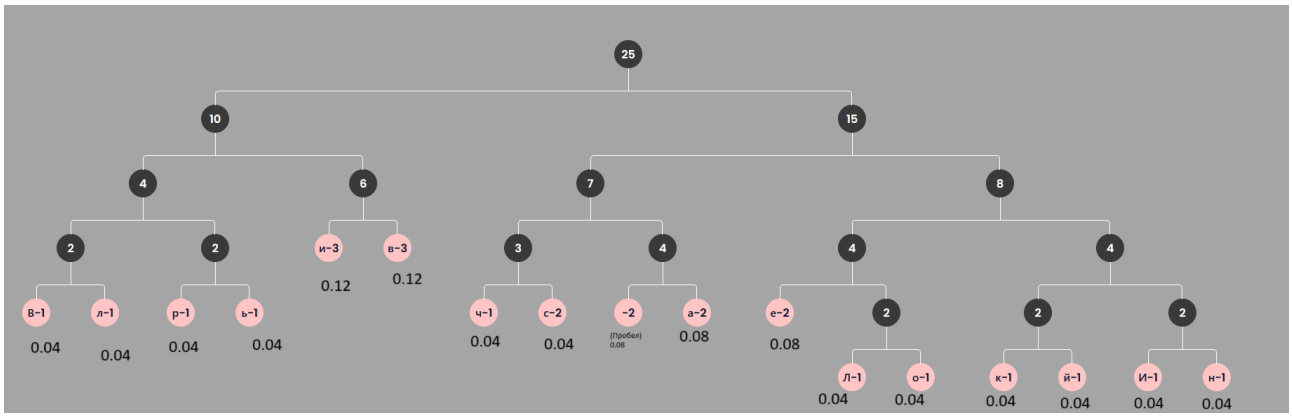
2.5 Задание 2

Исходная строка – ФИО : Лисовский Иван Валерьевич

Алфавит	Л	и	с	о	в	к	й	И	а
Кол-во	1	3	2	1	3	1	1	1	2
Вероятность									
Алфавит	н	В	л	е	р	ь	ч	< >	
Кол-во	1	1	1	2	1	1	1	2	
Вероятность									

Алфавит	и	в	с	а	е	< >	й	И	к
Кол-во	3	3	2	2	2	2	1	1	1
Вероятность									
Алфавит	н	В	л	о	р	ь	ч	Л	
Кол-во	1	1	1	1	1	1	1	1	
Вероятность									

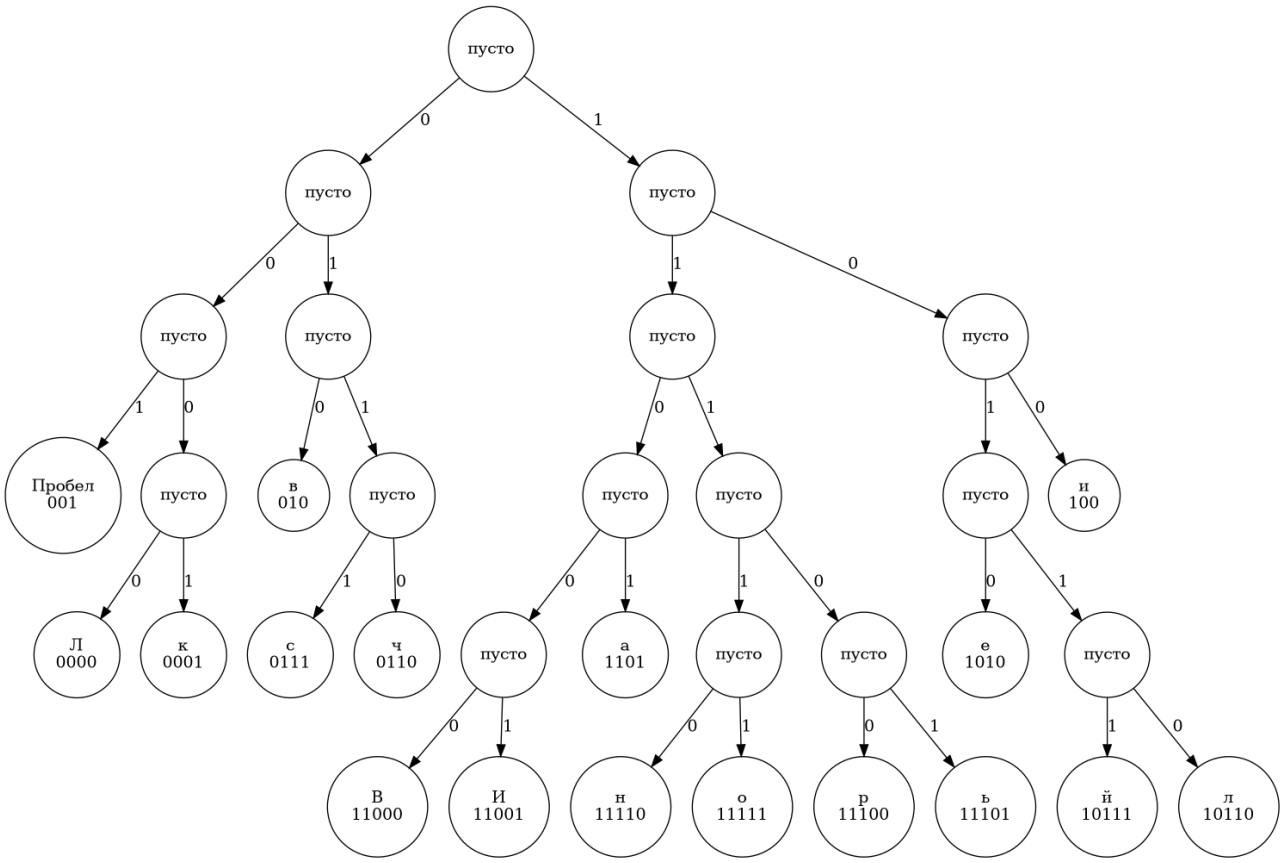
Дерево Хаффмана :



Определим коды символов:

Л и с о в с к и й И в а н В а л е р ь е в и ч

0000 100 0111 11111 010 0111 0001 100 10111 001 11001 010 1101 11110 001 11000 1101 10110 1010 11100 11101 1010 010 100 0110



Исходный размер данных (N – количество символов в строке, b – исходное количество бит для кодировки одного символа, т.к используем w_char – широкий формат кодировки символов):

$S = N * b = 25 * 16 = 400$ бит

Размер равномерного кода.(17 – количество уникальных символов) :

$L = \log_2(17) = 5$ (округление в большую сторону)

Общий размер при равномерном коде :

$S = N * L = 25 * 5 = 125$ бит

Размер данных, используя метод Хаффмана. (Средняя длина = 4, поэтому длина строки умножается на среднюю длину, см. ниже)

$S_{compressed} = 25 * 4$

Коэффициент сжатия относительно исходной кодировки:

K1 = 400 / 100 = 4

Коэффициент сжатия относительно равномерного кода:

K2 = 125 / 100 = 1.25

Расчет дисперсии.

Л	p_i = 1 / 25 = 0.0400, l_i = 5, p_i * l_i = 0.0400 * 5 = 0.2000
и	p_i = 3 / 25 = 0.1200, l_i = 3, p_i * l_i = 0.1200 * 3 = 0.3600
с	p_i = 2 / 25 = 0.0800, l_i = 4, p_i * l_i = 0.0800 * 4 = 0.3200
о	p_i = 1 / 25 = 0.0400, l_i = 4, p_i * l_i = 0.0400 * 4 = 0.1600
в	p_i = 3 / 25 = 0.1200, l_i = 3, p_i * l_i = 0.1200 * 3 = 0.3600
к	p_i = 1 / 25 = 0.0400, l_i = 5, p_i * l_i = 0.0400 * 5 = 0.2000
й	p_i = 1 / 25 = 0.0400, l_i = 5, p_i * l_i = 0.0400 * 5 = 0.2000
	p_i = 2 / 25 = 0.0800, l_i = 3, p_i * l_i = 0.0800 * 3 = 0.2400
И	p_i = 1 / 25 = 0.0400, l_i = 5, p_i * l_i = 0.0400 * 5 = 0.2000
а	p_i = 2 / 25 = 0.0800, l_i = 4, p_i * l_i = 0.0800 * 4 = 0.3200
н	p_i = 1 / 25 = 0.0400, l_i = 5, p_i * l_i = 0.0400 * 5 = 0.2000
В	p_i = 1 / 25 = 0.0400, l_i = 5, p_i * l_i = 0.0400 * 5 = 0.2000
л	p_i = 1 / 25 = 0.0400, l_i = 4, p_i * l_i = 0.0400 * 4 = 0.1600
е	p_i = 2 / 25 = 0.0800, l_i = 4, p_i * l_i = 0.0800 * 4 = 0.3200
р	p_i = 1 / 25 = 0.0400, l_i = 5, p_i * l_i = 0.0400 * 5 = 0.2000
ь	p_i = 1 / 25 = 0.0400, l_i = 4, p_i * l_i = 0.0400 * 4 = 0.1600
ч	p_i = 1 / 25 = 0.0400, l_i = 5, p_i * l_i = 0.0400 * 5 = 0.2000

Средняя длина символа: 4 бит на символ

Дисперсия будет рассчитана по формуле: D = Σ (p_i * (l_i - L_avg)^2)

Вычисления:

Символ: 'Л', l_i = 5, (l_i - L_avg) = 1.0000, (l_i - L_avg)^2 = 1.0000, p_i * (l_i - L_avg)^2 = 0.0400 * 1.0000 = 0.0400

Символ: 'и', l_i = 3, (l_i - L_avg) = -1.0000, (l_i - L_avg)^2 = 1.0000, p_i * (l_i - L_avg)^2 = 0.1200 * 1.0000 = 0.1200

Символ: 'с', l_i = 4, (l_i - L_avg) = -0.0000, (l_i - L_avg)^2 = 0.0000, p_i * (l_i - L_avg)^2 = 0.0800 * 0.0000 = 0.0000

Символ: 'о', l_i = 4, (l_i - L_avg) = -0.0000, (l_i - L_avg)^2 = 0.0000, p_i * (l_i - L_avg)^2 = 0.0400 * 0.0000 = 0.0000

Символ: 'в', l_i = 3, (l_i - L_avg) = -1.0000, (l_i - L_avg)^2 = 1.0000, p_i * (l_i - L_avg)^2 = 0.1200 * 1.0000 = 0.1200

Символ: 'к', l_i = 5, (l_i - L_avg) = 1.0000, (l_i - L_avg)^2 = 1.0000, p_i * (l_i - L_avg)^2 = 0.0400 * 1.0000 = 0.0400

Символ: 'й', l_i = 5, (l_i - L_avg) = 1.0000, (l_i - L_avg)^2 = 1.0000, p_i * (l_i - L_avg)^2 = 0.0400 * 1.0000 = 0.0400

Символ: ' ', l_i = 3, (l_i - L_avg) = -1.0000, (l_i - L_avg)^2 = 1.0000, p_i * (l_i - L_avg)^2 = 0.0800 * 1.0000 = 0.0800

Символ: 'И', l_i = 5, (l_i - L_avg) = 1.0000, (l_i - L_avg)^2 = 1.0000, p_i * (l_i - L_avg)^2 = 0.0400 * 1.0000 = 0.0400

Символ: 'а', l_i = 4, (l_i - L_avg) = -0.0000, (l_i - L_avg)^2 = 0.0000, p_i * (l_i - L_avg)^2 = 0.0800 * 0.0000 = 0.0000

Символ: 'н', l_i = 5, (l_i - L_avg) = 1.0000, (l_i - L_avg)^2 = 1.0000, p_i * (l_i - L_avg)^2 = 0.0400 * 1.0000 = 0.0400

Символ: 'В', l_i = 5, (l_i - L_avg) = 1.0000, (l_i - L_avg)^2 = 1.0000, p_i * (l_i - L_avg)^2 = 0.0400 * 1.0000 = 0.0400

Символ: 'л', l_i = 4, (l_i - L_avg) = -0.0000, (l_i - L_avg)^2 = 0.0000, p_i * (l_i - L_avg)^2 = 0.0400 * 0.0000 = 0.0000

Символ: 'е', l_i = 4, (l_i - L_avg) = -0.0000, (l_i - L_avg)^2 = 0.0000, p_i * (l_i - L_avg)^2 = 0.0800 * 0.0000 = 0.0000

Символ: 'р', l_i = 5, (l_i - L_avg) = 1.0000, (l_i - L_avg)^2 = 1.0000, p_i * (l_i - L_avg)^2 = 0.0400 * 1.0000 = 0.0400

Символ: 'ь', l_i = 4, (l_i - L_avg) = -0.0000, (l_i - L_avg)^2 = 0.0000, p_i * (l_i - L_avg)^2 = 0.0400 * 0.0000 = 0.0000

Символ: 'ч', l_i = 5, (l_i - L_avg) = 1.0000, (l_i - L_avg)^2 = 1.0000, p_i * (l_i - L_avg)^2 = 0.0400 * 1.0000 = 0.0400

Дисперсия длины кода:

D = 0.6400 бит^2

Тест сжатия с помощью программы на C++

Листинг программы

```
#include <iostream>
#include <fstream>
#include <string>
#include <locale>
#include <codecvt>
#include <map>
#include <queue>
#include <vector>

using namespace std;

struct Node {
    wchar_t ch;
    int freq;
    Node* left, * right;

    Node(wchar_t c, int f) : ch(c), freq(f), left(nullptr), right(nullptr) {}
    Node(wchar_t c, int f, Node* l, Node* r) : ch(c), freq(f), left(l), right(r) {}
};

struct Compare {
    bool operator()(Node* l, Node* r) {
        return l->freq > r->freq;
    }
};

Node* buildHuffmanTree(const map<wchar_t, int>& freqMap) {
    priority_queue<Node*, vector<Node*>, Compare> pq;

    for (auto pair : freqMap) {
        Node* node = new Node(pair.first, pair.second);
        pq.push(node);
```

```
    }

    while (pq.size() > 1) {
        Node* left = pq.top(); pq.pop();
        Node* right = pq.top(); pq.pop();

        Node* parent = new Node(0, left->freq + right->freq, left, right);
        pq.push(parent);
    }

    return pq.top();
}

void generateCodes(Node* root, const wstring& prefix, map<wchar_t, wstring>& codes)
{
    if (!root)
        return;

    if (!root->left && !root->right) {
        codes[root->ch] = prefix;
    }

    generateCodes(root->left, prefix + L"0", codes);
    generateCodes(root->right, prefix + L"1", codes);
}

void deleteTree(Node* root) {
    if (!root)
        return;
    deleteTree(root->left);
    deleteTree(root->right);
    delete root;
}

vector<uint8_t> packBits(const wstring& bitString) {
    vector<uint8_t> bytes;
    uint8_t byte = 0;
    int bitsFilled = 0;

    for (wchar_t bitChar : bitString) {
        byte <<= 1;
        if (bitChar == L'1') {
            byte |= 1;
```

```
        }
        bitsFilled++;

        if (bitsFilled == 8) {
            bytes.push_back(byte);
            byte = 0;
            bitsFilled = 0;
        }
    }

    if (bitsFilled > 0) {
        byte <<= (8 - bitsFilled);
        bytes.push_back(byte);
    }

    return bytes;
}

int main() {
    locale::global(locale("ru_RU.UTF-8"));
    locale locale("ru_RU.UTF-8");

    wifstream inFile("input.txt");
    inFile.imbue(locale);

    if (!inFile) {
        wcerr << L"Не удалось открыть входной файл.\n";
        return 1;
    }

    wstring text((istreambuf_iterator<wchar_t>(inFile)),
istreambuf_iterator<wchar_t>());
    inFile.close();

    map<wchar_t, int> freq;
    for (wchar_t ch : text) {
        freq[ch]++;
    }

    Node* root = buildHuffmanTree(freq);

    map<wchar_t, wstring> codes;
    generateCodes(root, L"", codes);
```

```
    wstring encodedText;
    for (wchar_t ch : text) {
        encodedText += codes[ch];
    }

    vector<uint8_t> packedData = packBits(encodedText);

    wofstream outFile("output.huff", ios::binary);
    outFile.imbue(locale);

    if (!outFile) {
        wcerr << L"Не удалось открыть выходной файл.\n";
        return 1;
    }

    outFile << codes.size() << L'\n';

    for (const auto& pair : codes) {
        outFile << static_cast<int>(pair.first) << L' ' << pair.second << L'\n';
    }

    outFile << L"-----\n";

    outFile << encodedText.length() << L'\n';

    ofstream binaryOut("binary.bin", ios::binary);
    if (!binaryOut) {
        wcerr << L"Не удалось открыть бинарный выходной файл.\n";
        return 1;
    }
    binaryOut.write(reinterpret_cast<const char*>(packedData.data()),
packedData.size());
    binaryOut.close();

    outFile.close();

    deleteTree(root);

    return 0;
}
```

На вход поступает файл с текстом. Исходный его вес: 8018 байт. После сжатия, файл сохраняется в файле с названием binary.bin с весом 5711 байт. Для сравнения был выбран архиватор Zip. После архивации, вес = 5121 байт. Для сжатия программой методом Хаффмана, коэффициент сжатия = 1.287, в то время как для Zip = 1.362. Практическая сложность алгоритма сжатия Хаффмана составляет O(n*logn), где n — количество уникальных символов. Это связано с необходимостью построен ия приоритетной очереди и дерева, основанного на частотах символов.

3 ВЫВОД

В рамках данной практической работы были изучены и реализованы алгоритмы сжатия данных, включая методы Хаффмана, Шеннона-Фано, LZ77 и LZ78. Были проведены исследования алгоритмов сжатия, результаты которых оформлены в виде таблицы с коэффициентами сжатия. Детально описан процесс восстановления сжатого текста, что помогло понять внутренние механизмы этих алгоритмов. Реализованы программы для различных методов сжатия:

1. **Метод Шеннона-Фано**:
    - Описан процесс построения префиксного дерева, а также алгоритмы кодирования и декодирования.
    - Программа протестирована, рассчитан коэффициент сжатия и проведено сравнение с результатами стандартного архиватора.
2. **Метод Хаффмана**:
    - Разработана программа с заданной постановкой задачи и подходом к решению.
    - Проведено тестирование, представлена оценка коэффициента сжатия.
3. **Методы LZ77 и LZ78**:
    - Реализованы алгоритмы, демонстрирующие их подходы к поиску и замене повторяющихся строк.
    - Оценены коэффициенты сжатия и проведен анализ их эффективности в сравнении с другими методами.

Работа позволила детально исследовать и сравнить различные подходы к сжатию данных. Это способствовало углублению знаний об алгоритмах и улучшению навыков программирования на C++. Отчет включает полные исходные коды и их результаты тестирования.

4 ЛИТЕРАТУРА

1. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона, 2010.
2. Кнут Д. Искусство программирования. Тома 1-4, 1976-2013.
3. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих, 2017.
4. Кормен Т.Х. и др. Алгоритмы. Построение и анализ, 2013.
5. Лафоре Р. Структуры данных и алгоритмы в Java. 2-е изд., 2013.
6. Макконнелл Дж. Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., 2018.
7. Скиена С. Алгоритмы. Руководство по разработке, 2011.
8. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2017.
9. Гасфилд Д. Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология, 2003.

По языку C++:

10. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
11. Павловская Т.А. C/C++. Программирование на языке высокого уровня, 2003.
12. Прата С. Язык программирования C++. Лекции и упражнения. - 6-е изд., 2012.
13. Седжвик Р. Фундаментальные алгоритмы на C++, 2001-2002
14. Хортон А. Visual C++ 2010. Полный курс, 2011.
15. Шилдт Г. Полный справочник по C++. 4-е изд., 2006.