



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

«МИРЭА - Российский технологический университет»

**РТУ МИРЭА**

---

---

Институт информационных технологий  
Кафедра математического обеспечения и стандартизации  
информационных технологий

**ОТЧЕТ**  
**ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 8**  
**«Кодирование и сжатие данных методами без потерь»**  
по дисциплине  
**«Структуры и алгоритмы обработки данных»**

Выполнил студент группы *ИКБО-03-22*

*Хохлинов Д.И.*

Принял

*Сорокин А.В.*

Практическая  
работа выполнена

«\_\_»\_\_\_\_\_2023 г.

«Зачтено»

«\_\_»\_\_\_\_\_2023 г.

Москва 2023

## СОДЕРЖАНИЕ

1 ЗАДАНИЕ 1 .....	3
1.1 Постановка задачи .....	3
1.2 Описание алгоритма RLE.....	3
1.3 Сжатие «удобного» текста алгоритмом RLE .....	3
1.5 Сжатие «неудобного» текста алгоритмом RLE. Модификация алгоритма RLE для сжатия неповторяющихся символов .....	4
2 ЗАДАНИЕ 2 .....	6
2.1 Постановка задачи .....	6
2.2 Описание метода LZ77 и его применение к заданному тексту.....	6
2.3 Метод LZ78 и его применение к заданному тексту .....	8
3 ЗАДАНИЕ 3 .....	10
3.1 Постановка задачи .....	10
3.2 Сжатие методом Шеннона-Фано.....	11
3.3 Сжатие методом Хаффмана .....	15
3.4 Применение метода Хаффмана для архивации файлов. Сравнение эффективности сжатия методом Хаффмана, с помощью базового архиватора Windows 10 и с помощью WinRAR .....	20
4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	30

## 1 ЗАДАНИЕ 1

### 1.1 Постановка задачи

Сжать текст, используя метод RLE (run length encoding/кодирование длин серий/групповое кодирование).

1) Описать процесс сжатия алгоритмом RLE.

2) Придумать текст, в котором есть длинные (в разумных пределах) серии из повторяющихся символов. Выполнить сжатие текста. Рассчитать коэффициент сжатия.

3) Придумать текст, в котором много неповторяющихся символов и между ними могут быть серии. Выполнить групповое сжатие, показать коэффициент сжатия. Применить алгоритм разделения текста при групповом кодировании, позволяющий повысить эффективность сжатия этого текста. Рассчитать коэффициент сжатия после применения алгоритма.

### 1.2 Описание алгоритма RLE

При сжатии алгоритмом RLE каждая группа из одинаковых символов (или «серия») заменяется на один символ и количество его повторений. Каждая серия обычно кодируется в два байта. Первый байт представляет количество символов в пробеге и называется счетчиком прогона. На практике кодированный прогон может включать от 1 до 128 или от 1 до 256 символов. Счетчик обычно содержит число символов минус один (значение в диапазоне значений от 0 до 127 или от 0 до 255). Второй байт — это значение символа в прогоне, которое содержится в диапазоне значений от 0 до 255 и именуется значением запуска.

### 1.3 Сжатие «удобного» текста алгоритмом RLE

Допустим, что имеется следующий текст:  
AAAAAAAAABBBBBAAAAACCCCCCCCCCCCCCCCCDDDD. Его длина в исходном состоянии равна 36 символам или 36 байтам, при условии, что на 1 символ выделяется 1 байт. Выполним сжатие этой строки алгоритмом RLE:

- 1) серия AAAAAAAAAA будет записана как 9A;
- 2) серия BBBB – как 5B;
- 3) серия CCCCCCCCCCCCCC – как 14C;
- 4) серия DDD – как 3D.

Итоговая запись будет выглядеть как 9A5B14C3D. Каждое число будет занимать 1 байт, следовательно, размер этой записи – 8 байт. Коэффициент сжатия равен  $\frac{36-8}{36} \approx 0,7777 \approx 77,7\%$ .

### **1.5 Сжатие «неудобного» текста алгоритмом RLE. Модификация алгоритма RLE для сжатия неповторяющихся символов**

Возьмем следующий текст: ABCDEFGGGGHABCDDEEFGHABC (26 байт). При попытке сжатия с помощью RLE получим следующую запись: 1A1B1C1D1E1F4G1H1A1B1C4D2E1F1G1H1A1B1C (38 байт). Коэффициент сжатия равен  $\frac{26-38}{26} \approx -0,462 = -46,2\%$ , что означает, что сжатая запись занимает больше места, чем исходная. Следовательно, метод RLE неэффективен для сжатия текстов с цепочками неповторяющихся символов.

Для решения этой проблемы можно провести модификацию метода RLE: теперь байт, хранящий число повторов, будет также хранить информацию о том, является ли следующий символ повторяющимся: 0 в старшем разряде байта - счетчика прогона будет означать повторение символа, записанного следующим байтом, указанное количество раз, а 1 в старшем разряде байта - счетчика прогона будет означать серию из стольких неповторяющихся символов, сколько указано в остальных разрядах байта - счетчика прогона.

Проведем сжатие того же текста модифицированным методом, в результате чего получим следующую запись (значения байтов – счетчиков прогона приведены в десятичной системе счисления): 134ABCDEF 4G

132HABC 4D 2E 134FGHABC (25 байт). Коэффициент сжатия в этом случае будет равен  $\frac{26-25}{26} \approx 0,038 = 3,8\%$ .

## 2 ЗАДАНИЕ 2

### 2.1 Постановка задачи

1) Выполнить каждую задачу варианта, представив алгоритм решения в виде таблицы и указав результат сжатия.

#### Задачи:

Сжатие данных по методу Лемпеля – Зива LZ77: Используя двухсимвольный алфавит (0, 1) закодировать следующую фразу: 101000100101010001011

Закодировать следующую фразу, используя код LZ78: какатанекатанекатата

2) Описать процесс восстановления сжатого текста.

### 2.2 Описание метода LZ77 и его применение к заданному тексту

Описание алгоритма кодирования методом LZ77:

Первоначально каждому символу алфавита присваивается определенный код (коды - порядковые номера, начиная с 0).

1. Выбирается первый (один) символ сообщения и заменяется на его код.

2. Выбираются следующие два символа и заменяются своими кодами. При этом комбинации двух символов присваивается свой код. Обычно это номер, равный числу уже использованных кодов. Так, если алфавит включает 8 символов, имеющих коды от 000 до 111, то первая двухсимвольная комбинация получит код 1000, следующая - код 1001 и т.д.

3. Выбираются из исходного текста очередные 2, 3,...N символов до тех пор, пока не образуется еще не встречавшаяся комбинация. Тогда этой комбинации присваивается очередной код, и поскольку совокупность A из первых N-1 символов уже встречалась, то она имеет свой код, который и записывается вместо этих N-1 символов. Т.е. можно представить

формирование кода в этом случае так: xxxxxxxx код из  $N(=8)$  символов выбран из кодируемой последовательности, тогда если для первых  $N-1$  символов уже был сформирован код, то заменяем эти  $N-1$  символы на их код. Каждый акт введения нового кода назовем шагом кодирования.

4. Процесс продолжается до исчерпания исходного текста.

Для примера рассмотрим строку 101000100101010001011. Проведем сжатие методом LZ77 (таблица 1):

Таблица 1 – Процесс сжатия методом LZ77

Содержимое окна	Содержимое буфера	Код новой последовательности	Текущий код
	10100010010 1010001011		
1	01000100101 010001011	1	1
01	00010010101 0001011	10	1.01
00	01001010100 01011	11	1.01.00
010	01010100010 11	100	1.01.00.100
0101	010001011	101	1.01.00.100.10 01
0100	01011	110	1.01.00.100.10 01.1000
01011		111	1.01.00.100.10 01.1000.1011

Получим LZ-код 1.01.00.100.1001.1000.1011.

Декодируем его, чтобы убедиться в правильности кодировки:

1 => 1; 01 => 01 (комбинация 10); 00 => 00 (комбинация 11); 100 => 010 (комбинация 100); 1001 => 0101 (комбинация 101); 1000 => 0100 (комбинация 110); 1011 => 01011 (комбинация 111). **Декодированная последовательность:** 101000100101010001011, что соответствует исходному тексту.

### 2.3 Метод LZ78 и его применение к заданному тексту

В отличие от LZ77, работающего с уже полученными данными, LZ78 ориентируется на данные, которые только будут получены (LZ78 не использует скользящее окно, он хранит словарь из уже просмотренных фраз). Алгоритм считывает символы сообщения до тех пор, пока накапливаемая подстрока входит целиком в одну из фраз словаря. Как только эта строка перестанет соответствовать хотя бы одной фразе словаря, алгоритм генерирует код, состоящий из индекса строки в словаре, которая до последнего введенного символа содержала входную строку, и символа, нарушившего совпадение. Затем в словарь добавляется введенная подстрока.

Рассмотрим строку «какатанекатанекатата». Проведем сжатие методом LZ78 (таблица 2):



Таблица 2 – Процесс сжатия методом LZ78

Номер записи словаря	Запись в словаре	Код
1	к	<0,к>
2	а	<0,а>
3	ка	<1,а>
4	т	<0,т>
5	ан	<2,н>
6	е	<0,е>
7	кат	<3,т>
8	ане	<5,е>
9	ката	<7,а>
10	та	<4,а>

В результате получим код 0к0а1а0т2н0е3т5е7а4а.

Декодирование проводится следующим образом: считываются очередные число и символ, после чего из них создается новая запись словаря, которая добавляется в конец результата.

Пример для полученного кода:

0к => к (текущий результат: к); 0а => а (текущий результат: ка); 1а => ка (текущий результат: кака); 0т => т (текущий результат: какат); 2н => ан (текущий результат: какатан); 0е => е (текущий результат: какатане); 3т => кат (текущий результат: какатанекат); 5е => ане (текущий результат: какатанекатане); 7а => ката (текущий результат: какатанекатанеката); 4а => та (текущий результат: какатанекатанекатата). **Результат декодирования:** «какатанекатанекатата», что соответствует исходному тексту.

### 3 ЗАДАНИЕ 3

#### 3.1 Постановка задачи

Разработать программы (или только алгоритмы на псевдокоде или словесно) сжатия и восстановления текста методами Шеннона-Фано и Хаффмана.

1. Сформировать отчет по разработке каждого алгоритма в соответствии с требованиями.

1.1. По методу Шеннона-Фано. **Исходный текст:**

Прибавь к ослиной голове

Еще одну, получишь две.

Но сколько б ни было ослов,

Они и двух не свяжут слов.

1) Привести постановку задачи, описать алгоритм формирования префиксного дерева и алгоритм кодирования, декодирования.

2) Представить таблицу формирования кода.

3) Изобразить префиксное дерево.

4) Рассчитать коэффициент сжатия.

1.2. По методу Хаффмана. **Исходный текст:** Хохлинов Дмитрий Иванович.

1) Привести постановку задачи, описать алгоритм формирования префиксного дерева и алгоритм кодирования, декодирования.

2) Построить таблицу частот встречаемости символов в исходной строке для чего сформировать алфавит исходной строки и посчитать количество вхождений (частот) символов и их вероятности появления.

3) Изобразить префиксное дерево Хаффмана.

4) Упорядочить построенное дерево слева-направо (при необходимости) и изобразить его.

5) Провести кодирование исходной строки по аналогии с примером.

6) Рассчитать коэффициенты сжатия относительно кодировки ASCII и относительно равномерного кода.

7) Рассчитать среднюю длину полученного кода и его дисперсию.

8) По результатам выполненной работы сделать выводы и сформировать отчет. Отобразить результаты выполнения всех требований, предъявленных в задании и оформить разработку программы: постановка, подход к решению, код, результаты тестирования.

1.3. Реализовать и отладить программу. 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.

### **3.2 Сжатие методом Шеннона-Фано**

При сжатии методом Шеннона-Фано просматривается частота появления символов в исходном файле. Затем создается код переменной длины, для которого верны два условия:

1) ни один код не является началом другого кода – это обеспечивает возможность декодирования;

2) чем чаще встречается символ, тем короче длина его кода.

Код формируется по следующему алгоритму:

1) если длина множества символов равна 1, то алгоритм завершает работу;

2) символы сортируются по частоте появления в порядке невозрастания;

3) множество символов делится на две части так, чтобы их суммарная частота примерно совпадала;

4) для элементов первого множества к текущему коду дописывается 0, для элементов второго множества – 1;

5) этот алгоритм рекурсивно выполняется для первого и второго множеств.

Рассмотрим работу этого алгоритма на примере (таблица 3).

Таблица 3 – Пример сжатия методом Шеннона-Фано

Символ	Кол-во	Разряды							Код	Кол-во бит
		1	2	3	4	5	6	7		
пробел	16	0	0	0					000	48
о	12	0	0	1					001	36
в	7	0	1	0					010	21
л	7	0	1	1	0				0110	28
и	6	0	1	1	1				0111	24
с	5	1	0	0	0				1000	20
н	5	1	0	0	1				1001	20
у	4	1	0	1	0	0			10100	20
е	4	1	0	1	0	1			10101	20
д	3	1	0	1	1	0			10110	15
б	3	1	0	1	1	1			10111	15
ь	3	1	1	0	0	0			11000	15

Продолжение таблицы 3

Символ	Кол-во	Разряды							Код	Кол-во бит
		1	2	3	4	5	6	7		
к	3	1	1	0	0	1			11001	15
,	2	1	1	0	1	0	0		110100	12
т	1	1	1	0	1	0	1		110101	6
ж	1	1	1	0	1	1	0		110110	6
я	1	1	1	0	1	1	1	0	1101110	7
х	1	1	1	0	1	1	1	1	1101111	7
О	1	1	1	1	0	0	0		111000	6
Н	1	1	1	1	0	0	1	0	1110010	7
ш	1	1	1	1	0	0	1	1	1110011	7
ч	1	1	1	1	0	1	0		111010	6
п	1	1	1	1	0	1	1	0	1110110	7
щ	1	1	1	1	0	1	1	1	1110111	7
Е	1	1	1	1	1	0	0		111100	6
г	1	1	1	1	1	0	1	0	1111010	7
й	1	1	1	1	1	0	1	1	1111011	7
а	1	1	1	1	1	1	0	0	1111100	7
р	1	1	1	1	1	1	0	1	1111101	7
П	1	1	1	1	1	1	1	0	1111110	7
.	1	1	1	1	1	1	1	1	1111111	7

Оценка объема и коэффициента сжатия методом Шеннона-Фано

Объем исходной фразы (кодировка UTF-8):  $8 \cdot 97 = 776$  бит.

Объем закодированной фразы: 423 бит.

Коэффициент сжатия:  $\frac{776-423}{776} \approx 0,4549 = 45,49\%$ .

Дерево, построенное с помощью метода, показано на рисунке 1.

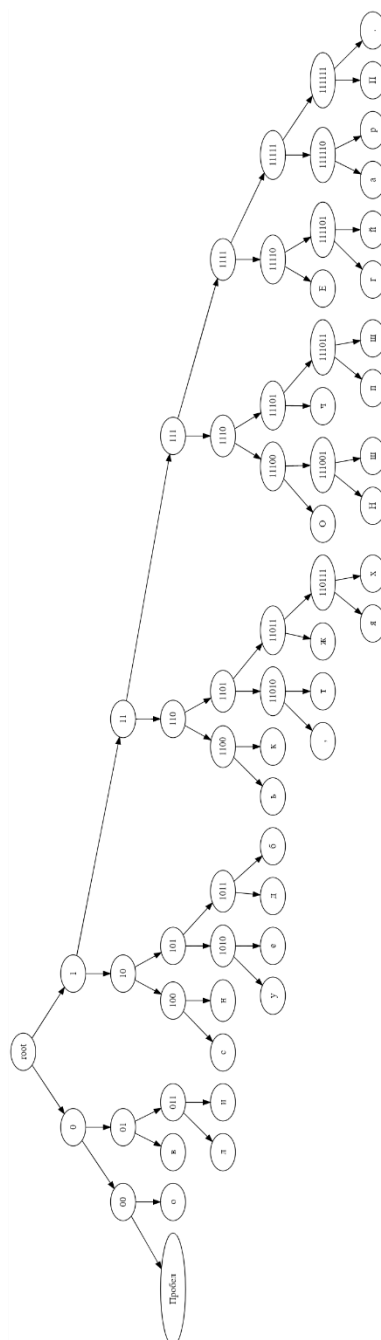


Рисунок 1 – Дерево, построенное методом

### 3.3 Сжатие методом Хаффмана

Сжатие методом Хаффмана проводится по следующему алгоритму:

- 1) Определить частоту появления символов в сжимаемом тексте;
- 2) Отсортировать символы по частоте появления в порядке невозрастания. Далее для простоты описания алгоритма будем считать символы узлами двоичного дерева;
- 3) Определить два узла с минимальной частотой появления. Создать новый узел дерева и установить для него левого и правого потомка равными узлам с минимальной частотой появления. Частота появления для этого узла будет равна сумме частот появления узлов - непосредственных потомков;
- 4) Отсортировать те узлы, которые не являются чьими-либо потомками, по частоте появления в порядке невозрастания.
- 5) Если количество узлов, не являющихся потомками, больше 1, то перейти к п. 3; если нет, то завершить работу.

Результатом работы этого алгоритма будет дерево кодирования Хаффмана, по которому каждому символу из текста ставится в соответствие двоичный код, определяемый маршрутом из корневого элемента до узла, содержащего этот символ.

Для получения дерева кодирования Хаффмана воспользуемся следующей программой на C++ (листинг 1):

Листинг 1 – main.cpp

```
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>

using namespace std;

struct node {
    string value = "";
    int count = 1;
    node* left = nullptr;
```

## Продолжение листинга 1

```
node* right = nullptr;

node(string ch)
{
    this->value = ch;
}

~node() {
    if (this->left) delete left;
    if (this->right) delete right;
}

void setLeft(node* left) {
    this->left = left;
}

void setRight(node* right) {
    this->right = right;
}

void setCount(int count) {
    this->count = count;
}

node* getLeft() {
    return this->left;
}

node* getRight() {
    return this->right;
}

int getCount() {
    return this->count;
}

void show(string path)
{
    if ((this->left == nullptr) && (this->right == nullptr))
    {
        for (int i = 0; i < path.size(); i++)
        {
            cout << "\t";
        }
        cout << " < " << this->value << " (count: " << this->count << ") > " << path << "\n";
    }
    else
    {
        if (this->right)
            this->right->show(path + "1");
        for (int i = 0; i < path.size(); i++)
        {
            cout << "\t";
        }
    }
}
```



## Продолжение листинга 1

```
        cout << "|" (count: " << this->count << ")" << endl;
        if (this->left)
            this->left->show(path + "0");
    }
}

};

class haffman_tree {
    node* root;
public:
    haffman_tree() {
        this->root = nullptr;
    }

    void show() {
        this->root->show("");
    }

    void construct(string input) {
        vector<node*> chs;
        vector<int> cnts;
        for (int i = 0; i < input.size(); i++)
        {
            bool modified = false;
            for (int j = 0; j < chs.size(); j++)
            {
                if (chs[j]->value[0] == input[i])
                {
                    modified = true;
                    chs[j]->setCount(++cnts[j]);
                    break;
                }
            }
            if (!modified)
            {
                string v = string(1, input[i]);
                chs.push_back(new node(v));
                cnts.push_back(1);
            }
        }

        while (chs.size() > 1)
        {
            for (int i = 0; i < chs.size(); i++)
            {
                for (int j = 0; j < chs.size(); j++)
                {
                    if (cnts[i] < cnts[j])
                    {
                        swap(cnts[i], cnts[j]);
                        swap(chs[i], chs[j]);
                    }
                }
            }
            node* ch1 = chs[0];
```

## Продолжение листинга 1

```
node* ch2 = chs[1];
node* ch3 = new node(ch1->value + ch2->value);
ch3->setLeft(ch1);
ch3->setRight(ch2);
ch3->setCount(ch1->getCount() + ch2->getCount());
chs.push_back(ch3);
cnts.push_back(cnts[0] + cnts[1]);

vector<node*> temp;
vector<int> temp2;
for (int i = 2; i < chs.size(); i++)
{
    temp.push_back(chs[i]);
    temp2.push_back(cnts[i]);
}
chs = temp;
cnts = temp2;
}

this->root = chs[0];
}

~haffman_tree() {
    delete root;
}

};

int main()
{
    setlocale(LC_ALL, "ru");
    string input = "Хохлинов Дмитрий Иванович";
    haffman_tree tree = haffman_tree();
    tree.construct(input);
    tree.show();
    return 0;
}
```

Результат работы программы приведен на рисунке 2.

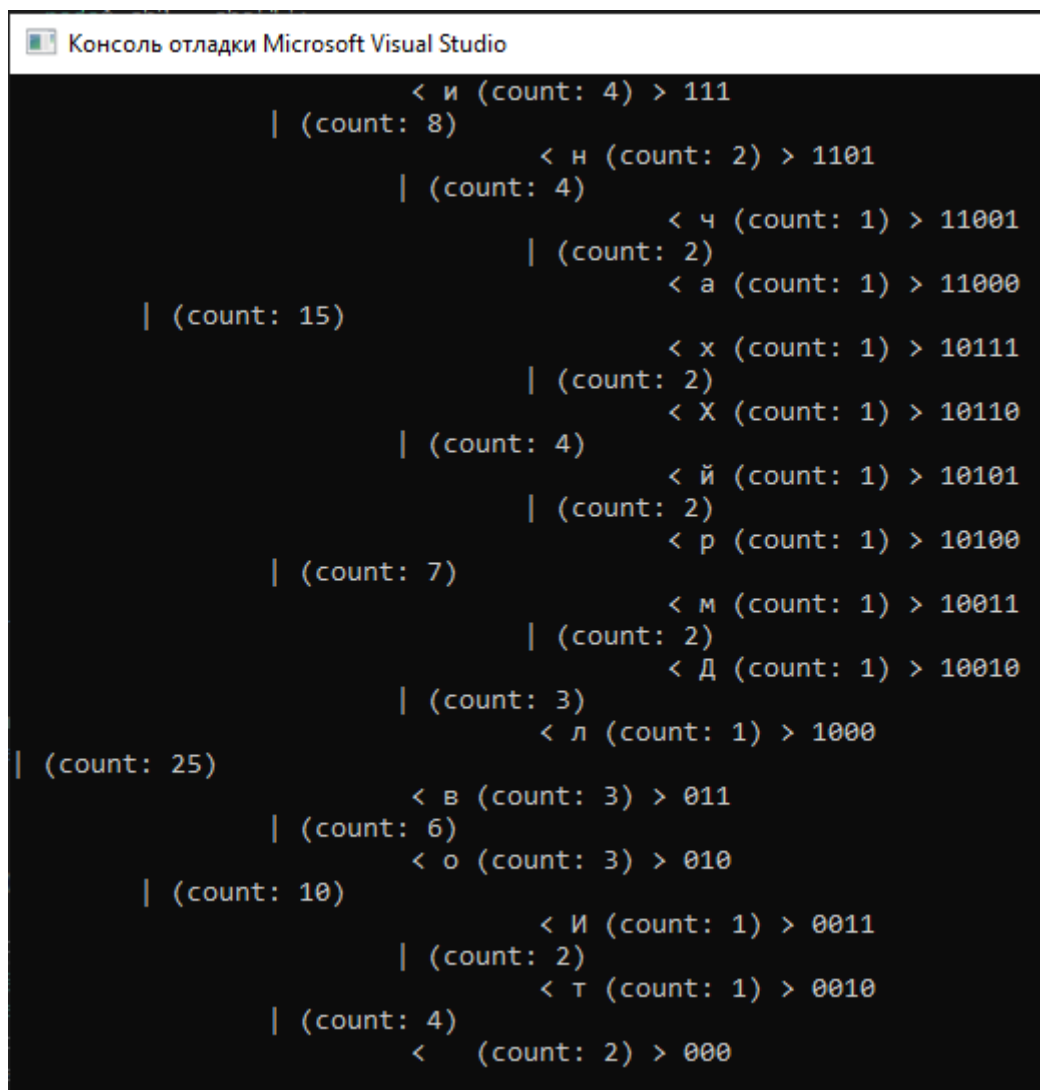


Рисунок 2 – Дерево кодирования Хаффмана, составленное для ФИО, с выводом частоты всех узлов

Таблица кодов для этого примера имеет следующий вид (таблица 4):

Таблица 4 – Таблица кодов для дерева кодирования Хаффмана

Символ	и	н	ч	а	х	Х	й	р	м	Д	л	в	о	И	т	пробел
Код	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	000
	1	1	1	1	0	0	0	0	0	0	0	1	1	0	0	
	1	0	0	0	1	1	1	1	0	0	0	1	0	1	1	
		1	0	0	1	1	0	0	1	1	0			1	0	
			1	0	1	0	1	0	1	0						
Кол-во бит на символ	3	4	5	5	5	5	5	5	5	5	4	3	3	4	4	3

Результат сжатия будет выглядеть следующим образом: 10110 010 10111 1000 111 1101 010 011 000 10010 10011 111 0010 10100 111 10101 000 0011 011 11000 1101 010 011 111 11001 (для лучшей читаемости биты сгруппированы в коды символов).

Рассчитаем коэффициент сжатия, среднюю длину кода и дисперсию для метода Хаффмана:

Объем исходного текста в кодировке UTF-8:  $8 \cdot 25 = 200$  бит.

Объем сжатого текста: 96 бит.

Коэффициент сжатия:  $\frac{200-96}{200} = 0,52 = 52\%$ .

Средняя длина кода: 4,25 бит.

Дисперсия:  $\sum_i \frac{n_i(w_{\text{cp}} - w_i)^2}{N} \approx 0,9425$  бит, где  $n_i$  – частота появления символа,  $N$  – общее количество символов в тексте,  $w_{\text{cp}}$  – средняя длина кода,  $w_i$  – длина кода символа.

### **3.4 Применение метода Хаффмана для архивации файлов. Сравнение эффективности сжатия методом Хаффмана, с помощью базового архиватора Windows 10 и с помощью WinRAR**

Доработаем программу из листинга 1, добавив в нее возможность архивировать текстовый файл путем сжатия текста методом Хаффмана (возможность декодирования в программе не представлена, так как постановка задачи этого не требует). Итоговая программа представлена в листинге 2.

Листинг 2 – Программа из листинга 1 с возможностью архивации

```
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
#include <fstream>
#include <queue>

using namespace std;
```

## Продолжение листинга 2

```
struct node {
    string value = "";
    int count = 1;
    node* left = nullptr;
    node* right = nullptr;

    node(string ch)
    {
        this->value = ch;
    }

    ~node() {
        if (this->left) delete left;
        if (this->right) delete right;
    }

    void setLeft(node* left) {
        this->left = left;
    }

    void setRight(node* right) {
        this->right = right;
    }

    void setCount(int count) {
        this->count = count;
    }

    node* getLeft() {
        return this->left;
    }

    node* getRight() {
        return this->right;
    }

    int getCount() {
        return this->count;
    }

    void show(string path)
    {
        if ((this->left == nullptr) && (this->right == nullptr))
        {
            for (int i = 0; i < path.size(); i++)
            {
                cout << "\t";
            }
            cout << " < " << this->value << " (count: " << this->count << ") > " << path << "\n";
        }
        else
        {
            if (this->right)
                this->right->show(path + "1");
        }
    }
};
```

## Продолжение листинга 2

```
        for (int i = 0; i < path.size(); i++)
        {
            cout << "\t";
        }
        cout << "| (count: " << this->count << ")" << endl;
        if (this->left)
            this->left->show(path + "0");
    }
}

};

class haffman_tree {
    node* root;
public:
    haffman_tree() {
        this->root = nullptr;
    }

    void show() {
        this->root->show("");
    }

    void convert_to_table(vector<char>& chars, vector<string>& codes)
    {
        vector<char> chs;
        vector<string> cds;
        queue<node*> queue1; //узлы
        queue<string> queue2; //коды
        queue1.push(root);
        queue2.push("");
        while (queue1.size() > 0)
        {
            //для очередного узла добавляем в очередь его потомков
            node* cur = queue1.front();
            string cur_path = queue2.front();
            queue1.pop();
            queue2.pop();
            if (cur->getLeft())
            {
                queue1.push(cur->getLeft());
                queue2.push(cur_path + "0");
            }

            if (cur->getRight())
            {
                queue1.push(cur->getRight());
                queue2.push(cur_path + "1");
            }

            //добавляем символ, если текущий узел - лист
            if ((cur->getLeft() == nullptr) && (cur->getRight() ==
nullptr))
            {
                chs.push_back(cur->value[0]);
                cds.push_back(cur_path);
            }
        }
    }
};
```

## Продолжение листинга 2

```
    }
}
chars = vector<char>(chs);
codes = vector<string>(cds);
}

void construct(string input) {
    vector<node*> chs;
    vector<int> cnts;
    for (int i = 0; i < input.size(); i++)
    {
        bool modified = false;
        for (int j = 0; j < chs.size(); j++)
        {
            if (chs[j]->value[0] == input[i])
            {
                modified = true;
                chs[j]->setCount(++cnts[j]);
                break;
            }
        }
        if (!modified)
        {
            string v = string(1, input[i]);
            chs.push_back(new node(v));
            cnts.push_back(1);
        }
    }

    while (chs.size() > 1)
    {
        for (int i = 0; i < chs.size(); i++)
        {
            for (int j = 0; j < chs.size(); j++)
            {
                if (cnts[i] < cnts[j])
                {
                    swap(cnts[i], cnts[j]);
                    swap(chs[i], chs[j]);
                }
            }
        }
        node* ch1 = chs[0];
        node* ch2 = chs[1];
        node* ch3 = new node(ch1->value + ch2->value);
        ch3->setLeft(ch1);
        ch3->setRight(ch2);
        ch3->setCount(ch1->getCount() + ch2->getCount());
        chs.push_back(ch3);
        cnts.push_back(cnts[0] + cnts[1]);

        vector<node*> temp;
        vector<int> temp2;
        for (int i = 2; i < chs.size(); i++)
        {
```

## Продолжение листинга 2

```
        temp.push_back(chs[i]);
        temp2.push_back(cnts[i]);
    }
    chs = temp;
    cnts = temp2;
}

    this->root = chs[0];
}

~haffman_tree() {
    delete root;
}

};

int main()
{
    setlocale(LC_ALL, "ru");
    fstream input_file("input.txt", ios::in);
    string input = "";
    while (!input_file.eof())
    {
        string temp = "";
        getline(input_file, temp);
        input += temp;
    }
    haffman_tree tree = haffman_tree();
    tree.construct(input);
    tree.show();
    vector<char> chars;
    vector<string> codes;
    tree.convert_to_table(chars, codes);
    string output = "";
    for (int i = 0; i < input.size(); i++)
    {
        for (int j = 0; j < chars.size(); j++)
        {
            if (input[i] == chars[j])
            {
                output += codes[j];
                break;
            }
        }
    }
    vector<unsigned char> output_chars;
    for (int i = output.size() - 8; i >= 0; i -= 8) //упаковка 8 бит
        в 1-байтовое число
    {
        string temp = "";
        for (int j = 0; j < 8; j++)
        {
            temp += output[i + j];
        }
        output_chars.push_back(unsigned char(strtol(temp.c_str(),
        nullptr, 2)));
    }
}
```



## Продолжение листинга 2


```
    }  
    fstream output_file("archived.txt", ios::out | ios::binary);  
    for (int i = output_chars.size() - 1; i >= 0; i--)  
    {  
        output_file.write((char*) &(output_chars[i]),  
sizeof(unsigned char));  
    }  
    output_file.close();  
    return 0;  
}
```

Программа получает на вход файл input.txt в кодировке Windows-1251 и на его основе создает файл archived.txt, сжатый с помощью метода Хаффмана.

Для тестирования в качестве входного файла использовался файл, в котором записано ФИО (Хохлинов Дмитрий Иванович). Его размер в кодировке Windows-1251 составляет 25 байт (рисунок 3).

Имя	Дата изменения	Тип	Размер
Debug	12.12.2023 23:18	Папка с файлами	
archived.txt	12.12.2023 23:22	Файл "TXT"	1 КБ
input.txt	12.12.2023 23:21	Файл "TXT"	1 КБ
siaod_p2_8.cpp	12.12.2023 23:23	C++ Source	5 КБ
siaod_p2_8.vcxproj	11.12.2023 21:34	VC++ Project	8 КБ
siaod_p2_8.vcxproj.filters	11.12.2023 21:34	VC++ Project Filte...	2 КБ
siaod_p2_8.vcxproj.user	11.12.2023 21:34	Per-User Project O...	1 КБ

input.txt  
Файл "TXT"




Дата изменения: 12.12.2023 23:21  
Размер: 25 байт  
Дата создания: 12.12.2023 23:18

Рисунок 3 – Размер файла до архивации

Файл archived.txt, полученный в результате работы программы, имеет размер 12 байт (рисунок 4).

Имя	Дата изменения	Тип	Размер
Debug	12.12.2023 23:18	Папка с файлами	
archived.txt	12.12.2023 23:22	Файл "TXT"	1 КБ
input.txt	12.12.2023 23:21	Файл "TXT"	1 КБ
siaod_p2_8.cpp	12.12.2023 23:23	C++ Source	5 КБ
siaod_p2_8.vcxproj	11.12.2023 21:34	VC++ Project	8 КБ
siaod_p2_8.vcxproj.filters	11.12.2023 21:34	VC++ Project Filte...	2 КБ
siaod_p2_8.vcxproj.user	11.12.2023 21:34	Per-User Project O...	1 КБ

archived.txt  
Файл "TXT"



Дата изменения: 12.12.2023 23:22  
Размер: 12 байт  
Дата создания: 12.12.2023 23:16

Рисунок 4 – Размер файла после архивации

Коэффициент сжатия файла:  $\frac{25-12}{25} = 0,52 = 52\%$ .

Для сравнения проведем сжатие этого же входного файла с помощью базового архиватора Windows 10 (рисунок 5) и WinRAR (рисунок 6).

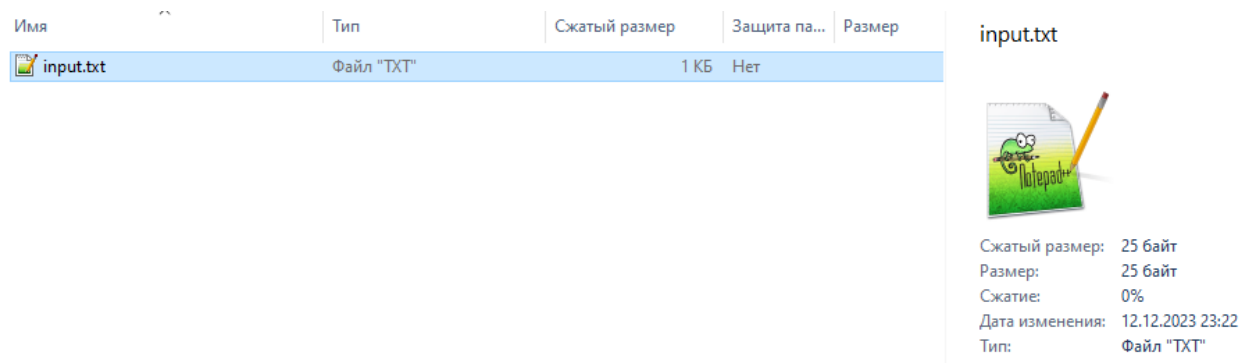


Рисунок 5 – Сжатие файла с помощью базового архиватора Windows 10

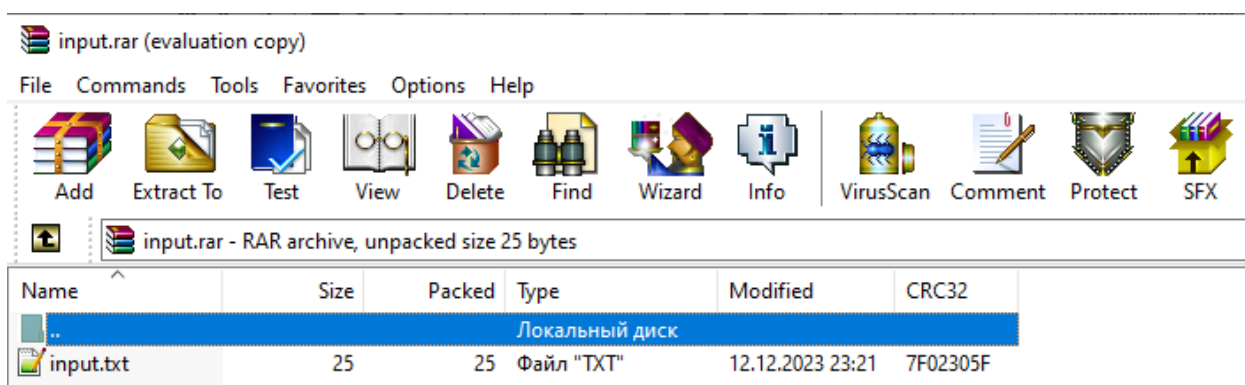


Рисунок 6 – Сжатие файла с помощью WinRAR

Из-за малого объема файла сжатие при помощи архиваторов не проводилось. Повторим эти действия с файлом большего объема.

Исходный файл в кодировке Windows-1251 (рисунок 7):



archived.txt

Файл "TXT"



Дата изменения: 12.12.2023 23:36

Размер: 1,37 КБ

Дата создания: 12.12.2023 23:16

Рисунок 10 – Размер архивированного файла

Размеры файла, архивированного при помощи Windows 10 и WinRAR (рисунки 11, 12):

input.txt



Сжатый размер: 1,22 КБ

Размер: 2,38 КБ

Сжатие: 49%

Дата изменения: 12.12.2023 23:35

Тип: Файл "TXT"

Рисунок 11 – Размер файла, архивированного базовым архиватором Windows 10

Name	Size	Packed	Type	Modified	CRC32
..			Локальный диск		
input.txt	2 439	1 309	Файл "TXT"	12.12.2023 23:35	2616BE22

Рисунок 12 – Размер файла, архивированного при помощи WinRAR

Рассчитаем коэффициенты сжатия:

1) для метода Хаффмана:  $\frac{2,38-1,37}{2,38} \approx 0,4243 \approx 42,43\%$ ;

2) для базового архиватора Windows 10: 49%;

3) для WinRAR:  $\frac{2439-1309}{2439} \approx 0,4633 \approx 46,33\%$ .

**Вывод:** метод Хаффмана имеет примерно такую же эффективность, что и методы, используемые в различных архиваторах, следовательно, он может применяться для архивации файлов. При этом эффективность метода Хаффмана не зависит от размера архивируемого файла, что было продемонстрировано на рисунках 3-6.

#### 4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Задание на самостоятельную работу: [https://online-edu.mirea.ru/pluginfile.php?file=%2F1144136%2Fmod\\_assign%2Fintroattachment%2F0%2FСиАОД%20Самостоятельная%20работа%208%20%28кодирование%20и%20сжатие%20данных%29.pdf](https://online-edu.mirea.ru/pluginfile.php?file=%2F1144136%2Fmod_assign%2Fintroattachment%2F0%2FСиАОД%20Самостоятельная%20работа%208%20%28кодирование%20и%20сжатие%20данных%29.pdf), дата обращения: 13.12.23
2. Структуры и алгоритмы обработки данных – Методы сжатия данных: [https://online-edu.mirea.ru/pluginfile.php?file=%2F1144196%2Fmod\\_folder%2Fcontent%2F0%2F%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D1%8B%20%D1%81%D0%B6%D0%B0%D1%82%D0%B8%D1%8F%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85.pdf](https://online-edu.mirea.ru/pluginfile.php?file=%2F1144196%2Fmod_folder%2Fcontent%2F0%2F%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D1%8B%20%D1%81%D0%B6%D0%B0%D1%82%D0%B8%D1%8F%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85.pdf), дата обращения: 13.12.23