

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"МИРЭА - Российский технологический университет"**

# РТУ МИРЭА

Институт информационных технологий Кафедра математического обеспечения и стандартизации ИТ

# ОТЧЕТ

**ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2.6:**  **Кодирование и сжатие данных методами без потерь**

# ПО ДИСЦИПЛИНЕ

**«** СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ**»**

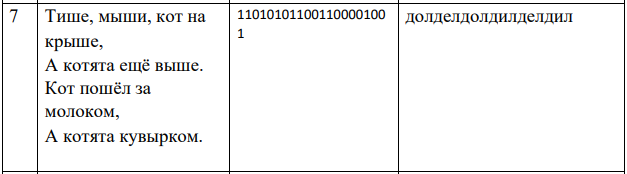
|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Выполнил студент группы ИКБО-01-21 |  |  | Бойко Д.А. |
| Принял старший преподаватель |  |  | Туманова М.Б. |
|  |  |  |  |
|  |  |  |  |
| Практическая работа  выполнена «26» ноября 2022 г. | | | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *(подпись студента)* |
|  | | |  |
| «Зачтено» « » октября 2022 г. | | | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *(подпись руководителя)* |

Москва 2022

**Цель:** Исследовать алгоритмы сжатия, а также разработать программы сжатия и восстановления текста методами Хаффмана и Шеннона-Фано. **Задание 1**

**Ход работы**

**Индивидуальный вариант** (7):



**Модель решения**:

1. Метод Шеннона-Фано состоит из следующих основных этапов:

1. Строится таблица вероятностей символов.
2. Эта таблица сортируется по убыванию.
3. Таблица рекурсивно делится так, чтобы суммарные вероятности получившихся частей были примерно равны друг другу, до тех пор, пока не получатся части по одному символу.
4. По получившимся частям строится бинарное дерево.

Рассмотрим пример из варианта: «Кот пошёл за молоком, А котята кувырком. Кот пришёл без молока, А котята ха – ха – ха.» Будем считать заглавные и строчные буквы разными. Получим следующую таблицу количества вхождений символов (табл. 1).

Таблица 1 – Таблица количества вхождений символов

|  |  |
| --- | --- |
| « » | 15 |
| о | 9 |
| к | 7 |
| т | 6 |
| ш | 5 |
| , | 4 |
| а | 4 |
| е | 4 |
| ы | 4 |
| м | 4 |
| . | 2 |
| А | 2 |
| ё | 2 |
| я | 2 |
| р | 2 |
| л | 2 |
| и | 2 |
| в | 2 |
| п | 1 |
| у | 1 |
| н | 1 |
| щ | 1 |
| з | 1 |
| Т | 1 |
| К | 1 |

Разделив каждое значение этой таблицы на длину строки (86) получим вероятности (таблица 2).

Таблица 2 – Таблица вероятностей вхождений символов

|  |  |
| --- | --- |
| « » | 0.176471 |
| о | 0.105882 |
| к | 0.0823529 |
| т | 0.0705882 |
| ш | 0.0588235 |
| , | 0.0470588 |
| а | 0.0470588 |
| е | 0.0470588 |
| ы | 0.0470588 |
| м | 0.0470588 |
| . | 0.0235294 |
| А | 0.0235294 |
| ё | 0.0235294 |
| я | 0.0235294 |
| р | 0.0235294 |
| л | 0.0235294 |
| и | 0.0235294 |
| в | 0.0235294 |
| п | 0.0117647 |
| у | 0.0117647 |
| н | 0.0117647 |
| щ | 0.0117647 |
| з | 0.0117647 |
| Т | 0.0117647 |
| К | 0.0117647 |

Произведя сортировку данной таблицы по не возрастанию вероятностей и осуществив операции деления поровну, получим таблицу кодировки символов (табл. 3).

Таблица 3 – Таблица кодировки символов

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Символ | Кол-во | 1-я  цифр  а | 2-я  цифр  а | 3-я  цифр  а | 4-я  цифр  а | 5-я  цифр  а | 6-я  цифр  а | 7-я  цифр  а | Код | Кол  -во бит |
| « » | 15 | 0 | 0 | 0 |  |  |  |  | 000 | 45 |
| о | 9 | 0 | 0 | 1 |  |  |  |  | 001 | 27 |
| к | 7 | 0 | 1 | 0 |  |  |  |  | 010 | 21 |
| т | 6 | 0 | 1 | 1 | 0 |  |  |  | 0110 | 24 |
| ш | 5 | 0 | 1 | 1 | 1 |  |  |  | 0111 | 20 |
| , | 4 | 1 | 0 | 0 | 1 | 0 |  |  | 10010 | 20 |
| а | 4 | 1 | 0 | 1 | 1 | 0 |  |  | 10110 | 20 |
| е | 4 | 1 | 0 | 0 | 0 |  |  |  | 1000 | 16 |
| ы | 4 | 1 | 0 | 1 | 0 |  |  |  | 1010 | 16 |
| м | 4 | 1 | 0 | 0 | 1 | 1 |  |  | 10011 | 20 |
| . | 2 | 1 | 1 | 0 | 0 | 1 |  |  | 11001 | 10 |
| А | 2 | 1 | 1 | 1 | 0 | 0 |  |  | 11100 | 10 |
| ё | 2 | 1 | 1 | 0 | 1 | 1 | 0 |  | 110110 | 12 |
| я | 2 | 1 | 1 | 0 | 1 | 1 | 1 |  | 110111 | 12 |
| р | 2 | 1 | 1 | 1 | 0 | 1 | 0 |  | 111010 | 12 |
| л | 2 | 1 | 1 | 0 | 0 | 0 |  |  | 11000 | 10 |
| и | 2 | 1 | 0 | 1 | 1 | 1 |  |  | 10111 | 10 |
| в | 2 | 1 | 1 | 0 | 1 | 0 |  |  | 11010 | 10 |
| п | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1111011 | 7 |
| у | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 111011 | 7 |
| н | 1 | 1 | 1 | 1 | 1 | 0 | 0 |  | 111100 | 6 |
| щ | 1 | 1 | 1 | 1 | 0 | 1 | 1 |  | 111011 | 6 |
| з | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1111110 | 7 |
| Т | 1 | 1 | 1 | 1 | 1 | 1 | 0 |  | 111110 | 6 |
| К | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1111010 | 7 |

Результат:

1111101011101111000100100001001110100111101111001000001000101100001111001011000001011101010100111100010010000111000000100010110110111011010110000100011101111011000011010101001111000110010001111010001011000011110110010111110110110000001111110101100001001100111000001010001100111001000011100000010001011011011101101011000001011111111101010101110100100011001111001

Длина незакодированной строки: 680 бит.

Длина закодированной строки: 361 бит.

2. Метод Лемпеля-Зива LZ77 для сжатия двоичного кода

Идея:

Если в тексте появляется последовательность из двух ранее **уже встречавшихся символов**, то эта последовательсность объявляется **новым символом**, для нее назначается **код**, который (при определенных условиях) может быть короче исходной последовательности

Коды – это порядковые номера, начиная с 0 (очередной номер, равный числу уже использованных кодов)

Так, если в алфавите 8 символов, то их двоичные коды – от 0 до 111; тогда первая 2-символьная комбинация получит код 1000, следующая – 1001 и т.д.

В сжатый текст на место исходной последовательности записывается её код

При декодировании повторяются аналогичные действия, поэтому становятся известными последовательности символов для каждого кода.

Алгоритм:

1. Выбирается первый символ сообщения и заменяется на его код.
2. Выбираются следующие два символа и заменяются своими кодами. Одновременно этой комбинации двух символов присваивается свой код.
3. Из исходного текста выбираются очередные 2,3,...,N символов, пока не образуется ещё не встречавшаяся комбинация. Тогда этой комбинации присваивается очередной код, и поскольку совокупность из первых N-1 символов уже встречалась, то она имеет свой код, который и записывается вместо этих N-1 символов. Каждый акт введения нового кода – шаг кодирования.
4. Процесс продолжается до исчерпания исходного текста.

Таким образом получаем таблицу 4 для исходной строки «110101011001100001001».

1.10.101.01.100.11.00.001.001

Таблица 4 – Таблица LZ77-кодирования

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Исходный код | 1. | 10. | 101. | 01. | 100. | 11. | 00. | 001. | 001 |
| LZ-код | 1. | 10. | 101. | 001. | 100. | 011. | 000. | 1111. | 1111 |
| R |  | 2 |  | 3 |  |  |  | 4 |  |
| Вводимые коды | - | 10 | 11 | 100 | 101 | 110 | 111 | 1000 | 1001 |

3. Метод сжатия Лемпеля-Зива LZ78.

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. Если словарь уже заполнен, то из него предварительно удаляют менее всех используемую в сравнениях фразу. Если в конце алгоритма мы не находим символ, нарушивший совпадения, то тогда мы выдаем код в виде (индекс строки в словаре без последнего символа, последний символ). Используя данный алгоритм получаем таблицу 5.

Рассматриваемая строка: «долделдолдилделдил».

(д о л де лд ол ди лде лди л)

Таблица 5 – Процесс сжатия алгоритмом LZ78

|  |  |  |
| --- | --- | --- |
| Словарь | Считываемое содержимое | Код |
|  | д | <0, д> |
| д=1; | о | <0, о> |
| д=1; о=2; | л | <0, л> |
| д=1; о=2; л=3; | де | <1, е> |
| д=1; о=2; л=3; де=4; | лд | <3, д> |
| д=1; о=2; л=3; де=4; лд=5; | ол | <2, л> |
| д=1; о=2; л=3; де=4; лд=5; ол=6; | ди | <1, и> |
| д=1; о=2; л=3; де=4; лд=5; ол=6; ди=7; | лде | <5, е> |
| д=1; о=2; л=3; де=4; лд=5; ол=6; ди=7; лде=8; | лди | <5, и> |
| д=1; о=2; л=3; де=4; лд=5; ол=6; ди=7; лде=8; лди=9; | л | <3, EOF> |

# Задание 2

# Ход работы

**1. Метод Шеннона-Фано.**

Алгоритм выполнения сжатия методом Шеннона-Фано был описан в первом пункте предыдущего задания.

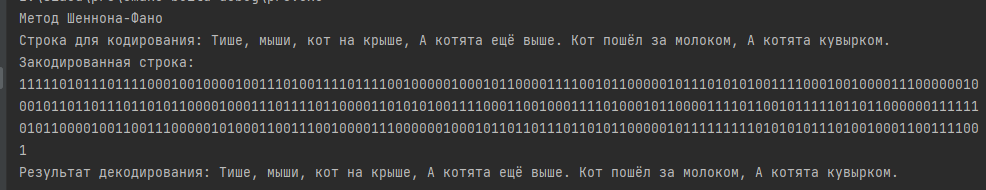


Рисунок 1 – Результат кодирования и декодирования строки методом Шеннона-Фано

Коэффициент сжатия: 0.53

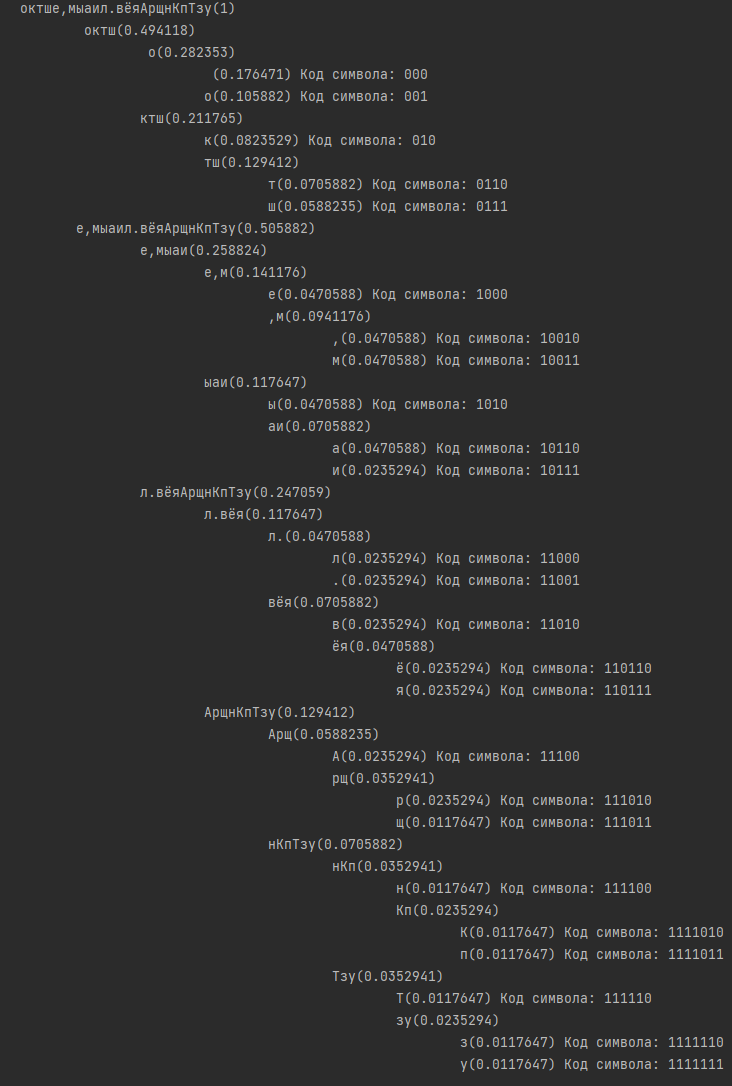


Рисунок 2 – Дерево для метода Шеннона-Фано

**2. Метод Хаффмана.**

Алгоритм заключается в следующем: как и в методе Шеннона-Фано строится таблица вероятностей символов, отсортированная по убыванию. Далее каждую итерацию два символа с наименьшей вероятностью добавляются в дерево и к ним присоединяется родительский узел. Эти итерации с учётом получившихся новых узлов повторяются до тех пор, пока не останется один узел, который становится корневым.



Рисунок 3 – Результат кодирования и декодирования строки методом Хаффмана

Коэффициент сжатия программой: 0.519

Коэффициент сжатия rar: 0.918

Средняя длина: 0.037\*5\*4 + 0.074\*5\*4 + 0.037\*8\*5 + 0,148 \* 3 = 4,144

Дисперсия: 0.037\*5\*(4-4,144)^2 + 0,074\*5\*(4-4,144)^2 + 0,037\*8\*(5-4,144)^2 + 0,148 \* (3-4,144)^2 = 0,422

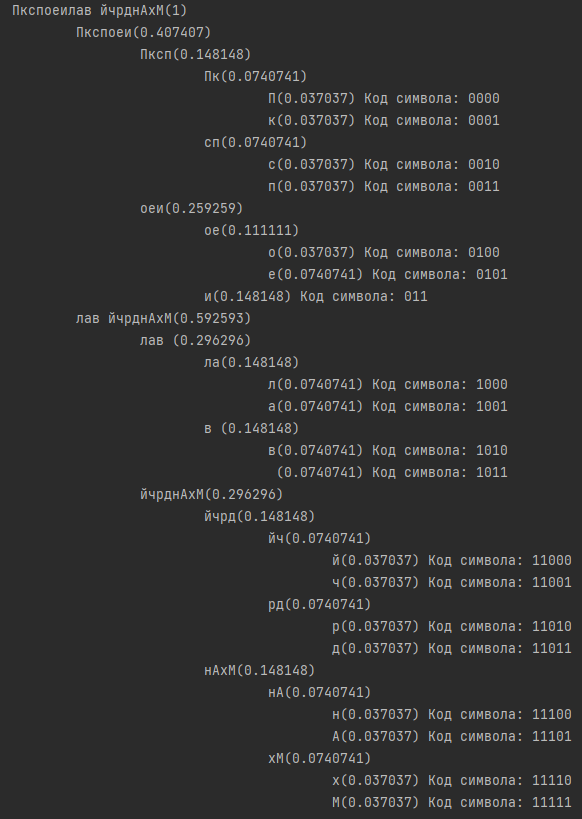


Рисунок 4 – Дерево для метода Хаффмана

# Код программы с комментариями

#include <iostream> #include <string> #include <vector> #include <algorithm> #include <map> #include <Windows.h>

using namespace std;

struct Node { // структура, представляющая собой узел бинарного дерева

string str;

double frequency; // частота появления str в исходной строке

Node\* left = nullptr, \*right = nullptr; // потомки узла

Node(pair<char, double> pair) { string str(1, pair.first); this->str = str;

this->frequency = pair.second;

}

Node(string& str, double frequency) { this->str = str;

this->frequency = frequency;

}

};

struct FrequencyTable {

vector<pair<char, double>> frequencies; // вектор пар символов и чисел двойной точности, служащий таблицей вероятности появления символов

// получение вероятности заданного символа

double getFrequency(char c) {

int index = getFrequencyIndex(c);

return (index == -1) ? NULL : frequencies.at(index).second;

}

// получение индекса символа в таблице

int getFrequencyIndex(char c) {

for (int i = 0; i < frequencies.size(); i++) { if (frequencies.at(i).first == c) {

return i;

}

}

return -1;

}

// добавление новой пары в таблицу

void push(pair<char, double> pair) { frequencies.push\_back(pair);

}

// прибавление единицы к количеству появлений символа

void incrementRepeat(char c) {

int index = getFrequencyIndex(c); frequencies.at(index).second++;

}

// преобразовать количество появлений символов в вероятности (поделить на длину строки)

void convertRepeatsToFrequency(int length) {

for (int i = 0; i < frequencies.size(); i++) { frequencies.at(i).second /= length;

}

}

// сортировка таблицы по убыванию

void sortFrequencies() {

sort(frequencies.begin(), frequencies.end(), [=](std::pair<char, double>& a, std::pair<char, double>& b)

{

return a.second > b.second;

}

);

}

// получение строки из символов исходной строки без повторов

string getUniqueCharsString() { string result = "";

for (auto const& pair : frequencies) { result += pair.first;

}

return result;

}

// получение длины таблицы

int length() {

return frequencies.size();

}

// получение вероятности по индексу записи

pair<char, double> getFreqByIndex(int index) { return frequencies.at(index);

}

};

// дерево для формирования кодов, используемых для сжатия

class EncodingTree { private:

string original; // исходная строка

Node\* root = nullptr; // указатель на корень дерева FrequencyTable frequencies; // таблица вероятностей символов map<char, string> encodingTable; // таблица кодов для сжатия

// построение дерева для метода Шеннона-Фано

void buildTreeSF() {

string unique = frequencies.getUniqueCharsString(); root = new Node(unique, 1);

splitNode(root); // делим каждый узел примерно пополам

}

// построение дерева для метода Хаффмана

void buildTreeHuffman() { vector<Node\*> nodes;

int length = frequencies.getUniqueCharsString().length(); for (int i = 0; i < length; i++) {

nodes.push\_back(new Node(frequencies.getFreqByIndex(i))); //

конвертируем все вероятности в узлы дерева

}

while (nodes.size() > 1) { // пока есть хотя бы 2 узла

Node\* left, \* right;

int index = getMinNodeIndex(nodes); // ищем, сохраняем и удаляем из вектора 2 наименьших по вероятности узла (left и right)

left = nodes.at(index); nodes.erase(nodes.begin() + index); index = getMinNodeIndex(nodes); right = nodes.at(index); nodes.erase(nodes.begin() + index);

string concat = left->str + right->str;

Node\* parent = new Node(concat, left->frequency + right->frequency);

// создаём новый узел, представляющий собой сумму двух наименьших

parent->left = left; // назначаем ему потомками наименьшие узлы

parent->right = right;

nodes.push\_back(parent); // добавляем узел в вектор

}

root = nodes.at(0); // когда остаётся один узел, он становится корнем

дерева

}

// получение минимального узла из вектора узлов

int getMinNodeIndex(vector<Node\*> v) { double minimum = DBL\_MAX;

int index = 0;

for (int i = 0; i < v.size(); i++) {

if (v.at(i)->frequency < minimum) { minimum = v.at(i)->frequency; index = i;

}

}

return index;

}

// деление узла примерно пополам (для Шеннона-Фано)

void splitNode(Node\* node) {

if (node->str.length() <= 1) { // если узел пустой или из 1 символа, делить

не нужно

}

return;

int mid = findMiddle(node->str); // ищем индекс для деления

string leftSubstr = node->str.substr(0, mid + 1), rightSubstr = node-

>str.substr(mid + 1); // части строк до и после середины

node->left = new Node(leftSubstr, sumFrequencies(leftSubstr)); // создание половин и назначение их в качестве потомков

node->right = new Node(rightSubstr, sumFrequencies(rightSubstr)); splitNode(node->left); // рекурсивное деление половин splitNode(node->right);

}

// поиск индекса половины для деления узла примерно пополам

int findMiddle(string& str) { int mid = 0;

double beforeSum = 0, afterSum = sumFrequencies(str), minDiff = DBL\_MAX; //

сумма вероятностей до середины, после середины и их минимальная разница

for (int i = 0; i < str.length(); i++) {

beforeSum += frequencies.getFrequency(str.at(i)); // меняем суммы до и после текущей середины

afterSum -= frequencies.getFrequency(str.at(i));

if (abs(beforeSum - afterSum) < minDiff) { // если разница меньше текущей минимальной, сохраняем индекс и новый минимум

mid = i;

minDiff = abs(beforeSum - afterSum);

}

}

return mid;

}

// сумма вероятностей всех символов строки

double sumFrequencies(string& str) { double result = 0;

for (int i = 0; i < str.length(); i++) {

result += frequencies.getFrequency(str.at(i));

}

return result;

}

// рекурсивный вывод ветки дерева в консоль

void printBranch(Node\* node, int tabOffset) { if (!node) return;

for (int i = 0; i < tabOffset; i++) { cout << '\t';

}

cout << node->str << '(' << node->frequency << ')'; if (node->str.length() == 1) {

cout << " Код символа: " << encodingTable[node->str.at(0)];

}

cout << endl;

printBranch(node->left, tabOffset + 1); printBranch(node->right, tabOffset + 1);

}

// подсчёт вероятностей символов

void calculateFrequencies() {

for (int i = 0; i < original.length(); i++) {

if (frequencies.getFrequencyIndex(original.at(i)) == -1) { // если символа в таблице ещё нет, добавляем его

frequencies.push(make\_pair(original.at(i), 0));

}

frequencies.incrementRepeat(original.at(i)); // прибавляем единицу

}

frequencies.convertRepeatsToFrequency(original.length()); // конвертируем кол-во повторов в вероятности

frequencies.sortFrequencies(); // сортировка по убыванию

}

// построение таблицы кодов

void buildEncodingTable(Node\* node, string code) { if (!node) return;

if (node->str.length() == 1) { // если строка узла состоит из одного

символа

encodingTable.insert(make\_pair(node->str.at(0), code)); // добавляем

в таблицу кодов код этого символа

return;

}

buildEncodingTable(node->left, code + "0"); // иначе рекурсивно проходимся по левому и правому поддереву

buildEncodingTable(node->right, code + "1");

}

// закодировать строку используя таблицу кодов

string encode() {

string result = "";

for (int i = 0; i < original.length(); i++) { result += encodingTable[original.at(i)];

}

return result;

}

public:

EncodingTree(string& original) { this->original = original; calculateFrequencies();

}

void printTree() {

if (!root) return; printBranch(root, 0);

}

string encodeHuffman() { buildTreeHuffman(); buildEncodingTable(root, ""); return encode();

}

string encodeShannonFano() { buildTreeSF(); buildEncodingTable(root, ""); return encode();

}

// раскодирование строки

string decode(string encoded) {

string decoded = "", currentPattern = ""; for (int i = 0; i < encoded.length(); i++) {

currentPattern += encoded.at(i); // посимвольно наращиваем текущий

код символа раскодировать

char decodedChar = decodeChar(currentPattern); // пытаемся

if (decodedChar != '\0') { // если раскодировано успешно, прибавляем

к результату раскодированный символ и обнуляем текущий код

decoded += decodedChar; currentPattern = "";

}

}

return decoded;

}

// расшифровка символа

char decodeChar(string& encoded) {

for (auto pair : encodingTable) { if (pair.second == encoded) {

return pair.first;

}

}

return '\0'; // если расшифровка не удалась, возвращаем \0

}

};

int main() {

SetConsoleOutputCP(1251);

string input = "Тише, мыши, кот на крыше, А котята ещё выше. Кот пошёл за молоком, А котята кувырком.";

EncodingTree tree(input);

string encoded = tree.encodeShannonFano();

cout << endl << endl << "Закодированная строка (Шеннон-Фано): " << encoded << endl;

cout << "Результат декодирования: " << tree.decode(encoded) << endl << endl; tree.printTree();

input = "Бойко Денис Алексеевич"; EncodingTree tree2(input);

encoded = tree2.encodeHuffman();

cout << "Закодированная строка (Хаффман): " << encoded << endl;

cout << "Результат декодирования: " << tree2.decode(encoded) << endl; tree2.printTree();

tree2.printTree();

}

# Вывод

В результате выполнения данной работы были освоены навыки работы с алгоритмами сжатия Лемпеля-Зива LZ77 и LZ78, Шеннона-Фано и Хаффмана. Также были написаны программы кодирования и раскодирования текста методами Хаффмана и Шеннона-Фано и проведено их тестирование. Сравнение их эффективности с алгоритмом сжатия rar файлов показало, что разработанные программы эффективнее, чем rar. Как итог, мной был получен практический опыт использования и реализации различных алгоритмов сжатия текста и других данных.