МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Демонстрация статического кодирования и декодирования текстового документа методами Хаффмана и Фано-Шеннона

Студент гр. 9304 _____ Атаманов С. Д.

Преподаватель _____ Филатов А. Ю.

Санкт-Петербург

2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Атаманов С. Д.

Группа 9304

Тема работы: Демонстрация статического кодирования и декодирования текстового документа методами Хаффмана и Фано-Шеннона

Исходные данные:

Для работы программы требуются компилятор языка C++, поддерживающий C++17 стандарт

Содержание пояснительной записки:

- Аннотация
- Содержание
- Введение
- Формальная постановка задачи
- Описание алгоритма
- Описание структур данных
- Описание функций
- Описание интерфейса пользователя
- Тестирование
- Заключение
- Список использованных источников
- Исходный код программы

Предполагаемый объем пояснительной записки:

Не менее 27 страниц.

Дата выдачи задания: 23.11.2020

Дата сдачи реферата: 28.12.2020

Дата защиты реферата: 28.12.2020

Студент _____ Атаманов С. Д.

Преподаватель _____ Филатов А. Ю.

АННОТАЦИЯ

Для выполнения курсовой работы была разработана программа на языке программирования C++, которая осуществляет демонстрацию алгоритмов статического кодирования Хаффмана и Фано-Шеннона.

SUMMARY

To perform the course work, a program was developed in the C++ programming language, which demonstrates the Huffman and Fano-Shannon static coding algorithms.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Алгоритмы сжатия данных представляют собой алгоритмические преобразования данных, производимое с целью уменьшения занимаемого им объёма. В данной работе будут рассмотрены два статических алгоритма: Хаффмана и Фано-Шеннона.

Цель работы: реализовать алгоритмы сжатия, используя язык программирования C++. Осуществить демонстрацию работы алгоритмов на примере текстового документа, закодировав и декодировав его обратно.

1. ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Реализовать визуализацию статического кодирования и декодирования Хаффмана и Шано-Феннона.

Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий.

2. ОПИСАНИЕ АЛГОРИТМА

Программа реализует алгоритмы кодирования Хаффмана и Фано-Шеннона.

Фано-Шеннон:

Работа алгоритма основывается на Бинарном дереве. Алгоритм получает таблицу символов с частотой их появления в тексте(далее — весом). Все символы складываются в единую строку, их веса складываются. В узлы дерева записываются отдельные символы или строка состоящая из них. Затем, если это строка, она разделятеся по весу напополам, в левое поддерево записывается строка с меньшим весом, в правое с большим. Так продолжается до тех пор, пока в листьях не будут отдельные символы. После по дереву проходятся, добавляя к каждому коду левого поддерева каждого узла 0, а к правому 1. В результате у каждого символа получается уникальный код который тем короче, чем чаще встречается символ и наоброт, тем длинее, чем символ встречается реже.

Хаффман:

Работа алгоритма также основывается на Бинарном дереве. Алгоритм получает на вход строку с весом всех символов в тексте, а также таблицу символов с их весом. Далее заполняется вектор строк, помещая каждый символ в отдельную ячейку. Затем берутся первые два элемента вектора, объединяются, создавая узел, с помощью отдельной функции. Данный узел затем помещается в отдельный вектор. Объединенный узел помещается обратно в исходный вектор. Данный вектор потом отсортировывается по весам. Данные действия повторяются до тех пор, пока в исходном векторе не останется 1 узла. В результате получается бинарное дерево, узлами которого являются строки с их весами, а листьями – отдельные символы с их весами. Чтобы получить коды символов, по дереву проходятся, добавляя к каждому коду левого поддерева каждого узла 0, а к правому 1. В результате у каждого символа получается уникальный код который тем короче, чем чаще встречается символ и наоброт, тем длинее, чем символ встречается реже.

3. ОПИСАНИЕ СТРУКТУР ДАННЫХ

1. Класс BinTreeNode – узел бинарного дерева.

Класс содержит следующие публичные поля: std::shared_ptr<BinTreeNode> left и right, которые хранят умный указатель на левое и правое поддерево, std::pair<std::string, int> data, которое хранит значение узла в паре строка – её вес. Также класс содержит публичные методы getShannonTree(), getHuffmanTree(), которые позволяют получить соответственно дерево Фано-Шеннона и Хаффмана. Метод makeNode() позволяет создать узел BinTreeNode из различных входных данных. Структура класса представлена на рисунке 1.

```
class BinTreeNode {
public:
    std::shared_ptr <BinTreeNode> left{nullptr};
    std::shared_ptr <BinTreeNode> right{nullptr};
    std::pair<std::string, int> data;

    std::shared_ptr <BinTreeNode>
    getShannonFanoTree(std::pair<std::string, int> stringWithWeight, std::map<char, int> usingSymbols,
                       std::map<char, std::string> &codes, std::string code);

    std::shared_ptr <BinTreeNode>
    getHuffmanTree(std::pair<std::string, int> stringWithWeight, std::map<char, int> usingSymbols);

    std::shared_ptr <BinTreeNode>
    makeNode(std::shared_ptr <BinTreeNode> leftNode, std::shared_ptr <BinTreeNode> rightNode,
             std::pair<std::string, int> leftFutureNode, std::pair<std::string, int> rightFutureNode);
}
```

Рисунок 1 – Структура класса BinTreeNode

4. ОПИСАНИЕ ФУНКЦИЙ

Функция HuffmanComparator() – используется для сортировки вектора в качестве компаратора.

Функция getListOfElem() – позволяет получить словарь символов, используемых в кодируемом тексте вместе с их весами.

Функция getStringWithWeigh() – позволяет получить из словаря символов цельную, лексикографически отсортированную строку, вместе с её весом.

Функция getCodesFromHuffman() – позволяет получить словарь с кодировкой каждого символа текста.

Функция printTree() – позволяет распечатать деревья как Шано-Феннона, так и Хаффмана.

Функция HuffmanStringComparator() – используется для сортировки строки в качестве компаратора.

Функция printHuffman() – распечатывает на экран ход алгоритма Хаффмана.

Функция printCodeTable() – распечатывает на экран таблицу символов.

Функция getCodesFromFile() – берет из текстового документа символы и их коды для декодирования.

Функция encode() – выполняет кодирование, вывод демонстрационной информации на экран и декодирование.

5. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Программа принимает на вход строку, которую нужно закодировать. Строка должна содержать символы ASCII таблицы. Количество символов: больше 1.

После ввода строки программа выполняет кодирование и декодирование двумя способами: методом Хаффмана и методом Шано-Феннона. Демонстрационная информация выводится на экран и сопровождается пояснениями. Более никаких действий от пользователя не требуется.

6. ТЕСТИРОВАНИЕ

Для тестирования были выбраны несколько предложений, которые будут закодированы. Для ух удобного прогона был написан bash script.

Пример работы программы представлен на рисунке 2.

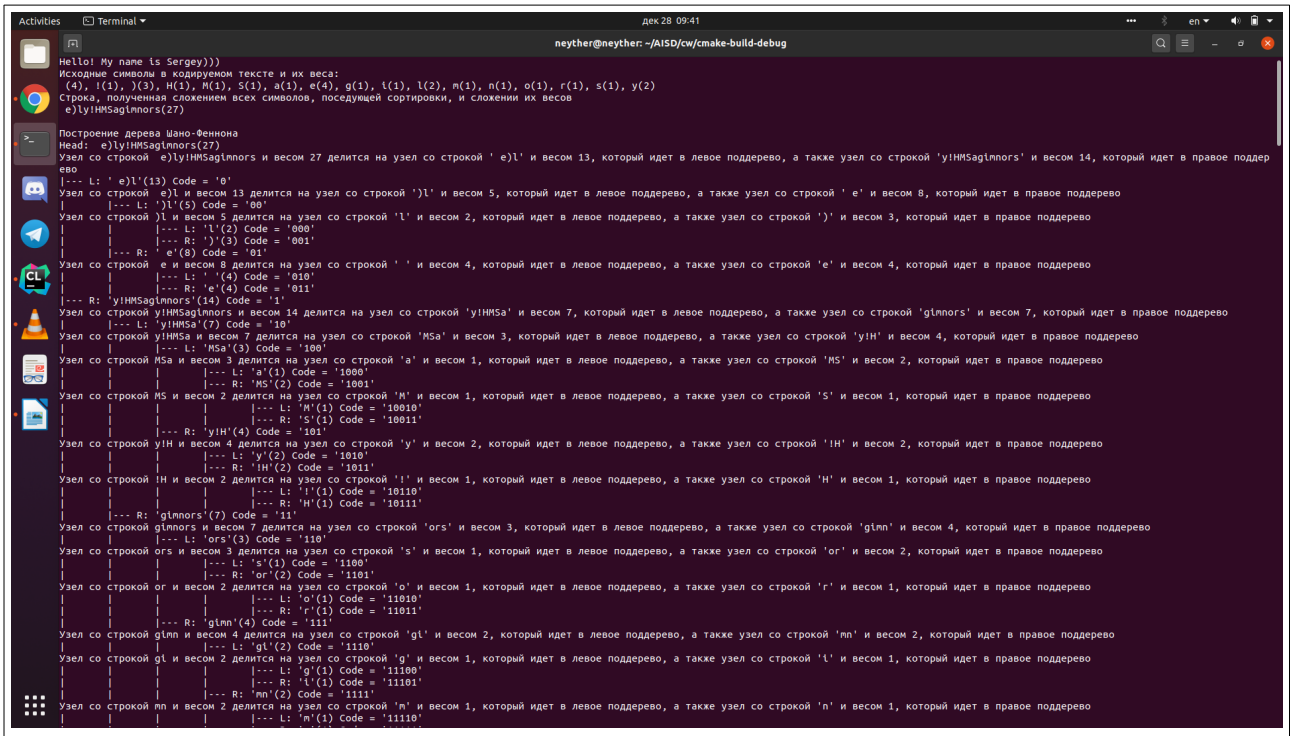

Рисунок 2 — Пример работы программы

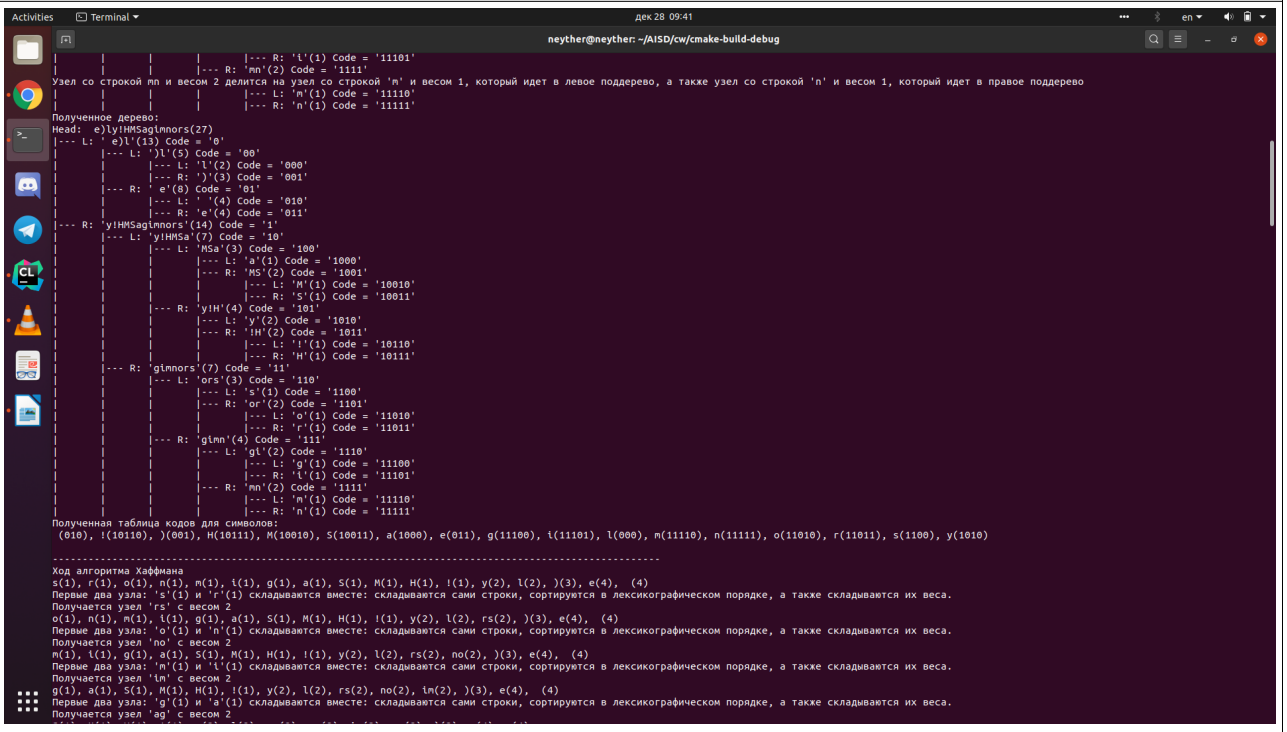

Рисунок 3 — Пример работы программы

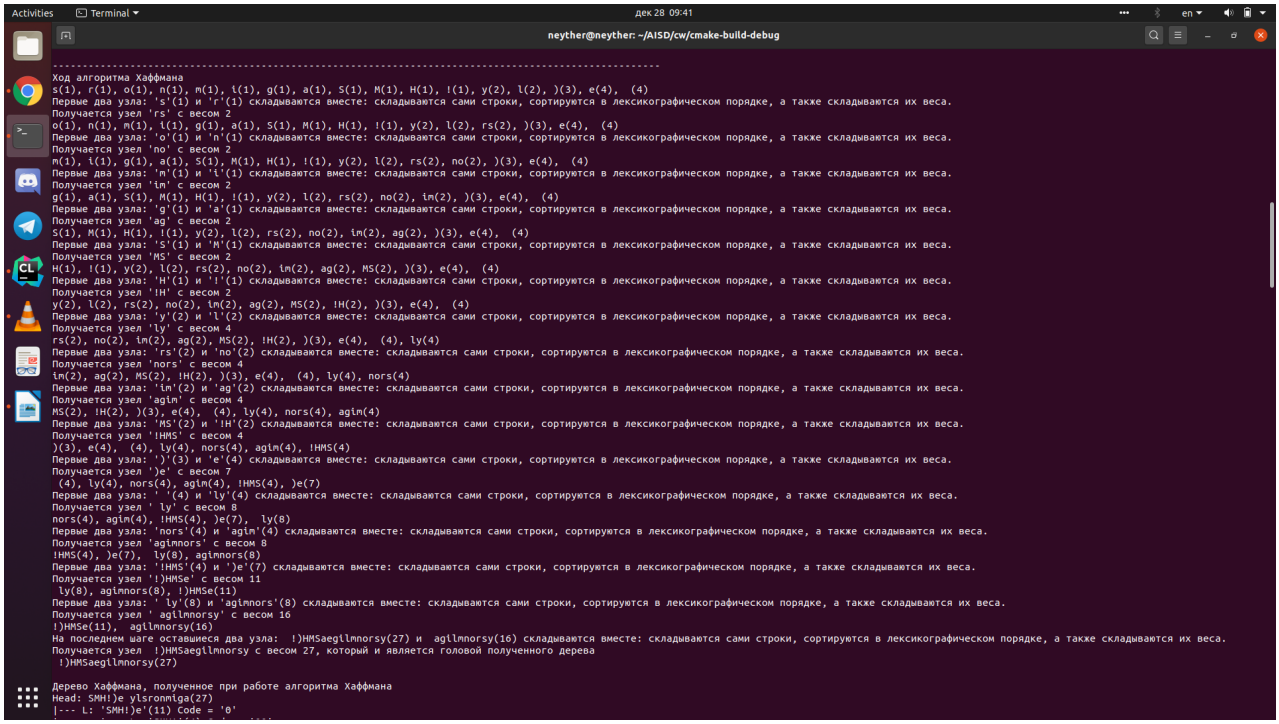

Рисунок 4 — Пример работы программы

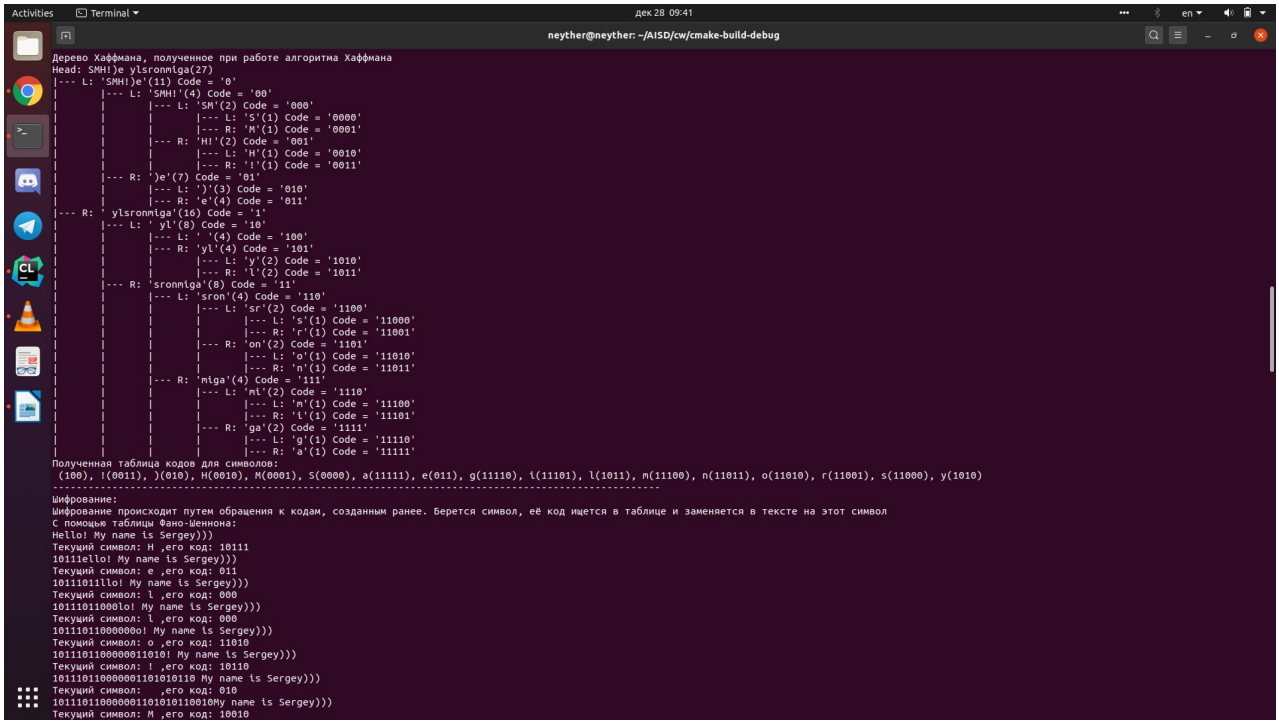

Рисунок 5 — Пример работы программы

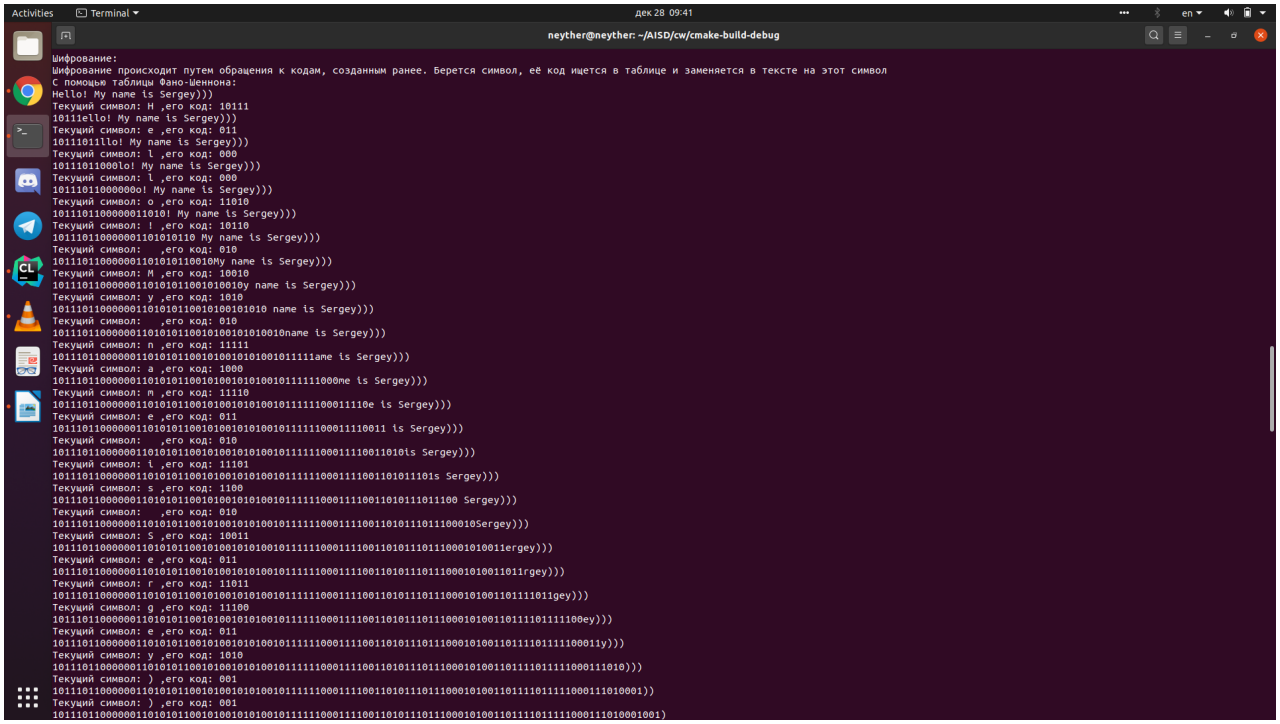

Рисунок 6 — Пример работы программы

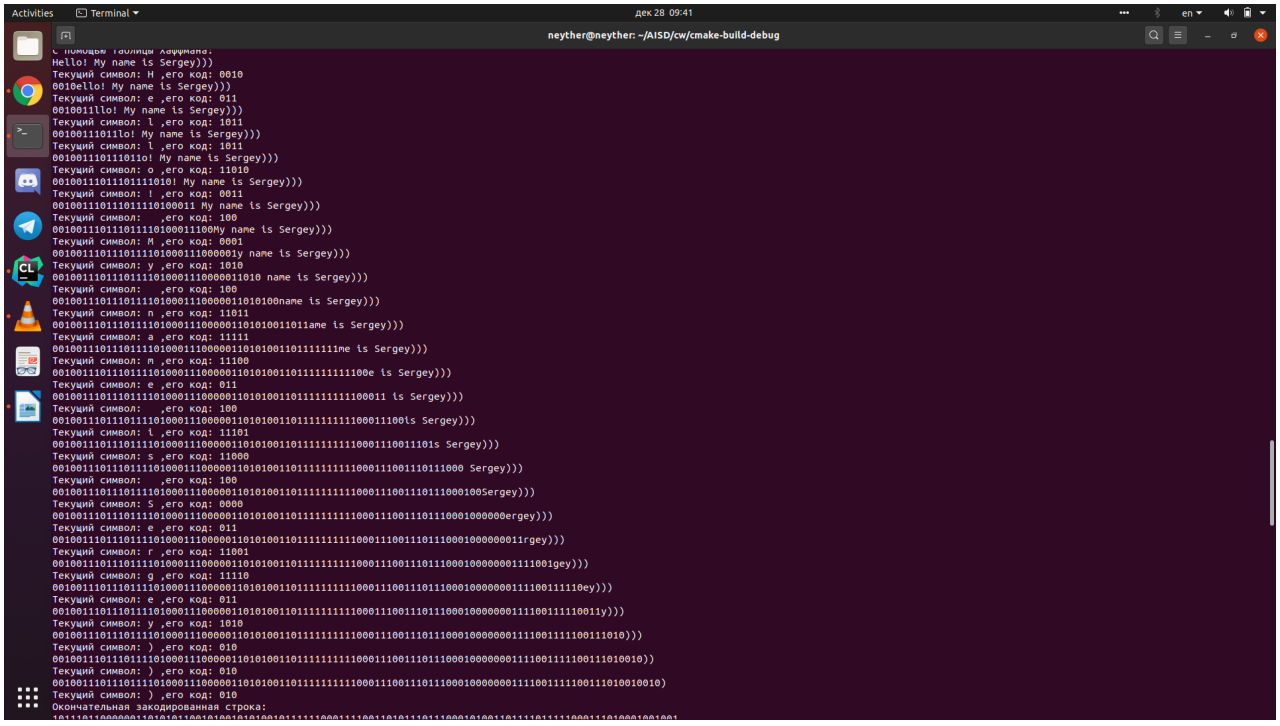

Рисунок 7 — Пример работы программы

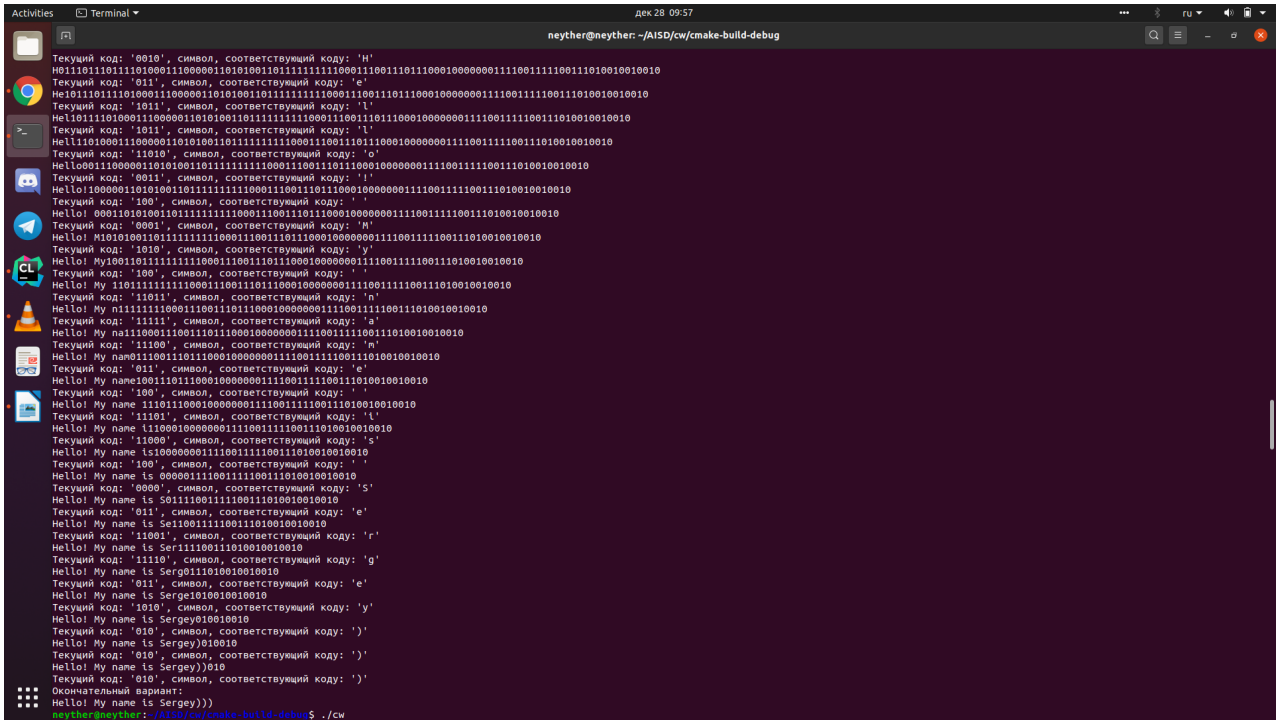

Рисунок 8 — Пример работы программы

Результаты тестирования представлены в таблице 1.

Таблица 1. Тестовые данные

Входные данные	Описание теста
Hello!	Короткий текст
My name is Sergey)	Текст подлиннее
Three pizzas, 4 cheeses... Is that seven cheese turns out?	Текст из двух предложений

ЗАКЛЮЧЕНИЕ

Алгоритм кодирования Хаффмана является одним из первых алгоритмов эффективного кодирования информации. Не смотря на его возраст, он до сих пор используется при кодировании, к примеру при сжатии фото- и видеоизображений (JPEG, MPEG).

Алгоритм Фано-Шеннона является одним из первых алгоритмов сжатия. Он имеет большое сходство с алгоритмом Хаффмана. Главной его идеей является замена часто встречающиеся символы более короткими кодами, а редко встречающиеся – более длинными кодами. Код является префиксным, что позволяет однозначно определить один закодированный символ, код которого не является префиксом любого другого кода.

В ходе выполнения работы была разработана программа на языке программирования C++, которая реализует алгоритмы Хаффмана и Фано-Шеннона, а также выполняет демонстрацию работы этих алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Код Хаффмана. URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0
2. Алгоритм Шеннона-Фано. URL: https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A8%D0%B5%D0%BD%D0%BD%D0%BE%D0%BD%D0%B0_%E2%80%94_%D0%A4%D0%B0%D0%BD%D0%BE
3. Алгоритм Хаффмана. URL: https://habr.com/ru/post/144200/

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp

```
#include "functions.h"

int main(int argc, char* argv[]){
	std::string encodeString;
	std::string encodedShannon, encodedHuffman;
	std::shared_ptr<BinTreeNode> shannonTree, huffmanTree;
	std::getline(std::cin, encodeString);

	auto iterBeg = encodeString.begin();
	encode(iterBeg, encodedShannon, encodedHuffman, shannonTree, huffmanTree);
    return 0;
}
```

Файл function.cpp

```
#include "functions.h"

static bool HuffmanComparator(std::pair<std::string, int> a1, std::pair<std::string, int> a2){
    if(a1.second >= a2.second)
        return false;
    else
        return true;
}

void getListOfElem(std::map<char, int>& map, std::string::iterator iterator){ // Мапа используемых символов
    if(*iterator == '\0')
        return;
    while(*iterator != '\0') {
        if (map.find(*iterator) != map.end()) {
            map[*iterator]++;
        } else {
            map.insert({*iterator, 1});
        }
        iterator++;
    }
}

void getStringWithWeigh(std::map<char, int> stringMap, std::pair<std::string, int>& weightString){ // Строка с весами(отсортированная)
    auto iterBeg = stringMap.begin();
    auto iterEnd = stringMap.end();
    std::pair<char, int> max;
    std::map<char, int> finder;
    int flag;
    while(1){
        flag = 0;
        for(;iterBeg != iterEnd;iterBeg++){
            if(iterBeg->second > max.second && finder.find((*iterBeg).first) == finder.end()){
                max.first = (*iterBeg).first;
                max.second = (*iterBeg).second;
                flag = 1;
            }
        }
        iterBeg = stringMap.begin();
        if(flag == 1){
```

```
            weightString.first += max.first;
            weightString.second += max.second;
            finder.insert({max.first, max.second});
            max.second = 0;
            continue;
        }
        break;
    }
}

void getCodesFromHuffman(std::shared_ptr<BinTreeNode> huffmanTree, std::map<char,
std::string>& codesHuffman, std::string code){//Заполнение мапы кодов Хаффмана
    if(huffmanTree == nullptr){
        return;
    }
    else{
        if(huffmanTree->data.first.length() == 1)
            codesHuffman.insert({huffmanTree->data.first[0], code});
        else{
            getCodesFromHuffman(huffmanTree->left, codesHuffman, code+'0');
            getCodesFromHuffman(huffmanTree->right, codesHuffman, code+'1');
        }
    }
}

void printTree(std::shared_ptr<BinTreeNode> tree, int level, std::string code, bool debug, bool
left){
    if(level == 0)
        std::cout << "Head: " << tree->data.first << "(" << tree->data.second << ")" << "\n";
    else {
        for (int i = 1; i < level; i++) {
            std::cout << "|\t";
        }
        if(left)
            std::cout << "|--- L: \'" << tree->data.first << "\'(" << tree->data.second << ") Code =
\'" << code << "\'" << "\n";
        else
            std::cout << "|--- R: \'" << tree->data.first << "\'(" << tree->data.second << ") Code
= \'" << code << "\'" << "\n";
    }
    if(debug){
        if(tree->left && tree->right) {
            std::cout << "Узел со строкой " << tree->data.first << " и весом " << tree-
>data.second;
            std::cout << " делится на узел со строкой \'" << tree->left->data.first << "\' и весом
" << tree->left->data.second << ", который идет в левое поддерево";
            std::cout << ", а также узел со строкой \'" << tree->right->data.first << "\' и весом "
<< tree->right->data.second << ", который идет в правое поддерево\n";
        }
    }
    if (tree->left)
        printTree(tree->left, level + 1, code + '0', debug, true);
    if (tree->right)
        printTree(tree->right, level + 1, code + '1', debug, false);
}


static bool HuffmanStringComparator(char a1, char a2){
    if(a1 >= a2)
        return false;
    else
```

```
        return true;
}

void printHuffman(std::pair<std::string, int> stringWithWeight, std::map<char, int>
usingSymbols, bool debug){
    std::cout << "\n-----------------------------------------------------------------------------------------------------------\n";
    std::cout << "Ход алгоритма Хаффмана\n";
    std::pair<std::string, int> temp;
    std::vector<std::pair<std::string, int>> huffmanString;
    for(int i=1;i<=stringWithWeight.first.length();i++){
        temp = std::make_pair(stringWithWeight.first[stringWithWeight.first.length()-i],
usingSymbols[stringWithWeight.first[stringWithWeight.first.length()-i]]);
        huffmanString.push_back(temp);
    }
    while(huffmanString.size() != 1){
        temp = std::make_pair(huffmanString[0].first+huffmanString[1].first,
huffmanString[0].second+huffmanString[1].second);
        std::sort(temp.first.begin(), temp.first.end(), HuffmanStringComparator);

        for(int i=0;i<huffmanString.size();i++){
            if(i == huffmanString.size() - 1)
                std::cout << huffmanString[i].first << "(" << huffmanString[i].second << ")\n";
            else
                std::cout << huffmanString[i].first << "(" << huffmanString[i].second << "), ";
        }
        if(debug && huffmanString.size() != 2){
            std::cout << "Первые два узла: \'" << huffmanString[0].first << "\'(" <<
huffmanString[0].second << ") и \'" << huffmanString[1].first << "\'(" <<
huffmanString[1].second << ")";
            std::cout << " складываются вместе: складываются сами строки, сортируются в
лексикографическом порядке, а также складываются их веса.\n";
            std::cout << "Получается узел \'" << temp.first << "\' с весом " << temp.second <<
"\n";
        }
        if(huffmanString.size() == 2) {
            huffmanString.erase(huffmanString.begin(), huffmanString.begin() + 2);
            huffmanString.push_back(temp);
            std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator);
            break;
        }
        else {
            huffmanString.erase(huffmanString.begin(), huffmanString.begin() + 2);
            huffmanString.push_back(temp);
            std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator);
        }
    }
    if(debug){
        std::cout << "На последнем шаге оставшиеся два узла: " << huffmanString[0].first <<
"(" << huffmanString[0].second << ") и " << huffmanString[1].first << "(" <<
huffmanString[1].second << ")";
        std::cout << " складываются вместе: складываются сами строки, сортируются в
лексикографическом порядке, а также складываются их веса.\n";
        std::cout << "Получается узел " << temp.first << " с весом " << temp.second << ",
который и является головой полученного дерева\n";
    }
    std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator);
    std::cout << huffmanString[0].first << "(" << huffmanString[0].second << ")\n\n";
}

void printCodeTable(std::map<char, std::string> codes){
    auto checker = codes.end();
```

```
    checker--;
    for(auto i = codes.begin();i != codes.end(); i++) {
        if (i == checker)
            std::cout << i->first << "(" << i->second << ")\n";
        else
            std::cout << i->first << "(" << i->second << "), ";
    }
}

std::map<std::string, char> getCodesFromFile(const std::string& fileName){
    std::map<std::string , char> codes;
    char symbol;
    std::string temp;
    std::string code;
    auto iterBeg = temp.begin();
    std::ifstream file;
    file.open(fileName);
    if(!file.is_open()){
        std::cout << "Error: Undeclared codeFile";
        exit(EXIT_FAILURE);
    }
    while(!file.eof()){
        std::getline(file, temp);
        iterBeg = temp.begin();
        symbol = *iterBeg;
        iterBeg += 2;
        while(*iterBeg != ';') {
            code += *iterBeg;
            iterBeg++;
        }
        codes.insert({code, symbol});
        code.clear();
        temp.clear();
    }
    file.close();
    return codes;
}

void encode(std::string::iterator encodeString, std::string& shannonString, std::string&
huffmanString, std::shared_ptr<BinTreeNode>& shannonTree, std::shared_ptr<BinTreeNode>&
huffmanTree) {
    std::map<char, int> map;//Мапа с используемыми символами и их весами
    std::map<char, std::string> codesShannon, codesHuffman; //Коды для расшифровки
    std::pair<std::string, int> stringWithWeight; //Строка с общим весом символов
    std::string code; //Служебная переменная для передачи в функцию
    std::string outputShannon, outputHuffman; //Строка для вывода

    //Получение деревьев и файлов с кодами
    getListOfElem(map, encodeString);
    getStringWithWeigh(map, stringWithWeight);
    shannonTree = shannonTree->getShannonFanoTree(stringWithWeight, map, codesShannon,
code);
    huffmanTree = huffmanTree->getHuffmanTree(stringWithWeight, map);
    getCodesFromHuffman(huffmanTree, codesHuffman, code);

    //Вывод деревьев и "пояснения за алгоритмы
    //-------------------------------------------------------------------------------------------------------------
    std::cout << "Исходные символы в кодируемом тексте и их веса:\n";
    auto checker = map.end();
    checker--;
    for (auto i = map.begin(); i != map.end(); i++) {
```

```
        if (i == checker)
            std::cout << i->first << "(" << i->second << ")\n";
        else
            std::cout << i->first << "(" << i->second << "), ";
    }
    std::cout << "Строка, полученная сложением всех символов, поседующей сортировки,
и сложении их весов\n";
    std::cout << stringWithWeight.first << "(" << stringWithWeight.second << ")\n\n";
    std::cout << "Построение дерева Шано-Феннона\n";
    printTree(shannonTree, 0, code, true, false);
    std::cout << "Полученное дерево:\n";
    printTree(shannonTree, 0, code, false, false);
    std::cout << "Полученная таблица кодов для символов:\n";
    printCodeTable(codesShannon);
    printHuffman(stringWithWeight, map, true);
    std::cout << "Дерево Хаффмана, полученное при работе алгоритма Хаффмана\n";
    printTree(huffmanTree, 0, code, false, false);
    std::cout << "Полученная таблица кодов для символов:\n";
    printCodeTable(codesHuffman);
    std::cout
            << "-----------------------------------------------------------------------------------------------------------\n";
    //-----------------------------------------------------------------------------------------------------------

    std::cout << "Шифрование:\n";
    std::cout
            << "Шифрование происходит путем обращения к кодам, созданным ранее.
Берется символ, её код ищется в таблице и заменяется в тексте на этот символ\n";
    std::string demonstrate;
    std::cout << "С помощью таблицы Фано-Шеннона:\n";
    auto temp = encodeString;
    auto save = encodeString;
    for (; *encodeString != '\0'; encodeString++) {
        temp = encodeString;
        for (; *temp != '\0'; temp++)
            demonstrate += *temp;
        std::cout << demonstrate << "\n";
        std::cout << "Текущий символ: " << *encodeString << " ,его код: " <<
codesShannon[*encodeString] << "\n";
        outputShannon += codesShannon[*encodeString];
        demonstrate.clear();
        demonstrate += outputShannon;
    }

    std::cout << "\nС помощью таблицы Хаффмана:\n";
    encodeString = save;
    demonstrate.clear();
    for (; *encodeString != '\0'; encodeString++) {
        temp = encodeString;
        for (; *temp != '\0'; temp++)
            demonstrate += *temp;
        std::cout << demonstrate << "\n";
        std::cout << "Текущий символ: " << *encodeString << " ,его код: " <<
codesHuffman[*encodeString] << "\n";
        outputHuffman += codesHuffman[*encodeString];
        demonstrate.clear();
        demonstrate += outputHuffman;
    }
    std::cout << "Окончательная закодированная строка:\n";
    std::cout << outputShannon << "\n";
    std::cout
            << "-----------------------------------------------------------------------------------------------------------\n";
```

```
    //Формирование файлов с кодами для декодирования
    std::ofstream outputShannonFile, outputHuffmanFile;
    outputShannonFile.open("./CodesShannon.txt");
    outputHuffmanFile.open("./CodesHuffman.txt");
    auto iterBeg = codesShannon.begin();
    for (; iterBeg != codesShannon.end(); iterBeg++)
        outputShannonFile << iterBeg->first << ":" << iterBeg->second << ";\n";
    iterBeg = codesHuffman.begin();
    for(;iterBeg != codesHuffman.end(); iterBeg++)
        outputHuffmanFile << iterBeg->first << ":" << iterBeg->second << ";\n";
    outputShannonFile.close();
    outputHuffmanFile.close();

    std::cout << "Декодирование:\n";
    std::map<std::string, char> decodesShannon = getCodesFromFile("./CodesShannon.txt");
    std::map<std::string, char> decodesHuffman = getCodesFromFile("./CodesHuffman.txt");
    std::cout << "Декодирование происходит в порядке, обратном кодированию: код
заменяется соответствующим символом\n";

    std::cout << "С помощью таблицы Шеннона-Фано:\n";
    auto decodeIter = outputShannon.begin();
    demonstrate.clear();
    std::string output;
    for (; decodeIter != outputShannon.end(); decodeIter++) {
        temp = decodeIter;
        for(;*temp != '\0'; temp++)
            demonstrate += *temp;
        while (true) {
            code += *decodeIter;
            if (decodesShannon.find(code) != decodesShannon.end())
                break;
            else {
                decodeIter++;
                continue;
            }
        }
        std::cout << demonstrate;
        std::cout << "\nТекущий код: \'" << code << "\', символ, соответствующий коду: \'"
<< decodesShannon[code] << "\'\n";
        output += decodesShannon[code];
        demonstrate.clear();
        demonstrate += output;
        code.clear();
    }
    std::cout << "Окончательный вариант:\n" << output << "\n";

    std::cout << "\nС помощью таблицы Хаффмена:\n";
    decodeIter = outputHuffman.begin();
    demonstrate.clear();
    output.clear();
    for (; decodeIter != outputHuffman.end(); decodeIter++) {
        temp = decodeIter;
        for(;*temp != '\0'; temp++)
            demonstrate += *temp;
        while (true) {
            code += *decodeIter;
            if (decodesHuffman.find(code) != decodesHuffman.end())
                break;
            else {
                decodeIter++;
```

```
            continue;
        }
      }
      std::cout << demonstrate;
      std::cout << "\nТекущий код: \'" << code << "\', символ, соответствующий коду: \'"
<< decodesHuffman[code] << "\'\n";
      output += decodesHuffman[code];
      demonstrate.clear();
      demonstrate += output;
      code.clear();
   }
   std::cout << "Окончательный вариант:\n" << output << "\n";
```

Файл function.h:

```
#ifndef CW_FUNCTIONS_H
#define CW_FUNCTIONS_H

#include <iostream>
#include <fstream>
#include <string>
#include <memory>
#include <map>
#include <algorithm>
#include <cmath>

#include "BinTreeNode.h"

static bool HuffmanComparator(std::pair<std::string, int> a1, std::pair<std::string, int> a2);

void getListOfElem(std::map<char, int>& map, std::string::iterator iterator);

void getStringWithWeigh(std::map<char, int> stringMap, std::pair<std::string, int>&
weightString);

void getCodesFromHuffman(std::shared_ptr<BinTreeNode> huffmanTree, std::map<char,
std::string>& codesHuffman, std::string code);

void printTree(std::shared_ptr<BinTreeNode> tree, int level, std::string code, bool debug, bool
left);

static bool HuffmanStringComparator(char a1, char a2);

void printHuffman(std::pair<std::string, int> stringWithWeight, std::map<char, int>
usingSymbols, bool debug);

void printCodeTable(std::map<char, std::string> codes);

std::map<std::string, char> getCodesFromFile(const std::string& fileName);

void encode(std::string::iterator encodeString, std::string& shannonString, std::string&
huffmanString, std::shared_ptr<BinTreeNode>& shannonTree, std::shared_ptr<BinTreeNode>&
huffmanTree);


#endif //CW_FUNCTIONS_H
```

Файл BinTreeNode.cpp:

```
#include "BinTreeNode.h"


static bool HuffmanComparator1(std::pair<std::string, int> a1, std::pair<std::string, int> a2){
   if(a1.second >= a2.second)
```

```
        return false;
    else
        return true;
}

std::shared_ptr<BinTreeNode> BinTreeNode::getShannonFanoTree(std::pair<std::string, int>
stringWithWeight, std::map<char, int> usingSymbols, std::map<char, std::string>& codes,
std::string code) {
    std::shared_ptr<BinTreeNode> tree = std::make_shared<BinTreeNode>();
    std::pair<std::string, int> left;
    std::pair<std::string, int> right;
    tree->data = stringWithWeight;
    if (stringWithWeight.first.length() == 1) {
        codes.insert({tree->data.first[0], code});
        return tree;
    }
    while (true) {
        if ((left.second + usingSymbols[stringWithWeight.first[0]]) > (stringWithWeight.second)
&& left.second == 0) {
            right.first += stringWithWeight.first[0];
            right.second += usingSymbols[stringWithWeight.first[0]];
            stringWithWeight.second -= usingSymbols[stringWithWeight.first[0]];
            stringWithWeight.first.erase(0, 1);
            left.first = stringWithWeight.first;
            left.second = stringWithWeight.second;
            break;
        } else if ((left.second + usingSymbols[stringWithWeight.first[0]]) >
(stringWithWeight.second))
            break;
        else {
            left.first += stringWithWeight.first[0];
            left.second += usingSymbols[stringWithWeight.first[0]];
            stringWithWeight.second -= usingSymbols[stringWithWeight.first[0]];
            stringWithWeight.first.erase(0, 1);
        }
    }
    if (right.second == 0) {
        right.first = stringWithWeight.first;
        right.second = stringWithWeight.second;
    }

    if (left.second > right.second)
        std::swap(left, right);
    if (left.second != 0) {
        tree->left = std::make_shared<BinTreeNode>();
        tree->left = getShannonFanoTree(left, usingSymbols, codes, code + '0');
    }
    if (right.second != 0) {
        tree->right = std::make_shared<BinTreeNode>();
        tree->right = getShannonFanoTree(right, usingSymbols, codes, code + '1');
    }
    return tree;
}

std::shared_ptr<BinTreeNode> BinTreeNode::getHuffmanTree(std::pair<std::string, int>
stringWithWeight, std::map<char, int> usingSymbols) {
    //Инициализация переменных
    std::shared_ptr<BinTreeNode> head; //Созданный узел
    std::map<std::string, std::shared_ptr<BinTreeNode>> tempNodes; //Мапа
нераспределенных узлов
    std::map<std::string, int> tempLeft, tempRight; //Мапы левых и правых сыновей
```

```
   std::vector<std::pair<std::string, int>> huffmanString; // Основная строка с весами
Хаффмана
   std::pair<std::string, int> temp, emptyPairLeft, emptyPairRight; // Текущая пара

   //Заполнение строки Хаффмана
   for (int i = 1; i <= stringWithWeight.first.length(); i++) {
      temp = std::make_pair(stringWithWeight.first[stringWithWeight.first.length() - i],
                        usingSymbols[stringWithWeight.first[stringWithWeight.first.length() - i]]);
      huffmanString.push_back(temp);
   }

   //Основной блок
   while (huffmanString.size() != 1) {

      //Если длина первых двух элементов равна по 1
      if (huffmanString[0].first.length() == 1 && huffmanString[1].first.length() == 1) {
         head = makeNode(nullptr, nullptr, huffmanString[0], huffmanString[1]);
         tempNodes.insert({huffmanString[0].first + huffmanString[1].first, head});
         temp = std::make_pair(huffmanString[0].first + huffmanString[1].first,
                           huffmanString[0].second + huffmanString[1].second);
         huffmanString.erase(huffmanString.begin(), huffmanString.begin() + 2);
         huffmanString.push_back(temp);
         std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator1);
      }

         //Если длина первого больше 1, а второго == 1
      else if (huffmanString[0].first.length() > 1 && huffmanString[1].first.length() == 1) {
         head = makeNode(tempNodes[huffmanString[0].first], nullptr, emptyPairLeft,
huffmanString[1]);
         tempNodes.erase(huffmanString[1].first);
         tempNodes.insert({huffmanString[0].first + huffmanString[1].first, head});
         temp = std::make_pair(huffmanString[0].first + huffmanString[1].first,
                           huffmanString[0].second + huffmanString[1].second);
         huffmanString.erase(huffmanString.begin(), huffmanString.begin() + 2);
         huffmanString.push_back(temp);
         std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator1);
      }

         //Если длина первого - 1, а второго больше 1
      else if (huffmanString[0].first.length() == 1 && huffmanString[1].first.length() > 1) {
         head = makeNode(nullptr, tempNodes[huffmanString[1].first], huffmanString[0],
emptyPairRight);
         tempNodes.erase(huffmanString[1].first);
         tempNodes.insert({huffmanString[0].first + huffmanString[1].first, head});
         temp = std::make_pair(huffmanString[0].first + huffmanString[1].first,
                           huffmanString[0].second + huffmanString[1].second);
         huffmanString.erase(huffmanString.begin(), huffmanString.begin() + 2);
         huffmanString.push_back(temp);
         std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator1);
      }

         //Если длина обоих больше 1
      else {
         head = makeNode(tempNodes[huffmanString[0].first],
tempNodes[huffmanString[1].first], emptyPairLeft,
                     emptyPairRight);
         tempNodes.erase(huffmanString[0].first);
         tempNodes.erase(huffmanString[1].first);
         tempNodes.insert({huffmanString[0].first + huffmanString[1].first, head});
         temp = std::make_pair(huffmanString[0].first + huffmanString[1].first,
                           huffmanString[0].second + huffmanString[1].second);
```

```
            huffmanString.erase(huffmanString.begin(), huffmanString.begin() + 2);
            huffmanString.push_back(temp);
            std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator1);
        }
    }
    return head;
}

std::shared_ptr<BinTreeNode> BinTreeNode::makeNode(std::shared_ptr<BinTreeNode>
leftNode, std::shared_ptr<BinTreeNode> rightNode, std::pair<std::string, int> leftFutureNode,
std::pair<std::string, int> rightFutureNode) {
    std::shared_ptr<BinTreeNode> node = std::make_shared<BinTreeNode>();
    std::pair<std::string, int> empty;
    std::string temp;
    if (leftNode && rightNode) {
        node->left = leftNode;
        node->right = rightNode;
        node->data.first = node->left->data.first + node->right->data.first;
        node->data.second = node->left->data.second + node->right->data.second;
        return node;
    }
    if (leftNode && rightFutureNode.second != 0) {
        node->left = leftNode;
        node->right = makeNode(nullptr, nullptr, empty, rightFutureNode);
        node->data.first = node->left->data.first + node->right->data.first;
        node->data.second = node->left->data.second + node->right->data.second;
        return node;
    }
    if (leftFutureNode.second != 0 && rightNode) {
        node->left = makeNode(nullptr, nullptr, leftFutureNode, empty);
        node->right = rightNode;
        node->data.first = node->left->data.first + node->right->data.first;
        node->data.second = node->left->data.second + node->right->data.second;
        return node;
    }
    if (leftFutureNode.second != 0 && rightFutureNode.second != 0) {
        node->left = makeNode(nullptr, nullptr, leftFutureNode, empty);
        node->right = makeNode(nullptr, nullptr, empty, rightFutureNode);
        node->data.first = node->left->data.first + node->right->data.first;
        node->data.second = node->left->data.second + node->right->data.second;
        return node;
    }
    if (leftFutureNode.second != 0) {
        node->data.first = leftFutureNode.first;
        node->data.second = leftFutureNode.second;
        return node;
    }
    if (rightFutureNode.second != 0) {
        node->data.first = rightFutureNode.first;
        node->data.second = rightFutureNode.second;
        return node;
    }
}
```

Файл BinTreeNode.h:

```
#ifndef CW_BINTREENODE_H
#define CW_BINTREENODE_H

#include <iostream>
#include <fstream>
#include <string>
```

```
#include <memory>
#include <map>
#include <algorithm>

class BinTreeNode {
public:
    std::shared_ptr<BinTreeNode> left{nullptr};
    std::shared_ptr<BinTreeNode> right{nullptr};
    std::pair<std::string, int> data;

    std::shared_ptr<BinTreeNode>getShannonFanoTree(std::pair<std::string, int>
stringWithWeight, std::map<char, int> usingSymbols, std::map<char, std::string> &codes,
std::string code);

    std::shared_ptr<BinTreeNode> getHuffmanTree(std::pair<std::string, int> stringWithWeight,
std::map<char, int> usingSymbols);

    std::shared_ptr<BinTreeNode> makeNode(std::shared_ptr<BinTreeNode> leftNode,
std::shared_ptr<BinTreeNode> rightNode, std::pair<std::string, int> leftFutureNode,
std::pair<std::string, int> rightFutureNode);
};

static bool HuffmanComparator1(std::pair<std::string, int> a1, std::pair<std::string, int> a2);

#endif //CW_BINTREENODE_H
```