

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

КУРСОВАЯ РАБОТА
по дисциплине по дисциплине «Алгоритмы и структуры данных»
Тема: Кодирование и декодирование

Студент гр. 9382

Бочаров Г.С.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Бочаров Г.С.

Группа 9382

Тема работы : Кодирование и декодирование

Исходные данные:

Вариант 4. Кодирование и декодирование методами Хаффмана и Фано-Шеннона – исследование

Исследование

Содержание пояснительной записки:

«Содержание», «Введение», «Постановка задачи», «Описание алгоритмов»,
«Описание основных функций и структур данных», «Тестирование»,
«Заключение», «Список использованных источников»)

Предполагаемый объем пояснительной записки:

Не менее 25 страниц.

Дата выдачи задания: 01.09.2020

Дата сдачи реферата: 16.12.2020

Дата защиты реферата: 17.12.2020

Студент

Бочаров Г.С.

Преподаватель

Фирсов М.А.

АННОТАЦИЯ

Курсовая работа представляет собой исследование алгоритмов кодирования и декодирования методами Хаффмана и Фано-Шеннона. Результатом работы является закодированное и декодированное сообщение. Также измеряется время работы алгоритма. В случае кодирования, считается длина закодированного сообщения и сравнивается с длиной сообщения закодированного с помощью кодов одинаковой длины.

SUMMARY

Course work is a study of algorithms for encoding and decoding using the methods of Huffman and Fano-Shannon. The result of the work is a coded and decoded message. The running time of the algorithm is also measured. In the case of encoding, the length of the encoded message is considered and compared with the length of the message encoded using codes of the same length.

СОДЕРЖАНИЕ

	Введение	5
1.	Постановка задачи	6
2.	Описание алгоритмов	7
2.1	Кодирование методом Фано-Шеннона	7
2.2	Кодирование методом Хаффмана	7
2.3	Анализ алгоритма Хаффмана и Фано-Шеннона.	8
3.	Описание основных функций и структур данных	7
4.	Тестирование	10
	Заключение	13
	Список использованных источников	14
	Приложение А. Исходный код программы	15

ВВЕДЕНИЕ

Целью работы является реализация алгоритмов кодирования и декодирования методами Хаффмана и Фано-Шеннона, сравнение алгоритмов, фиксация результатов работы алгоритмов, сопоставление их с теоретическими оценками, накопление статистики.

Алгоритмы Хаффмана и Фано-Шеннона — алгоритмы префиксного кодирования данных. Коды для символов задаются так, что ни один код не является началом другого. Коды символов вычисляются путем построения бинарного дерева, корнем которого является последовательность всех символов уникальных символов в кодируемом сообщении, а листьями сами уникальные символы.

Каждый узел содержит в себе последовательность символов и их частотами, и суммарную частоту всех символов — вес узла.

В алгоритме Фано-Шеннона построение дерева идет от корня к листьям, в алгоритме Хаффмана от листьев к корню.

Алгоритмы Фано-Шеннона и Хаффмана позволяют существенно сократить длину закодированного сообщения, однако для их расшифровки требуется дополнительной информации об исходном сообщении. Кроме того возникает ряд проблем связанных с вырождением бинарного дерева при большом количестве уникальных символов.

1. ПОСТАНОВКА ЗАДАЧИ

- 1.** Реализовать программу выполняющую кодирование и декодирование входных данных одним из предложенных алгоритмов.
- 2.** Протестировать программу на выборке входных данных с фиксацией результата работы программы.
- 3.** Проанализировать полученные результаты, сравнить их с теорией. Сформировать выводы на основе анализа результатов.

2. ОПИСАНИЕ АЛГОРИТМОВ

2.1 Кодирование методом Фано-Шеннона.

Главная идея этого метода - заменить часто встречающиеся символы более короткими кодами, а редко встречающиеся последовательности более длинными кодами. Таким образом, алгоритм основывается на кодах переменной длины. Для того, чтобы декомпрессор впоследствии смог раскодировать сжатую последовательность, коды Шеннона-Фано должны обладать уникальностью, то есть, не смотря на их переменную длину, каждый код уникально определяет один закодированный символ и не является префиксом любого другого кода.

Сам алгоритм можно разбить на несколько частей :

- 1) Считывание уникальных символов и их частот в массив.
- 2) Сортировка массива символов в порядке невозрастания их частот.
- 3) Построение бинарного дерева, корнем которого является структура, содержащая исходный массив символов с их частотами, сумму частот символов этого массива и код узла.
- 4) Рекурсивно для каждого узла создаются правый и левый потомок, которые получаются путем деления массива родительского узла на две части с приблизительно равными суммарными частотами символов. При этом код левого потомка получается из кода родителя путем дописывания в его конец нуля, код правого потомка получается аналогичным образом, с той лишь разницей, что к родительскому коду дописывается единица
- 5) Процесс создания потомков для узлов прекращается, когда в массиве символов данного узла остается 1 символ.

Таким образом, вычислен код каждого символа входного сообщения.

2.2 Кодирование методом Хаффмана

Алгоритм Хаффмана — алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью. В отличие от алгоритма Фано-Шеннона, алгоритм Хаффмана всегда выдает оптимальные коды для каждого

символа. Таким образом, закодированное по алгоритму Хаффмана сообщение имеет длину меньшую или равную длине того же сообщения, закодированного методом Фано-Шеннона.

Алгоритм можно разбить на следующие этапы:

- 1) Создание списка структур, содержащих массив символов, частоту символов, суммарную частоту символов в массиве.
- 2) Сортировка списка в порядке невозрастания.
- 3) Объединение двух элементов списка с наименьшими суммарными частотами символов следующим образом: создается новая структура, содержащая в себе символы двух старых. При этом старые структуры становятся левым и правым потомком новой и удаляются из списка.
- 4) Пункты 2 и 3 повторяются до тех пор, пока в списке не останется 1 элемент, указывающий на корень бинарного дерева, листьями которого являются кодируемые символы.
- 5) Далее, поиском в глубину, каждому узлу дерева присваивается код.

Таким образом, вычисляются коды для всех кодируемых символов.

2.3 Анализ алгоритма Хаффмана и Фано-Шеннона.

Преимуществом этих двух алгоритмов по сравнению с равномерным кодированием является то, что закодированное сообщение как правило получается короче, а в худшем случае не длиннее сообщения закодированное равномерным кодом.

При этом алгоритм Фано-Шеннона иногда выдает неоптимальные по длине коды. Связано это с тем, что разбиение массива символов на две приблизительно равные части от случая к случаю может происходить по-разному. Поэтому, предпочтительнее использовать алгоритм Хаффмана.

Существенных минусов у этих алгоритмов несколько:

1) При декодировании мы не знаем длину каждого кода, помимо закодированного сообщения и алфавита для декодирования нужно передвать дополнительную информацию об исходном сообщении (Например таблицу частот символов).

2) При кодировании таким способом, длина кода или вес некоторых узлов может стать слишком большой и не помещаться в ячейку для его хранения.

3. ОПИСАНИЕ ОСНОВНЫХ ФУНКЦИЙ И СТРУКТУР ДАННЫХ

`struct Symbol {char value_; int weight_; std::string code_;}` - `value_` - значение символа, `weight_` - кол-во повторений символа, `code_` - код символа. Структура хранит параметры символа.

`struct Symbols {std::vector<Symbol> values_;int weight_;}` -`values_` - массив структур `Symbol`, `weight_` - суммарное кол-во повторений элементов в массиве `values`. Структура хранит параметры массива символов.

`struct Node {Symbols symbols_;std::string code_;Node *left_; Node *right_;}`
`symbols` — массив символов с их параметрами, `code_` - код структуры, `left_` , `right_` указатели на левого и правого потомка соответственно. Структура используется для хранения узла бинарного дерева.

`template<typename StreamT>`

`std::string readSymbol(StreamT &in, std::vector<Symbol> &atoms)` — Функция принимает на вход поток ввода, и массив структур типа `Symbol`. Функция считывает символы и заносит их в массив структур.

`template<typename StreamT>`

`bool getCode(StreamT &in, Node *head, std::string &res)` — Функция принимает на вход поток ввода, указатель на корень бинарного дерева и строковой массив. Функция считывает очередной код из входного потока и записывает соответствующий ему символ в строку `res`.

`template<typename StreamT>`

`void readCountTable(StreamT &in, std::vector<Symbol> &symbols)` - Функция принимает на вход поток ввода, и массив структур типа `Symbol`. Функция

считывает из входного потока символы и количество их повторений в массив symbols.

`void cutZeroes(std::vector<Symbol> &symbols, Symbols &res)` — Функция принимает на вход массив символов и формирует из него новый, состоящий только из символов с не нулевым количеством повторений.

`void splitAtoms(Symbols &atoms, Symbols &res1, Symbols &res2)` — Функция принимает на вход 3 структуры типа Symbols, с первой она работает, в остальные записывает результат. Функция формирует 2 новые структуры, путем разделения массива символов старой на две части.

`void unitAtoms(Symbols &atoms1, Symbols &atoms2, Symbols &res)` - Функция принимает на вход 3 структуры типа Symbols. Функция объединяет первые две структуры, переданные ей в качестве аргументов в одну.

`void S_F_Code(Node *&head, std::vector<Symbol> &S)` — Функция вычисляет коды символов методом Шеннона-Фано.

`Node *Hafman_Code(Symbols &atoms, std::vector<Symbol> &S)`— Функция вычисляет коды символов методом Хаффмана.

`std::string codeMessage(std::string &message, std::vector<Symbol> &S)` - Функция кодирует сообщение и возвращает результат кодирования.

`template<typename StreamT>`

`void deCode(StreamT &in, Node *head, std::string &res)` — Функция принимает на вход поток ввода, указатель на корень бинарного дерева с кодами символов, и результирующую строку. Функция декодирует сообщение и заносит результат в результирующую строку.

template<typename StreamT>

void askAction(StreamT &in) — Функция запрашивает у пользователя действие и производит его.

void askFormat() -Функция запрашивает у пользователя формат ввода.

4. ТЕСТИРОВАНИЕ

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Скриншоты выполнения программы в приложении Б.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Somebody once told me the world is gonna roll me I ain't the sharpest tool in the shed	<p> <Символ - Код - 00 <Символ - e Код - 010 <Символ - o Код - 0110 <Символ - t Код - 0111 <Символ - h Код - 1000 <Символ - l Код - 1001 <Символ - n Код - 1010 <Символ - s Код - 10110 <Символ - d Код - 10111 <Символ - a Код - 1100 <Символ - r Код - 11010 <Символ - i Код - 11011 <Символ - m Код - 11100 <Символ - y Код - 111010 <Символ - w Код - 1110110 <Символ - p Код - 1110111 <Символ - g Код - 111100 <Символ - c Код - 1111010 <Символ - b Код - 1111011 <Символ - S Код - 111110 <Символ - I Код - 1111110 <Символ - ' Код - 1111111 1111100110111000101110110 11010111111010000110101011 11010010000111011010011011 10011100010000111100001000 11101100110110101001101110 01101110110001111000110101 01010110000110100110100110 01001110001000111111000110 01101110101111111011100011 11000010001011010001100110 10111011101010110011100011 10110011010010011011101000 01111000010001011010000101 0111 </p> <p>Кол-во символов в исходном</p>	Кодирование методом Фано_Шеннона.

		сообщении - 86 Размер Кода - 342 Размер при равномерном кодировании - 430 Время работы алгоритма - 0.001097	
2.	12345 1234 123 12 1 78 9	<Символ - 1 Код - 00 <Символ - 7 Код - 0100 <Символ - 5 Код - 0101 <Символ - 3 Код - 011 <Символ - Код - 10 <Символ - 2 Код - 110 <Символ - 4 Код - 1110 <Символ - 9 Код - 11110 <Символ - 8 Код - 11111 00110011111001011000110011 11101000110011100011010001 0010011111011110 Кол-во символов в исходном сообщении - 24 Размер Кода - 69 Размер при равномерном кодировании - 96 Время работы алгоритма - 0.000973	Кодирование методом Хаффмана.
3.	aaaaaaaaaaaaabbbbbbbccc cfffddddd	<Символ - a Код - 0 <Символ - b Код - 100 <Символ - d Код - 101 <Символ - c Код - 110 <Символ - f Код - 111 00000000000001001001001001 00100100110110110110111111 111101101101101101101 Кол-во символов в исходном сообщении - 33 Размер Кода - 73 Размер при равномерном кодировании - 99 Время работы алгоритма - 0.000243	Кодирование методом Фано - Шеннона.
4.	aaaaaaaaaaaaabbbbbbbccc cfffddddd	<Символ - d Код - 00 <Символ - b Код - 01 <Символ - f Код - 100	Кодирование методом Хаффмана

		<Символ - с Код - 101 <Символ - а Код - 11 11111111111111111111111111111111 01010101010101101101101101101 100100100000000000000000 Кол-во символов в исходном сообщении - 33 Размер Кода - 73 Размер при равномерном кодировании - 99 Время работы алгоритма - 0.000789	
5.	s 10 d 11 k 12 001100011100110	Исходное сообщение - kkskkksdksk Время работы алгоритма - 0.000682	Декодирование методом Фано_Шеннона.
6.	s 10 d 11 k 12 001100011100110	Исходное сообщение - kkdskksdskd Время работы алгоритма - 0.000131	Декодирование методом Хаффмана.
7.	1 5 2 4 3 3 4 2 5 1 7 1 8 1 9 1 1000111010110110001110 1010001111000101100011 10110	<Символ - 2 Код - 00 <Символ - 4 Код - 010 <Символ - 9 Код - 0110 <Символ - 8 Код - 0111 <Символ - 1 Код - 10 <Символ - 7 Код - 1100 <Символ - 5 Код - 1101 <Символ - 3 Код - 111 Исходное сообщение - 123451234123121789 Время работы алгоритма - 0.000282	Декодирование методом Хаффмана.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы были изучены алгоритмы кодирования и декодирования Хаффмана и Шеннона-Фано. Написана программа для кодирования и декодирования сообщений с использованием данных алгоритмов.

В результате исследования было выявлено, что кодирование методом Фано-Шеннона иногда выдает неоптимальные коды для символа, и длина сообщения получается большей, чем при кодировании методом Хаффмана, что сходится с теорией.

Также было отмечено, что длина закодированного сообщения при кодировании одним из этих методов, будет меньше длины сообщения закодированного равномерным кодом, а в худшем случае равна ей, что сходится с теорией.

Была исследована зависимость времени работы алгоритмов от количества уникальных символов в кодируемом и закодированном сообщении при фиксированном размере самих сообщений равному 500 символов. Ожидалось, что время работы алгоритма Хаффмана при кодировании и декодировании будет больше времени работы алгоритма Фано-Шеннона, так как в алгоритме Фано-Шеннона коды символов вычисляются при построении бинарного дерева, а в алгоритме Хаффмана сначала строится бинарное дерево, и только потом, с помощью обхода дерева в глубину свычисляются коды символов. Более того, само построение дерева в алгоритме Фано-Шеннона проще.

Результаты полученные при тестировании программы совпадают с теоретическими ожиданиями. Результаты тестирования представлены на следующих графиках.

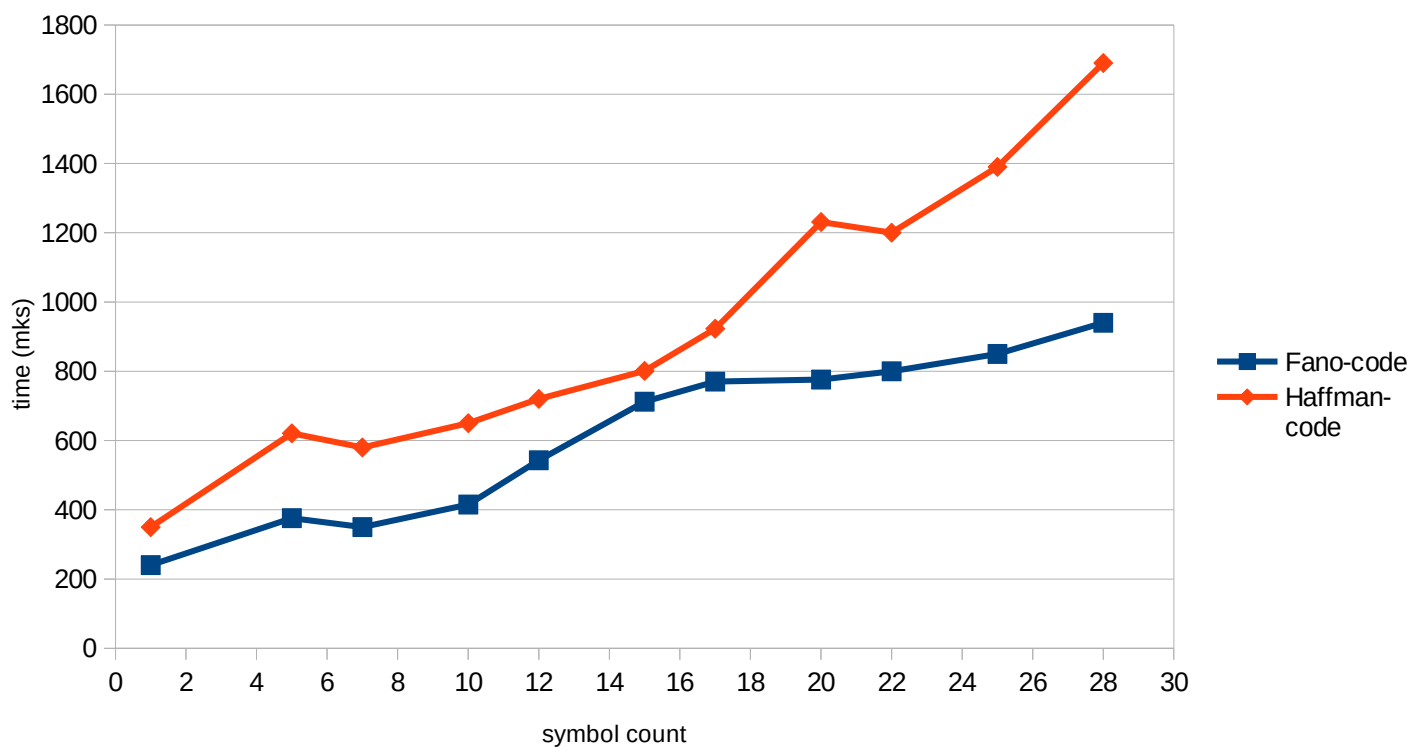


График зависимости времени работы алгоритмов кодирования от количества уникальных символов в исходном сообщении.

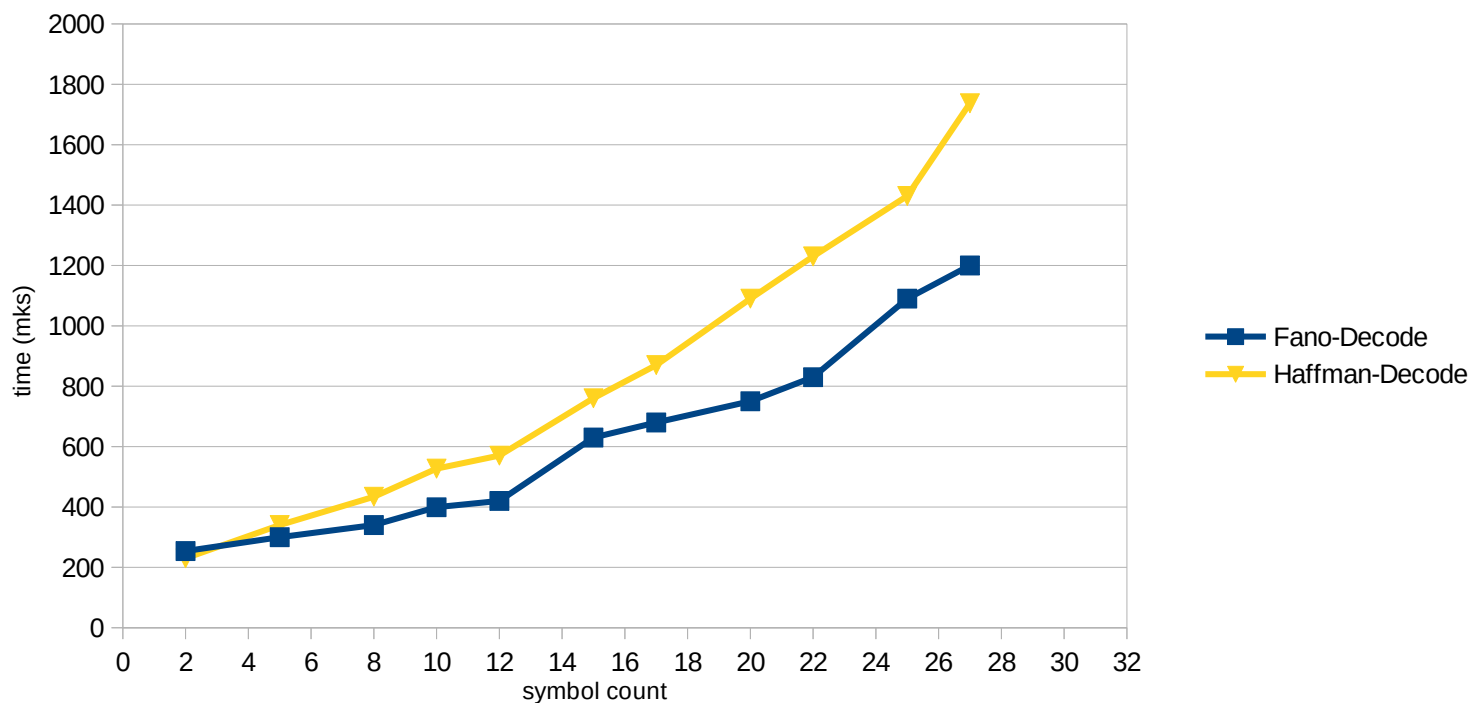


График зависимости времени работы алгоритмов декодирования от количества уникальных символов в закодированном сообщении.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. <https://habr.com/ru/post/137766/>
2. http://compression.ru/download/articles/huff/simakov_2002_huffcode.html
3. https://scask.ru/a_book_sel.php?id=8
4. http://algotlist.ru/compress/standard/shannon_fano.php
5. <https://studassistent.ru/charp/metod-shennona-fano-perevesti-s-c-na-c-c>

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <vector>
#include <sstream>
#include <string>
#include <memory>
#include <fstream>
#include <math.h>
#include <algorithm>
#include <list>
#include <time.h>
#include <chrono>

int depth = 0;

std::string outFileName;

struct Symbol {
    char value_; //СИМВОЛ
    int weight_; // КОЛИЧЕСТВО ПОВТОРЕНИЙ
    std::string code_;

    Symbol() {
        value_ = '\0';
        weight_ = 0;
        code_ = "";
    }

    Symbol(char value, int weight, std::string code) {
        value_ = value;
        weight_ = weight;
        code_ = code;
    }
};

struct Symbols {
    std::vector<Symbol> values_;
    int weight_;

    Symbols() {
        weight_ = 0;
    }
};

struct Node {
    Symbols symbols_;
    std::string code_;
    Node *left_;
    Node *right_;

    explicit Node() {
        code_ = "";
        left_ = nullptr;
        right_ = nullptr;
    }

    Node(const Symbols &symbols, const std::string &code) {
        symbols_ = symbols;
    }
};
```

```

        code_ = code;
        left_ = nullptr;
        right_ = nullptr;
    }

    void createLeft(const Symbols &symbols, const std::string &code) {
        if (left_ != nullptr)
            throw std::runtime_error("Левое поддереву не пусто");
        left_ = new Node(symbols, code);
    }

    void createRight(const Symbols &symbols, const std::string &code) {
        if (right_ != nullptr)
            throw std::runtime_error("Левое поддереву не пусто");
        right_ = new Node(symbols, code);
    }

    void addChild(const Symbols &symbols, const std::string &code) {
        if (left_ == nullptr)
            createLeft(symbols, code);
        else if (right_ == nullptr)
            createRight(symbols, code);
        else
            throw std::runtime_error("Некуда добавить потомка");
    }
};

// Компараторы
bool comp(const Symbol &i, const Symbol &j) {
    return i.weight_ > j.weight_;
}

struct comp1 {
    bool operator()(Node *i, Node *j) const {
        return i->symbols_.weight_ > j->symbols_.weight_;
    }
};

int getLog(int k) {
    if (k == 1)
        return 1;
    int t = k - 1;
    int res = 0;
    while (t != 0) {
        t /= 2;
        res++;
    }
    return res;
}

//СЧИТЫВАНИЕ
template<typename StreamT>
std::string readSymbol(StreamT &in, std::vector<Symbol> &atoms) {
    std::string result;
    char c;
    in >> std::noskipws;
    while (in >> c && c != '\n') {
        int k = (int) c;
        atoms.at(k).value_ = c;
        atoms.at(k).weight_ += 1;
    }
}

```

```

        result.push_back(c);
    }
    return result;
}

template<typename StreamT>
bool getCode(StreamT &in, Node *head, std::string &res) {
    char c;
    if (head->symbols_.values_.size() == 1) {
        res.push_back(head->symbols_.values_.at(0).value_);
        return true;
    }
    in >> std::noskipws;
    in >> c;

    if (c == '0')
        return getCode(in, head->left_, res);
    else if (c == '1')
        return getCode(in, head->right_, res);
    else if (c == '\n')
        return false;
    else
        throw std::runtime_error("Неверный формат закодированного сообщения" + c + ' --');
}

template<typename StreamT>
void deCode(StreamT &in, Node *head, std::string &res) {
    while (getCode(in, head, res));
}

template<typename StreamT>
void readCountTable(StreamT &in, std::vector<Symbol> &symbols) {
    std::string line;
    char c;
    int weight;
    in >> std::noskipws;
    while (std::getline(in, line) && !line.empty()) {
        std::istringstream str(line);
        str >> c;
        str >> weight;
        symbols.at(c).value_ = c;
        symbols.at(c).weight_ = weight;
    }
}

void printAtoms(Symbols &atoms) {
    for (auto &i:atoms.values_)
        std::cout << "Символ - " << i.value_ << " Кол-во вхождений - " << i.weight_ <<
std::endl;
    std::cout << "Общий вес строки = " << atoms.weight_ << std::endl;
}

//работа с символами

void cutZeroes(std::vector<Symbol> &symbols, Symbols &res) {
    for (auto &i:symbols)
        if (i.value_ != '\0' && i.weight_ != 0) {
            Symbol a(i.value_, i.weight_, i.code_);
            res.values_.push_back(a);
        }
}

```

```

        res.weight_ += a.weight_;
    }
}

void splitAtoms(Symbols &atoms, Symbols &res1, Symbols &res2) {
    bool k = true;
    for (auto &i:atoms.values_) {
        if (abs(res1.weight_ + i.weight_ - atoms.weight_ / 2) > abs(res1.weight_ - atoms.weight_ /
2))
            k = false;
        if (k) {
            res1.values_.push_back(i);
            res1.weight_ += i.weight_;
        } else {
            res2.values_.push_back(i);
            res2.weight_ += i.weight_;
        }
    }
}

void unitAtoms(Symbols &atoms1, Symbols &atoms2, Symbols &res) {
    for (auto &i:atoms1.values_) {
        res.values_.push_back(i);
        res.weight_ += i.weight_;
    }
    for (auto &i:atoms2.values_) {
        res.values_.push_back(i);
        res.weight_ += i.weight_;
    }
}

```

//Получение кодов символов

// Построение бинарного дерева Фано-Шенона

```

void S_F_Code(Node *head, std::vector<Symbol> &S) {
    if (head->symbols_.values_.size() == 1) {
        S.at(head->symbols_.values_.at(0).value_).code_ = head->code_;
        return;
    }
    Symbols arr1;
    Symbols arr2;
    splitAtoms(head->symbols_, arr1, arr2);
    head->addChild(arr1, head->code_ + '0');
    head->addChild(arr2, head->code_ + '1');
    S_F_Code(head->left_, S);
    S_F_Code(head->right_, S);
}

void printTree(Node *head) {
    depth++;
    if (head == nullptr)
        return;
    std::cout << "<Depth " << depth << " > " << std::endl;
    std::string s(depth, ' ');

    printAtoms(head->symbols_);
}

```

```

    printTree(head->left_);
    depth--;
    printTree(head->right_);
    depth--;
}

void printTreeCode(Node *head) {
    if (head == nullptr)
        return;
    if (head->left_ == nullptr && head->right_ == nullptr)
        std::cout << "<Символ - " << head->symbols_.values_.at(0).value_ << " Код - " <<
head->code_ << std::endl;

    printTreeCode(head->left_);
    printTreeCode(head->right_);
}

void initHahmanCodes(Node *head, std::string code, std::vector<Symbol> &S) {
    if (head == nullptr)
        return;

    head->code_ = code;
    if (head->symbols_.values_.size() == 1) {
        S.at(head->symbols_.values_.at(0).value_).code_ = head->code_;
    }
    initHahmanCodes(head->left_, head->code_ + "0", S);
    initHahmanCodes(head->right_, head->code_ + "1", S);
}

// Построение бинарного дерева Хаффмана
Node *Hafman_Code(Symbols &atoms, std::vector<Symbol> &S) {
    std::list<Node*> strings;
    for (auto &i:atoms.values_) {
        Node *n = new Node();
        n->symbols_.values_.push_back(i);
        n->symbols_.weight_ = i.weight_;
        strings.push_back(n);
    }
    while (strings.size() != 1) {
        strings.sort(comp1());
        Symbols s;

        Node *left = strings.back();
        strings.pop_back();

        Node *right = strings.back();
        strings.pop_back();

        Node *parent = new Node();
        parent->left_ = left;
        parent->right_ = right;

        unitAtoms(left->symbols_, right->symbols_, s);
        parent->symbols_ = s;
        strings.push_front(parent);
    }
    initHahmanCodes(strings.front(), "", S);
    return strings.front();
}

std::string codeMessage(std::string &message, std::vector<Symbol> &S) {

```

```

    std::string result;
    for (auto &i:message) {
        result += S.at(i).code_;
    }
    return result;
}

void printMenu() {
    std::cout << "1 - кодирование методом Фано-Шенона\n"
                "2 - кодирование методом Хаффмана\n"
                "3 - декодирование методом Фано-Шенона\n"
                "4 - декодирование методом Хаффмана\n" << std::endl;
}

template<typename StreamT>
void askAction(StreamT &in) {

    std::string line;
    std::getline(std::cin, line);
    std::istringstream str(line);
    int k;
    str >> k;

    switch (k) {
        case 1: {
            clock_t time;
            time = clock();
            std::vector<Symbol> symbols(256); //для кодирования
            Symbols atoms;
            std::string message = readSymbol(in, symbols);

            cutZeroes(symbols, atoms);

            std::sort(atoms.values_.begin(), atoms.values_.end(), comp);

            Node *head = new Node(atoms, "");
            S_F_Code(head, symbols);

            printTreeCode(head);

            std::string res = codeMessage(message, symbols);
            std::cout << res << std::endl;

            time = clock() - time;

            std::cout<<"Исходное сообщение - "<< message << "\nКол-во символов в
исходном сообщении - " << atoms.weight_
                << "\nРазмер Кода - " << res.size() <<
                "\nРазмер кода при равномерном кодировании - " <<
getLog(atoms.values_.size()) * atoms.weight_
                << "\nВремя работы алгоритма - " << (double) time / CLOCKS_PER_SEC
                << std::endl;

            outFileName = "Shannon_Fano_Code_" + outFileName;
            std::ofstream fout;
            fout.open(outFileName);
            fout << res<<std::endl;
            fout<<"Исходное сообщение - "<< message << "\nКол-во символов в исходном
сообщении - " << atoms.weight_

```



```

        << "\nРазмер Кода - " << res.size() <<
        "\nРазмер при равномерном кодировании - " << getLog(atoms.values_.size()) *
atoms.weight_
        << "\nВремя работы алгоритма - " << (double) time / CLOCKS_PER_SEC
        << std::endl;
        fout.close();

        break;
    }
    case 2: {
        clock_t time;
        time = clock();

        std::vector<Symbol> symbols(256); //для кодирования
        Symbols atoms;
        std::string message = readSymbol(in, symbols);

        cutZeroes(symbols, atoms);

        std::sort(atoms.values_.begin(), atoms.values_.end(), comp);

        Node *head = Hafman_Code(atoms, symbols);
        printTreeCode(head);
        std::string res = codeMessage(message, symbols);
        std::cout << res << std::endl;

        time = clock() - time;

        std::cout<<"Исходное сообщение - "<< message << "\nКол-во символов в
исходном сообщении - " << atoms.weight_
        << "\nРазмер Кода - " << res.size() <<
        "\nРазмер при равномерном кодировании - " << getLog(atoms.values_.size()) *
atoms.weight_
        << "\nВремя работы алгоритма - " << (double) time / CLOCKS_PER_SEC
        << std::endl;

        outFile_name = "Haffman_Code_" + outFile_name;
        std::ofstream fout;
        fout.open(outFile_name);

        fout << res<<std::endl;
        fout<<"Исходное сообщение - "<< message << "\nКол-во символов в исходном
сообщении - " << atoms.weight_
        << "\nРазмер Кода - " << res.size() <<
        "\nРазмер при равномерном кодировании - " << getLog(atoms.values_.size()) *
atoms.weight_
        << "\nВремя работы алгоритма - " << (double) time / CLOCKS_PER_SEC
        << std::endl;
        fout.close();
        break;
    }
    case 3: {
        clock_t time;

        time = clock();
        std::vector<Symbol> symbols(256); //для кодирования
        Symbols atoms;

        readCountTable(in, symbols);
        cutZeroes(symbols, atoms);

```

```

if(atoms.values_.empty())
{
    throw std::runtime_error("Пустые входные данные");
}

std::sort(atoms.values_.begin(), atoms.values_.end(), comp);

Node *head = new Node(atoms, "");
S_F_Code(head, symbols);

printTreeCode(head);
std::string result;

deCode(in, head, result);

time = clock() - time;

std::cout << "Исходное сообщение -" << result
            << "\nВремя работы алгоритма - " << (double) time / CLOCKS_PER_SEC
            << std::endl;

outFileName = "Shannon_Fano_Decode_" + outFileName;
std::ofstream fout;
fout.open(outFileName);

fout << "Исходное сообщение -" << result
        << "\nВремя работы алгоритма - " << (double) time / CLOCKS_PER_SEC
        << std::endl;
fout.close();
break;
}
case 4: {
    clock_t time;
    time = clock();
    std::vector<Symbol> symbols(256); //для кодирования
    Symbols atoms;

    readCountTable(in, symbols);
    cutZeroes(symbols, atoms);
    if(atoms.values_.empty())
    {
        throw std::runtime_error("Пустые входные данные");
    }

    std::sort(atoms.values_.begin(), atoms.values_.end(), comp);

    Node *head = Hafman_Code(atoms, symbols);

    std::string result;

    printTreeCode(head);

    deCode(in, head, result);

    time = clock() - time;

    std::cout << "Исходное сообщение -" << result

```

```

        << "\nВремя работы алгоритма - " << (double) time / CLOCKS_PER_SEC
        << std::endl;

    outFile = "Haffman_Decode_" + outFile;
    std::ofstream fout;
    fout.open(outFile);

    fout << "Исходное сообщение -" << result
        << "\nВремя работы алгоритма - " << (double) time / CLOCKS_PER_SEC
        << std::endl;
    fout.close();

    break;
}
default:
    break;
}
}

void askFormat() {
    int readFormat;
    std::cout << "0 - считать из файла, 1 - считать с консоли" << std::endl;

    std::string line;
    std::getline(std::cin, line);
    std::istringstream str(line);

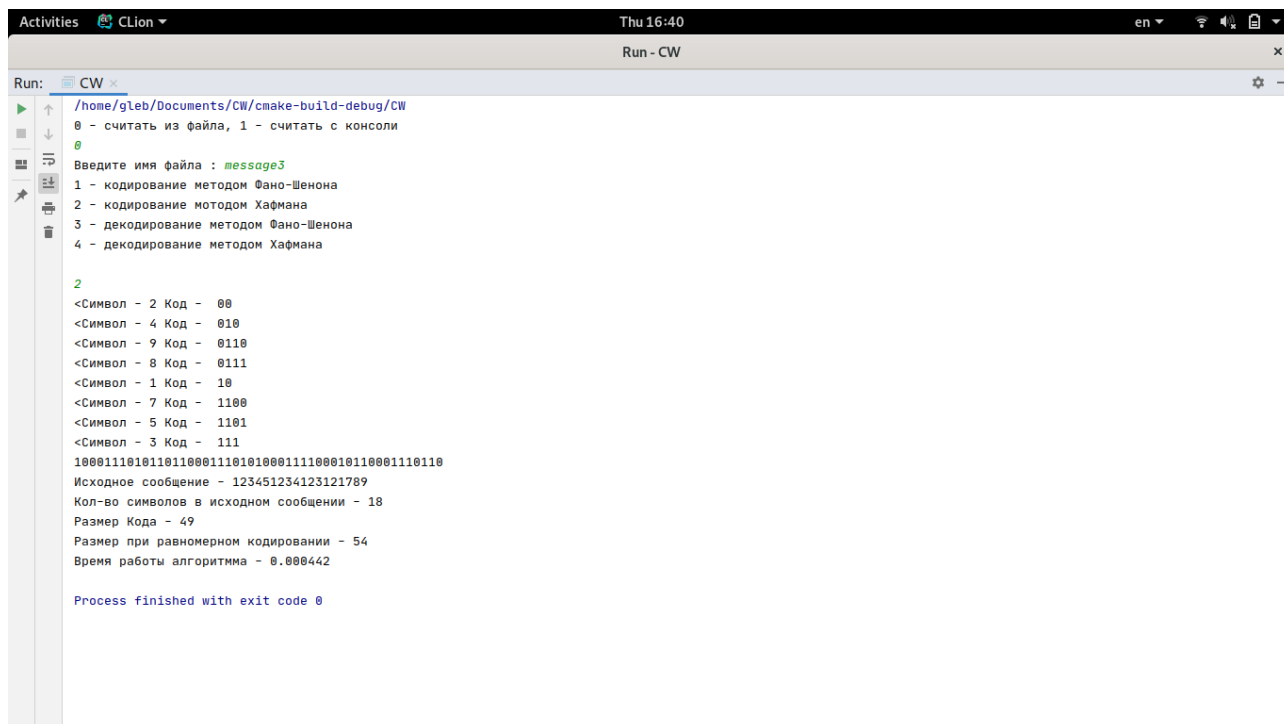
    str >> readFormat;
    switch (readFormat) {
        case 0: {
            std::cout << "Введите имя файла : ";
            std::ifstream in;
            std::string fileName;
            std::getline(std::cin, fileName);
            in.open(fileName);
            if (in) {
                outFile = fileName;
                printMenu();
                askAction(in);
            } else {
                throw std::runtime_error("Файл не найден!");
            }
            in.close();
            break;
        }
        case 1: {
            printMenu();
            std::cout
                << "Введите действие, затем кодируемое сообщение"
                << "или декодируемое сообщение(Должно содержать символы и кол-во их"
                << "повторений и закодированную последовательность)"
                << "\nПример кодируемой последовательности : abbssa"
                << "\nПример декодируемой последовательности : "
                << "\ns 5"
                << "\nk 10"
                << "\nl 7"
                << "\n"
                << "\n1111111111000000000010101010101010"
                << std::endl;
            askAction(std::cin);
            break;
        }
    }
}

```

```
    }  
    default: {  
        throw std::runtime_error("Неверное действие");  
    }  
}  
  
}  
  
int main() {  
    askFormat();  
    return 0;  
}
```

ПРИЛОЖЕНИЕ Б

СКРИНШОТЫ ВЫПОЛНЕНИЯ ПРОГРАММЫ

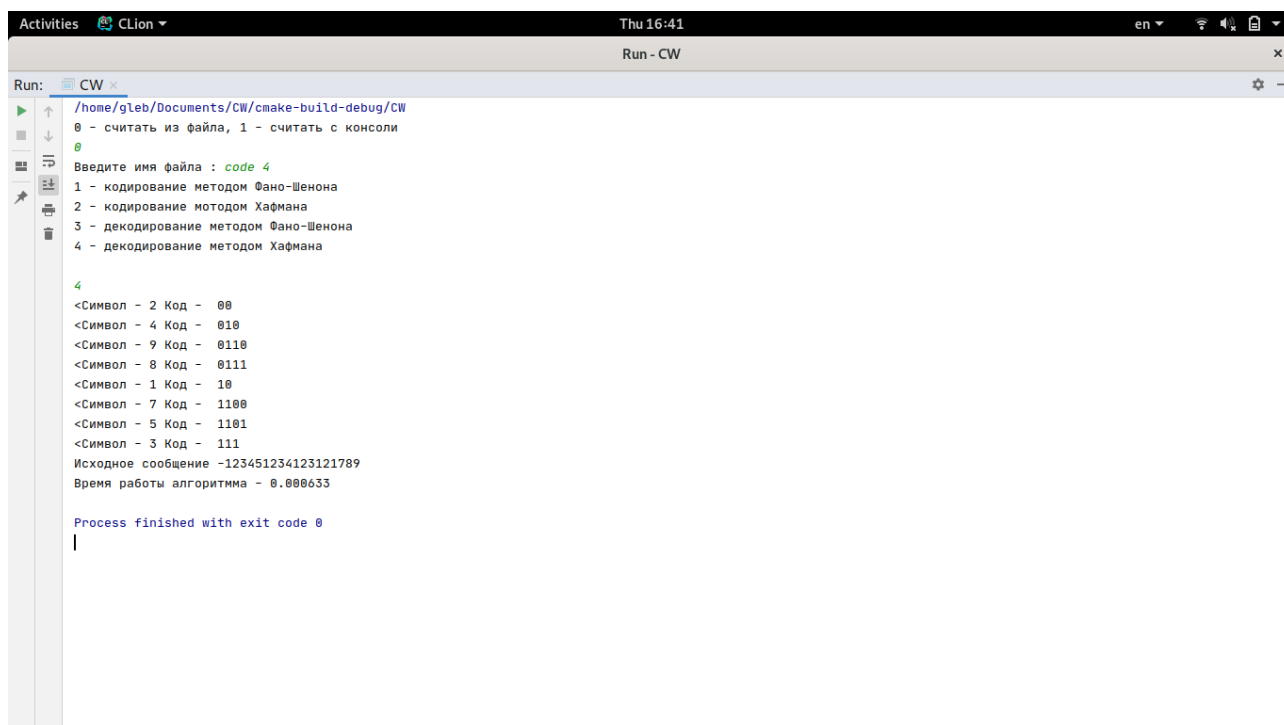


```
Run: CW x
/home/gLeb/Documents/CW/cmake-build-debug/CW
0 - считать из файла, 1 - считать с консоли
0
Введите имя файла : message3
1 - кодирование методом Фано-Шенона
2 - кодирование методом Хаффмана
3 - декодирование методом Фано-Шенона
4 - декодирование методом Хаффмана

2
<Символ - 2 Код - 00
<Символ - 4 Код - 010
<Символ - 9 Код - 0110
<Символ - 8 Код - 0111
<Символ - 1 Код - 10
<Символ - 7 Код - 1100
<Символ - 5 Код - 1101
<Символ - 3 Код - 111
10001110101101100011101000111100010110001110110
Исходное сообщение - 123451234123121789
Кол-во символов в исходном сообщении - 18
Размер Кода - 49
Размер при равномерном кодировании - 54
Время работы алгоритма - 0.000442

Process finished with exit code 0
```

Рисунок 1. Кодирование методом Хаффмана



```
Run: CW x
/home/gLeb/Documents/CW/cmake-build-debug/CW
0 - считать из файла, 1 - считать с консоли
0
Введите имя файла : code 4
1 - кодирование методом Фано-Шенона
2 - кодирование методом Хаффмана
3 - декодирование методом Фано-Шенона
4 - декодирование методом Хаффмана

4
<Символ - 2 Код - 00
<Символ - 4 Код - 010
<Символ - 9 Код - 0110
<Символ - 8 Код - 0111
<Символ - 1 Код - 10
<Символ - 7 Код - 1100
<Символ - 5 Код - 1101
<Символ - 3 Код - 111
Исходное сообщение -123451234123121789
Время работы алгоритма - 0.000633

Process finished with exit code 0
|
```

Рисунок 2. Декодирование методом Хаффмана

```
Activities CLion Thu 16:42 en Run - CW
Run: CW
/home/gleb/Documents/CW/cmake-build-debug/CW
0 - считать из файла, 1 - считать с консоли
0
Введите имя файла : message3
1 - кодирование методом Фано-Шенона
2 - кодирование методом Хаффмана
3 - декодирование методом Фано-Шенона
4 - декодирование методом Хаффмана

1
<Символ - 1 Код - 00
<Символ - 2 Код - 01
<Символ - 3 Код - 100
<Символ - 4 Код - 101
<Символ - 5 Код - 1100
<Символ - 7 Код - 1101
<Символ - 8 Код - 1110
<Символ - 9 Код - 1111
00011001011000001100101000110000010011011101111
Исходное сообщение - 123451234123121789
Кол-во символов в исходном сообщении - 18
Размер Кода - 49
Размер кода при равномерном кодировании - 54
Время работы алгоритма - 0.000455

Process finished with exit code 0
```

Рисунок 3. Кодирование методом Фано-Шеннона.

```
Activities CLion Thu 16:56 en Run - CW
Run: CW
/home/gleb/Documents/CW/cmake-build-debug/CW
0 - считать из файла, 1 - считать с консоли
0
Введите имя файла : code 4
1 - кодирование методом Фано-Шенона
2 - кодирование методом Хаффмана
3 - декодирование методом Фано-Шенона
4 - декодирование методом Хаффмана

3
<Символ - 1 Код - 00
<Символ - 2 Код - 01
<Символ - 3 Код - 100
<Символ - 4 Код - 101
<Символ - 5 Код - 1100
<Символ - 7 Код - 1101
<Символ - 8 Код - 1110
<Символ - 9 Код - 1111
Исходное сообщение -32724327212814327
Время работы алгоритма - 0.000305

Process finished with exit code 0
|
```

Рисунок 4. Декодирование методом Фано-Шеннона.