

**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: Кодирование и декодирование методами Хаффмана и**  
**Фано-Шеннона – исследование**

Студент гр. 9304

\_\_\_\_\_

Боблаков Д.С.

Преподаватель

\_\_\_\_\_

Филатов А.Ю.

Санкт-Петербург

2020

## ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Боблаков Д.С.

Группа 9304

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

Исходные данные: Программе на вход подается строка

Содержание пояснительной записки:

- Содержание
- Введение
- Описание алгоритмов
- Описание работы программы
- Тестирование
- Исследование
- Заключение
- Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 25 страниц.

Дата выдачи задания: 23.11.2020

Дата сдачи реферата: 28.12.2020

Дата защиты реферата: 28.12.2020

Студент

\_\_\_\_\_

Боблаков Д.С.

Преподаватель

\_\_\_\_\_

Филатов А.Ю.

## **АННОТАЦИЯ**

В данной курсовой работе была разработана программа на языке программирования C++, которая производит сравнение скорости и эффективности кодирования и декодирования алгоритмов Хаффмана и Шеннона — Фано. Также была реализована программа генерации входных данных, которая позволила собрать исчерпывающую статистику работы алгоритмов.

## **SUMMARY**

In this course work, a program was developed in the C++ programming language, which compares the speed and efficiency of encoding and decoding the Huffman and Shannon — Fano algorithms. Also, a program for generating input data was implemented, which allowed us to collect comprehensive statistics on the operation of algorithms.

## СОДЕРЖАНИЕ

	Введение	5
1	Формальная постановка задачи	5
2	Описание алгоритма	7
2.1	Статический алгоритм Хаффмана	7
2.2	Статический алгоритм Шеннона – Фано	8
3	Описание работы программы	10
4	Тестирование	12
5	Исследование	14
	Заключение	16
	Список использованных источников	17
	Приложение А. Исходный код	18

## **ВВЕДЕНИЕ**

Цель работы: Сравнить кодирование и декодирование методами Хаффмана и Фано-Шеннона. Фиксация и накопление статистики испытаний алгоритмов.

Задача: Реализовать программу, которая выполняет кодирование и декодирование входного текста, фиксируя основные параметры выполнения алгоритмов. Собрать статистику и сделать выводы по полученным результатам.

На практике алгоритмы Хаффмана и Фано-Шэннона используются для сжатия данных в различных архиваторах, что помогает при их хранении и передаче. Основной принцип обоих алгоритмов состоит в том, что некоторые символы повторяются в тексте чаще других, и для них создается более короткий код, а для символов, которые повторяются чаще – более длинный. Это в результате даёт текст меньшей длины.

## 1. ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

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

Исследование должно содержать:

1. Анализ задачи, цели, технологию проведения и план экспериментального исследования.
2. Генерацию представительного множества реализаций входных данных (с заданными особенностями распределения (для среднего и для худшего случаев)).
3. Выполнение исследуемых алгоритмов на сгенерированных наборах данных. При этом в ходе вычислительного процесса фиксируется как характеристики (например, время) работы программы, так и количество произведенных базовых операций алгоритма.
4. Фиксацию результатов испытаний алгоритма, накопление статистики.
5. Представление результатов испытаний, их интерпретацию и сопоставление с теоретическими оценками.

## 2. ОПИСАНИЕ АЛГОРИТМОВ

### 2.1. Статический алгоритм Хаффмана

Алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана .

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.
2. Выбираются два свободных узла дерева с наименьшими весами.
3. Создается их родитель с весом, равным их суммарному весу.
4. Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.
5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.
6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

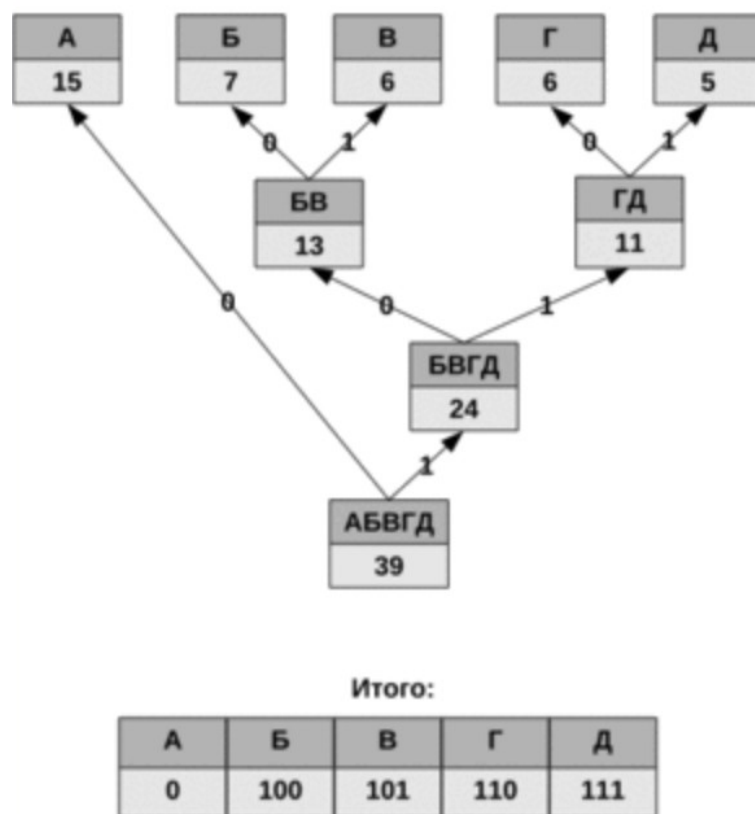


Рисунок 1 – пример работы алгоритма Хаффмана

## 2.2. Статический алгоритм Шеннона – Фано

1. Символы первичного алфавита  $m_1$  выписывают по убыванию вероятностей.
2. Символы полученного алфавита делят на две части, суммарные вероятности символов которых максимально близки друг другу.
3. В префиксном коде для первой части алфавита присваивается двоичная цифра «0», второй части — «1».
4. Полученные части рекурсивно делятся, и их частям назначаются соответствующие двоичные цифры в префиксном коде.

Когда размер подалфавита становится равен нулю или единице, то дальнейшего удлинения префиксного кода для соответствующих ему символов



первичного алфавита не происходит, таким образом, алгоритм присваивает различным символам префиксные коды разной длины.

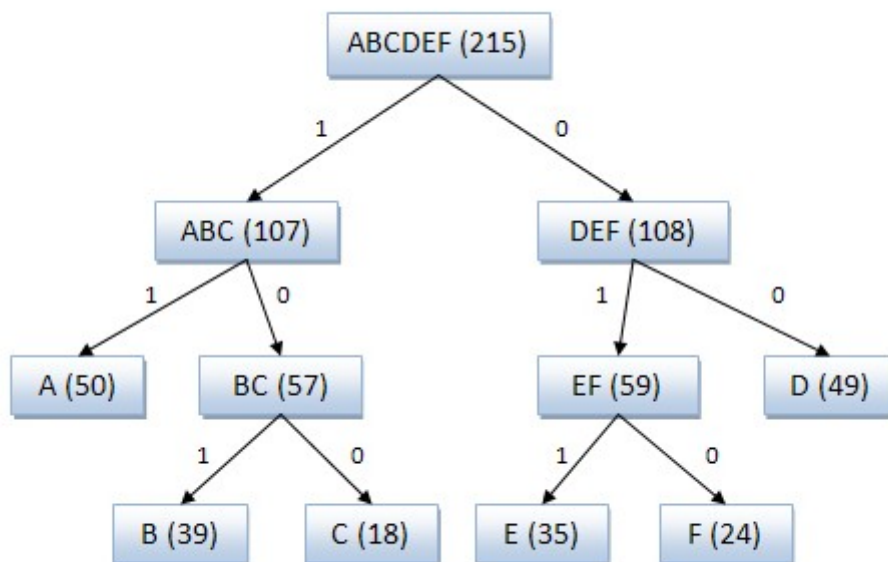


Рисунок 2 – пример работы алгоритма Шеннона – Фано

### 3. ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

#### Описание структур данных и функций

1. *Class Node* – объекты данного класса являются узлами деревьев.
2. *Struct Pair* – экземпляры данной структуры являются элементами словаря, где символ это ключ, а код – значение.
3. Функция *std::string load\_from\_file(const std::string& filename)* – данная функция считывает из файла данные и возвращает строку с этими данными.
4. Функция *void printTree(std::shared\_ptr<Node> node, int n = 0)* – данная функция печатает дерево. Эта функция носит отладочный характер.
5. Функция *bool comp(std::shared\_ptr<Node> &left, std::shared\_ptr<Node> &right)* – данная функция является компаратором для сортировки.
6. Функция *std::vector<std::shared\_ptr<Node>> count\_symbols(const std::string& string)* – данная функция возвращает вектор с узлами будущего дерева.
7. Функция *std::shared\_ptr<Node> cons\_tree\_huffman(const std::shared\_ptr<Node> &a, const std::shared\_ptr<Node> &b)* – данная функция объединяет узлы дерева Хаффмана.
8. Функция *std::shared\_ptr<Node> make\_tree\_huffman(std::vector<std::shared\_ptr<Node>> &vector)* – данная функция строит дерево Хаффмана.
9. Функция *void make\_pairs(std::shared\_ptr<Node> &head, std::vector<Pair> &pairs, std::string string)* – данная функция проходит по дереву и создает пары (словарь) символ – значение, которые сохраняются в векторе.
10. Функция *void unload\_pairs(const std::vector<Pair> &pairs, const std::string& filename = "pairs.txt")* – данная функция выгружает пары в файл.
11. Функция *void encode(const std::string& string, std::vector<Pair> pairs, const std::string& filename, std::string &res)* – данная функция кодирует по заданным парам исходный текст.
12. Функция *void decode(std::string encodedText, const std::string& decoded, const std::vector<Pair> &codes)* – данная функция декодирует полученную строку по имеющимся парам.

13. Функция *void cons\_fano\_tree(std::shared\_ptr<Node> &elem, std::vector<std::shared\_ptr<Node>> vector)* – данная функция рекурсивно строит дерево Фано.
14. Функция *void make\_fano\_tree(std::string &input\_str, const std::vector<std::shared\_ptr<Node>>& vector, std::shared\_ptr<Node> &head)* – данная функция запускает построение дерева Фано.
15. Функция *void read\_file(std::vector<std::string> &vector, std::ifstream& ifstream)* – данная функция считывает из файла данные в вектор.
16. Функция *int stat\_process(int argc, char\*\* argv)* – данная функция является основной для работы со статистическими данными.
17. Функция *int process()* – данная функция является основной для обычной работы с программой.
18. Функция *int main(int argc, char \*\* argv)* – данная функция с помощью директив условной компиляции запускает необходимый сценарий работы программы.

## 4. ТЕСТИРОВАНИЕ

Для запуска программы в обычном режиме достаточно собрать ее с помощью утилиты `make`, а затем запустить ее `./a.out`. Далее следует ввести текст, который будет кодироваться. Для запуска программы в режиме сбора статистики следует собрать ее с помощью команды `g++ main.cpp -D STAT`. Затем нужно запустить программу, указав имя файла для считывания текста: `./a.out test.txt`. Программа выводит результаты в консоль (только в обычном режиме) и в соответствующие файлы: `pairs.txt`, `pairs2.txt`, `res_dec_h.txt`, `res_dec_f.txt`, `res_enc_h.txt`, `res_enc_f.txt`. Исходный код программы см. в приложении А.

```
dmity@haze:~/cw$ ./a.out
billions and billions and billions
Huffman time encode = 0.000742 seconds
Huffman time decode = 0.000337 seconds
Fano time encode = 0.000522 seconds
Fano time decode = 0.000243 seconds

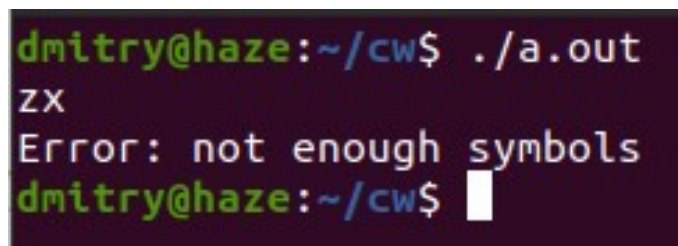
-----
Huffman pairs | i = 00 | b = 010 | d = 0110 | a = 0111 |   = 100 | n
= 101 | s = 1100 | o = 1101 | l = 111 |
Huffman code result = 01000111111001101101110010001111010110100010001
11111001101101110010001111010110100010001111110011011011100

-----
Fano pairs | i = 000 | l = 001 | n = 010 |   = 011 | b = 1000 | o = 1
001 | s = 101 | a = 110 | d = 111 |
Fano code result = 10000000010010001001010101011110010111011100000000
1001000100101010101111001011101110000000010010001001010101

-----
Time difference:
Fano's algorithm is 0.00022 seconds faster when encoding
Fano's algorithm is 9.4e-05 seconds faster when decoding

-----
Compression results:
Huffman's algorithm compressed better by 2 symbols
This algorithm more efficient by 1.88679 percent
dmity@haze:~/cw$
```

Рисунок 3 – пример работы программы

A terminal window with a dark background. The prompt is 'dmitry@haze:~/cw\$'. The user enters './a.out'. The program outputs 'zx'. Then, an error message 'Error: not enough symbols' is displayed. The prompt returns to 'dmitry@haze:~/cw\$' with a white cursor.

```
dmitry@haze:~/cw$ ./a.out
zx
Error: not enough symbols
dmitry@haze:~/cw$
```

Рисунок 4 – пример работы программы при вводе недостаточного количества  
СИМВОЛОВ

## 5. ИССЛЕДОВАНИЕ

Таблица 1 – Статистические данные

Серия тестов	1	2	3	4	5	6	7
Среднее время кодирования по Хаффману	0.086295	0.070627	0.056985	0.037566	0.027028	0.002446	0.001857
Среднее время кодирования по Шеннону – Фано	0.090252	0.068395	0.045548	0.022153	0.022172	0.002918	0.001173
Среднее время декодирования по Хаффману	0.203258	0.151324	0.101555	0.050833	0.031362	0.024814	0.010387
Среднее время декодирования по Шеннону – Фано	0.328531	0.247282	0.166319	0.080505	0.041422	0.013623	0.009258
Длина исходного текста (в символах)	39999	30000	19998	9999	4998	999	498
Длина закодированного сообщения алгоритмом Хаффмана (в бинарном виде)	228700	178502	121328	61130	31051	6342	3155
Длина закодированного сообщения алгоритмом Шеннона – Фано (в бинарном виде)	256921	192683	129015	64138	32101	6427	3169
Разница длин закодированных текстов (в процентах)	12.33974	7.94444	6.33571	4.92066	3.32691	1.13183	0.00423

Каждая серия тестов включает в себя трехкратный запуск программы с исходными данными, сгенерированными программой реализованной в generator.cpp.

### Выводы по таблице:

Скорость работы алгоритмов практически не имеет разницы на любых объемах данных. Алгоритм Хаффмана является усовершенствованной версией алгоритма Фано-Шэннона из-за чего длина закодированного им

сообщения всегда меньше либо равна длине строки после применения алгоритма Фано-Шэннона. Также в ходе исследования было замечено, что чем больше различных символов необходимо закодировать, тем больше становится разница в длине закодированных сообщений. Это происходит из-за не всегда оптимального деления символов при шаге алгоритма Фано-Шэннона.

От указанного недостатка свободна методика Хаффмана. Она гарантирует однозначное построение кода с наименьшим для данного распределения вероятностей средним числом символов на букву.

Метод Хаффмана производит идеальное сжатие (то есть, сжимает данные до их энтропии), если вероятности символов точно равны отрицательным степеням числа 2. Результаты эффективности кодирования по методу Хаффмана всегда лучше результатов кодирования по методу Шеннона-Фано.

## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы была реализована программа на языке программирования C++, которая кодирует/декодирует исходный текст методами Хаффмана и Шеннона – Фано. Была зафиксирована и собрана статистика эффективности работы алгоритмов. Был реализован генератор исходных данных на языке программирования C++. Также были закреплены знания по дисциплине «Алгоритмы и структуры данных», полученные на протяжении семестра.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

### 1. Алгоритм Шеннона –Фано:

[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](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) (дата обращения: 20.12.2020).

### 2. Алгоритм Шеннона –Фано: [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#:~:text=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC20%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0%20%E2%80%94%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%20%D0%BE%D0%BF%D1%82%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%BF%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81%D0%BD%D0%BE%D0%B3%D0%BE,%D0%B2%D0%BE%20%D0%BC%D0%BD%D0%BE%D0%B3%D0%B8%D1%85%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0%D1%85%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](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#:~:text=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC20%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0%20%E2%80%94%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%20%D0%BE%D0%BF%D1%82%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%BF%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81%D0%BD%D0%BE%D0%B3%D0%BE,%D0%B2%D0%BE%20%D0%BC%D0%BD%D0%BE%D0%B3%D0%B8%D1%85%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0%D1%85%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)

(дата обращения: 11.12.2020).

### 3. Генерация исходных данных:

<https://coderoad.ru/1813259/%D0%93%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F-%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D1%85-%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D0%BE%D0%B2-%D0%B8-%D1%86%D0%B5%D0%BB%D1%8B%D1%85-%D1%87%D0%B8%D1%81%D0%B5%D0%BB-%D0%B2-C> (дата обращения: 23.12.2020).

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД

**Имя файла: main.cpp**

```
#include <iostream>
#include <memory>
#include <string>
#include <fstream>
#include <vector>
#include <algorithm>
#include <ctime>
#define STAT

class Node{
public:
    char c{};
    int count{};
    std::string string;
    std::shared_ptr<Node> left {nullptr};
    std::shared_ptr<Node> right {nullptr};
};

struct Pair{
    std::string symbol;
    std::string code;
};

std::string load_from_file(const std::string& filename){
    std::string data;
    std::string string;
    std::ifstream file;
    file.open(filename);
    while(getline(file, string)){
        data+=string;
    }
    file.close();
    return data;
}

void printTree(std::shared_ptr<Node> node, int n =0 ){
    if (node!= nullptr) {
        std::cout << ' ' << node->string<<"="<<node->count;
        if(node->right!= nullptr) {
```

```

        printTree (node->right,n+1);
    }
    else std::cout <<"\n";

    if(node->left!= nullptr) {
        for (int i=1;i<=n;i++){
            std::cout << " ";
        }
        printTree (node->left,n+1);
    }
}
else {
    std::cout<<"Tree is empty!";
}
}

bool comp(std::shared_ptr<Node> &left, std::shared_ptr<Node> &right){
    return left->count > right->count;
}

std::vector<std::shared_ptr<Node>> count_symbols(const std::string& string){
    std::vector<std::shared_ptr<Node>> vector;

    bool exist;

    if (string.size()>0){
        for (char i : string) {
            exist= false;
            for (auto & j : vector) {
                if (i==j->c){
                    exist= true;
                    j->count++;
                }
            }
            if (!exist){
                std::shared_ptr<Node> node;
                node = std::make_shared<Node>();
                node->c=i;
                node->count=1;
                vector.push_back(node);
            }
        }
    }
    for (auto & i : vector) {
        i->string=i->c;
    }
    std::sort(vector.begin(), vector.end(),comp);
    //print_vector(vector);
    return vector;
}

```

```

std::shared_ptr<Node> cons_tree_huffman(const std::shared_ptr<Node>& a, const
std::shared_ptr<Node>& b){
    std::shared_ptr<Node> head = std::make_shared<Node>();

    head->left=a;
    head->right=b;
    head->count=a->count+b->count;
    head->string=a->string+b->string;

    return head;
}

std::shared_ptr<Node> make_tree_huffman(std::vector<std::shared_ptr<Node>>& vector){
    std::shared_ptr<Node> head;

    while (vector.size() >= 2){

        head= cons_tree_huffman(vector[vector.size() - 1], vector[vector.size() - 2]);
        vector.pop_back();
        vector.pop_back();
        vector.push_back(head);
        std::sort(vector.begin(), vector.end(), comp);

    }

    return head;
}

void make_pairs( std::shared_ptr<Node>& head, std::vector<Pair>& pairs, std::string string){

    if(head->left == nullptr && head->right == nullptr){
        pairs.push_back({head->string, string});
    }
    if(head->left != nullptr) {
        make_pairs(head->left, pairs, string + "0");
    }
    if(head->right != nullptr) {
        make_pairs(head->right, pairs, string + "1");
    }
}

void unload_pairs(const std::vector<Pair>& pairs, const std::string& filename= "pairs.txt"){
    std::ofstream file;
    file.open(filename);
    for(auto & pair : pairs){
        file<<pair.symbol<<" - "<< pair.code <<"\n";
    }
    file.close();
}

```

```

void encode(const std::string& string, std::vector<Pair> pairs, const std::string& filename,
std::string & res){
    res="";
    std::ofstream file;
    file.open(filename);

    for(char i : string){
        for(auto & pair : pairs){
            if(pair.symbol[0] == i){
                file << pair.code;
                res += pair.code;
            }
        }
    }

    // file<<"\n";
    file.close();
}

void decode(std::string encodedText, const std::string& decoded, const std::vector<Pair>& codes){
    std::ofstream file;
    file.open(decoded);
    std::string code;
    std::string::iterator iterator = encodedText.begin();
    while(encodedText.end() != iterator){
        code += *iterator;
        for(auto & i : codes){
            if(i.code == code){
                file << i.symbol;
                code.clear();
            }
        }
        iterator++;
    }
    file << "\n";
    file.close();
}

void cons_fano_tree(std::shared_ptr<Node> &elem, std::vector<std::shared_ptr<Node>> vector) {

    //Condition of end of recursion
    if(vector.size() <= 1) {
        return;
    }

    //Create child Nodes and vectors for them
    std::shared_ptr<Node> right_branch(new Node);
    std::shared_ptr<Node> left_branch(new Node);
    std::vector<std::shared_ptr<Node>> right_cnt_vector;
    std::vector<std::shared_ptr<Node>> left_cnt_vector;

    //Set connection

```

```

elem->right = right_branch;
elem->left = left_branch;

//Creating of right branch and vector
for(size_t i = vector.size() - 1; i >= 1; --i) {

    right_branch->count += vector[i]->count;
    right_branch->string.insert(0, vector[i]->string);
    right_cnt_vector.insert(right_cnt_vector.begin(), vector.back());
    vector.pop_back();

    if(right_branch->count + vector[i - 1]->count >= elem->count / 2) {
        break;
    }
}

//Creating of left branch and vector
left_cnt_vector = vector;
left_branch->string = elem->string;
left_branch->string.erase(left_branch->string.end() - right_branch->string.length(), left_branch-
>string.end());
left_branch->count = elem->count - right_branch->count;

//Recursive call
cons_fano_tree(left_branch, left_cnt_vector);
cons_fano_tree(right_branch, right_cnt_vector);

}

void make_fano_tree(std::string &input_str, const std::vector<std::shared_ptr<Node>>& vector,
std::shared_ptr<Node> &head) {

    int sum = 0;
    std::string all_chars;
    for (const auto & i : vector) {
        sum+=i->count;
        all_chars+=i->c;
    }

    head->count = sum;
    head->string = all_chars;

    cons_fano_tree(head, vector);
}

void read_file(std::vector<std::string> &vector, std::ifstream& ifstream) {
    std::string current_file_string;
    while (std::getline(ifstream, current_file_string))
    {
        if (current_file_string.back() == '\r')
            current_file_string.erase(current_file_string.end() - 1);
        vector.push_back(current_file_string);
    }
}

```

```

    }
}

int stat_process(int argc, char** argv){

    if (argc < 2)
    {
        std::cout << "Incorrect input file\n";
        return 1;
    }
    std::ifstream input(argv[1]);
    if (!input.is_open())
    {
        std::cout << "Incorrect input file\n";
        return 1;
    }

    std::vector<std::string> file_data;
    read_file(file_data, input);

    std::string string, string2;

    for (auto & i : file_data){
        string+=i;
    }
    string2=string;

    std::shared_ptr<Node> head;
    std::vector<Pair> pairs;
    std::string res_enc_huf;
    std::vector<std::shared_ptr<Node>> vector;
    double start_enc_huffman, end_dec_huffman, start_dec_huffman, start_enc_fano,
    start_dec_fano, end_enc_fano, end_dec_fano,
    huffman_enc_time,huffman_dec_time,fano_enc_time, fano_dec_time;

    std::transform(string.begin(), string.end(), string.begin(), tolower);
    if (string.length()<=2){
        std::cout<<"Error: not enough symbols";
        return EXIT_FAILURE;
    }

    //encode
    start_enc_huffman = clock();
    vector= count_symbols(string);
    head = make_tree_huffman(vector);
    make_pairs(head, pairs,"");
    encode(string, pairs, "res_enc_h.txt", res_enc_huf);

```

```

end_dec_huffman=clock();
huffman_enc_time= (end_dec_huffman - start_enc_huffman) / CLOCKS_PER_SEC;
std::cout << huffman_enc_time << "\n";

//decode huffman
start_dec_huffman=clock();
decode(res_enc_huf, "res_dec_h.txt", pairs);
end_dec_huffman =clock();
huffman_dec_time= (end_dec_huffman - start_dec_huffman) / CLOCKS_PER_SEC;
std::cout << huffman_dec_time << "\n";

//Fano encode
std::shared_ptr<Node> head_fano= std::make_shared<Node>();
std::string res_enc_fano;
std::vector<Pair> pairs2;
start_enc_fano=clock();
vector= count_symbols(string2);
make_fano_tree(string2, vector, head_fano);
make_pairs(head_fano, pairs2,"");
encode(string2, pairs2, "res_enc_f.txt", res_enc_fano);
end_enc_fano=clock();
fano_enc_time=(end_enc_fano-start_enc_fano)/CLOCKS_PER_SEC;
std::cout<<fano_enc_time<<"\n";

//decode FANO
start_dec_fano=clock();
decode(res_enc_fano, "res_dec_f.txt", pairs2);
end_dec_fano= clock();
fano_dec_time=(end_dec_fano-start_dec_fano)/CLOCKS_PER_SEC;
std::cout<<fano_dec_time<<"\n";

std::cout<<string.size()<<"\n";
std::cout << res_enc_huf.size() << "\n";
std::cout<<res_enc_fano.size()<<"\n";

return 0;
}

int process(){

    std::string string, string2;

    std::shared_ptr<Node> head;
    std::vector<Pair> pairs;
    std::string res_enc_huf;
    std::vector<std::shared_ptr<Node>> vector;
    double start_enc_huffman, end_dec_huffman, start_dec_huffman, start_enc_fano,
    start_dec_fano, end_enc_fano, end_dec_fano,
    huffman_enc_time,huffman_dec_time,fano_enc_time,fano_dec_time;

    std::getline(std::cin, string);

```



```

std::transform(string.begin(), string.end(), string.begin(), tolower);
if (string.length() <= 2) {
    std::cout << "Error: not enough symbols";
    return EXIT_FAILURE;
}
string2 = string;

//encode huffman
start_enc_huffman = clock();
vector = count_symbols(string);
head = make_tree_huffman(vector);
make_pairs(head, pairs, "");
unload_pairs(pairs);
encode(string, pairs, "res_enc_h.txt", res_enc_huf);
end_dec_huffman = clock();
huffman_enc_time = (end_dec_huffman - start_enc_huffman) / CLOCKS_PER_SEC;
std::cout << "Huffman time encode = " << huffman_enc_time << " seconds";

//decode huffman
start_dec_huffman = clock();
string = load_from_file("res_enc_h.txt");
decode(res_enc_huf, "res_dec_h.txt", pairs);
end_dec_huffman = clock();
huffman_dec_time = (end_dec_huffman - start_dec_huffman) / CLOCKS_PER_SEC;
std::cout << "\nHuffman time decode = " << huffman_dec_time << " seconds";

//Fano
std::shared_ptr<Node> head_fano = std::make_shared<Node>();
std::string res_enc_fano;
std::vector<Pair> pairs2;
start_enc_fano = clock();
vector = count_symbols(string2);
make_fano_tree(string2, vector, head_fano);
make_pairs(head_fano, pairs2, "");
unload_pairs(pairs2, "pairs2.txt");
encode(string2, pairs2, "res_enc_f.txt", res_enc_fano);
end_enc_fano = clock();
fano_enc_time = (end_enc_fano - start_enc_fano) / CLOCKS_PER_SEC;
std::cout << "\nFano time encode = " << fano_enc_time << " seconds";

//decode FANO
start_dec_fano = clock();
string2 = load_from_file("res_enc_f.txt");
decode(res_enc_fano, "res_dec_f.txt", pairs2);
end_dec_fano = clock();
fano_dec_time = (end_dec_fano - start_dec_fano) / CLOCKS_PER_SEC;
std::cout << "\nFano time decode = " << fano_dec_time << " seconds";

```

```

std::cout<<"\n_____";
std::cout<<"\nHuffman pairs | ";
for (auto & pair : pairs) {
    std::cout<<pair.symbol<<" = "<<pair.code<<" | ";
}
std::cout<<"\nHuffman code result = ";
std::cout << res_enc_huf;
std::cout<<"\n_____";

std::cout<<"\nFano pairs | ";
for (auto & pair : pairs2) {
    std::cout<<pair.symbol<<" = "<<pair.code<<" | ";
}
std::cout<<"\nFano code result = ";
std::cout<<res_enc_fano;

std::cout<<"\n_____";
std::cout<<"\nTime difference: \n";
if (fano_enc_time - huffman_enc_time > 0){
    std::cout<<"Huffman's algorithm is "<< fano_enc_time-huffman_enc_time<<" seconds faster
when encoding";
} else std::cout<<"Fano's algorithm is "<< huffman_enc_time-fano_enc_time<<" seconds faster
when encoding";
std::cout<<"\n";
if (fano_dec_time - huffman_dec_time > 0){
    std::cout<<"Huffman's algorithm is "<< fano_dec_time-huffman_dec_time<<" seconds faster
when decoding";
} else std::cout<<"Fano's algorithm is "<< huffman_dec_time-fano_dec_time<<" seconds faster
when decoding";
std::cout<<"\n_____";
std::cout<<"\nCompression results: \n";
if (res_enc_fano.size() - res_enc_huf.size() > 0){
    std::cout<<"Huffman's algorithm compressed better by "<< res_enc_fano.size() -
res_enc_huf.size()<<" symbols";
    std::cout<<"\nThis algorithm more efficient by "<<double (double
(res_enc_fano.size())/double(res_enc_huf.size()))*100-100<<" percent";
} else {std::cout<<"Fano's algorithm compressed better by "<< res_enc_huf.size()-
res_enc_fano.size()<<" symbols";
    std::cout<<"\nThis algorithm more efficient by "<<double (double
(res_enc_huf.size())/double(res_enc_fano.size()))*100-100<<" percent";
}
std::cout<<"\n";

return 0;
}

int main(int argc, char ** argv) {
#ifdef STAT
    return process();
#endif
#ifdef STAT

```

```

    return stat_process(argc, argv);
#endif

}

```

### Имя файла: tests\_script.sh

```

#!/bin/bash

printf "\nLaunching the tests\n\n"
for n in {1..1}
do
    printf "Test$n:\n"
    ./a.out < Test/test$n.txt
    printf "\n"
    cat "/Test/test$n.txt" | tr -d '\n'
    if cmp "/res_enc_h.txt" "/Test/true_out$n.txt" > /dev/null; then
        printf "\n*** - Passed\n"
    else
        printf "\n*** - Failed\n"
    fi
    printf "Desired result:\n"
    cat "/Test/true_out$n.txt"
    printf "Actual result:\n"
    cat "/res_enc_h.txt"
    printf "\n=====\n"
done
for n in {2..2}
do
    printf "Test$n:\n"
    ./a.out < Test/test$n.txt
    printf "\n"
    cat "/Test/test$n.txt" | tr -d '\n'
    if cmp "/res_enc_f.txt" "/Test/true_out$n.txt" > /dev/null; then
        printf "\n*** - Passed\n"
    else
        printf "\n*** - Failed\n"
    fi
    printf "Desired result:\n"
    cat "/Test/true_out$n.txt"
    printf "Actual result:\n"
    cat "/res_enc_f.txt"
    printf "\n=====\n"
done
for n in {3..3}
do

```

```

printf "Test$n:\n"
./a.out < Test/test$n.txt
printf "\n"
cat "./Test/test$n.txt" | tr -d '\n'
if cmp "./res_dec_h.txt" "./Test/true_out$n.txt" > /dev/null; then
    printf "\n*** - Passed\n"
else
    printf "\n*** - Failed\n"
fi
printf "Desired result:\n"
cat "./Test/true_out$n.txt"
printf "Actual result:\n"
cat "./res_dec_h.txt"
printf "\n===== \n"

done
for n in {4..4}
do
    printf "Test$n:\n"
    ./a.out < Test/test$n.txt
    printf "\n"
    cat "./Test/test$n.txt" | tr -d '\n'
    if cmp "./res_dec_f.txt" "./Test/true_out$n.txt" > /dev/null; then
        printf "\n*** - Passed\n"
    else
        printf "\n*** - Failed\n"
    fi
    printf "Desired result:\n"
    cat "./Test/true_out$n.txt"
    printf "Actual result:\n"
    cat "./res_dec_f.txt"
    printf "\n===== \n"
done
for n in {5..5}
do
    printf "Test$n:\n"
    ./a.out <Test/test$n.txt > out$n.txt
    printf "qw\n"
    if cmp "./out$n.txt" "./Test/true_out$n.txt" > /dev/null; then
        printf "*** - Passed\n"
    else
        printf "*** - Failed\n"
    fi
    printf "Desired result:\n"
    cat "./Test/true_out$n.txt"
    printf "Actual result:\n"
    cat "./out$n.txt"
    printf "\n===== \n"
done

```

**Имя файла: generator.cpp**

```

#include <iostream>
#include <ctime>
#include <string>
#include <fstream>
#define SIZE 40000

int main() {
    unsigned long long int val=0;
    std::ofstream file;
    file.open("tests_for_stats.txt");
    std::string str="";
    srand(time(nullptr));
    for(unsigned long long int i = 0; i < SIZE/3; i++){
        val+=3;
        if (val>=4000){
            val=0;
            file<<str;
            str="";
        }
        str+=char('a' + rand() % ('a' - 'z'));
        str+=char(' ' + rand() % (' ' - '@'));
        str+=char('A' + rand() % ('A' - 'Z'));
    }
    std::cout<<str;
    file<<str;
    file.close();

    return 0;
}

```

### **Имя файла: Makefile**

```

cw: Source/main.cpp
    g++ -std=c++17 Source/main.cpp
run_tests: cw
    ./tests_script

```