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. 9384 _____ Соседков К.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)

Студент Давыдов Д.С.

Группа 9384

Тема работы : Кодирование и декодирование методами Хаффмана и Фано-Шеннона - исследование

Исходные данные:
Генерация входных данных, использование их для измерения количественных характеристик структур данных, алгоритмов, действий, сравнение экспериментальных результатов с теоретическими

Содержание пояснительной записки:
«Содержание» «Введение» «Описание работы программы » «Визуализация» «Сопоставление с теоретическими оценками» «Тестирование» «Заключение» «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 15.11.2020

Дата сдачи курсовой работы: 05.02.2020

Дата защиты курсовой работы: 05.02.2020

Студент _____ Давыдов Д. С.

Преподаватель _____ Ефремов М.А.

АННОТАЦИЯ

Основная цель данного курсового проекта — это исследование методов кодирования и декодирования Хаффмана и Фано-Шеннона. Исследование будет представлять из себя генерацию данных, работу с ними, а также визуализацию сложности написанных алгоритмов, осуществляющих эту работу. Все расчеты будут производится с помощью стандартной библиотеки C++, а для визуализации и написания приложения со всеми необходимыми функциями для анализа полученных данных будет использован фреймворк Qt.

SUMMARY

The main goal of this course project is the explore Huffman and Fano-Shannon encoding and decoding methods. The research will consist of generating data, working with it, as well as visualization of the complexity of the written algorithms, that carry out this work. All calculations will be performed using the C ++ standard library, and the Qt framework will be used to visualize and write an application with all the necessary functions for analyzing the received data.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной курсовой работе будут представлены алгоритмы осуществляющие методы кодирования и декодирования Хаффмана и Фано-Шеннона. Для сравнительного анализа этих методов будут произведены необходимы тесты, представляющие из себя всевозможные графики для различных наборов данных.

1. ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

1.1. Основные теоретические сведения

Целью курсовой работы является исследование статических методов кодирования и декодирования Хаффмана и Шеннона — Фано.

Кодированию предшествует построение бинарного дерева. Далее алгоритм проходит по нему к каждому листу и получает бинарный код каждого закодированного символа.

Декодирование же представляет из себя считывание каждого бита закодированного сообщения и одновременного движения по бинарному дереву дот того, как алгоритм дойдет до листа с символом.

1.2. Описание методов и полей.

Для реализации кодирования и декодирования методом Хаффмана был разработан класс HuffmanTree со следующими методами и полями:

l) buildHuffmanTree – подсчет символов в строчке, постройка бинарного дерева, его кодировка (получение кодов символов), а также декодирование.

2) comp – компаратор для сравнения частот символов узлов/листьев бинарного дерева.

3) GetNode - создание узла/листка бинарного дерева.

4) IsLeaf – проверка вершины, на то, является ли она листком.

5) HuffmanCode – словарь, где ключи — это символы, а значения — бинарные коды этих символов.

Для реализации кодирования и декодирования методом Шеннона — Фано был разработан класс ShannonCode со следующими методами и полями:

1) encodeDecode — подсчет символов в строчке, тут же вызываются остальные методы.

2) buildFunc – построение виртуального дерева и поучения кодов символов

3) decode – декодирование сообщения путем последовательного считывания битов, добавлении их в строчку и проверки, не является ли эта строчка ключом такой ключ в словаре символов. Если есть

4) shannon_code1 – словарь, где ключи — это коды символов, а значения — эти самые символы.

Алгоритм Хаффмана работает следующим образом — создается очередь, которая заполняется «листьями» - узлами с частотой встречи определенного символа.

Так как это очередь с приоритетом, то после данного цикла, она будет отсортирована по убыванию.

Далее в цикле while будут доставаться два последних узла из очереди (с самыми наименьшими весами), их веса будут складываться и будет создаваться новый узел, который будет иметь пути к предыдущим двум узлам. Так будет продолжаться, пока размер очереди не будет равен 1.

Далее с помощью прямого хода по полученному бинарному дереву мы будем получать коды символов.

Разработанный программный код смотреть в приложении А.

1.3 Генерация данных и накопление статистики

Есть четыре возможных выбора генерации данных

Первый - средний случай. Пользователь вводит количество символов в тексте и программа заполняет строчку различными и не различными 95 ASCII символами).

Второй — худший случай. Пользователь вводит количество символов в тексте и программа заполняет строчку различными 126 ASCII символами. Поэтому для такой генерации стоит ограничение в 126 символов.

Третий — лучший случай. Пользователь вводит количество символов в тексте и программа заполняет всю строчку одним из 95 ASCII символов.

Четвертый — пользовательский ввод. Пользователь вводит собственную строчку. В цикле к строчке для кодирования последовательно добавляются символы из строки пользователя.

2. ВИЗУАЛИЗАЦИЯ

Визуализация полученной статистики выполняется с помощью классов QChart и QGraphicsView фреймворка Qt.

Есть 3 варианта отображения сгенерированных данных

Первый (Задать данные для программы → любой из пунктов) — показывается 4 графика. Первый показывает зависимость времени работы алгоритма кодировки методом Хаффмана в наносекундах от количества сгенерированных данных (количества символов), второй, третий и четвертый — те же самые зависимости, но уже для алгоритма декодирования методом Хаффмана, алгоритма кодирования методом Фано-Шеннона и алгоритм декодирования методом Фано-Шеннона. См. Рис.1.

Второй (Вид графа→ наложить графики) — показывает 2 графика с наложенными друга на друга числовыми рядами (зависимости времени работы алгоритма в наносекундах от количества данных для методов Хаффмана и Фано-Шеннона). Также для алгоритмов Фано-Шеннона справа была добавлена дополнительная ось, т.к между двумя числовыми рядами присутствует существенный разброс. См. Рис. 1.2.

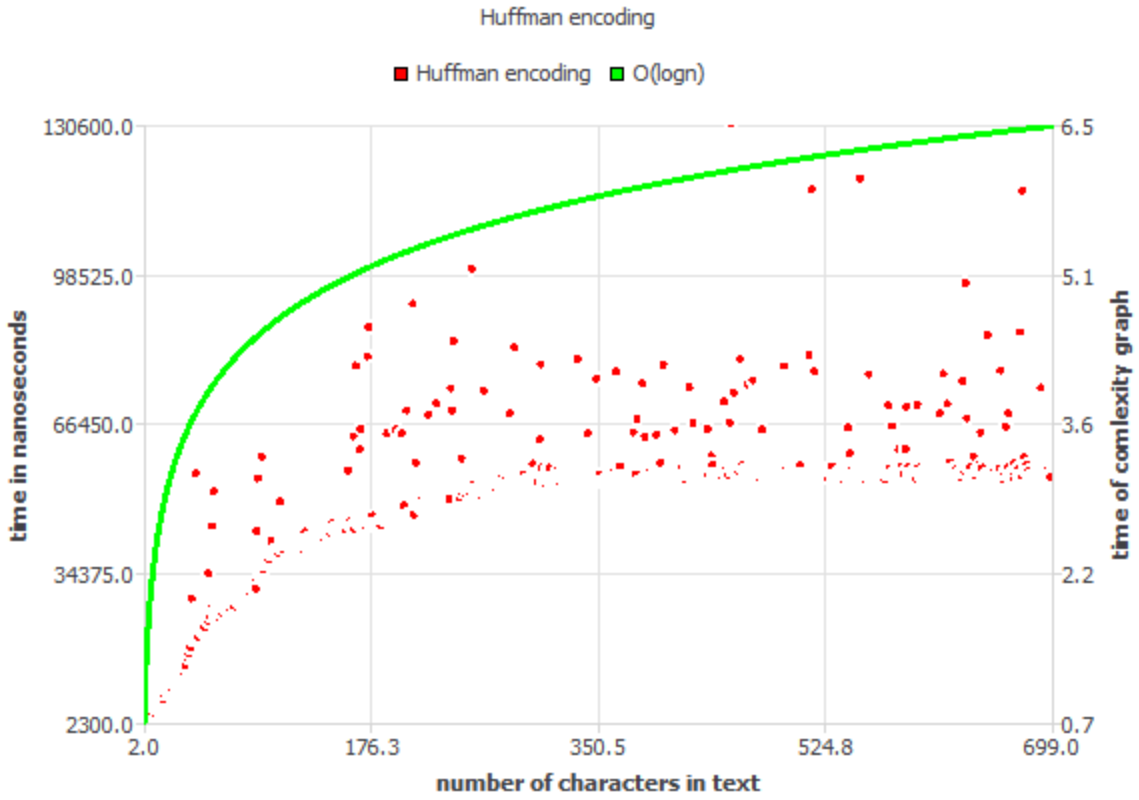
Третий (Вид→ Добавить графики сложностей) — аналогичен первому графику, но него были добавлены графики теоретических оценок сложностей алгоритмов. См. Рис. 1.3.

Изображения будут представлены в разделе «сопоставления с теоретическими оценками».

3. СОПОСТАВЛЕНИЕ С ТЕОРЕТИЧЕСКИМИ ОЦЕНКАМИ

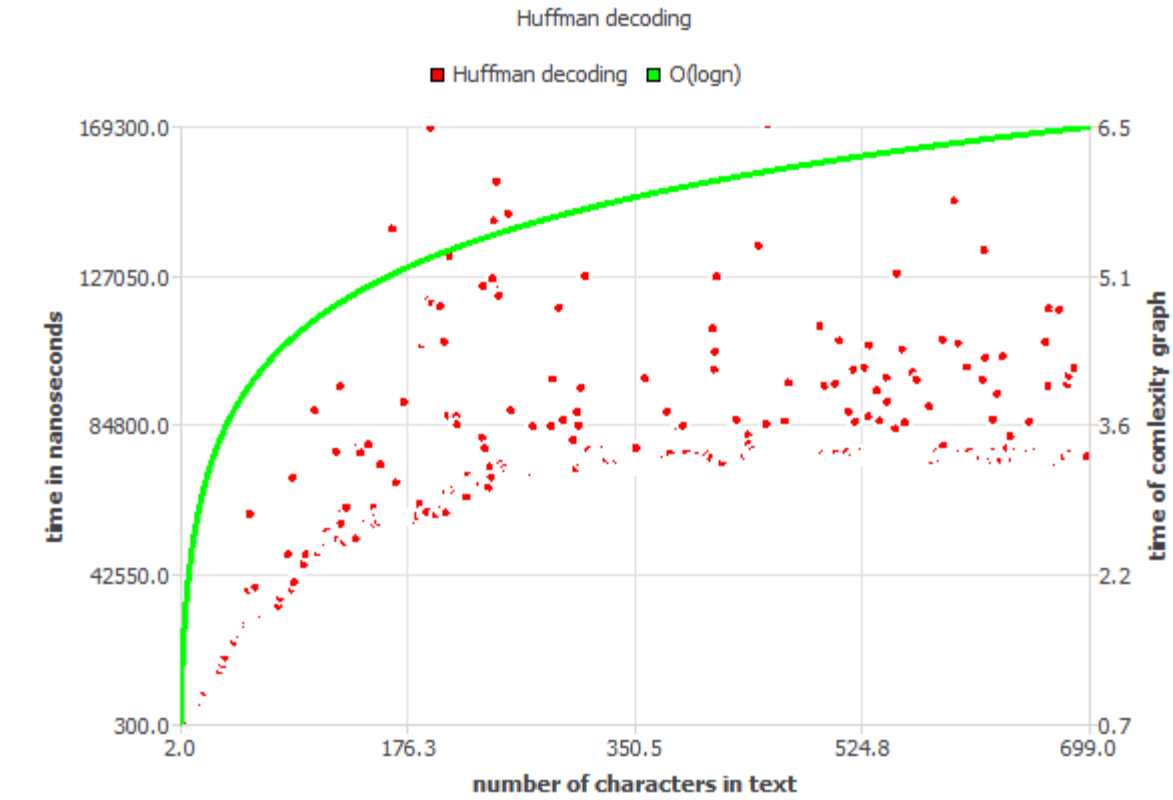
Кодирование методом Хаффмана

рис. 2



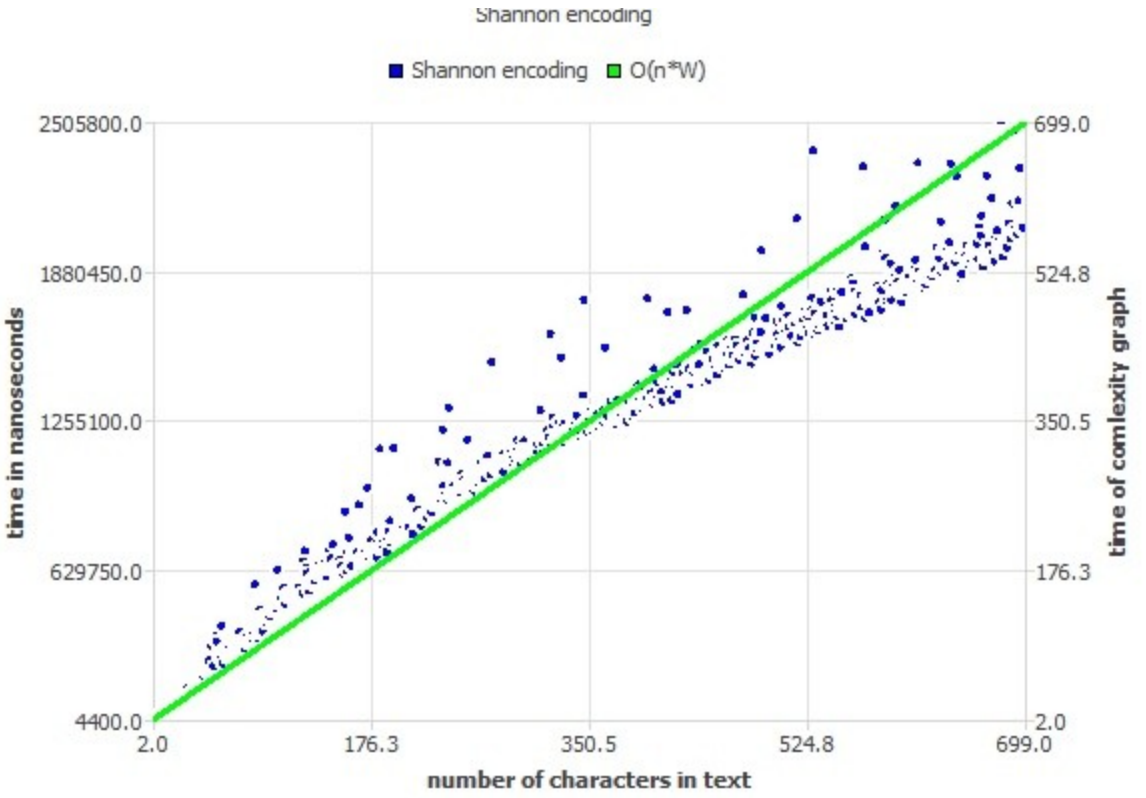
Декодирование методом Хаффмана

рис 2.1



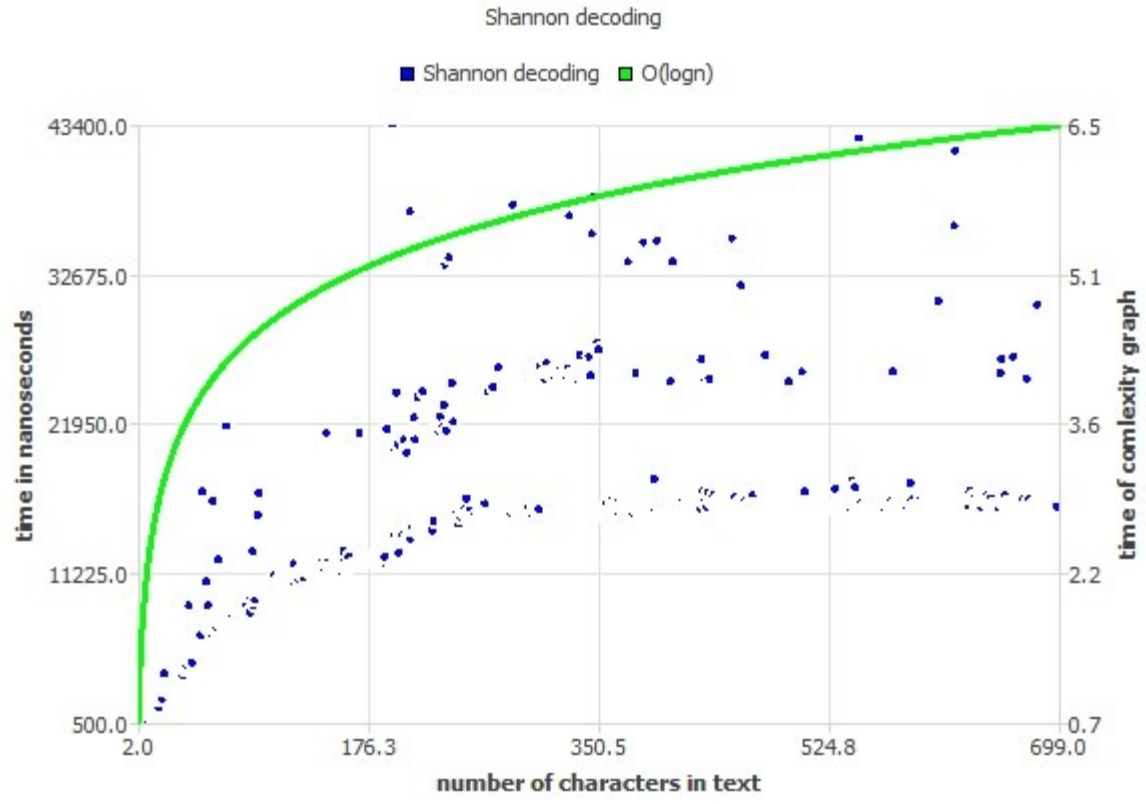
кодирование методом Фанно-Шеннона

рис. 2.3



декодирование методом Фанно-Шеннона

рис. 2.4



Теоретическая сложность методов кодирования и декодирования Хаффмана оценивается как O(logn). На графиках рис. 1 и рис. 2 можно заменить, что скопления точек как раз похожи на функцию logn. Сначала идет рост, а потом значения выходят на так называемое плато — вне зависимости от количества данных, кодирование или декодирование в среднем будет занимать одно время. На обоих графиках это плато начинается примерно в диапазоне от 250 - 300 символов. Связано это с тем, что примерно при таких значениях, одинаковых символов из 95 возможных будет достаточно для того, чтобы алгоритм не строил дополнительных ветвей бинарного дерева, что как раз и увеличивает время его выполнения. Тоже самое касается и декодирования.

Само плато находится на примерно одних и тех же прямых времени, различия между методами кодирования и декодирования в этом плане составляет где-то 30 000 наносекунд или же 0.03 миллисекунды.

Рассмотренный в первом разделе алгоритм для кодирования методом Фано-Шеннона имеет сложность O(W*n/2) + O(n/2) + O(W/2) + O(W/2*n/4) + O(n/4)+ O(W/4) + ... ≈ O(n*W), где W – это количество различных символов. На рис 2.3 скопления точек повторяют эту функцию. Различие во времени между кодированием Хаффмана и Фано-Шеннона довольно большое. При n = 700 кодирование занимает вплоть до 2 миллионов наносекунд (2 миллисекунды, в то время как у алгоритма на рис 1. среднее значение — это 60000 наносекунд).

Связано это с тем, что алгоритм перебирает всевозможные варианты для разделения группы символов на две части, сумма весов которых будет максимально равна.

Декодирование в свою очередь имеет сложность O(logn). На рис 2.4 видно, что скопления точек примерно повторяют форму графика функции logn. Плато в свою очередь начинается при том же количестве данных, что и у метода декодирования Хаффмана, но при этом времени занимает меньше. В отличии от одновременного считывания битов закодированного сообщения и проходу по бинарному дереву Хаффмана, данный алгоритм работает следующим образом: к строчке последовательно присоединяются биты

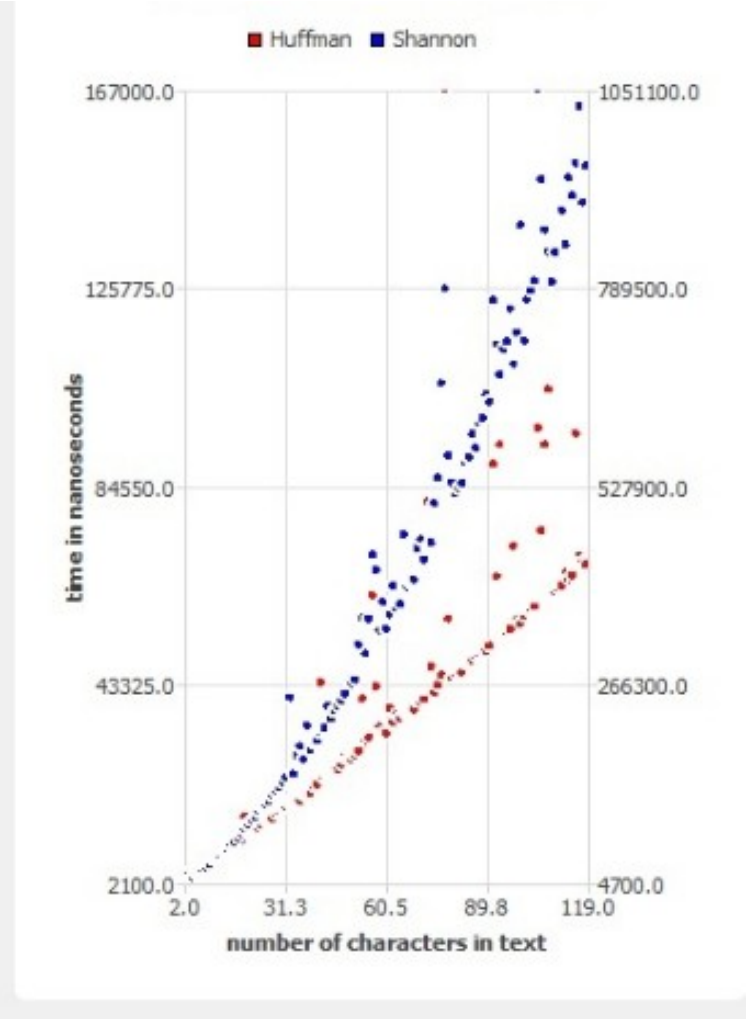
закодированного сообщения , далее проверяется не является ли эта строчка кодом какого либо символа. Если является, то получаем символ по данному ключу из словаря, аннулируем строчку для дальнейшего поиска. Так как пары код-символ (ключ-значение) хранятся в структуре данных std::map как сбалансированное двоичного дерево поиска, сложность поиска в нем будет равна O(logn), как и в случае с реализованным в этой курсовой работе бинарным деревом для кодировании методом Хаффмана. Разница во времени объясняется очевидно более качественно реализованной структурой стандартной библиотеки.

Рассмотрим теперь худший случай. На графике непросто увидеть схожесть расположения точек относительно графика теоретической сложности каждого из методов, т.к максимально возможное число данных равно 126, потому что строчка должна состоять из не повторяющихся символов.

Для метода кодирования Фано-Шеннона, у нас будет n символов и W различных символов, где W = n. Тогда сложность будет равна O(n^2), это также немного прослеживается на совместном с методом кодирования Хаффмана графике, (рис 2.5), у которого теоретическая сложность в худшем случае равна O(n).

Кодирование методами Хаффмана и Фано-Шеннона

рис 2.5

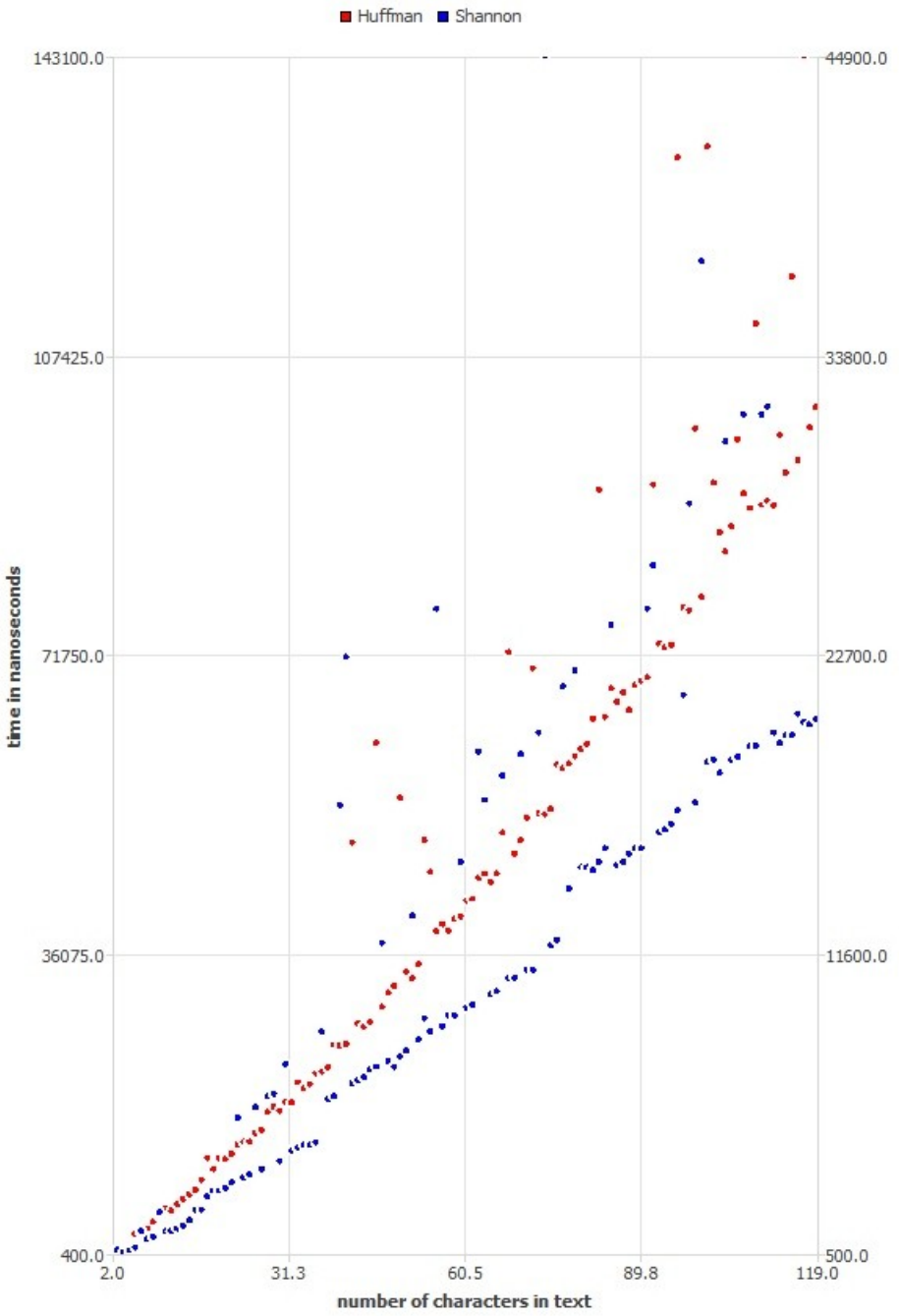


По правой оси Y, отвечающей за значения алгоритма кодирования Фано-Шеннона, видно, что в сравнении с графом алгоритма кодирования Хаффмана, он выполняется за намного больший промежуток времени.

Сложность алгоритмов декодирования также имеют сложность O(n). Значения времени их выполнения отличаются не сильно (рис 2.6).

Декодирование методами Фано-Шеннона и Хафмана

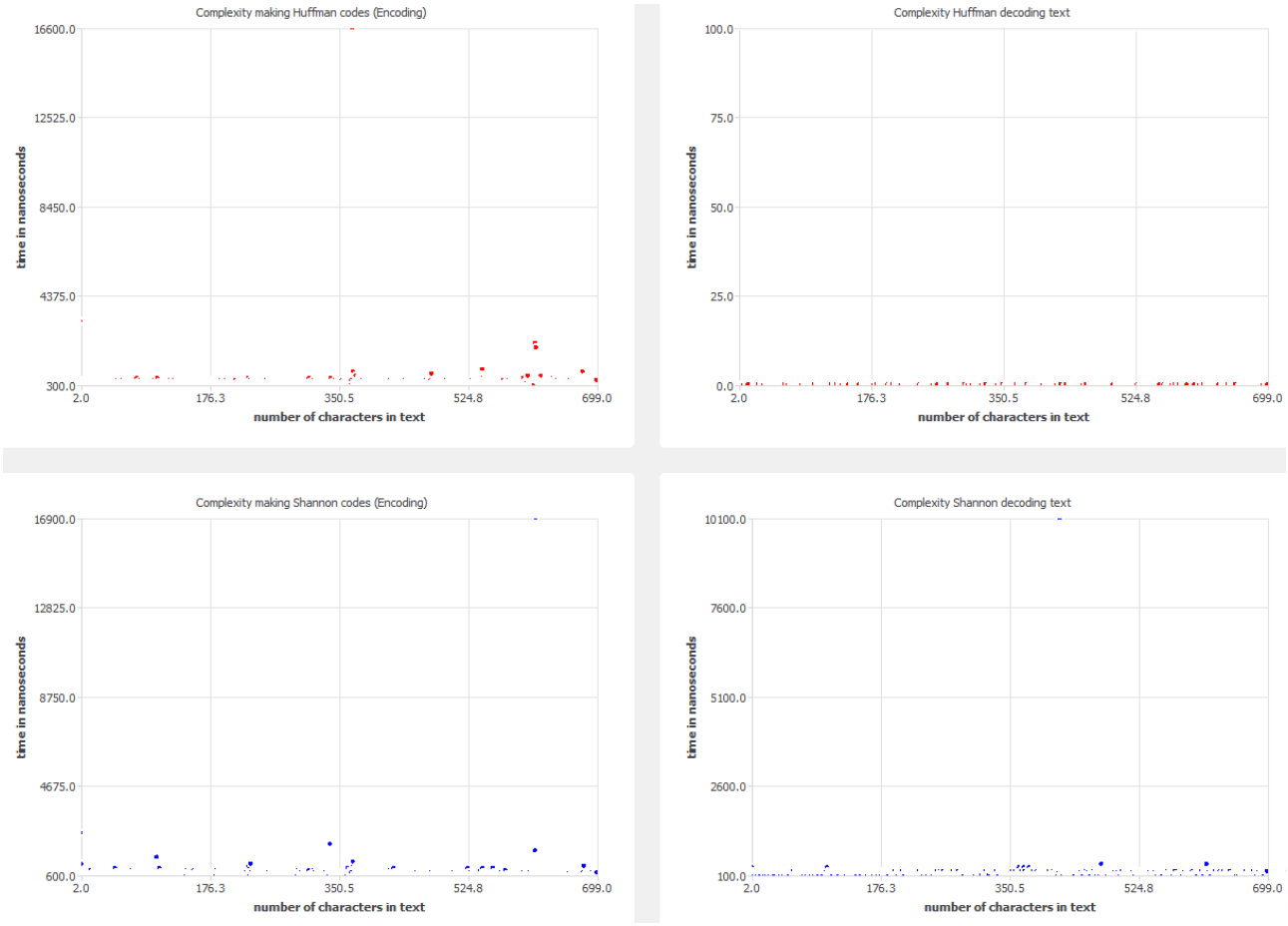
Рис. 2.6



Рассмотрим теперь лучший случай. Для всех алгоритмов кодирования и декодирования сложность равна константе, что видно на графике, хоть и некоторые точки немного ушли под ось x (рис 2.7).

Кодирование и декодирование методами Хаффана и Фано-Шеннона

Рис. 2.7



4. ТЕСТИРОВАНИЕ

Ниже представлены изображения показывающие результат работы программы при вводе разного количества символов в среднем случае, а также при ручном вводе текста.

Рис 3. Количество данных равно 5000.

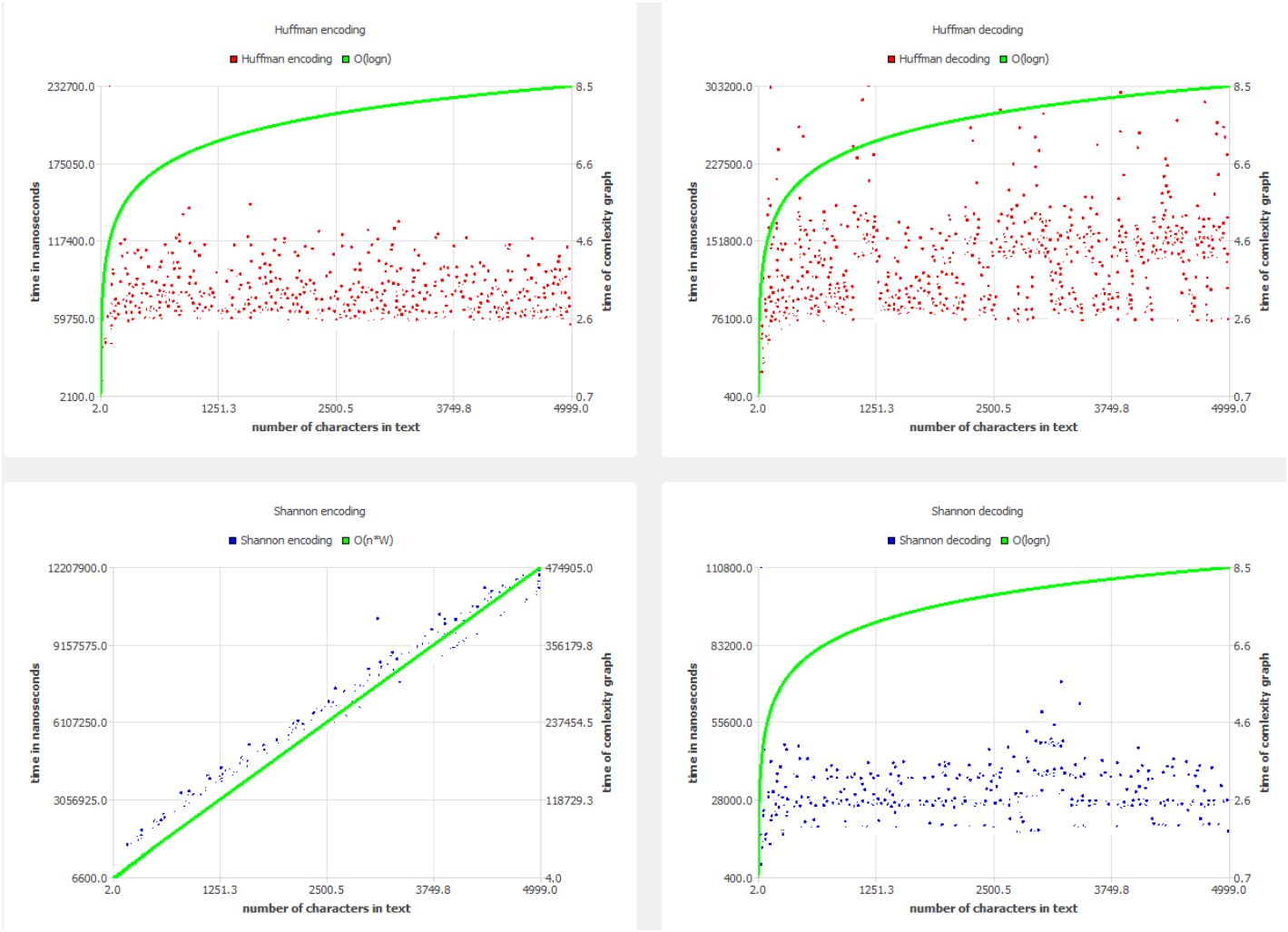
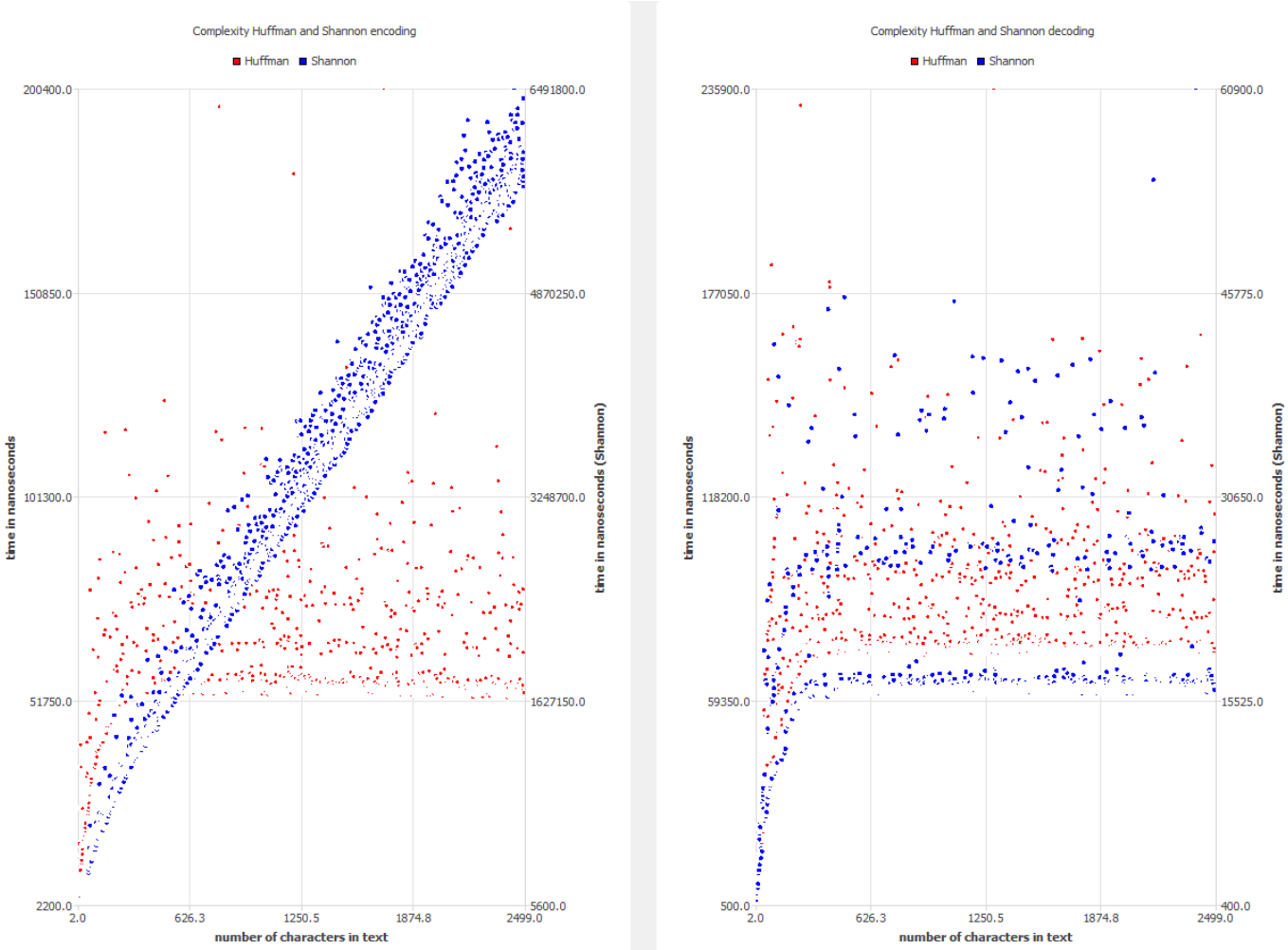


Рис 3.2 Количество данных равно 2500

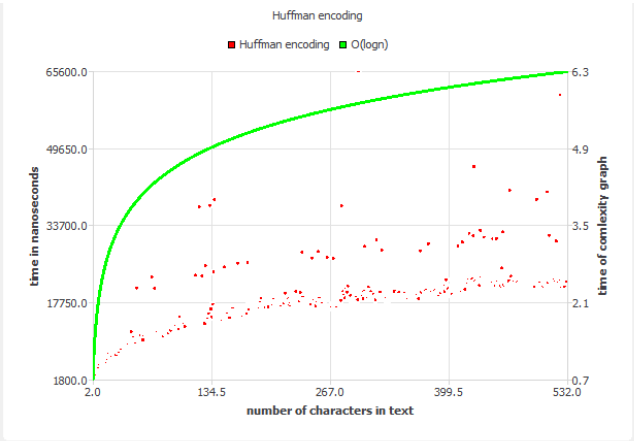


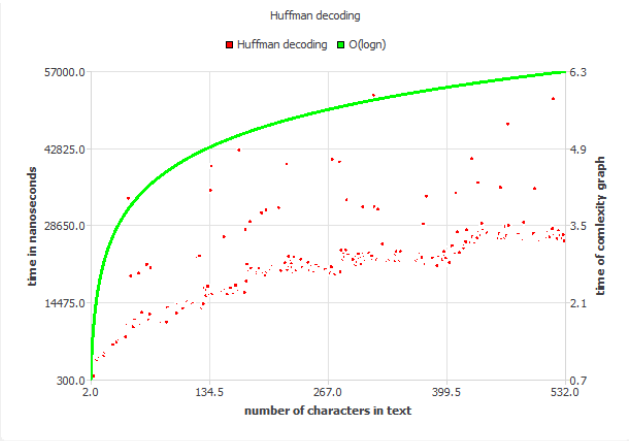
.

Рисунок 3.3. Пользователь ввел текст:

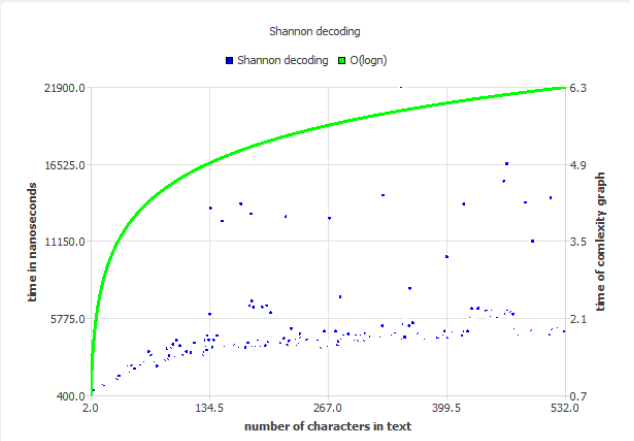
The universe is enormous, so the chances of us being the only living creatures are small. Although we think we are intelligent and that we know a lot about Space, we have only explored a very small area. We might be the only creatures that can travel in Space, but it is unlikely.

In fact, some people say that we might have been visited by aliens. These people point to ‘wonders’ such as Stonehenge in Britain and the Nazca lines in Peru as proof that aliens have been here. So, what are the chances that there is life out there?


Huffman encoding
Huffman encoding
O(logn)
65600.0
49650.0
33700.0
17750.0
1800.0
6.3
4.9
3.5
2.1
0.7
2.0
134.5
267.0
399.5
532.0
time in nanoseconds
time of comlexity graph
number of characters in text


Huffman decoding
Huffman decoding
O(logn)
57000.0
42825.0
28650.0
14475.0
300.0
6.3
4.9
3.5
2.1
0.7
2.0
134.5
267.0
399.5
532.0
time in nanoseconds
time of comlexity graph
number of characters in text


Shannon encoding
Shannon encoding
O(n*W)
952900.0
715750.0
478600.0
241450.0
4300.0
20216.0
15163.0
10110.0
5057.0
4.0
2.0
134.5
267.0
399.5
532.0
time in nanoseconds
time of comlexity graph
number of characters in text


Shannon decoding
Shannon decoding
O(logn)
21900.0
16525.0
11150.0
5775.0
400.0
6.3
4.9
3.5
2.1
0.7
2.0
134.5
267.0
399.5
532.0
time in nanoseconds
time of comlexity graph
number of characters in text

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была написана программа, позволяющая выполнить алгоритмы методов кодирования и декодирования Хаффмана и Фано-Шеннона при различных данных, накопить статистику и вывести её в виде графов для её сравнительного анализа.

Построенные графы соответствовали теоретическим оценкам и позволили сделать следующие заключения: метод кодирования Фано-Шеннона является медленным относительно метода кодирования Хаффмана, декодирование методом Хаффмана и Фано-Шеннона имеют почти одинаковое время работы, а плато, при котором время работы алгоритма не увеличивается, образуется примерно в одном и том же диапазоне значений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация // https://doc.qt.io/ // URL: https://doc.qt.io/qt-5/qtcharts-index.html (дата обращения 30.12.2020)

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <QApplication>
#include <QtWidgets/QMainWindow>
#include <QtCharts/QChartView>
#include <QtCharts/QBarSeries>
#include <QtCharts/QBarSet>
#include <QtCharts/QLegend>
#include <QtCharts/QBarCategoryAxis>
#include <QtCharts/QHorizontalStackedBarSeries>
#include <QtCharts/QLineSeries>
#include <QtCharts/QCategoryAxis>
#include <QtCharts/QPieSeries>
#include <QtCharts/QPieSeries>
#include <QtCharts/QPieSlice>
#include <QLabel>
#include <QObject>
#include <QLineEdit>
#include <QTableView>
#include <QHBoxLayout>
#include "mainwindow.h"
#include "huffmantree.h"
#include "shannoncode.h"
#include <QDebug>
#include <QScatterSeries>
#include <QColor>
#include <cmath>
#include <QLogValueAxis>
#include <QLineEdit>
#include <QIntValidator>
#include <QSpinBox>
#include <QMenuBar>

using namespace QtCharts;
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Название файла: mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <huffmantree.h>

QT_BEGIN_NAMESPACE
```

```
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:


    void on_action_triggered();

    void on_action_2_triggered();

    void on_action_3_triggered();

    void on_action_4_triggered();

    void on_action_5_triggered();

    void on_action_6_triggered();

private:

    std::vector< std::pair <int,double> > vectHuffEncode;
    std::vector< std::pair <int,double> > vectHuffDecode;
    std::vector< std::pair <int,double> > vectShannonEncode;
    std::vector< std::pair <int,double> > vectShannonDecode;
    std::vector<int> numberOfCharacters;

    static std::string generateAverageCase(int amount);
    static std::string generateWorstCase(int amount);
    static std::string generateBestCase(int amount);
    static std::string generateManuallyCase(int amount);
    QtCharts::QChart * getChart(QtCharts::QScatterSeries *series, QColor
color, QString graphName, QString axisXname, QString axisYname);
    QtCharts::QChart * getChartUnite(QtCharts::QScatterSeries *series,
QtCharts::QScatterSeries *series2, QString graphName, QString axisXname,
QString axisYname);
    QtCharts::QChart * getChartWithComlexity(QtCharts::QScatterSeries
*series, QtCharts::QLineSeries *series2,  QString graphName, QString
seriesName, QColor color);
    void plotResult(int charsAmount, std::string(*generate)(int), QString
userText = "");
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Название файла: shannoncode.h

```
#ifndef SHANNONCODE_H
#define SHANNONCODE_H
#include <iostream>
#include <map>
#include <string>
```

```
#include <vector>
#include <QScatterSeries>
#include <unordered_map>
#include <ctime>
#include <QLineSeries>
#define EMPTY_STRING ""

class ShannonCode
{
public:
    void encodeDecode(std::string text, QtCharts::QScatterSeries
*series1,QtCharts::QScatterSeries *series2, std::vector< std::pair
<int,double> > &vect,
                std::vector< std::pair <int,double> > &vect2);
private:

    void buildFunc(std::map<char, int> map1, std::string code, int W2);
    std::map<char, std::string> shannon_code;
    std::map<std::string, char> shannon_code1;
    void decode(std::string text);
};

#endif // SHANNONCODE_H
```

Название файла: huffmantree.h

```
#ifndef HUFFMANTREE_H
#define HUFFMANTREE_H

#include <QApplication>
#include <QtWidgets/QMainWindow>
#include <QtCharts/QChartView>
#include <QtCharts/QBarSeries>
#include <QtCharts/QBarSet>
#include <QtCharts/QLegend>
#include <QtCharts/QBarCategoryAxis>
#include <QtCharts/QHorizontalStackedBarSeries>
#include <QtCharts/QLineSeries>
#include <QtCharts/QCategoryAxis>
#include <QtCharts/QPieSeries>
#include <QtCharts/QPieSeries>
#include <QtCharts/QPieSlice>
#include <QLabel>
#include <QObject>
#include <QLineEdit>
#include <QTableView>
#include <QHBoxLayout>
#include <QMainWindow>
#include <QObject>
#include <QWidget>
#include <iostream>
#include <string>
#include <ctime>
#include <fstream>
#include <map>
#include <queue>
#include <unordered_map>
```

```
#include <chrono>
#include <QDebug>
#include <QScatterSeries>

#define EMPTY_STRING ""

class HuffmanTree
{

private:

    QtCharts::QChartView *chartView;
    struct Node{
            char ch;
            char character;
            int frequency;
            Node *left, *right;
        };

        Node* root = nullptr;
        std::map<char, std::string> huffmanCode;
        bool isLeaf(Node* root);
        Node* getNode(char ch, int freq, Node* left, Node* right);
        void encode(Node* root, std::string str = EMPTY_STRING);

        struct comp{
            bool operator()(Node* l, Node* r){
                return l->frequency > r->frequency;
            }
        };

public:
    void buildHuffmanTree(std::string text, QtCharts::QScatterSeries
*series1, QtCharts::QScatterSeries *series2, std::vector< std::pair
<int,double> > &vect,
                          std::vector< std::pair <int,double> > &vect2,
std::vector<int> &unique);
    void decode(Node* root, int &index, std::string str);
};



#endif // HUFFMANTREE_H
```

Название файла: huffmantree.cpp

```
#include "huffmantree.h"
#include "cmath"
#include <QDebug>
void HuffmanTree::buildHuffmanTree(std::string text,
QtCharts::QScatterSeries *series1, QtCharts::QScatterSeries *series2,
                                   std::vector< std::pair <int,double> >
&vect, std::vector< std::pair <int,double> > &vect2, std::vector<int>
&unique){
    if (text == EMPTY_STRING) {
        return;
    }
```

```
    std::unordered_map<char, int> freq;
    for (char ch: text) {
        freq[ch]++;
    }

    unique.push_back(freq.size());

    std::priority_queue<Node*, std::vector<Node*>, comp> pq;

    for (auto pair: freq) {
        pq.push(getNode(pair.first, pair.second, nullptr, nullptr));

    }

    while (pq.size() != 1){
        Node *left = pq.top();
        pq.pop();
        Node *right = pq.top();
        pq.pop();

        int sum = left->frequency + right->frequency;
        pq.push(getNode('\0', sum, left, right));

    }

    root = pq.top();

    auto start1 = std::chrono::steady_clock::now();
    encode(root,EMPTY_STRING);
    auto end1 = std::chrono::steady_clock::now();
    auto diff1 = end1 - start1;

    series1->append(text.size(), std::chrono::duration <double,
std::nano> (diff1).count());
    vect.push_back(std::make_pair(text.size(),std::chrono::duration
<double, std::nano> (diff1).count()));

    srand(time(NULL));
    std::string stringNew;
    for(int i = 0;i<huffmanCode.size();i++){
        auto it = huffmanCode.begin();
        std::advance(it, rand()% huffmanCode.size());
        stringNew+=huffmanCode[it->first];
    }

    auto start2 = std::chrono::steady_clock::now();
    int index = -1;
    while(index < (int)stringNew.size()-1){
        decode(root,index,stringNew);
    }
    auto end2 = std::chrono::steady_clock::now();
    auto diff2 = end2 - start2;
    series2->append(text.size(), std::chrono::duration <double,
std::nano> (diff2).count());
    vect2.push_back(std::make_pair(text.size(),std::chrono::duration
<double, std::nano> (diff2).count()));
```

```
    int bitsCount = 0;
    for (auto it = huffmanCode.begin(); it != huffmanCode.end(); ++it){
        bitsCount+=it->second.size();
    }

    qDebug()<<"number of bits for Huffman encode: "<<bitsCount<<'\n';
}

void HuffmanTree::encode(HuffmanTree::Node *root, std::string str) {
    if (root == nullptr) {
        return;
    }

    if (isLeaf(root)) {
        huffmanCode[root->ch] = (str != EMPTY_STRING) ? str : "1";
    }

    encode(root->left, str + "0");
    encode(root->right, str + "1");
}

HuffmanTree::Node *HuffmanTree::getNode(char ch, int freq,
HuffmanTree::Node *left, HuffmanTree::Node *right) {
    Node* node = new Node();

    node->ch = ch;
    node->frequency = freq;
    node->left = left;
    node->right = right;

    return node;
}

bool HuffmanTree::isLeaf(HuffmanTree::Node *root) {
    return root->left == nullptr && root->right == nullptr;
}

void HuffmanTree::decode(HuffmanTree::Node *root, int &index, std::string
str) {
    if (root == nullptr) {
        return;
    }

    if (isLeaf(root)) {
        return;
    }

    index++;

    if (str[index] == '0') {
        decode(root->left, index, str);
    } else {
        decode(root->right, index, str);
    }
}
```

Название файла: mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QInputDialog>
#include <iostream>
#include <huffmantree.h>
#include <QScatterSeries>
#include <shannoncode.h>
#include <QActionGroup>
#include <QRadioButton>
#include <QMessageBox>
#include <QCheckBox>
#include <QValueAxis>
#include <cmath>
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}


void MainWindow::on_action_2_triggered()
{
    plotResult(QInputDialog::getInt(this, "Ввод", "Введите количетсво
символов", 2, 2,126), generateWorstCase);
}

void MainWindow::on_action_triggered()
{
    plotResult(QInputDialog::getInt(this,"Ввод","Введите количество
символов", 2, 2), generateAverageCase);
}

void MainWindow::plotResult(int charsAmount, std::string(*generate)(int
amount), QString userText){

    //Creating series
    QtCharts::QScatterSeries *seriesHuffmanEncode = new
QtCharts::QScatterSeries();
    QtCharts::QScatterSeries *seriesHuffmanDecode = new
QtCharts::QScatterSeries();
    QtCharts::QScatterSeries *seriesShannonEncode = new
QtCharts::QScatterSeries();
    QtCharts::QScatterSeries *seriesShannonDecode = new
QtCharts::QScatterSeries();

    vectHuffEncode.clear();
    vectHuffDecode.clear();
    vectShannonDecode.clear();
    vectShannonEncode.clear();
    numberOfCharacters.clear();
```

```
    for(int i = 0;i<userText.size();i++){
        QChar c = userText.at(i);
        qDebug()<<c;
    }

    //filling series
    std::string text;
    srand(time(NULL));
    for(int i = 2; i<charsAmount;i++){
        HuffmanTree huffman;
        ShannonCode shannon;
        if(userText == ""){
            text = generate(i);
        } else {
           for(int j =0; j<i; j++){
               text.push_back(userText[j].toLatin1());
           }
        }

        huffman.buildHuffmanTree(text, seriesHuffmanEncode,
seriesHuffmanDecode, vectHuffEncode, vectHuffDecode,numberOfCharacters);
        shannon.encodeDecode(text, seriesShannonEncode,
seriesShannonDecode, vectShannonEncode, vectShannonDecode);
        text = "";
    }

    //Creating charts
    QtCharts::QChart *chartHuffmanEncode = getChart(seriesHuffmanEncode,
Qt::red,
                                                    "Complexity making
Huffman codes (Encoding)",
                                                    "number of characters
in text",
                                                    "time in
nanoseconds");
    QtCharts::QChart *chartHuffmanDecode = getChart(seriesHuffmanDecode,
Qt::red,
                                                    "Complexity Huffman
decoding text",
                                                    "number of characters
in text",
                                                    "time in
nanoseconds");

    QtCharts::QChart *chartShannonEncode = getChart(seriesShannonEncode,
Qt::blue,
                                                   "Complexity making
Shannon codes (Encoding)",
                                                    "number of characters
in text",
                                                    "time in
nanoseconds");

    QtCharts::QChart *chartShannonDecode = getChart(seriesShannonDecode,
Qt::blue,
                                                    "Complexity Shannon
decoding text",
```

```
                                                        "number of characters
in text",
                                                        "time in
nanoseconds");


    QtCharts::QChartView *chartViewHE = new
QtCharts::QChartView(chartHuffmanEncode);
    QtCharts::QChartView *chartViewHD = new
QtCharts::QChartView(chartHuffmanDecode);
    QtCharts::QChartView *chartViewSE = new
QtCharts::QChartView(chartShannonEncode);
    QtCharts::QChartView *chartViewSD = new
QtCharts::QChartView(chartShannonDecode);


    QGridLayout *Layout1 = new QGridLayout();
    Layout1->addWidget(chartViewHE,0,0);
    Layout1->addWidget(chartViewHD,0,1);
    Layout1->addWidget(chartViewSE,1,0);
    Layout1->addWidget(chartViewSD,1,1);

   // ui->scrollAreaWidgetContents->setLayout(Layout1);
    QWidget *windowLayot = new QWidget();
    windowLayot->setLayout(Layout1);
    setCentralWidget(windowLayot);
    show();
}

std::string MainWindow::generateAverageCase(int amount){
    int r;
    char c;
    std::string text;
    for(int j = 0; j < amount; j++){
        r = rand() % 95;
        c = ' ' + r;
        text.push_back(c);
    }
    return text;
};

std::string MainWindow::generateWorstCase(int amount){
    char c;
    int r;
    std::vector<char> symbols;
    std::string text;
    for(int i = 0; i <= 126; i++){
        c = 0 + i;
        symbols.push_back(c);
    }

    for(int i = 0; i<amount;i++){
        r = rand() % symbols.size();
        c = symbols[r];
        text.push_back(c);
        symbols.erase(symbols.begin() + r);
    }
```

```
    return text;
}

std::string MainWindow::generateBestCase(int amount){
    char c;
    int r;
    std::string text;
    r = rand() % 95;

    for(int i = 0; i<amount; i++){
        c = ' ' + r;
        text.push_back(c);
    }

    return text;
}

QtCharts::QChart * MainWindow::getChart(QtCharts::QScatterSeries *series,
QColor color, QString graphName, QString axisXname, QString axisYname){

    QtCharts::QChart *chart = new QtCharts::QChart();
    series->setMarkerSize(5);
    series->setColor(color);
    chart->legend()->hide();
    chart->addSeries(series);
    chart->setTitle(graphName);
    QtCharts::QValueAxis *axisX = new QtCharts::QValueAxis();
    QtCharts::QValueAxis *axisY = new QtCharts::QValueAxis();
    axisX->setTitleText(axisXname);
    axisY->setTitleText(axisYname);
    chart->addAxis(axisX, Qt::AlignBottom);
    chart->addAxis(axisY, Qt::AlignLeft);
    series->attachAxis(axisX);
    series->attachAxis(axisY);
    return chart;
};

QtCharts::QChart * MainWindow::getChartUnite(QtCharts::QScatterSeries
*series, QtCharts::QScatterSeries *series2, QString graphName, QString
axisXname, QString axisYname){
    QtCharts::QChart *chart = new QtCharts::QChart();
    series->setMarkerSize(5);
    series->setColor(Qt::red);
    series->setName("Huffman");
    series2->setMarkerSize(6);
    series2->setColor(Qt::blue);
    series2->setName("Shannon");
    chart->addSeries(series);
    chart->addSeries(series2);
    chart->setTitle(graphName);
    QtCharts::QValueAxis *axisX = new QtCharts::QValueAxis();
    QtCharts::QValueAxis *axisY = new QtCharts::QValueAxis();
    QtCharts::QValueAxis *axisY2 = new QtCharts::QValueAxis();
    axisX->setTitleText(axisXname);
    axisY->setTitleText(axisYname);
    axisY2->setTitleText("time in nanoseconds (Shannon)");
    chart->addAxis(axisX, Qt::AlignBottom);
    chart->addAxis(axisY, Qt::AlignLeft);
```

```
    chart->addAxis(axisY2, Qt::AlignRight);
    series->attachAxis(axisX);
    series->attachAxis(axisY);
    series2->attachAxis(axisX);
    series2->attachAxis(axisY2);
    return chart;
}

void MainWindow::on_action_3_triggered()
{
    plotResult(QInputDialog::getInt(this, "Ввод", "Введите количество
символов", 2, 2), generateBestCase);
}

void MainWindow::on_action_4_triggered()
{
    QString text = QInputDialog::getText(this, "Ввод", "Введите
строчку");
    plotResult(text.size(), nullptr,text);
}

void MainWindow::on_action_5_triggered()
{
    //Creating charts
    QtCharts::QScatterSeries *seriesHuffmanEncode = new
QtCharts::QScatterSeries();
    QtCharts::QScatterSeries *seriesHuffmanDecode = new
QtCharts::QScatterSeries();
    QtCharts::QScatterSeries *seriesShannonEncode = new
QtCharts::QScatterSeries();
    QtCharts::QScatterSeries *seriesShannonDecode = new
QtCharts::QScatterSeries();

    for(int i = 0;i<vectHuffEncode.size();i++){
        seriesHuffmanEncode-
>append(vectHuffEncode[i].first,vectHuffEncode[i].second);
        seriesHuffmanDecode-
>append(vectHuffDecode[i].first,vectHuffDecode[i].second);
        seriesShannonEncode-
>append(vectShannonEncode[i].first,vectShannonEncode[i].second);
        seriesShannonDecode-
>append(vectShannonDecode[i].first,vectShannonDecode[i].second);
    }

    QtCharts::QChart *chartHSEncode = getChartUnite(seriesHuffmanEncode,
seriesShannonEncode,
                                                    "Complexity Huffman
and Shannon encoding",
                                                    "number of characters
in text",
                                                    "time in
nanoseconds");
    QtCharts::QChart *chartHSDecode = getChartUnite(seriesHuffmanDecode,
seriesShannonDecode,
                                                    "Complexity Huffman
and Shannon decoding",
                                                    "number of characters
in text",
```

```
                                                        "time in
nanoseconds");

    QtCharts::QChartView *chartViewHSE = new
QtCharts::QChartView(chartHSEncode);
    QtCharts::QChartView *chartViewHSD = new
QtCharts::QChartView(chartHSDecode);
    QGridLayout *Layout1 = new QGridLayout();
    Layout1->addWidget(chartViewHSE,0,0);
    Layout1->addWidget(chartViewHSD,0,1);

   // ui->scrollAreaWidgetContents->setLayout(Layout1);
    QWidget *windowLayot = new QWidget();
    windowLayot->setLayout(Layout1);
    setCentralWidget(windowLayot);
    show();
}

void MainWindow::on_action_6_triggered()
{
    QtCharts::QScatterSeries *seriesHuffmanEncode = new
QtCharts::QScatterSeries();
    QtCharts::QScatterSeries *seriesHuffmanDecode = new
QtCharts::QScatterSeries();
    QtCharts::QScatterSeries *seriesShannonEncode = new
QtCharts::QScatterSeries();
    QtCharts::QScatterSeries *seriesShannonDecode = new
QtCharts::QScatterSeries();

    QtCharts::QLineSeries *HEcomplexity = new QtCharts::QLineSeries();
    QtCharts::QLineSeries *HDcomplexity = new QtCharts::QLineSeries();
    QtCharts::QLineSeries *SEcomplexity = new QtCharts::QLineSeries();
    QtCharts::QLineSeries *SDcomplexity = new QtCharts::QLineSeries();

    for(int i = 0;i<vectHuffEncode.size();i++){
        seriesHuffmanEncode-
>append(vectHuffEncode[i].first,vectHuffEncode[i].second);
        seriesHuffmanDecode-
>append(vectHuffDecode[i].first,vectHuffDecode[i].second);
        seriesShannonEncode-
>append(vectShannonEncode[i].first,vectShannonEncode[i].second);
        seriesShannonDecode-
>append(vectShannonDecode[i].first,vectShannonDecode[i].second);

        HEcomplexity-
>append(vectHuffEncode[i].first,log(vectHuffEncode[i].first));
        HDcomplexity-
>append(vectHuffDecode[i].first,log(vectHuffDecode[i].first));
        SEcomplexity->append(vectShannonEncode[i].first,
numberOfCharacters[i]*vectShannonEncode[i].first);
        SDcomplexity-
>append(vectShannonDecode[i].first,log(vectShannonDecode[i].first));
    }

    QtCharts::QChart *chartHE =
getChartWithComlexity(seriesHuffmanEncode,HEcomplexity,"Huffman
encoding", "O(logn)",Qt::red);
```

```
    QtCharts::QChart *chartHD =
getChartWithComlexity(seriesHuffmanDecode,HDcomplexity,"Huffman
decoding", "O(logn)",Qt::red);
    QtCharts::QChart *chartSE =
getChartWithComlexity(seriesShannonEncode,SEcomplexity,"Shannon
encoding", "O(n*W)",Qt::blue);
    QtCharts::QChart *chartSD =
getChartWithComlexity(seriesShannonDecode,SDcomplexity,"Shannon
decoding", "O(logn)",Qt::blue);

    QtCharts::QChartView *chartViewHE = new
QtCharts::QChartView(chartHE);
    QtCharts::QChartView *chartViewHD = new
QtCharts::QChartView(chartHD);
    QtCharts::QChartView *chartViewSE = new
QtCharts::QChartView(chartSE);
    QtCharts::QChartView *chartViewSD = new
QtCharts::QChartView(chartSD);


    QGridLayout *Layout1 = new QGridLayout();
    Layout1->addWidget(chartViewHE,0,0);
    Layout1->addWidget(chartViewHD,0,1);
    Layout1->addWidget(chartViewSE,1,0);
    Layout1->addWidget(chartViewSD,1,1);

   // ui->scrollAreaWidgetContents->setLayout(Layout1);
    QWidget *windowLayot = new QWidget();
    windowLayot->setLayout(Layout1);
    setCentralWidget(windowLayot);
    show();
}



QtCharts::QChart *
MainWindow::getChartWithComlexity(QtCharts::QScatterSeries *series,
QtCharts::QLineSeries *series2,  QString graphName, QString seriesName,
QColor color){
    QtCharts::QChart *chart = new QtCharts::QChart();
    QPen pen = series2->pen();
    pen.setWidth(3);
    pen.setBrush(QBrush(Qt::green));
    series2->setPen(pen);
    series2->setName(seriesName);

    series->setColor(color);
    series->setName(graphName);
    series->setMarkerSize(5);

    chart->addSeries(series);
    chart->addSeries(series2);
    chart->setTitle(graphName);

    QtCharts::QValueAxis *axisX = new QtCharts::QValueAxis();
    QtCharts::QValueAxis *axisY = new QtCharts::QValueAxis();
    QtCharts::QValueAxis *axisY1 = new QtCharts::QValueAxis();
```

```
    axisX->setTitleText("number of characters in text");
    axisY->setTitleText("time in nanoseconds");
    axisY1->setTitleText("time of comlexity graph");

    chart->addAxis(axisX, Qt::AlignBottom);
    chart->addAxis(axisY, Qt::AlignLeft);
    chart->addAxis(axisY1, Qt::AlignRight);

    series->attachAxis(axisX);
    series->attachAxis(axisY);

    series2->attachAxis(axisX);
    series2->attachAxis(axisY1);
    return chart;
};
```

Название файла:shannoncode.cpp

```
#include "shannoncode.h"
#include <QDebug>

void ShannonCode::encodeDecode(std::string text, QtCharts::QScatterSeries
*series1, QtCharts::QScatterSeries *series2, std::vector<std::pair
<int,double> > &vect,
                               std::vector< std::pair <int,double> > &vect2){
    if (text == EMPTY_STRING) {
        return;
    }

    std::map<char,int> freq;

    for (char ch: text) {
        freq[ch]++;
    }

    auto start = std::chrono::steady_clock::now();
    buildFunc(freq,EMPTY_STRING, text.size());
    auto end = std::chrono::steady_clock::now();
    auto diff = end - start;
    series1->append(text.size(),std::chrono::duration <double, std::nano>
(diff).count());
    vect.push_back(std::make_pair(text.size(),std::chrono::duration
<double, std::nano> (diff).count()));

    srand(time(NULL));
    std::string stringNew;
    for(int i = 0;i<int(shannon_code1.size());i++){
        auto it = shannon_code1.begin();
        std::advance(it, rand()% shannon_code1.size());
        stringNew+=shannon_code1[it->first];
    }

    auto start2 = std::chrono::steady_clock::now();
    decode(stringNew);
    auto end2 = std::chrono::steady_clock::now();
    auto diff2 = end2 - start2;
```

```
    series2->append(text.size(),std::chrono::duration <double, std::nano>
(diff2).count());
    vect2.push_back(std::make_pair(text.size(),std::chrono::duration
<double, std::nano> (diff2).count()));

    int bitsCount = 0;
    for (auto it = shannon_code1.begin(); it != shannon_code1.end(); +
+it){
        bitsCount+=it->first.size();
    }

    qDebug()<<"number of bits for Shanon encodeDecode: "<<bitsCount<<'\
n';
}

void ShannonCode::buildFunc(const std::map<char, int> map1, std::string
code, int W2) {
    if(map1.size() == 2){
        if(map1.begin()->second > map1.rbegin()->second){
            shannon_code1[code + '0'] = map1.begin()->first;
            shannon_code1[code + '1'] = map1.rbegin()->first;
        } else{
            shannon_code1[code + '1'] = map1.begin()->first;
            shannon_code1[code + '0'] = map1.rbegin()->first;
        }
        return;
    }

    if(map1.size() == 1) shannon_code1[code + '1'] = map1.rbegin()-
>first;

    int W = W2 / 2;
    int K[map1.size() + 1][W + 1];
    for (int i = 0; i < W + 1; i++) {
        K[0][i] = 0;
    }
    int i = 1;
    for (auto it = map1.begin(); it != map1.end(); ++it, i++) {
        for (int w = 0; w <= W; w++) {
            if (it->second <= w) {
                K[i][w] = std::max(it->second +
                                   K[i - 1][w - it->second], K[i - 1][w]);
            } else {
                K[i][w] = K[i - 1][w];
            }
        }
    }

    int res = K[map1.size()][W];
    int res1 = res;
    int w = W;
    i = map1.size();

    std::map<char, int> map2;
    std::map<char, int> map3 = map1;

    for (auto it = map1.rbegin(); it != map1.rend() && res > 0; ++it,
i--) {
```

```
            if (res == K[i - 1][w]) {
                continue;
            } else {
                res = res - it->second;
                w = w - it->second;
                map2.insert(std::pair<char, int>(it->first, it->second));
                map3.erase(it->first);
            }
        }

        if (map2.size() == 1 && map3.size() != 1) {
            shannon_code1[code + '1'] = map2.begin()->first;
            buildFunc(map3, code + '0', W2 - res1);
            return;
        } else if (map3.size() == 1 && map2.size() != 1) {
            shannon_code1[code + '0'] = map3.begin()->first;
            buildFunc(map2, code + '1', res1);
            return;
        } else if (map2.size() == 1 && map3.size() == 1) {
            shannon_code1[code + '1'] =  map2.begin()->first;
            shannon_code1[code + '0'] =  map3.begin()->first;
            return;
        } else {
            buildFunc(map2, code + '1', res1);
            buildFunc(map3, code + '0', W2 - res1);
        }
    }

    void ShannonCode::decode(std::string text){
        std::string text2;
        std::string text3;
        std::map<std::string, char>::iterator it;
        for(int i = 0;i<int(text.size());i++){
            text2.push_back(text[i]);
            it = shannon_code1.find(text2);
            if(it != shannon_code1.end()){
                text3.push_back(shannon_code1[text2]);
                text2 = "";
            }
        }
    }
```