

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: Динамическое кодирование и декодирование по Хаффману –
сравнительное исследование со “статическим” методом**

Студент гр. 9382

Демин В.В.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Демин В.В.

Группа 9382

Тема работы : Динамическое кодирование и декодирование по Хаффману – сравнительное исследование со “статическим” методом

Исходные данные:

На вход программе подается текст для кодирования или декодирования

Содержание пояснительной записки:

«Содержание», «Введение», «Заключение», «Список использованных источников», «Ход выполнения работы», «Описание алгоритмы», «Описание интерфейса пользователя», «Программный код», «Исследование»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 14.12.2020

Дата защиты реферата: 14.12.2020

Студент

Демин В.В.

Преподаватель

Фирсов М.А.

АННОТАЦИЯ

Была написана программа для анализа методов кодирования по таким данным как количество символов в строке и время за которое проводится кодирование или декодирование. По этим данным программа строит графики эффективности сжатия от количества символов и времени от количества символов.

Код программы приведён в приложении А.

SUMMARY

A program was written to analyze coding methods for such data as the number of characters in a line and the time the encoding or decoding is performed. Based on this data, the program builds graphs of the compression efficiency versus the number of characters and the time versus the number of characters.

Program code is given in attachment A.

СОДЕРЖАНИЕ

Введение	5
1. ХОД ВЫПОЛНЕНИЯ РАБОТЫ	6
1.1. Постановка задачи	6
1.2. Описание алгоритмов	6
1.2.1 Статическое кодирование Хаффмана	6
1.2.2 Статическое декодирование Хаффмана	6
1.2.3 Динамическое кодирование Хаффмана	7
1.2.3 Динамическое кодирование Хаффмана	8
1.3. Описание функций	8
1.3.1 Статическое кодирование Хаффмана	8
1.3.2 Статическое декодирование Хаффмана	9
1.3.3 Динамическое кодирование Хаффмана	9
1.3.4 Динамическое декодирование Хаффмана	9
1.4. Описание интерфейса пользователя	10
2. Исследование	11
2.1. Генератор строк.	11
2.2. Накопление статистики и анализ	11
Заключение	14
Список использованных источников	15
Приложение А	16

ВВЕДЕНИЕ

Цель данной курсовой работы сравнить статический метод Хаффмана с динамическим кодированием и декодированием Хаффмана. Для того чтобы анализировать методы кодирования и декодирования необходимо собрать статистику для исследования. Для этого была написана программа на C++ с использованием библиотеки Qt, которая генерирует данные для исследования и строит по полученным характеристическим значениям(такие как, время алгоритма, длина строки до обработки, длина строки после обработки, длина алфавита).

1. Ход Выполнения работы

1.1. Постановка задачи

Для исследования двух методов кодирования необходимо собрать статистические данные, а именно время выполнения кодирования, алфавит кодируемой строки, размеры строк до кодирования и после в битах. Эти данные нам покажут насколько эффективны методы.

1.2. Описание алгоритмов

1.2.1 Статическое кодирование Хаффмана

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (H-дерево).

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.

1.2.2 Статическое декодирование Хаффмана

Для декодирования кода Хаффмана необходимо знать алфавит кодированных символов, чтобы восстановить исходную строку. Зная

алфавит собирается дерево Хаффмана. И по нему происходит сопоставление строки декодирования и кодов в полученном дереве.

1.2.3 Динамическое кодирование Хаффмана

Лучше начинать моделирование с пустого дерева и добавлять в него символы только по мере их появления в сжимаемом сообщении. Но это приводит к очевидному противоречию: когда символ появляется в сообщении первый раз, он не может быть закодирован, так как его еще нет в дереве кодирования.

Чтобы разрешить это противоречие, введем специальный ESCAPE код, который будет означать, что следующий символ закодирован вне контекста модели сообщения. Например, его можно передать в поток сжатой информации как есть, не кодируя вообще. Метод "ЗакодироватьСимвол" в алгоритме адаптивного кодирования Хаффмана можно записать следующим образом.

```
ЗакодироватьСимвол(Символ)
{
    Если СимволУжеЕстьВТаблице(Символ)
        ВыдатьКодХаффманаДля(Символ);
    Иначе
    {
        ВыдатьКодХаффманаДля(ESC);
        ВыдатьСимвол(Символ);
    }
}
```

Использование специального символа ESC подразумевает определенную инициализацию дерева до начала кодирования и декодирования: в него помещаются 2 специальных символа: ESC и EOF (конец файла), с весом, равным 1.

Поскольку процесс обновления не коснется их веса, то по коду кодирования они будут перемещаться на самые удаленные ветви дерева и иметь самые длинные коды.

1.2.3 Динамическое кодирование Хаффмана

Декодер при восстановлении исходной последовательности работает подобно кодеру, т.е. он также последовательно строит дерево Хаффмана по мере декодирования последовательности, что позволяет корректно определять исходные символы.

1.3. Описание функций

Рассмотрим классы кодирования и декодирования

1.3.1 Статическое кодирование Хаффмана

1. `vector<pair<char, string>> coding(string &str) override;` - основная функция кодирования. `String& str` – строка которую нужно закодировать. Возвращает алфавит символов для дальнейшего декодирования, если нужно.
2. `string get_code() override;` - возвращает закодированную строку.
3. `void formAlphabet(string const &str, vector<pair<char, int>> &alphabet);` - функция которая формирует алфавит символов с частотой в тексте. Пробегает по строке и считает веса символов. `string const &str` – исходная строка для кодирования, `vector<pair<char, int>> &alphabet` - алфавит в который запишется необходимая информация.
4. `node * makeBinaryTree(vector<pair<char, int>> alphabet);` - функция формирования бинарного дерева. `vector<pair<char, int>> alphabet` – алфавит для построения с весами символов. Возвращает корень построенного дерева.
5. `void makeCode(node *tree, char ch, string &code, bool &flag);` - рекурсивная функция, которая обходит дерево с построением кода. `node *tree` – узел с которого необходимо

произвести построение. `char ch` – символ для которого строится код. `string &code` – строка в которую запишется код символа. `bool &flag` – флаг, который покажет сформировался ли код символа.

1.3.2 Статическое декодирование Хаффмана

1. `vector<pair<char, string>> decoding(string &str, vector<pair<char, string>> al)` override – основная функция декодирования. `string &str` – строка, которую необходимо декодировать. `vector<pair<char, string>> al` – алфавит символов с кодом символа.

Возвращает полученный алфавит.

2. `string get_code()` override; - функция которая возвращает декодированную строку

1.3.3 Динамическое кодирование Хаффмана

1. `string get_code()` override; - функция которая возвращает декодированную строку.

2. `vector<pair<char, string>> coding(string &str)` override – основная функция кодирования. `string &str` – строка, которую необходимо кодировать.

3. Возвращает полученный алфавит.

4. `void update(char a)` – функция для обновления дерева

1.3.4 Динамическое декодирование Хаффмана

1. `string get_code()` override; - функция которая возвращает декодированную строку.

2. `vector<pair<char, string>> decoding(string &str, vector<pair<char, string>> al)` override – основная функция декодирования. `string &str` – строка, которую необходимо декодировать. `vector<pair<char, string>> al` – алфавит символов с кодом символа.

Возвращает полученный алфавит.

3. `void update(char a)` – функция для обновления дерева

1.4. Описание интерфейса пользователя

При запуске программы выведется меню для взаимодействия программы. Статическое и динамическое кодирование Хаффмана, Статическое и динамическое декодирование Хаффмана – кнопки для открытия окна, которое позволяет произвести определенные операции со строкой. Кнопка генератор открывает окно для демонстрации генератора строк для анализа. Кнопка статистика открывает окно для демонстрации собранной статистики методов для последующего анализа. В этом окне команда обновить обновляет графики на экране. Можно задать количество генераций строк, чтобы собрать больше данных. И стереть все полученные данные.

2. ИССЛЕДОВАНИЕ

2.1. Генератор строк.

Генератор строк генерирует строку заданного размера, символы встречаются в строке равномерно, что позволяет анализировать методы от среднего случая к худшему. Чтобы добиться лучшего случая необходимо использовать нормальное распределение, при котором частота некоторых символов будет в разы отличаться от остальных. Количество символов ограничено только символами печатными символами таблицы `ascii`, так как если брать все символы, которые можно закодировать 8 битами. И при равномерном распределении будет максимально не эффективное кодирование.

2.2. Накопление статистики и анализ

В программе при каждом кодировании или декодировании строки данные заносятся в базу для дальнейшего анализа. Чтобы получить сразу массивный объем данных для анализа. Мы используем генератор строк, который будет генерировать строки длиной от 1 до заданного нами значения. Для анализа поставленной задачи было сгенерировано 3000 строк, таким образом для каждого метода было собрано 3000 строк данных по 4 характеристики, а именно размер алфавита, размер строки до кодирования, размер строки после кодирования. Собранные данные можно увидеть на рисунке 1.

Исходя из полученных данных можно сделать следующие выводы:

1. Статическое декодирование Хаффмана благодаря тому, что алфавит декодирования задан изначально, позволяет добиться максимальной производительности. От этого вытекает проблема, хранения алфавит декодирования, но при больших файлах, эта проблема устраняется высокой эффективностью.
2. Статическое кодирование Хаффмана быстрее, чем динамическое кодирование в несколько раз. В статическом методе Хаффмана происходит обход строки перед началом кодирования, далее

строится дерево Хаффмана и по нему кодируется строка. В то же время в динамическом кодировании нет обхода строки перед началом кодирования, но при этом дерево Хаффмана строится адаптивно, то есть оно изменяется при прохождении строки, что позволит получить выигрыш в кодировании строки, но на перестройку дерева необходимо затрачивать время.

3. Динамическое кодирование чуть быстрее чем, динамическое декодирование. Это связано с тем, что при декодировании происходят такие же операции, как и при кодировании. **В** частности адаптивное построение дерева Хаффмана.

4. Сравнивая с теоретическими оценками, динамический метод при адаптивном перестроении дерева имеет сложность $O(\log(k))$ для вставки в дерево и $O(\log(k))$ для поиска элемента в дереве. При этом кодирование строки составляет сложность $O(n)$. Таким образом общая сложность будет составлять $O(\log^2(k)*n)$.

Теоретическая оценка для статического метода $O(\log(n)*n)$ при использовании дерева.

Проанализировав график можно прийти к выводу, что наши данные совпадают с теоретическим.

5. Эффективность сжатия Динамического кодирования зависит от частотности появления новых символов. Ведь при появлении нового символа необходимо записать код «нулевого» символа и код самого символа. Таким образом при маленьких строках, но при равномерном распределении частоты символ эффективность метода уменьшается. Статическому методу не нужно записывать коды изначальных символов, записывается сразу код из дерева Хаффмана.

6. При равномерном распределении частотности символов, эффективность сжатия динамического и статического методы примерно равны. Исходя из того, что в строка с большой длиной при

равномерном распределении, не будет преобладающего элемента, который бы позволил дать больший прирост эффективности.

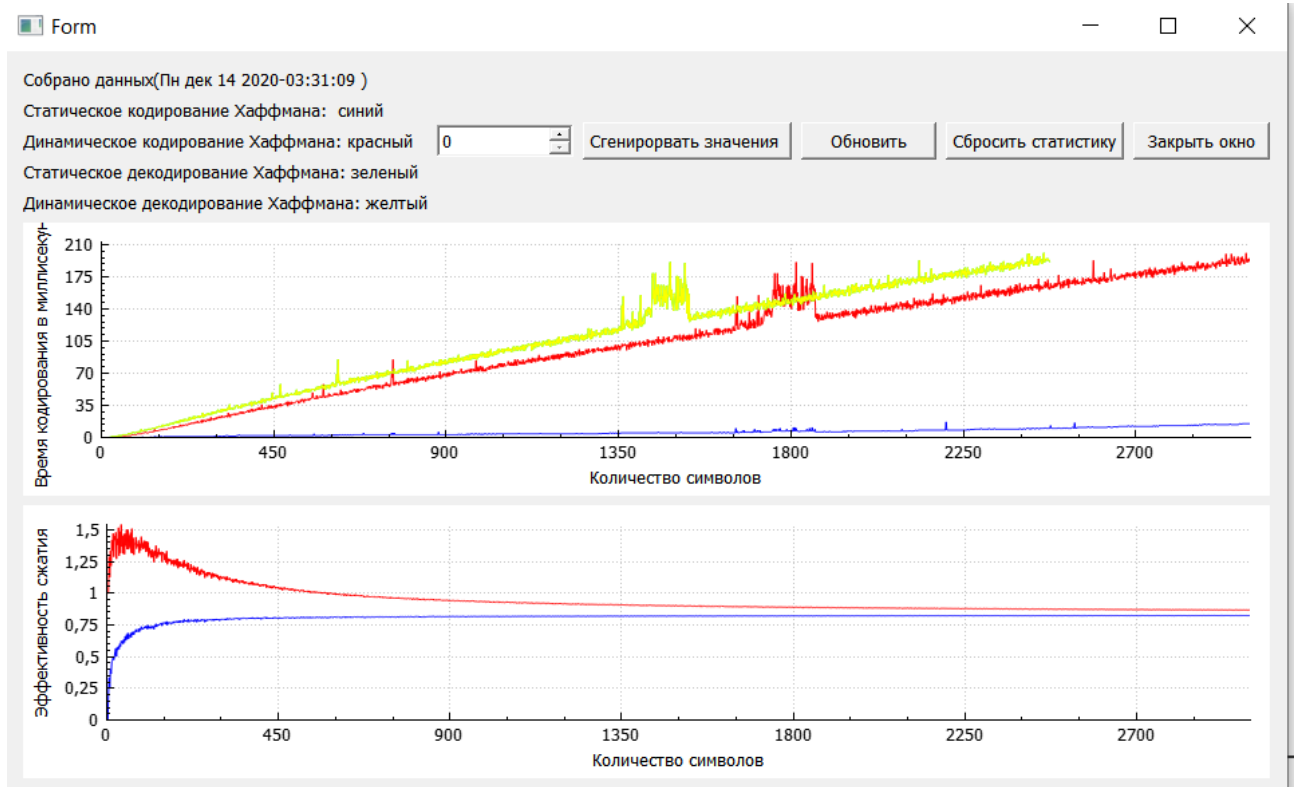


Рис. 1 Окно статистики в приложении.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была создана программа, позволяющая анализировать методы кодирования и декодирования, при генерации строк с равномерным распределением встречаемости символов в строке. Интерпретировав полученную статистику, можем сделать вывод, что динамический метод проигрывает в эффективности применения к строке при равномерном распределении частотности символов. Если необходима скорость сжатия, то лучше использовать статический метод, если необходимо эффективность сжатия, то для этой цели можно применить масштабирование символов для динамического метода, что при не равномерном распределении даст огромную эффективность в сжатии.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Адаптивные коды Хаффмана URL: https://scask.ru/a_lect_cod.php?id=32
2. Код Хаффмана URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0
3. Алгоритм Хаффмана на пальцах URL: <https://habr.com/ru/post/144200/>
4. Huffman Encoding
URL: <https://www.cs.auckland.ac.nz/software/AlgAnim/huffman.html#:~:text=Operation%20of%20the%20Huffman%20algorithm,iterations%2C%20one%20for%20each%20item.>
5. Adaptive Huffman coding – FGK URL: Adaptive Huffman coding – FGK

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

Название файла: coderstatichuffman,h

```
#ifndef CODERSTATICHAFFMAN_H
#define CODERSTATICHAFFMAN_H
#include "ICoder.h"
#include "BinaryTree.h"

class CoderStaticHuffman:public ICoder
{
    string full_code;
    void formAlpabet(string const &str, vector<pair<char, int>>
&alphabet);
    static bool compair_alphabet(pair<char, int> a1, pair<char,
int> a2);
    node * makeBinaryTree(vector<pair<char, int>> alphabet);
    void makeCode(node *tree, char ch, string &code, bool &flag);
    FileLog* file;

public:
    CoderStaticHuffman();
    vector<pair<char, string>> coding(string &str) override;
    string get_code() override;
    void notify(string message) override;

};

#endif // CODERSTATICHAFFMAN_H
```

Название файла: coderdynamic Huffman,h

```
#ifndef CODERDYNAMICHAFFMAN_H
#define CODERDYNAMICHAFFMAN_H
#include "ICoder.h"
#include <BinaryTree.h>

class CoderDynamicHuffman:public ICoder {
```



```

        Node *root;
        string full_code;
        unordered_map<char, Node *> getNode;
        vector<Node *> levelOrder();
        void update(char a);
        void new_node();
        FileLog* file;
public:

        string get_code() override;

        CoderDynamicHuffman();

        vector<pair<char, string>> coding(string& str) override;
        void notify(string message) override;
};

```

```

#endif // CODERDYNAMICHUFFMAN_H

```

Название файла: decoderdynamic Huffman, h

```

#ifndef DECODERDYNAMICHUFFMAN_H
#define DECODERDYNAMICHUFFMAN_H
#include "IDecoder.h"
#include <cmath>
#include <BinaryTree.h>

```

```

class DecoderDynamicHuffman: public IDecoder
{
        string full_code;
        Node *root;
        unordered_map<char, Node *> getNode;
        FileLog* file;
        vector<Node *> levelOrder();

        void update(char a);
        void new_node();

```

```

public:
    DecoderDynamicHuffman();
    vector<pair<char, string>> decoding(string
&str,vector<pair<char, string>> al) override;
    string get_code() override;
    bool need_alphabet() override;
    void notify(string message) override;
};

```

```
#endif // DECODERDYNAMICHUFFMAN_H
```

Название файла: decoderstatichuffman,h

```

#ifndef DECODERSTATICHUFFMAN_H
#define DECODERSTATICHUFFMAN_H
#include "IDecoder.h"
#include <QDebug>

class DecoderStaticHuffman: public IDecoder
{
    string full_code;
    FileLog* file;
public:
    DecoderStaticHuffman();
    vector<pair<char, string>> decoding(string
&str,vector<pair<char, string>> al) override;
    string get_code() override;
    bool need_alphabet() override;
    void notify(string message) override;
};

```

```
#endif // DECODERSTATICHUFFMAN_H
```

Название файла: BinaryTree,h

```

#ifndef BINARYTREE_H
#define BINARYTREE_H
typedef struct nodeTree node;

```

```

struct nodeTree { //узел бинарного дерева
    nodeTree(int w, char c) {

```

```

        count = 1;
        this->weight = w;
        this->ch = c;
        this->left = nullptr;
        this->right = nullptr;

    }

    nodeTree() {
        count = 0;
        weight = 0;
        ch = -1;
        left = nullptr;
        right = nullptr;
    }

    static node *cons(node *a, node *b) {
        node *c = new node();
        c->weight = a->weight + b->weight;
        c->ch = -1;
        if (a->weight > b->weight) { //узел у которого вес
        больше станет правым поддеревом
            c->right = a;
            c->left = b;
        } else {
            c->right = b;
            c->left = a;
        }
        c->count = a->count + b->count;
        return c;
    }

    // для сортировки узлов дерева
    static bool compairNode(node *a1, node *a2) {
        if (a1->weight == a2->weight) {
            if (a1->count == 1 && a2->count == 1) {

```

```

        return a1->ch > a2->ch;
    } else if (a1->count == a2->count && a1->count > 1)
{
    node* a=a1;
    while(a->left!= nullptr){
        a=a->left;
    }
    node* b=a2;
    while(b->left!= nullptr){
        b=b->left;
    }
    return a->ch>b->ch;
}
    }else{
        return a1->count < a2->count;
    }
}
    return a1->weight > a2->weight;
}

```

```

int weight;//вес узла
char ch;//символ в узле
node *left;//левое поддереве
node *right;//правое поддереве
int count;

```

```
};
```

```
struct Node {
```

```

    char c;
    int weight;
    Node *parent;
    Node *left;
    Node *right;

```

```
};
```

```
#endif // BINARYTREE_H
```

Название файла: Coderwindow,h

```

#ifndef CODERWINDOW_H
#define CODERWINDOW_H

#include <QWidget>
#include <QFileDialog>
#include "ICoder.h"
#include "printer.h"

namespace Ui {
class coderWindow;
}

class coderWindow : public QWidget
{
    Q_OBJECT

public:
    explicit coderWindow(QWidget *parent = nullptr, ICoder*
coder=nullptr);
    ~coderWindow();
    string coder_by_text(string& str_before);

signals:
    void mainWind();

private slots:
    void on_pushButton_2_clicked();

    void on_pushButton_3_clicked();

    void on_pushButton_clicked();

private:
    ICoder* coder;
    QFile* file;
    Ui::coderWindow *ui;
    QString path;
};

```

```

#endif // CODERWINDOW_H

Название файла: Decoderwindow,h

#ifndef DECODERWINDOW_H
#define DECODERWINDOW_H

#include <QWidget>
#include <QFile>
#include "IDecoder.h"
#include <QFileDialog>
#include "printer.h"
#include <QDebug>

namespace Ui {
class decoderWindow;
}

class decoderWindow : public QWidget
{
    Q_OBJECT

public:
    explicit decoderWindow(QWidget *parent = nullptr, IDecoder*
decoder=nullptr);
    ~decoderWindow();

signals:
    void mainWind();

private slots:
    void on_pushButton_2_clicked();

    void on_pushButton_3_clicked();

    void on_pushButton_clicked();

private:
    vector<pair<char, string>> make_alphabet(string str);

```

```

        string decoder_by_text(string str,vector<pair<char,string>>& al
);

        IDecoder* decoder;
        QFile* file;
        QString path;
        Ui::decoderWindow *ui;
};

```

```

#endif // DECODERWINDOW_H

```

Название файла: Fille,h

```

#ifndef FILE_H
#define FILE_H
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
class FileLog {
public:
        FileLog(const          char*          filename          )          :
name(filename),m_file_handle(std::fopen(filename, "a+"))
        {
                if( !m_file_handle )
                        throw std::runtime_error("file open failure") ;
        }
        ~FileLog()
        {
                if( std::fclose(m_file_handle) != 0 )
                {
                }
        }

        void write( const char* str )
        {
                if(std::fputs(str,m_file_handle) == EOF )
                        throw std::runtime_error("file write failure") ;
        }
        void clear(){
                std::fclose(m_file_handle);

```

```

        m_file_handle=std::fopen(name,"w+");
        std::fclose(m_file_handle);
        m_file_handle=std::fopen(name,"r+");
    }
    std::string read(){
        char c_str[256];
        fgets(c_str,256,m_file_handle);
        return std::string(c_str);
    }

private:
    const char* name;
    std::FILE* m_file_handle ;
    FileLog(const FileLog & ) ;
    FileLog & operator=(const FileLog & ) ;

};

```

```

#endif // FILE_H

```

Название файла: generator.h

```

#ifndef GENERATOR_H
#define GENERATOR_H
#include <ctime>
#include <string>
#include <algorithm>
#include<iostream>
using namespace std;

class generator {
public:
    int randnum(int a, int b);
    string generatorStr(int complexity);
};

```

```

#endif // GENERATOR_H

```

Название файла: ICoder.h


```

#ifndef ICODER_H
#define ICODER_H
#include <vector>
#include <algorithm>
#include <iostream>
#include<queue>
#include<unordered_map>
#include<algorithm>
#include <File.h>
#include <QElapsedTimer>
using namespace std;

class ICoder{
public:
    virtual vector<pair<char, string>> coding(string &str)=0;
    virtual string get_code()=0;
    virtual void notify(string message) = 0;
};

#endif // ICODER_H

```

Название файла: IDecoder.h

```

#ifndef IDECODER_H
#define IDECODER_H
#include <vector>
#include <algorithm>
#include <iostream>
#include<queue>
#include<unordered_map>
#include<algorithm>
#include <File.h>
#include <QElapsedTimer>

using namespace std;

class IDecoder{
public:
    virtual      vector<pair<char,      string>>      decoding(string
&str,vector<pair<char, string>> al)=0;

```

```

        virtual string get_code()=0;
        virtual bool need_alphabet()=0;
        virtual void notify(string message) = 0;
};

```

```

#endif // IDECODER_H

```

Название файла: mainwindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "coderwindow.h"
#include "coderstatichuffman.h"
#include "coderdynamic Huffman.h"
#include "decoderstatichuffman.h"
#include "decoderwindow.h"
#include "decoderdynamic Huffman.h"
#include "stringgeneratorwindow.h"
#include "statistics.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_pushButton_4_clicked();

    void on_pushButton_2_clicked();

```

```

void on_pushButton_5_clicked();

void on_pushButton_3_clicked();

void on_pushButton_clicked();

void on_pushButton_8_clicked();

void on_pushButton_6_clicked();

private:
    Ui::MainWindow *ui;
    coderWindow* staticC;
    coderWindow* dynamicC;
    decoderWindow* staticD;
    decoderWindow* dynamicD;
    stringGenerator* generatorWindow;
    Statistics* stat;
};
#endif // MAINWINDOW_H

```

Название файла: printer.h

```

#ifndef PRINTER_H
#define PRINTER_H
#include<iostream>
#include <vector>
#include <QString>
using namespace std;
class Printer
{
    static QString binary_char(int a);
public:
    Printer();
    static string write(vector<pair<char,string>> al);
    static QString write_binary(QString str);
};

```

```
#endif // PRINTER_H
```

Название файла: printer.h

```
#ifndef STATISTICS_H
```

```
#define STATISTICS_H
```

```
#include <QWidget>
```

```
#include <QDate>
```

```
#include <QTime>
```

```
#include <iostream>
```

```
#include "tester.h"
```

```
#include <QProgressDialog>
```

```
#include <QDebug>
```

```
using namespace std;
```

```
namespace Ui {
```

```
class Statistics;
```

```
}
```

```
class Statistics : public QWidget
```

```
{
```

```
    Q_OBJECT
```

```
public:
```

```
    explicit Statistics(QWidget *parent = nullptr);
```

```
    ~Statistics();
```

```
signals:
```

```
    void mainWind();
```

```
private slots:
```

```
    void on_pushButton_clicked();
```

```
    void on_pushButton_3_clicked();
```

```
    void on_pushButton_4_clicked();
```

```
    void on_pushButton_2_clicked();
```

```
private:
```

```
    void update();
```

```

        void clearFile();
        Ui::Statistics *ui;
};

#endif // STATISTICS_H
Название файла: stringgeneratorwindow.h
#ifndef STRINGGENERATOR_H
#define STRINGGENERATOR_H

#include <QWidget>
#include <generator.h>

namespace Ui {
class stringGenerator;
}

class stringGenerator : public QWidget
{
    Q_OBJECT

public:
    explicit stringGenerator(QWidget *parent = nullptr);
    ~stringGenerator();
signals:
    void mainWind();

private slots:
    void on_pushButton_7_clicked();

    void on_pushButton_4_clicked();

private:
    Ui::stringGenerator *ui;
    generator* gen;
};

```

```
#endif // STRINGGENERATOR_H
```

Название файла: tester.h

```
#ifndef TESTER_H
#define TESTER_H

#include <decoderdynamic Huffman.h>
#include <decoderstatic Huffman.h>
#include <coderdynamic Huffman.h>
#include <coderstatic Huffman.h>
#include <generator.h>

class Tester
{
public:
    Tester();
    void startTest(int count_of_tests);
};
```

```
#endif // TESTER_H
```

Название файла: coderdynamic Huffman.cpp

```
#include "coderdynamic Huffman.h"

CoderDynamic Huffman::CoderDynamic Huffman() {
    file=new FileLog("coderdh.txt");
}

string CoderDynamic Huffman::get_code() {
    return full_code;
}

vector<pair<char, string>> CoderDynamic Huffman::coding(string& str)
{
    QElapsedTimer clock;
    clock.start();
    vector<pair<char, string>> al;
    char a;
    full_code="";
    new_node();
```

```

for (auto it = str.begin(); it < str.end(); ++it) {
    a=*it;

    string code;
    Node *temp;
    if (getNode.find(a) != getNode.end())//если символ уже есть
        temp = getNode[a];
    else {
        temp = getNode['\0'];//если символа нет
        al.emplace_back(a,"код ascii");
    }

    //составляем код
    while (temp->parent != nullptr) {
        if (temp == temp->parent->left)
            code += '0';
        else code += '1';
        temp = temp->parent;
    }

    reverse(code.begin(), code.end());

    if (getNode.find(a) == getNode.end()) {
        string ascii;
        int c = (int) a;
        for (int i = 0; i < 8; i++) {
            ascii += c % 2 == 0 ? '0' : '1';
            c /= 2;
        }
        reverse(ascii.begin(), ascii.end());
        code.append(ascii);
    }
    update(a);
    full_code += code;
}
string result;

```

```

        result+=
        std::to_string(str.size()*8)+"
"+std::to_string(full_code.size())+"
        "+std::to_string(al.size())+"
"+std::to_string(clock.elapsed())+'\n';
        notify(result);
        return al;
    }

void CoderDynamicHuffman::update(char a) {
    if (getNode.find(a) == getNode.end()) {
        Node *p = new Node();
        p->c = a;
        p->weight = 0;
        p->parent = getNode['\0'];
        getNode[a] = p;
        getNode['\0']->right = p;

        //Creating new NYT node
        Node *n = new Node();
        n->c = '\0';
        n->weight = 0;
        n->parent = getNode['\0'];
        n->left = nullptr;
        n->right = nullptr;

        getNode['\0']->left = n;
        getNode['\0'] = n;
    }

    Node *temp = getNode[a];
    while (temp != nullptr) {
        temp->weight++;
        temp = temp->parent;
    }

    while (true) {
        vector<Node *> levelorder = levelOrder();
        int i;
        for ( i = 1; i < levelorder.size() - 1; i++)

```



```

        if (levelorder[i]->weight > levelorder[i + 1]->weight)
            break;
    if (i == levelorder.size() - 1)
        return;
    int j;
    for (j = levelorder.size() - 1; j >= 0; j--)
        if (levelorder[j]->weight == levelorder[i]->weight - 1)
            break;
    if (j < i)
        return;
    //Swap nodes at position i and j
    Node *parentofi = levelorder[i]->parent;
    Node *parentofj = levelorder[j]->parent;

    levelorder[i]->parent = parentofj;
    levelorder[j]->parent = parentofi;

    if (parentofi->left == levelorder[i] && parentofj->left ==
levelorder[j]) {
        parentofi->left = levelorder[j];
        parentofj->left = levelorder[i];
    } else if (parentofi->left == levelorder[i] && parentofj-
>right == levelorder[j]) {
        parentofi->left = levelorder[j];
        parentofj->right = levelorder[i];
    } else if (parentofi->right == levelorder[i] && parentofj-
>left == levelorder[j]) {
        parentofi->right = levelorder[j];
        parentofj->left = levelorder[i];
    } else if (parentofi->right == levelorder[i] && parentofj-
>right == levelorder[j]) {
        parentofi->right = levelorder[j];
        parentofj->right = levelorder[i];
    }

    levelorder = levelOrder();
    for (int i = 0; i < levelorder.size() - 1; i += 2)

```

```

            if (levelorder[i]->parent->weight != levelorder[i]-
>weight + levelorder[i + 1]->weight)
                levelorder[i]->parent->weight = levelorder[i]-
>weight + levelorder[i + 1]->weight;
        }
    }

```

```

vector<Node *> CoderDynamicHuffman::levelOrder(){
    vector<Node *> ans;
    if (!root) return ans;
    queue<Node *> q;
    q.push(root);
    while (!q.empty()) {
        int levelsize = q.size();
        for (int i = 0; i < levelsize; i++) {
            if (q.front()->right)
                q.push(q.front()->right);
            if (q.front()->left)
                q.push(q.front()->left);
            ans.push_back(q.front());
            q.pop();
        }
    }
    reverse(ans.begin(), ans.end());
    return ans;
}

```

```

void CoderDynamicHuffman::new_node(){

    getNode.clear();
    root = new Node();
    root->c = '\\0';
    root->weight = 0;
    root->parent = nullptr;
    root->left = nullptr;
    root->right = nullptr;
}

```

```

        getNode['\0'] = root;
    }
void CoderDynamicHuffman::notify(string message){
    file=new FileLog("coderdh.txt");
    file->write(message.c_str());
    delete file;
}

```

Название файла: coderstatichuffman.cpp

```

#include "coderstatichuffman.h"

CoderStaticHuffman::CoderStaticHuffman()
{
}

vector<pair<char, string>> CoderStaticHuffman::coding(string &str)
{
    QElapsedTimer clock;
    clock.start();

    vector<pair<char, int>> alphabet;
    vector<pair<char, string>> alphabetWithCode;
    formAlphabet(str, alphabet);

    sort(alphabet.begin(), alphabet.end(), compair_alphabet);
    node *binaryTree = makeBinaryTree(alphabet);

    string code;
    bool flag = true;

    if(alphabet.size()>1){
        for (auto it = begin(alphabet); it != end(alphabet);
++it) {

            makeCode(binaryTree, it->first, code, flag);
            alphabetWithCode.emplace_back(make_pair(it->first,
code));

            code = "";
            flag = true;

```

```

        }}else if(alphabet.size()==1){
            code="0";

alphabetWithCode.emplace_back(make_pair(alphabet[0].first, code));
        }
        full_code = "";
        for (auto ch = str.begin(); ch != str.end(); ++ch) {
            for (auto it = alphabetWithCode.begin(); it !=
alphabetWithCode.end(); ++it) {
                if (it->first == *ch.base()) {
                    full_code=full_code+it->second;
                }
            }
        }
        string result;
        result+=
                                std::to_string(str.size()*8)+"
"+std::to_string(full_code.size())+"
"+std::to_string(alphabetWithCode.size())+"
"+std::to_string(clock.elapsed())+'\n';
        notify(result);
        return alphabetWithCode;
    }

    void    CoderStaticHaffman::formAlpabet(string    const    &str,
vector<pair<char, int>> &alphabet) {

        bool flag = false;
        for (size_t i = 0; i < str.size(); ++i) {
            flag = false;
            for (auto it = begin(alphabet); it != end(alphabet); ++it)
{
                if (str[i] == it->first) {
                    flag = true;
                    it->second++;
                    break;
                }
            }
            if (flag) {

```

```

        continue;
    }
    alphabet.emplace_back(make_pair(str[i], 1));
}
};

bool    CoderStaticHaffman::compair_alphabet(pair<char,    int>    a1,
pair<char, int> a2) {
    return a1.second > a2.second;
}

node *  CoderStaticHaffman::makeBinaryTree(vector<pair<char,    int>>
alphabet) {
    vector<node *> tree;
    for (auto it = alphabet.begin(); it != alphabet.end(); ++it) {
        tree.push_back(new node(it->second, it->first));
    }
    sort(tree.begin(), tree.end(), node::compairNode);
    while (tree.size() != 1) {
        tree.push_back(node::cons(tree[tree.size()    -    2],
tree[tree.size() - 1]));
        sort(tree.begin(), tree.end(), node::compairNode);
        tree.pop_back();
        tree.pop_back();
    }

    return tree[0];
}

void    CoderStaticHaffman::makeCode(node    *tree,    char    ch,    string
&code, bool &flag) {
    if (flag) {
        if (tree != nullptr) {
            if (tree->ch == ch) {
                flag = false;
                return;
            }
            code += "0";

```

```

        makeCode(tree->left, ch, code, flag);
        if (flag) {
            code.pop_back();
        } else { return; }
        code += "1";
        makeCode(tree->right, ch, code, flag);
        if (flag) {
            code.pop_back();
        }
    }
}

string CoderStaticHaffman::get_code() {
    return full_code;
}

void CoderStaticHaffman::notify(string message) {

    file=new FileLog("codersh.txt");
    file->write(message.c_str());
    delete file;
}

```

Название файла: coderwindow.cpp

```

#include "coderwindow.h"

#include "ui_coderwindow.h"

coderWindow::coderWindow(QWidget *parent, ICoder* coder) :
    QWidget(parent), coder(coder),
    ui(new Ui::coderWindow)
{
    ui->setUpUi(this);
    file=nullptr;
    ui->radioButton->setChecked(true);
}

coderWindow::~coderWindow()
{
}

```

```

        file->close();
        delete file;
        delete ui;
    }

void coderWindow::on_pushButton_2_clicked()
{
    this->close();
    emit mainWind();
}

void coderWindow::on_pushButton_3_clicked()
{
    this->path = QFileDialog::getOpenFileName(0, "Open Dialog", "",
    "*.txt");
    if(path!=""){
        delete this->file;
        this->file=new QFile(path);

        file->open(QIODevice::ReadOnly);

        if(file->isOpen()){
            ui->nameFile->setText("Файл выбран: "+path);
        }else{
            ui->nameFile->setText("Файл не открылся: "+path);
        }
    }
}

void coderWindow::on_pushButton_clicked()
{
    QString str;
    QString str_binary;
    string str_after;
    string str_before;
    if(ui->radioButton->isChecked() && file){
        str=QString(file->readAll());
    }else if(ui->radioButton_2->isChecked()){

```

```

        str=ui->textEdit->toPlainText();
    }else{
        return;
    }
    str_before=str.toStdString();
    if(str.isEmpty()){
        ui->strBefore->setText("Строка пустая");
        return;
    }
    ui->strBefore->setText(str);

    str_after=coder_by_text(str_before);
    ui->strAfter->setText(QString(str_after.c_str()));

    ui->strBinary->setText(Printer::write_binary(str));
}

string coderWindow::coder_by_text(string& str_before){
    //засечь время и записать его
    vector<pair<char,string>> al;
    al=coder->coding(str_before);
    string str= coder->get_code();
    ui->alphabet->setText(QString(Printer::write(al).c_str()));
    return str;
};

```

Название файла: decoderdynamic Huffman.cpp

```

#include "decoderdynamic Huffman.h"

DecoderDynamicHuffman::DecoderDynamicHuffman()
{

}

string DecoderDynamicHuffman::get_code() {
    return full_code;
}

bool DecoderDynamicHuffman::need_alphabet() {

```



```

        return false;
    }

    vector<pair<char, string>> DecoderDynamicHuffman::decoding(string
&str,vector<pair<char, string>> al){
        //алфавит
        QElapsedTimer clock;
        clock.start();
        char a = 0;
        full_code = "";
        new_node();
        for (int i = 0; i < 8; i++) {
            if (str[i] == '1') {
                a += pow(2, (7 - i));
            }
        }
        full_code += a;
        al.emplace_back(make_pair(a,"код ascii"));
        update(a); //добавили первый символ
        Node *temp = root;

        for (int j = 8; j < str.size(); j++) {
            char a = 0;

            if (str[j] == '0') {
                temp = temp->left;
                if (temp == getNode['\0']) {
                    a = 0;
                    j++;
                    for (int i = 0; i < 8; i++) {
                        if (str[j] == '1') {
                            a += pow(2, (7 - i));
                        }
                    }
                    j++;
                }
                j--;
                full_code += a;
                al.emplace_back(make_pair(a,"код ascii"));
                update(a);
            }
        }
    }
}

```

```

        temp = root;
    }else {
        char ch=temp->c;
        if(ch!='\0'&&getNode.find(ch)!=getNode.end()){
            full_code+=ch;
            update(ch);
            temp = root;
        }
    }
} else {
    temp = temp->right;
    if (temp == getNode['\0']) {
        a = 0;
        j++;
        for (int i = 0; i < 8; i++) {
            if (str[j] == '1') {
                a += pow(2, (7 - i));
            }
            j++;
        }
        j--;
        full_code += a;
        al.emplace_back(make_pair(a,"код ascii"));
        update(a);
        temp = root;
    }else{
        char ch=temp->c;
        if(ch!='\0'&&getNode.find(ch)!=getNode.end()){
            full_code+=ch;
            update(ch);
            temp = root;
        }
    }
}
}
string result;

```

```

        result+=
        std::to_string(str.size())+"
"+std::to_string(full_code.size()*8)+"    "+std::to_string(al.size())+"
"+std::to_string(clock.elapsed())+'\n';
        notify(result);
        return al;
    }

void DecoderDynamicHuffman::update(char a) {
    if (getNode.find(a) == getNode.end()) {
        Node *p = new Node();
        p->c = a;
        p->weight = 0;
        p->parent = getNode['\0'];
        getNode[a] = p;
        getNode['\0']->right = p;

        //Creating new NYT node
        Node *n = new Node();
        n->c = '\0';
        n->weight = 0;
        n->parent = getNode['\0'];
        n->left = nullptr;
        n->right = nullptr;

        getNode['\0']->left = n;
        getNode['\0'] = n;
    }

    Node *temp = getNode[a];
    while (temp != nullptr) {
        temp->weight++;
        temp = temp->parent;
    }

    while (true) {
        vector<Node *> levelorder = levelOrder();
        int i;
    }
}

```

```

for (i = 1; i < levelorder.size() - 1; i++)
    if (levelorder[i]->weight > levelorder[i + 1]->weight)
        break;
if (i == levelorder.size() - 1)
    return;
int j;
for (j = levelorder.size() - 1; j >= 0; j--)
    if (levelorder[j]->weight == levelorder[i]->weight - 1)
        break;
if (j < i)
    return;
//Swap nodes at position i and j
Node *parentofi = levelorder[i]->parent;
Node *parentofj = levelorder[j]->parent;

levelorder[i]->parent = parentofj;
levelorder[j]->parent = parentofi;

if (parentofi->left == levelorder[i] && parentofj->left ==
levelorder[j]) {
    parentofi->left = levelorder[j];
    parentofj->left = levelorder[i];
} else if (parentofi->left == levelorder[i] && parentofj-
>right == levelorder[j]) {
    parentofi->left = levelorder[j];
    parentofj->right = levelorder[i];
} else if (parentofi->right == levelorder[i] && parentofj-
>left == levelorder[j]) {
    parentofi->right = levelorder[j];
    parentofj->left = levelorder[i];
} else if (parentofi->right == levelorder[i] && parentofj-
>right == levelorder[j]) {
    parentofi->right = levelorder[j];
    parentofj->right = levelorder[i];
}

levelorder = levelOrder();

```

```

        for (int i = 0; i < levelorder.size() - 1; i += 2)
            if (levelorder[i]->parent->weight != levelorder[i]-
>weight + levelorder[i + 1]->weight)
                levelorder[i]->parent->weight = levelorder[i]-
>weight + levelorder[i + 1]->weight;
    }
}

```

```

vector<Node *> DecoderDynamicHuffman::levelOrder() {
    vector<Node *> ans;
    if (!root) return ans;
    queue<Node *> q;
    q.push(root);
    while (!q.empty()) {
        int levelsize = q.size();
        for (int i = 0; i < levelsize; i++) {
            if (q.front()->right)
                q.push(q.front()->right);
            if (q.front()->left)
                q.push(q.front()->left);
            ans.push_back(q.front());
            q.pop();
        }
    }
    reverse(ans.begin(), ans.end());
    return ans;
}

```

```

void DecoderDynamicHuffman::new_node() {

    getNode.clear();
    root = new Node();
    root->c = '\0';
    root->weight = 0;
    root->parent = nullptr;
    root->left = nullptr;
    root->right = nullptr;
}

```

```

        getNode['\0'] = root;
    }
    void DecoderDynamicHuffman::notify(string message) {
        file=new FileLog("decoderdh.txt");
        file->write(message.c_str());
        delete file;
    }

```

Название файла: decoderstatichuffman.cpp

```

#include "decoderstatichuffman.h"

DecoderStaticHuffman::DecoderStaticHuffman()
{
}

vector<pair<char, string>> DecoderStaticHuffman::decoding(string
&str,vector<pair<char, string>> al) {
    QElapsedTimer clock;
    clock.start();
    full_code.clear();
    size_t n=0;
    for(auto it = str.begin();it<str.end();it++){
        for (auto i = al.begin(); i <al.end() ; ++i) {
            while(n<i->second.size() && *it==i->second[n]){
                it++;
                if(it==str.end()){
                    break;
                }
                n++;
            }
            if(n==i->second.size()){
                it--;
                n=0;
                full_code+=i->first;
                break;
            }
        }
    }
}

```

```

        }
        string result;
        result+=
std::to_string(str.size())+"
"+std::to_string(full_code.size()*8)+"      "+std::to_string(al.size())+"
"+std::to_string(clock.elapsed())+'\n';
        notify(result);
        return al;
    }

    string DecoderStaticHuffman::get_code() {
        return full_code;
    }

    bool DecoderStaticHuffman::need_alphabet() {
        return true;
    }

    void DecoderStaticHuffman::notify(string message) {
        file=new FileLog("decodersh.txt");
        file->write(message.c_str());
        delete file;
    }

```

Название файла: decoderwindow.cpp

```

#include "decoderwindow.h"
#include "ui_decoderwindow.h"

decoderWindow::decoderWindow(QWidget *parent, IDecoder* decoder) :
    QWidget(parent), decoder(decoder),
    ui(new Ui::decoderWindow)
{
    ui->setupUi(this);
    file=nullptr;
    ui->radioButton->setChecked(true);
    if(!decoder->need_alphabet()){
        ui->label_5->hide();
        ui->get_alphabet->hide();
    }
}

```

```

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

void decoderWindow::on_pushButton_2_clicked()
{
    this->close();
    emit mainWind();
}

void decoderWindow::on_pushButton_3_clicked()
{
    this->path = QFileDialog::getOpenFileName(0, "Open Dialog", "",
    "*.txt");
    if(path!=""){
        delete this->file;
        this->file=new QFile(path);

        file->open(QIODevice::ReadOnly);

        if(file->isOpen()){
            ui->nameFile->setText("Файл выбран: "+path);
        }else{
            ui->nameFile->setText("Файл не открылся: "+path);
        }
    }
}

void decoderWindow::on_pushButton_clicked()
{
    QString str;
    QString str_binary;
    QString str_al;
    string str_after;
    string str_before;
    vector<pair<char,string>> al;
    if(ui->radioButton->isChecked() && file) {

```



```

        str=QString(file->readAll());

    }else if(ui->radioButton_2->isChecked()){
        str=ui->textEdit->toPlainText();
    }else{
        return;
    }
    if(decoder->need_alphabet()){
        str_al=ui->get_alphabet->toPlainText();
        al=make_alphabet(str_al.toStdString());
        if(al.empty()){
            ui->strAfter->setText("Необходимо ввести алфавит для
кодируемого текста");
            return;
        }
        qDebug()<<"Алфавит сформирован";
    }
    str_before=str.toStdString();
    if(str.isEmpty()){
        ui->strBefore->setText("Строка пустая");
        return;
    }

    str_after=decoder_by_text(str_before,al);
    qDebug()<<"Строка раскодирована";
    ui->strBefore->setText(str);
    ui->strAfter->setText(QString(str_after.c_str()));
    ui->alphabet->setText(QString(Printer::write(al).c_str()));
    ui->strBinary-
>setText(Printer::write_binary(QString(str_after.c_str())));
}

vector<pair<char,string>> decoderWindow::make_alphabet(string str){
    vector<pair<char,string>> al;
    char a=' ';
    string code;
    for(auto it = str.begin();it<str.end();it++){
        if(*it=='\n') {

```

```

        continue;
    }
    while(*it!='-'){
        a=*it;
        it++;
    }
    it++;
    while(*it=='0' || *it=='1'){
        code+=*it;
        it++;
    }
    it--;
    if(!code.empty()){
        al.emplace_back(make_pair(a,code));
    }
    code.clear();
}
return al;
};

```

```

string                                decoderWindow::decoder_by_text(string
str,vector<pair<char,string>>& al){
    al=decoder->decoding(str,al);
    string str_after=decoder->get_code();

    return str_after;
};

```

Название файла: generator.cpp

```
#include "generator.h"
```

```

int generator::randnum(int a, int b) {
    return (rand()) % (b - a + 1)+a;
}

```

```

string generator::generatorStr(int complexity) {
    string str;
    for (int i = 0; i < complexity; ++i) {

```

```

        str+=(char) randnum(32,126);
    }
    return str;
}

```

Название файла: main.cpp

```

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    srand(static_cast<unsigned int>(time(0)));
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}

```

Название файла: mainwindow.cpp

```

#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    staticC=new coderWindow(nullptr,new CoderStaticHuffman());
    dynamicC=new coderWindow(nullptr,new CoderDynamicHuffman());
    connect(staticC,&coderWindow::mainWind,this,&MainWindow::show);

    connect(dynamicC,&coderWindow::mainWind,this,&MainWindow::show);
    staticD=new decoderWindow(nullptr,new DecoderStaticHuffman());
    dynamicD=new                                decoderWindow(nullptr,new
DecoderDynamicHuffman());

    connect(staticD,&decoderWindow::mainWind,this,&MainWindow::show);

    connect(dynamicD,&decoderWindow::mainWind,this,&MainWindow::show);
}

```

```

        generatorWindow=new stringGenerator();

connect(generatorWindow,&stringGenerator::mainWind,this,&MainWindow::show
);

        stat = new Statistics();
        connect(stat,&Statistics::mainWind,this,&MainWindow::show);
    }

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

void MainWindow::on_pushButton_4_clicked()
{
    staticC->show();
    this->close();
}

void MainWindow::on_pushButton_2_clicked()
{
    dynamicC->show();
    this->close();
}

void MainWindow::on_pushButton_5_clicked()
{
    this->close();
}

void MainWindow::on_pushButton_3_clicked()
{
    staticD->show();
    this->close();
}

void MainWindow::on_pushButton_clicked()

```

```

{
    dynamicD->show();
    this->close();
}

void MainWindow::on_pushButton_8_clicked()
{
    generatorWindow->show();
    this->close();
}

void MainWindow::on_pushButton_6_clicked()
{
    stat->show();
    this->close();
}

```

Название файла: printer.cpp

```
#include "printer.h"
```

```

Printer::Printer()
{

}

```

```

string Printer::write(vector<pair<char,string>> al){
    string str;
    str=str+"Symbol\t\tCode\n";
    for (auto it = al.begin(); it != al.end(); ++it) {
        if(it->second=="код ascii"){
            str=str+it->first+"\t\t"+binary_char(it-
>first).toStdString()+"\n";
            continue;
        }
        str=str+it->first+"\t\t"+it->second+"\n";
    }
    return str;
}

```

```

};

QString Printer::write_binary(QString str){
    QString binary_code;
    QByteArray arr = str.toLocal8Bit();
    for(auto it =arr.begin();it!=arr.end();it++){
        int a =(unsigned char)*it;
        binary_code += binary_char(a);
    }
    return binary_code;
};

QString Printer::binary_char(int a){
    QString binary_code;
    for (int i = 0; i < 8; ++i) {
        if (a % 2 == 1) {
            binary_code.append("1");
        } else {
            binary_code.append("0");
        }
        a=a/2;
    }
    reverse(binary_code.begin(), binary_code.end());
    return binary_code;
}

```

Название файла: statistics.cpp

```

#include "statistics.h"
#include "ui_statistics.h"

```

```

std::vector<std::string> split_string(const std::string &s) {
    std::vector<std::string> elems;
    string str;
    int i=0;
    for(int j=0;j<4;j++){
        while(s[i]!=' ' && s[i]!='\n'){

```

```

        str+=s[i];
        i++;
    }
    i++;
    elems.push_back(str);
    str="";
}
return elems;
}

Statistics::Statistics(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Statistics)
{
    ui->setupUi(this);
    QString
    str="Собрано
данных (" +QDate::currentDate().toString()+"-
"+QTime::currentTime().toString()+" )";
    ui->data->setText(str);
}

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

void Statistics::on_pushButton_clicked()
{
    this->close();
    emit mainWind();
}

void Statistics::on_pushButton_3_clicked()
{
    QProgressDialog win;
    win.setWindowTitle("Ожидайте!");
    win.show();
}

```

```

        Tester a;
        a.startTest(ui->spinBox->value());
        win.close();
        QString
даннЫХ (" +QDate::currentDate().toString()+"-
"+QTime::currentTime().toString()+" ");
        ui->data->setText(str);
    }
    void Statistics::clearFile(){
        FileLog* dsh=new FileLog("decodersh.txt");
        FileLog* csh=new FileLog("codersh.txt");
        FileLog* ddh=new FileLog("decoderdh.txt");
        FileLog* cdh=new FileLog("coderdh.txt");
        dsh->clear();
        csh->clear();
        ddh->clear();
        cdh->clear();
        delete dsh;
        delete ddh;
        delete csh;
        delete cdh;
    }

    void Statistics::on_pushButton_4_clicked()
    {
        clearFile();
    }
    void Statistics::update(){
        FileLog* csh=new FileLog("codersh.txt");
        string str;
        QVector<double> size_before;
        QVector<double> size_after;
        QVector<double> size_al;
        QVector<double> time;
        str=csh->read();
        while(!str.empty()){

```



```

        str=csh->read();
        if(str.empty()){
            break;
        }
        try{
            vector<string> a = split_string(str);
            if(a.size()==4){
                int f=stoi(a[0])/8;
                int b=stoi(a[1])/8;
                int c=stoi(a[2]);
                int d=stoi(a[3]);
                size_before.push_back(f);
                size_after.push_back(b);
                size_al.push_back(c);
                time.push_back(d);
            }
        }
        catch(...){
            break;
        }
    }
    FileLog* cdh=new FileLog("coderdh.txt");
    str="";
    QVector<double> size_before2;
    QVector<double> size_after2;
    QVector<double> size_al2;
    QVector<double> time2;
    str=cdh->read();
    while(!str.empty()){
        str=cdh->read();
        if(str.empty()){
            break;
        }
        try{
            vector<string> a = split_string(str);
            if(a.size()==4){
                int f=stoi(a[0])/8;
                int b=stoi(a[1])/8;
                int c=stoi(a[2]);

```

```

        int d=stoi(a[3]);
        size_before2.push_back(f);
        size_after2.push_back(b);
        size_al2.push_back(c);
        time2.push_back(d);
    }}
    catch(...){
        break;
    }
}

FileLog* dsh=new FileLog("decodersh.txt");
    str="";
    QVector<double> size_before3;
    QVector<double> size_after3;
    QVector<double> size_al3;
    QVector<double> time3;
    str=dsh->read();
    while(!str.empty()){
        str=dsh->read();
        if(str.empty()){
            break;
        }
        try{
            vector<string> a = split_string(str);
            if(a.size()==4){
                int f=stoi(a[0])/8;
                int b=stoi(a[1]);
                int c=stoi(a[2]);
                int d=stoi(a[3]);
                size_before3.push_back(f);
                size_after3.push_back(b);
                size_al3.push_back(c);
                time3.push_back(d);
            }}
        catch(...){
            break;
        }
    }

```

```

}

FileLog* ddh=new FileLog("decoderdh.txt");
QVector<double> size_before4;
QVector<double> size_after4;
QVector<double> size_al4;
QVector<double> time4;
str=ddh->read();
while(!str.empty()){
    str=ddh->read();
    if(str.empty()){
        break;
    }
    try{
        vector<string> a = split_string(str);
        if(a.size()==4){
            int f=stoi(a[0])/8;
            int b=stoi(a[1]);
            int c=stoi(a[2]);
            int d=stoi(a[3]);
            size_before4.push_back(f);
            size_after4.push_back(b);
            size_al4.push_back(c);
            time4.push_back(d);
        }
    }
    catch(...){
        break;
    }
}

{
ui->widget->clearGraphs();//Если нужно, но очищаем все графики
//Добавляем один график в widget
ui->widget->addGraph();
ui->widget->addGraph();
ui->widget->addGraph();
ui->widget->addGraph();
}

```

//Говорим, что отрисовать нужно график по нашим двум массивам x
и y

```
ui->widget->graph(0)->setData(size_before,time);  
ui->widget->graph(1)->setData(size_before2,time2);  
ui->widget->graph(1)->setPen(QColor(Qt::red));  
ui->widget->graph(2)->setData(size_before3,time2);  
    ui->widget->graph(2)->setPen(QColor(Qt::green));  
ui->widget->graph(3)->setData(size_before3,time2);  
    ui->widget->graph(3)->setPen(QColor(Qt::yellow));
```

```
ui->widget->xAxis->setLabel("Количество символов");  
ui->widget->yAxis->setLabel("Время кодирования в  
миллисекундах");
```

```
double minY =0, maxY = time[0];  
    for (int i=1; i<time.size(); i++)  
    {  
        if (time2[i]>maxY) maxY = time2[i];  
        if (time3[i]>maxY) maxY = time3[i];  
        if (time4[i]>maxY) maxY = time4[i];  
    }  
    double minX = 0, maxX = size_before[0];  
    for (int i=1; i<size_before.size(); i++)  
    {  
        if (size_before[i]>maxX) maxX = size_before[i];  
    }  
    ui->widget->yAxis->setRange(minY, maxY);  
    ui->widget->xAxis->setRange(minX, maxX);  
    ui->widget->replot();  
}  
QVector<double> k1;  
QVector<double> k2;  
for(int i =0;i<size_before.size()&&i<size_before.size();i++){  
    k1.push_back((size_after[i]/size_before[i]));
```

```

        k2.push_back(size_after2[i]/(size_before2[i]));
    }
    ui->widget_2->clearGraphs();//Если нужно, но очищаем все графики
    //Добавляем один график в widget
    ui->widget_2->addGraph();
    ui->widget_2->addGraph();

    //Говорим, что отрисовать нужно график по нашим двум массивам x и y
    ui->widget_2->graph(0)->setData(size_before,k1);
    ui->widget_2->graph(1)->setData(size_before2,k2);
    ui->widget_2->graph(1)->setPen(QColor(Qt::red));

    ui->widget_2->xAxis->setLabel("Количество символов");
    ui->widget_2->yAxis->setLabel("Эффективность сжатия");

    double minY =0, maxY = k1[0];
    for (int i=1; i<k1.size(); i++)
    {
        if (k1[i]>maxY) maxY = k1[i];
        if (k2[i]>maxY) maxY = k2[i];
    }

    double minX = 0, maxX = size_before[0];
    for (int i=1; i<size_before2.size(); i++)
    {
        if (size_before2[i]>maxX) maxX = size_before2[i];
        if (size_before[i]>maxX) maxX = size_before[i];
    }
    ui->widget_2->yAxis->setRange(minY, maxY);
    ui->widget_2->xAxis->setRange(minX, maxX);
    ui->widget_2->replot();
    delete csh;
}

void Statistics::on_pushButton_2_clicked()
{
    update();
}

```

```
}
```

Название файла: stringgeneratorwindow.cpp

```
#include "stringgeneratorwindow.h"
```

```
#include "ui_stringgeneratorwindow.h"
```

```
stringGenerator::stringGenerator(QWidget *parent) :
```

```
    QWidget(parent),
```

```
    ui(new Ui::stringGenerator)
```

```
{
```

```
    ui->setupUi(this);
```

```
    gen=new generator();
```

```
}
```

```
stringGenerator::~~stringGenerator()
```

```
{
```

```
    delete ui;
```

```
}
```

```
void stringGenerator::on_pushButton_7_clicked()
```

```
{
```

```
    this->close();
```

```
    emit mainWind();
```

```
}
```

```
void stringGenerator::on_pushButton_4_clicked()
```

```
{
```

```
    string str= gen->generatorStr(gen->randnum(1,1000));
```

```
    ui->textEdit_8->setText(QString(str.c_str()));
```

```
}
```

Название файла: tester.cpp

```
#include "tester.h"
```

```
Tester::Tester()
```

```
{
```

```
}
```

```
void Tester::startTest(int count_of_tests){
```

```

generator gen;
string str;
CoderDynamicHuffman cdh;
CoderStaticHuffman csh;
DecoderDynamicHuffman ddh;
DecoderStaticHuffman dsh;
vector<pair<char,string>> al;
for(int i = 2; i<count_of_tests;i++){
    str=gen.generatorStr(i);
    cdh.coding(str);
    al=csh.coding(str);
    str=csh.get_code();
    dsh.decoding(str,al);
    str=cdh.get_code();
    ddh.decoding(str,al);
}
}

```