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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

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

Студент гр. 9384 _____ Гурин С.Н.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Гурин С.Н.

Группа 9384

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

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 реферата: 18.02.2021

Дата защиты реферата: 18.02.2021

Студент _____ Гурин С.Н.

Преподаватель _____ Ефремов М.А.

АННОТАЦИЯ

Основная цель данной курсовой работы – исследование Динамического (Адаптивного) и Статического методов Хаффмана. Исследование представляет из себя генерацию данных, работу с ними, а так же визуализацию сложности написанных алгоритмов. Все расчеты будут производится с помощью стандартной библиотеки C++, а для визуализации используется фреймворк Qt.

SUMMARY

The main goal of this course work is to study the Dynamic (Adaptive) and Static Huffman methods. Research is data, work with them, as well as visualization of the complexity of the written algorithms. All calculations will be performed using the C ++ standard library, and the Qt framework is used for visualization.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной курсовой работе проводится исследование методов кодирования Хаффмана. Для данного исследования были произведены необходимые тесты, представляющие из себя средние/худшие/лучшие случаи для алгоритмов. Результаты тестирований выведены на графиках, а так же сравнены с теоретическими оценками.

1. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

1.1. Основные теоретические сведения

Сжа́тие да́нных (англ. data compression) — алгоритмическое преобразование данных, производимое с целью уменьшения занимаемого ими объёма. Применяется для более рационального использования устройств хранения и передачи данных. Синонимы — упаковка данных, компрессия, сжимающее кодирование, кодирование источника. Обратная процедура называется восстановлением данных (распаковкой, декомпрессией).

Сжатие без потерь позволяет полностью восстановить исходное сообщение, так как не уменьшает в нем количество информации, несмотря на уменьшение длины.

Алгоритм Хаффмана — алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью. Был разработан в 1952 году аспирантом Массачусетского технологического института Дэвидом Хаффманом при написании им курсовой работы. В настоящее время используется во многих программах сжатия данных.

Этот метод кодирования состоит из двух основных этапов:

1. Построение оптимального кодового дерева.
2. Построение отображения код-символ на основе построенного дерева.

1.2. Описание работы программы

При выполнении статического алгоритма Хаффмана подается строка символов для кодирования. Далее с помощью метода класса Coder _CountOfLetters() идет подсчет и запись в массив количество различных букв.

Далее происходит определение и создание с помощью метода класса Coder MakePriority() структуры priority, которая включает в себя символы и их приоритеты. Запись приоритетов происходит происходит с помощью удаления всех одинаковых символов в строке для того, чтобы обезопасить структуру от

повтоения символов. Затем происходит сортировка по возрастания полученного массива структур. После выполнения этого метода происходит вывод всех приоритетов

Затем с помощью метода MakeNodes() происходит создание из полученной ранее структуры двухсвязный список узлов вхождения символов. Реализация этого списка происходит с помощью класса List. Этот класс включает в себя методы Add() - добавление элемента в список, Pop() - удаления двух первых элементов списка, и SortNodes() - сортировка элементов списка по возрастанию.

После выполнения этого метода происходит кодирование элементов с помощью метода Huffman(). Кодирование выполняется рекурсивно созданием из первых двух минимальных элементов списка бинарного дерева, их удаления и добавления полученного дерева. Этот метод выполняется до тех пор, пока не останется один элемент списка.

Далее полученный элемент списка записывается в элемент класса Coder. Затем происходит определение массива структур code, который включает в себя бинарные коды символа и сами символы.

После этого с помощью метода WriteCodedValue() происходит ассоциация элементов этого дерева и кодов этих элементов. Код элемента строится таким образом, что при обходе по бинарному дереву записывается путь к данному элементу(left (zero) = 0, right (one) = 1).

Для реализации адаптивного алгоритма Хаффмана была реализована структура CodeTree. А так же функции для работы с ней:

make_leaf - создает лист.

make_node - создает узел.

fill_symbols_map - заполняет листья значениями.

Заполнение дерева реализовано через очередь.

Пользователь вводит строку или генерирует ее случайно. После чего к ней применяется ФГК алгоритм. Он позволяет динамически регулировать

дерево Хаффмана, не имея начальных частот. В ФГК дереве Хаффмана есть особый внешний узел, называемый 0-узел, используемый для идентификации входящих символов. То есть, всякий раз, когда встречается новый символ — его путь в дереве начинается с нулевого узла. Самое важное — то, что нужно усекать и балансировать ФГК дерево Хаффмана при необходимости, и обновлять частоту связанных узлов. Как только частота символа увеличивается, частота всех его родителей должна быть тоже увеличена. Это достигается путём последовательной перестановки узлов, поддеревьев или и тех и других.

Важной особенностью ФГК дерева является принцип братства (или соперничества): каждый узел имеет два потомка (узлы без потомков называются листами) и веса идут в порядке убывания.

Декодирование выполняется путем перехода к левому или правому дереву, в зависимости от символа (‘0’ лево, ‘1’ право), и смещением индекса проверяемого символа строки. Если, идя по строке и дереву, программа приходит в лист (нет левого и правого дерева), то в ответ записывается соответствующий символ. Дерево возвращается в корень, индекс переходит на следующий символ строки. Цикл работает пока не дойдет до конца строки. Получается раскодированное сообщение, которое было изначально.

2. ВИЗУАЛИЗАЦИЯ

Визуализация данных происходит с помощью классов QCharts фреймворка Qt.

При нажатии клавиши Generate пользователю выводится 6 графиков исследования работы алгоритмов кодирования и декодирования при худшем/лучшем/ среднем случаях. Данные графики показывают зависимость времени выполнения алгоритма в наносекундах от количества сгенерированных данных. Так же на графиках было продемонстрировано сопоставление с теоретическими оценками.

Демонстрация данных графиков происходит в следующем разделе.

3. СОПОСТАВЛЕНИЕ С ТЕОРЕТИЧЕСКИМИ ОЦЕНКАМИ

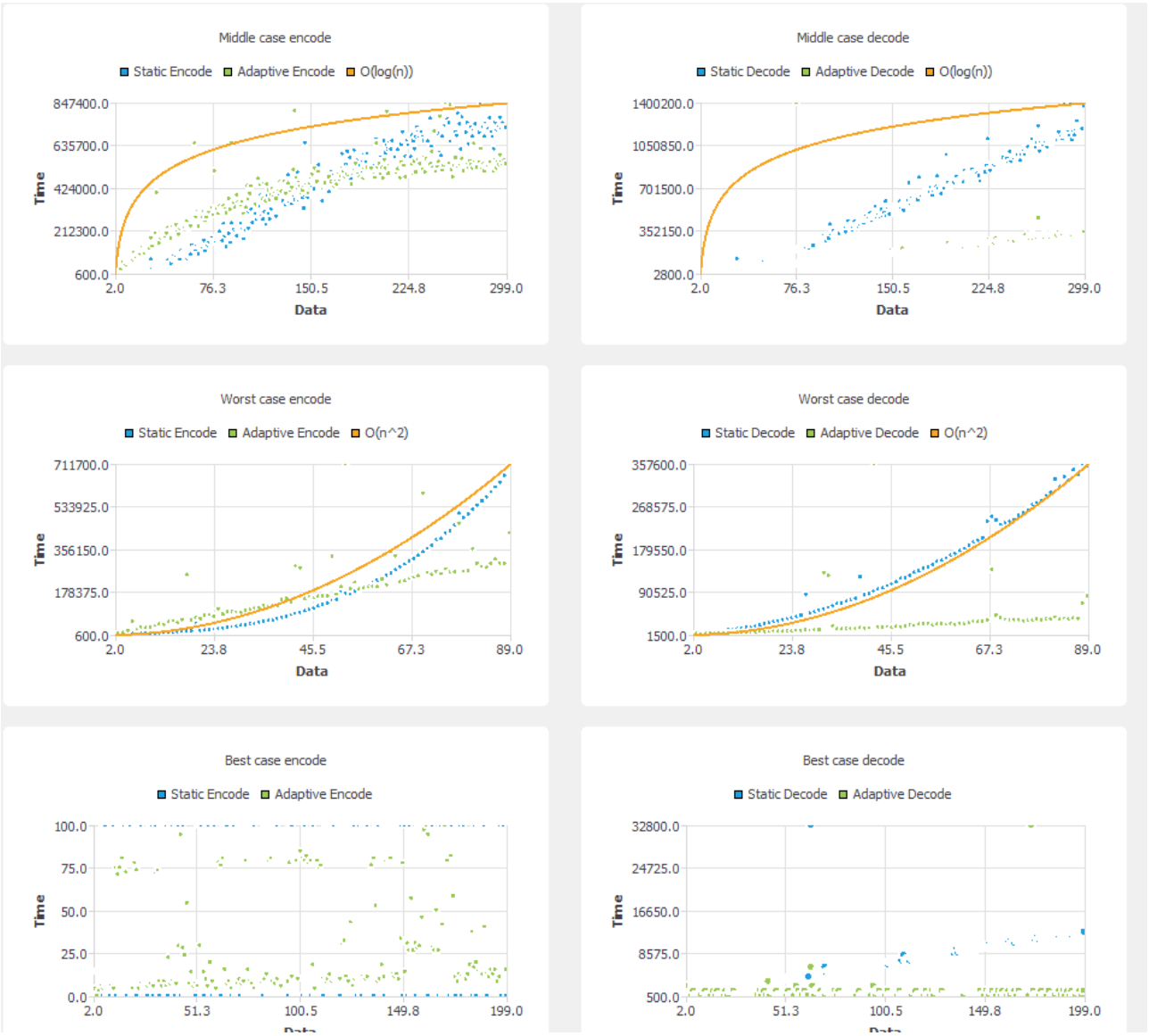

Рис. 1 – графики выполнения работы кодирования/декодирования алгоритмов Хаффмана при различных случаях

Теоретическая сложность алгоритмов Хаффмана при среднем случае оценивается, как O(logn). На первом графике можно в этом убедиться, так как время выполнения работы алгоритмов при более меньших значениях сначала растет, затем выходят на плато. Так же полученные точки походят на график данной функции. При декодировании происходит подобная ситуация. В случае декодирования статического метода можно лицезреть, что точки расположились по прямой, из чего следует, что декодирование статическим методом реализовано не эффективно. Но при рассмотрении худшего случая это алгоритм построен верно.

Для худшего случая оценкой сложности является O(n^2). При увеличении данных время выполнения программы с каждым проходом резко увеличивается, полученные данные, визуализированные на графики очень

легко сопоставимы с теоретическими, так как напоминают параболу. Для декодирования происходит такая же ситуация.

Оценивая лучший случай можно увидеть, что время выполнения алгоритмов выстраиваются в прямую. Оценкой данного случая является O(1). При декодировании можно так же увидеть подобную вещь.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы было проведено исследование алгоритмов Хаффмана при различных данных. Так же была получена объективная оценка реализованных методов исходящая из графиков.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: adaptive_encode.h

```
#ifndef ADAPTIVE_ENCODE_H
#define ADAPTIVE_ENCODE_H
#define MAX_CODE_LEN 1000


struct Symbol {
    char c;
    int weight;
};
bool symbol_less(const Symbol & l, const Symbol & r);
bool symbol_greater(const Symbol& l, const Symbol& r);

struct CodeTree {
    Symbol s;
    CodeTree* parent;
    CodeTree* left;
    CodeTree* right;
};

CodeTree* make_leaf(const Symbol& s);
CodeTree* make_node(int weight, CodeTree* left, CodeTree* right);
bool is_leaf(const CodeTree* node);
bool is_root(const CodeTree* node);
void fill_symbols_map(const CodeTree* node, const CodeTree**
symbols_map);
#endif // ADAPTIVE_ENCODE_H
```

Название файла: adaptive_encode.cpp

```
#include "adaptive_encode.h"
#include "iostream"
#include <climits>
#include <cstring>
#include <string>


bool symbol_less(const Symbol& l, const Symbol& r){
    return l.weight < r.weight;
}


bool symbol_greater(const Symbol& l, const Symbol& r){
    return l.weight > r.weight;
}


CodeTree* make_leaf(const Symbol& s){
    return new CodeTree{ s, nullptr, nullptr, nullptr };
}
```

```
CodeTree* make_node(int weight, CodeTree* left, CodeTree* right){
    Symbol s{ 0, weight };
    return new CodeTree{ s, nullptr, left, right };
}


bool is_leaf(const CodeTree* node){
    return node->left == nullptr && node->right == nullptr;
}


bool is_root(const CodeTree* node){
    return node->parent == nullptr;
}


void fill_symbols_map(const CodeTree* node, const CodeTree**
symbols_map){
    if (is_leaf(node))
        symbols_map[node->s.c - CHAR_MIN] = node;
    else {
        fill_symbols_map(node->left, symbols_map);
        fill_symbols_map(node->right, symbols_map);
    }
}
```

Название файла: priority_queue.h

```
#ifndef PRIORITY_QUEUE_H
#define PRIORITY_QUEUE_H
#include <utility>
#include <iostream>

using namespace std;

template <typename T>
struct PriorityQueueItem {
    int key;
    T data;
};

template <typename T>
struct PriorityQueue {
    int size_;
    int capacity_;
    PriorityQueueItem<T>* heap_;
};

template <typename T>
PriorityQueue<T>* create_pq(int capacity)
{
    PriorityQueue<T>* pq = new PriorityQueue<T>;
    pq->heap_ = new PriorityQueueItem<T>[capacity];
    pq->capacity_ = capacity;
    pq->size_ = 0;
    return pq;
```

```
}
template <typename T>
int sizeQ(PriorityQueue<T>* pq)
{
    return pq->size_;
}
template <typename T>
void sift_up(PriorityQueue<T>* pq, int index)
{
    int parent = (index - 1) / 2;
    while (parent >= 0 && pq->heap_[index].key < pq->heap_[parent].key) {
        std::swap(pq->heap_[index], pq->heap_[parent]);
        index = parent;
        parent = (index - 1) / 2;
    }
}

template <typename T>
bool push(PriorityQueue<T>* pq, int key, const T& data)
{
    if (pq->size_ >= pq->capacity_) return false;
    pq->heap_[pq->size_].key = key;
    pq->heap_[pq->size_].data = data;
    pq->size_++;
    sift_up(pq, pq->size_ - 1);
    return true;
}
template <typename T>
void sift_down(PriorityQueue<T>* pq, int index)
{
    int l = 2 * index + 1;
    int r = 2 * index + 2;
    int min = index;
    if (l < pq->size_ && pq->heap_[l].key < pq->heap_[min].key)
        min = l;
    if (r < pq->size_ && pq->heap_[r].key < pq->heap_[min].key)
        min = r;
    if (min != index) {
        std::swap(pq->heap_[index], pq->heap_[min]);
        sift_down(pq, min);
    }
}

template <typename T>
T pop(PriorityQueue<T>* pq)
{
    std::swap(pq->heap_[0], pq->heap_[pq->size_ - 1]);
    pq->size_--;
    sift_down(pq, 0);
        return pq->heap_[pq->size_].data;
}

template <typename T>
void destroy_pq(PriorityQueue<T>* pq)
{
    delete[] pq->heap_;
    delete pq;
```

```
}
#endif // PRIORITY_QUEUE_H
```

Название файла: static_encode.h

```
#ifndef STATIC_ENCODE_H
#define STATIC_ENCODE_H
#include <stdlib.h>
#include <iostream>
#include <vector>
#include <QChar>
#include <QString>

std::vector<bool> temp_values;

//int count = 0;

template <typename P>
class List {
private:
    void Pop(int &count) {
        Node* tmp = Head->Next;
        delete Head;
        Head = tmp;
        count--;
    }

    void Add(P x, int& count) {
        Node* temp = new Node;
        temp->Prev = 0;
        temp->x = x;
        temp->Next = Head;
        if (Head != 0)
            Head->Prev = temp;
        if (count == 0)
            Head = Tail = temp;
        else
            Head = temp;
        count++;
    }

    void SortNodes() {
        Node* left = Head;
        Node* right = Head->Next;
        Node* temp = new Node;
        while (left->Next) {
            while (right) {
                if ((left->x->GetFreq()) > (right->x->GetFreq())) {
                    temp->x = left->x;
                    left->x = right->x;
                    right->x = temp->x;
                }
                right = right->Next;
            }
            left = left->Next;
            right = left->Next;
        }
```

```
    }

public:
    struct Node {
        P x;
        Node* Next, * Prev;
    };
    Node* Head, * Tail;

    List() {
        this->Head = NULL;
        this->Tail = NULL;
    };

    int GetCount(int& count) {
        return count;
    }

    void _Pop(int& count) {
        Pop(count);
        Pop(count);
    }

    void _Add(P(x), int& count) {
        Add(x, count);
    }

    void _SortNodes() {
        SortNodes();
    }
};

//////////////////////////////////////////////////////////

template <typename T>
class Coder {
public:
    int freq;
    T root;
    Coder* zero;
    Coder* one;

    Coder() {
        this->freq = 0;
        this->root;
        this->zero = nullptr;
        this->one = nullptr;
    }

    Coder(Coder* Left, Coder* Right) {
        this->zero = Left;
        this->one = Right;
        this->freq = Left->GetFreq() + Right->GetFreq();
    }

    struct priority {
        int count = 0;
```

```
        char letter;
    };

    struct CodedValue {
        std::vector<bool> code;
        char letter;
    };

    int GetFreq() {
        return this->freq;
    }

    Coder* GetZero() {
        return this->zero;
    }

    Coder* GetOne() {
        return this->one;
    }

    T GetRoot() {
        return this->root;
    }

    void MakeNodes(priority prior[], int amount, List<Coder*>& ptr, int&
count) {//
        for (int i = 0; i < amount; i++) {
            Coder* p = new Coder;
            p->root = prior[i].letter;
            p->freq = prior[i].count;
            ptr._Add(p, count);
        }
    }

    int CountOfLetters(std::string tmp) {//
        int arr[95] = { 0 };
        CountingPriority(tmp, arr);
        int amount = 0;
        for (int i = 0; i < 95; i++)
        {
            if (arr[i] != 0) {
                amount++;
            }
        }
        return amount;
    }

    void Huffman(List<Coder*>& ptr, int& count) {//сборка дерева
        if (ptr.GetCount(count) != 1) {
            ptr._SortNodes();
            Coder* Left = ptr.Head->x;
            Coder* Right = ptr.Head->Next->x;
            ptr._Pop(count);
            Coder* p = new Coder(Left, Right);
            ptr._Add(p, count);
            Huffman(ptr, count);
        }
```

```
    }

    void MakePriority(priority prior[], std::string tmp, int amount) {//
        for (int i = 0; i < amount; i++) {
            prior[i].letter = tmp[0];
            prior[i].count = Count(prior[i].letter, tmp);
            DeleteLeter(prior[i].letter, tmp);
        }
        Sort(prior, amount);
    }

    void WriteCodedValue(Coder<char>* Tree, CodedValue code[], int&
index_1) {//
        if (Tree->GetZero() != NULL) {
            temp_values.push_back(0);
            WriteCodedValue(Tree->GetZero(), code, index_1);
        }
        if (Tree->GetOne() != NULL) {
            temp_values.push_back(1);
            WriteCodedValue(Tree->GetOne(), code, index_1);
        }
        if (Tree->GetZero() == NULL && Tree->GetOne() == NULL) {
            code[index_1].letter = Tree->GetRoot();
            code[index_1].code = temp_values;
            index_1++;
        }
        if (!temp_values.empty()) {
            temp_values.pop_back();
        }
    }

private:
    void CountingPriority(std::string tmp, int arr[]) {
        for (size_t i = 0; i < tmp.size(); i++)
        {
            arr[(int)tmp[i] - 32]++;
        }
    }


    void DeleteLeter(char letter, std::string& tmp) {
        for (size_t i = 0; i <= tmp.length(); i++) {
            if (tmp[i] == letter) {
                tmp.erase(i, 1);
                i--;
            }
        }
    }

    int Count(char letter, std::string tmp) {
        int count = 0;
        for (size_t i = 0; i <= tmp.length(); i++) {
            if (tmp[i] == letter) {
                count++;
            }
        }
        return count;
```

```
    }

    void Sort(priority prior[], int amount) {
        int temp1;
        char temp2;
        for (int i = 0; i < amount - 1; i++) {
            for (int j = 0; j < amount - i - 1; j++) {
                if (prior[j].count > prior[j + 1].count) {
                    temp1 = prior[j].count;
                    temp2 = prior[j].letter;
                    prior[j].count = prior[j + 1].count;
                    prior[j].letter = prior[j + 1].letter;
                    prior[j + 1].count = temp1;
                    prior[j + 1].letter = temp2;
                }
            }
        }
    }
};
#endif
```

Название файла: mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtCharts>
#include <QScatterSeries>
#include <QMessageBox>
#include <QInputDialog>

#include <string>
#include <vector>
#include <chrono>
#include <cmath>


#include "adaptive_encode.h"
#include "priority_queue.h"


QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

    CodeTree *Tree;
```

```
    const QString possibleCharacters =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz~!@#$%^&*()_+|/.<=>-
?`';:,\"1234567890 ";


    QString generateAverageCase(int amount);
    QString generateWorstCase(int amount);
    QString generateBestCase(int amount);

    CodeTree* Adaptive_MakeTree(const char* message);
    char* Adaptive_decode(const CodeTree* tree, vector<char> code);
    vector<char> Adaptive_encode(const CodeTree* tree, const char*
message);

    QString VectorToString(vector<char> code);

    double AdaptiveEncode_time(QString text);
    double StaticEncode_time(QString text);

    double AdaptiveDecode_time(QString text);
    double StaticDecode_time(QString text);

    QString GetRandomString(int randomStringLength);

    size_t Adaptive_bits(int count_of_letters);
    size_t Static_bits(int count_of_letters);

private slots:

    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Название файла: mainwindow.cpp

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include "static_encode.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

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

QString MainWindow::VectorToString(vector<char> code){
```

```
    QString str;
    for (size_t i = 0; i < code.size(); i++){
        str += code[i];
    }
    return str;
}

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

QString MainWindow::generateWorstCase(int amount){
    QString text;

    for(int i = 0; i<amount;i++){
        text += possibleCharacters[i];
    }

    return text;
}

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

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

    return text;
}

CodeTree* MainWindow::Adaptive_MakeTree(const char* message) {
    Symbol symbols[UCHAR_MAX];
    for (int i = 0; i < UCHAR_MAX; ++i) {
        symbols[i].c = i + CHAR_MIN;
        symbols[i].weight = 0;
    }
    int size = strlen(message);
    for (int i = 0; i < size; ++i)
        symbols[message[i] - CHAR_MIN].weight++;
    std::sort(symbols, symbols + UCHAR_MAX, symbol_greater);
    int len = 0;
    while (symbols[len].weight > 0 && len < UCHAR_MAX) len++;
    PriorityQueue<CodeTree*>* queue = create_pq<CodeTree*>(len);
    for (int i = 0; i < len; ++i){
```

```
        push(queue, symbols[i].weight, make_leaf(symbols[i]));
    }
    while (sizeQ(queue) > 1) {
        CodeTree* ltree = pop(queue);
        CodeTree* rtree = pop(queue);
        int weight = ltree->s.weight + rtree->s.weight;
        CodeTree* node = make_node(weight, ltree, rtree);
        ltree->parent = node;
        rtree->parent = node;
        push(queue, weight, node);

    }
    CodeTree* result = pop(queue);
    destroy_pq(queue);
    return result;
}

char* MainWindow::Adaptive_decode(const CodeTree* tree, vector<char>
code) {
    QString ans;
    char* message = new char[MAX_CODE_LEN];
    int index = 0;
    int len = code.size();
    const CodeTree* v = tree;
    for (int i = 0; i < len; ++i) {
        if (code[i] == '0'){
            v = v->left;
        }
        else{
            v = v->right;
        }
        if (is_leaf(v)) {
            message[index++] = v->s.c;
            v = tree;
        }
    }
    //ui->n31->setText(ans);
    message[index] = '\0';
    return message;
}

vector<char> MainWindow::Adaptive_encode(const CodeTree* tree, const
char* message){
    vector<char> code;
    const CodeTree** symbols_map = new const CodeTree * [UCHAR_MAX];
    for (int i = 0; i < UCHAR_MAX; ++i) {
        symbols_map[i] = nullptr;
    }
    fill_symbols_map(tree, symbols_map);
    int len = strlen(message);
    char path[UCHAR_MAX];
    std::string check;

        for (int i = 0; i < len; ++i) {
            const CodeTree* node = symbols_map[message[i] - CHAR_MIN];
            int j = 0;
            while (!is_root(node)) {
```

```
                if (node->parent->left == node)
                    path[j++] = '0';
                else
                    path[j++] = '1';
                node = node->parent;
            }
            path[j] = '\0';
            while (j > 0){
                code.push_back(path[--j]);
            }
        }
    delete[] symbols_map;
    return code;
}

vector<char> Static_encode(Coder<char>::CodedValue code[], QString text,
int amount){
    vector<char> code_string;
    for (int i = 0; i < text.size(); i++){
        for (int j = 0; j < amount; j++){
            if (text[i] == code[j].letter){
                for (size_t k = 0; k < code[j].code.size(); k++){
                    if (code[j].code[k] == true){
                        code_string.push_back('1');
                    }
                    if (code[j].code[k] == false){
                        code_string.push_back('0');
                    }
                }
             //       code_string += ' ';
            }
        }
    }
    return code_string;
}



QString Static_decode(Coder<char>* tree, vector<char> code){
    QString decode_string;
    int index = 0;
    const Coder<char>* v = tree;
    for (size_t i = 0; i < code.size(); i++){
        if (code[i] == '0'){
            v = v->zero;
        }
        else{
            v = v->one;
        }
        if (v->one == nullptr && v->zero == nullptr) {
            decode_string[index++] = v->root;
            v = tree;
        }
    }
    return decode_string;
}
```

```
QString MainWindow::GetRandomString(int randomStringLength)
{
   //const int randomStringLength = 12; // assuming you want random
strings of 12 characters

   QString randomString;
   for(int i=0; i<randomStringLength; ++i)
   {
       int index = qrand() % possibleCharacters.length();
       QChar nextChar = possibleCharacters.at(index);
       randomString.append(nextChar);
   }
   return randomString;
}

double MainWindow::StaticEncode_time(QString text){
    int count = 0;
    int index_1 = 0;
    Coder<char> Stree;
    List<Coder<char>*> ptr;
    int amount = Stree.CountOfLetters(text.toStdString());
    Coder<char>::priority* prior = new Coder<char>:: priority[amount];
    Stree.MakePriority(prior, text.toStdString(), amount);
    Stree.MakeNodes(prior, amount, ptr, count);

    auto start = std::chrono::steady_clock::now();

    Stree.Huffman(ptr, count);
    auto end = std::chrono::steady_clock::now();
    auto diff = end - start;
    return std::chrono::duration <double, std::nano> (diff).count();
}

double MainWindow::AdaptiveEncode_time(QString text){
    auto start = std::chrono::steady_clock::now();
    Tree = Adaptive_MakeTree(text.toStdString().c_str());
    auto end = std::chrono::steady_clock::now();
    auto diff = end - start;
    return std::chrono::duration <double, std::nano> (diff).count();
}

double MainWindow::AdaptiveDecode_time(QString text){
    Tree = Adaptive_MakeTree(text.toStdString().c_str());
    vector<char> code = Adaptive_encode(Tree,
text.toStdString().c_str());
    auto start = std::chrono::steady_clock::now();
    QString Adaptive_decodee = Adaptive_decode(Tree, code);
    auto end = std::chrono::steady_clock::now();
    auto diff = end - start;
    return std::chrono::duration <double, std::nano> (diff).count();
}


double MainWindow::StaticDecode_time(QString text){
    int count = 0;
    int index_1 = 0;
```

```
    Coder<char> Stree;
    List<Coder<char>*> ptr;
    int amount = Stree.CountOfLetters(text.toStdString());
    Coder<char>::priority* prior = new Coder<char>:: priority[amount];
    Stree.MakePriority(prior, text.toStdString(), amount);
    Stree.MakeNodes(prior, amount, ptr, count);
    Stree.Huffman(ptr, count);
    Coder<char>* result = ptr.Head->x;
    Coder<char>::CodedValue* Scode = new Coder<char>::CodedValue[amount];
    Stree.WriteCodedValue(result, Scode, index_1);
    auto start = std::chrono::steady_clock::now();
    vector<char> static_code = Static_encode(Scode, text, amount);
    auto end = std::chrono::steady_clock::now();
    auto diff = end - start;
    return std::chrono::duration <double, std::nano> (diff).count();
}


void MainWindow::on_pushButton_clicked()
{
    ui->label->setWordWrap(true);
    ui->label_6->setWordWrap(true);
    QString text = ui->lineEdit->text();
    if (!text.isEmpty()){
        Tree = Adaptive_MakeTree(text.toStdString().c_str());
        vector<char> code = Adaptive_encode(Tree,
text.toStdString().c_str());
        QString str_ad = VectorToString(code);
        ui->label->setText(str_ad);
        QString Adaptive_decodee = Adaptive_decode(Tree, code);
        ui->label_6->setText(Adaptive_decodee);

        int count = 0;
        int index_1 = 0;
        Coder<char> Stree;
        List<Coder<char>*> ptr;
        int amount = Stree.CountOfLetters(text.toStdString());
        Coder<char>::priority* prior = new Coder<char>::
priority[amount];
        Stree.MakePriority(prior, text.toStdString(), amount);
        Stree.MakeNodes(prior, amount, ptr, count);
        Stree.Huffman(ptr, count);
        Coder<char>* result = ptr.Head->x;
        Coder<char>::CodedValue* Scode = new
Coder<char>::CodedValue[amount];
        Stree.WriteCodedValue(result, Scode, index_1);

        vector<char> static_code = Static_encode(Scode, text, amount);
        QString str = VectorToString(static_code);
        ui->label_5->setText(str);

        ui->label_9->setText(Static_decode(result, static_code));
    }

}
```

```
void MainWindow::on_pushButton_2_clicked()
{
    QChart *chrt = new QChart();
    QChart *chrt1 = new QChart();
    QChart *chrt2 = new QChart();
    QChart *chrt3 = new QChart();
    QChart *chrt4 = new QChart();
    QChart *chrt5 = new QChart();

    ui->widget->setChart(chrt); // связь графика с элементом отображения
    ui->widget_2->setChart(chrt1);
    ui->widget_3->setChart(chrt2);
    ui->widget_4->setChart(chrt3);
    ui->widget_5->setChart(chrt4);
    ui->widget_6->setChart(chrt5);

    chrt->setTitle("Middle case encode");
    chrt1->setTitle("Middle case decode");
    chrt2->setTitle("Worst case encode");
    chrt3->setTitle("Worst case decode");
    chrt4->setTitle("Best case encode");
    chrt5->setTitle("Best case decode");

    QValueAxis *axisX = new QValueAxis;
    axisX->setTitleText("Data");
    QValueAxis *axisY = new QValueAxis;
    axisY->setTitleText("Time");
    QValueAxis *axisX1 = new QValueAxis;
    axisX1->setTitleText("Data");
    QValueAxis *axisY1 = new QValueAxis;
    axisY1->setTitleText("Time");
    QValueAxis *axisX2 = new QValueAxis;
    axisX2->setTitleText("Data");
    QValueAxis *axisY2 = new QValueAxis;
    axisY2->setTitleText("Time");
    QValueAxis *axisX3 = new QValueAxis;
    axisX3->setTitleText("Data");
    QValueAxis *axisY3 = new QValueAxis;
    axisY3->setTitleText("Time");
    QValueAxis *axisX4 = new QValueAxis;
    axisX4->setTitleText("Data");
    QValueAxis *axisY4 = new QValueAxis;
    axisY4->setTitleText("Time");
    QValueAxis *axisX5 = new QValueAxis;
    axisX5->setTitleText("Data");
    QValueAxis *axisY5 = new QValueAxis;
    axisY5->setTitleText("Time");

    QScatterSeries* series1 = new QScatterSeries(); //for middle encode
    series1->setName("Static Encode");
    series1->setMarkerSize(5);
    QScatterSeries* series2 = new QScatterSeries();
    series2->setName("Adaptive Encode");
    series2->setMarkerSize(5);

    QScatterSeries* series3 = new QScatterSeries(); //for middle decode
    series3->setName("Static Decode");
```

```
series3->setMarkerSize(5);
QScatterSeries* series4 = new QScatterSeries();
series4->setName("Adaptive Decode");
series4->setMarkerSize(5);

QScatterSeries* series5 = new QScatterSeries(); //for worst encode
series5->setName("Static Encode");
series5->setMarkerSize(5);
QScatterSeries* series6 = new QScatterSeries();
series6->setName("Adaptive Encode");
series6->setMarkerSize(5);

QScatterSeries* series7 = new QScatterSeries(); //for worst decode
series7->setName("Static Decode");
series7->setMarkerSize(5);
QScatterSeries* series8 = new QScatterSeries();
series8->setName("Adaptive Decode");
series8->setMarkerSize(5);

QScatterSeries* series9 = new QScatterSeries(); //for best encode
series9->setName("Static Encode");
series9->setMarkerSize(5);
QScatterSeries* series10 = new QScatterSeries();
series10->setName("Adaptive Encode");
series10->setMarkerSize(5);

QScatterSeries* series11 = new QScatterSeries(); //for best decode
series11->setName("Static Decode");
series11->setMarkerSize(7);
QScatterSeries* series12 = new QScatterSeries();
series12->setName("Adaptive Decode");
series12->setMarkerSize(7);


QLineSeries* seriesLog1 = new QLineSeries();
seriesLog1->setName("O(log(n))");
QLineSeries* seriesLog2 = new QLineSeries();
seriesLog2->setName("O(log(n))");
QLineSeries* seriesPowN = new QLineSeries;
seriesPowN->setName("O(n^2)");
QLineSeries* seriesPowN2 = new QLineSeries;
seriesPowN2->setName("O(n^2)");
// построение графиков функций
for(size_t i = 2; i<300; i ++)
{
    QString text = generateAverageCase(i);
    seriesLog1->append(i, log(i));
    seriesLog2->append(i, log(i));
    series1->append(i, StaticEncode_time(text));
    series2->append(i, AdaptiveEncode_time(text));
    series3->append(i, StaticDecode_time(text));
    series4->append(i, AdaptiveDecode_time(text));
}

for(size_t i = 2; i < possibleCharacters.size(); i ++)
{
    QString text = generateWorstCase(i);
```

```
        seriesPowN->append(i, pow(i,2));
        seriesPowN2->append(i, pow(i,2));
        series5->append(i, StaticEncode_time(text));
        series6->append(i, AdaptiveEncode_time(text));
        series7->append(i, StaticDecode_time(text));
        series8->append(i, AdaptiveDecode_time(text));
}

for(size_t i = 2; i<200; i ++)
{
    QString text = generateBestCase(i);
   // seriesLog->append(i, log(i));
        series9->append(i, StaticEncode_time(text));
        series10->append(i, AdaptiveEncode_time(text));
        series11->append(i, StaticDecode_time(text));
        series12->append(i, AdaptiveDecode_time(text));
}


chrt->addSeries(series1);
chrt->addSeries(series2);
chrt->addSeries(seriesLog1);

chrt1->addSeries(series3);
chrt1->addSeries(series4);
chrt1->addSeries(seriesLog2);

chrt2->addSeries(series5);
chrt2->addSeries(series6);
chrt2->addSeries(seriesPowN);

chrt3->addSeries(series7);
chrt3->addSeries(series8);
chrt3->addSeries(seriesPowN2);

chrt4->addSeries(series9);
chrt4->addSeries(series10);

chrt5->addSeries(series11);
chrt5->addSeries(series12);

// устанавливаем оси для каждого графика

chrt->setAxisX(axisX, series1);
chrt->setAxisY(axisY, series1);

chrt1->setAxisX(axisX1, series3);
chrt1->setAxisY(axisY1, series3);

chrt2->setAxisX(axisX2, series5);
chrt2->setAxisY(axisY2, series5);

chrt3->setAxisX(axisX3, series7);
chrt3->setAxisY(axisY3, series7);

chrt4->setAxisX(axisX4, series9);
chrt4->setAxisY(axisY4, series9);
```

```
    chrt5->setAxisX(axisX5, series11);
    chrt5->setAxisY(axisY5, series11);

//////////////////
}
```