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: кодироание и декодирование Хаффмана и Фано-Шеннона

Вариант 1

Студент гр. 8304 _____ Щука А. А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Щука А. А.

Группа 8304

Тема работы: кодирование и декодирование Хаффмана и Фано-Шеннона

Исходные данные:

На вход подается текст, либо буква и частота, случайные данные.

Данные берутся из файла.

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Щука А. А.

Преподаватель _____ Фирсов М.А.

Оглавление

Аннотация

В данной курсовой работе рассмотрен метод кодирования и декодирования Хаффмана и Шеннона-Фано(«Демонстрация»)

Работа выполнена в программной среде: Qt Creator.

Программный код, написанный на языке программирования C++, читается с легкостью и прост в понимании.

В данной программе считывание данных реализовано из файла.

Все результаты выводятся на экран.

Annotation

In this course work considers the encoding and decoding method of Huffman and Shannon-Fano ("Demonstration")

The work was done in the software environment: Qt Creator.

Program code written in the C ++ programming language is readable with ease and is easy to understand.

In this program, data reading is implemented from a file.

All results are displayed on the screen.

Введение

В данной курсовой работе используются два алгоритма для кодирования и декодирования:

1.Алгоритм Шеннона-Фано.

2.Алгоритм Хаффмана.

Алгоритм Шеннона — Фано — один из первых алгоритмов сжатия, который впервые сформулировали американские учёные Шеннон и Роберт Фано. Данный метод сжатия имеет большое сходство с алгоритмом Хаффмана, который является логическим продолжением алгоритма Шеннона. Алгоритм использует коды переменной длины: часто встречающийся символ кодируется кодом меньшей длины, редко встречающийся — кодом большей длины.

Алгоритм Хаффмана - жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью. В отличие от алгоритма Шеннона — Фано, алгоритм Хаффмана остаётся всегда оптимальным и для вторичных алфавитов с более чем двумя символами.

Цель работы

Написать программу с помощью, которой можно будет закодировать или декодировать введенное пользователем сообщение.

Изучить и закрепить знания по кодированию и декодированию алгоритмов Хаффмана и Шеннона-Фано

Продемонстрировать работу алгоритмов с помощью GUI.

Постановка задачи

Статическое кодирование и декодирование текстового файла методами Хаффмана и Фано-Шеннона – демонстрация

"Демонстрация" - визуализация структур данных, алгоритмов, действий. Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий.

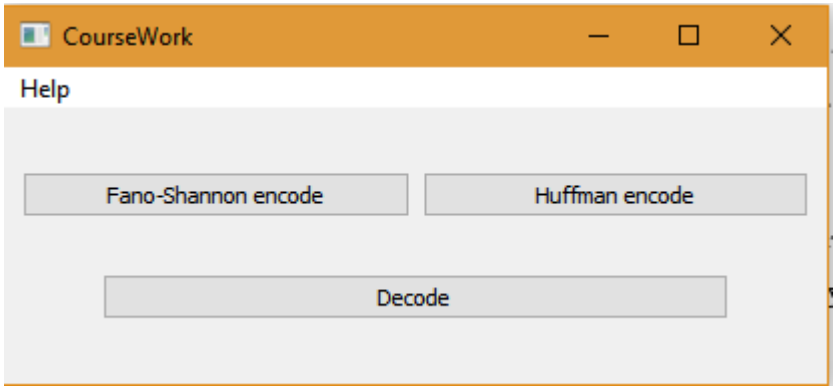
Спецификация программы

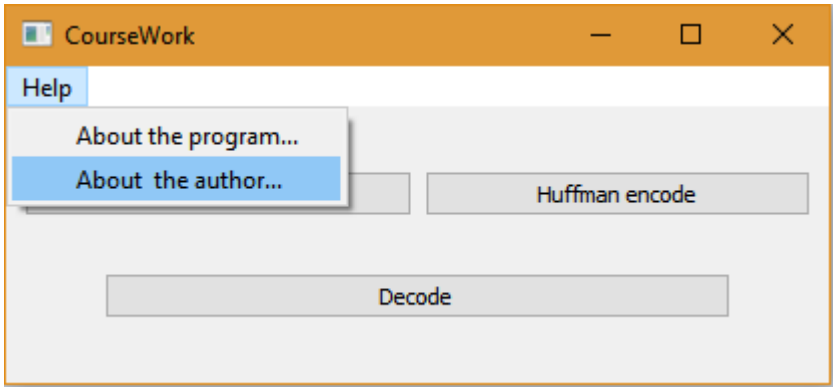
Программа написана на языке C++ с использование фреймворка Qt. Считывание происходит из файла.

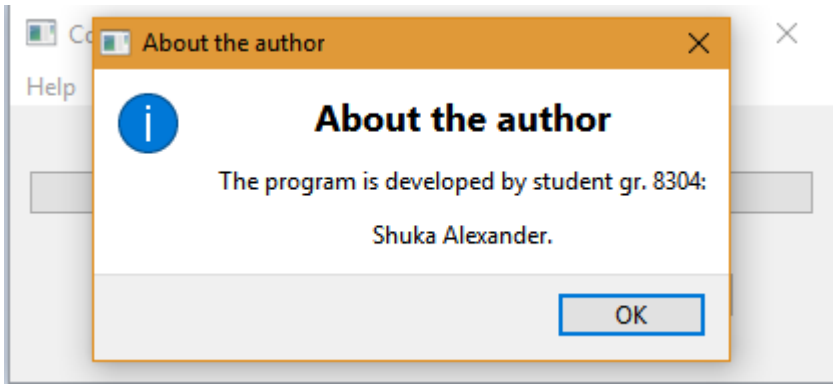
- В файле находятся буквы и их коды и закодированный текст.
- В файле находится текст и программа сама, считает сколько раз данная буква встречается в тексте.

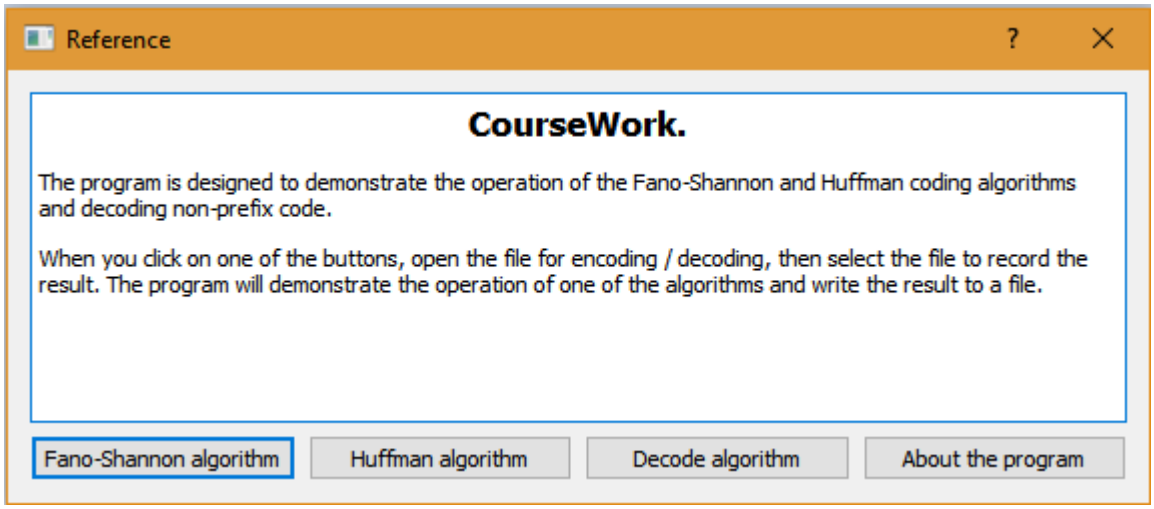
В файле находятся буквы и их коды и закодированный текст: программа считывает коды символов и создает словарь. Далее по алгоритму декодирует сообщение.

Если в файле находится текст, то программа сама считает частоту вхождений символов и дальше работает по алгоритму.









Описание структур данных и функций

Для кодирования и декодирование был разработан абстрактный класс EncodeTree, который реализует методы для работы с деревом, а также содержит несколько виртуальных методов для последующей реализации в классах-наследниках (дерево кодирования Хаффмана и дерево кодирования Фано-Шеннона).

Класс является узлом дерева и содержит указатели на правое и левое поддеревья, указатель на родителя, указатель на данные (структура с полями {символ, вес}).

Для алгоритма Фано-Шеннона был разработан класс FanoShannonTree – наследник EncodeTree. В нем реализованы виртуальные методы базового класса (создание дерева кодирования с помощью алгоритма Фано-Шеннона из исходной строки). Описание алгоритма и методов приведено в исходном коде (*fanoshannontree.h, fanoshannontree.cpp*) и в разделе *Алгоритм кодирования Фано-Шеннона*.

Для алгоритма Хаффмана был разработан класс HuffmanTree – наследник EncodeTree. В нем реализованы виртуальные методы базового класса (создание дерева кодирования с помощью алгоритма Хаффмана из исходной строки).

Описание алгоритма и методов приведено в исходном коде (*huffmamtree.h, huffmantree.cpp*) и в разделе *Алгоритм кодирования Хаффмана*.

Для декодирования были реализованы функции *decode* в пространстве имён *decode*. Подробнее в разделе *Алгоритм декодирования.*

Для демонстрации алгоритмов был разработан класс GraphicWindow, наследник QDialog. Класс содержит QGraphicsView по размеру окна. На QGraphicsView помещается QGraphicsScene.

При демонстрации алгоритмов кодирования сначала на экран выводятся входные данные. Затем рисуется дерево кодирования. В конце выводится закодированный текст.

При демонстрации алгоритма декодирования сначала выводятся коды символов. Затем выводится ззаскодированный текст, в котором подчеркнуты биты, соответствующие кодам символов. В конце выводится раскодированный текст.

Описание интерфейса пользователя

Пользователь может нажать одну из трёх кнопок: кодирование Хаффмана, кодирование Фано-Шеннона, декодирование, либо открыть окно справки.

При нажатии на одну из кнопок открывается диалоговое окно для выбора файла для считывания данных.

После того, как файл выбран, программа применяет выбранный алгоритм к входным данным и визуализирует работу этого алгоритма во всплывающем окне.

Алгоритм кодирования Хаффмана

1. На вход приходят упорядоченные по невозрастанию частот данные.
2. Выбираются две наименьших по частоте буквы алфавита, и создается родитель (сумма двух частот этих «листков»).

3. Потомки удаляются и вместо них записывается родитель, «ветви» родителя нумеруются: левой ветви ставится в соответствие «0», правой «1».

4. Шаг два повторяется до тех пор, пока не будет найден главный родитель — «корень».

Алгоритм кодирования Шэннона-Фано

1. На вход приходят упорядоченные по невозрастанию частот данные.

2. Находится середина, которая делит алфавит примерно на две части. Эти части (суммы частот алфавита) примерно равны. Для левой части присваивается «0», для правой «1», таким образом мы получим листья дерева

3. Шаг 2 повторяется до тех пор, пока мы не получим единственный элемент последовательности, т.е. листок

Алгоритм декодирования

Для декодирования можно воспользоваться построенным деревом либо получить коды символов из файла. Алгоритм декодирования:

1) Создается с помощью дерева, либо считывается из файла словарь вида {код_символа, символ}.

В цикле:

2) Считывается очередной бит, помещается в строку-буффер.

2) Переход к следующему биту.

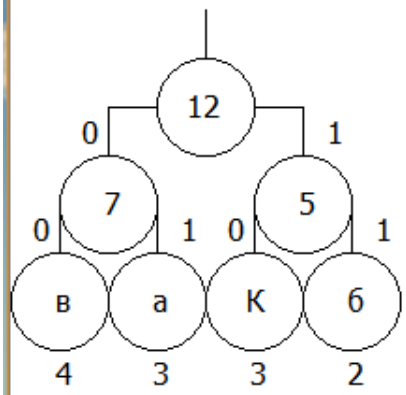
3) Если в строке-буффере содержится сущетсвующий код символа: считывается символ из словаря с данным кодом и записывается в результат декодирования, строка-буффер отчищается.

Пример работы программы

1. Кодирование Фано-Шеннона

Algorithm demonstration

Input text:
ааабббввввККК



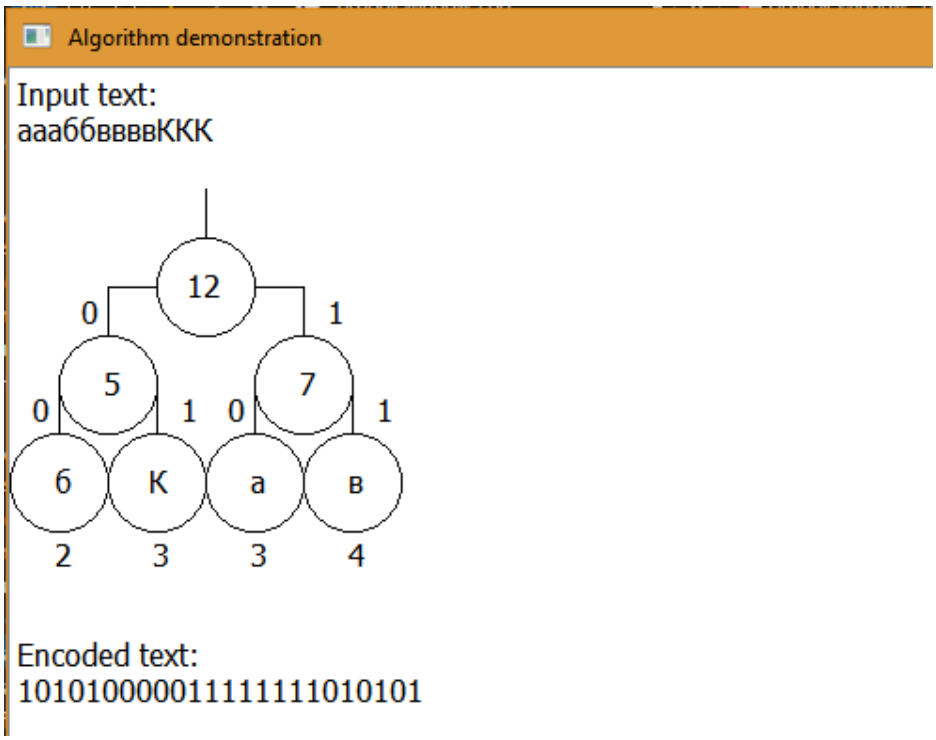
Encoded text:
01010111110000000101010

2. Декодирование

Algorithm demonstration

CharacterCodes:
а -> 10
в -> 11
б -> 00
К -> 01
Encoded text:
10101000001111111010101
Decoded text:
ааабббввввККК

3. Кодирование Хаффмана



Тестирование

N	Входные данные
0	a
1	empty
2	aab
3	ЙцЙцйhguytbynihgyjbhjукцУАК
4	купавкаыфчяЦВ вцаыкмасв\n

```
********* Start testing of TestEncodeFanoShannon *********
Config: Using QtTest library 5.12.5, Qt 5.12.5 (x86_64-little_e
PASS   : TestEncodeFanoShannon::initTestCase()
PASS   : TestEncodeFanoShannon::test(test0)
PASS   : TestEncodeFanoShannon::test(test1)
PASS   : TestEncodeFanoShannon::test(test2)
PASS   : TestEncodeFanoShannon::test(test3)
PASS   : TestEncodeFanoShannon::test(test4)
PASS   : TestEncodeFanoShannon::cleanupTestCase()
Totals: 7 passed, 0 failed, 0 skipped, 0 blacklisted, 2ms
********* Finished testing of TestEncodeFanoShannon *********
```

```
********* Start testing of TestEncodeHuffman *********
Config: Using QtTest library 5.12.5, Qt 5.12.5 (x86_64-lit
PASS   : TestEncodeHuffman::initTestCase()
PASS   : TestEncodeHuffman::test(test0)
PASS   : TestEncodeHuffman::test(test1)
PASS   : TestEncodeHuffman::test(test2)
PASS   : TestEncodeHuffman::test(test3)
PASS   : TestEncodeHuffman::test(test4)
PASS   : TestEncodeHuffman::cleanupTestCase()
Totals: 7 passed, 0 failed, 0 skipped, 0 blacklisted, 2ms
********* Finished testing of TestEncodeHuffman *********
```

```
********* Start testing of TestDecode *********
Config: Using QtTest library 5.12.5, Qt 5.12.5 (x86_64-little
PASS   : TestDecode::initTestCase()
PASS   : TestDecode::test(test0)
PASS   : TestDecode::test(test1)
PASS   : TestDecode::test(test2)
PASS   : TestDecode::test(test3)
PASS   : TestDecode::test(test4)
PASS   : TestDecode::cleanupTestCase()
Totals: 7 passed, 0 failed, 0 skipped, 0 blacklisted, 44ms
********* Finished testing of TestDecode *********
```

Заключение

В ходе выполнения данной курсовой работы была написана программа по кодированию и декодированию методами Хаффмана и Шеннона-Фано.

Также были закреплены знания в этих алгоритмах и в работе в среде Qt Creator (язык C++)

Список используемой литературы

1. ru.wikipedia.org/wiki/Алгоритм_Хаффмана

2. ru.wikipedia.org/wiki/Алгоритм_Шеннона_Фано

3. https://doc.qt.io/

Приложение А. Исходный код программы.

main.cpp

```
#include "./GUI/Headers/mainwindow.h"
#include <QApplication>


int main(int argc, char *argv[]) {
    QApplication a(argc, argv);

    MainWindow window;
    window.show();

    return a.exec();
```

```
}
```

encodetree.h

```
#ifndef ENCODETREE_H
#define ENCODETREE_H

#include <QMap>
#include <QString>
#include <QVector>


struct Symbol
{
    /*
     * Structure for storing a symbol and it's "weight".
    */

    QChar data_;
    size_t weight_;

    Symbol (const Symbol& data) : data_(data.data_),
        weight_(data.weight_) { }

    Symbol (QChar data = 0, size_t weight = 0) :
        data_(data), weight_(weight) { }
};


class EncodeTree
{
    /*
     * Base abstract tree class that implements methods for working with
     * binary coding tree.
    */

public:
    explicit EncodeTree();

    explicit EncodeTree (const Symbol& data);

    virtual ~EncodeTree();

    EncodeTree (const EncodeTree& tree);

    EncodeTree& operator= (const EncodeTree& tree);

    EncodeTree (const EncodeTree&& tree) = delete;

    EncodeTree& operator= (const EncodeTree&& tree) = delete;

    const EncodeTree* getLeft() const;
```

```
    const EncodeTree* getRight() const;

    const EncodeTree* getParent() const;

    const Symbol* getData() const;

    size_t getHeight() const;

    bool isEmptyNode() const;

    bool isLeaf() const;

    void setLeft (EncodeTree* left);

    void setRight (EncodeTree* right);

    void setParent (EncodeTree* parent);

    void setData (const Symbol& data);

    void concatenateTrees (size_t weight, EncodeTree* left, EncodeTree* right);

    QString encodeText (const QString& text);

    QMap<QString, QChar> getCharactersCode() const;

    QString printTree() const;

protected:
    virtual void createEncodeTree (const QString& text) = 0;

    virtual EncodeTree* copyNode() const = 0;

    const EncodeTree* findSymbol (QChar data) const;

    void getCharactersCode (QMap<QString, QChar>& charactersCode,
                            const QString& path) const;

    void printTree (QString& result, size_t depth = 1) const;

private:
    Symbol* data_;
    EncodeTree* parent_;
    EncodeTree* left_;
    EncodeTree* right_;
};


#endif // ENCODETREE_H
```

fanoshannontree.h

```
#ifndef FANOSHANNONTREE_H
#define FANOSHANNONTREE_H
```

```
#include "encodetree.h"


class FanoShannonTree : public EncodeTree
{
    /*
     * The class that implements the Fano-Shannon coding algorithm
     * using a binary coding tree.
    */

public:
    explicit FanoShannonTree() = default;

    explicit FanoShannonTree (const Symbol& data);

    virtual ~FanoShannonTree() = default;

    FanoShannonTree (const FanoShannonTree& tree) = default;

    FanoShannonTree& operator= (const FanoShannonTree& tree) = default;

    FanoShannonTree (const FanoShannonTree&& tree) = delete;

    FanoShannonTree& operator= (const FanoShannonTree&& tree) = delete;

protected:
    virtual void createEncodeTree (const QString& text);

    virtual void createEncodeTree (QMap<QChar, size_t>& charactersAndWeights,
                                   QVector<QChar>& characters,
                                   size_t leftIndex, size_t rightIndex,
                                   size_t sum);

    static size_t getMiddleIndex (QMap<QChar, size_t>& symbolMap,
                                  QVector<QChar>& symbolVector,
                                  size_t leftIndex, size_t sum,
                                  size_t& leftSum, size_t& rightSum);

    virtual FanoShannonTree* copyNode() const;
};

#endif // FANOSHANNONTREE_H
```

huffmantree.h

```
#ifndef HUFFMANTREE_H
#define HUFFMANTREE_H

#include <queue>

#include "encodetree.h"
```

```
class HuffmanTree : public EncodeTree
{
    /*
     * The class that implements the Huffman coding algorithm
     * using a binary coding tree.
    */

public:
    explicit HuffmanTree() = default;

    explicit HuffmanTree (const Symbol& data);

    virtual ~HuffmanTree() = default;

    HuffmanTree (const HuffmanTree& tree) = default;

    HuffmanTree& operator= (const HuffmanTree& tree) = default;

    HuffmanTree (const HuffmanTree&& tree) = delete;

    HuffmanTree& operator= (const HuffmanTree&& tree) = delete;

    class CompareHuffmanTree
    {
    public:
        bool operator() (HuffmanTree* first, HuffmanTree* second);
    };

protected:
    virtual void createEncodeTree (const QString& text);

    virtual void createEncodeTree (QMap<QChar, size_t>& charactersAndWeights);

    virtual HuffmanTree* copyNode() const;
};

#endif // HUFFMANTREE_H
```

decode.h

```
#ifndef DECODE_H
#define DECODE_H

#include <QString>
#include <QMap>
#include <QChar>

#include "encodetree.h"


namespace decode {
QString decode (QMap<QString, QChar>& charactersCode,
                const QString& encodedText);
```

```
QString decode (const EncodeTree* tree, const QString& encodedText);
};

#endif // DECODE_H
```

encodetree.cpp

```
#include "../Headers/encodetree.h"


EncodeTree::EncodeTree()
{
    this->parent_ = nullptr;
    this->left_ = nullptr;
    this->right_ = nullptr;
    this->data_ = nullptr;
}


EncodeTree::EncodeTree (const Symbol& data)
{
    /*
     * Constructor to create a leaf of a tree, accept the symbol
     * with his "weight".
    */

    this->data_ = new Symbol (data);
    this->parent_ = nullptr;
    this->left_ = nullptr;
    this->right_ = nullptr;
}


EncodeTree::~EncodeTree()
{
    if (left_ != nullptr) {
        delete left_;
    }

    if (right_ != nullptr) {
        delete right_;
    }

    if (data_ != nullptr) {
        delete data_;
    }
}


EncodeTree::EncodeTree (const EncodeTree& tree)
{
    this->left_ = tree.left_->copyNode();
    this->right_ = tree.right_->copyNode();
```

```
    left_->setParent(this);
    right_->setParent(this);

    this->data_ = new Symbol (*tree.data_);
}


EncodeTree& EncodeTree::operator= (const EncodeTree& tree)
{
    if (this == &tree) {
        return *this;
    }

    if  (left_ != nullptr) {
        delete left_;
    }

    if (right_ != nullptr) {
        delete right_;
    }

    if (data_ != nullptr) {
        delete data_;
    }

    this->left_ = tree.left_->copyNode();
    this->right_ = tree.right_->copyNode();

    left_->setParent (this);
    right_->setParent (this);

    this->data_ = new Symbol (*tree.data_);

    return *this;
}


const EncodeTree* EncodeTree::getLeft() const
{
    return left_;
}


const EncodeTree* EncodeTree::getRight() const
{
    return right_;
}


const EncodeTree* EncodeTree::getParent() const
{
    return parent_;
```

```
}


const Symbol* EncodeTree::getData() const
{
    return data_;
}


size_t EncodeTree::getHeight() const
{

    /*
     * Method for calculating tree height. Returns the height of the tree.
    */

    if (this->isEmptyNode()) {
        return 0;
    }

    size_t leftHeight = 0;
    if (this->left_ != nullptr) {
        leftHeight = this->left_->getHeight();
    }

    size_t rightHeight = 0;
    if (this->right_ != nullptr) {
        rightHeight = this->right_->getHeight();
    }

    return 1 + std::max (leftHeight, rightHeight);
}


bool EncodeTree::isEmptyNode() const
{
    return (this->data_ == nullptr &&
            this->left_ == nullptr &&
            this->right_ == nullptr);
}


bool EncodeTree::isLeaf() const
{
    return this->left_ == nullptr && this->right_ == nullptr;
}


void EncodeTree::setLeft (EncodeTree* left)
{
    left_ = left;
}
```

```
void EncodeTree::setRight (EncodeTree* right)
{
    right_ = right;
}


void EncodeTree::setParent (EncodeTree* parent)
{
    parent_ = parent;
}


void EncodeTree::setData (const Symbol& data)
{
    if (data_ == nullptr) {
        data_ = new Symbol;
    }

    *data_ = data;
}


void EncodeTree::concatenateTrees (size_t weight, EncodeTree* left,
                                   EncodeTree* right)
{
    /*
     * The method for creating a tree node takes the
     * "weight" of subtrees, left and right subtrees.
    */

    if (data_ == nullptr) {
        data_ = new Symbol;
    }

    this->data_->weight_ = weight;
    this->left_ = left;
    this->right_ = right;
}


QString EncodeTree::encodeText (const QString& text)
{
    /*
     * The method for encoding text.
     * Accepts input text, returns encoded text.
    */

    if (text.size() == 0) {
        return "";
    }

    this->createEncodeTree (text);
```

```
    QString code;
    QString path;

    for (auto elem : text) {
        const EncodeTree* node = this->findSymbol (elem);

        while (node->parent_ != nullptr) {
            if (node->parent_->left_ == node) {
                path = "0" + path;
            }
            else {
                path = "1" + path;
            }

            node = node->parent_;
        }

        code += path;
        path.clear();
    }

    return code;
}


const EncodeTree* EncodeTree::findSymbol (QChar data) const
{
    /*
     * The method for finding a character in a tree takes a pointer
     * to the root and a character, returns a pointer to the leaf
     * that contains that character, or if the character is not in
     * the tree, returns nullptr.
    */

    if (this->isLeaf()) {
        if (this->data_->data_ == data) {
            return this;
        }
    }

    if (this->left_ != nullptr) {
        const EncodeTree* buffer = this->left_->findSymbol (data);
        if (buffer != nullptr) {
            return buffer;
        }
    }

    if (this->right_ != nullptr) {
        const EncodeTree* buffer = this->right_->findSymbol (data);
        if (buffer != nullptr) {
            return buffer;
        }
```

```
    }

    return nullptr;
}


void EncodeTree::printTree (QString& result, size_t depth) const
{
    /*
     * The method for writing a tree to a string. Accepts
     * a link to the result string and current depth.
     * Writes a tree to a string as a list.
    */

    if (this->isLeaf()) {
        for (size_t i = 0; i < depth; ++i){
            result += "-";
        }

        QChar data = this->data_->data_;
        if (data == ' ') {
            result += "space";
        }
        else if (data == '\n') {
            result += "\\n";
        }
        else if (data == '\t') {
            result += "tab";
        }
        else {
            result += data;
        }

        result += "\n";
    }
    else {
        depth += 1;

        if (this->left_ != nullptr) {
            this->left_->printTree (result, depth);
        }

        if (this->right_ != nullptr) {
            this->right_->printTree (result, depth);
        }
    }
}


void EncodeTree::getCharactersCode (QMap<QString, QChar>& charactersCode,
                                    const QString& path) const
{
    /*
```

```
     * The method for obtaining character codes.
     * Accepts a dictionary of character codes and the current path in the
     * tree.
     * 1) If the tree node is empty, the method exits.
     * 2) If the current node is a leaf, writes a symbol with its code into
     * the dictionary.
     * 3) Otherwise, it is called recursively for the left and right subtrees.
    */

    if (this->isEmptyNode()) {
        return;
    }

    if (this->isLeaf()) {
        QChar data = this->data_->data_;
        charactersCode.insert (path, data);
    }
    else {
        if (this->left_ != nullptr) {
            this->left_->getCharactersCode (charactersCode, path + "0");
        }
        if (this->right_ != nullptr) {
            this->right_->getCharactersCode (charactersCode, path + "1");
        }
    }
}


QMap<QString, QChar> EncodeTree::getCharactersCode() const
{
    /*
     * The method for obtaining character codes., returns a
     * result map.
    */

    QMap<QString, QChar> charactersCode;

    if (this->isEmptyNode()) {
        return charactersCode;
    }

    if (this->left_ != nullptr) {
        this->left_->getCharactersCode (charactersCode, "0");
    }
    if (this->right_ != nullptr) {
        this->right_->getCharactersCode (charactersCode, "1");
    }

    return charactersCode;
}


QString EncodeTree::printTree() const
```

```
{
    /*
     * The method for writing a tree to a string
     * in the form of a list.
    */

    QString result;

    printTree (result);

    return result;
}
```

fanoshannontree.cpp

```
#include "../Headers/fanoshannontree.h"


FanoShannonTree::FanoShannonTree (const Symbol& data) : EncodeTree (data) { }


void FanoShannonTree::createEncodeTree (const QString& text)
{
    /*
     * The method for creating a binary coding tree that accepts text.
     * 1) A dictionary of character codes and an array of characters
     * are created. Unique characters are entered into the array.
     * 2) The array is sorted in descending order of the "weight"
     * of the characters.
     * 3) The case when the text contains one character
     * processed separately.
     * 4) If the text has more than 1 character, the method of creating a
     * tree is called, which creates subtrees from subarrays.
    */

    QMap<QChar, size_t> charactersAndWeights;
    QVector<QChar> characters;

    for (auto elem : text) {
        if (charactersAndWeights.count (elem) == 0) {
            charactersAndWeights.insert (elem, 1);
            characters.push_back (elem);
        }
        else {
            charactersAndWeights[elem] += 1;
        }
    }

    std::sort (characters.begin(), characters.end(),
               [&charactersAndWeights] (QChar first, QChar second) {
        return charactersAndWeights[first] > charactersAndWeights[second]; });

    if (characters.size() == 1) {
        QChar data = characters[0];
```

```
        Symbol symbol (data, 1);

        FanoShannonTree* leftTree = new FanoShannonTree (symbol);
        leftTree->setParent (this);
        this->setLeft (leftTree);
        this->setData (Symbol (0, 1));
    }
    else {
        createEncodeTree (charactersAndWeights, characters, 0,
                          static_cast <size_t> (characters.size()),
                          static_cast <size_t> (text.size()));
    }
}


void FanoShannonTree::createEncodeTree (QMap<QChar, size_t>& charactersAndWeights,
                                        QVector<QChar>& characters,
                                        size_t leftIndex, size_t rightIndex,
                                        size_t sum)
{
    /*
     * The method for creating a binary coding tree,
     * takes a dictionary of characters, an array of characters
     * sorted in descending order, the left and right index in
     * the array, and the sum of the "weights" of the characters.
     * 1) If the left index is greater than or equal to the right, nullptr
     * is returned.
     * 2) If there is one element between the left and right indexes,
     * a leaf is created.
     * 3) Otherwise, the array is split into trees of approximately equal
     * weight, and the left and right subtrees are created recursively.
    */

    if (leftIndex >= rightIndex) {
        return;
    }

    if (rightIndex == leftIndex + 1) {
        QChar data = characters[static_cast <long> (leftIndex)];
        Symbol symbol (data, charactersAndWeights[data]);
        this->setData (symbol);
    }
    else {
        size_t leftSum = 0;
        size_t rightSum = 0;
        size_t middle = getMiddleIndex (charactersAndWeights, characters,
                                        leftIndex, sum, leftSum, rightSum);

        FanoShannonTree* leftTree = new FanoShannonTree;
        leftTree->createEncodeTree (charactersAndWeights, characters,
                                    leftIndex, middle + 1, leftSum);

        FanoShannonTree* rightTree = new FanoShannonTree;
```

```
        rightTree->createEncodeTree (charactersAndWeights, characters,
                                     middle + 1, rightIndex, rightSum);

        leftTree->setParent (this);
        rightTree->setParent (this);

        this->concatenateTrees (sum, leftTree, rightTree);
    }
}


size_t FanoShannonTree::getMiddleIndex (QMap<QChar, size_t>& charactersAndWeights,
                                        QVector<QChar>& characters,
                                        size_t leftIndex, size_t sum,
                                        size_t &leftSum, size_t &rightSum)
{
    /*
     * The function of getting the middle of an array of characters takes a
     * dictionary of characters, sorted by non-increasing array, the left
     * index in the array, links to the sum of the weights to the left and
     * right of the middle, returns the middle index.
    */

    size_t middle = leftIndex;

    leftSum = charactersAndWeights[characters[static_cast <long> (middle)]];
    rightSum = sum - leftSum;

    long delta = static_cast <long> (leftSum) - static_cast <long> (rightSum);

    while (delta + static_cast <long> (charactersAndWeights[
                                       characters[
                                       static_cast <long> (middle)+1]])< 0) {
        ++middle;

        QChar data = characters[static_cast <long> (middle)];

        leftSum += charactersAndWeights[data];
        rightSum -= charactersAndWeights[data];

        delta = static_cast <long> (leftSum) - static_cast <long> (rightSum);
    }

    return middle;
}


FanoShannonTree* FanoShannonTree::copyNode() const
{
    FanoShannonTree* node = new FanoShannonTree;

    node->setData(*(new Symbol(*this->getData())));
    if (this->getLeft() == nullptr && this->getRight() == nullptr) {
```

```
        return node;
    }

    FanoShannonTree* left = nullptr;
    if (this->getLeft() != nullptr) {
        EncodeTree* leftTmp = const_cast <EncodeTree *> (this->getLeft());
        left = dynamic_cast <FanoShannonTree *> (leftTmp);
    }

    FanoShannonTree* right = nullptr;
    if (this->getRight() != nullptr) {
        EncodeTree* rightTmp = const_cast<EncodeTree *>(this->getRight());
        left = dynamic_cast <FanoShannonTree *> (rightTmp);
    }

    left->setParent (node);
    right->setParent (node);

    node->setLeft (left);
    node->setRight (right);

    return node;
}
```

huffmantree.cpp

```
#include "../Headers/huffmantree.h"


HuffmanTree::HuffmanTree (const Symbol& data) : EncodeTree (data) { }


void HuffmanTree::createEncodeTree (const QString& text)
{
    /*
     * The method for creating a binary coding tree that accepts text.
     * 1) A dictionary of character codes is creating.
     * 2) The case when the text contains one character
     * processed separately.
     * 3) If the text has more than 1 character, the method of creating a
     * tree is called, which creates subtrees.
    */

    QMap<QChar, size_t> charactersAndWeights;

    for (auto elem : text) {
        if (charactersAndWeights.count (elem) == 0) {
            charactersAndWeights.insert (elem, 1);
        }
        else {
            charactersAndWeights[elem] += 1;
        }
    }
```

```
    if (charactersAndWeights.size() == 1) {
        QChar data = text[0];
        Symbol symbol (data, 1);

        HuffmanTree* leftTree = new HuffmanTree (symbol);
        leftTree->setParent (this);
        this->setLeft (leftTree);
        this->setData (Symbol (0, 1));
    }
    else {
        createEncodeTree (charactersAndWeights);
    }
}


void HuffmanTree::createEncodeTree (QMap<QChar, size_t>& charactersAndWeights)
{
    /*
     * Method for creating a coding tree. Accepts a dictionary
     * of characters and their "weights".
     * 1) A priority queue is created from the dictionary.
     * 2) As long as there are at least 3 elements in the queue,
     * two elements with the smallest "weight" are extracted from the
     * queue and merged into a tree node. The tree node is queued.
     * 3) When 2 items remain in the queue, they are retrieved and
     * merged into a summary tree.
    */

    std::priority_queue<HuffmanTree*, std::vector<HuffmanTree*>,
            HuffmanTree::CompareHuffmanTree> nodes;

    for (auto elem : charactersAndWeights.keys()) {
        size_t weight = charactersAndWeights[elem];
        nodes.push (new HuffmanTree (Symbol (elem, weight)));
    }


    HuffmanTree* leftTree = nullptr;
    HuffmanTree* rightTree = nullptr;

    while (nodes.size() > 2) {
        leftTree = nodes.top();
        nodes.pop();

        rightTree = nodes.top();
        nodes.pop();

        HuffmanTree* node = new HuffmanTree;
        node->concatenateTrees(leftTree->getData()->weight_ +
                               rightTree->getData()->weight_,
                               leftTree, rightTree);

        leftTree->setParent (node);
```

```
        rightTree->setParent (node);

        nodes.push (node);
    }

    leftTree = nodes.top();
    nodes.pop();

    rightTree = nodes.top();
    nodes.pop();

    this->concatenateTrees (leftTree->getData()->weight_ +
                            rightTree->getData()->weight_,
                            leftTree, rightTree);

    leftTree->setParent (this);
    rightTree->setParent (this);
}


HuffmanTree* HuffmanTree::copyNode() const
{
    HuffmanTree* node = new HuffmanTree;

    node->setData(*(new Symbol(*this->getData())));
    if (this->getLeft() == nullptr && this->getRight() == nullptr) {
        return node;
    }

    HuffmanTree* left = nullptr;
    if (this->getLeft() != nullptr) {
        EncodeTree* leftTmp = const_cast<EncodeTree *>(this->getLeft());
        left = dynamic_cast<HuffmanTree *>(leftTmp);
    }

    HuffmanTree* right = nullptr;
    if (this->getRight() != nullptr) {
        EncodeTree* rightTmp = const_cast<EncodeTree *>(this->getRight());
        left = dynamic_cast<HuffmanTree *>(rightTmp);
    }

    left->setParent (node);
    right->setParent (node);

    node->setLeft (left);
    node->setRight (right);

    return node;
}


bool    HuffmanTree::CompareHuffmanTree::operator()(HuffmanTree    *first,    HuffmanTree
*second)
```

```
{
    return first->getData()->weight_ > second->getData()->weight_;
}
```

decode.cpp

```
#include "../Headers/decode.h"


QString decode::decode(QMap<QString, QChar>& charactersCode,
                       const QString& encodedText)
{
    QString result;
    QString buffer;

    for (auto elem : encodedText) {
        if (elem != '0' && elem != '1') {
            return "";
        }

        buffer += elem;

        if (charactersCode.find(buffer) != charactersCode.end()) {
            result += charactersCode[buffer];
            buffer.clear();
        }
    }
    return result;
}


QString decode::decode(const EncodeTree *tree, const QString &encodedText)
{
    QMap<QString, QChar> charactersCode = tree->getCharactersCode();

    return decode (charactersCode, encodedText);
}
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QString>
#include <QMessageBox>
#include <QFile>
#include <QFileDialog>

#include "referencewindow.h"
#include "graphicwindow.h"
```

```
QT_BEGIN_NAMESPACE
namespace Ui
{
class MainWindow;
}
QT_END_NAMESPACE


class MainWindow : public QMainWindow
{
    Q_OBJECT

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

    ~MainWindow();

signals:
    void demonstrateFanoShannonEncode (const QString& text);

    void demonstrateHuffmanEncode (const QString& text);

    void demonstrateDecode (const QString& text);

private slots:
    QString readFromFile (const QString& fileName);

    void on_actionAbout_program_triggered();

    void on_actionAbout_author_triggered();

    void on_fanoShannonEncode_pushButton_clicked();

    void on_huffmanEncode_pushButton_clicked();

    void on_decode_pushButton_clicked();

private:
    Ui::MainWindow* ui;
    ReferenceWindow* referenceWindow;
    GraphicWindow* graphicWindow;
    QFile* inputFile;
};

#endif // MAINWINDOW_H
```

graphicwindow.h

```
#ifndef GRAPHICWINDOW_H
#define GRAPHICWINDOW_H

#include <QWidget>
#include <QGraphicsScene>
#include <QPen>
```

```
#include <QGraphicsTextItem>
#include <QFont>
#include <QGraphicsTextItem>
#include <QFontMetrics>
#include <QMessageBox>
#include <QStringList>

#include <QDebug>

#include <math.h>

#include "../../BinaryTree/Headers/fanoshannontree.h"
#include "../../BinaryTree/Headers/decode.h"
#include "../../BinaryTree/Headers/huffmantree.h"



constexpr int DIAMETER_NODE = 52;
constexpr int SIZE_FONT = 12;
constexpr const char* const FONT = "Times";


QT_BEGIN_NAMESPACE
namespace Ui
{
class GraphicWindow;
}
QT_END_NAMESPACE


class GraphicWindow : public QWidget
{
    Q_OBJECT

public:
    explicit GraphicWindow (QWidget* parent = nullptr);

    ~GraphicWindow();

private slots:
    void closeEvent (QCloseEvent* event);

    void drawTree (const EncodeTree* node,
                   int depth, int nodeNo = 0);

    void drawText (const QString& text, int positionX,
                   int positionY);

    void drawEncoding (EncodeTree& node, const QString& text);

    void clearScene();

    int calculateNodePositionX (int depth, int nodeNo);
```

```
public slots:
    void huffmanEncoding (const QString& text);

    void fanoShannonEncoding (const QString& text);

    void decoding (const QString& text);

signals:
    void myCloseEvent();

private:
    Ui::GraphicWindow* ui;
    QGraphicsScene* graphicsScene;
    int currentPositionY;
};

#endif // GRAPHICWINDOW_H
```

referencewindow.h

```
#ifndef REFERENCEWINDOW_H
#define REFERENCEWINDOW_H

#include <QDialog>
#include <QFile>
#include <QString>


QT_BEGIN_NAMESPACE
namespace Ui
{
class ReferenceWindow;
}
QT_END_NAMESPACE


class ReferenceWindow : public QDialog
{
    Q_OBJECT

public:
    explicit ReferenceWindow (QDialog* parent = nullptr);

    ~ReferenceWindow();

signals:
    void change_textBrowser_signal (const QString& fileName);

private slots:
    void on_fanoShannon_pushButton_clicked();

    void on_huffman_pushButton_clicked();
```

```
    void on_decode_pushButton_clicked();

    void on_aboutTheProgram_pushButton_clicked();

    void change_textBrowser (const QString& fileName);

private:
    Ui::ReferenceWindow *ui;
};

#endif // REFERENCEWINDOW_H
```

mainwindow.cpp

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


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

    referenceWindow = new ReferenceWindow;
    inputFile = new QFile;
    graphicWindow = new GraphicWindow;

    connect (graphicWindow, &GraphicWindow::myCloseEvent,
             this, &MainWindow::show);

    connect (this, &MainWindow::demonstrateDecode,
             graphicWindow, &GraphicWindow::decoding);

    connect (this, &MainWindow::demonstrateHuffmanEncode,
             graphicWindow, &GraphicWindow::huffmanEncoding);

    connect (this, &MainWindow::demonstrateFanoShannonEncode,
             graphicWindow, &GraphicWindow::fanoShannonEncoding);
}


MainWindow::~MainWindow()
{
    delete ui;
    delete referenceWindow;
    delete graphicWindow;
    delete inputFile;
}


QString MainWindow::readFromFile (const QString& fileName)
{
    QFile file (fileName);
```

```
    file.open (QFile::Text | QFile::ReadOnly);

    if (!file.isOpen()) {
        return "";
    }

    QString text = file.readAll();

    file.close();

    return text;
}


void MainWindow::on_actionAbout_program_triggered()
{
    referenceWindow->setModal (true);
    referenceWindow->show();
}


void MainWindow::on_actionAbout_author_triggered()
{
    QString text = readFromFile (":/HTML/GUI/HTML/AboutTheAuthor.html");

    QMessageBox::information (this, "About the author", text);
}


void MainWindow::on_fanoShannonEncode_pushButton_clicked()
{
    QString fileName = QFileDialog::getOpenFileName (this, "Open file");
    QString text = readFromFile (fileName);

    if (text == "") {
        QMessageBox::warning(this, "Error", "Empty data");
        return;
    }

    this->close();

    emit (demonstrateFanoShannonEncode (text));
}


void MainWindow::on_huffmanEncode_pushButton_clicked()
{
    QString fileName = QFileDialog::getOpenFileName (this, "Open file");
    QString text = readFromFile (fileName);

    if (text == "") {
        QMessageBox::warning(this, "Error", "Empty data");
        return;
```

```
    }

    this->close();

    emit (demonstrateHuffmanEncode (text));
}


void MainWindow::on_decode_pushButton_clicked()
{
    QString fileName = QFileDialog::getOpenFileName (this, "Open file");
    QString text = readFromFile (fileName);

    if (text == "") {
        QMessageBox::warning(this, "Error", "Empty data");
        return;
    }

    this->close();

    emit (demonstrateDecode (text));
}
```

graphicwindow.cpp

```
#include "./GUI/Headers/graphicwindow.h"
#include "ui_graphicwindow.h"


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

    graphicsScene = new QGraphicsScene;

    currentPositionY = 0;

    ui->graphicsView->setAlignment (Qt::AlignLeft | Qt::AlignTop);
    ui->graphicsView->setScene (graphicsScene);
}



void GraphicWindow::closeEvent (QCloseEvent* event)
{
    Q_UNUSED (event)

    emit (myCloseEvent());
}


void GraphicWindow::drawTree (const EncodeTree* node,
```

```
                                int depth, int nodeNo)
{
    if (node == nullptr) {
        return;
    }

    QPen blackPen (Qt::black);
    QFont timesNewRoman (FONT, SIZE_FONT);
    QFontMetrics fontMetrics (timesNewRoman);


    int positionY = currentPositionY - depth * DIAMETER_NODE;
    int positionX = calculateNodePositionX (depth, nodeNo);

    graphicsScene->addEllipse (positionX, positionY,
                               DIAMETER_NODE, DIAMETER_NODE, blackPen);

    graphicsScene->addLine (positionX + DIAMETER_NODE / 2, positionY,
                            positionX + DIAMETER_NODE / 2,
                            positionY - DIAMETER_NODE / 2, blackPen);

    if (node->getLeft() != nullptr) {
        graphicsScene->addLine (positionX, positionY + DIAMETER_NODE / 2,
                                calculateNodePositionX (depth - 1,
                                                        nodeNo * 2) +
                                DIAMETER_NODE / 2, positionY +
                                DIAMETER_NODE / 2, blackPen);

        drawText ("0", calculateNodePositionX (depth - 1,
                                               nodeNo * 2) +
                  DIAMETER_NODE / 2 - fontMetrics.width ("0") *
                  2, positionY + DIAMETER_NODE / 2);
    }

    if (node->getRight() != nullptr) {
        graphicsScene->addLine (positionX + DIAMETER_NODE,
                                positionY + DIAMETER_NODE / 2,
                                calculateNodePositionX (depth - 1,
                                                        nodeNo * 2 + 1) +
                                DIAMETER_NODE / 2, positionY +
                                DIAMETER_NODE / 2,
                                blackPen);

        drawText("1", calculateNodePositionX (depth - 1,
                                              nodeNo * 2 + 1) +
                 DIAMETER_NODE / 2 + fontMetrics.width ("1"),
                 positionY + DIAMETER_NODE / 2);
    }

    if (node->isLeaf()) {
        QChar data = node->getData()->data_;

        QString result;
```

```
        if (data == ' ') {
            result = "space";
        }
        else if (data == '\n') {
            result = "\\n";
        }
        else if (data == '\t') {
            result = "tab";
        }
        else {
            result = data;
        }

        drawText (result, positionX + (DIAMETER_NODE -
                                       fontMetrics.width (result) / 2 * 3) / 2,
                  positionY + (DIAMETER_NODE - fontMetrics.height() / 2 * 3)/2);

        result = std::to_string(node->getData()->weight_).c_str();

        drawText (result, positionX + (DIAMETER_NODE -
                                       fontMetrics.width (result) / 2 * 3) / 2,
                  positionY + (DIAMETER_NODE - 1));
    }
    else {
         QString result(std::to_string(node->getData()->weight_).c_str());

         drawText (result, positionX + (DIAMETER_NODE -
                                        fontMetrics.width (result) / 2 * 3) / 2,
                   positionY + (DIAMETER_NODE - fontMetrics.height() / 2 * 3)/2);

    }

    drawTree (node->getLeft(), depth - 1, nodeNo * 2);
    drawTree (node->getRight(), depth - 1, (nodeNo * 2) + 1);
}


void GraphicWindow::drawText (const QString& text, int positionX,
                              int positionY)
{
    QGraphicsTextItem* textItem = nullptr;
    QFont timesNewRoman (FONT, SIZE_FONT);

    textItem = graphicsScene->addText (text, timesNewRoman);
    textItem->setPos (positionX, positionY);

    if (positionY == currentPositionY) {
        QFontMetrics fontMetrics (timesNewRoman);
        int nLines = text.count ('\n');
        int maxHeightChar = fontMetrics.height();
        currentPositionY += (nLines + 1) * maxHeightChar;
    }
```

```
}


void GraphicWindow::drawEncoding (EncodeTree& encodedTree,
                                  const QString& text)
{
    clearScene();

    QString encodedText = encodedTree.encodeText (text);

    {
        drawText ("Input text:", 0, currentPositionY);
        drawText (text, 0, currentPositionY);
    }

    {
        size_t height = encodedTree.getHeight();
        currentPositionY += height * DIAMETER_NODE;
        drawTree (&encodedTree, static_cast <int> (height - 1));
        currentPositionY += 2 * DIAMETER_NODE;
    }

    {
        drawText ("Encoded text:", 0, currentPositionY);
        drawText (encodedText, 0, currentPositionY);
    }

    this->show();
}


void GraphicWindow::clearScene()
{
    graphicsScene->clear();
    currentPositionY = 0;
}


int GraphicWindow::calculateNodePositionX (int depth, int nodeNo)
{
    double currentPositionX = ((pow(2, depth) - 1) / 2 +
                               pow(2, depth) * nodeNo) * DIAMETER_NODE;

    return static_cast <int> (currentPositionX);
}


void GraphicWindow::huffmanEncoding (const QString& text)
{
    HuffmanTree encodedTree;
    drawEncoding (encodedTree, text);
}
```

```
void GraphicWindow::fanoShannonEncoding (const QString& text)
{
    FanoShannonTree encodedTree;
    drawEncoding (encodedTree, text);
}


void GraphicWindow::decoding (const QString& text)
{
    clearScene();

    if (text.count ("\n\n") != 1) {
        QMessageBox::warning (this, "Error", "Incorrect input data!");
        return;
    }

    QStringList stringList = text.split ("\n\n");

    QMap <QString, QChar> characterCodes;
    QString characterCodesString = stringList[0];
    for (auto codeLine : characterCodesString.split ("\n")) {
        QStringList buffer = codeLine.split (" ");
        characterCodes.insert(buffer[buffer.size() - 1], buffer[0][0]);
    }

    qDebug() << characterCodes;

    QString encodedText = stringList[1];

    drawText ("CharacterCodes:", 0, currentPositionY);
    drawText (characterCodesString, 0, currentPositionY);

    drawText ("Encoded text:", 0, currentPositionY);
    drawText (encodedText, 0, currentPositionY);

    QPen redPen (Qt::red);
    QFont timesNewRoman (FONT, SIZE_FONT);
    QFontMetrics fontMetrics (timesNewRoman);

    int beginPositionX = 0;
    int endPositionX = 3;
    int positionY = currentPositionY + 2;

    QString buffer;
    for (auto elem : encodedText) {
        buffer += elem;
        endPositionX += fontMetrics.width (elem);

        if (characterCodes.contains (buffer)) {
            graphicsScene->addLine (beginPositionX + 3, positionY,
                                    endPositionX, positionY, redPen);
```

```
            beginPositionX = endPositionX;
            buffer.clear();
        }
    }

    QString result = decode::decode(characterCodes, encodedText);

    drawText ("Decoded text:", 0, currentPositionY);
    drawText (result, 0, currentPositionY);

    this->show();
}


GraphicWindow::~GraphicWindow()
{
    delete ui;
    delete graphicsScene;
}
```

referencewindow.cpp

```
#include "../Headers/referencewindow.h"
#include "ui_referencewindow.h"


ReferenceWindow::ReferenceWindow (QDialog* parent) :
    QDialog (parent),
    ui (new Ui::ReferenceWindow)
{
    ui->setupUi (this);

    connect (this, &ReferenceWindow::change_textBrowser_signal,
             this, &ReferenceWindow::change_textBrowser);
}


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


void ReferenceWindow::on_fanoShannon_pushButton_clicked()
{
    emit (change_textBrowser_signal (":/HTML/GUI/HTML/Fano-"
                                     "ShannonAlgorithm.html"));
}


void ReferenceWindow::on_huffman_pushButton_clicked()
{
    emit (change_textBrowser_signal (":/HTML/GUI/HTML/HuffmanAlgorithm.html"));
}
```

```
void ReferenceWindow::on_decode_pushButton_clicked()
{
    emit (change_textBrowser_signal (":/HTML/GUI/HTML/DecodeAlgorithm.html"));
}


void ReferenceWindow::on_aboutTheProgram_pushButton_clicked()
{
    emit (change_textBrowser_signal (":/HTML/GUI/HTML/AboutTheProgram.html"));
}


void ReferenceWindow::change_textBrowser(const QString& fileName)
{
    QFile file (fileName);
    file.open (QFile::Text | QFile::ReadOnly);

    QString text = file.readAll();
    ui->textBrowser->setHtml (text);

    file.close();
}
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>411</width>
    <height>158</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>CourseWork</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <layout class="QVBoxLayout" name="verticalLayout">
    <item>
     <layout class="QHBoxLayout" name="horizontalLayout">
      <item>
       <widget class="QPushButton" name="fanoShannonEncode_pushButton">
        <property name="text">
         <string>Fano-Shannon encode</string>
        </property>
       </widget>
      </item>
      <item>
```

```
   <widget class="QPushButton" name="huffmanEncode_pushButton">
    <property name="text">
     <string>Huffman encode</string>
    </property>
   </widget>
  </item>
 </layout>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout_2">
  <item>
   <spacer name="horizontalSpacer">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Minimum</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QPushButton" name="decode_pushButton">
    <property name="sizePolicy">
     <sizepolicy hsizetype="Preferred" vsizetype="Fixed">
      <horstretch>0</horstretch>
      <verstretch>0</verstretch>
     </sizepolicy>
    </property>
    <property name="text">
     <string>Decode</string>
    </property>
   </widget>
  </item>
  <item>
   <spacer name="horizontalSpacer_2">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Minimum</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
```

```
       </spacer>
      </item>
     </layout>
    </item>
   </layout>
  </widget>
  <widget class="QMenuBar" name="menubar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>411</width>
     <height>20</height>
    </rect>
   </property>
   <widget class="QMenu" name="menu">
    <property name="title">
     <string>Help</string>
    </property>
    <addaction name="actionAbout_program"/>
    <addaction name="actionAbout_author"/>
   </widget>
   <addaction name="menu"/>
  </widget>
  <action name="action">
   <property name="text">
    <string>Выход</string>
   </property>
  </action>
  <action name="actionAbout_program">
   <property name="text">
    <string>About the program...</string>
   </property>
  </action>
  <action name="actionAbout_author">
   <property name="text">
    <string>About  the author...</string>
   </property>
  </action>
 </widget>
 <resources/>
 <connections/>
</ui>
```

graphicwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>GraphicWindow</class>
 <widget class="QWidget" name="GraphicWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
```

```
  <width>1040</width>
  <height>585</height>
 </rect>
</property>
<property name="sizePolicy">
 <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
  <horstretch>0</horstretch>
  <verstretch>0</verstretch>
 </sizepolicy>
</property>
<property name="minimumSize">
 <size>
  <width>1040</width>
  <height>585</height>
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>1040</width>
  <height>585</height>
 </size>
</property>
<property name="windowTitle">
 <string> Algorithm demonstration</string>
</property>
<widget class="QGraphicsView" name="graphicsView">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>1040</width>
   <height>585</height>
  </rect>
 </property>
 <property name="sizePolicy">
  <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
   <horstretch>0</horstretch>
   <verstretch>0</verstretch>
  </sizepolicy>
 </property>
 <property name="minimumSize">
  <size>
   <width>1040</width>
   <height>585</height>
  </size>
 </property>
 <property name="maximumSize">
  <size>
   <width>1040</width>
   <height>585</height>
  </size>
 </property>
 <property name="sizeAdjustPolicy">
```

```
    <enum>QAbstractScrollArea::AdjustToContents</enum>
   </property>
  </widget>
 </widget>
 <resources/>
 <connections/>
</ui>
```

referencewindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>ReferenceWindow</class>
 <widget class="QWidget" name="ReferenceWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>570</width>
    <height>216</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Reference</string>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout">
   <item>
    <widget class="QTextBrowser" name="textBrowser">
     <property name="html">
      <string>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.0//EN&quot;
&quot;http://www.w3.org/TR/REC-html40/strict.dtd&quot;&gt;
&lt;html&gt;&lt;head&gt;&lt;meta name=&quot;qrichtext&quot; content=&quot;1&quot;
/&gt;&lt;style type=&quot;text/css&quot;&gt;
p, li { white-space: pre-wrap; }
&lt;/style&gt;&lt;/head&gt;&lt;body style=&quot; font-family:'MS Shell Dlg 2'; font-
size:8.25pt; font-weight:400; font-style:normal;&quot;&gt;
&lt;h2 align=&quot;center&quot; style=&quot; margin-top:16px; margin-bottom:12px; mar-
gin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;&lt;span
style=&quot; font-family:'MS Shell Dlg 2'; font-size:8pt; font-
weight:600;&quot;&gt;CourseWork.&lt;/span&gt;&lt;/h2&gt;
&lt;p style=&quot; margin-top:12px; margin-bottom:12px; margin-left:0px; margin-
right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;&lt;span style=&quot; font-
family:'MS Shell Dlg 2'; font-size:8pt;&quot;&gt;The program is designed to demonstrate
the operation of the Fano-Shannon and Huffman coding algorithms and decoding non-prefix
code.&lt;/span&gt;&lt;/p&gt;
&lt;p style=&quot; margin-top:12px; margin-bottom:12px; margin-left:0px; margin-
right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;&lt;span style=&quot; font-
family:'MS Shell Dlg 2'; font-size:8pt;&quot;&gt;When you click on one of the buttons,
open the file for encoding / decoding, then select the file to record the result. The
program will demonstrate the operation of one of the algorithms and write the result to
a file.&lt;/span&gt;&lt;/p&gt;
&lt;p style=&quot;-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-
left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px; font-family:'MS Shell
```

```
Dlg                          2';                          font-size:8pt;&amp;quot;&amp;gt;&amp;lt;br
/&amp;gt;&amp;lt;/p&amp;gt;&amp;lt;/body&amp;gt;&amp;lt;/html&amp;gt;</string>
     </property>
    </widget>
   </item>
   <item>
    <layout class="QHBoxLayout" name="horizontalLayout">
     <item>
      <widget class="QPushButton" name="fanoShannon_pushButton">
       <property name="text">
        <string>Fano-Shannon algorithm</string>
       </property>
      </widget>
     </item>
     <item>
      <widget class="QPushButton" name="huffman_pushButton">
       <property name="text">
        <string>Huffman algorithm</string>
       </property>
      </widget>
     </item>
     <item>
      <widget class="QPushButton" name="decode_pushButton">
       <property name="text">
        <string>Decode algorithm</string>
       </property>
      </widget>
     </item>
     <item>
      <widget class="QPushButton" name="aboutTheProgram_pushButton">
       <property name="text">
        <string>About the program</string>
       </property>
      </widget>
     </item>
    </layout>
   </item>
  </layout>
 </widget>
 <resources/>
 <connections/>
</ui>
```

AboutTheAuthor.html

```
<h2 style="text-align: center;">About the author</h2>
<p><span lang="en" tabindex="0">The program is developed by student gr.
8304:</span></p>
<p style="text-align: center;"><span lang="en" tabindex="0">Shuka Alexander.</span></p>
```

AboutTheProgram.html

```
<h2 center="" style="text-align: center;"><span>CourseWork</span></h2>
```

```
<p justify="" style="text-align: justify;"><span>The program is designed to demonstrate the operation of the Fano-Shannon and Huffman coding algorithms and decoding non-prefix code.</span></p>
<p justify="" style="text-align: justify;"><span>When you click on one of the buttons, open the file for encoding / decoding, then select the file to record the result. The program will demonstrate the operation of one of the algorithms and write the result to a file.</span></p>
<p style="text-align: justify;"><span></span></p>
```

DecodeAlgorithm.html

```
<h2 style="text-align: center;"><strong>Decoding algorithm</strong></h2>
<p>For decoding, you can use the constructed tree or get character codes from a file. Decoding Algorithm:</p>
<ol>
<li>It is created using a tree, or a dictionary of the form "symbol_code, symbol" is read from a file</li>
In the loop:
<li>The next bit is read, placed in the line-buffer.</li>
<li>Go to the next bit.</li>
<li>If there is an existing character code in the buffer line: the symbol from the dictionary with the given code is read and written to the de-encoding result, the buffer line is cleared.</li>
</ol>
```

Fano-ShannonAlgorithm.html

```
<h2 style="text-align: center;"><span>Shannon-Fano coding algorithm</span></h2>
<ol>
<li><span> The input comes in order of non-increasing frequency data. </span></li>
<li><span>There is a middle that divides the alphabet into approximately two parts. These parts (sums of alphabet frequencies) are approximately equal.</span></li>
<li><span> For the left side, &amp;ldquo;0&amp;rdquo; is assigned, for the right side &amp;ldquo;1&amp;rdquo;, so we get the leaves of the tree. </span></li>
<li><span>Step 2 is repeated until we get a single element of the sequence, i.e. sheet</span></li>
</ol>
```

HuffmanAlgorithm.html

```
<h2 style="text-align: center;"><span lang="en" tabindex="0">Huffman coding algorithm</span></h2>
<ol>
<li><span lang="en" tabindex="0"> The input comes in order of non-increasing frequencies.</span></li>
<li><span lang="en" tabindex="0"> The two lowest-frequency letters of the alphabet are selected, and a parent is created (the sum of the two frequencies of these &amp;ldquo;leaves&amp;rdquo;).</span></li>
<li><span lang="en" tabindex="0"> Descendants are deleted and the parent is written in their place, the &amp;ldquo;branches&amp;rdquo; of the parent are numbered: the left branch is assigned &amp;ldquo;0&amp;rdquo;, the right &amp;ldquo;1&amp;rdquo;. </span></li>
<li><span lang="en" tabindex="0">Step two is repeated until the main parent, the &amp;ldquo;root,&amp;rdquo; is found.</span></li>
</ol>
```

tst_testdecode.cpp

```
#include <QtTest>
#include <QMap>
#include <QString>
#include <QCoreApplication>

#include "../../Source/BinaryTree/Headers/decode.h"


typedef  QMap <QString, QChar> myMap;


class TestDecode : public QObject
{
    Q_OBJECT

private slots:
    void test_data();
    void test();
};


void TestDecode::test_data()
{
    QTest::addColumn<myMap> ("map");
    QTest::addColumn<QString> ("encodedText");
    QTest::addColumn<QString> ("result");

    QTest::newRow ("test0") << myMap ({std::make_pair ("0", 'a')}) <<
                               "000" << "aaa";

    QTest::newRow ("test1") << myMap() << "" << "";

    QTest::newRow ("test2") << myMap ({std::make_pair ("0", 'a'),
                                       std::make_pair ("10", 'b'),
                                       std::make_pair ("11", 'c')}) <<
                               "010111011" << "abcbc";

    QTest::newRow ("test3") << myMap({std::make_pair ("0", 'q'),
                                      std::make_pair ("11", 'w'),
                                      std::make_pair ("100", 'r'),
                                      std::make_pair ("101", 'e')}) <<
                               "00111001011111110001100101" <<
                               "qqwrewwwqqqwqqe";

    QTest::newRow ("test4") << myMap({std::make_pair ("00", L'к'),
                                      std::make_pair ("010", L'с'),
                                      std::make_pair ("011", L'м'),
                                      std::make_pair ("10", L'п'),
                                      std::make_pair ("111", L'а'),
                                      std::make_pair ("1100", L'П'),
                                      std::make_pair ("1101", L'г')}) <<
                               "1100110110011000011111111010010" <<
```

```
                                "Пгпмккааппс";

    QTest::newRow ("test5") << myMap ({std::make_pair ("00", 'a'),
                                       std::make_pair ("01", 'b')}) <<
                                "111111" << "";
}


void TestDecode::test()
{
    QFETCH (myMap, map);
    QFETCH (QString, encodedText);
    QFETCH (QString, result);

    QCOMPARE (decode::decode (map, encodedText), result);
}


QTEST_APPLESS_MAIN (TestDecode)

#include "tst_testdecode.moc"
```

tst_encodefanoshannon.cpp

```
#include <QtTest>
#include <QCoreApplication>

#include "../../Source/BinaryTree/Headers/fanoshannontree.h"


class TestEncodeFanoShannon : public QObject
{
    Q_OBJECT

private slots:
    void test_data();
    void test();
};


void TestEncodeFanoShannon::test_data()
{
    QTest::addColumn <QString> ("text");
    QTest::addColumn <QString> ("encodedText");

    QTest::newRow ("test0") << "a" << "0";

    QTest::newRow ("test1") << "" << "";

    QTest::newRow ("test2") << "aab" << "001";

    QTest::newRow ("test3") << "ЙцЙцйhguytbynihgyjbhjукцУАК" <<
                               "10000001000000110100011001101010"
                               "10110011001010111110000110010100l"
```

```
                                "1101100010111110111110000011101111"
                                "11011111";

    QTest::newRow ("test4") << "купавкаыфчяЦВ       "
                                "вцаыкмасв\n" <<
                                "0111011010111001010001101010101010101"
                                "1111100011001110101101100000000000000"
                                "0001001110001010100111110101011110"
                                "10011111";
}


void TestEncodeFanoShannon::test()
{
    QFETCH (QString, text);
    QFETCH (QString, encodedText);

    FanoShannonTree tree;
    QCOMPARE (tree.encodeText(text), encodedText);
}


QTEST_APPLESS_MAIN (TestEncodeFanoShannon)

#include "tst_testencodefanoshannon.moc"
```

tst_testencodehuffman.cpp

```
#include <QtTest>
#include <QCoreApplication>

#include "../../Source/BinaryTree/Headers/huffmantree.h"


class TestEncodeHuffman : public QObject
{
    Q_OBJECT

private slots:
    void test_data();
    void test();

};


void TestEncodeHuffman::test_data()
{
    QTest::addColumn <QString> ("text");
    QTest::addColumn <QString> ("encodedText");

    QTest::newRow ("test0") << "a" << "0";

    QTest::newRow ("test1") << "" << "";
```

```
    QTest::newRow ("test2") << "aab" << "110";

    QTest::newRow ("test3") << "ЙцЙцйhguytbynihgyjbhjукцУАК" <<
                               "11100111110011111100011010110010101"
                               "10000000010101111011000110100101101"
                               "00000111011001110001011110001001011111";

    QTest::newRow ("test4") << "купавкаыфчяЦВ       "
                               "вцаыкмасв\n" <<
                               "11111110110111100001111111000001"
                               "11001110000000101011101101010101"
                               "01010100110100100000111111010110100"
                               "1110000111010";
}


void TestEncodeHuffman::test()
{
    QFETCH (QString, text);
    QFETCH (QString, encodedText);

    HuffmanTree tree;
    QCOMPARE (tree.encodeText(text), encodedText);
}


QTEST_APPLESS_MAIN (TestEncodeHuffman)

#include "tst_testencodehuffman.moc"
```

CourseWorkAll.pro

```
TEMPLATE = subdirs

SUBDIRS += \
    ./Tests/TestDecode/TestDecode.pro \
    ./Source/CourseWork.pro \
    Tests/TestEncodeFanoShannon \
    Tests/TestEncodeHuffman
```

CourseWork.pro

```
QT       += core gui

TEMPLATE = app

TARGET = CourseWork

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++17

# The following define makes your compiler emit warnings if you use
```

```
# any Qt feature that has been marked deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated
before Qt 6.0.0

SOURCES += \
    ./BinaryTree/Source/decode.cpp \
    ./BinaryTree/Source/huffmantree.cpp \
    ./BinaryTree/Source/fanoshannontree.cpp \
    ./BinaryTree/Source/encodetree.cpp \
    GUI/Source/graphicwindow.cpp \
    main.cpp \
    ./GUI/Source/mainwindow.cpp \
    ./GUI/Source/referencewindow.cpp

HEADERS += \
    ./BinaryTree/Headers/decode.h \
    ./BinaryTree/Headers/huffmantree.h \
    ./BinaryTree/Headers/fanoshannontree.h \
    ./BinaryTree/Headers/encodetree.h \
    ./GUI/Headers/mainwindow.h \
    ./GUI/Headers/referencewindow.h \
    GUI/Headers/graphicwindow.h

FORMS += \
    ./GUI/UI/mainwindow.ui \
    ./GUI/UI/referencewindow.ui \
    GUI/UI/graphicwindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

RESOURCES += \
    resourse.qrc
```

TestDecode.pro

```
QT += testlib
QT -= gui
QT += core

CONFIG += qt console warn_on depend_includepath testcase

TEMPLATE = app

SOURCES +=  tst_testdecode.cpp \
```

```
            ../../Source/BinaryTree/Source/decode.cpp \
            ../../Source/BinaryTree/Source/encodetree.cpp

HEADERS += ../../Source/BinaryTree/Headers/decode.h \
           ../../Source/BinaryTree/Headers/encodetree.h
```

TestEncodeFanoShannon.pro

```
QT += testlib
QT -= gui

CONFIG += qt console warn_on depend_includepath testcase
CONFIG -= app_bundle

TEMPLATE = app

SOURCES +=  tst_testencodefanoshannon.cpp \
            ../../Source/BinaryTree/Source/fanoshannontree.cpp \
            ../../Source/BinaryTree/Source/encodetree.cpp

HEADERS += ../../Source/BinaryTree/Headers/fanoshannontree.h \
           ../../Source/BinaryTree/Headers/encodetree.h
```

TestEncodeHuffman.pro

```
QT += testlib
QT -= gui

CONFIG += qt console warn_on depend_includepath testcase
CONFIG -= app_bundle

TEMPLATE = app

SOURCES +=  tst_testencodehuffman.cpp \
            ../../Source/BinaryTree/Source/huffmantree.cpp \
            ../../Source/BinaryTree/Source/encodetree.cpp

HEADERS += ../../Source/BinaryTree/Headers/huffmantree.h \
           ../../Source/BinaryTree/Headers/encodetree.h
```