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 (текущий контроль)

Вариант 2

Студент гр. 8304 _____ Николаева М. А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Николаева М. А.

Группа 8304

Тема работы: кодирование и декодирование Хаффмана и Фано-Шеннона

Исходные данные:

Пользователю предоставляется возможность выбора количетсва вариантов для генерации. А также возможность задать оперделенные параметры. Есть возможность ввода данных непосредственно в консоль или в виде текстового файла.

Предпологаемый объем пояснительной записки:
Не менее сраниц 14.

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Николаева М. А.

Преподаватель _____ Фирсов М.А.

Содержание

Аннотация

В данной работе были изучены метод кодирования и декодирования Хаффмана и Шеннона-Фано, на языке с++ реализованы оба этих метода.

Для «Текущего контроля» пользователю предоставлены возможности выбора подходящего метода и количества генерируемых заданий. Результаты работы программы предоставляются пользователю как в виде файла, так и в виде текста в консоли.

Работа выполнена в среде Qt Creator.

Summary

In this paper, we studied the Huffman and Shannon-Fano coding and decoding method, both of these methods are implemented in c ++.

For "Current control" the user is given the opportunity to choose the appropriate method and the number of generated tasks. The results of the program are provided to the user both in the form of a file and in the form of text in the console.

The work was done in the Qt Creator environment.

Введение

В данной курсовой работе реализуются 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

Целью данной работы является создание программы, с помощью которой можно будет генерировать задания с ответами к ним для проведения текущего контроля среди студентов.

Постановка задачи

Генерация заданий по кодированию и декодированию методами Хаффмана и Фано-Шеннона. Генерация происходит по заданным пользователем критериям. При этом задания и ответы должны быть в удобной для использования форме.

Спецификация программы

Программа написана на языке C++ с использование фреймворка Qt. Считывание происходит как из файла, так и при помощи консоли.

В файле находятся следующие данные: количество вариантов для последующей генерации, длина незакодированного текста для генерации, а также указание насчет типа задания (кодирование/декодирование, Хаффмана/Фано-шеннона). Эти же данные запрашиваются при считывании с консоли.

```
Enter the number of options to generate:
1
Enter the length of text to generate:
1
Choose the type of jobs: enter 'e' to encode or 'd' to decode
e
Choose the type of encode/decode: enter 'F' to FanoShannon or 'H' to Huffman
F
```

Описание структур данных и функций

Для кодирования и декодирования был создан абстрактный класс EncodeTree, который реализует методы для работы с деревом кодирования. Он содержит виртуальный метод для создания дерева кодирования:

```
virtual void createEncodeTree(const std::string& message) = 0;
```

реализованный в классах наследниках HuffmanTree и FanoShannonTree

Класс является узлом дерева и содержит указатели на правое и левое поддеревья, указатель на родителя, указатель на данные. Данные – это структура {символ, вес}.

```
Symbol* data_;
EncodeTree* parent_;
EncodeTree* left_;
EncodeTree* right_;
```

Методы класса EncodeTree:

```
const EncodeTree* getLeft() const;
```

Возвращает указатель на правое поддерево.

```
const EncodeTree* getRight() const;
```

Возвращает указатель на левое поддерево.

```
const EncodeTree* getParent() const;
```

Возвращает указатель на родителя.

```
const Symbol* getData() const;
```

Возвращает указатель на хранящиеся данные.

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

Объединяет два дерева в общее.

```
void setLeft(EncodeTree* left);
```

Устанавливает левое поддерево.

```
void setRight(EncodeTree* right);
```

Устанавливает правое поддерево.

```
void setParent(EncodeTree* parent);
```

Устанавливает родителя.

```
void setData(const Symbol& data);

bool isEmptyNode() const;
```

Проверяет, является ли узел пустым, возвращает true или false.

```
bool isLeaf() const;
```

```
std::map<std::string, char> getCharactersCode() const;
```
Возвращает словарь, содержащий символы и их коды.

```
std::string encode(const std::string& message);
```
Производить кодирование сообщения. Возвращает закодированное сообщение.

```
void printTree(std::string& result, size_t level = 0) const;
```
Метод для вывода дерева кодирования.

```
std::string createCodeSymbols(const std::string& message) const;
```
Создает код для символов.

Класс FanoShannonTree, написанный для алгоритма Фано-Шеннона, – наследник EncodeTree. В нем реализован виртуальный метод базового класса (создание дерева кодирования с помощью алгоритма Фано-Шеннона из исходной строки).

```
static size_t getMiddle(std::map<char, size_t>& symbolMap,
                        std::vector<char>& symbolVector,
                        size_t left, size_t sum, size_t& leftSum,
                size_t& rightSum);
```
Функция получения середины массива символов. Возвращает индекс середины.

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).

```
HuffmanTree* HuffmanTree::copyNode() const
```
Метод для копирования узла.

Для декодирования были реализованы функции *decode* в пространстве имён *decode*.

Описание интерфейса пользователя

Интерфейс программы для работы с пользователем написан в файле main. Для каждого варианта задания генерируется рандомная строка состоящая из маленьких букв латинского алфавита, содержащая заданное количесвто символов. Если пользователю требуются варианты работ, проверяющие навыки кодирования, то эти строки становятся исходным текстом и записываются в задание. В случае, когда требуется декодирование, программа кодирует рандомную строку, а в задание записывает дерево кодирования и закодированный текст. Также после каждого варианта задания находится ответ на этот вариант, который должен предоставить студент. В задании указано, в каком виде должен быть предоставлен ответ.

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 использовать 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. 2 Варианта заданий.

```
Enter the number of options to generate:
2
Enter the length of text to generate:
4
Choose the type of jobs: enter 'e' to encode or 'd' to decode
e
Choose the type of encode/decode: enter 'F' to FanoShannon or 'H' to Huffman
F

Parameters:
Number of options:2
Length of text:2
encode FanoShannon

OPTION NUMBER 1:
Encode text using the Fano-Shannon algorithm. The result should contain a table of character codes and encoded text.
Text:qppw

Answer(Option 1):
encoded text:100011
character codes:
q  -->  10
p  -->  0
w  -->  11


OPTION NUMBER 2:
Encode text using the Fano-Shannon algorithm. The result should contain a table of character codes and encoded text.
Text:omns

Answer(Option 2):
encoded text:00011011
character codes:
o  -->  00
m  -->  01
n  -->  10
s  -->  11
```

2. Декодирование Фано-Шеннона. 1 Вариант задания.

```
Enter the number of options to generate:
1
Enter the length of text to generate:
2
Choose the type of jobs: enter 'e' to encode or 'd' to decode
d
Choose the type of encode/decode: enter 'F' to FanoShannon or 'H' to Huffman
F

Parameters:
Number of options:1
Length of text:1
decode FanoShannon

OPTION NUMBER 1:
Decode text using the FanoShannon algorithm. The result should contain a table of character codes and decoded text.
encoded text:01
encoding tree:
-q
-k

Answer(Option 1):
decoded text:qk
character codes
q  -->  0
k  -->  1
```

3. Кодирование Хаффмана. 1 Вариант.

```
Enter the number of options to generate:
1
Enter the length of text to generate:
5
Choose the type of jobs: enter 'e' to encode or 'd' to decode
e
Choose the type of encode/decode: enter 'F' to FanoShannon or 'H' to Huffman
H

Parameters:
Number of options:1
Length of text:1
encode Huffman

OPTION NUMBER 1:
Encode text using the Huffman algorithm. The result should contain a table of character codes and encoded text.
Text:arkam

Answer(Option 1):
encoded text:1100011110
character codes:
a  -->  11
r  -->  00
k  -->  01
m  -->  10
```

Тестирование

N	Входные данные
1	2 13 e F
2	1 10 d H
3	1 13 d F

Результаты работы программы представлены ниже в виде скриншотов, так как обьем не позволяет сохранить их в таблице.

Тест 1:

```
Parameters:
Number of options:2
Length of text:2
encode FanoShannon

OPTION NUMBER 1:
Encode text using the Fano-Shannon algorithm. The result should contain a table of character codes and encoded text.
Text:wvlyssgogtpyy

Answer(Option 1):
encoded text:1001010101100010010011110011111011110000
character codes:
w  -->  100
v  -->  1010
l  -->  1011
y  -->  00
s  -->  010
g  -->  011
o  -->  110
t  -->  1110
p  -->  1111


OPTION NUMBER 2:
Encode text using the Fano-Shannon algorithm. The result should contain a table of character codes and encoded text.
Text:ykmwmrpflnxvo

Answer(Option 2):
encoded text:00101000001100000111100101010111100110111101111
character codes:
y  -->  001
k  -->  010
m  -->  000
w  -->  0110
r  -->  0111
p  -->  100
f  -->  1010
l  -->  1011
n  -->  1100
x  -->  1101
v  -->  1110
o  -->  1111
```

Тест 2:

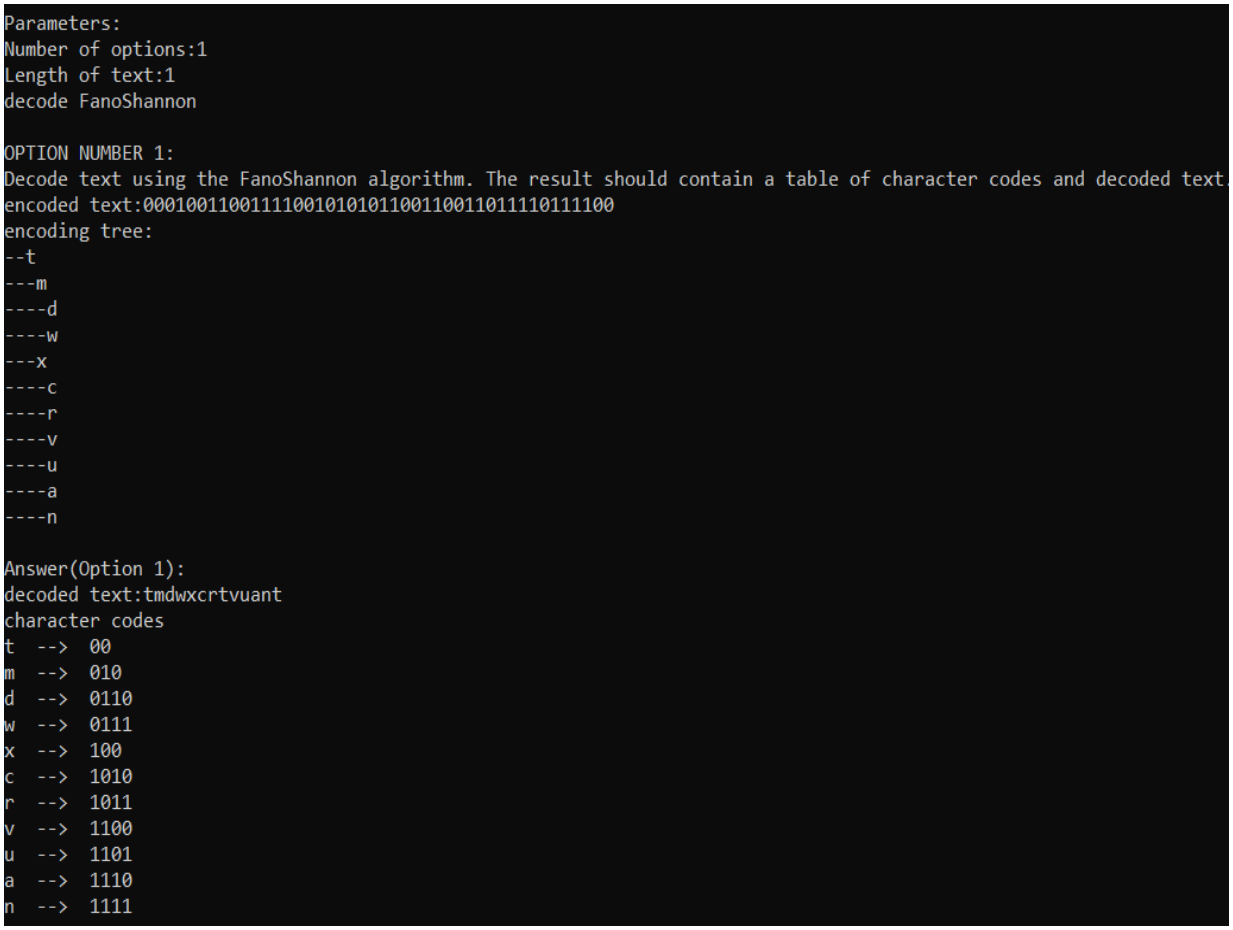
```
Enter the number of options to generate:
1
Enter the length of text to generate:
10
Choose the type of jobs: enter 'e' to encode or 'd' to decode
y
Choose the type of encode/decode: enter 'F' to FanoShannon or 'H' to Huffman
P

Parameters:
Number of options:1
Length of text:1
decode Huffman

OPTION NUMBER 1:
Decode text using the Huffman algorithm. The result should contain a table of character codes and decoded text
encoded text:11001100111101110000101111000 1
encoding tree:
---b
---v
--s
---y
---e
---x
----w
----u

Answer(Option 1):
decoded text:xsyusxbewv
character codes
x  -->  110
s  -->  01
y  -->  100
u  -->  1111
b  -->  000
e  -->  101
w  -->  1110
v  -->  001
```

Тест 3:



Заключение

В ходе выполнения данной курсовой работы была написана программа по генервции заданий для студентов по темам кодирование и декодирование методами Хаффмана и Шеннона-Фано.

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 <iostream>
#include <string>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
#include <fanoshannontree.h>
#include <huffmantree.h>
#include <decode.h>
#include <random>
#include <time.h>

using std::cout;
using std::cin;
using std::cerr;

std::string readTask(std::istream& in, size_t& numberOfOptions, size_t& lengthOfText, bool& encode,
                     bool& decode, bool& fanoShannon, bool& huffman);

std::string readTask(std::ifstream& in, size_t& numberOfOptions, size_t& lengthOfText, bool& encode,
                     bool& decode, bool& fanoShannon, bool& huffman);

int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "");

    if (argc < 3) {
        std::map<std::string, std::string> codeTable;
        std::string task;
        std::string result;

        size_t numberOfOptions;
        size_t lengthOfText;
        bool encode;
        bool decode;
        bool fanoShannon;
        bool huffman;

        if (argc == 2) {
            std::string fileName = argv[1];
            result += "Read encoded text from file " + fileName + "\n";

            std::ifstream inputFile(fileName, std::ios::in);
            task = readTask(inputFile, numberOfOptions, lengthOfText, encode, decode,
                            fanoShannon, huffman);

            inputFile.close();
        }
        else {
            task = readTask(cin, numberOfOptions, lengthOfText, encode, decode, fanoShannon, huffman);
        }

        result += task;

        if (task != "lol") {
            std::string randomStr;
            std::string encodedText;
            std::string decodedText;
            srand(time(NULL));
            for (size_t i = 0; i < numberOfOptions; ++i) {
                result += "OPTION NUMBER " + std::to_string(i + 1) + ":\n";
                randomStr = "";
                for (size_t j = 0; j < lengthOfText; j++){
```

```
                    randomStr += 'a' + rand() % ('z' - 'a');
                }
                if (encode){
                    if (fanoShannon){
                        FanoShannonTree tree;
                        result += "Encode text using the Fano-Shannon algorithm. The result should contain a
table of character codes and encoded text.\n";
                        result += "Text:" + randomStr + "\n\nAnswer(Option " + std::to_string(i + 1) + ")" +
":\nencoded text:" +
                                  tree.encode(randomStr) + "\ncharacter codes:\n";
                        result += tree.createCodeSymbols(randomStr) + "\n\n";
                    } else {
                        HuffmanTree tree;
                        result += "Encode text using the Huffman algorithm. The result should contain a table
of character codes and encoded text.\n";
                        result += "Text:" + randomStr + "\n\nAnswer" + "(Option " + std::to_string(i + 1) +
")" + ":\nencoded text:" +
                                  tree.encode(randomStr) + "\ncharacter codes:\n";
                        result += tree.createCodeSymbols(randomStr) + "\n\n";
                    }
                }
                else {
                    if (fanoShannon){
                        FanoShannonTree tree;
                        encodedText = tree.encode(randomStr);
                        decodedText = decode::decode(&tree, encodedText);
                        result += "Decode text using the FanoShannon algorithm. The result should contain a
table of character codes and decoded text.\n";
                        result += "encoded text:" + encodedText + "\nencoding tree:\n";
                        tree.printTree(result);
                        result += "\nAnswer(Option " + std::to_string(i + 1) + ")" + ":\ndecoded text:" +
decodedText  + "\ncharacter codes\n";
                        result += tree.createCodeSymbols(randomStr) + "\n\n";
                    } else {
                        HuffmanTree tree;
                        encodedText = tree.encode(randomStr);
                        decodedText = decode::decode(&tree, encodedText);
                        result += "Decode text using the Huffman algorithm. The result should contain a table
of character codes and decoded text.\n";
                        result += "encoded text:" + encodedText + "\nencoding tree:\n";
                        tree.printTree(result);
                        result += "\nAnswer(Option " + std::to_string(i + 1) + ")" + ":\ndecoded text:" +
decodedText  + "\ncharacter codes\n";
                        result += tree.createCodeSymbols(randomStr) + "\n\n";
                    }
                }
            }
        } else {
            result += "Incorrect entering";
        }
        cout << result;

        std::ofstream resultFile("result.txt", std::ios::app);
        resultFile << result;
        resultFile.close();

    }
    else {
        cerr << "Error: incorect console's arguments\n";
    }

    return 0;
}


std::string readTask(std::istream& in, size_t& numberOfOptions, size_t& lengthOfText,bool& encode,
                     bool& decode, bool& fanoShannon, bool& huffman) {
```

```
    std::string oneLineStr;
    std::string task;

    cout << "Enter the number of options to generate:\n";
    std::cin >> numberOfOptions;
    std::cin.ignore(32767, '\n');

    task += "\nParameters:\nNumber of options:";
    task += std::to_string(numberOfOptions);
    task += "\n";

    cout << "Enter the length of text to generate:\n";
    std::cin >> lengthOfText;
    std::cin.ignore(32767, '\n');


    task += "Length of text:";
    task += std::to_string(numberOfOptions);
    task += "\n";

    cout << "Choose the type of jobs: enter 'e' to encode or 'd' to decode\n";
    getline(in, oneLineStr);

    if (oneLineStr == "e") {
        encode = true;
        decode = false;
        task += "encode ";
    } else {
        encode = false;
        decode = true;
        task += "decode ";
    }

    cout << "Choose the type of encode/decode: enter 'F' to FanoShannon or 'H' to Huffman\n";
    getline(in, oneLineStr);

    if (oneLineStr == "F") {
        fanoShannon = true;
        huffman = false;
        task += "FanoShannon\n\n";
    } else {
        fanoShannon = false;
        huffman = true;
        task += "Huffman\n\n";
    }

    return task;
}

std::string readTask(std::ifstream& in, size_t& numberOfOptions, size_t& lengthOfText, bool& encode,
                     bool& decode, bool& fanoShannon, bool& huffman) {
    std::string oneLineStr;
    std::string task;

    getline(in, oneLineStr);
    numberOfOptions = atoi(oneLineStr.c_str());
    task += "\nParameters:\nNumber of options:";
    task += std::to_string(numberOfOptions);
    task += "\n";

    getline(in, oneLineStr);
    lengthOfText = atoi(oneLineStr.c_str());
    task += "Length of text:";
    task += std::to_string(numberOfOptions);
    task += "\n\n";

    getline(in, oneLineStr);
```

```
    if (oneLineStr == "e") {
        encode = true;
        decode = false;
        task += "encode ";
    } else {
        encode = false;
        decode = true;
        task += "decode ";
    }

    getline(in, oneLineStr);

    if (oneLineStr == "F") {
        fanoShannon = true;
        huffman = false;
        task += "FanoShannon\n";
    } else {
        fanoShannon = false;
        huffman = true;
        task += "Huffman\n";
    }

    return task;
}
```

encodetree.h

```
#ifndef ENCODETREE_H
#define ENCODETREE_H

#include <iostream>
#include <map>
#include <string>
#include <vector>
#include <algorithm>


struct Symbol
{
    char data_;
    size_t weight_;

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


class EncodeTree
{

public:
    EncodeTree();

    EncodeTree(const Symbol& data);

    virtual ~EncodeTree();

    const EncodeTree* getLeft() const;

    const EncodeTree* getRight() const;

    const EncodeTree* getParent() const;

    const Symbol* getData() const;
```

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

    void setLeft(EncodeTree* left);

    void setRight(EncodeTree* right);

    void setParent(EncodeTree* parent);

    void setData(const Symbol& data);

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

    bool isEmptyNode() const;

    bool isLeaf() const;

    std::map<std::string, char> getCharactersCode() const;

    std::string encode(const std::string& message);

    void printTree(std::string& result, size_t level = 0) const;

    std::string createCodeSymbols(const std::string& message) const;

protected:
    virtual void createEncodeTree(const std::string& message) = 0;

    const EncodeTree* findSymbol(char data) const;


    void getCharactersCode (std::map<std::string, char>& charactersCode,
                            const std::string& path) 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
{

public:
    FanoShannonTree() = default;
    FanoShannonTree(const Symbol& data);
    ~FanoShannonTree() = default;

private:
    void createEncodeTree(const std::string& message);

    void createEncodeTree(std::map<char, size_t>& symbolMap,
                          std::vector<char>& symbolVector,
                          size_t left, size_t right, size_t sum);
```

```
    static size_t getMiddle(std::map<char, size_t>& symbolMap,
                            std::vector<char>& symbolVector,
                            size_t left, size_t sum, size_t& leftSum,
                            size_t& rightSum);
};

#endif // FANOSHANNONTREE_H
```

huffmantree.h

```
#ifndef HUFFMANTREE_H
#define HUFFMANTREE_H

#include <queue>


#include "encodetree.h"



class HuffmanTree : public EncodeTree
{

public:
    HuffmanTree() = default;

    HuffmanTree (const Symbol& data);

    ~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);
    };

private:
    void createEncodeTree (const std::string& text);

    void createEncodeTree (std::map<char, size_t>& charactersAndWeights,  std::vector<char> &symbolVector);

    HuffmanTree* copyNode() const;
};

#endif // HUFFMANTREE_H
```

decode.h

```
#ifndef DECODE_H
#define DECODE_H

#include <string>
#include <map>

#include "encodetree.h"
```

```
namespace decode {
std::string decode (std::map<std::string, char>& charactersCode,
                const std::string& encodedText);

std::string decode (const EncodeTree* tree, const std::string& encodedText);
};

#endif // DECODE_H
```

encodetree.cpp

```
#include "encodetree.h"


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


EncodeTree::EncodeTree (const Symbol& data)
{
    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_;
    }
}
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_;
}


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

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


std::string EncodeTree::encode(const std::string &message)
{
    this->createEncodeTree(message);

    std::string code;
    std::string path;

    for (auto elem : message) {
        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();
    }

    std::string res;
    res += code;

    return res;
}

const EncodeTree* EncodeTree::findSymbol (char data) const
{
    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 (std::string& result, size_t depth) const
{
    if (this->isLeaf()) {
        for (size_t i = 0; i < depth; ++i){
            result += "-";
        }

        char 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);
```

```
        }
    }
}



std::string EncodeTree::createCodeSymbols(const std::string &message) const
{
    std::vector<char> symbolVector;

    std::string path;
    std::string res;

    for (auto elem : message){
        if (find(symbolVector.begin(), symbolVector.end(), elem) == symbolVector.end() )
            symbolVector.push_back(elem);
    }

    for (size_t i = 0; i < symbolVector.size(); ++i){
        const EncodeTree* node = this->findSymbol(symbolVector[i]);
        while (node->parent_ != nullptr) {
            if (node->parent_->left_ == node) {
                path = "0" + path;
            }
            else {
                path = "1" + path;
            }

            node = node->parent_;
        }

        if (symbolVector[i] == '\n') {
            res += "\\n";
        }
        else if (symbolVector[i] == ' ') {
            res += "space";
        }
        else if (symbolVector[i] == '\t') {
            res += "tabulation";
        }
        else {
            res.push_back(symbolVector[i]);
        }

        res += "  -->  " + path + "\n";
        path.clear();
    }

    return res;
}

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::getCharactersCode (std::map<std::string, char>& charactersCode,
                                    const std::string& path) const
{
```

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

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

std::map<std::string, char> EncodeTree::getCharactersCode() const
{
    std::map<std::string, char> 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;
}
```

fanoshannontree.cpp

```
#include "fanoshannontree.h"


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


void FanoShannonTree::createEncodeTree(const std::string &message)
{
    std::map<char, size_t> symbolMap;
    std::vector<char> symbolVector;
    for (auto elem : message) {
        if (symbolMap.count(elem) == 0) {
            symbolMap.insert(std::make_pair(elem, 1));
            symbolVector.push_back(elem);
        }
        else {
            symbolMap[elem] += 1;
        }
    }

    std::sort(symbolVector.begin(), symbolVector.end(),
              [&symbolMap] (char first, char second) {
        return symbolMap[first] > symbolMap[second]; });

    if (symbolVector.size() == 1) {
        char data = symbolVector[0];
```

```
        Symbol symbol(data, symbolMap[data]);
        FanoShannonTree* leftTree = new FanoShannonTree(symbol);
        this->setLeft(leftTree);
        this->setData(Symbol(0, symbolMap[data]));
        leftTree->setParent(this);
    }

    return createEncodeTree(symbolMap, symbolVector, 0,
                            symbolVector.size(), message.length());
}


void FanoShannonTree::createEncodeTree(std::map<char, size_t> &symbolMap,
                                       std::vector<char> &symbolVector,
                                       size_t left, size_t right, size_t sum)
{
    if (left >= right) {
        return;
    }
    if (right == left + 1) {
        char data = symbolVector[left];
        Symbol symbol(data, symbolMap[data]);
        this->setData(symbol);
        return;
    }

    size_t leftSum = 0;
    size_t rightSum = 0;
    size_t middle = getMiddle(symbolMap, symbolVector, left, sum, leftSum, rightSum);

    FanoShannonTree* leftTree = new FanoShannonTree();
    leftTree->createEncodeTree(symbolMap, symbolVector, left,
                               middle + 1, leftSum);

    FanoShannonTree* rightTree = new FanoShannonTree();
    rightTree->createEncodeTree(symbolMap, symbolVector, middle + 1,
                                right, rightSum);

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


size_t FanoShannonTree::getMiddle(std::map<char, size_t> &symbolMap,
                                  std::vector<char> &symbolVector,
                                  size_t left, size_t sum, size_t &leftSum,
                                  size_t &rightSum)
{
    size_t middle = left;

    leftSum = symbolMap[symbolVector[middle]];
    rightSum = sum - leftSum;

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

    while (delta + static_cast<long>(symbolMap[symbolVector[middle+1]]) < 0) {
        ++middle;
        char data = symbolVector[middle];
        leftSum += symbolMap[data];
        rightSum -= symbolMap[data];
        delta = static_cast<long>(leftSum) - static_cast<long>(rightSum);
    }
    return middle;
}
```

huffmantree.cpp

```
#include "huffmantree.h"


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


void HuffmanTree::createEncodeTree (const std::string& text)
{
    std::map<char, size_t> charactersAndWeights;
    std::vector<char> symbolVector;
    for (auto elem : text) {
        if (charactersAndWeights.count (elem) == 0) {
            charactersAndWeights.insert (std::make_pair(elem, 1));
            symbolVector.push_back(elem);
        }
        else {
            charactersAndWeights[elem] += 1;
        }
    }

    if (charactersAndWeights.size() == 1) {
        char 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, symbolVector);
    }
}



void HuffmanTree::createEncodeTree (std::map<char, size_t>& charactersAndWeights,  std::vector<char>
&symbolVector)
{
    std::priority_queue<HuffmanTree*, std::vector<HuffmanTree*>,
            HuffmanTree::CompareHuffmanTree> nodes;

    for (auto elem : symbolVector) {
        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 "decode.h"


std::string decode::decode(std::map<std::string, char>& charactersCode,
                        const std::string& encodedText)
{
   std::string result;
   std::string 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;
}


std::string decode::decode(const EncodeTree *tree, const std::string &encodedText)
{
    std::map<std::string, char> charactersCode = tree->getCharactersCode();

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