

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

**ОТЧЕТ**  
**по лабораторной работе №5**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Сортировки**

Студент гр. 9382

\_\_\_\_\_

Рыжих Р.В.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2020

## **Цель работы.**

Изучить алгоритмы статического кодирования и декодирования Фано-Шеннона и Хаффмана, научиться использовать их на практике.

## **Основные теоретические положения.**

Кодирование — это преобразование информации из одной ее формы представления в другую, наиболее удобную для её хранения, передачи или обработки.

При кодировании используется двоичный код.

Кодирование алгоритмами Фано-Шеннона и Хаффмана производится с помощью бинарных деревьев.

Бинарное дерево - корневое дерево, каждая вершина которого имеет не более двух дочерних, чаще всего чётко упорядоченных : левую и правую вершин.

Если каждый узел бинарного дерева, не являющийся листом, имеет непустые правые и левые поддеревья, то дерево называется строго бинарным деревом.

Требования к бинарному кодовому дереву Фано-Шеннона: корнем дерева является множество всех кодируемых элементов, расположенных в порядке по убыванию веса или, если веса равны, по возрастанию кодов символов в таблице ASCII; любой левый брат всегда не превышает своего правого брата; разница весов между любыми двумя братьями всегда минимальна; если при разделении элемента на два других, оба элемента имеют одинаковый вес, то левым становится тот, первый символ которого идет в таблице ASCII раньше (для букв это алфавитный порядок).

Требования к бинарному кодовому дереву Хаффмана: изначально все элементы выстраиваются по убыванию весов или , если веса равны, по возрастанию кодов символов в таблице ASCII (для букв это алфавитный порядок) ; вес любого левого брата не превышает вес любого право брата; при объединении 2 элементов в один, новый элемент перемещается влево, пока слева

от него не будет элемента большего чем он, а справа — не превышающего его ; если при объединении оба элемента равны по весу, то левым потомком становится последний элемент в списке, а правым — предпоследний.

Декодирование — это процесс восстановления информации из ее представления в закодированном виде к исходному виду.

Алгоритм Фано-Шеннона — один из первых алгоритмов сжатия. Алгоритм использует коды переменной длины: часто встречающийся символ кодируется кодом меньшей длины, редко встречающийся — кодом большей длины. Коды Фано-Шеннона префиксные, то есть никакое кодовое слово не является префиксом любого другого. Это свойство позволяет однозначно декодировать любую последовательность кодовых слов.

Один из первых алгоритмов эффективного кодирования информации был предложен Д.А.Хаффманом в 1952 году. Идея алгоритма состоит в следующем: зная вероятности появления символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. В коде Хаффмана также никакое кодовое слово не является префиксом любого другого.

Оба алгоритма на вход получают таблицу частот символов и на основании этой таблицы строят бинарное дерево кодов символов. Затем с помощью этого дерева информация кодируется.

Для декодирования также необходимо бинарное дерево, на основании которого восстанавливается закодированная информация.

### **Задание.**

#### **Вариант №2.**

На вход подается текстовое сообщение. Требуется закодировать и декодировать содержимое статическими методами Хаффмана и ФаноШеннона. Коды символов хранятся в бинарном дереве.

### **Ход работы:**

#### **1) Алгоритм построения бинарного дерева Хаффмана.**

1. На вход подается строка символов.
2. Высчитываются частоты символов в строке.
3. Частоты символов упорядочиваются в порядке по убыванию.
4. Дерево Хаффмана строится от листьев к корню.
5. Из двух элементов с наименьшим весом образуется новый элемент, вес которого равен сумме этих элементов.
6. Меньший по весу из двух элементов становится левым потомком нового элемента, больший — правым.
7. Новый элемент «передвигается» на свое место, чтобы слева от него были элементы с большим весом, а справа с таким же весом, либо с меньшим (чтобы высота дерева была минимальной).
8. Затем операция 5 — 7 повторяется рекурсивно.
9. Когда останется всего один элемент, он будет являться деревом Хаффмана.

#### **2) Алгоритм построения бинарного дерева Фано-Шеннона**

1. На вход подается строка символов
2. Высчитываются частоты символов в строке.
3. Частоты символов упорядочиваются в порядке по убыванию.
4. Дерево Фано-Шеннона строится от корня к листьям.
5. Корнем всего дерева является элемент, состоящий из всех символов строки, расположенных в порядке по убыванию их частот, вес корня равен сумме всех частот.
6. Корень дерева разбивается на два поддеревья, таким образом, чтобы разница между весами этих поддеревьев была минимальной.

7. Поддерево с меньшим весом становится левым потомком, а с большим весом — правым потомком.

8. Операция 6-7 рекурсивно повторяется для каждого поддерева, пока количество символов в корне поддерева больше 1.

### 3) Кодирование сообщения.

1. Каждому символу сопоставляется его код из бинарного дерева.
2. Происходит проход по сообщению, вместо каждого символа подставляется его код.

### 4) Декодирование сообщения.

1. Посимвольно считывается закодированное сообщение.
2. Одновременно с этим происходит обход по бинарному дереву.
3. Если встречается 1 — происходит переход в правое поддерево, 0 — влевое.
4. Когда процесс достигнет листа, вместо кода символа подставляется самсимвол.

### 5) Программа, реализующая алгоритм содержит в себе:

1. Класс бинарного дерева `BinaryTree`, который определен в файле `binarytree.h`. Бинарное дерево содержит корень, который является строкой, число `num`, которое определяет вес корня и указатели на левое и правое поддерева. В классе реализованы базовые методы бинарного дерева ( `getRoot()`, `getLeft()`, `getRight()`, `getNum()`, `setLeft()`, `setRight()` ). Также реализован метод, изображающий бинарное дерево в виде уступчатого списка `printBT()`.

2. В классе бинарного дерева перегружены 3 конструктора. Первый получает на вход 2 бинарных дерева, в результате чего создается бинарное дерево, левым и правым поддеревом которого являются переданные деревья, корнем этого дерева является строка, которая состоит из корней переданных деревьев. Второй конструктор получает на вход

строку и число, в результате чего создается дерево, корнем которого является переданная строка, а весом корня - переданное число. Третий конструктор получает вектор бинарных деревьев, в результате получается дерево, корнем которого является строка, полученная соединением всех корней деревьев вектора, а весом корня — сумма всех весов переданных деревьев.

3. Функция `createBinaryTreeHuff()` строит бинарное дерево Хаффмана. Навход функция получает вектор бинарных деревьев (в самом первом вызове это деревья, не имеющие потомков). Затем 2 элемента с наименьшим весом соединяются в один элемент и функция вызывается рекурсивно, пока в векторе больше 1 элемента. Если в векторе остался один элемент - то этот элемент является деревом Хаффмана и оно возвращается из функции.

4. Функция `createBinaryTreeSF()` строит бинарное дерево Шеннона-Фано. Функция принимает вектор бинарных деревьев, состоящий из деревьев без потомков, а также бинарное дерево, корнем которого является строка, образованная соединением всех корней дерева, а весом корня — сумма весов корней деревьев вектора. В функции переданное дерево разбивается на два дерева так, чтобы разница между весами этих деревьев была минимальной. Затем устанавливается левый потомок исходного дерева — рекурсивным вызовом функции `createBinaryTree()`, в качестве аргумента функции передается наименьшее (по весу корня) дерево, которое получилось в результате разбиения исходного на два. Правый потомок устанавливается аналогично.

5. Функция `destroy()`, которая рекурсивно освобождает память, выделенную под бинарное дерево.

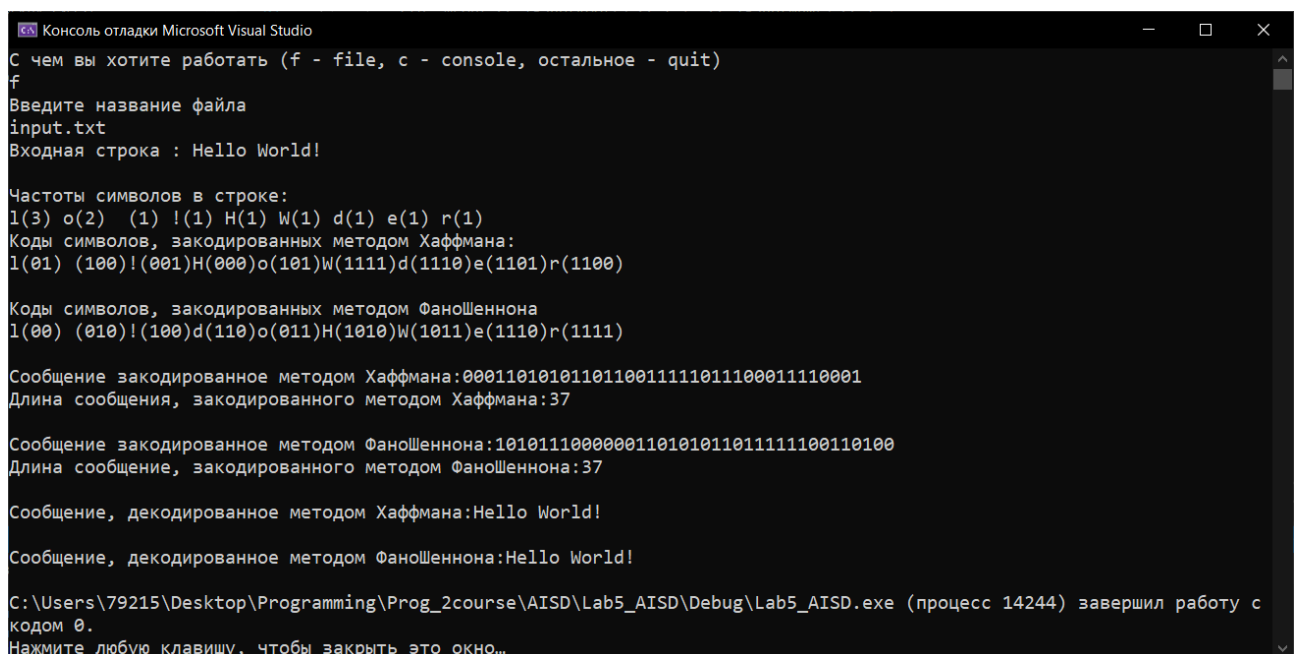
6. Класс `File`, определенный в файле `File.h`. Класс предназначен для работы с файлом. В конструкторе класса открывается файл, в деструкторе он закрывается. С помощью метода `readFile()` считывается информация с файла.

7. Функция `decoding()`, файла `decoding.h`, предназначена для декодирования сообщения. С помощью бинарного дерева эта функция восстанавливает переданное сообщение.
8. Функция `createBTVector()`, которая возвращает массив деревьев, состоящих из одного элемента, с заполненными частотами.
9. Функция `printInfo()`, которая выводит коды символов, закодированных методами Хаффмана и Фано-Шеннона.
10. Функция `startCoding()`, в которой, с помощью бинарных деревьев Фано-Шеннона и Хаффмана, кодируется и затем выводится переданная строка.
11. Функция `main()`, в которой считывается переданная строка, вызывается функция `startCoding()`, для кодирования информации и затем функция `decoding()` для декодирования информации.

Исходный код программы см. в Приложении А.

### Примеры работы программы

На рисунках 1, 2, 3, 4, 5 представлены примеры работы программы с различными исходными данными.



```
Консоль отладки Microsoft Visual Studio
С чем вы хотите работать (f - file, c - console, остальное - quit)
f
Введите название файла
input.txt
Входная строка : Hello World!

Частоты символов в строке:
l(3) o(2) (1) !(1) H(1) W(1) d(1) e(1) r(1)
Коды символов, закодированных методом Хаффмана:
l(01) (100)!(001)H(000)o(101)W(1111)d(1110)e(1101)r(1100)

Коды символов, закодированных методом ФаноШеннона
l(00) (010)!(100)d(110)o(011)H(1010)W(1011)e(1110)r(1111)

Сообщение закодированное методом Хаффмана:0001101010110011111011100011110001
Длина сообщения, закодированного методом Хаффмана:37

Сообщение закодированное методом ФаноШеннона:101011100000011010101101111100110100
Длина сообщения, закодированного методом ФаноШеннона:37

Сообщение, декодированное методом Хаффмана:Hello World!

Сообщение, декодированное методом ФаноШеннона:Hello World!

C:\Users\79215\Desktop\Programming\Prog_2course\AISD\Lab5_AISD\Debug\Lab5_AISD.exe (процесс 14244) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно...
```

Рисунок 1 — Демонстрация работы программы с входными данными №1

```
Консоль отладки Microsoft Visual Studio
с
Введите строку
Привет, Андрей!
Входная строка : Привет, Андрей!

Частоты символов в строке:
е(2) р(2) (1) !(1) ,(1) А(1) П(1) в(1) д(1) и(1) й(1) н(1) т(1)
Коды символов, закодированных методом Хаффмана:
(000)е(010)р(001)!(1111),(1110)А(1101)П(1100)в(1011)д(1010)и(1001)й(1000)н(0111)т(0110)

Коды символов, закодированных методом ФаноШеннона
(000)е(010)р(011)!(0010),(0011)А(1000)П(1001)в(1010)д(1011)и(1100)й(1101)н(1110)т(1111)

Сообщение закодированное методом Хаффмана:11000011001101101001101111000011010111101000101010001111
Длина сообщения, закодированного методом Хаффмана:55

Сообщение закодированное методом ФаноШеннона:100101111001010010111100110001000111010110110110101010
Длина сообщение, закодированного методом ФаноШеннона:55

Сообщение, декодированное методом Хаффмана:Привет, Андрей!

Сообщение, декодированное методом ФаноШеннона:Привет, Андрей!

C:\Users\79215\Desktop\Programming\Prog_2course\AISD\Lab5_AISD\Debug\Lab5_AISD.exe (процесс 14816) завершил работу с
кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно...
```

Рисунок 2 — Демонстрация работы программы с входными данными №2

```
Консоль отладки Microsoft Visual Studio
Текущая кодовая страница: 1251
С чем вы хотите работать (f - file, c - console, остальное - quit)
с
Введите строку
Привет мир! Привет мир! Привет мир!
Входная строка : Привет мир! Привет мир! Привет мир!

Частоты символов в строке:
и(6) р(6) (5) !(3) П(3) в(3) е(3) м(3) т(3)
Коды символов, закодированных методом Хаффмана:
(100)!(011)П(010)в(001)е(000)и(110)р(101)м(1111)т(1110)

Коды символов, закодированных методом ФаноШеннона
и(00) (010)!(100)е(110)р(011)П(1010)в(1011)м(1110)т(1111)

Сообщение закодированное методом Хаффмана:0101011100010001110100111111010101110001010111000100011101001111110101110
001010111000100011101001111110101011
Длина сообщения, закодированного методом Хаффмана:111

Сообщение закодированное методом ФаноШеннона:1010011001011111011110101110000111000101010011001011110111101011100001110
0010101001100101111011111010111000011100
Длина сообщение, закодированного методом ФаноШеннона:111

Сообщение, декодированное методом Хаффмана:Привет мир! Привет мир! Привет мир!

Сообщение, декодированное методом ФаноШеннона:Привет мир! Привет мир! Привет мир!
```

Рисунок 3 — Демонстрация работы программы с входными данными №3





## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

#### binarytree.h:

```
#ifndef BINARYTREEHUFF_H
#define BINARYTREEHUFF_H
#include <string>
#include <vector>
#include <algorithm>
typedef std::wstring Elem;
class BinaryTree
{
public:
    BinaryTree(BinaryTree* left, BinaryTree* right); //конструктор
    BinaryTree(Elem elem, int num); //конструктор
    BinaryTree(std::vector<BinaryTree*>& vect); //конструктор
    Elem getRoot(); //возвращает значение корня
    BinaryTree* getLeft(); //возвращает левое поддерево
    BinaryTree* getRight(); //возвращает правое поддерево
    void setRight(BinaryTree* bt); //устанавливает правое поддерево
    void setLeft(BinaryTree* bt); //устанавливает левое поддерево
    int getNum(); //возвращает вес корня
    void setNum(int num); //устанавливает вес корня
    void printBT(int level = 0); //изображает дерево в виде уступчатого списка
    void printCods();
    void
        getCods(std::vector<std::pair<std::wstring, std::wstring>>& vec,
            std::wstring str);

private:
    Elem root;
    BinaryTree * left = nullptr;
    BinaryTree* right = nullptr;
    int num;
};
void destroy(BinaryTree* tree); //освобождение памяти
BinaryTree* createBinaryTreeHuff(std::vector<BinaryTree*>& elements); //создает дерево Хаффмана
BinaryTree* createBinaryTreeSF(BinaryTree* tree, std::vector<BinaryTree*>
elements); //создает дерево Шеннона-Фано
#endif // BINARYTREEHUFF_H
```

#### binarytree.cpp:

```
#include "binarytree.h"
#include <iostream>
template<class T> void swp(T & el1, T & el2) //меняет элементы местами
{
    T temp = el1;
    el1 = el2;
    el2 = temp;
}
BinaryTree::BinaryTree(BinaryTree* left, BinaryTree* right) //конструктор
{
    this->left = left;
    this->right = right;
    this->num = left->getNum() + right->getNum();
    this->root = left->root + right->root;
}
BinaryTree::BinaryTree(Elem elem, int num) //конструктор
{
    this->root = elem;
    this->num = num;
}
BinaryTree::BinaryTree(std::vector<BinaryTree*>& vect) //конструктор
```

```

{
    std::sort(vect.begin(), vect.end(), [](BinaryTree* e1, BinaryTree* e2)
    {
        if (e1->getNum() != e2->getNum())
            return e1->getNum() > e2->getNum();
        return e1->getRoot()[0] < e2->getRoot()[0];
    });
    std::wstring temp;
    int n = 0;
    for (auto i : vect)
    {
        temp += i->getRoot();
        n += i->getNum();
    }
    this->root = temp;
    this->num = n;
}
void destroy(BinaryTree* tree)//очистение памяти
{
    if (tree != nullptr)
    {
        destroy(tree->getLeft());
        destroy(tree->getRight());
        delete tree;
    }
}
Elem BinaryTree::getRoot();//возвращает корень дерева
{
    return this->root;
}
BinaryTree* BinaryTree::getRight();//возвращает правое поддерево
{
    return this->right;
}
BinaryTree* BinaryTree::getLeft();//возвращает левое поддерево
{
    return this->left;
}
int BinaryTree::getNum();//возвращает вес корня дерева
{
    return this->num;
}
void BinaryTree::setNum(int num)//устанавливает вес корня дерева
{
    this->num = num;
}
void BinaryTree::printBT(int level)//рекурсивно изображает дерево в виде уступчатого списка
{
    if (left != nullptr)
        left->printBT(level + 1);
    for (int i = 0; i < level; i++)std::wcout << " --- ";
    std::wcout << root << '(' << num << ')' << '\n';
    if (right != nullptr)
        right->printBT(level + 1);
}
void BinaryTree::setLeft(BinaryTree* bt)//устанавливает левое поддерево
{
    this->left = bt;
}
void BinaryTree::setRight(BinaryTree* bt)//устанавливает правое поддерево
{
    this->right = bt;
}
BinaryTree* createBinaryTreeHuff(std::vector<BinaryTree*>&

```

```

        elements)//строится дерево Хаффмана
    {
        //если в векторе остался один элемент, то этот элемент является построенным деревом
        Хаффмана и он возвращается из функции
        if (elements.size() == 1)
            return elements[0];
        BinaryTree* temp1 = elements[elements.size() - 1];
        BinaryTree* temp2 = elements[elements.size() - 2];
        elements.pop_back(); //удаляем элемент из конца вектора
        elements.pop_back(); //удаляем элемент из конца вектора
        BinaryTree* parent = new BinaryTree(temp1, temp2); //создаем бинарное дерево, в котором
        temp1 и temp2 будут потомками
        elements.push_back(parent); //добавляем новое дерево в вектор
        //перемещаем на нужное место добавленный элемент (чтобы вектор был поубыванию)
        for (auto i = elements.end() - 1; i != elements.begin(); i--)
        {
            if ((*i)->getNum() >= (*(i - 1))->getNum())
            {
                swp(*i, *(i - 1));
            }
            else
            {
                break;
            }
        }
        return createBinaryTreeHuff(elements);
    }
}
BinaryTree* createBinaryTreeSF(BinaryTree* tree, std::vector<BinaryTree*>
elements)//строится дерево Фано- Шеннона
{
    //если длина корня дерева равна 1, то возвращаем это дерево
    if (tree->getRoot().length() == 1)
    {
        return tree;
    }
    std::wstring rootString = tree->getRoot(); //получаем значение корня
    std::wstring rootBLeft = L""; //строка для корня левого поддерева
    std::wstring rootBRight = L""; //строка для корня правого поддерева
    int weightBLeft = 0; //вес для корня левого поддерева
    int weightBRight = 0; //вес для корня правого поддерева
    int lastWeight; //для веса корня последнего элемента, добавленного в правое поддерево
    std::wstring lastSymb; //для корня последнего элемента, который будет добавлен
    в правое поддерево
    for (auto i : rootString) //проходимся по каждому символу корня
    {
        for (auto j : elements) //проходимся по каждому элементу вектора
        {
            if (i == j->getRoot()[0])
            {
                //если текущий вес корня правого поддерева меньше половины
                //веса дерева, то
                //добавляем к правому поддереву еще один элемент,
                иначе добавляем элемент к левому поддереву
                if (weightBRight <= tree->getNum() / 2)
                {
                    rootBRight += j->getRoot();
                    weightBRight += j->getNum();
                    lastWeight = j->getNum();
                    lastSymb = j->getRoot();
                }
                else
                {
                    rootBLeft += j->getRoot();
                    weightBLeft += j->getNum();
                }
            }
        }
    }
}

```

```

    }
    }
    }
    //проверяем получили ли минимальную разницу между весами корней правого и левого
    поддерева
    //если нет, то удаляем из правого поддерева последний элемент и ставим его в
    начало левого поддерева
    int diff = weightBTRight - weightBTLeft;
    if (abs(weightBTRight - lastWeight - (weightBTLeft + lastWeight)) <= diff)
    {
        weightBTRight -= lastWeight;
        weightBTLeft += lastWeight;
        rootBTLeft = lastSymb + rootBTLeft;
        rootBTRight.pop_back();
        if (weightBTLeft >= weightBTRight)
        {
            swp(weightBTLeft, weightBTRight);
            swp(rootBTLeft, rootBTRight);
        }
    }
    tree->setLeft(createBinaryTreeSF(new
        BinaryTree(rootBTLeft, weightBTLeft), elements));
    tree->setRight(createBinaryTreeSF(new
        BinaryTree(rootBTRight, weightBTRight), elements));
    return tree;
}
void
BinaryTree::getCods(std::vector<std::pair<std::wstring, std::wstring>>&vec, std::wstring str
= L"")
{
    if (this->left != nullptr && this->right != nullptr)
    {
        this->left->getCods(vec, str + L"0");
        this->right->getCods(vec, str + L"1");
    }
    else
    {
        vec.push_back(std::pair<std::wstring, std::wstring>(this->getRoot(),
str));
    }
}
void BinaryTree::printCods()
{
    std::vector<std::pair<std::wstring, std::wstring>> vec;
    getCods(vec, L"");
    std::sort(vec.begin(), vec.end(), [](std::pair<std::wstring, std::wstring>i,
std::pair<std::wstring, std::wstring> j)
    {
        if (i.second.length() != j.second.length())
        {
            return i.second.length() < j.second.length();
        }
        return i.first < j.first;
    });
    std::wstring str = L"";
    for (auto i : vec)
    {
        str += i.first;
        str += L"(";
        str += i.second;
        str += L")";
    }
    std::wcout << str << '\n';
}

```

```
}
```

### decoding.h:

```
#ifndef DECODING_H
#define DECODING_H
#include <string>
#include "binarytree.h"
std::wstring decoding(BinaryTree* bt, std::wstring message); //функция декодирования
сообщение(объявление)
#endif // DECODING_H
```

### decoding.cpp:

```
#include "decoding.h"
#include <iostream>
std::wstring decoding(BinaryTree * bt, std::wstring message)
//функция декодирования сообщение
{
    std::wstring result = L"";
    BinaryTree* root = bt;
    int i = 0;
    while (i <= message.size())
    {
        if (i == message.size())
        {
            result += bt->getRoot();
            break;
        }
        if (message[i] == '0') //если 0, то идем в левое поддерево, если оно есть, если
его нет, то добавляем значение корня к результату
        {
            if (bt->getLeft() != nullptr)
            {
                bt = bt->getLeft();
                i++;
            }
            else
            {
                result += bt->getRoot();
                bt = root;
            }
        }
        else //если 1, то идем в правое поддерево, если оно есть, если его нет, то
добавляем значение корня к результату
        {
            if (bt->getRight() != nullptr)
            {
                bt = bt->getRight();
                i++;
            }
            else
            {
                result += bt->getRoot();
                bt = root;
            }
        }
    }
    return result;
}
```

### file.h:

```
#ifndef FILE_H
#define FILE_H
#include <fstream>
#include <iostream>
#include <vector>
```

```

#include "binarytree.h"
#include <regex>
class File
{
public:
    File(std::wstring file);
    ~File();
    void readFile(std::wstring& message);
private:
    std::wifstream* fInput = nullptr;
};
#endif // FILE_H

file.cpp:
#include "file.h"
File::File(std::wstring file)
{
    fInput = new std::wifstream; //создание объекта wifstream
    try
    {
        fInput->open(file);
        //открытие файла
        if (!fInput->is_open())
            throw 1;
    }
    catch (int)
    {
        std::cerr << "Не удалось открыть файл!\n";
        exit(0);
    }
}
File::~File()
{
    if (fInput->is_open())
        fInput->close(); //закрытие файла
    if (fInput != nullptr)
        delete fInput;
}
void File::readFile(std::wstring& message) //считывание строк с файла
{
    std::getline(*fInput, message);
    // *fInput >> message;
}

```

### **main.cpp:**

```

#include <iostream>
#include "file.h"
#include "decoding.h"
#include <locale>
#include <Windows.h>

std::vector<BinaryTree*> createBTVector(std::wstring message) //вернуть вектора деревьев
{
    std::vector<BinaryTree*> elements;
    bool metka;
    for (auto i : message)
    {
        metka = true;
        for (int j = 0; j < elements.size(); j++)
        {
            if (i == elements[j]->getRoot()[0])
            {
                elements[j]->setNum(elements[j]->getNum() + 1);
                metka = false;
                break;
            }
        }
    }
}

```

```

    }
    if (metka)
    {
        std::wstring wstr = L"";
        wstr += i;
        elements.push_back(new BinaryTree(wstr, 1));
    }
}
std::sort(elements.begin(), elements.end(), [](BinaryTree* i, BinaryTree* j)
{
    if (i->getNum() != j->getNum())
    {
        return i->getNum() > j->getNum();
    }
    else
    {
        return i->getRoot() < j->getRoot();
    }
});
return elements;
}
void printInfo(BinaryTree* btHuff, BinaryTree* btSF)//выписать коды символов
{
    std::wcout << L"Коды символов, закодированных методом Хаффмана:" << std::endl;
    btHuff->printCods();
    std::wcout << std::endl << L"Коды символов, закодированных методом ФаноШеннона" <<
std::endl;
    btSF->printCods();
}
void startCoding(std::wstring message, std::wstring
& cMessage1, std::wstring& cMessage2, BinaryTree** btHuff, BinaryTree** btSF)
{
    std::vector<BinaryTree*> elements1 = createBTVector(message);
    std::vector<BinaryTree*> elements2;
    for (auto i : elements1)elements2.push_back(new BinaryTree(i->getRoot(), i ->
getNum()));
    *btHuff = createBinaryTreeHuff(elements1);
    *btSF = createBinaryTreeSF(new
    BinaryTree(elements2), elements2);
    printInfo(*btHuff, *btSF);
    std::vector<std::pair<std::wstring, std::wstring>>
    vecCods1, vecCods2;
    (*btHuff)->getCods(vecCods1, L"";
    (*btSF)->getCods(vecCods2, L"";
    for (auto i : message)
    {
        for (int j = 0; j < vecCods1.size(); j++)
        {
            if (i == vecCods1[j].first[0])
            {
                cMessage1 += vecCods1[j].second;
            }
            if (i == vecCods2[j].first[0])
            {
                cMessage2 += vecCods2[j].second;
            }
        }
    }
    std::wcout << L"\nСообщение закодированное методом Хаффмана:" << cMessage1
    << std::endl;
    std::wcout << L"Длина сообщения, закодированного методом
Хаффмана:"<<cMessage1.length()<<std::endl;

```



```

        std::wcout << L"\nСообщение закодированное методом ФаноШеннона:" << cMessage2
<< std::endl;
        std::wcout << L"Длина сообщения, закодированного методом ФаноШеннона:" <<
cMessage2.length() << std::endl;
        for (auto i : elements2)
            delete i;
    }
    void frequency(std::wstring message)//вывод частот символов в строке
    {
        std::vector<BinaryTree*> elements = createBTVector(message);
        std::sort(elements.begin(), elements.end(), [](BinaryTree* i, BinaryTree* j)
        {
            if (i->getNum() != j->getNum())
                return i->getNum() > j->getNum();
            else
                return i->getRoot() < j->getRoot();
        });
        std::wcout << L"Частоты символов в строке: \n";
        for (auto i : elements)
        {
            std::wcout << i->getRoot() << L "(" << i->getNum() << L ") ";
        }
        std::wcout << L "\n";
    }
    int main()
    {
        //SetConsoleCP(1251); // установка кодовой страницы win-ср 1251 в поток ввода
        //SetConsoleOutputCP(1251); // установка кодовой страницы win-ср 1251 в поток вывода
        system("chcp 1251");
        std::locale system(""); //для кириллицы
        std::locale::global(system); //для кириллицы
        //SetConsoleCP(1251); // установка кодовой страницы win-ср 1251 в поток ввода
        //SetConsoleOutputCP(1251); // установка кодовой страницы win-ср 1251 в поток вывода
        std::wcout << L"С чем вы хотите работать (f - file, c - console, остальное - quit)"
<< '\n';
        std::wstring message;
        wchar_t c;
        std::wcin >> c;
        switch (c)
        {
            case 'f':
            {
                std::wcout << L"Введите название файла" << '\n';
                std::wstring input;
                std::wcin.ignore();
                std::getline(std::wcin, input);
                //std::wcin >> input;
                File file(input);
                file.readFile(message);
                break;
            }
            case 'c':
            {
                std::wcout << L"Введите строку" << '\n';
                std::wcin.ignore();
                std::getline(std::wcin, message);
                break;
            }
            default:
            {
                return 0;
            }
        }
        std::wcout << L"Входная строка : " << message << '\n' << '\n';
        frequency(message);
        std::wstring codeHuff, codeSF;
        BinaryTree* btHuff, * btSF;
        startCoding(message, codeHuff, codeSF, &btHuff, &btSF);
    }
}

```

```

        std::wcout << L"\nСообщение, декодированное методом
Хаффмана:"<<decoding(btHuff,codeHuff)<<std::endl;
        std::wcout << L"\nСообщение, декодированное методом
ФаноШеннона:"<<decoding(btSF,codeSF)<<std::endl;
        destroy(btHuff);
        destroy(btSF);
        return 0;
}

```