

**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: Статическое кодирование и декодирование текстового файла**  
**методами Хаффмана и Фано-Шеннона**

Студент гр. 9382

\_\_\_\_\_

Рыжих Р.В.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2020

## **ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)**

Студент Рыжих Р.В.

Группа 9382

Тема работы: Статическое кодирование и декодирование текстового файла методами Хаффмана и Фано-Шеннона. Текущий контроль.

Исходные данные: входная строка (из файла, из консоли или случайно-сгенерированная) для дальнейшего кодирования и декодирования методами Хаффмана и Фано-Шеннона и проверки пользователя (студента) на знания этих тем.

Содержание пояснительной записки: «Содержание», «Введение», «Основные теоретические сведения», «Описание алгоритмов», «Описание структур данных и используемых функций», «Текущий контроль», «Тестирование», «Заключение», «Исходный код программы»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 20.12.2020

Дата защиты реферата: 20.12.2020

Студент

\_\_\_\_\_

Рыжих Р.В.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

## **АННОТАЦИЯ**

В данной курсовой работе была реализована программа, которая выполняет статическое кодирование и декодирование Хаффмана и Фано-Шеннона. Текущий контроль включает в себя вывод заданий по переданному тексту и проверке студента на знание кодирования и декодирования методами Хаффмана и Фано-Шеннона. Реализованную программу можно использовать в обучении для проверки знаний студентов.

## **SUMMARY**

In this course work, a program was implemented that performs static encoding and decoding of Huffman and Fano-Shannon. The current control includes the output of assignments according to the transmitted text and the student's verification of knowledge of encoding and decoding using the Huffman and Fano-Shannon methods. The implemented program can be used in teaching to test the knowledge of students.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	5
Основные теоретические положения .....	6
Описание алгоритмов.....	9
Описание структур данных и используемых функций.....	11
Структуры данных для реализации деревьев Фано-Шеннона и Хаффмана:.....	11
Используемые функции .....	11
Описание интерфейса пользователя .....	14
ТЕКУЩИЙ КОНТРОЛЬ .....	15
Тестирование.....	17
Заключение.....	18
Список использованных источников.....	19
ПРИЛОЖЕНИЕ А .....	20
ИСХОДНЫЙ КОД ПРОГРАММЫ.....	20

## ВВЕДЕНИЕ

В данной курсовой работе были реализованы статическое кодирование и декодирование методами Хаффмана и Фано-Шеннона, а также реализован текущий контроль студентов.

Цель:

Целью данной работы является реализация работы статического кодирования и декодирования Хаффмана и Фано-Шеннона, а также реализация текущего контроля студентов.

Для реализации данной цели нужно решить следующие задачи:

- Собрать теоретические сведения по изучаемым алгоритмам, структуре данных и анализируемым функциям
- На основе теоретических данных написать программу на языке C++, реализующую заданную структуру данных и необходимые алгоритмы
- Реализовать текущий контроль студентов, для определения их понимания данной темы.

## 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.

## Кодировочная таблица ASCII

двоичный код	символ	двоичный код	символ	двоичный код	символ	двоичный код	символ
000 0000	[NUL]	010 0000	space	100 0000	@	110 0000	`
000 0001	[SOH]	010 0001	!	100 0001	A	110 0001	a
000 0010	[STX]	010 0010	"	100 0010	B	110 0010	b
000 0011	[ETX]	010 0011	#	100 0011	C	110 0011	c
000 0100	[EOT]	010 0100	\$	100 0100	D	110 0100	d
000 0101	[ENQ]	010 0101	%	100 0101	E	110 0101	e
000 0110	[ACK]	010 0110	&	100 0110	F	110 0110	f
000 0111	[BEL]	010 0111	'	100 0111	G	110 0111	g
000 1000	[BS]	010 1000	(	100 1000	H	110 1000	h
000 1001	[TAB]	010 1001	)	100 1001	I	110 1001	i
000 1010	[LF]	010 1010	*	100 1010	J	110 1010	j
000 1011	[VT]	010 1011	+	100 1011	K	110 1011	k
000 1100	[FF]	010 1100	,	100 1100	L	110 1100	l
000 1101	[CR]	010 1101	-	100 1101	M	110 1101	m
000 1110	[SO]	010 1110	.	100 1110	N	110 1110	n
000 1111	[SI]	010 1111	/	100 1111	O	110 1111	o
001 0000	[DLE]	011 0000	0	101 0000	P	111 0000	p
001 0001	[DC1]	011 0001	1	101 0001	Q	111 0001	q
001 0010	[DC2]	011 0010	2	101 0010	R	111 0010	r
001 0011	[DC3]	011 0011	3	101 0011	S	111 0011	s
001 0100	[DC4]	011 0100	4	101 0100	T	111 0100	t
001 0101	[NAK]	011 0101	5	101 0101	U	111 0101	u
001 0110	[SYN]	011 0110	6	101 0110	V	111 0110	v
001 0111	[ETB]	011 0111	7	101 0111	W	111 0111	w
001 1000	[CAN]	011 1000	8	101 1000	X	111 1000	x
001 1001	[EM]	011 1001	9	101 1001	Y	111 1001	y
001 1010	[SUB]	011 1010	:	101 1010	Z	111 1010	z
001 1011	[ESC]	011 1011	;	101 1011	[	111 1011	{
001 1100	[FS]	011 1100	<	101 1100	\	111 1100	
001 1101	[GS]	011 1101	=	101 1101	]	111 1101	}
001 1110	[RS]	011 1110	>	101 1110	^	111 1110	~
001 1111	[US]	011 1111	?	101 1111	_	111 1111	[DEL]

Рисунок 1 — Таблица ASCII



## Описание алгоритмов

### 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 — в левое.
4. Когда процесс достигнет листа, вместо кода символа подставляется сам символ.

## Описание структур данных и используемых функций

### Структуры данных для реализации деревьев Фано-Шеннона и Хаффмана:

Для реализации деревьев Хаффмана и Фано-Шеннона был создан класс `BinaryTree`. В нем определены следующие поля:

- `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 printCods();`//печатает на экран коды символов
- `void getCods(std::vector<std::pair<std::wstring, std::wstring>>& vec, std::wstring str);`//заносит коды символов в вектор
- `Elem getCods();`//возвращает строку, содержащую все коды символов
- `Elem root;`//возвращает элемент, лежащий в узле бинарного дерева
- `BinaryTree* left = nullptr;`//левое поддерево
- `BinaryTree* right = nullptr;`//правое поддерево
- `int num;`//вес узла

### Используемые функции

*main()* — в данной функции происходит считывание строки из файла, из консоли или генерация случайной строки (по выбору пользователя), а также вывод на консоль вопросов пользователю и запись правильных ответов в файл `answers.txt`.

*void destroy(BinaryTree\* tree)* – рекурсивно освобождает память, выделенную под бинарное дерево.

*BinaryTree\* createBinaryTreeHuff(std::vector<BinaryTree\*>&elements)* – строит бинарное дерево Хаффмана. На вход 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.

*BinaryTree\* createBinaryTreeSF(BinaryTree\* tree, std::vector<BinaryTree\*> elements)* – 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.

*wstring decoding(BinaryTree\* bt, wstring message)* – предназначена для декодирования сообщения. С помощью бинарного дерева эта функция восстанавливает переданное сообщение.

*void readFile(std::wstring& message)* – читает 1 строку из файла.

*vector<BinaryTree\*> createBTVector(std::wstring message)* – возвращает массив деревьев, состоящих из одного элемента, с заполненными частотами.

*void printInfo(BinaryTree\* btHuff, BinaryTree\* btSF)* – выводит коды символов, закодированных методами Хаффмана и Фано-Шеннона.

*void startCoding(std::wstring message, std::wstring& cMessage1, std::wstring& cMessage2, BinaryTree\*\* btHuff, BinaryTree\*\* btSF)* – начинает процесс кодировки текста.

*wstring frequency(std::wstring message)* – выводит частоту символов в строке и возвращает её

*wstring randomString()* – генерирует случайную строку

*int wstrcmp(const std::wstring str1, const std::wstring str2)* – сравнивает 2 строки

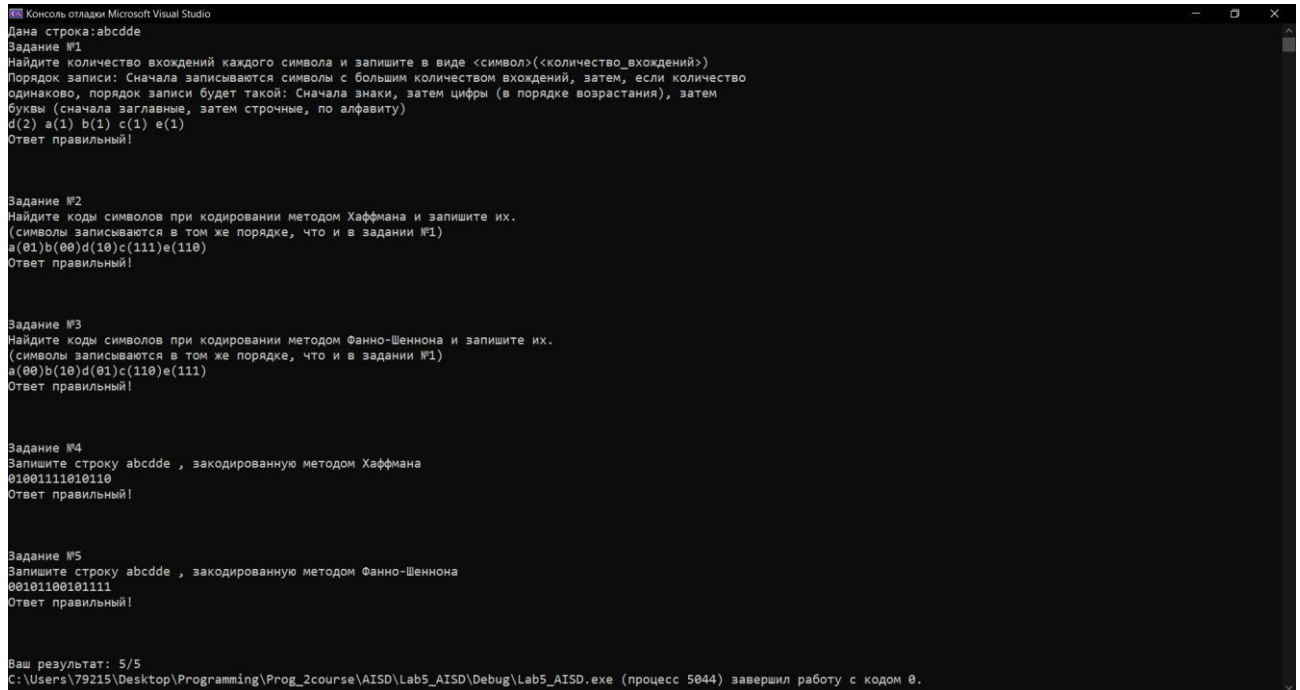
*void questions(std::wstring str, std::wstring\* answers)* – выполняет текущий контроль: выводит вопросы, считывает ответы и сверяет их с правильными.

## **Описание интерфейса пользователя**

В начале работы программа узнает у пользователя, откуда считывать текст: 'f' – из файла, 'c' – консоль, 'r' – генерация рандомной строки, всё остальное – выход из программы. Затем программа выполняет алгоритмы кодирования и декодирования, очищает экран и выводит пользователю вопросы, на которые он должен ответить. В случае правильного ответа, на экран будет выводиться строка «Ответ правильный!», в ином случае, будет строка «Ответ неправильный :с».

## ТЕКУЩИЙ КОНТРОЛЬ

В текущем контроле задаются заранее подготовленные вопросы, которые относятся к переданной программе строке.



```
Консоль отладки Microsoft Visual Studio
Дана строка:abcdde
Задание №1
Найдите количество вхождений каждого символа и запишите в виде <символ>(<количество_вхождений>)
Порядок записи: Сначала записываются символы с большим количеством вхождений, затем, если количество
одинаково, порядок записи будет такой: Сначала знаки, затем цифры (в порядке возрастания), затем
буквы (сначала заглавные, затем строчные, по алфавиту)
d(2) a(1) b(1) c(1) e(1)
Ответ правильный!

Задание №2
Найдите коды символов при кодировании методом Хаффмана и запишите их.
(символы записываются в том же порядке, что и в задании №1)
a(01)b(00)d(10)c(111)e(110)
Ответ правильный!

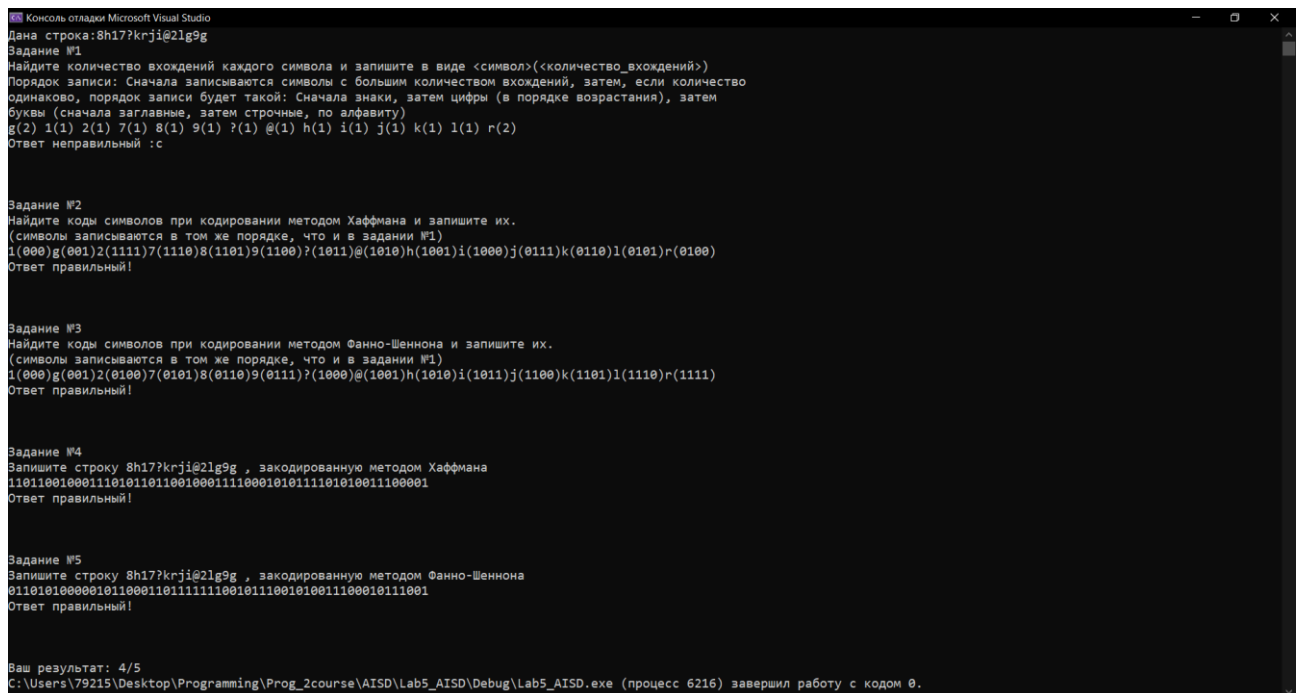
Задание №3
Найдите коды символов при кодировании методом Фанно-Шеннона и запишите их.
(символы записываются в том же порядке, что и в задании №1)
a(00)b(10)d(01)c(110)e(111)
Ответ правильный!

Задание №4
Запишите строку abcdde , закодированную методом Хаффмана
01001111010110
Ответ правильный!

Задание №5
Запишите строку abcdde , закодированную методом Фанно-Шеннона
00101100101111
Ответ правильный!

Ваш результат: 5/5
C:\Users\79215\Desktop\Programming\Prog_2course\AISD\Lab5_AISD\Debug\Lab5_AISD.exe (процесс 5044) завершил работу с кодом 0.
```

Рисунок 2 - Пример работы с входными данными № 1



```
Консоль отладки Microsoft Visual Studio
Дана строка:8h17?krji@2lg9g
Задание №1
Найдите количество вхождений каждого символа и запишите в виде <символ>(<количество_вхождений>)
Порядок записи: Сначала записываются символы с большим количеством вхождений, затем, если количество
одинаково, порядок записи будет такой: Сначала знаки, затем цифры (в порядке возрастания), затем
буквы (сначала заглавные, затем строчные, по алфавиту)
g(2) 1(1) 2(1) 7(1) 8(1) 9(1) ?(1) @(1) h(1) i(1) j(1) k(1) l(1) r(2)
Ответ неправильный :c

Задание №2
Найдите коды символов при кодировании методом Хаффмана и запишите их.
(символы записываются в том же порядке, что и в задании №1)
1(000)g(001)2(1111)7(1110)8(1101)9(1100)?(1011)@(1010)h(1001)i(1000)j(0111)k(0110)l(0101)r(0100)
Ответ правильный!

Задание №3
Найдите коды символов при кодировании методом Фанно-Шеннона и запишите их.
(символы записываются в том же порядке, что и в задании №1)
1(000)g(001)2(0100)7(0101)8(0110)9(0111)?(1000)@(1001)h(1010)i(1011)j(1100)k(1101)l(1110)r(1111)
Ответ правильный!

Задание №4
Запишите строку 8h17?krji@2lg9g , закодированную методом Хаффмана
11011001000111010110110010001111000101011110101001110001
Ответ правильный!

Задание №5
Запишите строку 8h17?krji@2lg9g , закодированную методом Фанно-Шеннона
01101010000010110001101111110010111001010011100010111001
Ответ правильный!

Ваш результат: 4/5
C:\Users\79215\Desktop\Programming\Prog_2course\AISD\Lab5_AISD\Debug\Lab5_AISD.exe (процесс 6216) завершил работу с кодом 0.
```

Рисунок 3 - Пример работы с входными данными № 2

```
Консоль отладки Microsoft Visual Studio
Дана строка:ab
Задание №1
Найдите количество вхождений каждого символа и запишите в виде <символ>(<количество_вхождений>)
Порядок записи: Сначала записываются символы с большим количеством вхождений, затем, если количество
одинаково, порядок записи будет такой: Сначала знаки, затем цифры (в порядке возрастания), затем
буквы (сначала заглавные, затем строчные, по алфавиту)
a
Ответ неправильный :c

Задание №2
Найдите коды символов при кодировании методом Хаффмана и запишите их.
(символы записываются в том же порядке, что и в задании №1)
a
Ответ неправильный :c

Задание №3
Найдите коды символов при кодировании методом Фанно-Шеннона и запишите их.
(символы записываются в том же порядке, что и в задании №1)
a
Ответ неправильный :c

Задание №4
Запишите строку ab , закодированную методом Хаффмана
a
Ответ неправильный :c

Задание №5
Запишите строку ab , закодированную методом Фанно-Шеннона
a
Ответ неправильный :c

Ваш результат: 0/5
C:\Users\79215\Desktop\Programming\Prog_2course\AISD\Lab5_AISD\Debug\Lab5_AISD.exe (процесс 3064) завершил работу с кодом 0.
```

Рисунок 4 - Пример работы с входными данными № 3



# Тестирование

```
Консоль отладки Microsoft Visual Studio
Дана строка:aaaabbbccsd
Задание №1
Найдите количество вхождений каждого символа и запишите в виде <символ>(<количество_вхождений>)
Порядок записи: Сначала записываются символы с большим количеством вхождений, затем, если количество
одинаково, порядок записи будет такой: Сначала знаки, затем цифры (в порядке возрастания), затем
буквы (сначала заглавные, затем строчные, по алфавиту)
a(4) b(3) c(2) d(1)
Ответ правильный!

Задание №2
Найдите коды символов при кодировании методом Хаффмана и запишите их.
(символы записываются в том же порядке, что и в задании №1)
a(0)b(10)c(111)d(110)
Ответ правильный!

Задание №3
Найдите коды символов при кодировании методом Фанно-Шеннона и запишите их.
(символы записываются в том же порядке, что и в задании №1)
a(0)b(10)c(111)d(110)
Ответ правильный!

Задание №4
Запишите строку aaaabbbccsd , закодированную методом Хаффмана
0000101010111111110
Ответ правильный!

Задание №5
Запишите строку aaaabbbccsd , закодированную методом Фанно-Шеннона
0000101010111111110
Ответ правильный!

Ваш результат: 5/5
C:\Users\79215\Desktop\Programming\Prog_2course\AISD\Lab5_AISD\Debug\Lab5_AISD.exe (процесс 3420) завершил работу с кодом 0.
```

Рисунок 5 - Пример работы с входными данными № 3  
Входные данные: aaaabbbccsd

```
Консоль отладки Microsoft Visual Studio
Дана строка:Прыгнуть со скалы
Задание №1
Найдите количество вхождений каждого символа и запишите в виде <символ>(<количество_вхождений>)
Порядок записи: Сначала записываются символы с большим количеством вхождений, затем, если количество
одинаково, порядок записи будет такой: Сначала знаки, затем цифры (в порядке возрастания), затем
буквы (сначала заглавные, затем строчные, по алфавиту)
(2) c(2) y(2) П(1) a(1) r(1) k(1) л(1) н(1) o(1) p(1) т(1) y(1) ь(1)
Ответ правильный!

Задание №2
Найдите коды символов при кодировании методом Хаффмана и запишите их.
(символы записываются в том же порядке, что и в задании №1)
(001)c(000)П(1110)а(1101)г(1100)к(1011)л(1010)н(1001)о(1000)р(0111)т(0110)у(0101)ь(1111)ь(0100)
Ответ правильный!

Задание №3
Найдите коды символов при кодировании методом Фанно-Шеннона и запишите их.
(символы записываются в том же порядке, что и в задании №1)
(000)c(001)ь(010)П(0110)а(0111)г(1000)к(1001)л(1010)н(1011)о(1100)п(1101)т(1110)у(11110)ь(11111)
Ответ правильный!

Задание №4
Запишите строку Прыгнуть со скалы , закодированную методом Хаффмана
111001111111110010010010010000010001000001000101110110101111
Ответ правильный!

Задание №5
Запишите строку Прыгнуть со скалы , закодированную методом Фанно-Шеннона
01101101010100010111110110111000001100000010010111011010010
Ответ правильный!

Ваш результат: 5/5
C:\Users\79215\Desktop\Programming\Prog_2course\AISD\Lab5_AISD\Debug\Lab5_AISD.exe (процесс 14080) завершил работу с кодом 0.
```

Рисунок 7 - Пример работы с входными данными № 4  
Входные данные: Прыгнуть со скалы

## **Заключение**

Были изучены алгоритмы статического кодирования и декодирования Хаффмана и Шеннона-Фано. Была разработана программа для построения бинарных деревьев Хаффмана и Шеннона-Фано на языке C++. Был представлен текущий контроль.

## Список использованных источников

1. Описание электронного ресурса // Википедия. URL: [Алгоритм Шеннона — Фано — Википедия \(wikipedia.org\)](#) (дата обращения: 5.12.2020)
2. Описание электронного ресурса // Википедия. URL: [Код Хаффмана — Википедия \(wikipedia.org\)](#). (дата обращения: 5.12.2020)
3. Описание электронного ресурса // Habr, URL: [Алгоритм Хаффмана на пальцах / Хабр \(habr.com\)](#)

## 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);
    Elem getCods();
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 & e1, T & e2) //меняет элементы местами
{
    T temp = e1;
    e1 = e2;
    e2 = 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 rootBTLeft = L""; //строка для корня левого поддерева
    std::wstring rootBTRight = L""; //строка для корня правого поддерева
    int weightBTLeft = 0; //вес для корня левого поддерева
    int weightBTRight = 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 (weightBTRight <= tree->getNum() / 2)
                {
                    rootBTRight += j->getRoot();
                    weightBTRight += j->getNum();
                    lastWeight = j->getNum();
                    lastSymb = j->getRoot();
                }
                else
                {
                    rootBTLeft += j->getRoot();

```

```

        weightBTLeft += 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';
}

```

```

}

Elem BinaryTree::getCods()
{
    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")";
    }
    str += L'\n';
    return str;
}

```

### decoding.h:

```

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

```

### decoding.cpp:

```

#include "decoding.h"
#include <iostream>
std::wstring decoding(BinaryTree * bt, std::wstring message)
//функция декодирования сообщения
{
    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 поддереву, 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 поддереву, 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 сообщение, закодированного методом ФаноШеннона:" <<
cMessage2.length() << std::endl;
for (auto i : elements2)
    delete i;
}

std::wstring frequency(std::wstring message)//Вывод частот символов в строке
{
    std::wstring answer = L "";
    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)
    {
        answer += i->getRoot() + L "(" + std::to_wstring(i->getNum()) + L ") ";
        std::wcout << i->getRoot() << L "(" << i->getNum() << L ") ";
    }
    std::wcout << L "\n";
    return answer;
}

std::wstring randomString()
{
    std::wstring str = L "";
    int x;
    int len = 15;
    char a[45] =
    {
        'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', '[', ']', 'a', 's', 'd',
        'f', 'g', 'h', 'j', 'k', 'l', 'z', 'x', 'c', 'v', 'b', 'n', 'm', '!', '@',
        '#', '$', '%', '&', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '?'
    };
    for (int i = 0; i < len; i++)

```

```

    {
        x = rand() % 45;
        str += a[x];
    }
    return str;
}

int wstrcmp(const std::wstring str1, const std::wstring str2) {
    std::wstring str1 = str1;
    std::wstring str2 = str2;
    for (int i = 0; i < str1.length(); i++) {
        if (str1[i] == ' ' || str1[i] == '\n' || str1[i] == '\0') {
            str1.erase(i, 1);
            i--;
        }
    }
    for (int i = 0; i < str2.length(); i++) {
        if (str2[i] == ' ' || str2[i] == '\n' || str2[i] == '\0') {
            str2.erase(i, 1);
            i--;
        }
    }
    if (!str1.compare(str2)) {
        std::wcout << L"Ответ правильный!\n";
        return 1;
    }
    else {
        std::wcout << L"Ответ неправильный :c\n";
        return 0;
    }
}

void questions(std::wstring str, std::wstring* answers) {
    int count = 0;
    std::wcout << L"Дана строка:" << str << std::endl;
    std::wcout << L"Задание №1\n";
    std::wcout << L"Найдите количество вхождений каждого символа и запишите в виде  

<символ>(количество_вхождений)\n";
    std::wcout << L"Порядок записи: Сначала записываются символы с большим количеством  

вхождений, затем, если количество\n";
    std::wcout << L"одинаково, порядок записи будет такой: Сначала знаки, затем цифры (в  

порядке возрастания), затем\n";
    std::wcout << L"буквы (сначала заглавные, затем строчные, по алфавиту)\n";
    std::wstring answ;
    std::getline(std::wcin, answ);
    count += wstrcmp(answ, answers[0]);
    std::wcout << L"\n\nЗадание №2\n";
    std::wcout << L"Найдите коды символов при кодировании методом Хаффмана и запишите  

их.\n";
    std::wcout << L"(символы записываются в том же порядке, что и в задании №1)\n";
    std::getline(std::wcin, answ);
    count += wstrcmp(answ, answers[1]);
    std::wcout << L"\n\nЗадание №3\n";
    std::wcout << L"Найдите коды символов при кодировании методом Фанно-Шеннона и  

запишите их.\n";
    std::wcout << L"(символы записываются в том же порядке, что и в задании №1)\n";
    std::getline(std::wcin, answ);
    count += wstrcmp(answ, answers[2]);
    std::wcout << L"\n\nЗадание №4\n";
    std::wcout << L"Запишите строку " << str << L" , закодированную методом Хаффмана\n";
    std::getline(std::wcin, answ);
    count += wstrcmp(answ, answers[3]);
    std::wcout << L"\n\nЗадание №5\n";
}

```

```

        std::wcout << L"Запишите строку " << str << L" , закодированную методом Фанно-
Шеннона\n";
        std::getline(std::wcin, answ);
        count += wstrcmp(answ, answers[4]);
        std::wcout << L"\n\nВаш результат: " << count << L"/5";
    }

int main()
{
    system("chcp 1251");
    std::locale system(""); //для кириллицы
    std::locale::global(system); //для кириллицы
    std::wcout << L"С чем вы хотите работать (f - file, c - console, r - random string,
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);
        File file(input);
        file.readFile(message);
        break;
    }
    case 'c':
        std::wcout << L"Введите строку" << '\n';
        std::wcin.ignore();
        std::getline(std::wcin, message);
        break;
    case 'r':
        message = randomString();
        std::wcin.ignore();
        break;
    default:
        return 0;
    }
    if (message.length() == 0)
        return 0;
    std::wcout << L"Входная строка : " << message << '\n' << '\n';
    std::wstring answer;
    answer = frequency(message);
    answer += L'\n';
    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;
    std::system("cls");
    std::wofstream file("answers.txt");
    codeHuff += L'\n';
    codeSF += L'\n';
    file << answer << btHuff->getCods() << btSF->getCods() << codeHuff << codeSF;
    file.close();
    std::wstring answers[5];
    answers[0] = answer;
    answers[1] = btHuff->getCods();

```

```
    answers[2] = btSF->getCods();  
    answers[3] = codeHuff;  
    answers[4] = codeSF;  
    questions(message, answers);  
    destroy(btHuff);  
    destroy(btSF);  
    return 0;  
}
```