# 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(ка) гр. 9381	Андрух И.А.
Преподаватель	 Фирсов М.А.

Санкт-Петербург 2020

#### ЗАДАНИЕ

#### НА КУРСОВУЮ РАБОТУ

Студентка Андрух И.А. Группа 9381 Тема работы: Вариант 2. Статическое кодирование и декодирование текстового файла методами Хаффмана и Фано-Шеннона. Текущий контроль. Исходные данные: Файл с текстом для кодирования Содержание пояснительной записки: «Содержание» «Введение» «Ход выполнения» «Тестирование» «Заключение» «Список использованных источников» Предполагаемый объем пояснительной записки: Не менее 15 страниц. Дата выдачи задания: 31.10.2020 Дата сдачи реферата: 16.12.2020 Дата защиты реферата: 17.12.2020 Студентка Андрух И.А.

Фирсов М.А.

Преподаватель

#### **АННОТАЦИЯ**

Задача курсовой работы состоит в разработке программы для генерации заданий на кодирование и декодирование текста методами Хаффмана и Фано-Шеннона. Задания предназначены для текущего контроля студентов.

Курсовая работа состоит из пояснительной записки и исходного кода программы.

В ходе работы была реализована программа с консольным интерфейсом для генерации заданий на кодирование и декодирование текстовых файлов методами Хаффмана и Фано-Шеннона. Использовался язык C++.

#### **SUMMARY**

The task of the course work is to develop a program for generating tasks for encoding and decoding text using Huffman and Fano-Shannon methods. Tasks are intended for the current control of students.

The course work consists of an explanatory note and the source code of the program.

In the course of the work, a program with a console interface was implemented for generating tasks for encoding and decoding text files using huffman and fano-shannon methods. The C++language was used.

# СОДЕРЖАНИЕ

	Введение	5
1.	Задание	6
2.	Описание алгоритма	6
3.	Описание структур данных и функций.	7
3.1	struct s_expr	7
3.2	struct binTree	7
3.3	Функции пространства имен binTree_modul	7
3.4	struct elem_code	8
3.5	struct list	8
3.6	struct haffm_codes	8
3.6.1	Поле	8
3.6.2	Методы	8
3.7	Функции для кодирования и декодирования.	9
4.	Описание интерфейса пользователя.	11
5.	Заключение	11
6.	Список литературы	12
7.	Приложение А. Тестирование.	13
	Приложение Б. исходный код программы.	43

#### введение.

Целью работы является написание программы, которая будет генерировать задания с ответами. Задания должны содержать текст для кодирования/декодирования. Кодирование должно быть реализовано по алгоритмам Хаффмана и Фано-Шеннона. В результате работы программы должен генерироваться файл с заданиями и ответами, готовый для передачи студентам.

Алгоритм кодирования Хаффмана: на вход приходят упорядоченные по невозрастанию частот данные, выбираются две наименьших по частоте буквы алфавита, создается родитель(сумма частот этих «листков»). Потомки удаляются, вместо них записывается родитель, «ветви» родителя нумеруются «0» и «1». Процесс повторяется, пока не будет найден главный родитель – корень.

Алгоритм кодирования Фано-Шеннона: на вход приходят упорядоченные по невозрастанию частот данные, находится середина, которая делит алфавит примерно на две части(примерно равные по сумме частот символов), частям присваевается «0» и «1». Процесс повторяется, пока не будет получен листок.

#### 1. ЗАДАНИЕ.

Реализовать алгоритмы статического кодирования и декодирования текстового файла методами Хаффмана и Фано-Шеннона, генерируя с их помощью задания с ответами для текущего контроля студентов.

#### 2. ОПИСАНИЕ АЛГОРИТМА.

При помощи консольного меню программа запрашивает у пользователя тип генерируемого задания и количество вариантов. После этого берётся строка из заданного изначально массива (преподаватель может менять его, дополняя новыми предложениями) и передается сначала в функцию для добавления символа и частоты его встречаемости в вектор, потом в функцию для сортировки вектора символов, и в функцию кодирования, где происходит создание бинарного дерева одним из двух алгоритмов и сам процесс кодирования. Промежуточные данные, полученные в ходе сортировки, построения дерева и кодирования строки, выводятся в консоль. Таким образом генерируется задание на кодирование строки. Далее генерируется задание на декодирование - выводится закодированная строка и таблица символов(она создается при помощи созданного ранее вектора). Функция декодирования получает закодированную строку и демонстрирует процесс декодирования в консоль, ответ также записывается в файл. Выводится консольное меню для генерации новых заданий или выхода из программы.

## 3. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ.

#### 3.1 struct s\_expr.

```
бинарного
                                                 дерева(пространство
     Структура
                  для
                         создания
                                                                         имен
binTree modul).
struct s_expr {
    bool tag; // true если Leaf, false если Inter
    union //идентификаторы полей попадают во внешнюю область
видимости
    {
      base leaf; //
                     лист дерева
       binTree inter; //звено дерева(не лист)
    } node;
  };
   3.2 struct binTree.
Структура бинарного дерева(пространство имен binTree_modul).
struct binTree
  {
    s expr* lt; // указатель на левое поддерево
    s expr* rt; // указатель на правое поддерево
  };
  3.3 Функции пространства имен binTree_modul.
  lisp create(void); // создание пустого бинарного дерева
  lisp left(const lisp); // возвращает указатель на левое поддерево, аргумент –
элемент типа s_expr
  lisp right(const lisp); // возвращает указатель на правое поддерево, аргумент
- элемент типа s_expr
  lisp consBT(const lisp lt, const lisp rt); // создание дерева из двух
поддеревьев, аргументы –два элемента типа s_expr
  void destroy(lisp&); // уничтожает дерево, аргумент –адрес элемента типа
s_expr
```

bool isLeaf(const lisp s); // проверяет, является ли элемент листом, аргумент – элемент типа s\_expr

lisp makeLeaf(const base x); // создает лист со значением x, аргумент – переменная шаблонного типа.

#### 3.4 struct elem\_code.

```
Структура для хранения символа и соответствующего ему кода.
```

```
struct elem_code {
    char symbol;//символ
    string code;//строка для хранения кода символа
};
```

#### 3.5 struct list.

Структура для хранения символа и количества повторений этого символа в строке.

```
struct list {
    char element; //символ
    int count;//количество повторений символа в предложении
    lisp bt;//элемент типа s_expr, хранит информацию о том, является ли
элемент листом
```

#### 3.6. struct haffm\_codes.

Структура для хранения кодов символов, полученных по Хаффману.

#### 3.6.1. Поле.

**}**;

vector<elem\_code> codes; - вектор для хранения символов и кодов;

#### 3.6.2. Методы.

char get\_symbol(string code) - декодирование символа, аргумент – строка, содержащая код символа;

string get\_code(char symbol) – получение кода символа, аргумент – символ для кодирования;

void append(char symbol, string code) — добавление символа и кода в поле codes, аргументы — символ и код символа;

string encodeString(string plain) – кодирование строки методом Хаффмана(если код для встреченного символа есть, записываем его в строку результата), аргумент – строка для кодирования;

void decode\_H(string str\_to\_decode, const lisp b, string& decoded\_str) - декодирование строки методом Хаффмана, аргументы — строка для декодирования, бинарное дерево и строка для результата соответственно;

#### 3.7. Функции для кодирования и декодирования.

- void add(vector<list>& symbols, char s) функция добавления в вектор символа и частоты его встречаемости, аргументы вектор и символ для добавления;
- void sort(vector<list>& symbols) сортировка символов вектора в порядке невозрастания частот, аргумент – ветор с символами и частотами;
- void middle(vector<list> simbols, int S, int B, int& node, int& SL, int& SR)- деление символов на две части по сумме частот для алгоритма Фано-Шеннона, аргументы вектор возможных символов, длина сообщения, начало поиска, счетчик добавленных в левую часть символов, сумма частот левой части, сумма частот правой части;
- void method\_FSH(vector<list> smbl, int lenght, int start\_of\_search, int end\_of\_search, string Code, lisp& bin\_t,int deep) создание бинарного дерева для метода Фано-Шеннона, аргументы- вектор символов, длина сообщения, начало обрабатываемого интервала, конец обрабатываемого интервала, строка для формирования кода, бинарное дерево, глубина рекурсии;
- void show\_method\_FSH(vector<list> smbl, int lenght, int start\_of\_search, int end\_of\_search, string Code, lisp& bin\_t,int deep) функция, аналогичная предыдущей, но без демонстрации промежуточных результатов;

- void final\_coding(vector<list>& smbl, char elm, string& coded\_string, int& l, string codes[]) функция кодирования элемента и записи его в результрующую строку, для метода Фано-Шеннона, аргументы вектор символов, элемент для кодирования, строка результата, длина строки результата, массив строк, хранящих коды символов;
- void close\_final\_coding(vector<list>& smbl, char elm, string& coded\_string, int& l, string codes[]) функция, аналогичная предыдущей, но без вывода промежуточных результатов;
- void method\_H(vector<list>& smbl, lisp& bin\_tree, string& code) создание бинарного дерева элементов по Хаффману, аргументы вектор символов, бинарное дерево, код символа;
- void search\_of\_min\_element(vector<list>& smbl, list& elmn) поиск элемента с минимальной частотой повторений, аргументы вектор символов, проверяемый элемент;
- void codeHuffman(string& code, vector<list>& smbl, lisp bin\_tree,int deep) -создание бинарного дерева кодов по Хаффману, аргументы –строка с уже полученным кодом, вектор символов, бинарное дерево, глубина рекурсии;
- void decode\_FSH(vector<list> smbl, string str\_to\_decode, const lisp b, string& decoded\_str) функция декодирования, аргументы вектор символов, строка для декодирования, бинарное дерево, строка для результата.

#### 4. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ.

Для взаимодействия пользователя и программы выводится консольное меню. Пользователь может выбрать тип задания(Кодирование по Хаффману и декодирование, или кодирование по Фано-Шеннону и декодирование). Далее предлагается выбрать количество генерируемых вариантов. Задания и ответы выводятся в файл. Задания, промежуточные результаты и ответы для проверки выводятся в консоль.

#### ЗАКЛЮЧЕНИЕ.

В результате выполнения курсовой работы была написана программа на языке C++ для генерации заданий на статическое кодирование методами Хаффмана и Фано-Шеннона и декодирование. Задания и ответы сохраняются в файл и готовы для передачи студентом. Массив возможных строк для обработки может обновлятся и дополняться преподавателем.

## СПИСОК ЛИТЕРАТУРЫ.

- 1. Керниган Б. И Ритчи Д. Язык программирования Си.
- 2. <a href="https://habr.com/ru/post/438512/">https://habr.com/ru/post/438512/</a>
- 3. <a href="https://habr.com/ru/post/137766/">https://habr.com/ru/post/137766/</a>

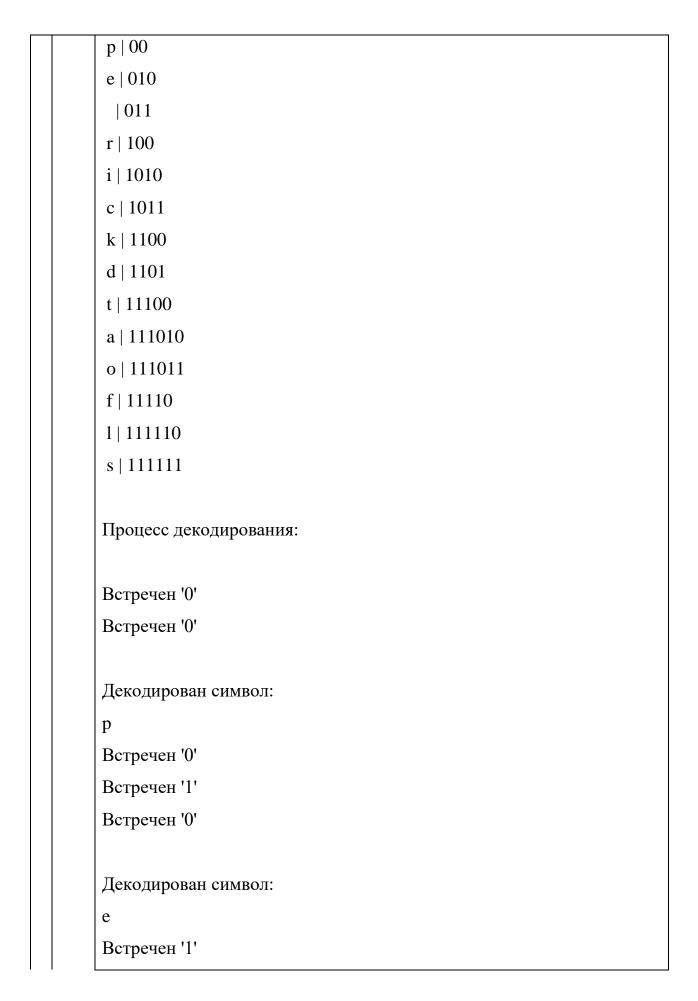
# **ПРИЛОЖЕНИЕ А ТЕСТИРОВАНИЕ.**

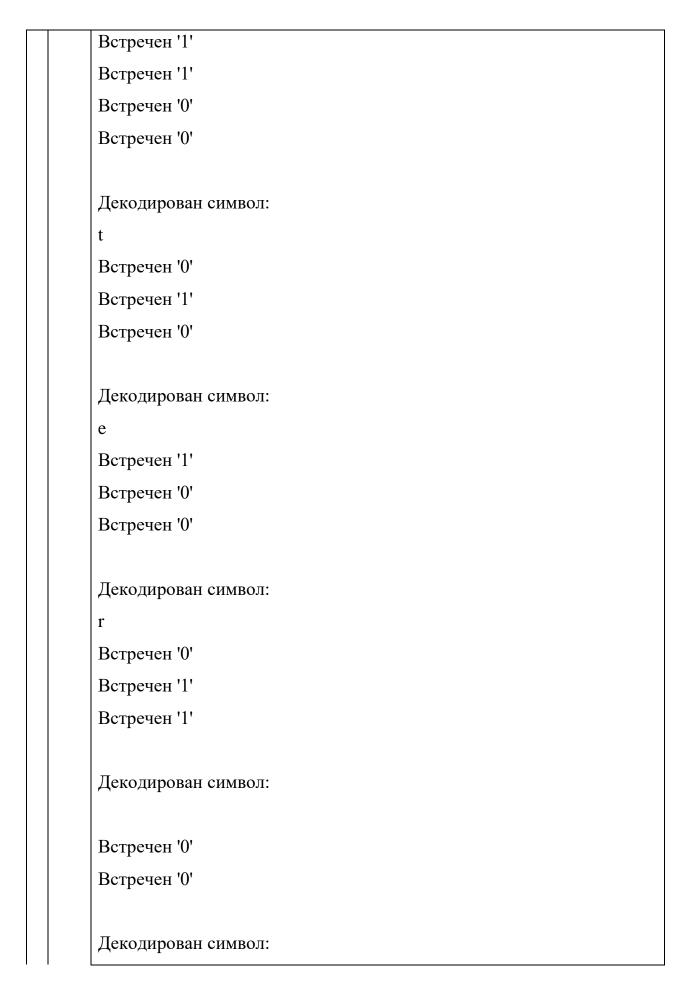
N	Bx	Результат
	од	
	ны	
	e	
	дан	
	ны	
	e	
1	1	ВАРИАНТ №1
	1	Задание 1.
		Закодируйте предложение методом Хаффмана:<<
		peter piper picked a peck of pickled peppers
		Промежуточные выводы:
		Символы отсортированы по частоте встречаемости:
		p-9
		e-8
		-7
		r-3
		i-3
		c-3
		k-3
		d-2
		t-1
		a-1
		o-1
		f-1
		1-1
		s-1

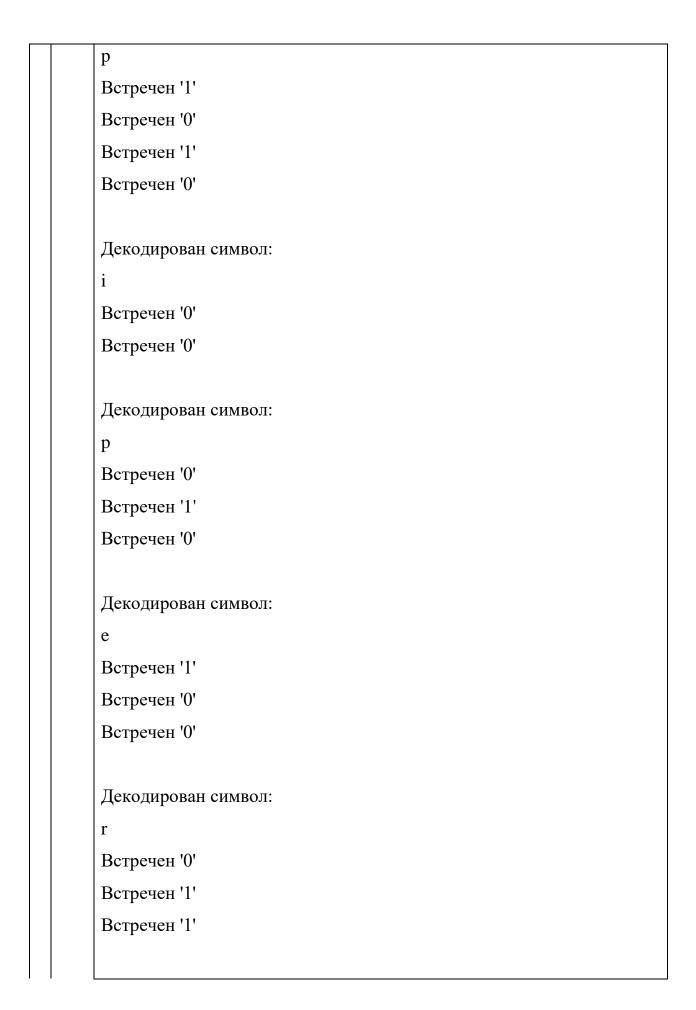
```
Построение бинарного дерева:
0
 00
  000
   0000
   0001
    00010
    00011
  001
   0010
    00100
    00101
   0011
    00110
    00111
 01
1
 10
  100
   1000
   1001
  101
   1010
   1011
 11
  110
  111
Процесс кодирования:
```

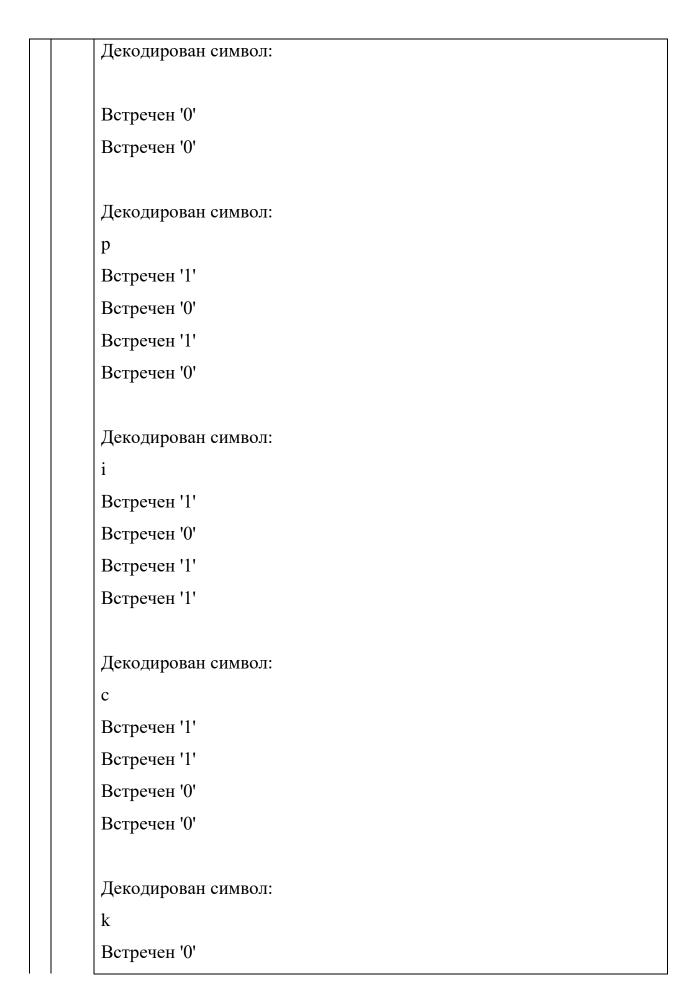
p-01
e-111
t-00010
e-111
r-1000
-110
p-01
i-1001
p-01
e-111
r-1000
-110
p-01
i-1001
c-1010
k-1011
e-111
d-0000
-110
a-00011
-110
p-01
e-111
c-1010
k-1011
-110
o-00100
f-00101

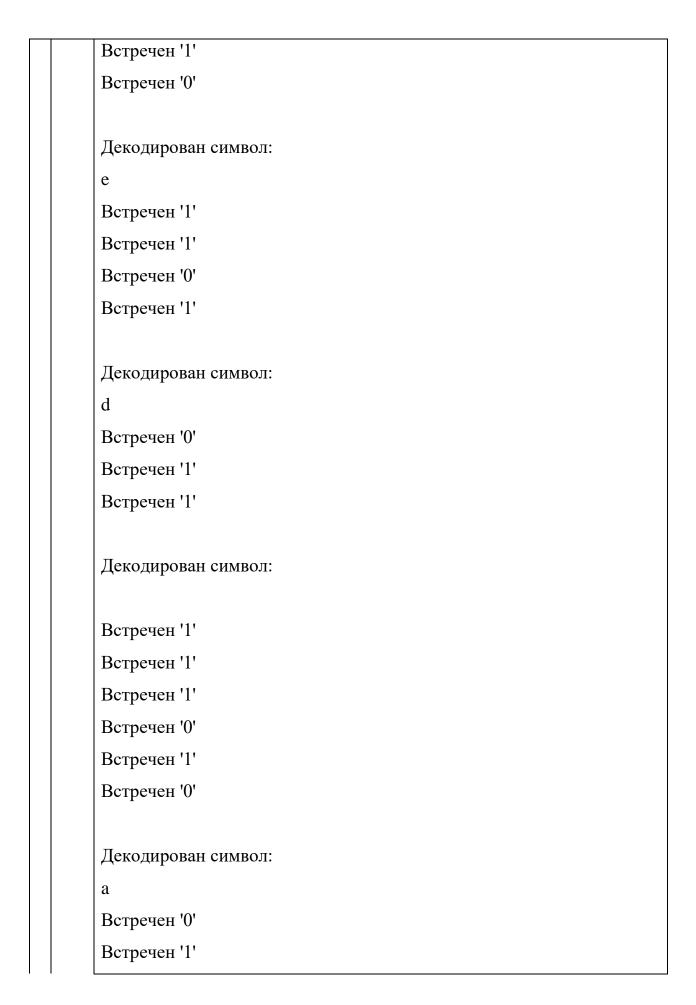
	-110
	p-01
	i-1001
	c-1010
	k-1011
	1-00110
	e-111
	d-0000
	-110
	p-01
	e-111
	p-01
	p-01
	e-111
	r-1000
	s-00111
	Ответ:
	01111000101111100011001100101111110001100110011010
	110000111100111110101011111000100001011100110011010
	11100001100111101011111100000111
	Задание 2.
	Раскодируйте строку:
	00010111000101000110010100001010001100101
	111101001100010101111100011111101111111
	00101101011000100000010100111111
	Коды символов:
1	



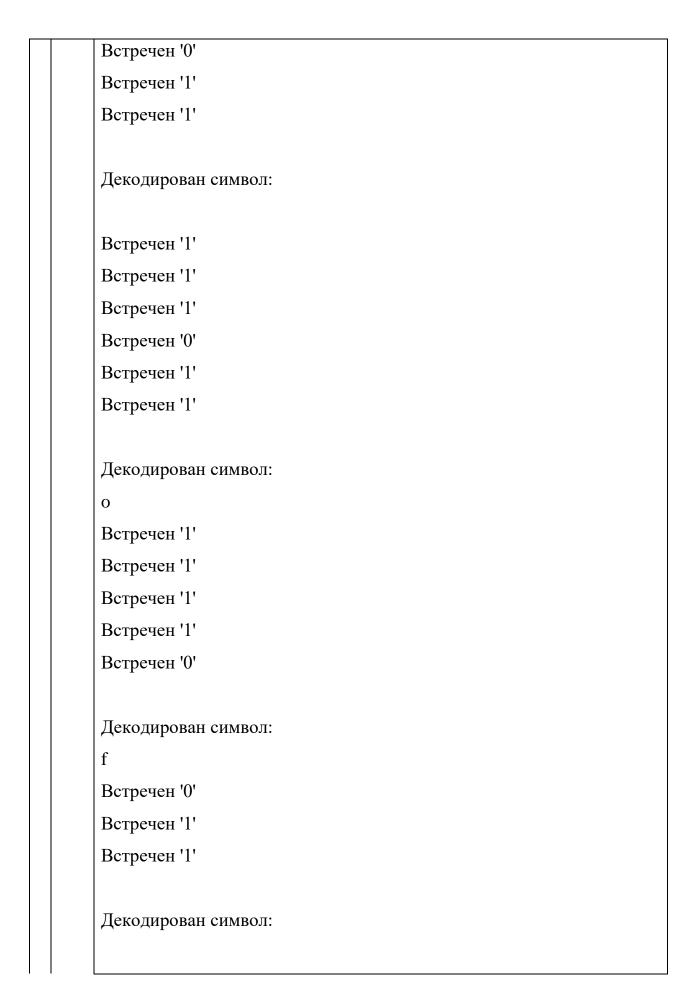


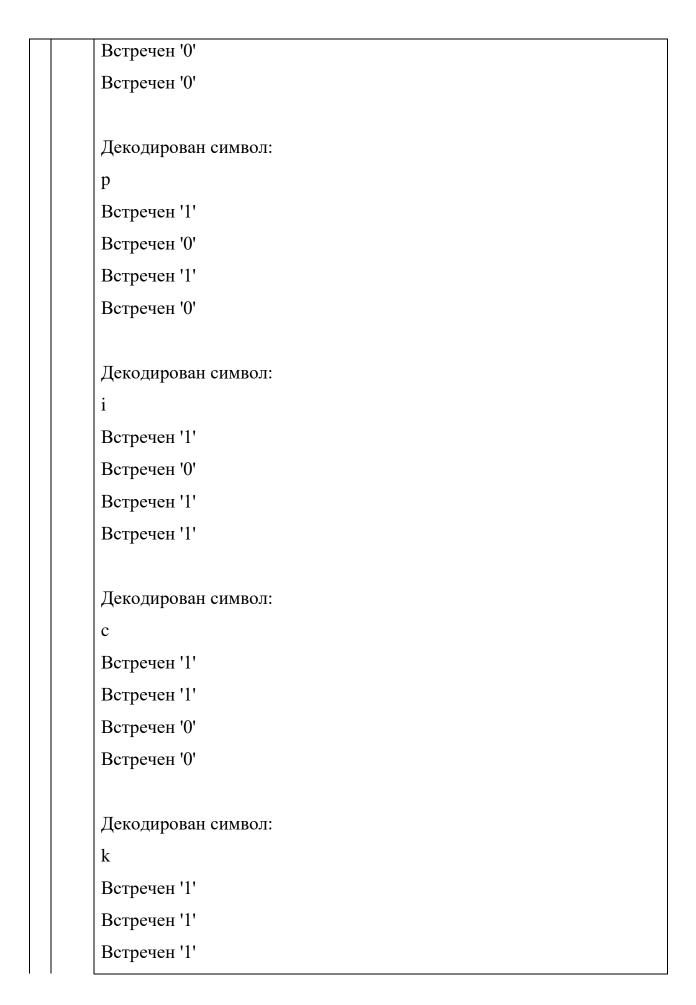


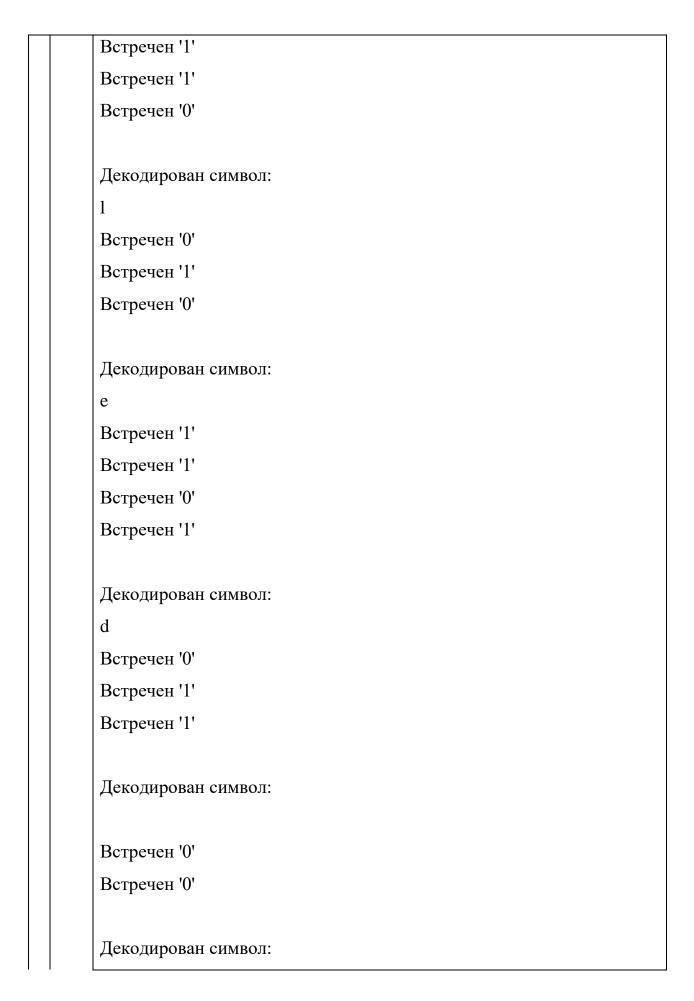


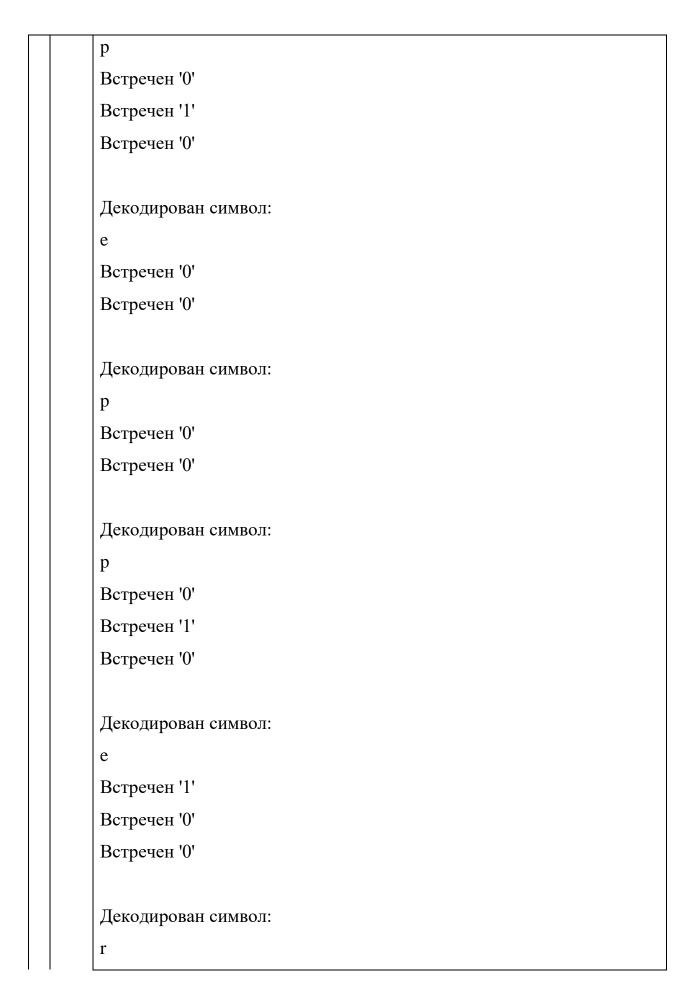


В	стречен '1'
Д	[екодирован символ:
В	встречен '0'
В	встречен '0'
Д	[екодирован символ:
p	
В	Стречен '0'
В	стречен '1'
В	стречен '0'
Д	[екодирован символ:
e	
В	стречен '1'
В	встречен '0'
В	стречен '1'
В	встречен '1'
Д	(екодирован символ:
c	
В	Стречен '1'
В	стречен '1'
В	стречен '0'
В	стречен '0'
Д	(екодирован символ:
k	









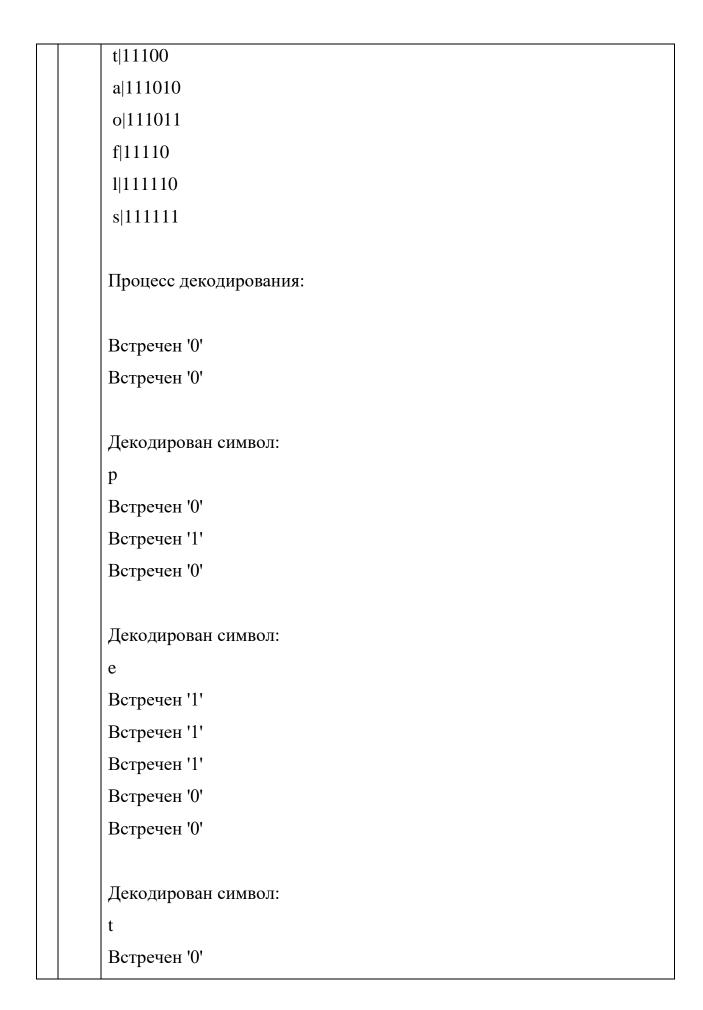
		Встречен '1'
		Встречен '1'
		Декодирован символ:
		S
		Ответ:
		peter piper picked a peck of pickled peppers
		Варианты записаны в файл fout.txt
2	2	ВАРИАНТ №1
	1	Задание 1.
		Закодируйте предложение методом Фано - Шеннона:
		peter piper picked a peck of pickled peppers
		Символы отсортированы по частоте встречаемости:
		p-9
		e-8
		-7
		r-3
		i-3
		c-3
		k-3
		d-2
		t-1
		a-1
		0-1

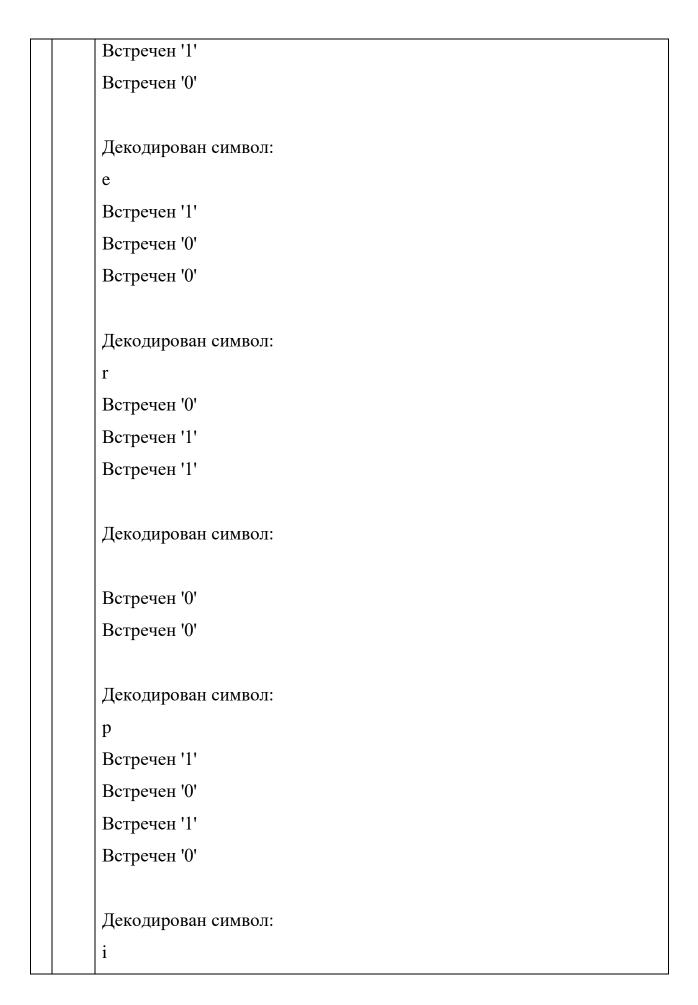
```
f-1
1-1
s-1
Построение бинарного дерева Шеннона-Фано:
0
 00
 01
  010
  011
1
 10
  100
  101
   1010
   1011
 11
  110
   1100
   1101
  111
   1110
    11100
    11101
     111010
     111011
   1111
    11110
    11111
```

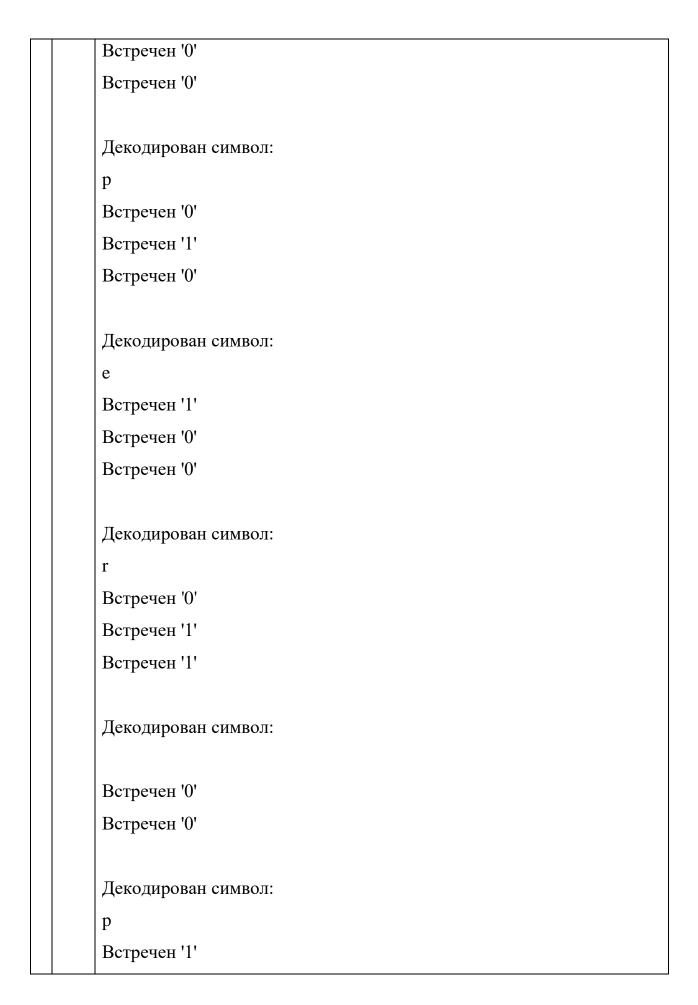
111110
111111
Коды символов:
p 00
e 010
011
r 100
i 1010
c 1011
k 1100
d 1101
t 11100
a 111010
o 111011
f 11110
1 111110
s 111111
Процесс кодирования:
символ р заменён на код 00
символ е заменён на код 010
символ t заменён на код 11100
символ е заменён на код 010
символ r заменён на код 100
символ заменён на код 011
символ р заменён на код 00
символ і заменён на код 1010

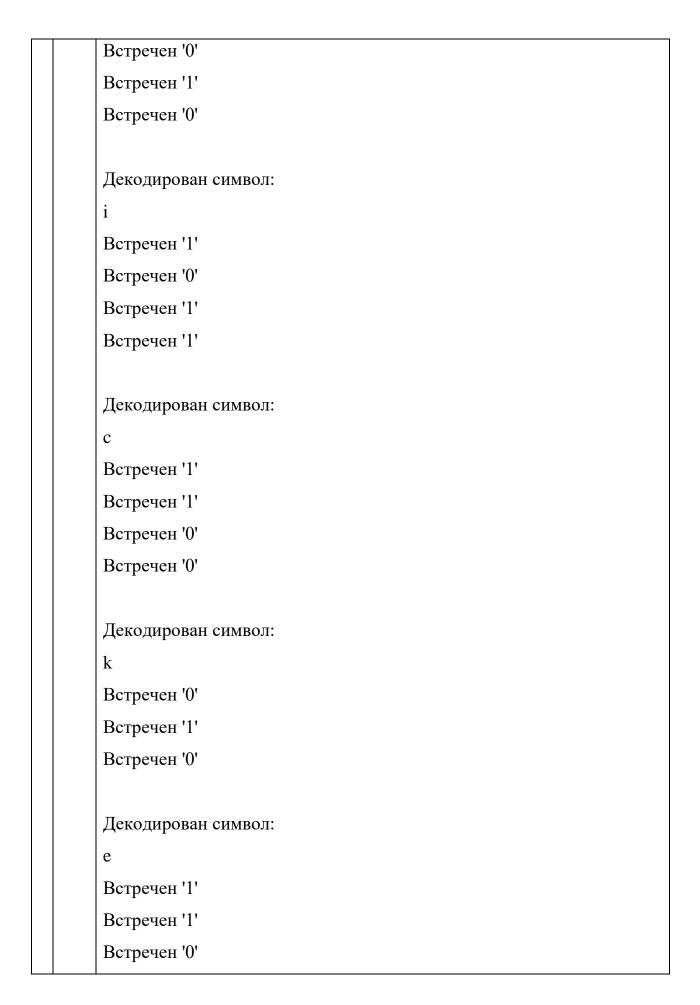
символ р заменён на код 00 символ е заменён на код 010 символ г заменён на код 100 символ заменён на код 011 символ р заменён на код 00 символ і заменён на код 1010 символ с заменён на код 1011 символ k заменён на код 1100 символ е заменён на код 010 символ d заменён на код 1101 символ заменён на код 011 символ а заменён на код 111010 символ заменён на код 011 символ р заменён на код 00 символ е заменён на код 010 символ с заменён на код 1011 символ k заменён на код 1100 символ заменён на код 011 символ о заменён на код 111011 символ f заменён на код 11110 символ заменён на код 011 символ р заменён на код 00 символ і заменён на кол 1010 символ с заменён на код 1011 символ k заменён на код 1100 символ 1 заменён на код 111110 символ е заменён на код 010 символ d заменён на код 1101 символ заменён на код 011

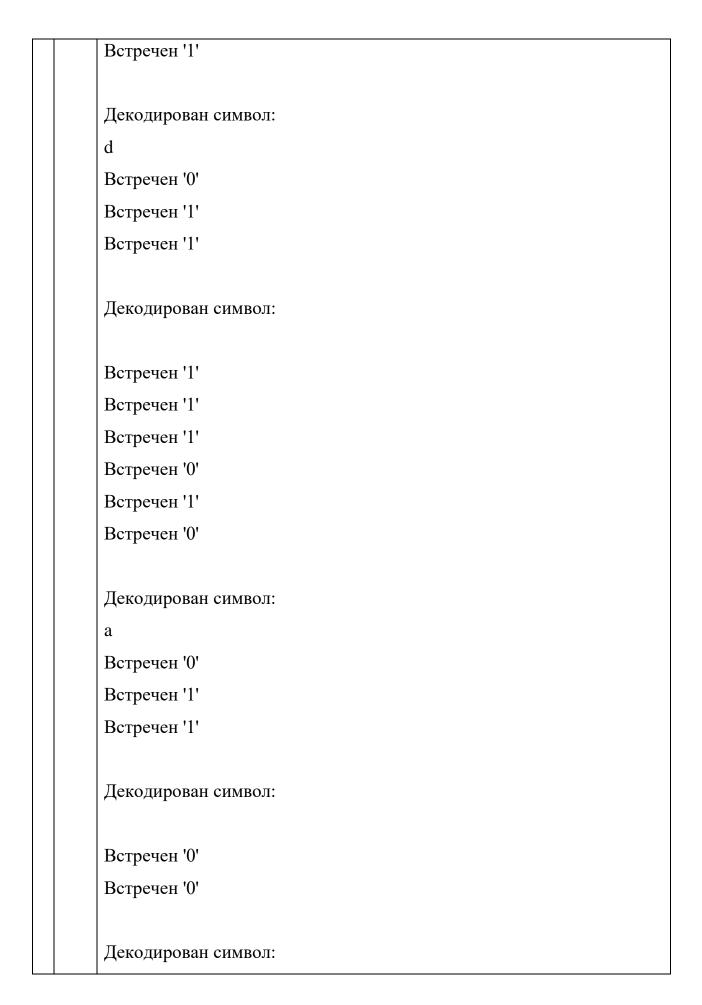
символ р заменён на код 00 символ е заменён на код 010 символ р заменён на код 00 символ р заменён на код 00 символ е заменён на код 010 символ г заменён на код 100 символ ѕ заменён на код 111111 Ответ: 00101101011000100000010100111111Задание 2. Раскодируйте строку: 00101101011000100000010100111111 Коды символов: p|00e|010 011 r|100i|1010c|1011 k|1100d|1101

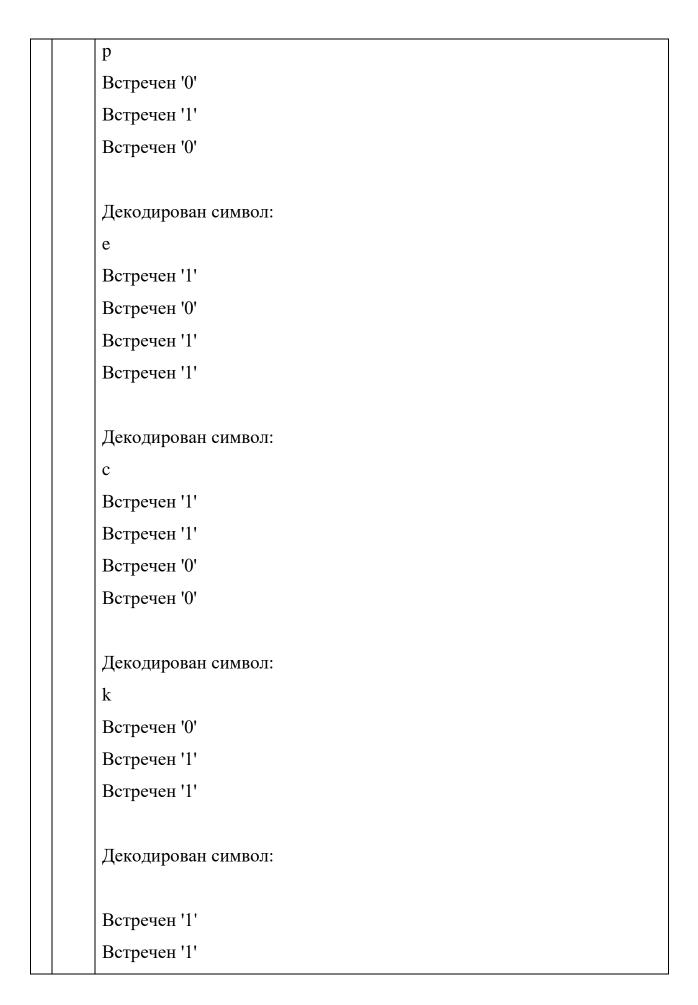


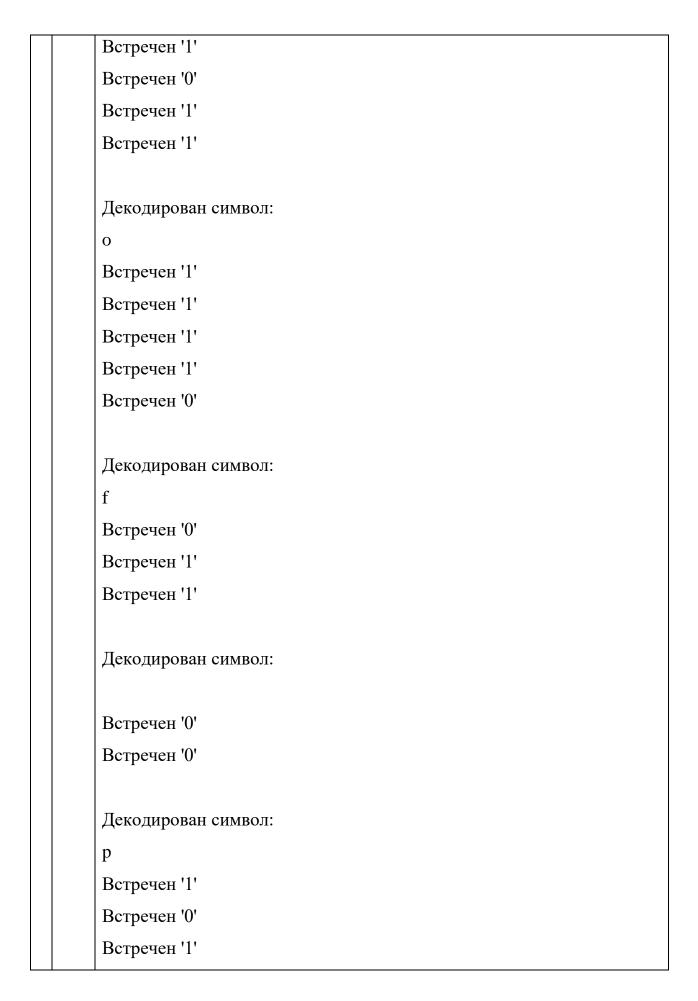


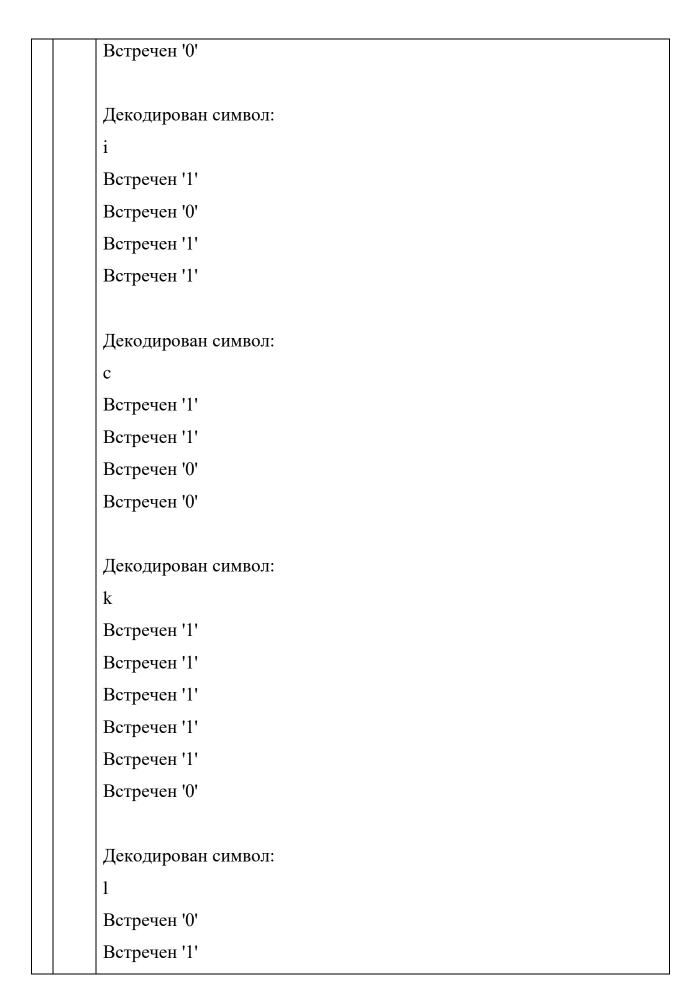


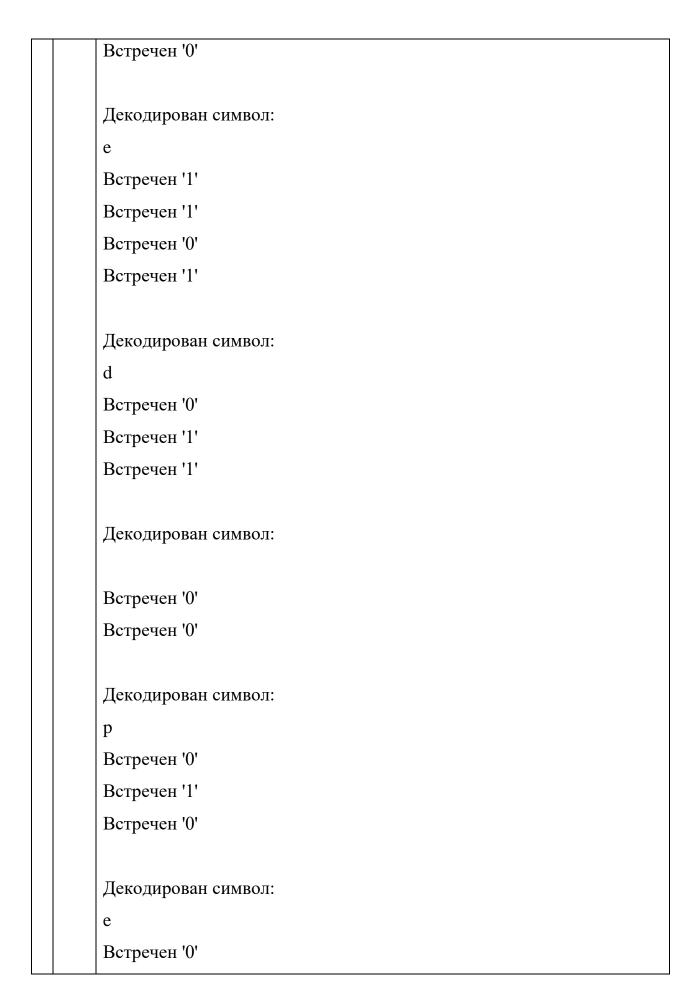


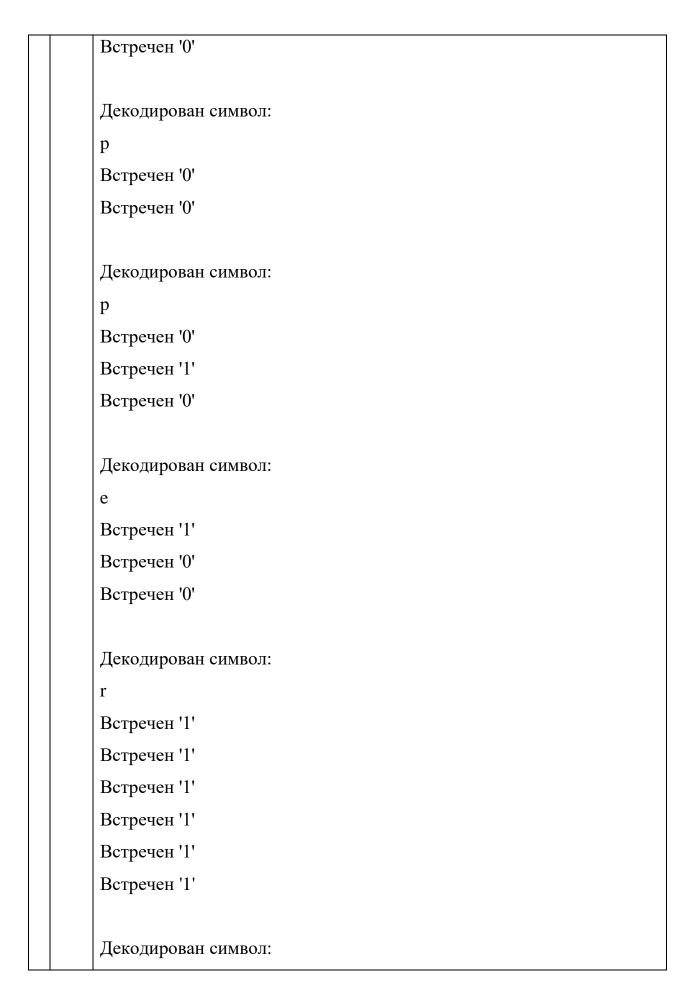












	S
	Ответ: peter piper picked a peck of pickled peppers
	Варианты записаны в файл fout.txt

## ПРИЛОЖЕНИЕ Б ИСХОДНЫЙ КОД ПРОГРАММЫ

## Файл main.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <stdlib.h>
#include <locale.h>
#include "header.h"
using namespace std;
using namespace binTree modul;
struct elem code {//структура для хранения символла и соответствующего ему
кода
    char symbol;
    string code;
} ;
struct haffm codes {
   vector<elem code> codes; //вектор для хранения кодов символов при
кодировании по Хаффману
    char get symbol(string code) {//декодирование строки
        for (vector<elem code>::iterator it = codes.begin(); it !=
codes.end(); it++) {
            if ((*it).code == code) {
                return (*it).symbol;
            }
        }
        return '\0';
    };
    string get code(char symbol) {//поиск кода для символа
        for (vector<elem code>::iterator it = codes.begin(); it !=
codes.end(); it++) {
            if ((*it).symbol == symbol) {
```

```
return (*it).code;
            }
        }
        return "";
    };
    void append(char symbol, string code) {//добавление символа и кода в
codes
        elem code new code;
        new code.code = code;
        new code.symbol = symbol;
        codes.push back(new code);
    };
    string encodeString(string plain) { //кодирование строки методом Хаффмана
        string result = "";
        string step code = "";
        cout<<"Процесс кодирования:"<<endl;
        for (int i = 0; i < plain.length(); i++) {</pre>
            step code = get code(plain[i]);
            if (step code != "") {
                                                //Если код для встреченного
символа есть, записываем его в строку результата
                result +=step code;
                cout << "\n" <<plain[i]<<"-" << step code;</pre>
            }
            else {
            }
        }
       return result;
    };
    void decode H(string str to decode, const lisp b, string&
decoded str)//декодирование строки методом Хаффмана
    {
        string curr code = "";
        decoded str = "";
        int i = 0;
```

```
while (i <= str to decode.size() - 1)
            lisp p = b;
            curr code = "";
            while (!isLeaf(p))
                if (i >= str_to_decode.size())
                    break;
                curr_code += str_to_decode[i];
                i++;
                if (curr_code[curr_code.length() - 1] == '0') p = Left(p);
//если встречен 0 спускаемся по левой ветке дерева
                else p = Right(p);//иначе - по правой
            }
            decoded_str += get_symbol(curr_code);//дойдя до листа извлекаем
из него символ
        }
    } ;
};
struct list
    char element; //символ
    int count;//количество повторений символа в предложении
   lisp bt;
};
haffm codes h codes;
string code_of_elementFSH[50];//массив для хранения кодов символов по Шеннону
void add(vector<list>& symbols, char s);
void sort(vector<list>& symbols);
```

```
void method FSH(vector<list> smbl, int lenght, int start of search, int
end of search, string Code, lisp& bin t, int deep);
void show method FSH(vector<list> smbl, int lenght, int start of search, int
end of search, string Code, lisp& bin t, int deep);
void final coding(vector<list>& smbl, char elm, string& coded string, int& 1,
string codes[]);
void close final coding(vector<list>& smbl, char elm, string& coded string,
int& 1, string codes[]);
void decode FSH(vector<list> smbl, string str to decode, const lisp b,
string& decoded str);
void method H(vector<list>& smbl, lisp& bin tree, string& code);
void search of min element(vector<list>& smbl, list& elmn);
void codeHuffman(string& code, vector<list>& smbl, lisp bin tree,int deep);
const char* ar[] = { "peter piper picked a peck of pickled peppers", "the
shells she sells are the sea-shore shells", "better than the bitter butter",
                     "i thought a thought", "sea-serpents swam the seven
seas", "she sits she shines"};
int main()
{
    setlocale(LC ALL, "rus");
    ofstream fout("/home/ira/CLionProjects/AiSD course work/fout.txt");
    unsigned int count variant;
    int action;
   bool exit = false;
    while (!exit)
    {
        cout << "Выберите тип варианта: " << endl;
        cout << "1. Кодирование Хаффмана и декодирование Фано - Шеннона." <<
endl;
        cout << "2. Кодирование Фано - Шеннона и декодирование Хаффмана. "
<< endl;
        cout << "3. Выход. \n ----- << endl;
        cin >> action;
        switch (action)
            case 1:
```

```
cout << "Здравствуйте! Введите количество нужных вариантов: "
<< endl;
                cin >> count variant;
                for (int i = 0; i < count variant; i++)</pre>
                    fout << "BAPMAHT №" << i + 1 << endl
                          << "Задание 1.\nЗакодируйте предложение методом
            << ";
Хаффмана:
                    cout << "BAPMAHT №" << i + 1 << endl
                          << "Задание 1.\nЗакодируйте предложение методом
            << "<<endl;
Хаффмана:
                    string main str = ar[i]; //берем строку из массива
                    cout<<main str;</pre>
                    fout<<main str;</pre>
                    int lenght of main str = main str.size();
                    cout<<"\nПромежуточные выводы:\n";
                    vector<list> symbolsH;//вектор для метода Хаффмана
                    vector<list> symbolsFSH;//вектор для частот и
символов, дерева Фано-Шеннон
                    lisp binary tree = Create();//дерево для кодирования и
декодирования
                    lisp binary treeH = Create();
                    string codeFSH;//код символа,будет помещен в массив
code of element..
                    string codeH;
                    string coded stringFSH;//закодированная строка
                    string coded stringH;
                    string decodedFSH;//декодированная строка
                    string decodeH;
                    string curr str;//дополнительная строка для заполнения
вектора
                    curr str = string(main str);
                    for (int i = 0; i < curr str.length(); <math>i++) //добавление
в вектор символа и частоты встречаемости этого символа
                     {
                        add(symbolsFSH, curr str[i]);
                        add(symbolsH, curr_str[i]);
                    }
```

```
sort(symbolsFSH); //сортируем символы для Шеннона-Фано в
порядке невозрастания частот
                     cout << "Символы отсортированы по частоте
встречаемости:\n";
                     int k = 0;
                     sort(symbolsH);//сортируем символы для Хаффмана в порядке
невозрастания частот
                     while(symbolsH[k].element){
                         cout << symbolsH[k].element << "-" <<</pre>
symbolsH[k].count << endl;</pre>
                         k++;
                     }
                     //далее создаем бинарное дерево кодов по алгоритму
Шеннона-Фано
                    method FSH(symbolsFSH, lenght of main str, 0,
symbolsFSH.size() - 1, codeFSH, binary tree,0);
                     for (int i = 0, l = 0; i < lenght_of_main_str; i++)</pre>
                     {
                         close final coding(symbolsFSH, main str[i],
coded stringFSH, l, code of elementFSH); // Кодирование строки \Phi-Ш
                     }
                     cout<<"\nПостроение бинарного дерева: "<<endl;
                     method H(symbolsH, binary treeH, codeH);//создаем
бин.дерево символов по Хаффману
                     codeHuffman(codeH, symbolsH, binary treeH,0);//по дереву
получаем коды символов
                     coded stringH = h codes.encodeString(main str);//кодируем
строку методом Хаффмана
                     cout<<"\nOTBeT:\n";
                     h codes.decode H(coded stringH, binary treeH, decodeH);
                     cout << coded stringH << endl</pre>
                          << "\nЗадание 2.\nРаскодируйте строку:" << endl;
                     cout << coded stringFSH << endl << endl;</pre>
                     cout << "Коды символов: " << endl;
                     cout << "----" << endl;
                     for (int i = 0; i < symbolsFSH.size(); i++)</pre>
                         cout << " " << symbolsFSH[i].element << " | " <<</pre>
code of elementFSH[i] << endl;</pre>
                     }
                     string str;
```

```
fout << main str << ">>" << endl
                        << "\nЗадание 2.\nРаскодируйте строку:" << endl;
                   fout << coded stringFSH << endl << endl;</pre>
                   fout << "\nКоды символов: " << endl;
                   fout << "----" << endl;
                   for (int i = 0; i < symbolsFSH.size(); i++)</pre>
                       fout << " " << symbolsFSH[i].element << " | " <<</pre>
code of elementFSH[i] << endl;</pre>
                   }
                   cout << "\nПроцесс декодирования:\n"<<endl;
decode_FSH(symbolsFSH,coded_stringFSH,binary_tree,str);//декодируем строку
                   cout<<"\nOTBeT:\n"<<main str<<endl;</pre>
                   fout << "-----
                     -----" << endl
                        << "\n
                                             Вариант №" << i + 1 << endl
                                 Ответы.
                        << "\nЗадание 1.\n" << "Метод Хаффмана:\n" <<
coded stringH << endl;</pre>
                   fout << "\nЗадание 2.\n" << endl << "<<" << main_str <<
">>" << endl;//раскодированная строка
                   fout <<
               \n\n" << endl;</pre>
               }
               cout << "Варианты записаны в файл fout.txt" << endl;
               break;
           case 2:
               cout << "Здравствуйте! Введите количество нужных вариантов: "
<< endl;
               cin >> count variant;
               for (int i = 0; i < count variant; i++)</pre>
                {
                   string main_str = ar[i];
                   fout << "BAPMAHT Nº" << i + 1 << endl
                        << "Задание 1.\nЗакодируйте предложение методом Фано
- Шеннона: \n "<<main str<<endl;
```

```
cout << "BAPMAHT №" << i + 1 << endl
                         << "Задание 1.\nЗакодируйте предложение методом Фано
- Шеннона: \n "<<main str<<endl;
                    int lenght of main str = main str.size();
                    vector<list> symbolsH;//вектрор для метода Хаффмана
                    vector<list> symbolsFSH;//вектор для частот и
символов, дерева Фано-Шеннон
                    lisp binary tree = Create();//дерево для кодирования и
декодирования
                    lisp binary treeH = Create();
                    string codeFSH;//код символа,будет помещен в массив
code of element..
                    string codeH;
                    string coded stringFSH;//закодированная строка
                    string coded stringH;
                    string decodedFSH;
                    string decodeH;
                    string curr str;//дополнительная строка для заполнения
вектора
                    curr str = string(main str);
                    for (int i = 0; i < curr str.length(); <math>i++) //добавление
в вектор символа и частоты
                        add(symbolsFSH, curr str[i]);
                        add(symbolsH, curr str[i]);
                    cout<<"Символы отсортированы по частоте
встречаемости:"<<endl;
                    sort(symbolsFSH);//сортируем символы в порядке
невозрастания частот
                    int k;
                    while(symbolsFSH[k].element){
                        cout << symbolsFSH[k].element << "-" <<</pre>
symbolsFSH[k].count << endl;</pre>
                        k++;
                    }
                    cout<<"\nПостроение бинарного дерева Шеннона-Фано:
"<<endl;
                    //строим бинарное дерево по Шеннону-Фано
```

```
show_method_FSH(symbolsFSH, length of main str, 0,
symbolsFSH.size() - 1, codeFSH, binary tree,0);
                    cout << "\nКоды символов: " << endl;
                    cout << "----" << endl;
                     for (int i = 0; i < symbolsFSH.size(); i++)</pre>
                         cout << " " << symbolsFSH[i].element << "|" <<</pre>
code of elementFSH[i] << endl;</pre>
                    cout<<"\nПроцесс кодирования:"<<endl;
                    for (int i = 0, l = 0; i < length of main str; <math>i++)
                         final coding(symbolsFSH, main str[i],
coded stringFSH, 1, code of elementFSH); // Кодирование строки Ф-Ш
                    cout<<"\nOTBeT: "<<coded stringFSH<<endl;</pre>
                    cout << "\nЗадание 2.\nРаскодируйте строку:" << endl;
                    cout << coded stringFSH << endl << endl;</pre>
                    cout << "Коды символов: " << endl;
                    cout << "----" << endl;
                    for (int i = 0; i < symbolsFSH.size(); i++)</pre>
                         cout << " " << symbolsFSH[i].element << "|" <<</pre>
code of elementFSH[i] << endl;</pre>
                     }
                    fout << "\nЗадание 2.\nРаскодируйте строку:" << endl;
                    fout << coded stringFSH << endl << endl;</pre>
                     fout << "Коды символов: " << endl;
                    fout << "----" << endl;
                    string str;
                    for (int i = 0; i < symbolsFSH.size(); i++)</pre>
                         fout << " " << symbolsFSH[i].element << "|" <<</pre>
code of elementFSH[i] << endl;</pre>
                     }
                    cout << "\nПроцесс декодирования:\n"<<endl;
decode FSH(symbolsFSH,coded stringFSH,binary tree,str);//декодирование строки
                    cout<<"Ответ: "<<main str<<endl;
```

```
-----" << endl
                                            Вариант №" << i + 1 << endl
                                 Ответы.
                        << "\nЗадание 1.\n" << "Метод Фано-Шеннон:\n" <<
coded_stringFSH << endl// << "Метод Хаффмана:\n" << coded_stringH << endl
                        << "\nЗадание 2.\n" << endl << "<<" << main str <<
">>" << endl;//раскодированная строка
                   fout <<
                \n\n" << endl;
               }
               cout << "Варианты записаны в файл fout.txt \n -----
-" << endl;
               break;
           case 3:
               exit = true;
               break;
           default:
               cout << "Incorect num \n -----" << endl;</pre>
               break;
       }
    }
    system("pause");
}
void add(vector<list>& symbols, char s)//функция добавления в вектор символа
и частоты его встречаемости
   bool isFound = false;
    for (int i = 0; i < symbols.size(); i++) {
       if (symbols[i].element == s) {
           symbols[i].count++;
           isFound = true;
           break;
```

fout << "-----

```
}
    }
    if (!isFound) {
        list e;
        e.element = s;
        e.count = 1;
        symbols.push_back(e);
    }
}
void sort(vector<list>& symbols)//сортировка символов вектора в порядке
невозрастания частот
{
    int size = symbols.size();
    for (int i = 0; i < size - 1; i++)
    {
        for (int j = 0; j < size - i - 1; j++)
        {
            if (symbols[j + 1].count > symbols[j].count)
            {
                swap(symbols[j], symbols[j + 1]);
            }
        }
    }
}
void middle(vector<list> simbols, int S , int B , int& node, int& SL, int&
SR)//деление символов на две части по сумме частот
{
    int D;
    node = B;
    SL = simbols[node].count;
    SR = S - SL;
    D = SL - SR;
    while ((D + simbols[node + 1].count) < 0) {
        node++;
```

```
SL = SL + simbols[node].count;
        SR = SR - simbols[node].count;
        D = SL - SR;
    }
}
//демонстрация создания дерева по Шеннону-Фано
void show_method_FSH(vector<list> smbl, int lenght, int start_of_search, int
end of search, string Code, lisp& bin t,int deep)
{
    int node;
    int sum_of_Lt, sum_of_Rt;
    if (start of search == end of search)//если 1 элемент
    {
        bin t = makeLeaf(start of search);
        code of elementFSH[start of search] = Code;
    }
    else {
        lisp bL = Create();
        lisp bR = Create();
        middle(smbl, lenght, start_of_search, node, sum_of_Lt,
sum of Rt);//находим середину по сумме частот
        string str1 = Code;
        string str2 = Code;
        strl.append("0");//в код левой части добавляем 0
        str2.append("1");//в код правой части добавляем 1
        for(int i=0;i<deep;i++)</pre>
            cout<<" ";
        cout<<str1<<endl;
        show method FSH(smbl, sum of Lt, start of search, node, strl,
bL, deep+1);//рекурсивный вызов для каждой из частей
        for(int i=0;i<deep;i++)</pre>
            cout<<" ";
        cout<<str2<<endl;
        show method FSH(smbl, sum of Rt, node + 1, end of search, str2,
bR, deep+1);
        bin t = ConsBT(bL, bR);
    }
}
```

```
//создание дерева по методу Шеннона-Фано без демонстрации
void method FSH(vector<list> smbl, int lenght, int start of search, int
end of search, string Code, lisp& bin t,int deep)
    int node;
    int sum of Lt, sum of Rt;
    if (start of search == end of search)//если 1 элемент
    {
        bin t = makeLeaf(start of search);
        code of elementFSH[start of search] = Code;
    }
    else {
        lisp bL = Create();
        lisp bR = Create();
        middle(smbl, lenght, start_of_search, node, sum of Lt,
sum of Rt);//находим середину по сумме частот
        string str1 = Code;
        string str2 = Code;
        strl.append("0");//в код левой части добавляем 0
        str2.append("1");//в код правой части добавляем 1
        method_FSH(smbl, sum_of_Lt, start_of_search, node, str1,
bL, deep+1);//рекурсивный вызов для каждой из частей
        method FSH(smbl, sum of Rt, node + 1, end of search, str2,
bR, deep+1);
        bin t = ConsBT(bL, bR);
    }
}
void final coding(vector<list>& smbl, char elm, string& coded string, int& l,
string codes[])//функция кодирования с демонстрацией
{
    for (int i = 0; i < smbl.size(); i++) {</pre>
        if (elm == smbl[i].element)//если встреченный символ равен символу из
вектора
            1 += code of elementFSH[i].size();
```

```
coded string.append(codes[i]);//в закодированную строку добавляем
код этого символа
            cout << "символ " << elm << " заменён на код " << codes[i] <<
endl;
    }
}
//функция кодирования без демонстрации
void close final coding(vector<list>& smbl, char elm, string& coded string,
int& 1, string codes[])
{
    for (int i = 0; i < smbl.size(); i++) {</pre>
        if (elm == smbl[i].element)
            1 += code of elementFSH[i].size();
            coded string.append(codes[i]);
        }
    }
}
void method H(vector<list>& smbl, lisp& bin tree, string& code)//создаем
дерево элементов по Хаффману
    for (int i = 0; i < smbl.size(); i++) { smbl[i].bt =</pre>
makeLeaf(smbl[i].element); }//дополним вектор деревом
    while (smbl.size() > 1)
    {
        list min_tree1, min_tree2;
        search of min element(smbl, min tree1);
        search of min element(smbl, min tree2);//находим два минимальных
элемента
        list b;
```

```
b.bt = ConsBT(min tree1.bt, min tree2.bt);//создаем поддерево из
минимальных элементов
        b.count = min tree1.count + min tree2.count;//складвыаем частоты
        smbl.push back(b);//добавляем в вектор получившееся дерево
    }
    bin tree = smbl[0].bt;
}
void search of min element(vector<list>& smbl, list& elmn)//поиск элемента с
минимальной частотой повторений
{
    int deleteItem = 0, i = 0;
    elmn = smbl[0];
    while (i < smbl.size())</pre>
        if (smbl[i].count < elmn.count)</pre>
        {
            elmn = smbl[i];
            deleteItem = i;
        }
        i++;
    }
    smbl.erase(smbl.begin() + deleteItem);
}
void codeHuffman(string& code, vector<list>& smbl, lisp bin tree,int
deep)//создаем бинарное дерево кодов по Хаффману
    if (isLeaf(bin tree))//если дошли до листа, то в h.codes описываем символ
и код
    {
        h_codes.append(bin_tree->node.leaf, code);
    }
    else
```

```
{
        string str1 = code;
        string str2 = code;
        str1.append("0");//для левого поддерева в код добавляем 0
        str2.append("1");//для правого поддерева в код добавляем 1
        for(int i=0;i<deep;i++)</pre>
            cout<<" ";
        cout<<str1<<endl;
        codeHuffman(str1, smbl, Left(bin tree),deep+1);//вызываем рекурсивно
для левого и правого поддеревьев
        for(int i=0;i<deep;i++)</pre>
            cout<<" ";
        cout<<str2<<endl;
        codeHuffman(str2, smbl, Right(bin tree), deep+1);
}
void decode FSH(vector<list> smbl, string str to decode, const lisp b,
string& decoded str)//функция декодирования
{
    char x;
    int i = 0;
    while (i != str to decode.size())
        lisp p = b;
        if (!isLeaf(p))
        {
            while (!isLeaf(p))//пока не дошли до листа
            {
                x = str to decode[i];
                if (x == '0') {//если встречен 0, спускаемся по левому
поддереву
                    p = Left(p);
                    cout<<"Bcrpeчeн '0'"<<endl;
                else { //если встречен 1, спускаемся по правому поддереву
                    p = Right(p);
```

```
cout<<"Bcrpeчen '1'"<<endl;
              }
           }
           cout <<"\nДекодирован символ:\n"<<smbl[p-
>node.leaf].element<<endl;</pre>
       else i++;
       decoded str.push back(smbl[p->node.leaf].element);//добавляем элемент
в декодированную строку
Файл header.h
namespace binTree modul
   //-----
   typedef int base; // базовый тип элементов (атомов)
   struct s_expr;
   struct binTree
       s expr* lt; // указатель на левое поддерево
       s expr* rt; // указатель на правое поддерево
   };
   struct s_expr {
       bool tag; // true если Leaf, false если Inter
       union //идентификаторы полей попадают во внешнюю область видимости
       {
          base leaf; // лист дерева
           binTree inter; //звено дерева (не лист)
       } node;
   };
   typedef s_expr* lisp;
```

```
lisp Create(void); // создание пустого бинарного дерева
    lisp Left(const lisp); // возвращает указатель на левое поддерево
    lisp Right(const lisp); // возвращает указатель на правое поддерево
    lisp ConsBT(const lisp lt, const lisp rt); // создание дерева из двух
поддеревьев
   void destroy(lisp&); // уничтожает дерево
   bool isLeaf(const lisp s); // проверяет, является ли элемент листом
    lisp makeLeaf(const base x); // создает лист со значением x
Файл header.cpp
#include <iostream>
#include <cstdlib>
#include "header.h"
using namespace std;
namespace binTree modul
{
   bool isLeaf(const lisp s)
       if (s == NULL) return false;
       else return (s->tag);
    //----
    lisp Create()
       return NULL;
    }
    lisp Left(const lisp b)
    {
```

if (b != NULL) {

```
if (!isLeaf(b))return b->node.inter.lt;
       else {
          cout << b->node.leaf << "\n";</pre>
          return NULL;
       }
   }
lisp Right(lisp b)
   if (b != NULL) {
       if (!isLeaf(b))return b->node.inter.rt;
       else {
          cout << b->node.leaf << "\n";</pre>
          return NULL;
       }
   }
}
//----
lisp ConsBT(const lisp lt, const lisp rt)
   lisp p;
   p = new s expr;
   if (p == NULL) { cerr << "Memory not enough\n"; exit(1); }</pre>
   else {
       p->tag = false;
       p->node.inter.lt = lt;
       p->node.inter.rt = rt;
      return p;
//----
void destroy(lisp& b)
{
   if (b != NULL) {
       destroy(b->node.inter.lt);
       destroy(b->node.inter.rt);
```

}