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 Хаффмана, динамическое кодирование и декодирование Хаффмана

Студент гр. 8383 _____ Шишкин И.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Шишкин И.В.

Вариант 3.

Группа 8383

Тема работы: сравнение динамического декодирования Хаффмана со статическим и динамического кодирования Хаффмана со статическим (исследование)

Исходные данные:

На вход подается либо файл с текстом, который нужно закодировать, либо файл с ключами и закодированной строкой, которую нужно декодировать.

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 11.10.2019

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Шишкин И.В.

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

В данной курсовой работе были сравнены алгоритмы статического кодирования Хаффмана с динамическим, а также статическое декодирование Хаффмана с динамическим. Для этого была написана программа на языке C++. Считывание данных производится из файла.

SUMMARY

In this course work were compared static coding Huffman algorithm with dynamic coding Huffman algorithm, and also static decoding Huffman algorithm with dynamic decoding Huffman algorithm. For this, a program was written in C ++. Reading data is made from a file.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной курсовой работе было реализовано 4 алгоритма: статическое кодирование и декодирование Хаффмана и динамическое кодирование и декодирование Хаффмана. Целью курсовой работы было узнать какой алгоритм эффективнее: статический или динамический. Для этого были сравнены по скорости выполнения алгоритмы статического кодирования с динамическим и статического декодирования с динамическим, а также посчитано количество бит для закодированной динамически строки и статически.

Постановка задачи:

Динамическое кодирование и декодирование по Хаффману – сравнительное исследование со “статическим” методом.

1. ОПИСАНИЕ АЛГОРИТМОВ

1.1. Описание статического кодирования Хаффмана

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. Коды Хаффмана обладают свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана.

1) Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.

2) Выбираются два свободных узла дерева с наименьшими весами.

3) Создается их родитель с весом, равным их суммарному весу.

4) Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.

5) Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.

6) Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

1.2. Описание статического декодирования Хаффмана

При статическом декодировании, на вход подаются помимо закодированной строки ключи, нужные для декодирования.

1) По полученным ключам строится дерево, на листьях которого будут символы.

2) По закодированной строке производится обход дерева. Когда алгоритм упирается в лист дерева - в результирующую строку записывается символ, расположенный на данном листе.

1.3. Описание динамического кодирования Хаффмана

В начале работы алгоритма дерево кодирования содержит только один

специальный символ, всегда имеющий частоту 0. Он необходим для занесения в дерево новых символов. Этот символ называют escape-символом (<esc>). Левые ветви дерева помечаются 0, а правые – 1. При нарушении упорядоченности дерева (после добавлении нового листа или изменении веса имеющегося листа) его необходимо упорядочить. Чтобы упорядочить дерево необходимо поменять местами два узла: узел, вес которого нарушил упорядоченность, и последний из следующих за ним узлов меньшего веса. После перемены мест узлов необходимо пересчитать веса всех их узлов-предков.

В адаптивном алгоритме сжатия Хаффмана используется упорядоченное бинарное дерево. Бинарное дерево называется упорядоченным, если его узлы могут быть перечислены в порядке неубывания веса. Перечисление узлов происходит по ярусам снизу-вверх и слева-направо в каждом ярусе. Узлы, имеющие общего родителя, находятся рядом на одном ярусе.

1) Элементы входного сообщения считываются посимвольно.

2) Если входной символ присутствует в дереве, в выходной поток записывается код, соответствующий последовательности нулей и единиц, которыми помечены ветки дерева, при проходе от корня дерева к данному

листу. Вес данного листа увеличивается на 1. Веса узлов-предков корректируются. Если дерево становится неупорядоченным - упорядочивается.

3) Если очередной символ, считанный из входного сообщения при сжатии, отсутствует в дереве, в выходной поток записывается новый символ. В дерево вместо escape-символа добавляется ветка: родитель, два потомка. Левый потомок становится escape-символом, правый - новым добавленным в дерево символом. Веса узлов-предков корректируются, а дерево при необходимости упорядочивается.

1.4. Описание динамического декодирования Хаффмана

1) Элементы входного сообщения считываются побитово.

2) Каждый раз при считывании 0 или 1 происходит перемещение от корня вниз по соответствующей ветке бинарного дерева Хаффмана, до тех пор, пока не будет достигнут какой-либо лист дерева.

3) Если достигнут лист, соответствующий символу, в выходное сообщение записывается данный символ символа. Вес листа увеличивается на 1, веса узлов-предков корректируются, дерево при необходимости упорядочивается.

4) Если же достигнут escape-символ, из входного сообщения считываются символ. В выходное сообщение записывается символ. В дерево добавляется новый символ, веса узлов-предков корректируются, затем при необходимости производится его упорядочивание.

1.5. Сравнение статического и динамического алгоритма Хаффмана

Так как алгоритм Хаффмана сжимает данные за счёт вероятностей появления символов в источнике, следовательно для того чтоб успешно что-то сжать и разжать нам потребуется знать эти самые вероятности для каждого символа. Статические алгоритмы справляются с этим таким образом: перед тем как начать сжатие файла программа пробегается по самому файлу и подсчитывает какой символ сколько раз встречается. Затем, в соответствии с

вероятностями появлений символов, строится двоичное дерево Хаффмана - откуда извлекаются соответствующие каждому символу коды разной длины. И на третьем этапе снова осуществляется проход по исходному файлу, когда каждый символ заменяется на свой код. Таким образом статическому алгоритму требуется два прохода по файлу источнику, чтоб закодировать данные.

Динамический алгоритм позволяет реализовать однопроходную модель сжатия. Не зная реальных вероятностей появлений символов в исходном файле - программа постепенно изменяет двоичное дерево с каждым встречаемым символом увеличивая частоту его появления в дереве и перестраивая в связи с этим само дерево. Однако становится очевидным, что выиграв в количестве проходов по исходному файлу - алгоритм начинает терять в качестве сжатия, так как в статическом алгоритме частоты встречаемости символов были известны с самого начала и длины кодов этих символов были более близки к оптимальным, в то время как динамическая модель, изучая источник, постепенно доходит до его реальных частотных характеристик и узнаёт их лишь полностью пройдя исходный файл. Однако у динамического алгоритма есть и преимущество. Так как динамическое двоичное дерево постоянно модифицируется новыми символами - нам нет необходимости запоминать их частоты заранее - при разархивировании, программа, получив из архива код символа, точно так же восстановит дерево, как она это делала при сжатии и увеличит на единичку частоту его встречаемости. Более того, нам не требуется запоминать какие символы в двоичном дереве не встречаются. Все символы которые будут добавляться в дерево и есть те, которые нам потребуются для восстановления первоначальных данных. В статическом алгоритме с этим делом немного сложнее - в самом начале сжатого файла требуется хранить информацию о встречаемых в файле источнике символах и их вероятностных частотах. Это обусловлено тем, что ещё до начала разархивирования нам необходимо знать какие символы будут встречаться и каков будет их код.

2. ОПИСАНИЕ СТРУКТУР И ФУНКЦИЙ ДЛЯ СТАТИЧЕСКОГО АЛГОРИТМА

2.1. Описание использованных структур для статического кодирования и декодирования Хаффмана

Первая структура - `struct node`, используется для записи дерева при статическом кодировании. Обладает следующими полями:

- `string list` - строка, содержащая сочетания символов;
- `int i` - частота встречаемости символа/сочетания символов;
- `int left` - индекс левого поддерева;
- `int right` - индекс правого поддерева.

Так как дерево для статического кодирования строится на базе массива, то и индекс левого и правого поддерева хранятся в виде целочисленной переменной.

Следующая структура - `struct ltcode`, используется для записи символов, их частот и кода. Обладает следующими полями:

- `unsigned long long frequency` - переменная, содержащая частоту встречаемости символа;
- `unsigned char letter` - содержит элемент;
- `string code` - код для буквы.

Данная структура используется как для кодирования, так и для кодирования.

Последняя структура для статического алгоритма - `struct nodeDecode`, используется для записи дерева при статическом декодировании. Обладает следующими полями:

- `unsigned char letter` - элемет;
- `bool isNode` - если элемент дерева - узел, то хранит 1, если элемент дерева - лист, то хранит 0;
- `string code` - код элемента;

- `nodeDecode* left` - указатель на левое поддерево;
- `nodeDecode* right` - указатель на правое поддерево.

2.2. Описание функции encodeThis

Функция `void encodeThis(vector <node>& tree, ltcode* count, ifstream& in, string* res)`, на вход которой подается пустое дерево `tree`, пустой элемент структуры `ltcode`, файл, из которого считывается строка `in`, и строка, в которую будет записан результат кодирования - `res`. Дерево и структура, содержащая в себе коды элементов подаются, чтобы в случае, если понадобится вывести дерево и коды символов, можно было сделать это в головной программе. Для начала функция считывает строку и запоминает частоту встречаемости каждого символа, занося данные в массив `count`. Затем сортирует данный массив в порядке убывания частоты встречаемости символа. Далее создается вектор, в котором складываются символы (то есть если символ 'a' встречается 1 раз и символ 'b' встречается 1 раз, то в вектор заносится сочетание символов "ab", а так же складывается их частота). В ходе цикла добавления этих элементов в вектор, он сортируется. Затем вызывается функция `huffmanEncoding` , в которой на основе дерева и начального массива происходит кодирование.

2.3. Описание функции huffmanEncoding

Рекурсивная функция `void huffmanEncoding(int num, string code, vector <B>& tree, ltcode* letterCodes)` , на вход которой подается: `num` - индекс элемента вектора tree (дерево реализовано на базе массива), строка `code` - код для символа или сочетания символов, вектор `tree` типа B, элемент `letterCodes` структуры `ltcode`. Функция начинается с того, что записывает код символа в поле `code` вектора `tree`. Затем, если в данном векторе хранится сочетание символов, то вызывается эта же функция для левого поддерева, при этом добавляя к коду "0", а потом для правого поддерева,

добавляя к коду "1". Если же в векторе хранится не сочетание символов, значит в нем хранится одиночный символ. В таком случае в элементе `letterCodes` структуры `ltcode` производится поиск данного элемента. Когда элемент находится, в поле `code` этой структуры заносится код для символа.

2.4. Описание функции decodeThis

Функция `void decodeThis(nodeDecode * tree, ltcode * count, ifstream & in, string * res)`, на вход которой подается те же переменные, что и в случае кодирования. В начале с помощью цикла while производится считывание ключей для декодирования. Важно отметить, что ключи и закодированная строка, которую нужно декодировать, должны разделяться двумя символами '\n'. Символы и коды записываются в переменную `stringLinks`. Далее считывается строка, которая заносится в переменную `S`. Затем сортируется `stringLinks` в порядке возрастания длины кода. Далее по этой переменной строится дерево. Например, для ключей a - 0, b - 10, c - 11 дерево см. на рис. 1.

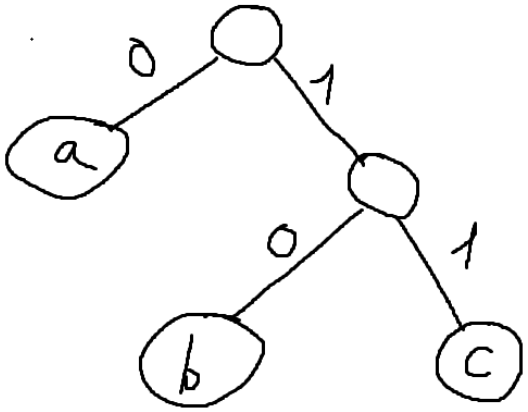


Рисунок 1 - Иллюстрация дерева

Далее вызывается функция `huffmanDecoding`.

2.5. Описание функции huffmanDecoding

По полученной строке просматривается дерево и производится обход по нему. Когда функция упирается в лист - к результату добавляется элемент,

который хранится в этом листе. То есть функция каждый раз проходит дерево с головы до листа.

3. ОПИСАНИЕ СТРУКТУР И ФУНКЦИЙ ДЛЯ ДИНАМИЧЕСКОГО АЛГОРИТМА

3.1. Описание использованных структур для динамического кодирования и декодирования Хаффмана

Для динамического алгоритма Хаффмана были созданы классы `Tree` и `Sym`. Создана структура `struct node`, поля которой содержат следующие элементы:

- `char elem` — значение элемента.
- `int left` — индекс левого узла (-1 по умолчанию).
- `int weight` — индекс правого узла (-1 по умолчанию).
- `bool isELem` — вес данного узла.

Функции, описанные в классе:

`void addElem(char c)` - функция добавляет элемент в дерево, где `char c` - элемент, который нужно добавит.

`bool findElem(char c, int ind)` - функция поиска элемента в дереве (по ходу поиска строит путь), где `char c` — искомый элемент, `int ind` - индекс узла начала поиска символа. Возвращает true если элемент найдет, в противном случае возвращает false.

`void reSum(int ind)` - функция пересчитывает веса узлов. `int ind` — индекс элемента, с которого нужно начать.

`void check1()` - Функция проверяет правильность порядка расположения листьев, если расположение неверно, то она исправляет его.

`void addOne(char c)` - функция добавляет к весу элемента `c` единицу. `char c` - элемент, к которому нужно добавить 1.

`void check2()` - функция проверяет упорядоченность дерева, если оно непорядочно, то упорядочивает его.

`char processSym(char c)` - функция обрабатывает входной символ из закодированного файла, где `char c` - входной символ. Функция возвращает

пустой символ, если входной символ является частью кода, и возвращает символ из дерева, если он был найдет по коду.

`int readFile(ifstream& file)` - функция для динамического декодирования, которая считывает ключи из файла `file`.

`void print_Codes(ofstream & out)` - печатает коды символов и сами символы в файл `out`.

`void print_Tree(int p, int level, int n)` - функция, печатающая дерево, где `int p` - индекс корня дерева, `int level` - уровень глубины рекурсии, `int n` - правое (0) или левое (1) ответвление.

3.2. Описание функции dynamicEncoding

Функция `void dynamicEncoding(ifstream & fin, ofstream & fout, string * res)`. Производится считывание строки с помощью цикла while, а затем проверяется и строится дерево с помощью функций, описанных в предыдущих классах.

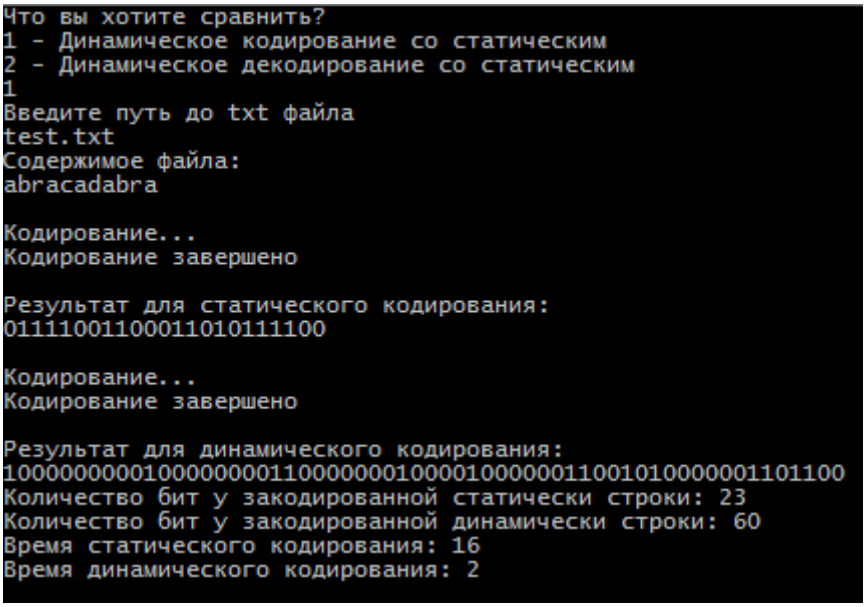
3.3. Описание функции dynamicDecoding

Функция `void dynamicDecoding(ifstream & fin, ofstream & fout, string * res)`. С помощью функции `readFile`, описанной в классе `Sym`, производится считывание кодов символов и самих символов. Затем проверяется и строится дерево с помощью функций, описанных в предыдущих классах.

4. ТЕСТИРОВАНИЕ ПРОГРАММЫ

4.1. Тестирование сравнения динамического и статического кодирования Хаффмана

Пример выполнения программы при сравнении кодирования:



Было произведено 5 вычислений для каждого файла и найдена средняя скорость кодирования. Скорость замеряется с помощью библиотеки ctime, значение выводится в количестве тиков.

Номер теста	Вес файла, КБ	Время статического кодирования	Количество бит у закодированной статически строки	Время динамического кодирования	Количество бит у закодированной динамически строки
1	7,83	123,5	35388	8712,5	37027
2	0,011	16	23	2	60
3	0,231	56	1052	145,75	1437

Анализируя результаты тестирования можно сделать вывод, что при кодировании больших файлов по времени статическое кодирование справляется намного быстрее динамического, при этом количество бит у закодированных строк обеими способами примерно одинаковое.

С другой стороны, при кодировании строк маленькой длины динамическое кодирование справляется по времени намного быстрее, хоть количество бит и больше.

Большое количество бит у закодированной динамически строки объясняется тем, что символы записываются в виде 8-битового их представления в коде символа ASCII.

4.2. Тестирование сравнения динамического и статического декодирования Хаффмана

Пример выполнения программы при сравнении декодирования:

```
Что вы хотите сравнить?
1 - Динамическое кодирование со статическим
2 - Динамическое декодирование со статическим
2
Введите название файла для статического декодирования (если файл называется outp
utStaticE.txt - нажмите Enter)

Содержимое файла:
a 0
b 111
r 10
c 1100
d 1101


01111001100011010111100

Декодирование...
Декодирование завершено

Результат для статического декодирования:
abracadabra
Введите название файла для динамического декодирования (если файл называется out
putDynamicE.txt - нажмите Enter)

Содержимое файла:


10000000001000000001100000001000010000001100101000000001101100

Декодирование...
Декодирование завершено

Результат для динамического декодирования:


Время статического декодирования: 4

Время динамического декодирования: 1
```

Было произведено 5 вычислений для каждого файла и найдена средняя скорость декодирования. Скорость замеряется после кодирования тех же файлов, которые были представлены для кодирования, за исключением одного.

Четвертый опыт был сделан после кодирования одного и того же символа, который повторяется 100 раз.

Номер опыта	Вес файла, КБ	Время статического декодирования	Время динамического декодирования
1	7,83	34	136
2	0,011	4	1
3	0,231	11	8
4	0,11	4	1

Анализируя результаты тестирования можно сделать вывод, что при кодировании больших файлов по времени статическое декодирование справляется быстрее динамического.

С другой стороны, при декодировании строк небольшой длины или при малом количестве различных символов, динамическое декодирование справляется по времени намного быстрее, чем статическое.

Большое время для больших файлов в динамическом кодировании можно объяснить тем, что в файле большое количество различных символов, поэтому каждый раз дерево приходится "пробегать" заново и перестраивать его.

4.3. Тестирование пограничных случаев

Input	Output
Кодирование пустого файла	Вы выбрали пустой файл
Кодирование единичного символа	Результат для статического: 0 Результат для динамического: 10000000

Так же учтен тот случай, если при декодировании написать не соответствующие символы. В таком случае программа остановится на том момент, когда встретит этот символ.

ЗАКЛЮЧЕНИЕ

Проанализировав результаты выполнения программы, можно сделать следующие выводы.

Сравнение статического и динамического кодирования. Количество бит у закодированной динамически строки будет всегда больше, чем у статического алгоритма, особенно учитывая то, что символы записываются в коде ASCII. Что касается времени, то для больших строк статическое кодирование справляется намного быстрее динамического. Проблема заключается в том, что динамическое кодирование не считывает заранее строку, в следствие этого не знает частоту встречаемости каждого символа, поэтому каждый раз дерево приходится перестраивать.

Сравнение статического и динамического декодирования. При декодировании складывается такая же ситуация, как и при кодировании, то есть большие строки с различными символами динамически декодируется дольше, чем статически. Но, следует отметить, что разница здесь не такая большая, как при кодировании. При небольших размерах строк динамическое декодирование справляется быстрее.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Википедия // wikipedia. URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0 (дата обращения: 05.11.2019).

2. Статья о динамическом и статическом сжатии Хаффмана // compression.ru. URL: http://compression.ru/download/articles/huff/yankovoy_2004_huffman/dynamic_huffman.html (дата обращения: 19.11.2019).

3. Научная работа о адаптивном алгоритме Хаффмана сжатия информации / Кудрина М.А., Кудрин К.А., Дегтярева О.А., Сопченко Е.В. // Адаптивный алгоритм Хаффмана сжатия информации.

ПРИЛОЖЕНИЕ А

СОДЕРЖАНИЕ ФАЙЛА MAIN.CPP

```
#define _CRT_SECURE_NO_WARNINGS

#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
#include <fstream>
#include <ctime>
#include "dynamicHuffman.h"
#include "SymCode.h"

using namespace std;

struct ltcode {
	unsigned long long frequency = 0;	//частота встречаемости
символа
	unsigned char letter = NULL;		//буква
	string code = "";			//код для буквы
};

struct node {
	string list;	//сочетание символов
	int i;		//частота встречаемости символа/сочетания
символов

	int left;	//индекс левого поддерева
	int right;	//индекс правого поддерева

	string code = "";	//код для сочетания символов

	node() {		//конструктор
		left = 0;
		right = 0;
		list = "";
		i = 0;
	}
};

struct nodeDecode {
	unsigned char letter = NULL;
	bool isNode = 1;  //если это узел, то хранит 1
```

```
	string code = "";
	nodeDecode* left = NULL;
	nodeDecode* right = NULL;
};

void zeroset(ltcode* c);			//обнуляет все значения элемента с
bool cmpForStr(ltcode x, ltcode y);				//следующие 4 функции
- это компараторы для сортировки
bool cmpForStrDecoding(ltcode x, ltcode y);
bool cmpForTree(node x, node y);
bool cmpForTreeDecoding(node x, node y);
void encodeThis(vector <node>& tree, ltcode* count, ifstream& in, string*
res);		//функция, которая начинает статическое кодирование
void huffmanEncoding(int num, string code, vector <node>& tree, ltcode*
letter_codes);	//функция, которая продолжает статическое кодирование
void huffmanDecoding(nodeDecode* tree, string s, string* res);
//функция, которая продолжает статическое декодирование
void decodeThis(nodeDecode* tree, ltcode* count, ifstream& in, string*
res);		//функция, которая начинает статическое декодирование
void dynamicEncoding(ifstream& fin, ofstream& fout, string* res);
//функция, в которой производится динамическое кодирование
void dynamicDecoding(ifstream& fin, ofstream& fout, string* res);
//функция, в которой производится динамическое декодирование
char reading(ifstream& in);
//посимвольное считывание
void printTree(int num, vector <node>& tree, int level = 0);
//печать дерева при статическом кодировании
void printDecodeTree(nodeDecode* p, int indent);		//печать дерева
при статическом декодировании

int main() {
	setlocale(LC_ALL, "Russian");
	string S;					//строка, которую
вводит пользователь
	ltcode* count = new ltcode[256];		//выделяем память
под массив типа ltcode, который хранит в себе частоту встречаемости
введенных символов
	ifstream in;
	ofstream out;
	char how = NULL;
	string filePath;
	vector <node> ttree;
	nodeDecode* decodeTree = NULL;
	string res = "";
```

```
	Tree dynamicEncodingTree;
	//блок переменных для хранения времени выполнения функций
	long long staticDecodingStart = 0;
	long long staticDecodingEnd = 0;
	long long staticEncodingStart = 0;
	long long staticEncodingEnd = 0;
	long long dynamicEncodingStart = 0;
	long long dynamicEncodingEnd = 0;
	long long dynamicDecodingStart = 0;
	long long dynamicDecodingEnd = 0;

	cout << "Что вы хотите сравнить?\n1 - Динамическое кодирование со
статическим\n2 - Динамическое декодирование со статическим\n";
	how = getchar();
	if (how == '1') {
		getchar();
		cout << "Введите путь до txt файла\n";
		cin >> filePath;
		in.open(filePath);
		if (!in.is_open()) {
			cout << "Неверно введен путь до файла\n";
			return 0;
		}
		cout << "Содержимое файла:\n";
		string file = "";                              //переменная,
которая хранит в себе содержимое файла
		while (!in.eof()) {
			getline(in, file);
			cout << file << endl;
		}
		in.close();
		in.open(filePath);

		//-------------------------------------------------------------
---выполнение статического кодирования

		staticEncodingStart = clock();
		encodeThis(ttree, count, in, &res);
		staticEncodingEnd = clock();

		out.open("outputStaticE.txt");
		int i = 0;
		while (count[i++].frequency) {
```

```
                out << count[i - 1].letter << " " << count[i - 1].code
<< endl;
            }
            out << endl << endl << res;
            out.close();
            in.close();
            in.open(filePath);
            cout << endl << "Результат для статического кодирования:\n" <<
res << endl;
            int staticBytes = res.size();

            //-------------------------------------------------------------
---выполнение динамического кодирования

            res = "";
            //memset(res, NULL, strlen(res));
            out.open("outputDynamicE.txt");
            dynamicEncodingStart = clock();
            dynamicEncoding(in, out, &res);
            dynamicEncodingEnd = clock();
            out << endl << endl << res;
            out.close();
            cout << endl << "Результат для динамического кодирования:\n"
<< res << endl;
            cout << "Количество бит у закодированной статически строки: "
<< staticBytes << endl;
            cout << "Количество бит у закодированной динамически строки: "
<< res.size() << endl;

            cout << "Время статического кодирования: " <<
staticEncodingEnd - staticEncodingStart << endl;
            cout << "Время динамического кодирования: " <<
dynamicEncodingEnd - dynamicEncodingStart << endl;
      }

      else if (how == '2') {
            getchar();
            how = NULL;
            cout << "Введите название файла для статического декодирования
(если файл называется outputStaticE.txt - нажмите Enter)\n";
            how = getchar();
            if (how == '\n') filePath = "outputStaticE.txt";
            else cin >> filePath;
            in.open(filePath);
```

```
            if (!in.is_open()) {
                  cout << "Неверно введен путь до файла\n";
                  return 0;
            }
            cout << "Содержимое файла:\n";
            string file = "";                              //переменная,
которая хранит в себе содержимое файла
            while (!in.eof()) {
                  getline(in, file);
                  cout << file << endl;
            }
            in.close();
            in.open(filePath);

            //-------------------------------------------------------------
---выполнение статического декодирования

            staticDecodingStart = clock();
            decodeThis(decodeTree, count, in, &res);
            staticDecodingEnd = clock();

            out.open("outputStaticD.txt");
            out << res;
            out.close();
            cout << endl << "Результат для статического декодирования:\n"
<< res << endl;
            in.close();

            //-------------------------------------------------------------
---выполнение динамического декодирования

            how = NULL;
            cout << "Введите название файла для динамического
декодирования (если файл называется outputDynamicE.txt - нажмите
Enter)\n";
            how = getchar();
            if (how == '\n') filePath = "outputDynamicE.txt";
            else cin >> filePath;
            in.open(filePath);
            if (!in.is_open()) {
                  cout << "Неверно введен путь до файла\n";
                  return 0;
            }
            cout << "Содержимое файла:\n";
```

```
            file = "";
            while (!in.eof()) {
                  getline(in, file);
                  cout << file << endl;
            }
            in.close();
            in.open(filePath);

            res = "";
            //memset(res, NULL, strlen(res));
            out.open("outputDynamicD.txt");
            dynamicDecodingStart = clock();
            dynamicDecoding(in, out, &res);
            dynamicDecodingEnd = clock();
            out << res;
            cout << endl << "Результат для динамического декодирования:\n"
<< res << endl;
            out.close();
            in.close();

            cout << "\nВремя статического декодирования: " <<
staticDecodingEnd - staticDecodingStart << endl;
            cout << "\nВремя динамического декодирования: " <<
dynamicDecodingEnd - dynamicDecodingStart << endl;
      }
      else {
            cout << "Неверно выбран вариант сравнения\n";
            system("pause");
            return 0;
      }

      delete[] count;
      return 0;
}

void zeroset(ltcode* c) {                //функция, обнуляющая массив
структур c
      for (int i = 0; i < 256; i++) {
            c[i].frequency = 0;
            c[i].letter = i;
      }
}
```

```
bool cmpForStr(ltcode x, ltcode y) {           //компаратор для
сортировки строки
     if (x.frequency != y.frequency) return x.frequency > y.frequency;
     else return x.letter < y.letter;
}

bool cmpForStrDecoding(ltcode x, ltcode y) {
     if ((x.code).size() != (y.code).size()) return (x.code).size() <
(y.code).size();
     else return x.letter < y.letter;
}

bool cmpForTree(node x, node y) {            //компаратор для сортировки
дерева
     if (x.i != y.i) return x.i < y.i;
     else return x.list.size() > y.list.size();
}

bool cmpForTreeDecoding(node x, node y) {
     if ((x.code).size() != (y.code).size()) return (x.code).size() >
(y.code).size();
     else return x.list.size() > y.list.size();
}

void encodeThis(vector <node> & tree, ltcode * count, ifstream & in,
string * res) {
     cout << "\nКодирование...\n";
     char c = NULL;
     ltcode* stringLinks = new ltcode[256];
     string S;            //хранит в себе изначальную строку из букв

     zeroset(count);              //обнуление всех переменных массива
структур count

     while (c = reading(in)) {                        //производится
посимвольное считывание строки с подсчетом частоты встречаемости
          //if (c == '\n' || in.eof()) break;
          if (in.eof()) break;
          S += c;
          ++count[c].frequency;
     }

     if (S.size() == 0) {
          cout << "Вы выбрали пустой файл\n";
```

```
            return;
      }

      sort(count, count + 256, cmpForStr);              //сортировка
элементов массива count в порядке убывания частоты встречаемости символа
      node tmp;
      tmp.list = "0";
      int j = 0;
      for (j = 0; count[j].frequency; j++) {
            tmp.i = count[j].frequency;
            tmp.list[0] = count[j].letter;
            tree.push_back(tmp);                  //добавление в конец
вектора
      }
      int maxSize = j;
      long long size = 0;

      if (maxSize == 1) {
            count[0].code = "0";
            for (int i = 0; i < S.size(); i++) *res += count[0].code;
            return;
      }

      sort(tree.begin(), tree.end(), cmpForTree);    //сортировка дерева
в порядке возрастания частоты встречаемости

      for (j = 0; size < maxSize; ) {
            tmp.list = tree[j].list + tree[j + 1].list;
//складываем символы или сочетания символов
            tmp.i = tree[j].i + tree[j + 1].i;                //складываем
частоты встречаемости
            tmp.left = j;
            tmp.right = j + 1;

            size = tmp.list.size();

            tree.push_back(tmp);
            j += 2;
            sort(tree.begin() + j, tree.end(), cmpForTree);
//сортируем в порядке возрастания частоты все элементы, кроме тех,
которые уже прошли проверку
      }
      /*cout << "Дерево:\n";
      printTree(tree.size() - 1, tree);
```

```
	cout << endl;*/

	cout << tree.size() - 1 << endl;

	/*if (maxSize == 1) {
		count[0].code = "0";
	}*/
	huffmanEncoding(tree.size() - 1, "", tree, count);

	//cout << "\nКоды для букв:\n";
	for (j = 0; (count[j].frequency); ++j) {
		//cout << '\'' << count[j].letter << "' - " << count[j].code
<< "  ";
		stringLinks[count[j].letter].code = count[j].code;
	}

	//cout << "\n\nЗакодированная строка:\n";
	size = 0;
	for (int i = 0; i < S.size(); i++) {
		*res += stringLinks[S[i]].code;
		//strcat(*res, stringLinks[S[i]].code.c_str());
		//cout << stringLinks[S[i]].code;
		//size += stringLinks[S[i]].code.size();
	}
	//cout << "\nКоличество бит: " << size << endl;
	cout << "Кодирование завершено\n";
	delete[] stringLinks;
}

void huffmanEncoding(int num, string code, vector <node> & tree, ltcode *
letterCodes) {
	//cout << "Частота: " << tree[num].i << " , сочетание символов: "
<< tree[num].list << " , код: " << code << endl;
	tree[num].code = code;
	if (tree[num].list.size() > 1) {
		//cout << "Рекурсивный вызов функции для левого поддерева:\n";
		huffmanEncoding(tree[num].left, code + "0", tree,
letterCodes);
		//cout << "\nРекурсивный вызов функции для правого
поддерева:\n";
		huffmanEncoding(tree[num].right, code + "1", tree,
letterCodes);
		//cout << endl;
	}
```

```
	else {
		//cout << "Это единичный символ";
		for (int j = 0; ; j++) {                                //поиск
в элементе letterCodes структуры ltcode нужного символа и запись туда
кода
			if (letterCodes[j].letter == tree[num].list[0]) {
				letterCodes[j].code = code;
				break;
			}
		}
	}
}

void decodeThis(nodeDecode * tree, ltcode * count, ifstream & in, string
* res) {
	cout << "\nДекодирование...\n";
	char sym = NULL;    //считывание символа
	ltcode* stringLinks = new ltcode[256];
	std::string S;            //хранит в себе строку, которую нужно
декодировать
	string result;
	int numOfKeys = 0;
	int k = 0;       //переменная для записи ключей из ltcode
stringLinks в vector <node> tree

	while (!in.eof()) {                        //производится считывание
ключей
		char tmp = NULL;    //для считывания ненужных символов
		string codee;        //считывание кода символа
		sym = reading(in);
		if (sym == '\n') {
			sym = reading(in);
			if (sym != ' ') break;
			else sym = '\n';
		}
		else {
			tmp = reading(in);
			if (tmp != ' ') {
				cout << "Неверно введены ключи!\n";
				return;
			}
		}
		in >> codee;
		if (codee[0] != '0' && codee[0] != '1') {
```

```
                cout << "Неверно введены ключи!\n";
                return;
            }
            tmp = reading(in);

            stringLinks[sym].code = codee;
            stringLinks[sym].letter = sym;
            stringLinks[sym].frequency = 1;
            k++;
            //cout << stringLinks[sym].code << " " <<
stringLinks[sym].letter << " " << (int)stringLinks[sym].letter << endl;
      }
      in >> S;

      sort(stringLinks, stringLinks + 256, cmpForStrDecoding);
//сортировка элементов массива count в порядке возрастания длины кода
      //cout << "Количество бит: " << S.size() << endl;
      //for (int i = 0; i < 256; i++) cout << "sym:" <<
stringLinks[i].letter << " code " << stringLinks[i].code << endl;
      numOfKeys = k;
      k = 256 - k;
      //if (stringLinks[k].code == "") k++;

      tree = new nodeDecode;
      nodeDecode* head = tree;
      //int forTree = 0;  //для подсчета количества узлов дерева. Дерево
начинается с 0
      for (int i = k; i < 256; i++) {
            //cout << stringLinks[i].letter << endl;
            int j = 0;

            for (j = 0; j < stringLinks[i].code.size() - 1; j++) {
                  tree->isNode = 1;
                  if (stringLinks[i].code[j] == '0') {
                        if (tree->left == NULL)
                              tree->left = new nodeDecode;
                        tree = tree->left;
                  }
                  else {
                        if (tree->right == NULL)
                              tree->right = new nodeDecode;
                        tree = tree->right;
                  }
            }
```

```
            if (tree == NULL)
                  tree = new nodeDecode;
            tree->isNode = 1;
            if (stringLinks[i].code[j] == '0') {
                  tree->left = new nodeDecode;
                  tree->left->code += stringLinks[i].code;
                  tree->left->isNode = 0;
                  tree->left->letter = stringLinks[i].letter;
            }
            else {
                  tree->right = new nodeDecode;
                  tree->right->code += stringLinks[i].code;
                  tree->right->isNode = 0;
                  tree->right->letter = stringLinks[i].letter;
            }
            tree = head;
      }

      huffmanDecoding(tree, S, res);

      //cout << "Дерево:\n";
      //printDecodeTree(tree, 0);
      cout << "Декодирование завершено\n";

}

void huffmanDecoding(nodeDecode * tree, string s, string * res) {
      nodeDecode* head = tree;

      for (int i = 0; i < s.size() + 1; i++) {
            if (s[i] == '0') {
                  if (tree->left) {
                        tree = tree->left;
                  }
                  else {
                        *res += tree->letter;
                        tree = head;
                        tree = tree->left;
                  }
            }
            else {
                  if (tree->right) {
                        tree = tree->right;
                  }
```

```
                else {
                        *res += tree->letter;
                        tree = head;
                        tree = tree->right;
                }
        }
    }
}

void dynamicEncoding(ifstream & fin, ofstream & fout, string * res) {
    cout << "\nКодирование...\n";
    Tree t;
    Sym a;
    char c;
    string str;
    string answer;

    while (!fin.eof()) {
        fin.get(c);

        if (fin.eof()) {
            break;
        }

        if (!fin.eof()) {

            if (t.findElem(c, 2)) {
// поиск символа в дереве
                t.addOne(c);
// елси найден, то увеличиваем вес
                *res += t.retCode();
            }
            else {
                t.flag = 1;
                t.addElem(c);
// елси не найден, то добавляем в дерево
                t.findElem(c, 2);
                t.flag = 0;

                str = t.retCode();
// получаем код

                str.pop_back();
```

```
                    str = str + a.code(c);
// обрабатываем полученный код

                    *res += str;
                    str.clear();
               }
               t.reSum(2);
// проверяем и сортируем дерево
               t.check1();
               t.reSum(2);
               t.check2();
               t.reSum(2);
          }
     }
     fin.close();
     if ((*res).size() == 0) {
          cout << "Вы выбрали пустой файл\n";
     }
     //a.print_Codes(fout);
     cout << "Кодирование завершено\n";
}

void dynamicDecoding(ifstream & fin, ofstream & fout, string * res) {
     cout << "\nДекодирование...\n";
     char c;
     int flag = 1;
     Tree t;
     Sym a;
     string str;
     string answer;

     if (a.readFile(fin)) {
// читаем файл со статическими кодами
          cout << " Файл с кодами неверный" << endl;
          fin.close();
          return;
     }

     while (!fin.eof()) {
          fin.get(c);
          //cout << *res << endl;
          if (fin.eof()) {
               cout << "Декодирование завершено" << endl;
               return;
```

```
            }

            //cout << "Был получен символ: " << c << endl;

            if ((c != '1' && c != '0' && !fin.eof())) {           // елси
считаный символ не 0 или не 1
                  cout << "Неверный файл" << endl;
                  return;
            }


            if (!fin.eof()) {


                  if (t.retEsc()) {
// елси находися в Esc, то считываем
                        //cout << "  Считываем статический код символа и
ищем его в фале с кодами"  << endl;  // статический код символа
                        str.push_back(c);

                        for (int i = 0; i < 7; i++) {
// считваение

                              if ((c != '1' && c != '0' && !fin.eof())) {
                                    cout << "Неверный файл" << endl;
                                    return;
                              }

                              fin.get(c);
                              str.push_back(c);

                              t.flag = 1;
                        }

                        c = a.whatSym(str);                        //
определяем символ по коду

                     /* if(c != '\n')
                              cout << "  Статический код " << str << "
принадлежит символу " << c << endl;
                        else if(c != ' ')
                              cout << "  Статический код " << str << "
принадлежит символу space" << endl;
                        else
```

```
                        cout << "  Статический код " << str << "
принадлежит символу \\n" <<  endl;

                    cout << "      Записываем полученный символ в
дерево" << endl;*/

                    str.clear();
               }



               c = t.processSym(c);                    // обрабаываем
символ
               t.findElem(c, 2);
               t.flag = 0;

               if (c != '\0' && !fout.eof()) {

                    if (flag) {
                         str = t.retCode();
                         /*if(c == '\n')
                              cout << "  Наден символ: \\n";
                         else if(c == ' ')
                              cout << "  Наден символ: space" ;
                         else
                              cout << "  Наден символ: " << c;

                         cout << " по коду " << str << endl;
                         cout << "    Увеличиваем вес символа" <<
endl;*/
                         str.clear();
                    }

                    flag = 1;

                    //cout << "      Проверяем и сортируем деерво" <<
endl;
                    t.reSum(2);
// упорядочиваем дерево
                    t.check1();
                    t.reSum(2);
                    t.check2();
                    t.reSum(2);
```

```
                    //fout << c;
                    *res += c;
               }
               /*else if( !fin.eof()){
                    cout << "  Данный символ является частью
динамического кода" << endl
                         << "  Продолжаем считывание" << endl;
               }*/
          }
     }
     cout << "Декодирование завершено\n";
}

char reading(ifstream & in) {
     char c;
     if (in.is_open()) c = in.get();
     else c = getchar();
     return c;
}

void printTree(int num, vector <node> & tree, int level) {
     if (tree[num].list.size() > 1) {
          if (tree[tree[num].right].list.size() >= 1)
printTree(tree[num].right, tree, level + 4);
          if (level) {
               for (int i = 0; i < level; i++) cout << " ";
               cout << " ";
          }
          if (tree[tree[num].right].list.size() >= 1) {
               cout << " /\n";
               for (int i = 0; i < level; i++) cout << " ";
               cout << " ";
          }
          cout << tree[num].list << "\n ";
          if (tree[tree[num].left].list.size() >= 1) {
               for (int i = 0; i < level; i++) cout << " ";
               cout << " " << " \\\n";
               printTree(tree[num].left, tree, level + 4);
          }
     }
     else if (tree[num].list.size() == 1) {
          if (level) {
               for (int i = 0; i < level; i++) cout << " ";
          }
```

```
            cout << tree[num].list << "\n ";
      }
}

void printDecodeTree(nodeDecode * p, int indent) {
      if (p != NULL) {
            if (p->right) {
                  printDecodeTree(p->right, indent + 4);
            }
            if (indent) {
                  for (int i = 0; i < indent; i++) cout << " ";
            }
            if (p->right) {
                  cout << " /\n";
                  for (int i = 0; i < indent; i++) cout << " ";
            }
            if (p->isNode == 1) cout << "ND";
            cout << p->letter << "\n ";
            if (p->left) {
                  for (int i = 0; i < indent; i++) cout << " ";
                  cout << ' ' << " \\\n";
                  printDecodeTree(p->left, indent + 4);
            }
      }

}
```

ПРИЛОЖЕНИЕ Б

СОДЕРЖАНИЕ ФАЙЛА DYNAMICHUFFMAN.H

```
#include <string>
#include <vector>

using namespace std;


class Tree {

private:

	struct node {

		char elem;		// хранимый эелмент
		bool isElem;		// явдяеься ли элементом (если нет, то это
узел)
		int weight;		// вес узла

		int left;		// индекс левого ответвления
		int right;		// индекс правого ответвления

		node() {
			left = right = weight = -1;
			elem = '\0';
		}
	};


	vector <node> t;			// дерево

	vector <int> p;			// вектора нужные для проверок
	vector  <int> e;			//

	string way;				// код элемента

	int parentEsc;			// родительнь Esc символа
	int len;				// длина вектора t
	int head;				// индекс головы дерева

	int nowElem = 0;			// индекс нахождения в дереве
	int Esc = 1;				// флаг = 1 если левый элемент Esc
```

```
//------------PUBLIC------------------

public:

	int flag;

	Tree() {
		node a;                 // создаем дерево, состоящее из Esc
символа
		a.isElem = true;
		a.weight = 0;
		head = 0;
		len = 0;
		parentEsc = 0;
		t.push_back(a);
	}

	//------------------------------------ добавляет новый символ в
дерево

	void addElem(char c) {

		node a;                     // создание листа с символом
		a.isElem = true;
		a.elem = c;
		a.weight = 1;
		t.push_back(a);
		len++;

		node b;                    // создание узла
		b.isElem = false;
		b.weight = 1;
		b.left = 0;                 // слева Esc символ
		b.right = len;              // справа созданый лист
		t.push_back(b);
		len++;

		if (len == 2) {                   // заменяем Esc на созданый
узел
			head = 2;
```

```
                parentEsc = 2;
            }
            else {
                t[parentEsc].left = len;
                parentEsc = len;
            }
        }

        //------------------------------------ ищет в дереве элемент с

        bool findElem(char c, int ind) {

            bool answer = false;

            if (head == 0) {              // пустое дерево
                return false;
            }

            int l = t[ind].left;
            int r = t[ind].right;

            if (t[l].isElem && t[l].elem == c) {          // если элемент,
то возвращаем true

                if (flag) {                       //  постоение кода
                    way = c;
                }
                else {
                    way = '0';
                }

                return true;
            }

            if (t[r].isElem && t[r].elem == c) {          // если элемент,
то возвращаем true

                if (flag) {                           //  постоение кода
                    way = c;
                }
                else {
                    way = '1';
                }
```

```
                return true;
          }

          if (!t[l].isElem) {                    // если узел, то идем в
него
                answer = findElem(c, l);

                if (answer) {                  //  постоение кода
                      way = '0' + way;
                }

          }

          if (!answer && !t[r].isElem) {         // если узел, то идем в
него
                answer = findElem(c, r);

                if (answer) {              //  постоение кода
                      way = '1' + way;
                }
          }

          return answer;
     }
     //-------------------------------------- добавляет 1 к весу
элемента с

     void addOne(char c) {

          int i;

          for (i = 0; i <= len; i++) {

                if (t[i].elem == c) {
                      (t[i].weight)++;
                      break;
                }
          }
     }
     //--------------------------------------  пересчитывает веса узлов
дерева

     void reSum(int ind) {
```

```
            int l = t[ind].left;
            int r = t[ind].right;

            if (!t[l].isElem) {
                  reSum(l);
            }

            if (!t[r].isElem) {
                  reSum(r);
            }

            t[ind].weight = t[l].weight + t[r].weight;

      }
      //--------------------------------------- функции возврата

      string retCode() {
            return way;
      }

      int retNowElem() {
            return nowElem;
      }

      int retEsc() {
            return Esc;

      }
      //--------------------------------------- проверка на соответсвтия
листьев в деревк

      void check1() {

            int i = 2;
            int j = -1;

            int lenE = 0;
            int lenP = 0;
            int next;

            p.push_back(2);
            e.push_back(2);
```

```
            while (i != parentEsc) {              // строим векторы листьев
и узлов
                  j++;
                  i = p[j];


                  int l = t[i].left;
                  int r = t[i].right;

                  if (t[r].isElem) {              // если лист, то добавлем
в вектор e
                        e.push_back(r);
                        lenE++;
                  }
                  else {                          // еслу узел добавляем в
вектор p
                        p.push_back(r);
                        lenP++;
                  }

                  if (t[l].isElem) {
                        e.push_back(l);
                        lenE++;
                  }
                  else {
                        p.push_back(l);
                        lenP++;
                  }
            }

            j = lenE;
            next = e[j];

            do {                                              // ищем неверный
лист
                  i = next;
                  if (j > 0) next = e[j - 1];

                  if (j != 0 && t[i].weight > t[next].weight) {
                        break;
                  }

                  j--;
```

```
            } while (j > 0);

            if (j <= 0) {            // если неверных листов нет, то
звершаем работу
                  p.clear();
                  e.clear();
                  return;
            }

            j = 0;

            do {                     // если найден, то ставим его на
верное место
                  next = e[j];

                  if (t[next].weight < t[i].weight) {
                        node a = t[next];
                        t[next] = t[i];
                        t[i] = a;
                        p.clear();
                        e.clear();
                        return;
                  }
                  j++;

            } while (j <= lenE);

            p.clear();
            e.clear();
      }

      //------------------------------------- проверяет упорядоченность
дерева

      void check2() {

            int i = 2;
            int j = 0;

            int r, l;
            int next;

            p.push_back(2);
```

```
            do {                            // составляем векторо вершин и
листов в правильном порядке

                  if (!t[i].isElem) {

                        r = t[i].right;
                        l = t[i].left;

                        p.push_back(r);
                        p.push_back(l);
                  }

                  j++;
                  i = p[j];

            } while (i != 0);

            j = len;

            do {                                    // проверяем
упорядоченность
                  i = p[j];

                  if (t[i].weight > t[p[j - 1]].weight) {
                        break;
                  }

                  j--;

            } while (j != 1);

            if (j == 1) {                       // если все верно, то
завершаем работу
                  p.clear();
                  e.clear();
                  return;
            }

            j = 0;

            do {                                // иначе ставим неправильный
элемент в нужное место
```

```
                next = p[j];


                if (t[next].weight < t[i].weight) {
                      node a = t[next];
                      t[next] = t[i];
                      t[i] = a;

                      if (next == parentEsc) {
                            parentEsc = i;
                      }

                      p.clear();
                      e.clear();
                      return;
                }

                j++;


          } while (j <= len);

          p.clear();
          e.clear();
     }

     //--------------------------------------  обрабатывает входной
символ из зикодированного файла
     char processSym(char c) {

          int l = t[nowElem].left;
          int r = t[nowElem].right;


          if (nowElem == 0) {               // находимся в Esc символе,
входной элемент символ
                flag = 1;
                addElem(c);                 // добавляем его в дерево
                nowElem = 2;
                Esc = 0;
                return c;
          }

          if (nowElem == 2) {
```

```
                way.clear();
            }



            if (c == '0' && t[l].isElem && l != 0) {      // если слева
символ
                way.push_back(c);
                t[l].weight++;                            // увеличиваем
его вес
                nowElem = 2;
                return t[l].elem;                         // возращаем его
            }

            else if (c == '1' && t[r].isElem && r != 0) { // если справа
символ (аналогично)
                way.push_back(c);
                t[r].weight++;
                nowElem = 2;
                return t[r].elem;
            }

            if (c == '0' && !t[l].isElem) {          // если слева узел,
идем в него
                way.push_back(c);
                nowElem = l;
            }

            else if (c == '1' && !t[r].isElem) {  // если справа узел идем
в него
                way.push_back(c);
                nowElem = r;
            }

            else if (c == '0' && t[l].isElem) {  // если слева Esc, то
идем в него
                way.push_back(c);                // следующий элемент
будет символ
                nowElem = l;
                Esc = 1;
            }

            return '\0';
        }
```

```
	//-------------------------------------- печатает дерево (ПКЛ)

	void print_Tree(int p, int level, int n) {
		if (p != -1)
		{
			print_Tree(t[p].right, level + 1, 0);
			for (int i = 0; i < level; i++) cout << "     ";

			if (n == 1)
				cout << "\\";
			else if (n == 0)
				cout << "/";

			if (t[p].isElem && t[p].elem != '\n')
				cout << t[p].elem << "(" << t[p].weight << ")" <<
endl;

			else if (t[p].isElem && t[p].elem == '\n')
				cout << "\\n(" << t[p].weight << ")" << endl;

			else
				cout << "S" << (int)t[p].elem << "(" << t[p].weight
<< ")" << endl;

			print_Tree(t[p].left, level + 1, 1);
		}
	}

	//--------------------------------------
};
```

ПРИЛОЖЕНИЕ В

СОДЕРЖАНИЕ ФАЙЛА SYMCODE.H

```
#ifndef SymCode_h
#define SymCode_h

#include <string>
#include <vector>

class Sym {

private:

	struct elem {
		char c;			// символ
		string code;		// статический код

		elem() {
			c = '\0';
			code = "00000000";
		}
	};

	int num = 0;			// длина вектора arr
	vector <elem> arr;		// вектор хранит символы и их коды


public:
	//-------------------------------------- задает код символу
	string code(char c) {
		int i = 0;
		int n = ++num;
		int j;
		elem a;
		a.c = c;

		for (i = 0; n > 0; i++) {	// код символа задается как
его индекс
							// в векторе arr в
двоичной системе
			j = n % 2;

			n /= 2;
```

```
                if (j)
                        a.code[i] = '1';

        }

        arr.push_back(a);

        return a.code;
    }

    //-------------------------------------- записывает в файл символы
и их коды

    void writeFile(ofstream& file) {

        int i;

        for (i = 0; i < num; i++) {

                file << arr[i].code << " " << arr[i].c << endl;
        }

    }
    //-------------------------------------- считывает из фала символы
и их коды

    int readFile(ifstream& file) {
        char sym;
        elem a;

        while (!file.eof()) {                        //производится
считывание ключей
                char tmp = NULL;    //для считывания ненужных символов
                string codee;        //считывание кода символа
                file.get(sym);
                if (sym == '\n') {
                        file.get(sym);
                        if (sym != ' ') break;
                        else sym = '\n';
                }
                else file.get(tmp);
                file >> codee;
                if (!isdigit(codee[0])) return 1;
                file.get(tmp);
```

```
                a.code = codee;
                a.c = sym;
                num++;
                arr.push_back(a);
          }

          return 0;

     }


     //-------------------------------------- находит символ по коду

     char whatSym(string code) {

          int i;

          for (i = 0; i < num; i++) {

                if (arr[i].code == code)
                     return arr[i].c;


          }

          return '\0';

     }

     //-------------------------------------- печатает коды символов

     void print_Codes(ofstream & out) {
          int i = 0;

          while (i < num) {
                out << arr[i].c << " " << arr[i].code << endl;
                i++;
          }
     }
};

#endif
```