Федеральное агентство связи   
Федеральное бюджетное образовательное учреждение высшего образования   
«Сибирский государственный университет телекоммуникаций и информатики»   
(ФГОУ ВПО «СибГУТИ»)

Факультет ИВТ

Кафедра вычислительных систем

**Отчет по курсовой работе**

**Сжатие данных**

Выполнил:

студент гр. ИВ - 521

Прокопенко Р. П.

Проверила:

Старший преподаватель Кафедры ВС

Перышкова Е.Н.

Новосибирск

2016

**Задание**

Алгоритм Лемпела – Зива (Lempel – Ziv) LZ77

Реализовать программу lz77compress сжатия текстовых файлов на английском языке алгоритмом Зива-Лемпела. Сжатие осуществляется с аргументом командной строки -c (compress), а распаковка – с аргументом -d (decompress). Опция -o указывает имя выходного файла. Например:



Критерии оценки:

* Оценка «хорошо»: реализован алгоритм сжатия, для записи кодов в файл используются структуры данных.
* Оценка «отлично»: можно задать любой размер словаря и буфера, для формирования файлового элемента используется битовый массив (как описано в общей информации к разделу 4).

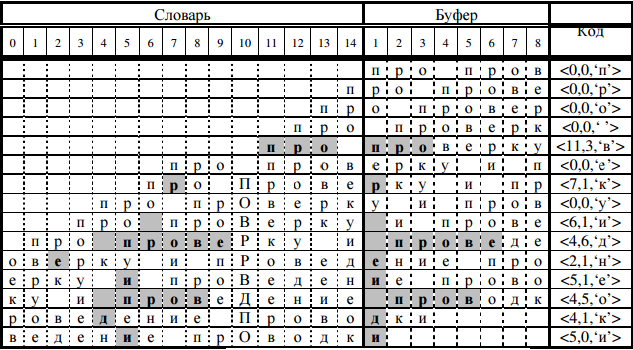
Указание к выполнению задания

LZ77 использует скользящее по сообщению окно. Метод кодирования согласно принципу скользящего окна учитывает уже ране встречавшуюся информацию, то есть информацию, которая уже известна для кодировщика и декодировщика (второе и последующие вхождения некоторой строки символов в сообщении заменяются ссылками на ее первое вхождение). Окно состоит из двух частей – словаря (большая часть) и буфера. Первая, большая по размеру, включает уже просмотренную часть сообщения. Вторая, меньшая по размеру, содержит еще незакодированные символы входного потока. Алгоритм пытается найти в словаре фрагмент, совпадающий с содержимым буфера. Алгоритм LZ77 выдает коды, состоящие из трех элементов:

* смещение подстроки, совпадающей с началом содержимого буфера, относительно начала словаря;
* длина этой подстроки;
* первый символ буфера, следующий за подстрокой.

В конце итерации алгоритм сдвигает окно на длину равную длине подстроки, обнаруженной в словаре.

Рассмотрим алгоритм на примере. Пусть нам необходимо закодировать строку: «про проверку и проведение проводки». При кодировании будем использовать окно размером 23 символа, где первые 15 символов будут словарем, а следующие 8 – буфером.



Закодированный текст «про проверку и проведение проводки» представлен на рисунке 1.



Рисунок - Закодированный текст

**Анализ задачи**

Можно сказать, что алгоритмы семейства LZ представляют собой более сложное обобщение простого и интуитивного способа сжатия данных, используемого в RLE. Для понимания данного алгоритма необходимо разобраться с двумя его составляющими: принципом скользящего окна и механизмом кодирования совпадений.

В данной работе требуется написать программный продукт, который позволяет не только кодировать файл по алгоритму lz77, но и который сможет выполнить декодирование уже закодированных файлов.

Принцип скользящего окна

Метод кодирования согласно принципу скользящего окна учитывает уже ранее встречавшуюся информацию, то есть информацию, которая уже известна для кодировщика и декодировщика (второе и последующие вхождения некоторой строки символов в сообщении заменяются ссылками на её первое вхождение).

Благодаря этому принципу алгоритмы LZ иногда называются методами сжатия с использованием скользящего окна. Скользящее окно можно представить в виде буфера (или более сложной динамической структуры данных), который организован так, чтобы запоминать «сказанную» ранее информацию и предоставлять к ней доступ. Таким образом, сам процесс сжимающего кодирования согласно LZ77 напоминает написание программы, команды которой позволяют обращаться к элементам «скользящего окна», и вместо значений сжимаемой последовательности вставлять ссылки на эти значения в «скользящем окне». Размер скользящего окна может динамически изменяться и составлять 2, 4, 8, 16, 32 килобайта и так далее. Следует также отметить, что размер окна кодировщика может быть меньше или равен размеру окна декодировщика, но не наоборот.

Приведенное выше сравнение процесса кодирования с «программированием» может натолкнуть на преждевременный вывод о том, что алгоритм LZ77 относится к [методам контекстного моделирования](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D1%8B_%D1%81%D0%B6%D0%B0%D1%82%D0%B8%D1%8F_%D1%81_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%D0%BC_%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F&action=edit&redlink=1). Поэтому следует отметить, что алгоритм LZ77 принято классифицировать как метод словарного сжатия данных, когда вместо понятия «скользящего окна» используется термин «динамического словаря».

Механизм кодирования совпадений

Перед тем, как перейти к рассмотрению механизма кодирования, уточним понятие *совпадения* (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *match*). Рассмотрим последовательность из N элементов. Если все элементы последовательности уникальны, то такая последовательность не будет содержать ни одного повторяющегося элемента, или, иначе говоря, в последовательности не найдется хотя бы двух равных друг другу или совпадающих элементов.

В стандартном алгоритме LZ77 совпадения кодируются парой:

* длина совпадения (match length);
* смещение (offset) или дистанция (distance).

В продолжение уже приведенной аналогии с программированием отметим, что в большинстве статей, посвященных алгоритму LZ77, кодируемая пара трактуется именно как команда копирования символов из скользящего окна с определенной позиции, или дословно как: «Вернуться в буфере символов на *значение смещения* и скопировать *значение длины* символов, начиная с текущей позиции».

Хотя для приверженцев [императивного программирования](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) такая интерпретация может показаться интуитивно понятной, она, к сожалению, мало говорит о сущности алгоритма LZ77 как метода сжатия. Особенность данного алгоритма сжатия заключается в том, что использование кодируемой пары *длина-смещение* является не только приемлемым, но и эффективным в тех случаях, когда значение длины превышает значение смещения.

Пример с командой копирования не совсем очевиден: «Вернуться на 1 символ назад в буфере и скопировать 7 символов, начиная с текущей позиции». Каким образом можно скопировать 7 символов из буфера, когда в настоящий момент в буфере находится только 1 символ? Однако следующая интерпретация кодирующей пары может прояснить ситуацию: каждые 7 последующих символов совпадают (эквивалентны) с 1 символом перед ними.

Это означает, что каждый символ можно однозначно определить, переместившись назад в буфере — даже если данный символ еще отсутствует в буфере на момент декодирования текущей пары *длина-смещение*. Такая кодируемая пара будет представлять собой многократное (определяемое значением смещения) повторение последовательности (определяемой значением длины) символов, что представляет собой более общую форму [RLE](https://ru.wikipedia.org/wiki/RLE).

Недостатки

* невозможность кодирования подстрок, отстоящих друг от друга на расстоянии, большем длины словаря;
* длина подстроки, которую можно закодировать, ограничена размером буфера;
* малая эффективность при кодировании незначительного объёма данных.

Блок-схема работы программы:

Какой тип операции?

Да

Данные корректны?

Ввод типа операции

Начало

Нет

2

(декодирование)

1

(кодирование)

Ввод имен декодируемого файла и файла для записи результата

Ввод имен кодируемого файла и файла для записи результата

Процедура декомпрессии файла

Оставить размеры буфера и словаря по умолчанию?

Да

Ввод размеров буфера и словаря

Процедура записи данных в файл

Нет

3

2

1

3

2

1

Конец

Вывод статистики по результатам работы

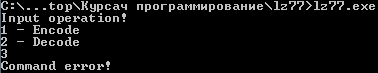
Вывод статистики по результатам работы

Процедура записи данных в файл

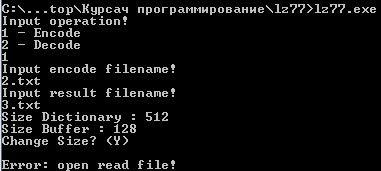
Процедура сжатия файла

**Тестовые данные**

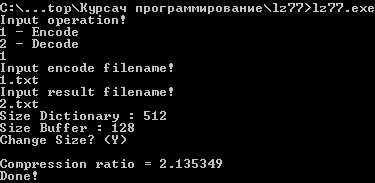
1. Введение некорректной команды для выбора операции:

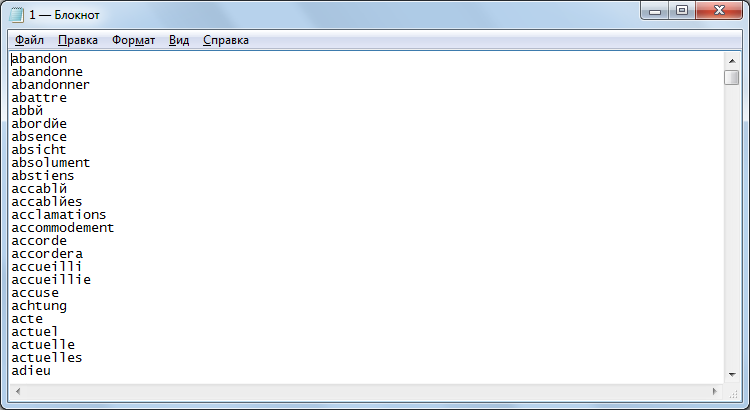


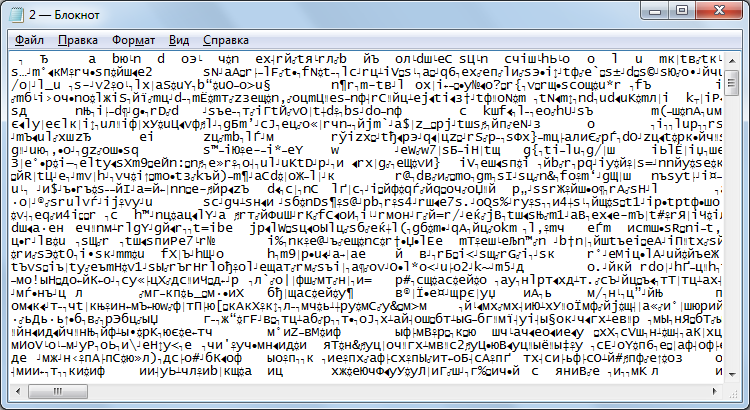
1. Введение несуществующего имени кодируемого или декодируемого файла:



1. Кодирование файла:

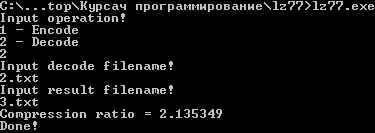


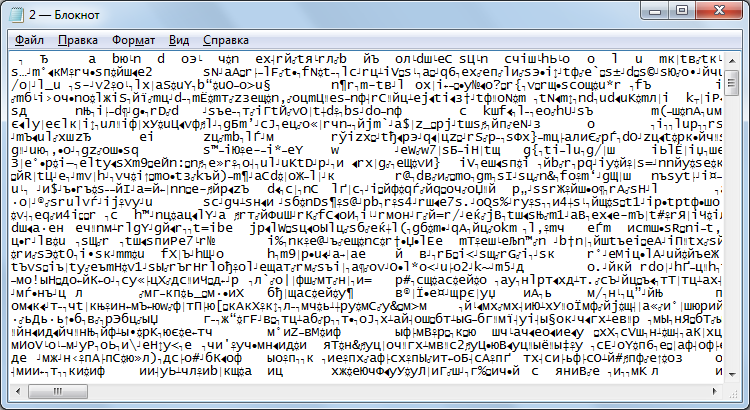


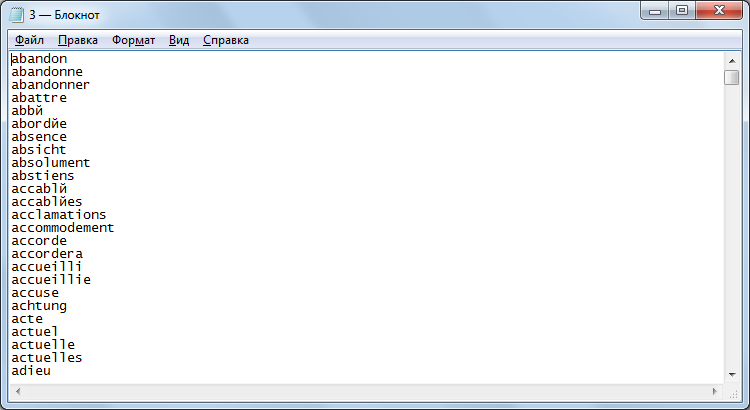




1. Декодирование файла:

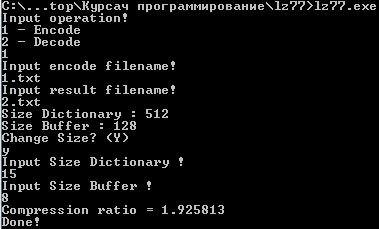


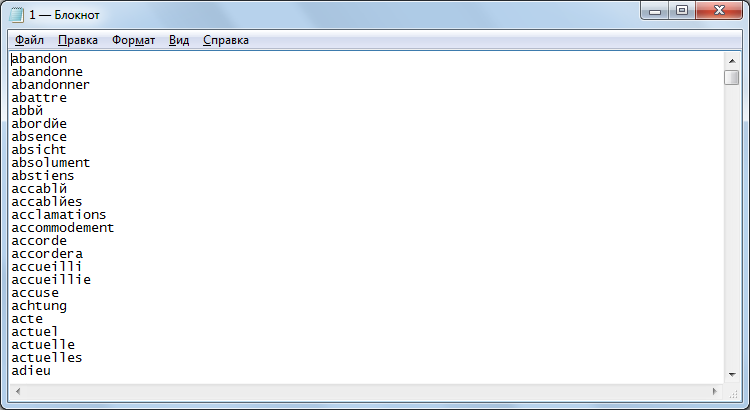


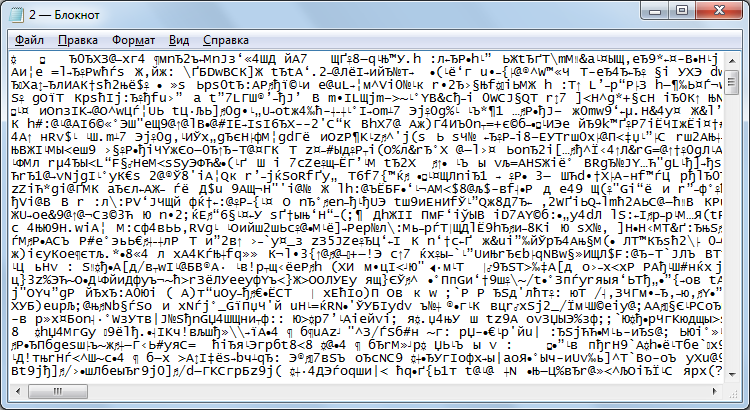




1. Сжатие файла с измененными размерами буфера и словаря:









**Листинг программы**

**main.c**

#include "lz77.h"

int main (int argc, char \*\* argv)

{

FILE \*fin, \*fout;

int M, N; // размеры задаваемого словаря и буфера

char finname[30], foutname[30];

char ch, op = 0;

M = 512; N = 128; // по умолчанию

if (argc > 3) //если заданы параметры в комм. строке

{

// проверить операцию

if (strcmp(argv[1], "-c") == 0) op = '1';

else if (strcmp(argv[1], "-d") == 0) op = '2';

else

{

printf("Command error!\n");

return 1 ;

}

// взять имена файлов

strcpy(&finname , argv[2]);

printf("Source File : %s\n", finname);

strcpy(&foutname , argv[3] );

printf("Result File : %s\n", foutname);

}

// иначе задаем через меню

else

{

printf("Input operation!\n1 - Encode\n2 - Decode\n");

scanf("%c",&op);

if (op == '1') // выбор операции кодирования

{

printf("Input encode filename!\n");

scanf("%s",&finname);

printf("Input result filename!\n");

scanf("%s",&foutname);

}

else if (op == '2') //декодирования

{

printf("Input decode filename!\n");

scanf("%s",&finname);

printf("Input result filename!\n");

scanf("%s",&foutname);

}

else

{

printf("Command error!\n");

return 2;

}

}

// при воборе кодирования

if (op == '1')

{

printf("Size Dictionary : %d\n", M);

printf("Size Buffer : %d\n", N);

// приглашение изменить размеры словаря и буфера

printf("Change Size? (Y) \n", M);

fflush(stdin);

scanf("%c", &ch);

if (ch == 'Y' || ch == 'y')

{

printf("Input Size Dictionary !\n");

scanf("%d", &M);

printf("Input Size Buffer !\n");

scanf("%d", &N);

}

}

//открытие исходного файла

fin = fopen(&finname, "rb");

if (fin == NULL)

{

printf("Error: open read file!");

return 3;

}

// открытие файла для записи результат

fout = fopen(&foutname, "wb");

if (fout == NULL)

{

printf("Error: open write file!");

fclose(fin);

return 3;

}

if (op == '1') { // вызов ф-ции кодирования

Encode( fin, fout, M, N );

}

else { //декодирования

Decode(fin, fout);

}

// закрыть файлы

fclose(fin);

fclose(fout);

printf("Done!\n");

system("pause");

return 0;

}

**lz77.h**

#include <stdio.h>

#include <malloc.h>

#include <string.h>

#include <math.h>

//структура кода

struct Code {

int num; //индекс в словаре

int len; //длина найденной подстроки

char c; //символ стоящий за подстрокой из буфера

};

// прототипы вызываемых ф-ций кодирования/декодирования

void Encode(FILE\* , FILE\* , int , int);

void Decode(FILE\* , FILE\*);

**lz77.c**

#include "lz77.h"

// установить служебную информацию в выходной массив при кодировании

void SetInfo(int\* array, int M, int N) {

array[0] = M; array[1] = N;

}

// взять сл. информацию из входного массива при декодировании

void GetInfo(int\* array, int\* M, int\* N) {

\*M = array[0]; \*N = array[1];

}

// установка массива бит в выходной массив при кодировании

// array - выходной массив

// offset- смещение в битах от начала

// value - устанавливаемое значение

// len - и его длина в битах

int SetBits(int \*array, int offset, unsigned int value, int value\_len) {

int i, d ;

i = offset / 32; //индекс для массива из int

d = offset % 32; //находящийся остаток в битах

// обнулить все биты числа выходящие за длину битовой последовательности

value &= (int) pow(2.0, (float)value\_len) - 1;

// добавить в выходной массив

// сдвинув влево на остаток в битах находящийся уже в массиве

array[i] += value << d;

// если вставили меньше чем длина битовой последовательности

if ( 32 - d < value\_len ) {

// дописываем последовательность в сл. ячейку массива

array[i+1] += value >> (32 - d );

}

// изменить смещение на длину вставленной последовательности бит

return offset + value\_len;

}

// считать массив бит из входного массива при декодировании

// array - входной массив

// offset - смещение в битах от начала

// value - аддрес переменной для взятия битовой последовательности

// len - и его длина в битах

int GetBits(int \*array, int offset, unsigned int\* value, int value\_len) {

int i, d;

i = offset / 32;

d = offset % 32;

\*value = 0;

// взять из массива значение сдвинутое на осаток смещения

\*value = array[i] >> d;

// если взяли меньше чем длина в битах

if (32 - d < value\_len)

{

// оставить только те биты которые уже взяли

// остальные сбросить в 0

\*value &= (int) pow(2.0, (float)(32-d)) - 1;

// взять оставшуюся часть

\*value += array[i+1] << (32 - d );

}

// обнулить все биты кроме тех что взяли

\*value &= (int) pow(2.0, (float)value\_len) - 1;

// изменить смещение на взятое число бит

return offset + value\_len;

}

// взять размеры файла

int GetFileSize(FILE \* f)

{

int size;

fseek(f, 0, SEEK\_END); // сместится в конец

size = ftell(f); // взять смещение в байтах от начала

fseek(f, 0, SEEK\_SET); // вернутся в начало файла

return size; // вернуть размер в байтах

}

// поиск наибольшей подстроки буфера в словаре

struct Code Find(char\* dict, char \*buff , int M, int N) {

int i, num, tmp;

char \*pstr;

struct Code code; // возвращаемая структура кода

pstr = malloc(N+1); // выделить память под максимальную подстроку равную длине буфера - 1

for (i = 0; i < N + 1; i++) {

pstr[i] = 0; // обнулить

}

tmp = 0; // временный(предыдущий) адрес найденной подстроки в словаре

// исследовать на всю длину буфера - 1

// последний символ должен быть записан за подстрокой

// в случае если найдется подстрока максимальной длины

// потому N - 1

for(i = 0; i < N - 1 ; i++)

{

// взять символ из буфера во временную подстроку

// увеличая ее в длине и проводя поиск

pstr[i] = buff[i];

// найти его расположение в словаре

num = strstr(dict, pstr);

// если подстрока найдена за пределами словаря в самом буфере

if (num >= buff)

{

// и нет предыдущего адреса подстроки

// то символ встречается впервые

// запишем её код

if (tmp == 0) {

// последний символ за подстрокой

code.c = pstr[i];

// длина и смещение в словаре в 0

code.len = 0;

code.num = 0;

// освободить память

free(pstr);

// вернуть код

return code;

}

// иначе если была найдена подстрока до того как пришли в буфер

else {

// взять текущую итерацию за длину

code.len = i;

// вычислить смещение в словаре

code.num = M - (num - tmp);

// если вышли за пределы длины файла

if (pstr[i] == -1) {

// взять предыдущий символ

code.c = pstr[i-1];

// отнять длину на 1

code.len--;

}

// иначе взять текущий символ

else code.c = pstr[i];

// освободить подстроку

free(pstr);

// вернуть код

return code;

}

}

// пока идет цикл

// сохраняем

else tmp = num;

}

// если цикл завершен

// то вся подстрока длиной буфера - 1 символ найдена в словаре

if (tmp == num && i == N - 1) {

// запишем длину

code.len = N - 1;

// последний символ в буфере

code.c = buff[code.len];

// смещение в словаре от начала

code.num = M - ((int)buff - num);

}

// освобождаем память

free(pstr);

// возвращаем код

return code;

}

// ф-ция кодирования файла

void Encode(FILE \*fin, FILE \*fout, int M, int N) {

char \*input; // входной и выходной массивы

char \*output;

int offset, Mlen, Nlen;

struct Code code;

int i, k, num, fsize,fosize ;

fsize = GetFileSize(fin); // размер файла

output = malloc(fsize \* 4); // выделим память под выходной массив

for(i = 0; i < fsize \* 4; i++) {

output[i] = 0; // обнулить

}

SetInfo(output, M, N); // установить служебную информацию

offset = 64; // смещение в битах от начала на 64 бита больше

// входной массив

input = malloc(fsize + M + 1);

input[fsize + M] = -1;

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

input[i] = -1; // инициализируем в -1

}

// считать файл сместившись в массиве на длину словаря

fread(&input[M], 1, fsize, fin);

// вычислить длину словаря и массива в битах

Mlen = (log((float)M) / log((float)2) + 0.999999);

Nlen = (log((float)N) / log((float)2) + 0.999999);

//процесс кодирования

code.len = 0;

for(i = 0; i < fsize; i += code.len + 1) {

// поиск записываемого кода

code = Find(&input[i], &input[i + M], M, N);

// установка битового массива в выходной массив

offset = SetBits(output, offset, code.num, Mlen);

offset = SetBits(output, offset, code.len, Nlen);

offset = SetBits(output, offset, (int)code.c, 8);

}

// найти длину выходного файлв в байтах

fosize = offset / 8;

// если есть остаток, увеличим на 1

if (offset % 8)

fosize++;

// записать выходной файл

if (fwrite(output, 1, fosize, fout))

{

// освободить память

free(input);

free(output);

}

// расчитать коэфф. сжатия

printf("Compression ratio = %f\n",(float)fsize / (float)fosize);

}

// ф-ция декодирования

void Decode(FILE \*fin, FILE \*fout) {

char \*input;

char \*output;

int offset, Mlen, Nlen, M, N;

struct Code code;

int i, j, k, num, fsize;

fsize = GetFileSize(fin); // размер входного файла

input = malloc(fsize + 1); // выделить память под массив

// проинициализировать в 0

for(i = 0; i < fsize + 1; i++) {

input[i] = 0;

}

// считать файл

fread(input, 1, fsize, fin);

M = 0;

N = 0;

// взять служебную информацию о размерах словаря и буфера

GetInfo(input, &M, &N);

offset = 64; // смещение в битах от начала на 64 бита больше

// выделить память под выходной массив

output = malloc(fsize \* 4);

for(i = 0; i < fsize \* 4; i++) {

output[i] = -1; // инициализация в -1

}

// расчет длины словаря и буфера в битах

Mlen = (log((float)M) / log((float)2) + 0.999999);

Nlen = (log((float)N) / log((float)2) + 0.999999);

//процесс декодирования

i = 0;

while ((fsize - offset / 8) \* 8 >= Mlen + Nlen + 8) {

// взять значения из входного массива

offset = GetBits(input, offset, &code.num, Mlen);

offset = GetBits(input, offset, &code.len, Nlen);

offset = GetBits(input, offset, (int)&code.c, 8);

// если подстроки нет в словаре

if (code.num == 0 && code.len == 0)

// запишем в буфер как есть

output[i + M] = code.c;

else

{

// перепишем всю подстроку в буфер

for(k = 0; k < code.len; k++) {

output[i + M + k] = output[i + code.num + k];

}

// дописать в конце подстроки символ

output[i + M + code.len] = code.c;

}

// сместить окно на длину подстроки

i += code.len + 1;

}

// записать выходной файл

if (fwrite(&output[M], 1, i, fout))

{

// освободить память

free(input);

free(output);

}

// рассчитать коэфф. сжатия

printf("Compression ratio = %f\n",(float)i / (float)fsize);

}