Министерство образования и науки Российской Федерации

Санкт-Петербургский Политехнический Университет Петра Великого

—

Институт кибербезопасности и защиты информации

ЛАБОРАТОРНАЯ РАБОТА № 7

«Кодирование и упаковка данных»

1. по дисциплине «Основы информационной безопасности»
2. Выполнил
3. студент гр. 4851004/10001 Веремейчук Я.Ю.

<подпись>

1. Преподаватель
2. асс. преподавателя Панков И.Д.

<подпись>

1. Санкт-Петербург
2. 2022
3. Цель работы

Приобретение навыков по защите информации с помощью методов кодирования, получение прикладных знаний в области исследования п реализации алгоритмов упаковки данных.

1. Постановка задачи

Разрабатываемая программа должна реализовывать два алгоритма кодирования и сжатия информации: алгоритм Фано и алгоритм Зива-Лемпеля.

1. Теоретические исследования

Задача кодирования данных формально представляется следующим образом. Заданы алфавиты А = {а1, а2, … аn} и В = {b1, b2, … bm}, а также функция F: S -> B\*, где S – подмножество в алфавите А, SA\*, А\* – множество всех слов в алфавите А, В\* – множество всех слов в алфавите В.

Функция F называется кодированием, элементы множества S – сообщениями, элементы , , B\* – кодами соответствующих сообщений. Обратная функция F^(-1) (если она существует) называется декодированием. Если |B| = m, то F – m-ичное кодирование.

Типичная задача кодирования формулируется следующим образом: при заданных алфавитах А, В и множестве сообщений S найти кодирование F, которое обладает определенными свойствами, то есть удовлетворяет заданным ограничениям, и оптимально в некотором смысле. Критерий оптимальности, как правило, связан с минимизацией длин получаемых кодов. Таким образом, кодирование – преобразование информации, в том числе с защитной целью, а упаковка (сжатие, архивирование) является побочным эффектом кодирования.

Сжатие (упаковка) данных – кодирование, позволяющее построить без потери данных код сообщения меньшей длины по сравнению с исходным. Качество сжатия определяется коэффициентом сжатия, показывающим, насколько закодированное сообщение короче исходного.

Сжатие бинарных файлов дольше и сложнее сжатия текстовых файлов. Для более подробного изучения вопроса был создан файл в формате .doc (рисунок 1), который после переименования стал файлом формата .txt (рисунок 2). Далее в программе Notepad++ был использован плагин HEX-Editor, позволяющий рассмотреть код получившегося файла (рисунок 3).

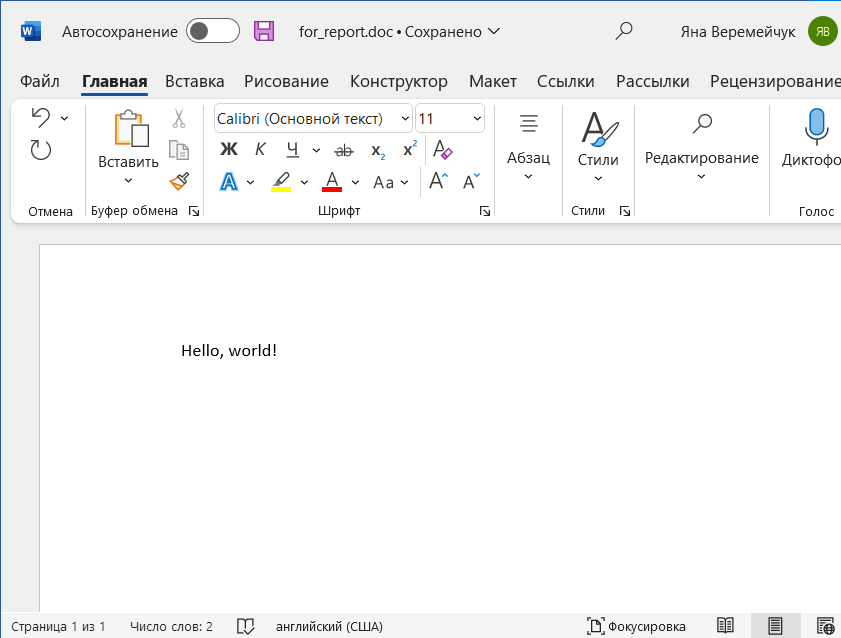


Рисунок – Исходный вид файла

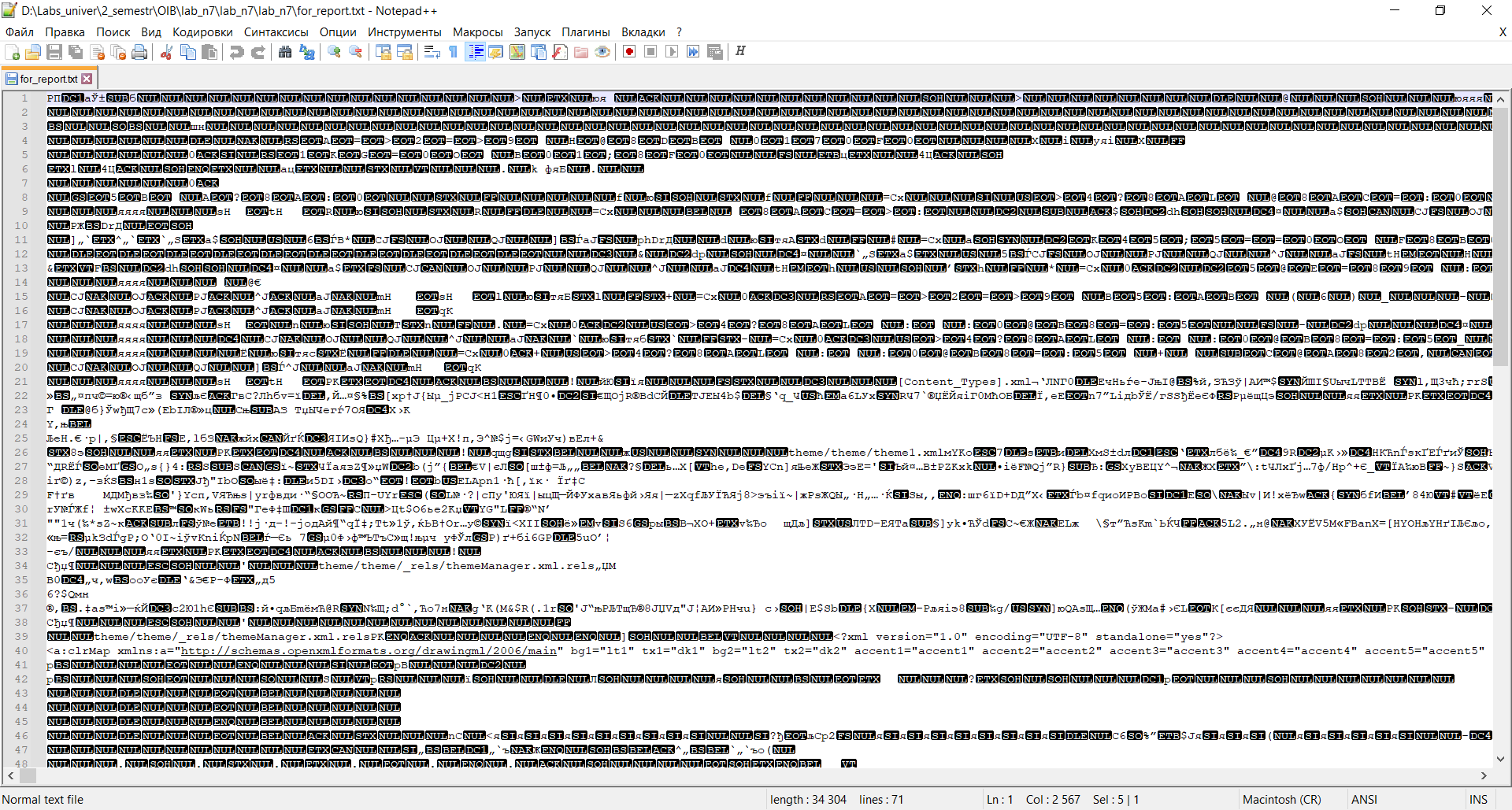


Рисунок – Вид файла после изменения формата

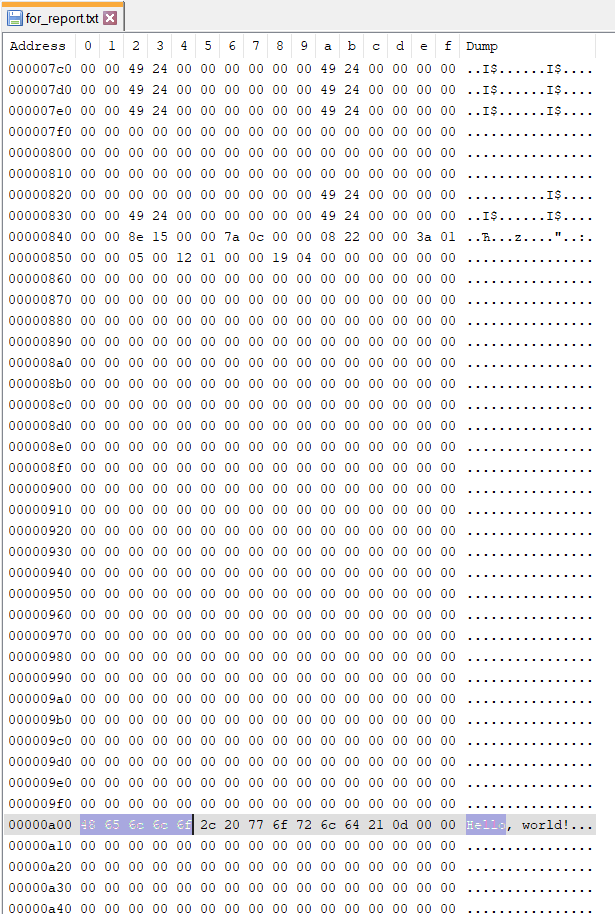


Рисунок – Вид файла в HEX-представлении

Можно заметить, что помимо фразы «Hello, world!» файл хранит массу дополнительной информации. И если речь идет лишь о кодировании текста, то оптимальным вариантом для его хранения будет файл формата .txt. Для проверки факта того, что файл формата .txt хранит исключительно текст, был создан новый файл данного формата и рассмотрены его свойства (рисунок 4) и представление в HEX (рисунок 5). Из свойств видно, что файл занимает ровно 12 байт – количество символов в записанной строке. А в HEX-представлении можно увидеть код этой строки.

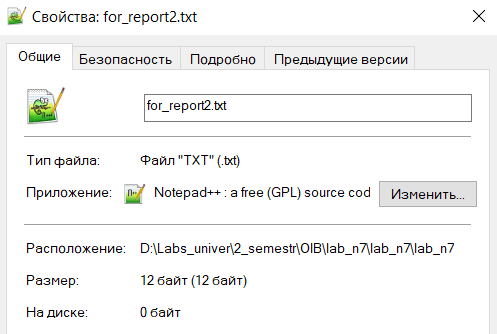


Рисунок – Свойства файла формата .txt

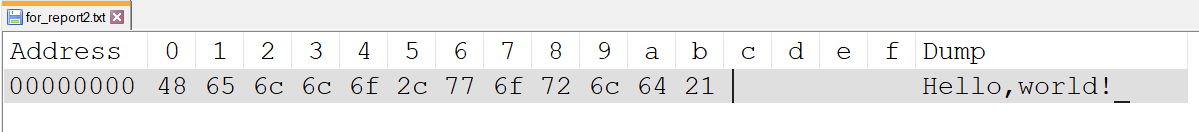


Рисунок – Вид файла в HEX-представлении

Различают статические методы сжатия и адаптивные. Статические методы создают отображение из множества букв алфавита во множество кодовых слов до начала процесса кодирования, а при сжатии каждая буква алфавита заменяется соответствующим кодовым словом. Для построения такого отображения до начала сжатия необходим предварительный проход по сообщению (файлу), чтобы собрать необходимую информацию о сообщении. При этом в выходной поток должны быть записаны данные о собранной статистике, чтобы декодер смог расшифровать сжатые данные.

Адаптивные методы не нуждаются в предварительном просмотре сообщения, так как они динамически меняют схему кодирования в зависимости от исходных данных. Такие алгоритмы не требуют передачи в выходной поток информации об использованном кодировании. Вместо этого декодер, считывая кодированный поток, изменяет схему кодирования, начиная с некоторой предопределенной. Адаптивное кодирование может дать большую степень сжатия, поскольку могут быть учтены локальные изменения частот.

Одним из статических методов сжатия является алгоритм Фано, который выполняется следующим образом. N букв входного алфавита располагаются в порядке убывания их вероятностей. Далее последовательность букв разбивается на две группы так, чтобы суммарные вероятности букв в каждой из групп были как можно более близки друг другу. Буквам из одной группы в качестве первого символа кодового слова приписывается символ 0, буквам из второй группы – 1. По тому же принципу каждая из групп снова разбивается на две подгруппы, и это разбиение определяет значение второго символа кодового слова.

Процедура продолжается до тех пор, пока все множество групп не будет разбито на отдельные буквы. В результате каждой из букв будет сопоставлено кодовое слово из нулей и единиц. Чем более вероятно появление буквы во входном потоке, тем быстрее она образует самостоятельную группу п тем более коротким словом она будет закодирована. Это обстоятельство обеспечивает высокую экономность кода Фано. Данный алгоритм позволяет построить близкую к оптимальной схему кодирования.

Арифметическое кодирование основано на соответствии букв алфавита вероятностным интервалам в промежутке [0, 1), длины которых равны вероятностям букв.

Кодирование реализуется путем сужения исходного интервала в зависимости от приходящей буквы. При получении первой буквы сообщение кодер определяет, какой ей соответствует интервал, и сужает исходный интервал [0, 1) до него. При получении следующей буквы кодер определяет ее интервал и ставит этому интервалу в соответствии новый, внутри суженного на предыдущем шаге. Новый интервал в пропорции к суженному будет таким же, как интервал буквы в пропорции к исходному интервалу.

При получении следующих символов интервал снова сужается. После сужения его в последний раз любое число, лежащее внутри полученного интервала, будет являться кодом сообщения.

Алгоритм LZ — адаптивный метод кодирования и не требует предварительного просмотра сообщения.

Изначально в словаре определена пустая строка, имеющая номер 0. Поступающий первый символ приписывается к пустой строке, образуя тем самым новую строку. Эта строка заносится в выходной поток. Также строка записывается в словарь под номером 1. Фактически сохраняется не строка, а номер пустой строки и сам добавляемый символ. При поступлении следующего символа (то есть строки, состоящей из одного символа) смотрится, есть ли эта строка в словаре. Если она обнаружена в словаре, то к этой строке добавляется следующий символ из входного потока. Полученная в результате новая строка проверяется на наличие в словаре. Если ее нет, то она записывается в словарь под номером, следующим после последнего на текущий момент, в виде пары (номер образующей строки; дополнительный символ). Эта же пара записывается в выходной поток. Если же эта строка уже имеется в словаре, то к ней приписывается следующий символ из потока, и вновь проверяется наличие полученной строки в словаре. Так продолжается до тех пор, пока новая строка не будет записана в словарь. Затем читается следующий символ из входного потока и действия повторяются. Каждая новая пара, добавляемая в словарь, записывается в выходной поток.

При декодировании после получения первой пары (номер пустой строки (0); первый символ) декодер записывает ее в словарь, который изначально пуст, а в выходной поток записывает первый символ. После получения n-ой пары (номер составляющей строки k; символ) декодер строит строку, состоящую из символа и стоящей перед ним строки, находящейся в словаре под номером k, которая в свою очередь тоже состоит из символа и стоящей перед ним строки. Декодер приписывает в начало строки по символу, пока не дойдет до строки, состоящей из пустой строки и символа. Этот символ ставится в начало строки, и полученная в результате строка записывается в выходной поток.

Данный метод эффективен для сжатия текстов, так как в них присутствует большое количество повторений одних и тех же последовательностей символов (окончаний, суффиксов слов и т.п.) и слов. Но LZ-кодирование не подходит для малых объемов текста, ведь на выходе может получиться строка, которая содержит больше символов, чем исходная. В подобных случаях кодирование не имеет смысла.

Алгоритм LZW был предложен Терри Уэлчем в 1984 г., и доработка по сравнению с алгоритмом LZ заключается в следующем: при заполнении словаря сброс словаря не производится и новые фрагменты в словарь не добавляются, пока степень сжатия данных остается «достаточно» высокой. Степень «достаточности» определяется субъективно, но простейший вариант «достаточности»: пока степень сжатия не меньше, чем на момент заполнения словаря. В этом случае резко возрастает скорость компрессии, поскольку операция вставки новых фрагментов в словарь не производится.

1. Описание решения

Для удобства использования алгоритмов кодирования работа была выполнена в виде «библиотек». То есть каждый из алгоритмов может быть применен в других проектах.

Разрабатываемая программа должна поддерживать запуск через командную строку. Для этого был использован оператор switch-case, который, в зависимости от выбора пользователя, осуществляет переход в необходимый алгоритм кодирования или декодирования.

При кодировании по Фано считывается текст из файла, каждый символ которого записывается в виде структуры, содержащей в себе информацию о значении этого символа, количестве его повторений, а также кода, который будет присвоен после анализа всего текста. После этого массив структур сортируется по количеству повторений символа, и происходит рекурсивное кодирование символов. Далее снова происходит чтение исходного файла и запись во временный массив соответствующего для каждого символа полученного кода. Далее происходит упаковка и запись в файл. Первая строка в файле – это длина зашифрованного текста (бит), вторая – это размер алфавита, третья - это количество символов сжатия, получившихся в ходе кодирования, далее записывается каждый символ алфавита и его код, далее сам зашифрованный текст, а затем биты, которые «не влезли» в стандартный размер char (эта строка может быть пустой, если общее число получившихся бит кратно 8). Имея такую информацию, декодер без труда может прочитать закодированный текст.

Декодер преобразует символы сжатия в бинарный код, добавляет биты, которые «не влезли», и подбирает символы из алфавита с применением условия Фано: никакое кодовое слово не является началом другого кодового слова.

Для случая, когда в исходном файле записан лишь один символ, были написаны отдельные функции кодирования во избежание ошибок.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок – Вид структуры, в которой хранится каждый новый символ при кодировании по алгоритму Фано

LZ алгоритм считывает данные из файла, записывая каждый новый символ в словарь, а находя символы, которые уже есть в словаре, алгоритм записывает их в так называемый «префикс», формируя сочетания букв. Полученный словарь записывается в выходной файл по порядку записанных элементов. Словарь хранится в виде массива структур, каждый элемент словаря хранит номер образующей строки, символ и слово, получившееся путем «склеивания» символов.

При декодировании программа вновь формирует словарь. Каждый полученный символ, который встречался ранее, приписывается к слову. Для вывода результата распечатываются слова словаря.

1. Тестирование и результаты работы программы

Был создан текстовый файл (рисунок 7). Далее файл был закодирован по алгоритму Фано. Результат работы программы представлен на рисунке 8. Файл, который получился на выходе, можно увидеть на рисунке 9. Далее файл был раскодирован (рисунок 10).

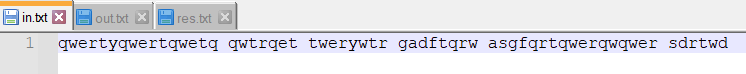


Рисунок – Пример тестового текстового файла

Изображение выглядит как текст

Автоматически созданное описание

Рисунок – Результат работы программы при кодировании по алгоритму Фано

Изображение выглядит как текст, стол

Автоматически созданное описание

Рисунок – Файл, получившийся после кодирования по алгоритму Фано

Изображение выглядит как текст

Автоматически созданное описание

Рисунок – Файл, получившийся после декодирования по алгоритму Фано

Далее файл был закодирован по алгоритму LZ. Результат работы программы представлен на рисунке 11. Файл, который получился на выходе, можно увидеть на рисунке 12.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок – Результат работы программы при кодировании по алгоритму LZ

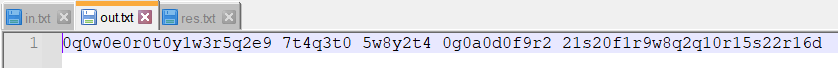


Рисунок – Файл, получившийся после кодирования по алгоритму LZ

Далее алгоритм Фано и алгоритм LZ были протестированы на разных типах файлов. С результатами можно ознакомиться на таблице 1 и таблице 2. На рисунке 13 можно увидеть гистограмму, где сравнивается коэффициент сжатия для небольшого файла.

Таблица – Пробные запуски для алгоритма Фано

|  |  |  |  |
| --- | --- | --- | --- |
|  | Текстовый файл малого алфавита | Бинарный файл | Файл с одним символом |
| Длительность кодирования (мкс) | 3954.5 | 33467.3 | 2614.5 |
| Длительность декодирования (мкс) | 2715.8 | 11420.5 | 2341.4 |
| Коэффициент сжатия | 3.846154 | 2.543509 | 1 |

Таблица – Пробные запуски для алгоритма LZ

|  |  |  |  |
| --- | --- | --- | --- |
|  | Текстовый файл малого алфавита | Бинарный файл | Файл с одним символом |
| Длительность кодирования (мкс) | 2679.1 | 29964.5 | 2596.1 |
| Длительность декодирования (мкс) | 2566.5 | 24651.6 | 2254.8 |
| Коэффициент сжатия | 1.012195 | 0.959821 | 0.500000 |

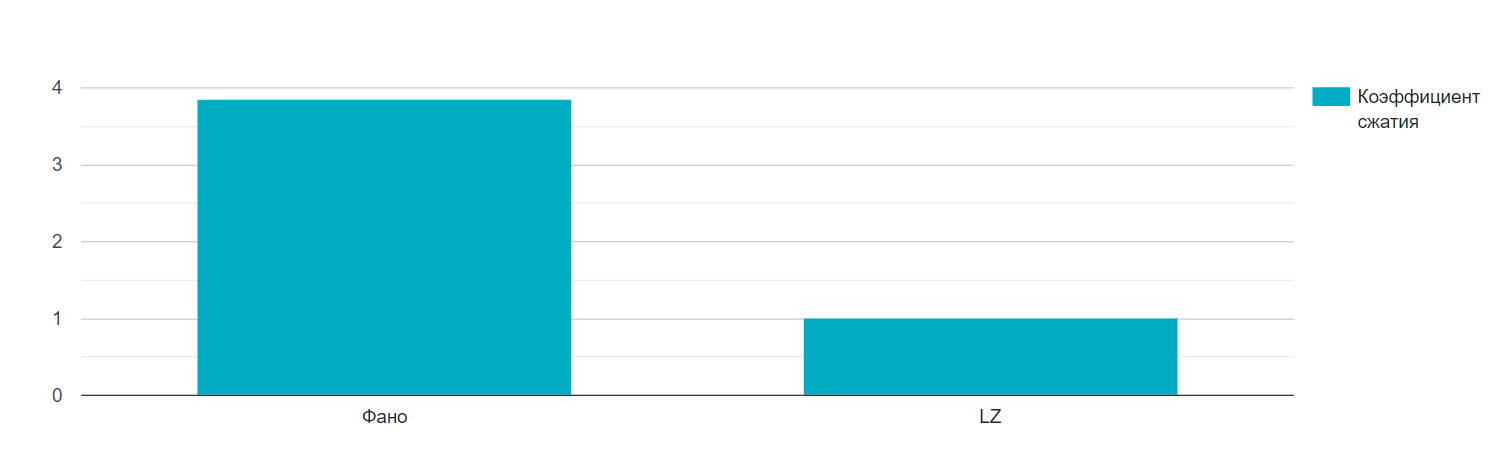


Рисунок – Коэффициент сжатия для алгоритмов Фано и LZ

1. Выводы

В процессе работы были приобретены знания в области исследования и реализации алгоритмов упаковки данных, навыки защиты информации с помощью методов кодирования. Были изучены алгоритмы Фано и LZ. Опытным путем было доказано, что алгоритм Фано быстрее и эффективнее, когда речь идет о небольшом файле. Алгоритм LZ больше подходит для больших текстов.

Список используемых источников

1. Зегжда П.Д. Основы информационной безопасности. Введение в профессиональную деятельность. Лабораторный практикум : учеб. пособие / П. Д. Зегжда, М. О. Калинин. – СПб. : ПОЛИТЕХ-ПРЕСС, 2019. – 112 с.
2. Алгоритмы LZ77 и LZ78 [Электронный ресурс]. – Режим доступа: http://neerc.ifmo.ru/wiki/index.php?title=Алгоритмы\_LZ77\_и\_LZ78, свободный.
3. Алгоритмы LZW, LZ77 и LZ78 [Электронный ресурс]. – Режим доступа: https://habr.com/en/post/132683/, свободный.

ПРИЛОЖЕНИЕ А

Листинг программы «Кодирование и упаковка данных. Главный файл»

#include <stdio.h>

#include <Windows.h>

#include "Fano\_2.0.h"

#include "LZ.h"

#define MAX\_FILE 30

void usage();

/\*Требования к файлу декодирования по Фано

- первая строка - это длина зашифрованного текста (бит)

- вторая - это размер алфавита

- третья - это количество символов сжатия, получившихся в ходе кодирования

- предпоследняя строка файла содержит символы сжатия

- последняя содержит биты, которые не влезли в стандартный размер char (эта строка может быть пустой,

если общее число получившихся бит кратно 8)

- остальные строки начинаются с символа "\*" и содержат информацию о кодировке исходных символов

в такой последовательности: символ (сразу после служебного символа "\*"), пробел, знак "-" (минус), пробел, кодировка символа.

Пример файла декодирования:

50

7

6

\*a - 00

\*Y - 01

\*n - 10

\*k - 110

\*o - 1110

\*c - 11110

\*h - 11111

HKЅь%„

10

\*/

void CheckArgv(int argc, char\*\* argv);

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

{

CheckArgv(argc, argv);

char\* c = 0;

double PCFreq = 0.0;

\_\_int64 CounterStart = 0;

LARGE\_INTEGER li;

if (!QueryPerformanceFrequency(&li))

{

printf("Error!\n");

return 0;

}

PCFreq = ((double)(li.QuadPart)) / 1000000.0;

QueryPerformanceCounter(&li);

CounterStart = li.QuadPart;

while (\*++argv)

{

c = \*argv;

switch (\*c)

{

case 'f':

FanoEncod(\*argv++, \*++argv);

break;

case 'F':

FanoDecod(\*argv++, \*++argv);

break;

case 'l':

LZ78Encod(\*argv++, \*++argv);

break;

case 'L':

LZ78Decod(\*argv++, \*++argv);

break;

default:

usage();

break;

}

}

QueryPerformanceCounter(&li);

printf("\ntime = %.3lf microseconds\n\n", ((double)(li.QuadPart - CounterStart)) / PCFreq);

return 0;

}

void CheckArgv(int argc, char\*\* argv)

{

if (argc == 4) {

if (!(strcmp(argv[1], "f") || strcmp(argv[1], "F") || strcmp(argv[1], "l") || strcmp(argv[1], "L")))

{

printf("Enter correct value of algorithm! f, F, l or L!\n");

exit(-1);

}

int i = 0;

for (; (argv[2][i] != '\0') && (i < MAX\_FILE); i++);

if (!((argv[2][i - 4] == '.' && argv[2][i - 3] == 'd' && argv[2][i - 2] == 'o' && argv[2][i - 1] == 'c') || (argv[2][i - 4] == '.' && argv[2][i - 3] == 't' && argv[2][i - 2] == 'x' && argv[2][i - 1] == 't')))

{

printf("Enter correct file! .txt or .doc!\n");

exit(-1);

}

for (; (argv[3][i] != '\0') && (i < MAX\_FILE); i++);

if (!(argv[3][i - 4] == '.' && argv[3][i - 3] == 't' && argv[3][i - 2] == 'x' && argv[3][i - 1] == 't'))

{

printf("Enter correct .txt file!\n");

exit(-1);

}

}

else {

printf("Wrong count of argv!\n");

usage();

exit(-1);

}

return 1;

}

void usage()

{

printf("\nEnter name of algorithm:\n\

f - to encode Fano alg.\n\

F - to decode Fano alg.\n\

l - to encode LZ alg.\n\

L - to decode LZ alg.\n\

Tnen enter two files: .doc or .txt");

}

ПРИЛОЖЕНИЕ B

Листинг программы «Кодирование и упаковка данных. Заголовочный файл алгоритма Фано»

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <math.h>

typedef unsigned long long ULL;

void FanoEncod(char\* file\_in, char\* file\_out);

int StructStrstr(char c, struct SymbolData\* alph, int alph\_size);

void Sorting(struct SymbolData\* alph, int n);

void MakeBinCode(struct SymbolData\* alph, int alph\_size, int step);

void PrintData(struct SymbolData\* alph, int n);

void PrintBinToFile(char\* file\_out, int\* bin\_text, int len\_bin\_text, int\* symbols\_for\_compression, int\* not\_fit);

void FanoDecod(char\* file\_out, char\* file\_res);

void DecToBin(unsigned char n, int\* s, int\* tmp, int step);

int WhatFile(char\* in);

int DocToText(char\* file\_in);

int ExeToText(char\* new\_name);

void FileError();

int TXTSize(new\_name);

void OneSymbolEncod(char\* file\_out, struct SymbolData\* alph);

void OneSymbolDecod(char\* file\_out, char\* file\_res);

ПРИЛОЖЕНИЕ C

Листинг программы «Кодирование и упаковка данных. Исполняемый файл алгоритма Фано»

#include "Fano\_2.0.h"

#define BYTE 8

#define NOTFOUND -1

#define MAX\_FILE 30

enum FILES

{

TEXT,

BIN,

EXE,

ERROR

};

enum COINCIDENCE

{

NO,

YES

};

typedef struct SymbolData

{

char symbol;

int repetitions;

char bin\_code[BYTE\*2];

} SymbolData;

void FanoDecod(char\* file\_out, char\* file\_res)

{

//printf("file\_in %s\tfile\_out %s\n", file\_out, file\_res);

int len\_bin\_text = 0;//прочитаем из файла

int alph\_size = 0;//прочитаем из файла

int count\_of\_sym\_compression = 0;//прочитаем из файла

FILE\* out = fopen(file\_out, "r");

if (!out)FileError();

fscanf(out, "%d\n", &len\_bin\_text);

fscanf(out, "%d\n", &alph\_size);

fscanf(out, "%d\n", &count\_of\_sym\_compression);

if (alph\_size == 1)

{

fclose(out);

OneSymbolDecod(file\_out, file\_res);

return;

}

SymbolData\* alph = (SymbolData\*)calloc(alph\_size,sizeof(SymbolData));

char c;

//читаем шаблон "символ - код"

for (int i = 0; i < alph\_size; i++)

{

fscanf(out, "\*%c - %s\n", &c, (&alph[i])->bin\_code);

(alph + i)->symbol = c;

//printf("%c -> %s\n", c, (&alph[i])->bin\_code);

}

//printf("%d\n", count\_of\_sym\_compression);

char\* symbols\_from\_file = (char\*)malloc(sizeof(char));

char\* tmp = NULL;

int symbols\_for\_compression = 0;

//чтение символов (без 0 и 1)

if (len\_bin\_text >= BYTE)

{

while (/\*(!feof(out))\*/ /\*&& (c = fgetc(out)) && (c != '\n') && (c != -1)\*/ (count\_of\_sym\_compression>0))

{

count\_of\_sym\_compression--;

c = fgetc(out);

symbols\_for\_compression++;

tmp = (char\*)realloc(symbols\_from\_file, symbols\_for\_compression \* sizeof(char) + 1);

symbols\_from\_file = tmp;

symbols\_from\_file[symbols\_for\_compression - 1] = c;

}

}

//printf("compression symbols were read\n");

while (c != '\n')

{

c = fgetc(out);

}

//чтение 0 и 1 (биты, которые не влезли в запись символа)

int not\_fit = 0;///////////////////////////////////////////////////////че делать exe не работает, не читает биты, выдает я аоаоаоао

while ((!feof(out)) /\*&& (c = fgetc(out)) && (c != -1)\*//\*&&((c=='0') || (c == '1'))\*/)

{

c = fgetc(out);

if ((c == '0') || (c == '1'))

{

++not\_fit;

tmp = (char\*)realloc(symbols\_from\_file, (symbols\_for\_compression + not\_fit) \* sizeof(char) + 1);

if (tmp == NULL) printf("no memory!");

symbols\_from\_file = tmp;

symbols\_from\_file[symbols\_for\_compression + not\_fit - 1] = c;

}

//else printf("error\n");

}

//printf("free bits were read\nnot\_fit == %d\n", not\_fit);

symbols\_from\_file[symbols\_for\_compression + not\_fit] = '\0';

//printf("%s\n", symbols\_from\_file);

fclose(out);

//int len\_bin\_text = symbols\_for\_compression \* BYTE + not\_fit; //длина двоичного кода, который подвергался сжатию

int index\_in\_symbol\_array = 0;

int\* bin\_text\_from\_file = (int\*)calloc(len\_bin\_text + 1, sizeof(int));

if (bin\_text\_from\_file == NULL) printf("no memory!");

int index\_in\_bin\_text = 0;

int tmp2;

unsigned char x;

for (; index\_in\_symbol\_array < symbols\_for\_compression;++index\_in\_symbol\_array)

{

index\_in\_bin\_text = BYTE \* (index\_in\_symbol\_array + 1) - 1;

x = symbols\_from\_file[index\_in\_symbol\_array];

DecToBin(x, bin\_text\_from\_file, &index\_in\_bin\_text, -1);

}

if(index\_in\_bin\_text!=0) index\_in\_bin\_text++;

//запись 0 и 1 в бинарный массив

//int not\_fit = len\_bin\_text - symbols\_for\_compression \* BYTE; //единицы и нолики, которые не влезли в символы

for (; index\_in\_bin\_text < len\_bin\_text;++index\_in\_bin\_text)

{

bin\_text\_from\_file[index\_in\_bin\_text] = symbols\_from\_file[index\_in\_symbol\_array] - '0';

++index\_in\_symbol\_array;

}

//for (int e = 0; e < len\_bin\_text;e++) //отладка

//{

// printf("%d ", bin\_text\_from\_file[e]);

//}

//начинаем дешифрование и запись результата в файл

FILE\* res = fopen(file\_res, "w");

if (!res)FileError();

int symbol\_index;

int count\_of\_bin;

enum COINCIDENCE flag\_coincidence;

int readed = 0;

while (readed < len\_bin\_text)

{

symbol\_index = 0;//чтобы ходить по алфавиту

count\_of\_bin = 0;//кол-во битов в коде символа

flag\_coincidence = YES;//совпадение/несовпадение

while (flag\_coincidence == YES)

{

//считаем, сколько битов занимает код символа

for (count\_of\_bin = 0;(symbol\_index < alph\_size) && (count\_of\_bin < BYTE) && (alph[symbol\_index].bin\_code[count\_of\_bin] != '\0'); count\_of\_bin++);

for (int q = 0; q < count\_of\_bin; q++)

{

if ((alph[symbol\_index].bin\_code[q] - '0') != bin\_text\_from\_file[readed + q])

//несовпадение

{

flag\_coincidence = NO;

break;

}

}

if (flag\_coincidence == YES) break;

else

{

symbol\_index++;

flag\_coincidence = YES;

}

}

if (flag\_coincidence == YES)

{

fprintf(res, "%c", alph[symbol\_index].symbol);

readed += count\_of\_bin;

}

}

free(symbols\_from\_file);

}

void FanoEncod(char\* file\_in, char\* file\_out)

{

//printf("file\_in %s\tfile\_out %s\n", file\_in, file\_out);

int text\_size\_for\_report = 0;

enum FILES f1 = ERROR, f2 = ERROR; //для хранения расширения файла

int count\_of\_name = 0; //для передачи другой переменной имени входного файла, чтобы при перезаписи исходника не портить переменную

while (file\_in[count\_of\_name] != '\0') //сколько символов в имени файла

{

++count\_of\_name;

}

char\* new\_name = (char\*)malloc(count\_of\_name \* sizeof(char));

if (new\_name == NULL) printf("no memory!");

strcpy(new\_name, file\_in);//копируем название файла

//далее имена двух файлов проверяются на расширение.

//если первый придет файл doc, нужно переписать его в текстовый. остальные вариации файлов приведут к ошибке

f1 = WhatFile(new\_name);

f2 = WhatFile(file\_out);

int textSize = 0;

if (f1 == TEXT && f2 == TEXT)

{

//printf("TXT\n"); ////для отладки

textSize = TXTSize(new\_name);

}

else if (f1 == BIN && f2 == TEXT)

{

//printf("DOC\n"); /// для отладки

textSize = DocToText(new\_name); //копирование текста из doc в txt

//printf("qwer\n");

}

else if (f1 == EXE && f2 == TEXT)

{

//printf("EXE\n"); /// для отладки

textSize = ExeToText(new\_name); //копирование текста из exe в txt

//rename(file\_in, "1.txt");

//if (rename(file\_in, "1.txt") == 0) // переименование файла

// printf( "File was renamed. Good\n");

//else

//{

// printf("Invalid file name!\n");

// exit(-1);

//}

//

//strcpy(new\_name, "1.txt");

//printf("okey\n");

}

else

{

printf("Invalid file name!\n");

exit(-1);

}

text\_size\_for\_report = textSize;

//открытие файла

FILE\* f\_in = fopen(new\_name, "r");

if (!f\_in)FileError();

//подготовка к чтению входного файла

int text\_size = 0;

int alph\_size = 0;

SymbolData\* alph = (SymbolData\*)malloc(sizeof(SymbolData));

SymbolData\* tmp = NULL;

char c = 0;

int k = 0;

//чтение тектового файла и создание алфавита

while ((!feof(f\_in)) /\*&& (c = fgetc(f\_in)) && (c != EOF)\*/&&(textSize!=0))

{

textSize--;

text\_size++;

c = fgetc(f\_in);

k = StructStrstr(c, alph, alph\_size);

if (k != NOTFOUND)//символ "с" уже есть в алфавите, добавляем единицу в счетчик

{

alph[k].repetitions++;

}

else//такого символа еще не было, записываем в алфавит

{

alph\_size++;

if ((tmp = (SymbolData\*)realloc(alph, alph\_size \* sizeof(SymbolData))))//выделяем место для еще одного символа

{

alph = tmp;

alph[alph\_size - 1].symbol = c;

alph[alph\_size - 1].repetitions = 1;

}

else

{

perror("Error! No memory!\n");

exit(-1);

}

}

}

//printf("text\_size = %d\n", text\_size);

//printf("alph\_size = %d\n", alph\_size);

if (alph\_size == 0)

{

printf("Error! No text!");

exit(-1);

}

if (alph\_size == 1)//отдельная обработка для одного символа!

{

OneSymbolEncod(file\_out, alph);

return;

}

//сортировка букв по убыванию повторений

Sorting(alph, alph\_size);

//кодирование алфавита

MakeBinCode(alph, alph\_size, 0);

//печать алфавита и кода каждого символа

//PrintData(alph, alph\_size);

//подготовка к кодированию текста

int\* bin\_text = (int\*)malloc(sizeof(int));

int len\_bin\_text = 0;

f\_in = fopen(new\_name, "r");

if (!f\_in)FileError();

int\* tmpB = NULL;

int len\_sym\_bc = 0;

//кодирование

while ((!feof(f\_in)) && (text\_size != 0) /\*&& (c = fgetc(f\_in)) && (c != -1)\*/)

{

text\_size--;

c = fgetc(f\_in);

len\_sym\_bc = 0;

for (int i = 0; i < alph\_size; i++)

{

if (c == alph[i].symbol)

{

for (int q = 0; alph[i].bin\_code[q] != '\0'; q++)

{

len\_sym\_bc++;

}

len\_bin\_text += len\_sym\_bc;

if (tmpB = (int\*)realloc(bin\_text, len\_bin\_text \* sizeof(int)))

{

bin\_text = tmpB;

for (int k = 0; k < len\_sym\_bc; k++)

{

bin\_text[len\_bin\_text - len\_sym\_bc + k] = alph[i].bin\_code[k] - '0';

}

break;

}

else

{

perror("Error! No memory!\n");

exit(-1);

}

}

}

}

fclose(f\_in);

//для отладки

/\*for (int i = 0; i < len\_bin\_text; i++)

{

printf("%d ", bin\_text[i]);

}\*/

//printf("\n%d\n", len\_bin\_text);

//запись алфавита для декодирования

FILE\* out = fopen(file\_out, "w");

if (out == NULL)FileError();

fprintf(out,"%d\n%d\n%d\n", len\_bin\_text, alph\_size, len\_bin\_text / BYTE);//длина зашифрованного текста и размер алфавита

for (int i = 0; i < alph\_size; i++)

{

fprintf(out, "\*%c - %s\n", alph[i].symbol, alph[i].bin\_code);

}

fclose(out);

int symbols\_for\_compression = 0;

int not\_fit = 0;

PrintBinToFile(file\_out, bin\_text, len\_bin\_text, &symbols\_for\_compression, &not\_fit);

free(alph);

free(bin\_text);

printf("\nCompression = %lf", (text\_size\_for\_report / (double)(symbols\_for\_compression + not\_fit)));

//printf("finish!\n");

}

void OneSymbolEncod(char\* file\_out, SymbolData\* alph)

{

alph[0].bin\_code[0] = '0';

alph[0].bin\_code[1] = '\0';

FILE\* out = fopen(file\_out, "w");

if (out == NULL)FileError();

fprintf(out, "%d\n%d\n", alph->repetitions, 1);//длина зашифрованного текста и размер алфавита

for (int i = 0; i < 1; i++)

{

fprintf(out, "\*%c - %s\n", alph[i].symbol, alph[i].bin\_code);

}

//сколько нулей можно сжать

int count\_symbols = alph->repetitions / BYTE;

for (int i = 0; i < count\_symbols; i++)

{

fprintf(out, "%d", 0);

}

fprintf(out,"\n");

//сколько нулей нельзя сжать

int not\_fit = alph->repetitions - count\_symbols \* BYTE;

for (int i = 0; i < not\_fit; i++)

{

fprintf(out, "%d", 0);

}

fclose(out);

}

void OneSymbolDecod(char\* file\_out, char\* file\_res)

{

int len\_bin\_text = 0;//прочитаем из файла

int alph\_size = 0;//прочитаем из файла

FILE\* out = fopen(file\_out, "r");

if (!out)FileError();

FILE\* res = fopen(file\_res, "w");

if (!out)FileError();

fscanf(out, "%d\n", &len\_bin\_text);

fscanf(out, "%d\n", &alph\_size);

SymbolData\* alph = (SymbolData\*)calloc(alph\_size, sizeof(SymbolData));

char c;

//читаем шаблон "символ - код"

fscanf(out, "\*%c - %s\n", &c, (&alph[0])->bin\_code);

alph->symbol = c;

//printf("%c -> %s\n", c, (&alph[0])->bin\_code);

fclose(out);

for (int i = 0; i < len\_bin\_text;i++)

{

fprintf(res, "%c", c);

}

fclose(res);

free(alph);

}

int StructStrstr(char c, SymbolData\* alph, int alph\_size)

{

if (alph\_size == 0) return NOTFOUND;

for (int i = 0; i < alph\_size; i++)

{

if (alph[i].symbol == c)

{

return i;

}

}

return NOTFOUND;

}

void Sorting(SymbolData\* alph, int n) //по убыванию

{

SymbolData tmp;

for (int i = 0; i < n - 1; i++)

{

for (int j = 0; j < n - i - 1; j++)

{

if (alph[j].repetitions < alph[j + 1].repetitions)

{

tmp = \*(alph + j);

\*(alph + j) = \*(alph + j + 1);

\*(alph + j + 1) = tmp;

}

}

}

}

void MakeBinCode(SymbolData\* alph, int alph\_size, int position)

{

//printf("\n-----------------\n");

if (alph\_size == 1)

{

alph[0].bin\_code[position] = '\0';

return;

}

int i = 0;

int beg = 0;

int end = 0;

for (; i < alph\_size; i++)

{

int left = 0;

int right = 0;

for (int j = 0; j <= i; j++)

{

left += alph[j].repetitions;

}

for (int j = i + 1; j < alph\_size; j++)

{

right += alph[j].repetitions;

}

beg = (left - right);

end = (left + alph[i + 1].repetitions - (right - alph[i + 1].repetitions));

if (abs(beg) <= abs(end)) break;

}

i++;

for (int j = 0; j < i; j++)

{

alph[j].bin\_code[position] = '0';

//printf("%c -> %s\n", alph[j].symbol, alph[j].bin\_code);

}

for (int j = i; j < alph\_size; j++)

{

alph[j].bin\_code[position] = '1';

//printf("%c -> %s\n", alph[j].symbol, alph[j].bin\_code);

}

MakeBinCode(&alph[0], i, ++position);

//printf("\n---------m--------\n");

MakeBinCode(&alph[i], alph\_size - i, position);

return;

}

void PrintData(SymbolData\* alph, int n)

{

for (int i = 0; i < n; i++)

{

printf("\nSymbol ='%c'\n", alph[i].symbol);

printf("Count of repetitions = %d\n", alph[i].repetitions);

printf("Binary code = %s\n\n", alph[i].bin\_code);

}

}

void PrintBinToFile(char\* file\_out, int\* bin\_text, int len\_bin\_text, int\* symbols\_for\_compression, int\* not\_fit)

{

FILE\* out = fopen(file\_out, "a");

if (out == NULL)FileError();

\*symbols\_for\_compression = len\_bin\_text / BYTE; //получаем целое кол-во символов, которое мы можем записать для сжатия

int index = 0; //чтобы двигаться по массиву bin\_text\_from\_file

char\* s\_f\_c = (char\*)calloc(\*symbols\_for\_compression + 1, sizeof(char));

if (s\_f\_c == NULL)

{

printf("no memory");

exit(-1);

}

//перевод из двоичного кода в символы сжатия

//printf("compression symbols:\n");

for (int i = 0; i < \*symbols\_for\_compression; i++)

{

for (int j = 0; j < BYTE; j++)

{

s\_f\_c[i] += (char)(bin\_text[index]) \* (char)pow(2, BYTE - 1 - j);

index++;

}

/\*if (s\_f\_c[i] == '\r')

{

fprintf(out, "\n");

fprintf(out, "\\r");

fprintf(out, "\n");

}

else if (s\_f\_c[i] == '\n')

{

fprintf(out, "\n");

fprintf(out, "\\n");

fprintf(out, "\n");

}

else if (s\_f\_c[i] == '\t')

{

fprintf(out, "\n");

fprintf(out, "\\t");

fprintf(out, "\n");

}

else

{

fprintf(out, "%c", s\_f\_c[i]);

}\*/

}

fwrite(s\_f\_c, sizeof(char), \*symbols\_for\_compression, out);

//printf("\ns\_f\_c - %s\n", s\_f\_c);

//fprintf(out, "%s\n", s\_f\_c);

//отладка

//printf("%s", s\_f\_c);

free(s\_f\_c);

fprintf(out, "\n");

//запись ноликов и единичек, которые не влезли в символы сжатия

\*not\_fit = len\_bin\_text - \*symbols\_for\_compression \* BYTE;

for (int j = 0; j < \*not\_fit; j++)

{

fprintf(out, "%d", bin\_text[index]);

//отладка

//printf("%d", bin\_text[index]);

//

index++;

}

fclose(out);

}

void DecToBin(unsigned char n, int\* s, int\* tmp, int step)

{

step++;

if (n >= 2)

{

DecToBin(n / 2, s, tmp, step);

}

int q = n % 2;

int w = \*tmp - step;

s[w] = q;

}

int WhatFile(char\* in)

{

int i = 0;

for (; (in[i] != '\0') && (i < MAX\_FILE); i++);

if (in[i - 3] == 'd' && in[i - 2] == 'o' && in[i - 1] == 'c')

return BIN;

else if (in[i - 3] == 't' && in[i - 2] == 'x'&& in[i - 1] == 't')

return TEXT;

else if (in[i - 3] == 'e' && in[i - 2] == 'x' && in[i - 1] == 'e')

return EXE;

else return ERROR;

}

int DocToText(char\* new\_name)

{

FILE\* fin = fopen(new\_name, "rb");

if (fin == NULL)FileError();

//считаем кол-во символов в тексте

fseek(fin, 0xa00, SEEK\_SET);

char c\_f\_in;

int count = 0;

while ((c\_f\_in = fgetc(fin)) && (c\_f\_in != NULL))

{

count++;

}

char\* text\_from\_bin\_file = (char\*)calloc(count, sizeof(char));

if (text\_from\_bin\_file == NULL) printf("no memory!");

//читаем эти символы и записываем в массив

fseek(fin, 0xa00, SEEK\_SET);

fread(text\_from\_bin\_file, sizeof(char), count, fin);

text\_from\_bin\_file[count-1] = '\0';

fclose(fin);

//printf("%s\n", text\_from\_bin\_file);

//создание имени для нового тектового файла

int count\_of\_name = 0;

while (new\_name[count\_of\_name] != '\0')

{

++count\_of\_name;

}

new\_name[count\_of\_name - 3] = 't';

new\_name[count\_of\_name - 2] = 'x';

new\_name[count\_of\_name - 1] = 't';

//printf("%s\n", new\_name);

fin = fopen(new\_name, "w");

if (fin == NULL) FileError();

fwrite(text\_from\_bin\_file, sizeof(char), count, fin);

fclose(fin);

free(text\_from\_bin\_file);

return count;

}

int ExeToText(char\* new\_name)

{

FILE\* fin = fopen(new\_name, "rb");

if (fin == NULL)FileError();

//считаем кол-во символов в тексте

fseek(fin, 0x00, SEEK\_END);

char c\_f\_in;

int count = ftell(fin);

// printf("%d\n", count);

rewind(fin);

//while (!feof(fin)) //не читается весь файл

//{

// c\_f\_in = fgetc(fin);

// count++;

//}

char\* text\_from\_bin\_file = (char\*)calloc(count, sizeof(char));

if (text\_from\_bin\_file == NULL) printf("no memory!");

//читаем эти символы и записываем в массив

fseek(fin, 0, SEEK\_SET);

int help\_count = fread(text\_from\_bin\_file, sizeof(char), count, fin);

// printf("%d\n", help\_count);

text\_from\_bin\_file[count - 1] = '\0';

fclose(fin);

// printf("%s\n", text\_from\_bin\_file);

//создание имени для нового тектового файла

int count\_of\_name = 0;

while (new\_name[count\_of\_name] != '\0')

{

++count\_of\_name;

}

new\_name[count\_of\_name - 3] = 't';

new\_name[count\_of\_name - 2] = 'x';

new\_name[count\_of\_name - 1] = 't';

// printf("new\_name = %s\n", new\_name);

fin = fopen(new\_name, "w");

if (fin == NULL)FileError();

fwrite(text\_from\_bin\_file, sizeof(char), count, fin);

fclose(fin);

free(text\_from\_bin\_file);

return count;

}

void FileError()

{

perror("Error: file not open");

exit(-1);

}

int TXTSize(char\* new\_name)

{

FILE\* fin = fopen(new\_name, "r");

if (fin == NULL)FileError();

//считаем кол-во символов в тексте

fseek(fin, 0x00, SEEK\_END);

char c\_f\_in;

int count = ftell(fin);

// printf("TextSize = %d\n", count);

fclose(fin);

return count;

}

ПРИЛОЖЕНИЕ D

Листинг программы «Кодирование и упаковка данных. Заголовочный файл алгоритма LZ»

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <math.h>

//#include <ctype.h>

void MemoryError();

void FileErrorLZ();

struct WORD\* Init\_word0(struct WORD\* dictionary);

void LZ78Encod(char\* file\_in, char\* file\_out);

void WriteToFile(char\* file\_out, struct WORD\* Dictionary, int size\_of\_dictionary);

int find(char\* tmp, struct WORD\* Dictionary, int size\_of\_dictionary);

void FreeWords(struct WORD\* Dictionary, int size\_of\_dictionary);

void LZ78Decod(char\* file\_out, char\* file\_res);

void SearchForWord(struct WORD\* Dictionary, int ind, int size\_of\_dictionary, char\* encode\_str, int i);

void WriteAnswer(struct WORD\* Dictionary, int size\_of\_dictionary, char\* file\_res);

ПРИЛОЖЕНИЕ E

Листинг программы «Кодирование и упаковка данных. Исполняемый файл алгоритма LZ»

#include "LZ.h"

#define START\_SIZE 2

#define PREFIX\_SIZE 10

#define MAX\_WORD\_SIZE 10

int count\_of\_encode\_symbols\_for\_report = 0;

//enum modes

//{

// START,

// INDEX,

// CHAR

//};

//enum Bool

//{

// FALSE,

// TRUE

//};

enum FILESlz

{

TXT,

DOC,

ERROR

};

typedef struct WORD

{

char\* word;

int index;

char symbol;

}WORD;

void MemoryError()

{

perror("Error: no memory!");

exit(-1);

}

void FileErrorLZ()

{

perror("Error: file not open!");

exit(-1);

}

void LZ78Encod(char\* file\_in, char\* file\_out)

{

WORD\* dictionary = NULL;//словарь, хранит пару идекс-символ для каждого "слова" и само слово

dictionary = (WORD\*)malloc(sizeof(WORD));

if (dictionary == NULL) MemoryError();

dictionary[0].word = (WORD\*)malloc(PREFIX\_SIZE\*sizeof(WORD));

if (dictionary[0].word == NULL) MemoryError();

dictionary = Init\_word0(dictionary);

FILE\* f\_in = fopen(file\_in, "rb");

if (!f\_in)FileErrorLZ();

int len = 0;

char\* txt = NULL;

enum FILESlz f = ERROR;

f = WhatFileLZ(file\_in);

if (f == DOC)

{

char c\_f\_in = 0;

fseek(f\_in, 0xa00, SEEK\_SET);

while ((c\_f\_in = fgetc(f\_in)) && (c\_f\_in != NULL))

{

len++;

}

txt = (char\*)malloc((len) \* sizeof(char));

fseek(f\_in, 0xa00, SEEK\_SET);

fread(txt, sizeof(char), len, f\_in);

txt[len - 1] = '\0';

//printf("%s\n", txt);

}

else if (f == TXT)

{

fseek(f\_in, 0x00, SEEK\_END);

len = ftell(f\_in);

//printf("%d\n", len);

rewind(f\_in);

txt = (char\*)malloc((len + 1) \* sizeof(char));//запись из файла в массив

if (txt == NULL) MemoryError();

if (fread(txt, sizeof(char), len, f\_in) == 0)

{

printf("Empty file!\n");

exit(0);

}

txt[len] = '\0';

//printf("%s\n", txt);

}

else

{

printf("File is not .txt or .doc!\n");

exit(-1);

}

fclose(f\_in);

char\* prefix = (char\*)calloc(PREFIX\_SIZE, sizeof(char));

if (prefix == NULL) MemoryError();

char\* character = (char\*)calloc(2, sizeof(char));

if (character == NULL) MemoryError();

char\* tmp = NULL;

int word\_index = -2;

int index\_for\_prefix = 1;

int size\_of\_dictionary = 0;

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

{

//printf("\nIteration %d\n", i);

character[0] = txt[i];

character[1] = '\0';

//printf("txt[%i] = %s\n", i, character);

tmp = (char\*)realloc(tmp, ((PREFIX\_SIZE+2) \* sizeof(char)) \* sizeof(char));

if (tmp == NULL) MemoryError();

strcpy(tmp, prefix);

strcat(tmp, character); //tmp = prefix + character

//printf("tmp = %s\n", tmp);

index\_for\_prefix = find(tmp, dictionary, size\_of\_dictionary);

if (index\_for\_prefix != -1)// найдено в словаре

{

strcat(prefix, character);

//printf("prefix = %s\n", prefix);

if (len - i == 1) //конец файла, когда остался один символ

{

size\_of\_dictionary++;

dictionary = (WORD\*)realloc(dictionary, (size\_of\_dictionary + 1) \* sizeof(WORD));

dictionary[size\_of\_dictionary].word = (WORD\*)calloc(PREFIX\_SIZE, sizeof(WORD));

strcpy(dictionary[size\_of\_dictionary].word, tmp);

dictionary[size\_of\_dictionary].index = index\_for\_prefix;

dictionary[size\_of\_dictionary].symbol = '\0';

}

}

else // не найдено, новая запсь в словаре

{

if (prefix[0] == '\0')

{

word\_index = 0; // префикс пустой, запись с индексом 0

}

else

{

word\_index = find(prefix, dictionary, size\_of\_dictionary); // поиск индекса для префикса

if (word\_index == -1)

{

word\_index = 0;

}

}

size\_of\_dictionary++;

dictionary = (WORD\*)realloc(dictionary, (size\_of\_dictionary+1) \* sizeof(WORD));

dictionary[size\_of\_dictionary].word = (WORD\*)calloc(PREFIX\_SIZE, sizeof(WORD));

strcpy(dictionary[size\_of\_dictionary].word, tmp);//размеры..

//printf("dictionary[%d].word = %s\n", size\_of\_dictionary, dictionary[size\_of\_dictionary].word);

dictionary[size\_of\_dictionary].index = word\_index;

//printf("txt[%i] = %c\n", i, txt[i]);

dictionary[size\_of\_dictionary].symbol = txt[i];

//printf("dictionary[%d].symbol = %c\n", size\_of\_dictionary, dictionary[size\_of\_dictionary].symbol);

//prefix[0] = '\0';

memset(prefix, 0x00, PREFIX\_SIZE);

memset(tmp, 0x00, PREFIX\_SIZE+2);

character[0] = '\0';

}

}

free(txt);

free(prefix);

free(character);

free(tmp);

WriteToFile(file\_out, dictionary, size\_of\_dictionary);

FreeWords(dictionary, size\_of\_dictionary);

free(dictionary);

printf("\nCompression = %lf", (len / (double)(count\_of\_encode\_symbols\_for\_report)));

}

void LZ78Decod(char\* file\_out, char\* file\_res)

{

FILE\* f\_out = fopen(file\_out, "rb");

if (!f\_out)FileErrorLZ();

fseek(f\_out, 0x00, SEEK\_END);

int len = ftell(f\_out);

//printf("%d\n", len);

rewind(f\_out);

char\* encode\_str = (char\*)malloc((len + 1) \* sizeof(char));//запись из файла в массив

if (encode\_str == NULL) MemoryError();

if (fread(encode\_str, sizeof(char), len, f\_out) == 0)

{

printf("Empty file!\n");

exit(0);

}

encode\_str[len] = '\0';

//printf("%s\n", encode\_str);

fclose(f\_out);

WORD\* dictionary = NULL;

dictionary = (WORD\*)malloc(sizeof(WORD));

if (dictionary == NULL) MemoryError();

dictionary[0].word = (WORD\*)malloc(PREFIX\_SIZE \* sizeof(WORD));

if (dictionary[0].word == NULL) MemoryError();

dictionary = Init\_word0(dictionary);

int size\_of\_dictionary = 0;

char sym;

//int num\_digit = 0; //индикатор разряда числа

//enum modes mode = START;

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

{

/\*if (mode == START)

{

mode = INDEX;

size\_of\_dictionary++;

dictionary = (WORD\*)realloc(dictionary, (size\_of\_dictionary + 1) \* sizeof(WORD));

dictionary[size\_of\_dictionary].word = (WORD\*)calloc(PREFIX\_SIZE, sizeof(WORD));

dictionary[size\_of\_dictionary].index = atoi(encode\_str[i]);

i++;

dictionary[size\_of\_dictionary].word[0] = encode\_str[i];

dictionary[size\_of\_dictionary].symbol = encode\_str[i];

}\*/

//else if (mode == INDEX/\* && (isdigit(encode\_str[i])\*/)

//{

size\_of\_dictionary++;

dictionary = (WORD\*)realloc(dictionary, (size\_of\_dictionary + 1) \* sizeof(WORD));

if (dictionary == NULL) MemoryError();

dictionary[size\_of\_dictionary].word = (WORD\*)calloc(PREFIX\_SIZE, sizeof(WORD));

if (dictionary[size\_of\_dictionary].word == NULL) MemoryError();

if (size\_of\_dictionary < 10)

{

//printf("%d\n", encode\_str[i] - '0');

dictionary[size\_of\_dictionary].index = encode\_str[i] - '0';

i++;

dictionary[size\_of\_dictionary].symbol = encode\_str[i];

SearchForWord(dictionary, dictionary[size\_of\_dictionary].index, size\_of\_dictionary, encode\_str, i); //сохранение слова в dictionary[size\_of\_dictionary].word

}

else if (size\_of\_dictionary < 100)

{

dictionary[size\_of\_dictionary].index = encode\_str[i] - '0';

i++;

if (encode\_str[i] < '0' || encode\_str[i] > '9') //следующий символ - буква. Значит, разряд == 1

{

dictionary[size\_of\_dictionary].symbol = encode\_str[i];

SearchForWord(dictionary, dictionary[size\_of\_dictionary].index, size\_of\_dictionary, encode\_str, i);

}

else //следующий символ - цифра. Значит, разряд == 2

{

dictionary[size\_of\_dictionary].index = 10 \* dictionary[size\_of\_dictionary].index + (encode\_str[i] - '0');

i++;

dictionary[size\_of\_dictionary].symbol = encode\_str[i];

SearchForWord(dictionary, dictionary[size\_of\_dictionary].index, size\_of\_dictionary, encode\_str, i);

}

}

else if (size\_of\_dictionary < 1000)

{

dictionary[size\_of\_dictionary].index = encode\_str[i] - '0';

i++;

if (encode\_str[i] < '0' || encode\_str[i] > '9') //следующий символ - буква. Значит, разряд == 1

{

dictionary[size\_of\_dictionary].symbol = encode\_str[i];

SearchForWord(dictionary, dictionary[size\_of\_dictionary].index, size\_of\_dictionary, encode\_str, i);

}

else if (encode\_str[i] >= '0' && encode\_str[i] <= '9' && (encode\_str[i+1] < '0' || encode\_str[i+1] > '9')) //следующий символ - цифра, а еще следующий - буква. Значит, разряд == 2

{

dictionary[size\_of\_dictionary].index = 10 \* dictionary[size\_of\_dictionary].index + (encode\_str[i] - '0');

i++;

dictionary[size\_of\_dictionary].symbol = encode\_str[i];

SearchForWord(dictionary, dictionary[size\_of\_dictionary].index, size\_of\_dictionary, encode\_str, i);

}

else //следующий символ - цифра, и еще следующий - цифра. Значит, разряд == 3

{

dictionary[size\_of\_dictionary].index = 10 \* dictionary[size\_of\_dictionary].index + (encode\_str[i] - '0');

i++;

dictionary[size\_of\_dictionary].index = 10 \* dictionary[size\_of\_dictionary].index + (encode\_str[i] - '0');

i++;

dictionary[size\_of\_dictionary].symbol = encode\_str[i];

SearchForWord(dictionary, dictionary[size\_of\_dictionary].index, size\_of\_dictionary, encode\_str, i);

}

}

//else if (size\_of\_dictionary < 100)

//{

// dictionary[size\_of\_dictionary].index = atoi(encode\_str[i]);

// if (encode\_str[i + 1] >= '0' && encode\_str[i + 1] <= '9' && ((i+2>= len) || (encode\_str[i + 2] <= '0' && encode\_str[i + 2] >= '9'))) //значит двузначное

// {

// if (i + 2 >= len) //если мы в конце строки

// {

// i++;

// dictionary[size\_of\_dictionary].symbol = encode\_str[i];

// }

// i++;

// dictionary[size\_of\_dictionary].index = 10 \* dictionary[size\_of\_dictionary].index + atoi(encode\_str[i]);

// }

//

//}

//else if (size\_of\_dictionary <= 100)

//{

// dictionary[size\_of\_dictionary].index = atoi(encode\_str[i]);

// if (encode\_str[i + 1] >= '0' && encode\_str[i + 1] <= '9' && ((i + 2 >= len) || (encode\_str[i + 2] <= '0' && encode\_str[i + 2] >= '9'))) //значит двузначное

// {

// i++;

// dictionary[size\_of\_dictionary].index = 10 \* dictionary[size\_of\_dictionary].index + atoi(encode\_str[i]);

//

// }

// else if(encode\_str[i + 1] >= '0' && encode\_str[i + 1] <= '9' && ((i + 2 >= len) || (encode\_str[i + 2] >= '0' && encode\_str[i + 2] <= '9')) && ((i + 3 >= len) || (encode\_str[i + 3] <= '0' && encode\_str[i + 3] >= '9')))

//}

//else if (size\_of\_dictionary < 1000)

//{

//}

// }

// }

// else if (mode == CHAR)

// {

//}

}

WriteAnswer(dictionary, size\_of\_dictionary, file\_res);

free(encode\_str);

FreeWords(dictionary, size\_of\_dictionary);

free(dictionary);

}

void WriteAnswer(WORD\* Dictionary, int size\_of\_dictionary, char\* file\_res)

{

FILE\* f\_res = fopen(file\_res, "w");

if (!f\_res)FileErrorLZ();

for (int i = 1; i <= size\_of\_dictionary; i++)

{

fprintf(f\_res, "%s", Dictionary[i].word);

//printf("%s", Dictionary[i].word);

}

fclose(f\_res);

}

//dictionary[size\_of\_dictionary].index

void SearchForWord(WORD\* Dictionary, int ind, int size\_of\_dictionary, char\* encode\_str, int i)

{

if (ind == 0)

{

Dictionary[size\_of\_dictionary].word[0] = encode\_str[i];

Dictionary[size\_of\_dictionary].word[1] = '\0';

//printf("dictionary[%d].word (ind = 0) = %s\n", size\_of\_dictionary, dictionary[size\_of\_dictionary].word);

}

else

{

//printf("dictionary[%d].word (new) = %s\n", size\_of\_dictionary, dictionary[size\_of\_dictionary].word);

//printf("dictionary[%d].word (old) = %s\n", ind, dictionary[size\_of\_dictionary].word);

strcpy(Dictionary[size\_of\_dictionary].word, Dictionary[ind].word);

//printf("dictionary[%d].word (res) = %s\n", size\_of\_dictionary, dictionary[size\_of\_dictionary].word);

//printf(" dictionary[size\_of\_dictionary].symbol = %c", dictionary[size\_of\_dictionary].symbol);

char\* a = (char\*)malloc(2 \* sizeof(char));

a[0] = Dictionary[size\_of\_dictionary].symbol;

a[1] = '\0';

strcat(Dictionary[size\_of\_dictionary].word, a/\*dictionary[size\_of\_dictionary].symbol\*/);

//printf("dictionary[%d].word (res) = %s\n", size\_of\_dictionary, dictionary[size\_of\_dictionary].word);

free(a);

}

}

void FreeWords(WORD\* Dictionary, int size\_of\_dictionary)

{

for (int i = 1; i <= size\_of\_dictionary; i++)

{

free(Dictionary[i].word);

}

}

void WriteToFile(char\* file\_out, WORD\* Dictionary, int size\_of\_dictionary)

{

FILE\* f\_out = fopen(file\_out, "w");

if (!f\_out)FileErrorLZ();

for (int i = 1; i <= size\_of\_dictionary; i++)

{

fprintf(f\_out, "%d%c", Dictionary[i].index, Dictionary[i].symbol);

count\_of\_encode\_symbols\_for\_report += 2;

if (Dictionary[i].index >= 10)count\_of\_encode\_symbols\_for\_report++;

if (Dictionary[i].index >= 100)count\_of\_encode\_symbols\_for\_report++;

//printf("%d%c ", Dictionary[i].index, Dictionary[i].symbol);

}

fclose(f\_out);

}

//prefix+character

int find(char\* tmp, WORD\* Dictionary, int size\_of\_dictionary)

{

int i = size\_of\_dictionary;

while (i)

{

//printf("dictionary[%i].word = %s\n", i, dictionary[i].word);

if (strcmp(tmp, Dictionary[i].word)==0)

{

return i;

}

i--;

}

return -1;

}

WORD\* Init\_word0(WORD\* dictionary)

{

dictionary[0].index = 0;

dictionary[0].word[0] = '\0';

dictionary[0].symbol = '\0';

return dictionary;

}

int WhatFileLZ(char\* in)

{

int i = 0;

for (; (in[i] != '\0') && (i < 30); i++);

if (in[i - 3] == 'd' && in[i - 2] == 'o' && in[i - 1] == 'c')

return DOC;

else if (in[i - 3] == 't' && in[i - 2] == 'x' && in[i - 1] == 't')

return TXT;

else return ERROR;

}