**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»**

**ННК «ІПСА»**

Кафедра системного проектування

РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА

з дисципліни

"Теорія інформації і кодування"

на тему: "Програмна реалізація методу

словарного стиснення даних LZW"

Студента II курсу

групи ДА-51

Болобан Олег Анатолійович

Керівник доц., к.т.н. Капшук О.О.

Київ – 2017

Зміст

[1. Вступ 3](#_Toc405514840)

[2. Загальні теоретичні відомості 4](#_Toc405514841)

[3. Алгоритм реалізації методу 5](#_Toc405514842)

[4. Вибір засобів для реалізації програми 6](#_Toc405514843)

[5. Структура, інтерфейс та опис програми 7](#_Toc405514844)

[6. Текст програми 8](#_Toc405514845)

[7. Результати тестування програми 21](#_Toc405514846)

[8. Висновки 23](#_Toc405514847)

[9. Список використаних джерел 24](#_Toc405514848)

# Вступ

**Алгори́тм Ле́мпеля — Зіва — Ве́лча** ([**англ.**](http://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0)**Lempel — Ziv — Welch**, **LZW**) — це універсальний [алгоритм стиснення даних](http://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B8%D1%81%D0%BD%D0%B5%D0%BD%D0%BD%D1%8F_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) без втрат, створений Абрахамом Лемпелем ([англ.](http://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) Abraham Lempel), Якобом Зівом ([англ.](http://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) Jacob Ziv) і Террі Велчем ([англ.](http://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) Terry Welch). Він був опублікований Велчем в [1984 році](http://uk.wikipedia.org/w/index.php?title=1984_%D1%80%D0%BE%D1%86%D1%96&action=edit&redlink=1) в якості покращеної реалізації алгоритму [LZ78](http://uk.wikipedia.org/w/index.php?title=LZ78&action=edit&redlink=1), опублікованого Лемпелем і Зівом в [1978 році](http://uk.wikipedia.org/w/index.php?title=1978_%D1%80%D0%BE%D1%86%D1%96&action=edit&redlink=1). Алгоритм розроблений так, щоб його можна було швидко реалізувати, але він не обов'язково є оптимальним, оскільки він не проводить ніякого аналізу вхідних даних.

Акронім «LZW» вкадує на прізвища винахідників алгоритму: Лемпель, Зів и Велч, але багато хто стверджує, що, оскільки патент належав Зіву, то метод повинен називатися *алгоритмом Зіва — Лемпеля — Велча*.

На момент своєї появи алгоритм LZW давав кращий коефіцієнт стиснення для більшості застосувань, ніж будь-який інший добре відомий метод того часу. Він став першим широковживаним на комп'ютерах методом стиснення даних.

Алгоритм був реалізований в програмі compress, яка стала більш чи менш стандартною утилітою Unix-систем приблизно в 1986 році. Кілька інших популярних утиліт-архіваторів також використовують цей метод або близькі до нього.

В [1987 році](http://uk.wikipedia.org/w/index.php?title=1987_%D1%80%D0%BE%D1%86%D1%96&action=edit&redlink=1) алгоритм став частиною стандарту на формат зображень [GIF](http://uk.wikipedia.org/wiki/GIF). Він також може (опціонально) використовуватися в форматі [TIFF](http://uk.wikipedia.org/wiki/TIFF).

На даний момент алгоритм утримується в стандарті [PDF](http://uk.wikipedia.org/wiki/PDF).

# Загальні теоретичні відомості

Даний алгоритм при стисненні (кодуванні) динамічно створює таблицю перетворення рядків: певним послідовностям символів (словам) ставляться у відповідність групи бітів фіксованої довжини.

Спочатку таблиця ініціюється всіма 1-символьними рядками (у випадку 8-бітних символів – це 256 записів).

По мірі кодування алгоритм переглядає текст символ за символом і зберігає кожен новий унікальний 2-символьний рядок в таблицю у вигляді пари код/символ, де код посилається на відповідний перший символ. Після того, як новий 2-символьний рядок збережений в таблиці, на вихід передається код першого символу. Коли на вході читається черговий символ, для нього по таблиці знаходиться рядок, що вже зустрічався, максимальної довжини, після чого в таблиці зберігається код цього рядка з наступним символом на вході; на вихід видається код цього рядка, а наступний символ використовується у якості початку наступного рядка.

Дуже цікавою особливістю LZW є те, що алгоритму декодування на вході потрібен лише закодований текст, оскільки він може відтворювати відповідну таблицю перетворень безпосередньо по закодованому тексту.

# Алгоритм реалізації методу

1. Створення словника всіма можливими односимвольними фразами.
2. Ініціалізація вхідної фрази А першим символом повідомлення.
3. Пошук у словнику рядок А найбільшої довжини, яка співпадає з останніми прийнятими символами.
4. Зчитати черговий символ В з кодованого повідомлення.
5. Якщо КІНЕЦЬ\_ПОВІДОМЛЕННЯ, то видати код для А, інакше - крок 5.
6. Якщо фраза АВ вже є в словнику, присвоїти вхідній фразі А значення АВ і перейти до кроку 3, інакше - видати код А, додати АВ в словник, присвоїти вхідній фразі А значення В і перейти до кроку 3.
7. Кінець.

# Вибір засобів для реалізації програми

Програма була написана для терміналу за процедурною парадигмою програмування мовою с++ в середовищі DevC++.

# Структура, інтерфейс та опис програми

Після запуску програми користувачу пропонується ввести назву файлу, який потрібно архівувати. З цього файлу буде зчитано вхідне повідомлення.

Потім створюється базовий словник ( довжина слів якого 1 символ), після чого відбувається процес кодування, алгоритм якого наведений вище. В результаті кодування створюється вектор строкових змінних, котрі заповнені символами 0 і 1.

Наступним кроком створюється вихідний файл «out\_x.txt», в який записується закодоване повідомлення. На цьому робота архіватора завершується.

Після запуску розархіватора відбувається процес декодування (з символьних змінних до вихідного повідомлення). Для декодування програма зчитує дані з файлу «out\_x.txt».

По завершенню роботи програма виводить у вихідний файл «out\_y.txt» вихідне повідомлення.

# Текст програми

**Лістинг кодера:**

#include <fstream>

#include <iostream>

#include <vector>

#include <string>

#include <cstdlib>

#include <cstdio>

#include <sstream>

#include <ctime>

using namespace std;

string itos(int number)

{

stringstream ss;

ss << number;

string str = ss.str();

return str;

}

string check(string temp, vector<string> &myvector)

{

string buf;

for(int i=0; i < myvector.size(); i++)

if(myvector[i] == temp)

{

buf = itos(i);

return buf;

}

buf = "0";

return buf;

}

string mirror(string buf)

{

string res;

for(int i=buf.length()-1; i>=0; i--)

{

res += buf[i];

}

return res;

}

string codeToBinary(string code)

{

int temp, buf;

string res;

temp = atoi(code.c\_str());

while(buf != 0)

{

buf = temp / 2;

if(buf \* 2 == temp)

res += '0';

else

res += '1';

temp /= 2;

}

return mirror(res);

}

int MaxCdSz(vector<string> code)

{

int res = 0;

for(int i=0; i < code.size(); i++)

{

if(code[i].length() > res)

res = code[i].length();

}

return res;

}

string codeaddon(string code, char code\_length)

{

string res;

if(code.length() < code\_length)

{

int c = code\_length - code.length();

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

res += '0';

for(int i=0; i < code.length(); i++)

res += code[i];

return res;

}

return code;

}

int main()

{

string file;

cout<<"enter file name: ";

cin>>file;

ifstream iff(file.c\_str(), ios::binary);

if(!iff)

{

cout<<"file does not exist"<<endl;

exit(0);

}

string temp;

char buf;

char bar[1000]="";

iff.seekg(0, iff.end);

int filelength = iff.tellg();

iff.seekg(0, iff.beg);

if(!filelength)

{

cout<<"input file is empty"<<endl;

iff.close();

exit(0);

}

vector<string> myvector;

string newletter;

for(int i=0; i < 256; i++)

{

buf = i;

newletter = buf;

myvector.push\_back(newletter);

}

string P, C, PC;

vector<string> code;

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

{

iff.get(buf);

C = buf;

PC = P + C;

if(check(PC, myvector) == "0")

{

newletter = PC;

myvector.push\_back(newletter);

code.push\_back(check(P, myvector));

P = C;

}

else

{

P = PC;

}

}

code.push\_back(check(P, myvector));

iff.close();

ofstream off("offile.txt", ios::binary);

for(int k=0; k < code.size(); k++)

code[k] = codeToBinary(code[k]);

char cdsz = MaxCdSz(code);

off<<cdsz;

for(int k=0; k < code.size(); k++)

code[k] = codeaddon(code[k], cdsz);

string result;

for(int k=0; k < code.size(); k++)

result += code[k];

char buffer = 0;

int j=0;

for (int i=0; i < result.length(); i+=8)

{

buffer = 0;

for (j=0; j < 7; j++)

{

if (result[i+j] == '1')

{

buffer += 1;

buffer <<= 1;

}

else

buffer <<= 1;

}

if (result[i+j] == '1')

buffer += 1;

off<<buffer;

bar[i]='>';

cout<<bar[i];

}

off.close();

cout<<endl<<"End coding";

system("pause");

}

**Лістинг декодера:**

#include <fstream>

#include <iostream>

#include <vector>

#include <string>

#include <cstdlib>

#include <cmath>

#include <cstdio>

#include <sstream>

using namespace std;

string itos(int number)

{

stringstream ss;

ss << number;

string str = ss.str();

return str;

}

string check(string temp, vector<string> &myvector)

{

string buf;

for(int i=0; i < myvector.size(); i++)

if(myvector[i] == temp)

{

buf = itos(i+1);

return buf;

}

buf = "0";

return buf;

}

int main()

{

ifstream iff("offile.txt", ios::binary);

if(!iff)

{

cout<<"file does not exist"<<endl;

exit(0);

}

string temp;

char buf;

iff.seekg(0, iff.end);

int filelength = iff.tellg();

iff.seekg(0, iff.beg);

if(!filelength)

{

cout<<"input file is empty"<<endl;

iff.close();

exit(0);

}

char cdsz;

iff.get(cdsz);

vector<string> myvector;

string newletter;

for(int i=0; i < 256; i++)

{

buf = i;

newletter = buf;

myvector.push\_back(newletter);

}

vector<string> codes;

ofstream off("res", ios::binary);

int count8 = 0, j=0;

buf = 0;

temp.clear();

iff.get(buf);

while(!iff.eof())

{

while((j < cdsz) && !iff.eof())

{

if((buf>>(7 - (count8 % 8))) & 1)

temp += '1';

else

temp += '0';

count8++;

j++;

if(!(count8 % 8))

iff.get(buf);

}

j =0;

codes.push\_back(temp);

temp.clear();

}

if(codes[codes.size() - 1].length() < cdsz)

codes.pop\_back();

vector<int> code;

int bufint = 0, i, k;

for(int j=0; j<codes.size(); j++)

{

for(i = cdsz - 1, k=0; i>=0; i--, k++)

if(codes[j][i] == '1')

bufint += pow(2, k);

code.push\_back(bufint);

bufint = 0;

}

string P, C, PC;

for(int i=0; i<code.size(); i++)

{

C = myvector[code[i]];

off<<C;

if(i == 0)

P = C;

else

{

PC = P + C[0];

if(check(PC, myvector) == "0")

{

newletter = PC;

myvector.push\_back(newletter);

P = C;

}

}

}

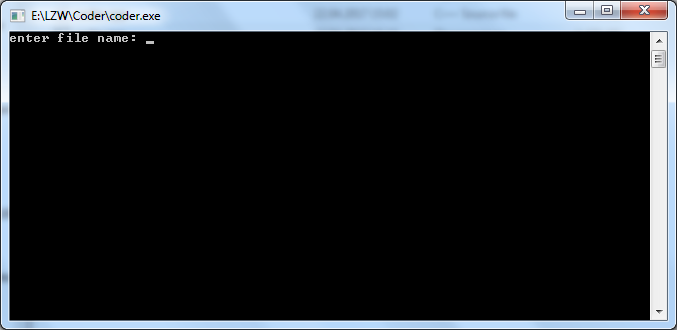
iff.close();

off.close();

cout<<"Decoding end!\nName: res";

}

# Результати тестування програми



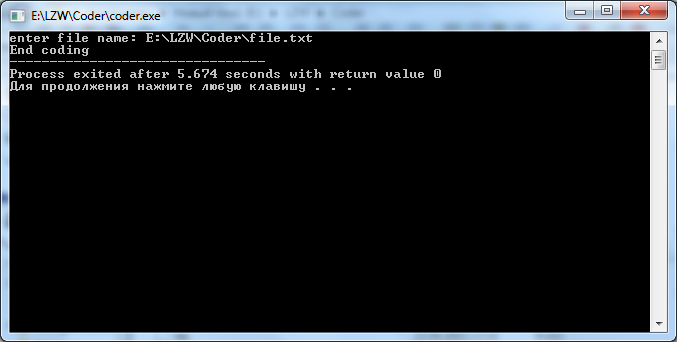


Рис. 7.1 – Програма працює без збоїв

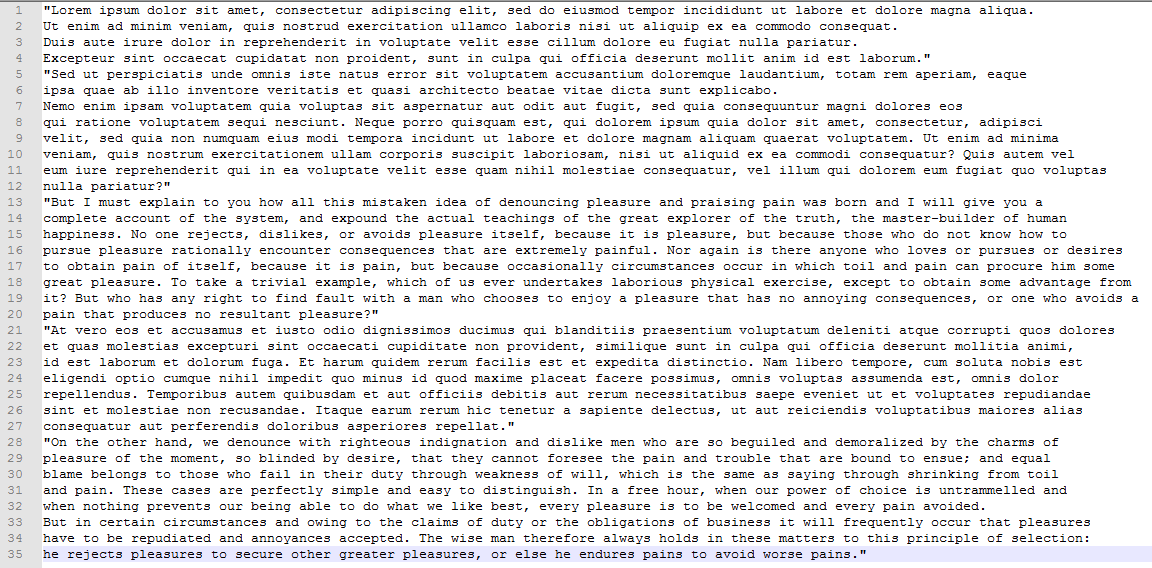


Рис. 7.2 – Вхідне повідомлення (файл «file.txt»)

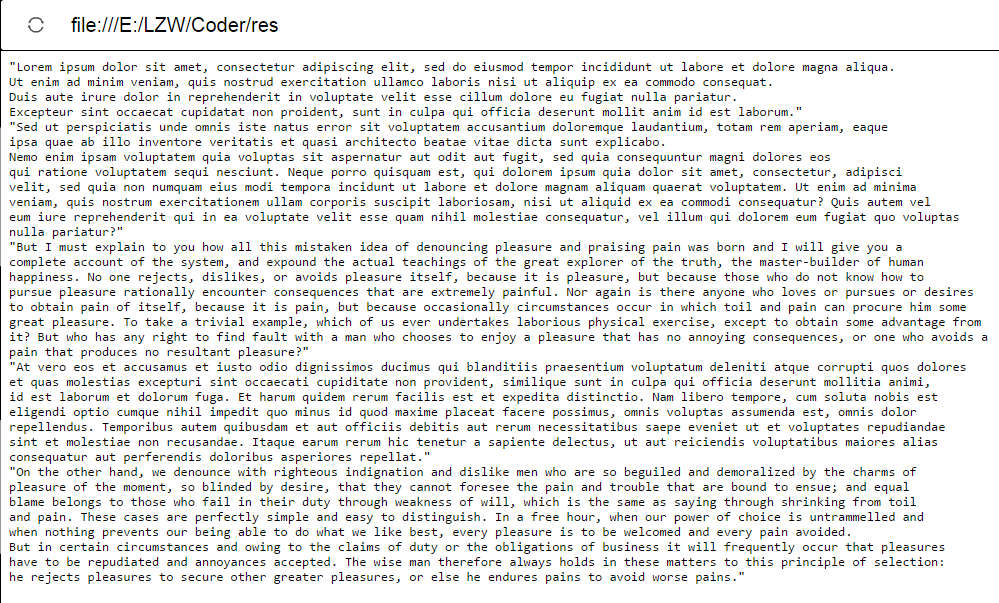


Рис. 7.3 – Відновлене повідомлення (файл «res»)

# Висновки

Переваги LZW:

* 1. алгоритм є однопрохідним;
  2. для декомпресії не треба зберігати таблицю рядків у файл для розпакування (алгоритм побудований таким чином, що ми в змозі відновити таблицю рядків, користуючись тільки потоком кодів).

Недолік LZW:

алгоритм не обов'язково є оптимальним, оскільки він не проводить жодного аналізу вхідних даних.

У випадках, коли в повідомленнях мало повторюваних комбінацій символів, алгоритм не дає значного виграшу в пам’яті, і навпаки.

# Список використаних джерел

1. <https://uk.wikipedia.org/w/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F-%D0%97%D1%96%D0%B2%D0%B0-%D0%92%D0%B5%D0%BB%D1%87%D0%B0>
2. <http://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_LZW>
3. <http://habrahabr.ru/post/132683/>