**Course Code: CPSC5330-01**

**Course Title: Advance Multimedia Processing**

**Project Report**

**Submitted To: Dr. Jiangjiang (Jane) Liu**

**Submitted By: Shubhashish Mishra**

**LU ID: L20365063**

**Date: 11/17/2014**

Contents

[1. LZW: An Introduction 1](#_Toc403991444)

[2. LZW Encoding and Decoding Algorithm 2](#_Toc403991445)

[2.1 LZW Encoding Algorithm 2](#_Toc403991446)

[2.2 LZW Decoding Algorithm 2](#_Toc403991447)

[3. Algorithm Implementation 3](#_Toc403991448)

[4. Testing Result & Comparison Analysis 3](#_Toc403991449)

[4.1 Program Testing: 3](#_Toc403991450)

[4.2 Compression Comparison and Analysis 6](#_Toc403991451)

[5. Appendix – A Readme.txt 7](#_Toc403991452)

[6. Appendix – B LZWEncoder.java 9](#_Toc403991453)

[7. Appendix – C LZWDecoder.java 15](#_Toc403991454)

[8. Appendix – D SCRIPT File: encoder.sh 19](#_Toc403991455)

[9. Appendix – E SCRIPT File: decoder.sh 20](#_Toc403991456)

[10. Appendix – F makefile 21](#_Toc403991457)

# LZW: An Introduction

LZW is a dictionary based data compression algorithm named after the initial of the proposer of the algorithm: Abraham Lempel, Jakob Ziv and Terry Welch. It is a lossless compression technique that scans the given file for sequence of characters that occur more than once. These sequence of character are then stored in the dictionary with an associated index. The index is repeated whenever the sequence is encountered and a new sequence of character with an unmatched character is entered into the dictionary with increasing index number.

LZW algorithm was successor of LZ78 data compression algorithm which in turn was successor of LZ77data compression algorithm. LZ77 was the first algorithm described by Lempel and Ziv in 1977 which is also a dictionary based data compression algorithm but maintains the dictionary within the data itself.

Similarly, LZ78 was the second algorithm proposed by Lempel and Ziv in 1978 which opposite to its predecessor maintains a separate dictionary.

Finally, LZW was third data compression algorithm published by Welch in 1984 as an improved implementation of the predecessor LZ78 algorithm.

Fundamentally speaking, LZW compression algorithm replaces string of characters with an index number. It does not perform any kind of analysis of the incoming text to get the statistical knowledge of data. So, it seems ideal technique to encode data stream in real-time as don’t take statistical knowledge into consideration to perform compression.

The length of the code outputted by LZW can be of any arbitrary length; however it must contain more bits in it than a single character. So, the first 256 codes (when using eight bit characters) i.e. 28, are by default assigned to the standard character set and the remaining codes are assigned to strings as the algorithm proceeds. However, the given program runs as shown with 12 bit codes i.e. 212 = 4096 codes. This means codes 0-255 are assigned represent single bytes from the input file, while codes 256-4095 are assigned to represent sequence of bytes.

LZW compression became the first widely used universal data compression method on computers. A large English text file can typically be compressed via LZW to about half its original size. LZW became very widely used when it became part of the GIF image format in 1987. It may also (optionally) be used in TIFF and PDF files.

# LZW Encoding and Decoding Algorithm

## LZW Encoding Algorithm

**1. Initialize table with single character strings**

**2. P = first input character**

**3. WHILE not end of input stream**

**C = next input character**

**IF P + C is in the string table**

**P = P + C**

**ELSE**

**output the code for P**

**add P + C to the string table**

**P = C**

**END WHILE**

**4. output code for P**

## LZW Decoding Algorithm

**1. Initialize table with single character strings**

**2. OLD = first input code**

**3. output translation of OLD**

**4. WHILE not end of input stream**

**NEW = next input code**

**IF NEW is not in the string table**

**S = translation of OLD**

**S = S + C**

**ELSE**

**S = translation of NEW**

**output S**

**C = first character of S**

**OLD + C to the string table**

**OLD = NEW**

**END WHILE**

# Algorithm Implementation

The program has two main classes and one inner static class to effectively accomplish the task of Compression and Decompression of different files type using LZW techniques. The class “LZWEncoder” is the one responsible for handling of file compression. It handles all the reading of filenames to be compressed or decompressed along with the availability of them. If the files are not present in the directory, it throws an exception. The other class is LZWDecoder, which is responsible for decoding the previously encoded file back to original form. The class LZWEncoder has a method compress() which is invoked to initiate the compression process while the class LZWDecoder has a method decompress() which is invoked to initiate the decompression process. The compressed file is automatically assigned “.lzw” as file type. All of the JAVA files are placed in directory called Project AMP along with directory like original, compressed and decompressed

# Testing Result & Comparison Analysis

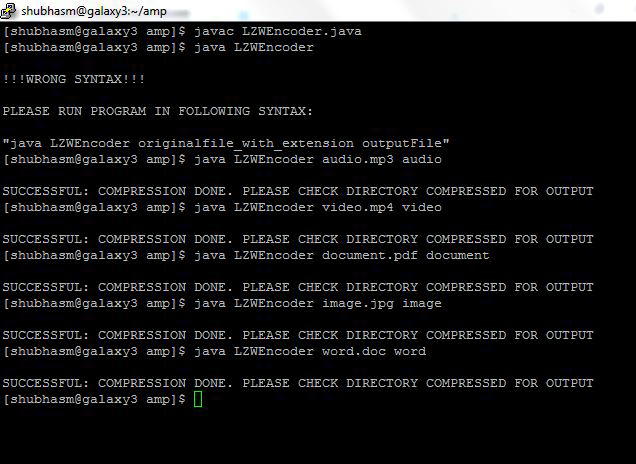
## 4.1 Program Testing:

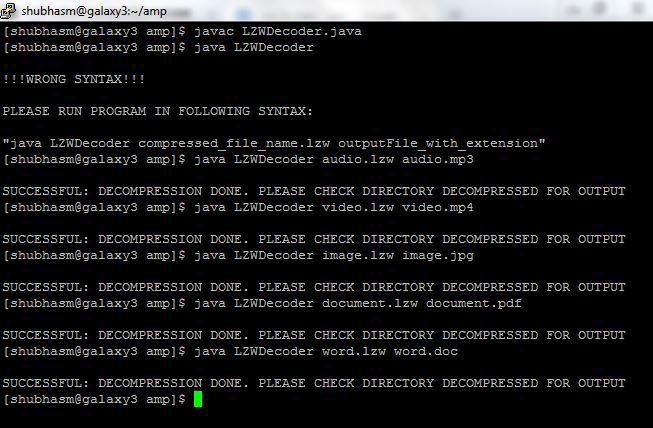
The program was tested for compression of following five different types of files.

1. audio.mp3
2. video.mp4
3. image.jpg
4. document.pdf
5. word.doc

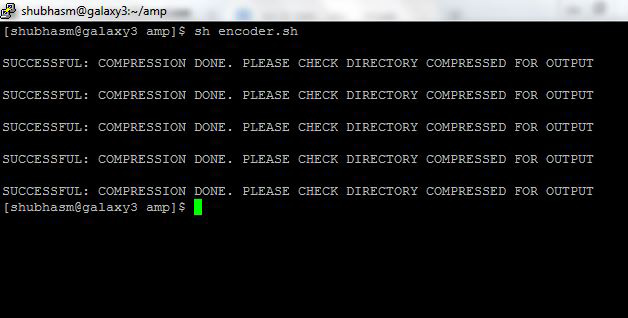
All of the above files were kept in the directory “original”. The files were then taken compressed and kept in the directory “compressed”. The file types for all compressed files were “.lzw” type. The decompressed files were found in directory “decompressed”. If the name and file type of output file while decompression is provided, the decompressed file can be opened using suitable program which were same to the original file. Of all file types compressed only .doc file showed some compression else all other file had a greater size.

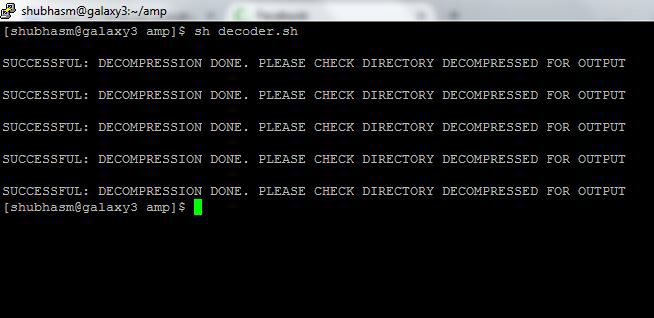
**Snapshot of Program Execution (Unix)**



****

**Execution through Shell (UNIX)**

****

****

## 4.2 Compression Comparison and Analysis

The original size of the five files, Compression size using LZW and using WINRAR are tabulated below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S.No. | File Name | Original size (KB) | Compressed Size- LZW (KB) | Compressed Size- WINRAR  (KB) | Compression  Ration  LZW | Compression  Ration  WINRAR |
| 1 | audio.mp3 | 4,452 | 6,233 | 4,378 | 1.40 | 0.98 |
| 2 | document.pdf | 21,998 | 29,028 | 13,415 | 1.32 | 0.61 |
| 3 | image.jpg | 5,989 | 8,224 | 5,484 | 1.37 | 0.92 |
| 4 | video.mp4 | 68,870 | 97,481 | 68,130 | 1.42 | 0.99 |
| 5 | word.doc | 2,318 | 1,255 | 352 | 0.54 | 0.15 |

The bar diagram representation of Compression Ratio of both LZW and WINRAR are as follow

For all file type WinRAR achieved less than unity Compression Ratio while LZW had only Compression Ratio less than Unity for Word file type and rest of them had more than unity compression ratio. WinRAR achieved a higher Compression Ratio because it compress file as per its type while the LZW technique applied same approach to all file compression. Besides, it can be inferred that LWZ compression technique is not suitable for compression all type of data file. It’s only effective for text files.5. Appendix – A Readme.txt

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Guidelines to compile and run programs\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. First of all copy the main folder "Project AMP" to a machine.

2. The folder contains the programs and other directories original, compressed, decompressed, and WinRAR.

The directory original contains audio.mp3, image.JPG, document.pdf, video.mp4, and word.doc.

The directories compressed and decompressed are to store output file of LZWEncoder and LZWDecoder class respectively.

The files inside directory compressed will have ".lzw" extionsion and those inside directory decompressed will have extension as given by user.

The directory WinRAR contains the input files compressed with WinRAR. The programs do not manipulate any of these files.

3. If you want to run this program in windows please make the following change.

a. Class LZW

Line 143: change "original/" to "original\\"

Line 145: change "compressed" to "compressed\\"

b. Class Decoder

Line 95: change "compressed/" to "compressed\\"

Line 97 change "decompressed/" to "decompressed\\"

4. To compile and run the class LZW manually the following commands should be prompted.

javac LZWEncoder.java

java LZWEncoder audio.mp3 audio

java LZWEncoder image.JPG image

java LZWEncoder pdf.pdf pdf

java LZWEncoder video.mp4 video

java LZWEncoder word.doc word

or to run automatically the following command should be typed

sh encoder.sh

5. To compile and run the class Decoder manually the following commands should be prompted.

javac LZWDecoder.java

java LZWDecoder audio.lzw audio

java LZWDecoder image.lzw image

java LZWDecoder pdf.lzw pdf

java LZWDecoder video.lzw video

java LZWDecoder word.lzw word

or to run automatically the following command should be typed

sh decoder.sh

Finally, see the directory compressed for encoded files and decompressed for decoded files.

# 6. Appendix – B LZWEncoder.java

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Public Class LZWEncoder that deals with the encoding process\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Author: Shubhashish Mishra

L20365063

Advance Multimedia Processing : Project--LZW Compression

\*/

import java.io.\*;

public class LZWEncoder {

private static final int EOF = -1;

private static final int HASHING\_SHIFT = 4;

private static final int BITS = 12; // for indicating 12 bit codes

private static final int TABLE\_SIZE = 5021; // maximum number of table codes

private static final int MAX\_VALUE = (1 << BITS ) - 1;

private static final int MAX\_CODE = MAX\_VALUE - 1;

private static BufferedInputStream inputFile = null;

private static BufferedOutputStream outputFile = null;

private int output\_bit\_count = 0;

private int output\_bit\_buffer = 0;

private short[] code\_value = new short[ TABLE\_SIZE ];

private short[] prefix\_code = new short[ TABLE\_SIZE ];

private short[] append\_character = new short[ TABLE\_SIZE ];

public void compress()

{

short next\_code = 0;

short character = 0;

short string\_code = 0;

short index = 0;

next\_code = 256; // Next code is the total number of single byte available

for ( short i = 0; i < TABLE\_SIZE; i++ ) // Removes entry from the code Table

code\_value[ i ] = -1;

try

{

string\_code = ( short ) inputFile.read(); // First character and is expected to be one of 256 ASCII character

while ( ( character = ( short ) inputFile.read() ) != EOF ) // The main execution loop where compression happens.

//The loop stop when all character are parsed and End Of File is found.

{

index = find\_match ( string\_code, character ); // Search for the match

if ( code\_value[ index ] != -1 )

{ // Gets the code value if match found /

string\_code = code\_value[ index ];

}

else // If match not found then add it

{

if ( next\_code <= MAX\_CODE )

{

code\_value [ index ] = next\_code++;

prefix\_code [ index ] = string\_code;

append\_character[ index ] = character;

}

output\_code( string\_code );

string\_code = character;

}

}

output\_code( string\_code ); // The last code is outputted

output\_code( ( short ) MAX\_VALUE ); // The end of buffer code is outputted

output\_code( ( short ) 0 ); // This code clears the output buffer

//Input and Output files are closed

outputFile.close();

inputFile.close();

}

catch ( IOException ioe )

{

System.out.println( "IOException in compress()" );

System.exit( 1 );

}

}

/\*

\*\* This is the hashing routine. It tries to find a match for the prefix+char

\*\* string in the string table. If it finds it, the index is returned. If

\*\* the string is not found, the first available index in the string table is

\*\* returned instead.

\*/

private short find\_match ( short hash\_prefix, short hash\_character )

{

int index = 0;

int offset = 0;

index = ( hash\_character << HASHING\_SHIFT ) ^ hash\_prefix;

if ( index == 0 )

offset = 1;

else

offset = TABLE\_SIZE - index;

while ( true )

{

if ( code\_value[ index ] == -1 )

return ( short ) index;

if ( prefix\_code[ index ] == hash\_prefix && append\_character[ index ] == hash\_character )

return ( short ) index;

index -= offset;

if ( index < 0 )

index += TABLE\_SIZE;

}

}

private void output\_code( short code )

{

output\_bit\_buffer |= code << ( 32 - BITS - output\_bit\_count );

output\_bit\_count += BITS;

while ( output\_bit\_count >= 8 )

{

try

{

outputFile.write( output\_bit\_buffer >> 24 );

}

catch( IOException ioe )

{

System.out.println( "IOException in output\_code()" );

System.exit( 1 );

}

output\_bit\_buffer <<= 8;

output\_bit\_count -= 8;

}

}

public static void main ( String[] args ) throws IOException

{

String in, out;

if(args.length >= 1){

in = "original/" + args[0];

inputFile = new BufferedInputStream(new FileInputStream(in));

out = "compressed/"+ args[1]+ ".lzw";

outputFile = new BufferedOutputStream(new FileOutputStream(out));

}

else{

System.out.print("\n!!!WRONG SYNTAX!!!\n\nPLEASE RUN PROGRAM IN FOLLOWING SYNTAX:\n\n\"java LZWEncoder originalfile\_with\_extension outputFile\"\n");

System.exit(1);

}

LZWEncoder sample = new LZWEncoder();

sample.compress();

try

{

inputFile.close();

outputFile.close();

}

catch ( IOException ioe )

{

System.out.println( "IOException in main()." );

System.exit(1);

}

System.out.println( "\nSUCCESSFUL: COMPRESSION DONE. PLEASE CHECK DIRECTORY COMPRESSED FOR OUTPUT");

}

}

# 7. Appendix – C LZWDecoder.java

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Public Class LZWDecoder that deals with the decoding process\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Author: Shubhashish Mishra

L20365063

Advance Multimedia Processing : Project--LZW Decompression

\*/

import java.io.\*;

import java.util.\*;

public class LZWDecoder{

static class Element

{

int prefix;

int suffix;

public Element(int first, int last)

{

prefix = first;

suffix = last;

}

}

final static int EXCESS = 4;

final static int MAX\_CODES = 4096; // total number of codes in table

final static int BYTE\_SIZE = 8;

final static int MASK = 15;

final static int ALPHA= 256; // total character

static int [] s;

static int size;

static Element [] h;

static int leftOver;

static boolean bitsLeftOver;

static BufferedInputStream in;

static BufferedOutputStream out;

private static void output(int code)throws IOException{ // write out the decoded sequence in the file opened

size = -1;

while(code>=ALPHA){

s[++size]=h[code].suffix;

code = h[code].prefix;

}

s[++size]=code;

for(int i=size; i>=0; i--)

out.write(s[i]);

}

private static int getCode() throws IOException{ // get the code from code table of the sequence

int c = in.read();

if(c == -1)return -1;

int code;

if(bitsLeftOver)

code = (leftOver<<BYTE\_SIZE)+c;

else{

int d = in.read();

code = (c<<EXCESS)+(d>>EXCESS);

leftOver = d&MASK;

}

bitsLeftOver = !bitsLeftOver;

return code;

}

private static void decompress() throws IOException{ // method that initiates the decompression process

int codeUsed = ALPHA;

s = new int[MAX\_CODES];

h = new Element[MAX\_CODES];

int pcode = getCode(), ccode;

if(pcode>=0){

s[0] = pcode;

out.write(s[0]);

size = 0;

do{

ccode = getCode();

if(ccode<0)break;

if(ccode<codeUsed){

output(ccode);

if(codeUsed<MAX\_CODES)

h[codeUsed++] = new Element(pcode, s[size]);

}

else{

h[codeUsed++] = new Element(pcode, s[size]);

output(ccode);

}

pcode = ccode;

}while(true);

}

out.close();

in.close();

}

public static void main(String [] args) throws IOException{ // main method

String inputFile, outputFile;

if(args.length >= 1){

inputFile = "compressed/" + args[0];

in = new BufferedInputStream(new FileInputStream(inputFile));

outputFile = "decompressed/"+ args[1];

out = new BufferedOutputStream(new FileOutputStream(outputFile));

}

else{

System.out.print("\n!!!WRONG SYNTAX!!!\n\nPLEASE RUN PROGRAM IN FOLLOWING SYNTAX:\n\n\"java LZWDecoder compressed\_file\_name.lzw outputFile\_with\_extension\"\n");

System.exit(1);

}

decompress();

System.out.println("\nSUCCESSFUL: DECOMPRESSION DONE. PLEASE CHECK DIRECTORY DECOMPRESSED FOR OUTPUT");

}

}

# 8. Appendix – D SCRIPT File: encoder.sh

javac LZWEncoder.java

chmod 777 encoder.sh

java LZWEncoder audio.mp3 audio

java LZWEncoder image.jpg image

java LZWEncoder document.pdf document

java LZWEncoder video.mp4 video

java LZWEncoder word.doc word

# 9. Appendix – E SCRIPT File: decoder.sh

javac LZWDecoder.java

chmod 777 decoder.sh

java LZWDecoder audio.lzw audio.mp3

java LZWDecoder image.lzw image.jpg

java LZWDecoder document.lzw document.pdf

java LZWDecoder video.lzw video.mp4

java LZWDecoder word.lzw word.doc

# 10. Appendix – F makefile

all:

javac LZWEncoder.java

javac LZWDecoder.java