



**Cairo University** **faculty of engineering**

# Multimedia Project

## Semester spring 2018

# Introduction:

This is project for Cairo university faculty of engineering computer department for semester 2017/2018 second year, multimedia.

The main purpose of the project is write program that use the best lossless compression technique to compress Arabic text (set of Arabic book reviews).

# History:

First we tried **simple implantation algorithm** that convert each char from 16 bit to 7 bit it gives us good result (more than 2 compression ratio & close range) ,then we tried **Huffman algorithm** ,better result than the first (2-3.2 compression ratio) but has very wide range ,we implement **Extend Huffman algorithm** but it take very long time for compress and decompress so we skip it, we tried to implement and test **arithmetic algorithm** but bad result in the most text, we read about **Deflate**, **LZ77** & **LZW** decide to use LZW for more efficient, tried to apply Huffman but no good result ,we discovered very interesting way to compress binary sequence(we will talk about it in the next content) that give us very good compression ratio(3.80-3.97) ,then we tried to compress the binary sequence by **Run-Length encoded** but we had not any good result(file size was bigger than the original file) ,we also read and search about **LZMA algorithm** and **integer arithmetic algorithm** but very complex and hard for us ,can't implement it in the time.

# The final compression algorithm:

**Main algorithm:** we decide to use LZW for the project but we improved it for better result, we store dictionary include all letter that appear in all files (113 character).

**Improved algorithm:** for LZW we have very special array-see fig1-, we thought it is waste for store each number in 32bit (about 10 power 9 number) when most time the number didn't exceed 24bit, so decide to limit dictionary so each number has less than 32 bit, we note that different limit size number give us different result for each file, (tried 14,16,18,20,22,24,26 bit) for some file best result was for 16bit and 20 bit but there are problem that some file give best result with 16bit and the another give best result with 20bit , but after that we discover no need to store each number with the limit size of number and implement an algorithm that store number by different number of bit depend on the position of the number , as example for the first number we know it will be less than 114 less than 2 power 7bit ,then store it in only 7 bit and so on until size of dictionary will exceed 128

now the maximum number can store it in 8 bit and so on , we apply same algorithm for decode.

When most of number are very closed to their neighbors, there are another way by , divide the array to **X** parts, each parts has **N** numbers then take the smallest number store it in the first then store only the first **K** bit (ex: if the array are: 1000000009, 1000000011, 1000000001, 1000000005, 1000000007 > then we can store it like 1000000001:8, 10, 0, 4, 6) or find the longest common number in them(10000000) and store it –will take more time- but there are no time for testing it.

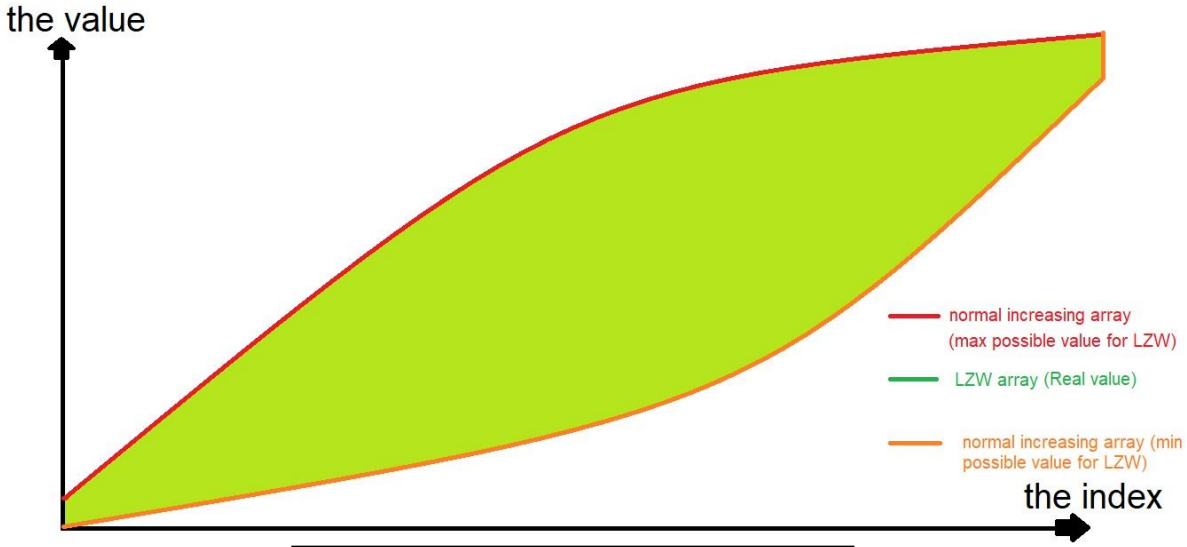


Fig1: approximate function for LZW Array

For OOP we implement each algorithm in independent class.

# Content of the compressed files:

Only need dictionary with all letter for all data set.

# Result:

For the pervious compression technique:

- Simple implantation algorithm: ratio about 2.2
- Huffman: ratio about 3
- Extend Huffman : unknown

- Arithmetic :0.5-1
- Run length encoding : about 0.5
- LZW with limit : 3-3.5
- Improved LZW: about 3.89

| Name | original size | compression size | compression ratio |
|---|---|---|---|
| DataSet_01 | 1949KB | 496KB | 3.929435483870968 |
| DataSet_02 | 1720KB | 440KB | 3.909090909090909 |
| DataSet_03 | 1357KB | 346KB | 3.921965317919075 |
| DataSet_04 | 1807KB | 465KB | 3.886021505376344 |
| DataSet_05 | 2122KB | 540KB | 3.929629629629630 |
| DataSet_06 | 1747KB | 442KB | 3.952488687782805 |
| DataSet_07 | 1885KB | 488KB | 3.862704918032787 |
| DataSet_08 | 1756KB | 453KB | 3.876379690949227 |
| DataSet_09 | 2033KB | 511KB | 3.978473581213307 |
| DataSet_10 | 2202KB | 559KB | 3.939177101967800 |
| DataSet_11 | 1923KB | 493KB | 3.918864097363083 |
| DataSet_12 | 1800KB | 473KB | 3.805496828752643 |
| DataSet_13 | 1935KB | 499KB | 3.877755511022044 |
| DataSet_14 | 1835KB | 480KB | 3.822916666666667 |
| DataSet_15 | 1851KB | 478KB | 3.872384937238494 |
| DataSet_16 | 1723KB | 449KB | 3.837416481069042 |
| DataSet_17 | 1834KB | 477KB | 3.844863731656184 |
| DataSet_18 | 1985KB | 515KB | 3.801941747572816 |
| DataSet_19 | 1835KB | 476KB | 3.855042016806723 |
| DataSet_20 | 1885KB | 490KB | 3.846938775510204 |
| total | 36.3MB | 9.33MB | 3.890675241157556 |

# Task board:

| Name | Task |
|---|---|
| Omar Mohammed | Read & write on the file |
| | Interface |
| | search |
| | Testing |
| Yahia Ali | Algorithm |
| | integration |
| | Document |
| | Search |
| | Testing |

Note: integration include rename variable & files, add important comment, edit for more optimize & validation, and remove all unimportant code.

# Glossary

| | |
|---|---|
| LZMA | Lempel–Ziv–Markov chain algorithm |
| LZW | Lempel–Ziv–Welch algorithm |
| LZ77 | Lempel–Ziv algorithm at 1977 |