Faculty of Engineering
Cairo University
Computer Engineering Department

Multimedia Project Report - Round 2

CMP206 - Multimedia - SPRG2020

Student Information:

- Name: Saad Eldeen Mohamed Mohamed
- ID: 9180609
- Email: SaadBadr23@gmail.com
- Section: 1
- Bench Number: 28

Submitted to

Dr. Ahmed Hamdy

Contents

Case Analysis

- File Size: 95.3 MB
- File Name: enwik8
- Content Type: text data
    - UTF-8 clean. All characters are in the range U'0000 to U'10FFFF with valid encodings of 1 to 4 bytes.
    - Contains some URL encoded XHTML tags.

Best Existing Algorithms

Due to the high probability of string repetition, theoretically the best results in this case will be achieved using lossless dictionary compression algorithms such as LZ77, LZ78, LZW, etc.…

And due to characters repetition, Good results will be achieved too using entropy encoding such as: Huffman coding, Arithmetic coding, etc.…

Also, there are:

• LZMA (Lempel–Ziv–Markov chain algorithm) which is used in 7z and it uses a dictionary compression scheme similar to the LZ77 algorithm.

• BWT (Burrows–Wheeler transform) which is used in bzip2 and it rearranges a character string into runs of similar characters.

My Program

As I stated, dictionary algorithms and entropy encoding work fine for this case, so I implemented Huffman encoding algorithm in my program.

I have tried to implement LZ77 too and it works fine but it takes high encoding time "200 seconds" with low compression ratio "1.08" which makes it useless in either using it alone or using it before Huffman as Huffman takes only 7 seconds as encoding time with compression ratio equals 1.57.

So, **I decided to keep Huffman only**.

LZ77 encoding time: 200 s LZ77 decoding time: 2 s LZ77 compression ratio: 1.08 Sliding window size: 4096 bytes, Look ahead buffer size: 16 bytes	Huffman encoding time: 7 s Huffman decoding time: 6 s Huffman compression ratio: 1.57

This is a small comparison between LZ77 and Huffman both implemented by me from scratch and running on my pc - which I described its specifications below in Experiments Description – to encode file 'enwik8'.

Note: I have attached all my source code as required, and as I didn't use my LZ77 implementation in the final program and only used Huffman, I added a folder called "NOT_USED_LZ77" which contains a zip file "LZ77.zip" contains my implementation for LZ77 encoder and decoder.

Experiments Description

Hardware Environment

Experiments done on device of these specifications:

- Processor: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
- Installed RAM: 16.0 GB (15.9 GB usable)
- System type: 64-bit operation system, x64-based processor
- Benchmarks:
    - CPU Score: 9.1
    - D3D Score: 9.9
    - Disk Score: 8.05
    - Graphics Score: 6.3
    - Memory Score: 9.1

Software Environment

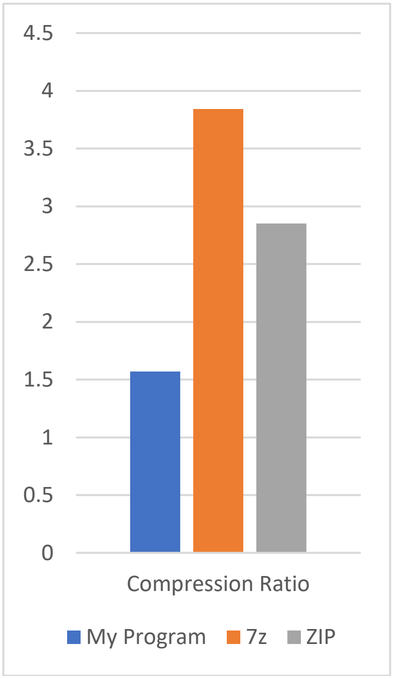
- OS: Windows 10 Home
    - Version: 1909
    - Build: 18363.900
- My Program:
    - Written in **C++**
    - Build using:
        - **g++** (x86_64-posix-seh-rev0, Built by MinGW-W64 project) **8.1.0**
        - command: **g++ -O3 Source.cpp**
- 7z and ZIP:
    - Used software: 7-Zip 18.05 (x64)
    - Software setting are set to default (normal mode)

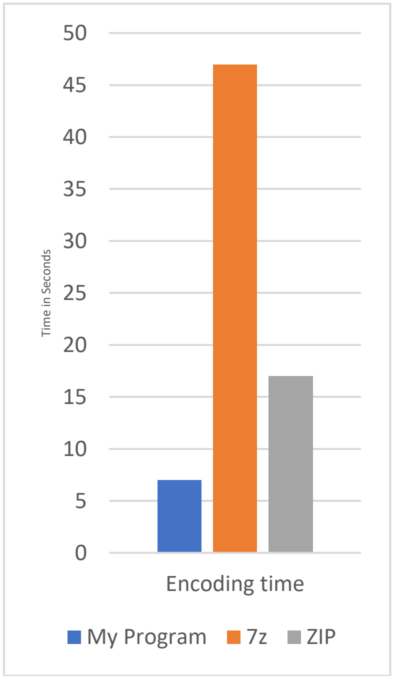
Results

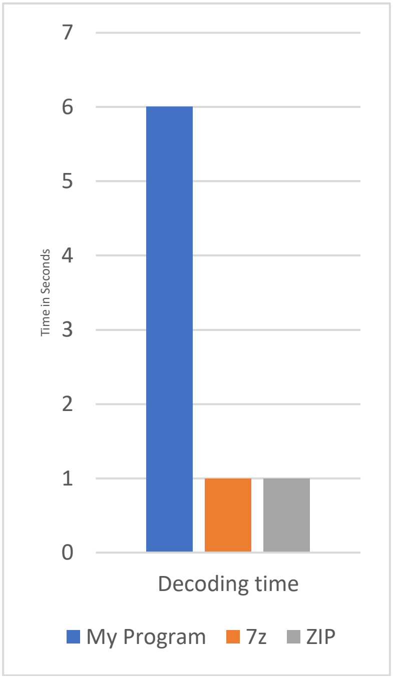
Comparison table

	My Program **(Huffman Coding)**	**7z**	**ZIP**
Original file size (MB)	95.3		
Compressed file size (MB)	60.9	24.8	33.5
Compression ratio	1.57	3.84	2.85
Encoding time (Second)	7	47	17
Decoding time (Second)	6	1	1

Charts


Compression Ratio
My Program
7z
ZIP


Time in Seconds
Encoding time
My Program
7z
ZIP


Time in Seconds
Decoding time
My Program
7z
ZIP