University of Information Technology

Multimedia project

Compression algorithms

Truong Phuc Anh 14520040
Lam Han Vuong 14521106
Trieu Trang Vinh 14521097

January 18, 2018

Contents

f

List of Figures

List of Tables

1 Introduction

Data compression is a set of steps for packing data into a smaller space, while allowing for the original data to be seen again. Compression is a two-way process: a compression algorithm can be used to make a data package smaller, but it can also be run the other way, to decompress the package into its original form. Data compression is useful in computing to save disk space, or to reduce the bandwidth used when sending data (e.g., over the Internet).

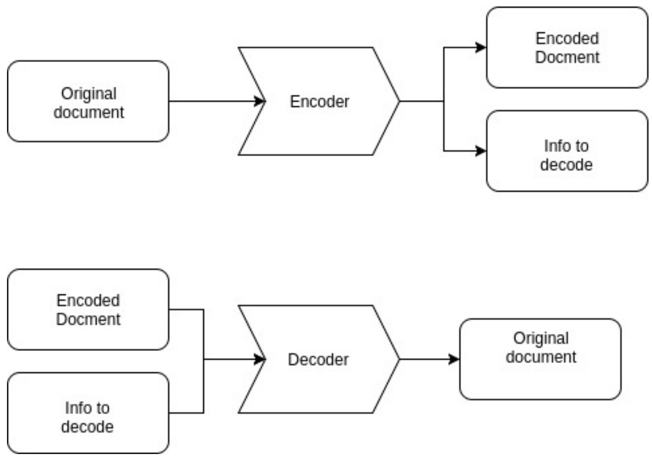

Figure 1: The flow chart show how data can be compressed and decompressed.

Compression can be either lossy or lossless. No information is lost in lossless compression. Lossy compression reduces bits by removing unnecessary or less important information.

Two main goals of this projects are:

1. Implements three lossless compression algorithms (run length, shannon-fano and jpeg) on real life data to see how it really works on different kinds of data.

2. Build a simple web application to work on the Internet environment for practice.

2 Dataset

Our dataset is collected from variety of sources such as Flicker, Google Images, news, html files from different sites, etc. We have 15 high quality images with smallest size is 1024x720 (formated as .jpg file) and 16 text file with different content includes some tutorial documents, news, source code (C++, Python), Wikipedia documents, letters, mails, binary files and some manual written files. For detail, you can check it out from *./data*

3 Methods

3.1 Run-length coding (RLC)

Input

The document need to compress.

Output

The compressed document.

Basic idea

Run-length encoding simple replaces sequences of the same data values within a file by a count number and a single value.

Example

Suppose the following string of data (17 bytes) has to be compressed: ABBBBBBBB-BCDEEEEF

Using RLE compression, the compressed file takes up 12 bytes and could look like this: 1A 8B 1C 1D 4E 1F

And for decompression, all we need is "write down" exactly "count number" times for each symbol. Finally, the result would be the same:
ABBBBBBBBBCDEEEEF

Pseudo-code

You can find pseudo-code (Python style) for Run-length encoding and decoding in *pseudo-code/run-length.py*

3.2 Shannon-Fano coding

Input

Set of symbol S.
The document need to compress.

Output

The compressed document.
The table of frequency (or number of times symbol appears in the document).

Basic idea

1. For a given list of symbols, calculate table of probabilities or frequency counts for the document

2. Sort the lists of symbols according to frequency (descending order)

3. Divide the list into two parts, with the total frequency counts of the left part being as close to the total of the right as possible.

4. The left part of the list is started with the code 0, and the right part is started with code 1.

5. Recursively apply the steps 3 and 4 to each of the two halves, subdividing groups and adding bits to the codes until each symbol has become a corresponding code leaf on the tree.

Example

The document need to compress: "ABBACAABCECAABADDDE".
Set of symbols S = A, B, C, D, E.
Encoded message: "0001010010000001101111000000100110110110111".
Decoded message: "ABBACAABCECAABADDDE".

Symbol	*Count*	*Probability*
A	7	0.37
B	4	0.21
C	3	0.16
D	3	0.16
E	2	0.11

Table 1: Calculate the table of frequency (descending oder)

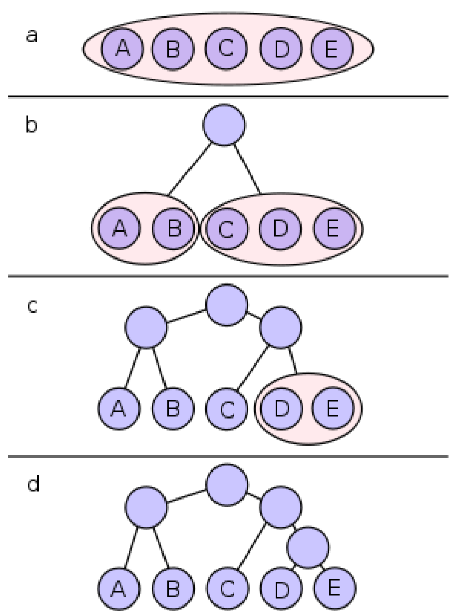

Figure 2: Divide the list of symbols and assign code

Symbol	*Code*
A	00
B	01
C	10
D	110
E	111

Table 2: The final code of symbols

Pseudo-code

You can find pseudo-code (Python style) for Shannon-Fano encoding and decoding in *pseudo-code/shannon-fano.py*

3.3 Lossless jpeg

Input

The input of jpeg usually is an image need to compress.

Output

The compressed "image"

Basic idea

For Compression:

1. Choose a predictor P.
2. Apply the predictor P on the input image I to get I_p (the first pixel [0, 0] mustn't change).
3. Calculate the different image $I_d = I - I_p$.
4. Run a lossless compression algorithm on I_d and get the compressed image.

For decompression:

1. Run the lossless decompression (same with the 4th step above) we get I_d
2. Apply the predictor P fro each pixel p_i from the first pixel [0, 0] of I_d
3. Then sum up the value from step 2 with the value of p_i in I_d and we will pixel value of the original image.

Examples

Suppose we have a gray-scale image (4x4) I like this:

$$\begin{matrix} 90 & 92 & 89 & 70 \\ 75 & 65 & 67 & 69 \\ 90 & 92 & 89 & 70 \\ 90 & 92 & 89 & 70 \end{matrix}$$

Now, we apply the predictor $X = A + B - C$ to compute I_p:

$$\begin{matrix} 90 & 90 & 92 & 89 \\ 90 & 77 & 62 & 48 \\ 75 & 65 & 82 & 83 \\ 75 & 75 & 89 & 93 \end{matrix}$$

Then, we have the different image I_d:

$$\begin{matrix} 90 & 2 & -3 & -19 \\ -15 & -12 & 5 & 21 \\ 0 & 15 & -1 & 4 \\ -5 & 10 & 1 & -13 \end{matrix}$$

Finally, we can choose another lossless compressor to compress I_d.

Pseudo-code

You can find pseudo-code (Python style) for Lossless jpeg encoding and decoding in *pseudo-code/lossless-jpeg.py*

4 Experiments

Notice that the implementation for compression algorithms was written as readable as possible so the performance may not good.

Testing process For each algorithm we run the encoder, saved the encoded files and calculate the compression ratio as a table. Then, we run the decoder on encoded files to check whether it right or wrong (the same as the original file or not). Some special parameters values would be shown in tables below. RLC and Shannon-Fano Coding were perform on text data and JPEG-Lossless on images data.

As a simplest compression algorithms, RLC works extremely bad on real life document (news, article, mail, etc).Table 3 shows that RLC even expends data instead of reduce it(!). The only time RLC really works is the text file that we manually create just for seeing RLC works.

input file	bits before encode	bits after encode	ratio
../data/text/0.txt	12136	23824	0.51
../data/text/1.txt	20952	41152	0.51
../data/text/2.txt	15224	21856	0.70
../data/text/3.txt	24032	46896	0.51
../data/text/4.txt	28584	56240	0.51
../data/text/5.txt	224616	438512	0.51
../data/text/6.txt	46016	90016	0.51
../data/text/7.txt	218432	428384	0.51
../data/text/8.txt	27112	52184	0.52
../data/text/9.txt	106088	166952	0.64
../data/text/10.txt	37192	73040	0.51
../data/text/11.txt	38800	75104	0.52
../data/text/12.txt	70528	138320	0.51
../data/text/13.txt	58992	115216	0.51
../data/text/14.txt	128000	249184	0.51
../data/text/15.txt	808	240	3.37

Table 3: Run-length coding result on text data

input file	bits before encode	bits after encode	ratio
../data/text/0.txt	12136	8742	1.39
../data/text/1.txt	20952	14089	1.49
../data/text/2.txt	15224	9791	1.55
../data/text/3.txt	24032	16701	1.44
../data/text/4.txt	28584	21824	1.31
../data/text/5.txt	224616	158819	1.41
../data/text/6.txt	46016	31292	1.47
../data/text/7.txt	218432	163974	1.33
../data/text/8.txt	27112	21897	1.24
../data/text/9.txt	106088	81003	1.31
../data/text/10.txt	37192	27224	1.37
../data/text/11.txt	38800	27650	1.40
../data/text/12.txt	70528	52135	1.35
../data/text/13.txt	58992	44004	1.34
../data/text/14.txt	128000	101360	1.26
../data/text/15.txt	808	334	2.42

Table 4: Shannon Fano coding result on text data

The Shannon-fano coding is much more better RLC. At least, it really reduce the size of data. The table 5 shows that compression ratio is between 1.3 and

2.4, it seems Shannon-fano works but it's not good enough for what we expected.

input	bits before encode	bits after encode	entropy	ratio
0.jpg	6000000	5450900	7.10	1.10
1.jpg	6000000	6874242	7.32	0.87
2.jpg	6000000	7628772	7.74	0.79
3.jpg	6000000	8077175	7.18	0.74
4.jpg	6000000	5921214	7.94	1.01
5.jpg	6000000	12281604	7.70	0.49
6.jpg	6000000	8544825	6.79	0.70
7.jpg	6000000	9238257	7.64	0.65
8.jpg	6000000	2859180	7.12	2.10
9.jpg	6000000	11057427	7.34	0.54
10.jpg	6000000	10509966	7.70	0.57
11.jpg	6000000	11082918	7.57	0.54
12.jpg	6000000	4677102	7.79	1.28
13.jpg	6000000	4292124	7.51	1.40
14.jpg	6000000	4487562	7.65	1.34

Table 5: Jpeg-lossless coding result on Images data

The Jpeg-lossess coding is the simple compression algorithm on Image data. It uses a predictive scheme based on the three nearest (causal) neighbors (upper, left, and upper-left). It is very useful for image which uses few color.

Web Application In this application, we use Node js framework for building back-end system, every request from Client will be forwarded and executing python file. From the Client Side, we use HTML, CSS, Angular, JS, ... to design front page ... After received the request from Client, Node.js will connect to Python to process the files. After that, It will response the result to Client.

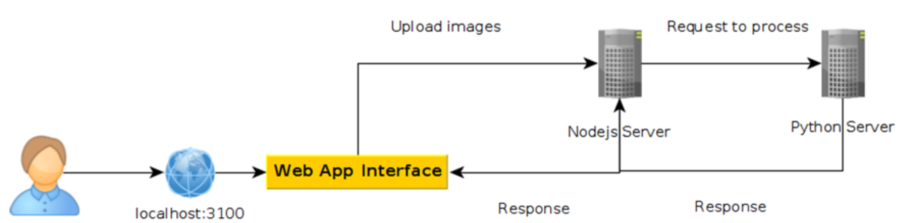

Figure 3: The main flow of our web application.

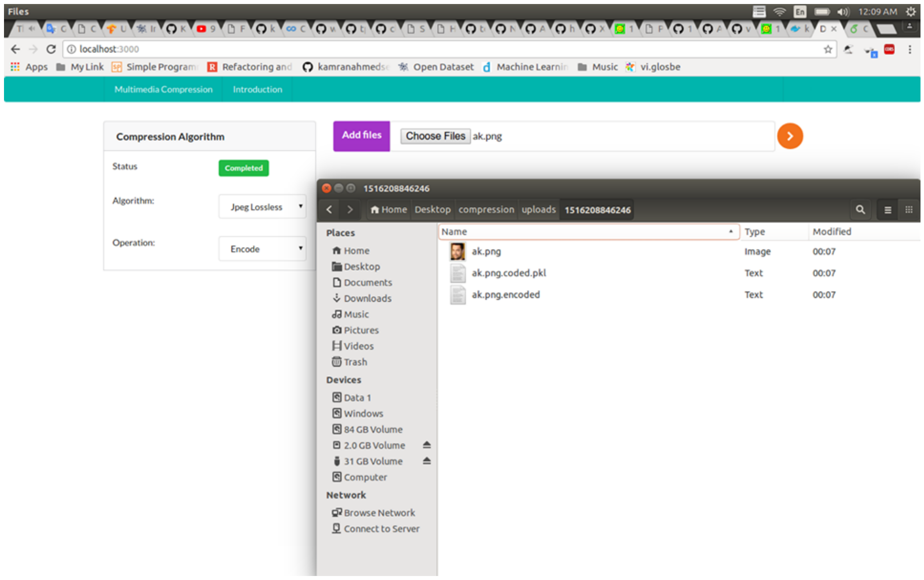

Figure 4: Simple GUI for web application.

5 Conclusions

Real and quick! No compression, no Youtube, no streaming, no source control, etc.

6 References

7 Appendices