Implementation of a Modified PCX Image Compression Using Java

CHE-CHERN LIN SHEN-CHIEN CHEN

National Kaohsiung Normal University
116 Ho-ping First Road,
Kaohsiung, Taiwan

cclin@nknucc.nknu.edu.tw; xol700@gmail.com

Abstract: - In this paper, we present a new image compression algorithm based on the PCX algorithm, an image compression method used in the computer package of PC Paintbrush Bitmap Graphic. We first introduce the principles of image compressions and the structure of image file formats. We demonstrate the procedures of compression and decompression of the PCX algorithm. The original PCX algorithm only compresses one fourth of data using run-length encoding. The compression efficiency depends on the repeatability of data in the compressed area. If the repeatability is low, the compression performance will be bad. To avoid this, we propose a modified PCX algorithm which selects the best area for compression. We designed a computer package to implement the modified PCX algorithm using java programming language. The Unified Modeling Language (UML) was used to describe the structure and behaviors of the computer package. The pseudo codes for the compression and decompression of the modified PCX algorithm are also provided in this paper. We did an experiment to compare the performance between the original and modified algorithms. The experimental results show that the modified PCX algorithm is better than the original one in compression performance.

Keywords: - PCX, Data compression, Image compression, Run-length encoding

1 Introduction

Data compression is a technique to reduce the data size by removing redundant or irrelevant information. Recently with the development of internet, a lot of data transmissions are through internet. Many internet-based applications rely on a large amount of data transmission such as Voice over IP (VoIP), video conferencing, file transfer, etc. Less mount of data transferred means less network bandwidth consumed. Therefore the data compression technologies are more and more important nowadays. Image compression is a kind of data compressions used to encode two or three dimensional data. An image compression algorithm depicts the procedure of encoding image data. In general, an image compression algorithm is associated with a corresponding decompression algorithm which is served to recover back the data from a compressed image.

There are two types of image compressions: lossy and lossless compressions [1]. A lossless compression is an error-free method which exactly recovers the compressed image data without losing any information. A lossy compression, however, misses some image data during compression and decompression. Several image compression methods are adopted for academic and commercial usage. The most straight forward one is to count the successive pixel value and then encode the repetitive number of the value and the repeated value itself. This is called run-length encoding [1]. A differential compression method is to encode the value difference between two adjacent pixels [2]. For a highly related image, the differential compression works efficiently. A variable-length compression is basically a statistical method to calculate the histogram of an image and encodes the data in different lengths based on the probabilities of symbols (pixel values) [1], [2]. Huffman compression is the famous one [1], [2]. It uses a bottom-up unbalance binary tree to arrange symbols form bottom to top according to the probabilities of symbols. Some other image compression methods transfer the original image domain (spatial domain) to other domains using appropriate transforms. For example, the Fourier transform calculates the frequency components (spectrum) based on a global scope of time interval. In some practical considerations, we might need some local spectrum information to get better compression results. Like short-time Fourier Transform, wavelet compression simultaneously calculates time (or spatial) and frequency components [3]. This is useful in image compression since a lot of images are locally related. FAX compression encodes images to Huffman-based codes according to a pre-defined code book. [1], [2]. It works efficiently in compressing binary document images. Joint Photographic Experts Group (JPEG) standard is a block-based compression method using discrete cosine transform to transfer original images to cosine domain. It is a complicated compression technique consisting of several fundamental compression methods such as the differential compression, quantization, and Huffman compression. It also utilizes some data processing steps to get better compression performance including offset and Zigzag coding pattern. JPEG is often utilized in photo compression since its compression ratio is excellent. The drawback of JPEG is that it loses some image data resulted from quantization errors. Lempel-Ziv-Welch (LZW) is also a popular compression standard which uses a dynamic code book to encode image data [4]. PCX is another compression standard used in PC Paintbrush Bitmap Graphic, supporting binary, 16 colors, 256, and true color images [5]. It is a lossless compression algorithm and originally from z-soft company. Basically, PCX uses run-length encoding to compress image data in some particular area of pixel values. However, the compression performance is not good if the particular compression area does not have a high repeatability of image data.

In this paper, we propose a modified PCX compression algorithm to overcome the drawback of the PCX compression. We designed a computer package to implement the modified PCX algorithm using java programming language. The Unified Modeling Language (UML) was used to describe the structure and behaviors of the computer package. The pseudo codes for the compression and decompression of the modified PCX algorithm are also provided in this paper. We also conducted an experiment to compare the compression performance between the original PCX and the modified PCX algorithms. Experimental results show that the proposed algorithm is better than the original one in compression ratio.

2 Fundamentals

2.1 The structure of Image File Formats

In general, there are two portions in an image file: information header and image data. The header portion describes image information and the data portion stores the image data. An image header includes signature, size information, resolution, image type, palette, and compression method. Figure 1 shows a general structure of image information header. The signature is an identifier to recognize a particular image format and also provides version information. The size part gives the width and height of the image. The resolution indicates the length of a pixel, generally described in bits per pixel. The type depicts the nature of the image, i.e., a true color, indexed-color, gray scale, or binary image. The palette is an optional part providing the color information (only available for an indexed-color image). The compression explains the compression method for the image if it supports multi compression methods. Besides, some image files also include some particular fields for their special usages. Most image formats reserve some fields for future usages.

signature: DWORD
width: DWORD
height: : DWORD
resolution: : DWORD
compression: : DWORD
type: DWORD
palette: n * 3 bytes (n = No. of colors) (optional: only available for indexed-color images)

Fig. 1: A general description of an image header.

The detailed header of PCX image format can be found in [2], [5].

2.2 The Original PCX Compression Algorithm

The PCX algorithm utilizes run-length encoding to compress image data. Unlike other multi- passes compression methods, it uses only one pass to perform compression and decompression. It doesn't need to know the whole range of image data when encoding (compressing) current image data. The run-length encoding simply uses two bytes to encode data. The first byte serves as a counter computing the number of successive pixel value in the current run. The second byte stores the repeated value. The compression efficiency depends on the repeatability of the pixel values. To avoid bad compression ratios, PCX encodes only one fourth of image data (pixel values 192 to 255 for 8-bit resolution images). The rest of image data remain the same (uncompressed). To do this, PCX uses the first two most significant bits (MSBs) of the count byte as a flag for compression. The rest of six bits of the count byte represents the repetitive number of the value with a maximum number of 63. The repeated value itself is then stored in the next byte. The decompression procedure takes the reverse steps of the compression procedure [2]. The compression procedure of the original PCX is shown in appendix [1-5].

The following is an illustrative example of PCX compression where notation 0X denotes a hexadecimal number.

- Original data: <u>0X11</u> <u>0X20</u> <u>0X40</u> <u>0X40</u> <u>0XDE</u> <u>0XFE</u> <u>0XFE</u> <u>0XFE</u> <u>0XFE</u> <u>0XFE</u>
- Compressed data: <u>0X11</u> <u>0X20</u> <u>0XC2</u> <u>0X40</u> <u>0XC1</u> <u>0XDE</u> 0<u>XC5</u> <u>0XFE</u>

3 The Modified PCX Algorithm

3.1 Idea

As mentioned above, PCX only compresses the image data in the pixel value ranging from 192 to 255. The compression efficiency depends on the repeatability of the data in the range. If the repeatability is low, the compression performance will be bad. Consider a black blocked image with a size of 200 $\times$ 200 and a resolution of 8 bits. The original size is 40,000 bytes. If we use PCX standard to compress the image, the size of the compressed image is the same (40,000 bytes). However, if we use the pixel range between 0 and 63 as the compressed area, the size of the compressed image is 1600 bytes. It significantly improves the compression performance. In some worse cases, the size of a compressed image could be larger than that of the original one if the repeatability of data in the pixel values 192 to 255 is extreme low.

For an 8-bits resolution image, we divide the image into four individual areas according to the pixel values as follows:

Area 1: pixel values 0 to 63
Area 2: pixel values 64 to 127
Area 3: pixel values 128 to 191
Area 4: pixel values 192 to 255

The main idea of the proposed algorithm is to select the best area to compress image data among the four individual areas.

3.2 UML analyses

We designed a computer package for the proposed modified PCX algorithm. The computer package was implemented by java program language. The computer package provides three main functionalities including image displaying and storing, compression, and decompression. We use UML to describe the behaviors and the architecture of the computer package [6,7].

Figure 2 is the use case diagram to describe the overall behaviors of the package from the viewpoints of users. The three use cases (***Display/Store image***, ***Compress image***, and ***Decompress image***) are associated with the three functionalities of the computer packages with a one-to-one corresponding relationship, respectively.

Figure 3 displays the class diagram of the computer package where the five java classes are used to represent the structure of the computer package including ***PcxApp***, ***Compressor***, ***Decompressor***, ***FileProcessor***, and ***Examiner***. The ***PcxApp*** class is the main class for running the computer package with user interfaces and performs the flow control based on users' behaviors. The ***Compressor*** class is the class to perform the modified PCX compression. The ***Decompressor*** class is the class to perform the modified PCX decompression. The ***FileProcessor*** class serves to process file inputs and outputs, e.g., reading files and converting them into arrays, and vice versa. The ***Examiner*** class can be used to examine if a compression/decompression works correctly. It also provides the function of calculating compression ratios for users. Table 1 shows the detailed descriptions of all methods and selected variables of the five java classes. Figure 4 demonstrates the sequence for the use case of ***Compress image***. Figure 5 displays the sequence for the use case of ***Decompress image***.

3.3 Pseudo Codes

In this section, we present the pseudo codes for the compression and decompression of the proposed modified PCX algorithm. Figure 6 demonstrates the pseudo code of the modified compression algorithm of the modified PCX algorithm. Figure 7 displays the pseudo code of the modified decompression algorithm of the modified PCX algorithm.

4 Experimental Results

In the experiment, we used four pictures to compare the compression performance between the original and modified PCX algorithms. All of them are 256 gray scales images. The first two were originally from PhotoImpact picture library, the third one is a medical image originally from an x-ray picture, and the last one was generated by PC Paintbrush software. Figures 8 to 11 show the four pictures. We used the four original and their color-reversed images as test images for the experiment. The experimental results are shown in Table 2.

5 Conclusions

We introduced the principles of image compressions and the structure of image file formats. We also demonstrated the procedures of compression and decompression of PCX. The original PCX algorithm encodes one fourth of data using run-length encoding. The compression efficiency depends on the repeatability of data in the compressed area. If the repeatability is low, the compression performance will be bad. To improve this, we proposed a modified PCX algorithm which selected the best area for compression. We designed a computer package to implement the modified PCX algorithm using java programming language. The UML was used to describe the structure and behaviors of the computer package. The pseudo codes for the compression and decompression of the modified PCX algorithm are also provided in this paper.

We conducted an experiment to compare the compression performance between the original PCX and the modified PCX algorithms. The experimental results showed that the modified PCX algorithm is better than the original one in compression performance.

As about the directions for future studies, we suggest using some reserved field of the PCX header as a flag to indicate the best area for compression. Besides, it might be interesting topics to utilize some optimization methods to determine the best bit positions for masking according to the repeatability of image data.

References

[1] Gonzalez, C.R., Woods, R.E.: Digital Image Processing, Addison-Wesley Publishing Company, Inc, Reading , MA, USA, 1992

[2] Lian, G-Z: Digital Image Processing, Ru Lin (Scholars Books) Publishing company, Taipei, Taiwan, 2004 (in Chinese)

[3]http://www.owlnet.rice.edu/~elec301/Projects00/wavelet_image_comp/index.html

[4] http://dogma.net/markn/articles/lzw/lzw.htm

[5]http://courses.ece.uiuc.edu/ece390/books/labmanual/graphics-pcx.html

[6] Booch, G, Rumbaugh, J, and Jacobson, I., *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA, USA, 1999.

[7] Rumbaugh, J., Jacobson, I., and Booch, G,, *The Unified Modeling Language Reference Manual Guide*, Addison-Wesley, Reading, MA, USA, 1999.

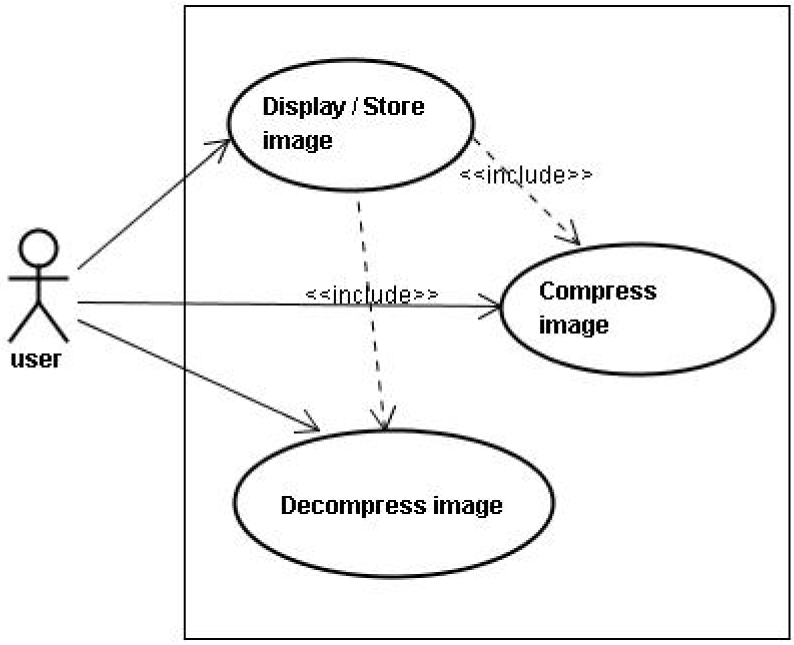


Fig. 2: The use case diagram of the compute package.

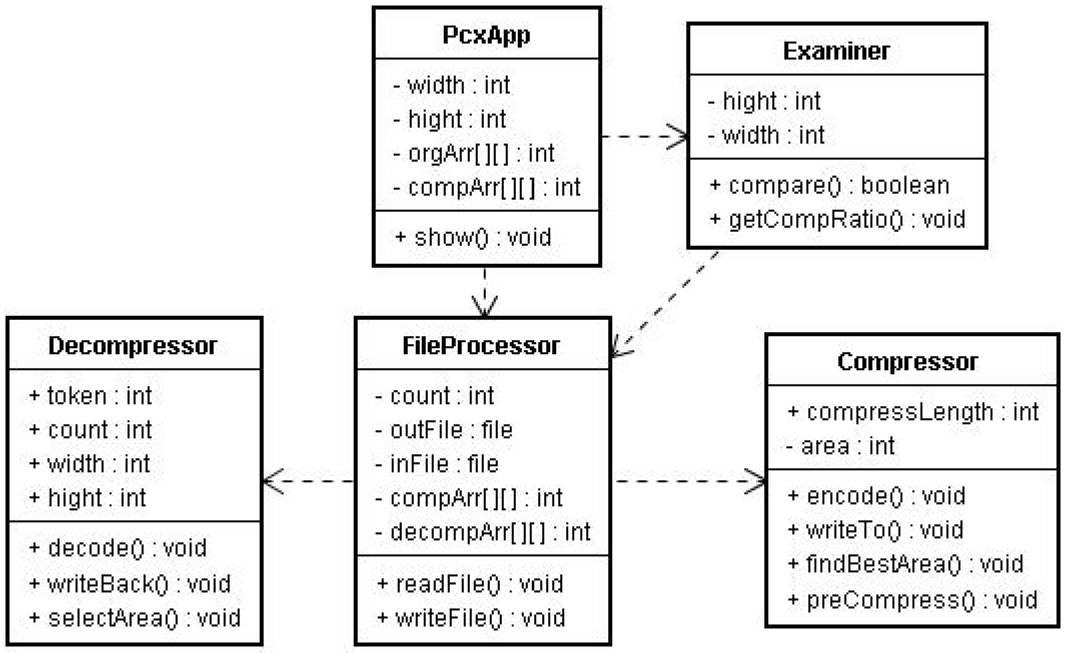


Fig. 3: The class diagram of the computer package.

Table 1 : The descriptions of all method and selected variables for the five java classes in Figure 3.

Class	Variable / Method	Description
PcxApp	width, height	width and height of the image
	oriArr[][]	storing the pixel values of the original image in a two-dimensional array with a size of height × width
	compArr[][]	storing the pixel values of the compressed image in a two-dimensional array with a size of height × width
	show()	display the image
Compressor	area	the area which has the best compression performance
	compLength	length of the compressed image
	endode ()	perform run-length encoding
	writeTo ()	write the byte back without run-length encoding
	findBestArea ()	find the best area for compression
	preCompress ()	pre-compress the image for determining the best area
Decompressor	decode ()	perform run-length decoding
	writeBack ()	write the byte back without run-length decoding
	selectArea ()	select the best area for decompression
FileProcessor	inFile, outFile	input file and output file
	compArr [][]	storing the pixel values of the compressed image in a two-dimensional array with a size of height × width
	deCompArr [][]	storing the pixel values of the uncompressed image in a two-dimensional array with a size of height × width
	readFile ()	read a file
	writeFile ()	write a file
Examiner	compare ()	compare if two files are the same
	getCompRatio ()	get the compression ratio

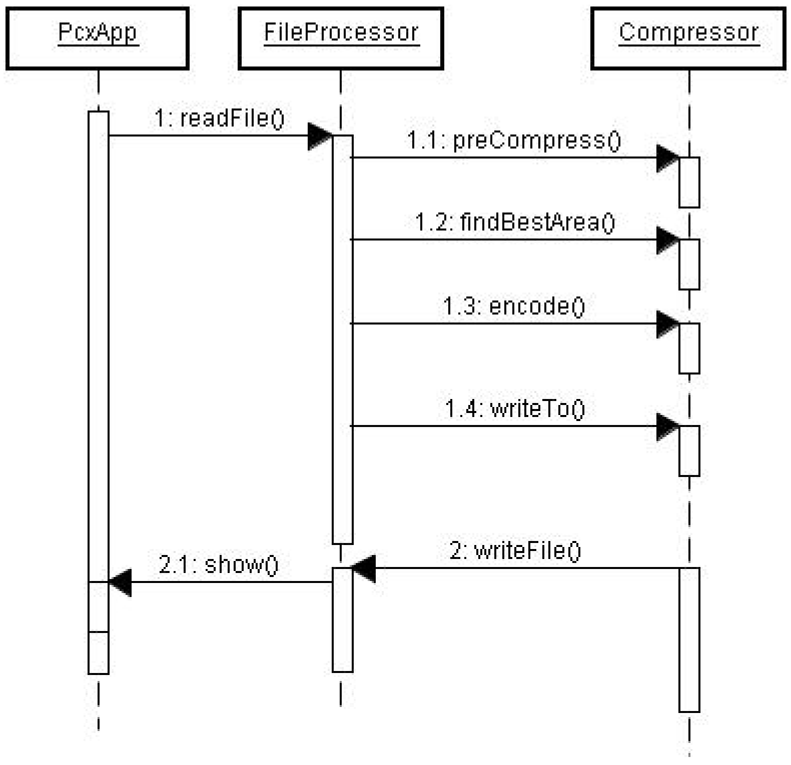


Fig. 4: The sequence diagram for the use case of ***Compress image*** shown in Fig. 2.

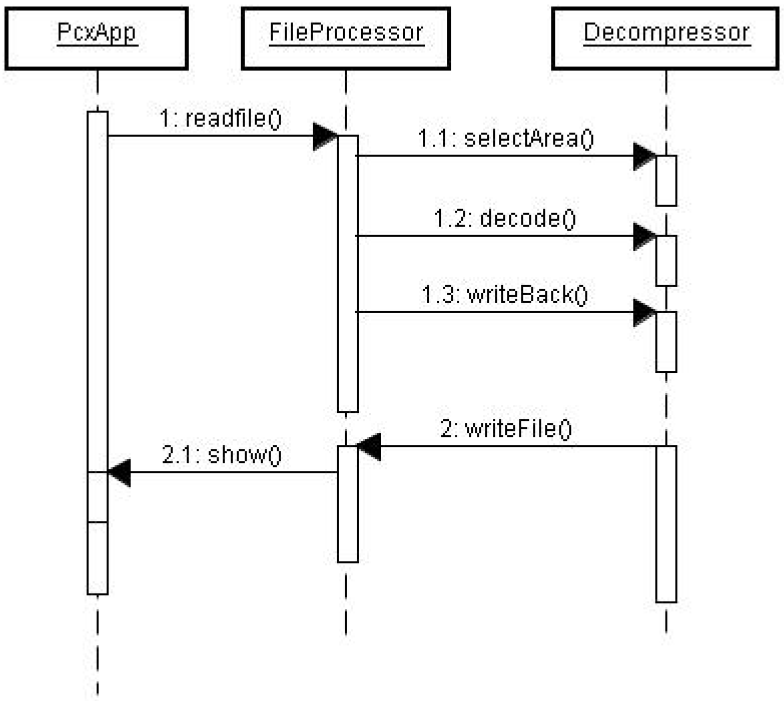


Fig. 5: The sequence diagram for the use case of ***Decompress image*** shown in Fig. 2.

```
offset := area * 64;
FOR i := 0 TO height -1
    count := 0;
    old_token := orgArr[i][0];
    FOR j := 0 TO width-1
        token := orgArr[i][j];
        IF ((token BITWISE AND 192) IS NOT EQUAL TO mask) THEN
            compressedArr [index] := token
            index := index +1
        ELSE
            old_token := orgArr[i][j];
            token := orgArr[i][j];;
            count := 0;
            WHILE ((token IS EQUAL TO old_token) AND (j < width) AND
                        (count < 63)) DO
                count := count +1
                j := j +1
            END DO
            IF (j < width) THEN
                token := orgArr[i][j]
            END IF
        END IF
        compressedArr[index] := count + offset
        index := index +1
        compressedArr[index] := old_token
        index := index +1
        old_token := token
        j := j+1
    END FOR j
```

----

where
width = image width
height = image height,
index = encoding index,
orgArr[height][width] = 2-dimensional array storing the original image data with a size of height $\times$ width
compressedArr[l] = 1-dimensional array storing the compressed data with a size of l
token = byte to be encoded
offset = adjusting value for the selected compression area
area = the compression area
mask = the mask to determine the compression area

Remark: := denotes an assignment operation.

Fig. 6: The pseudo code of the modified PCX compression algorithm.

```
h := 0
w := 0
FOR i := 0 TO length-1
    token := compressedArr[i];
   IF ( (token BITWISE AND 192) IS NOT EQUAL TO mask ) THEN
      decompressedArr[h][w] := token
      w := w +1
   ELSE
       count := token - offset
       i := i + 1
       token := compressedArr[i]
       FOR j := 0 TO count-1
           decompressedArr[h][w] := token
           w := w +1
       END FOR
   END IF
   IF (w >= width) THEN
     h := h +1
     w := 0
   END IF
END FOR
------------------------------------------------------------------
where
h = iterative index for image height
w = iterative index for image width
decompressedArr[height][width] = two-dimensional array storing the decompressed data with a size
                                                    of height × width
length = the length of compressedArr. END FOR i
Remark: := denotes an assignment operation.
```

Fig. 7: The pseudo code of the modified PCX decompression algorithm.

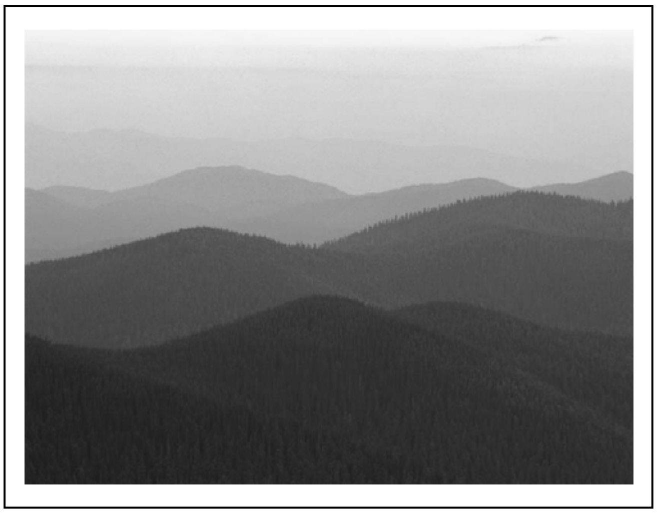


Fig. 8: Test image 1 (size: 800 × 600, 256 gray scales)

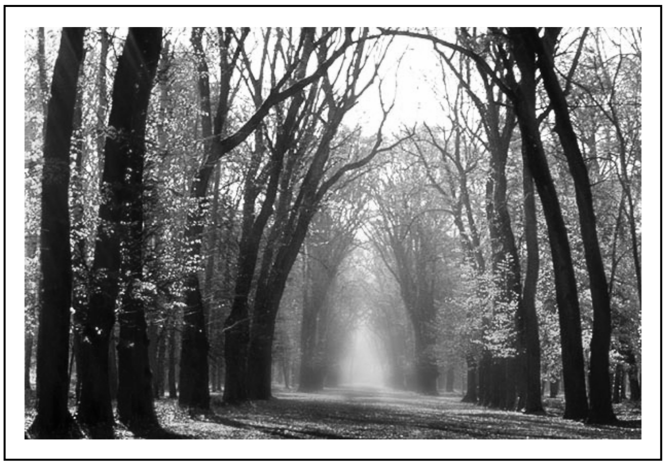


Fig. 9: Test image 2 (size: 600 × 400, 256 gray scales)

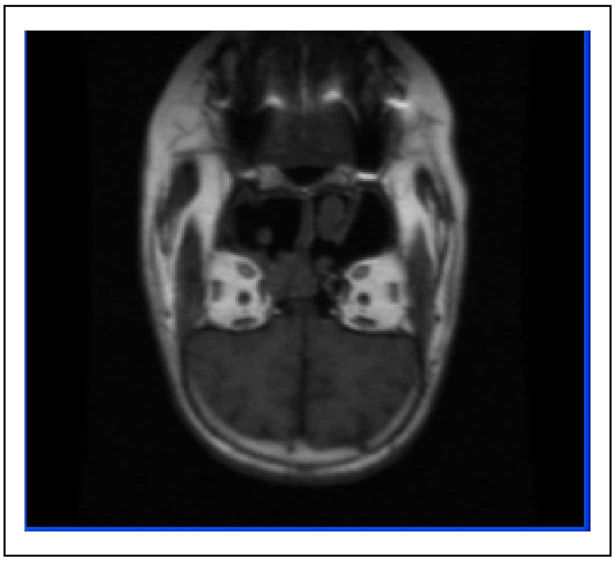


Fig. 10: Test image 3 (size: 256 × 256, 256 gray scales)

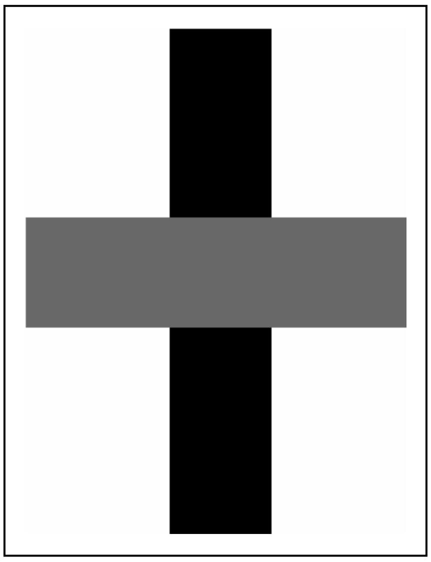


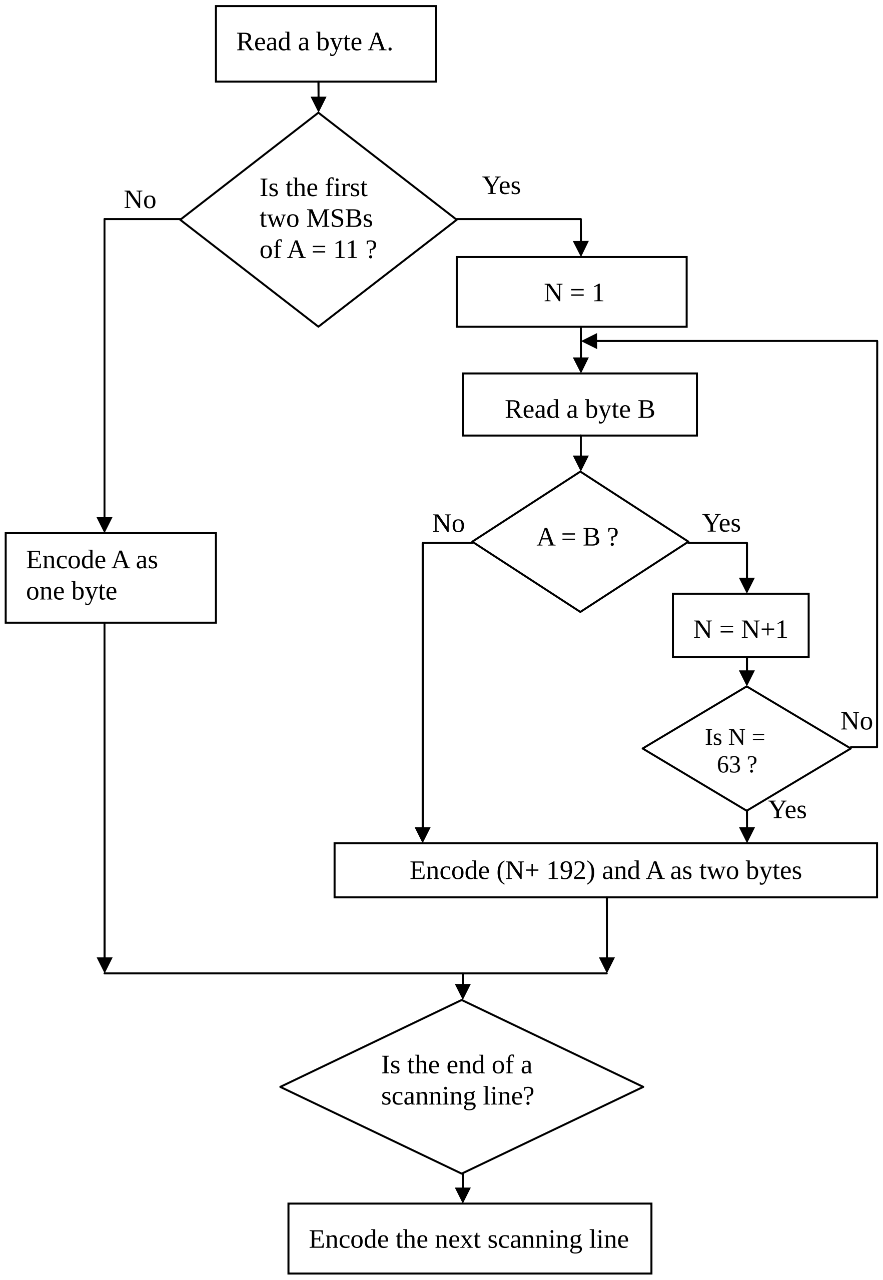
Fig.11: Test image 4 (size: 300 × 400, 256 gray scales)

Table 2: The experiment results

Method	Compression ratio (Compressed size / Original size)							
	Test Image 1		Test Image 2		Test Image 3		Test Image 4	
	Ori.	Rev.	Ori.	Rev.	Ori.	Rev.	Ori.	Rev.
Original PCX	0.92	1.16	1.12	1.27	1.00	0.95	0.45	0.80
Modified PCX	0.92	0.92	1.12	1.12	0.95	0.95	0.45	0.45

Remark: Ori. = original image; Rev. = color-reversed image

Appendix: The compression procedure of the original PCX [1-5].


Read a byte A.
Is the first two MSBs of A = 11 ?
No
Yes
N = 1
Read a byte B
A = B ?
No
Yes
N = N+1
Is N = 63 ?
No
Yes
Encode A as one byte
Encode (N+ 192) and A as two bytes
Is the end of a scanning line?
Encode the next scanning line