A lossless approach to bi-level and Grey-scale image Compression using 2-dimensional run-length encoding algorithm

Subhabrata Bhattacharya

B.E. **7th** Semester,
Computer Science & Engineering,
Asansol Engineering College

Acknowledgments

I pay my sincere regards to Prof. Bhabatosh Chanda, Electronics and Communication Sciences Unit, Computer and communication Sciences Division, Indian Statistical Institute, Kolkata; without whose guidance the project could never been accomplished.

References

Digital Image Processing, Second Edition
Rafael C. Gonzalez and Richard E. Woods
Pearson Education Inc**. 2002**

Software Environment and tools used

The entire software is written in ANSI C programming language in GNU/Linux (kernel 2.4.7-10) platform. The software is compiled using Gnu C compiler version 2.96 and entirely free for use and redistribution as long as the terms and conditions of the GNU Public License are satisfied.

General Constraints

[1] The software is currently developed to work only in Unix-based platforms and cannot be used under Windows based systems.
[2] Currently the software is fully functional for monochrome Windows bitmap files. Extended support for OS/2 bitmaps and other file-formats are under development.
[3] Support for color and gray scale images are under development.

1. Introduction

Every day, an enormous amount of information is stored, processed and transmitted digitally. Companies provide business associates, investors and potential customers with financial data, annual reports and inventory and product information over the Internet. Order entry and tracking, two of the most basic online transactions are routinely performed from the comfort of one's own home. Because much of this on-line information is graphical or pictorial in nature, the storage and communication requirements are immense. Methods of compressing the data prior to storage and/or transmission are of significant practical and commercial interest.

Image compression addresses the problem of reducing the amount of data required to represent a digital image. The underlying basis of the reduction process is the removal of redundant data. From a mathematical viewpoint, this amounts to transforming a 2-D pixel array into a statistically uncorrelated data set. The transformation is applied prior to storage or transmission of the image. At some later time, the compressed image is decompressed to reconstruct the original image or an approximation of it.

Image compression techniques broadly fall into two categories: information preserving (loss-less) and lossy. The former allows an image to be compressed and decompressed without losing information while the latter provide higher degrees of data reduction but result in a less than perfect reproduction of the original image. Loss-less image compression techniques are useful in image archiving. Lossy image compression techniques are used in television broadcast and video-conferencing where a certain degree of error is an acceptable trade-off for increased compression performance. An image that can be compressed must have data that are non-essential and simply restate that which is already known. This is called data redundancy. The type of data redundancy that is being exploited to compress an image in this context is interpixel redundancy. Because the value of any pixel can be predicted from the value of its neighbours, the information carried by the individual pixels is relatively small. Much of the visual contribution of a single pixel to an image is redundant; it could have been guessed on the basis of the values of its neighbors. To reduce the interpixel redundancies in an image, the 2-D pixel array normally used for human viewing and interpretation must be transformed into a more efficient (but usually "non-visual") format. In case of loss-less image compression, transformations of this type are reversible in nature because the image can be reconstructed from the transformed data set.
The algorithm implemented in the project is an improvement over the one-dimensional run-length algorithm. In the one-dimensional algorithm the correlation between contiguous pixels in a row is exploited and this in turn reduces the inter-pixel redundancy. While in its improved counterpart the key to elimination of inter-pixel redundancy lies in exploitation of the correlation between the adjacent rows in addition to the correlation that is already discussed. Although the method is complicated and time resource intensive; yet it ensures higher degree of compression. It has been seen that for any row of a given image, the rows adjacent to it are more or less identical i.e. the gray-level distribution of pixels in adjacent rows of an image is somewhat uniform.

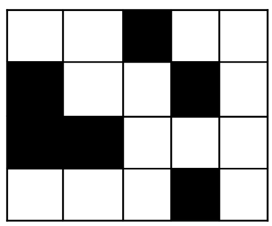
2. Working Methodology

The project broadly consists of two softwares: the encoder and the decoder. In this context we describe each of them in detail. Both the encoder and decoder are subdivided into different modules and the way control and data flow from one module to the other are being shown in the subsequent schematic diagrams

Encoder: The encoder has the following modules : File format reader, Matrix generator, 2D runlength encoder, ASCII converter and File formatter

File format reader: This module is responsible for taking input from the user in the form of different image files. It is programmed to read the file header and pass the necessary information like the image resolution, color content and algorithm used to encode the image is passed to the next module. Currently it has been programmed to accept only Windows bitmap files (.bmp) with only two gray levels (black & white) but the support for different file formats and multiple gray-level can be increased with minor modifications in the existing software.

Matrix generator: On the basis of the information gathered from the previous module, it generates a two-dimensional array or matrix containing the gray-level value of each pixels of the image. Thus the resolution of the image (width and height), color content and method used while encoding the image becomes an absolute must for this module to generate an accurate matrix that maps each of its elements to the corresponding position as well as the gray-level value at that position in the actual image.



A 5x4 monochrome bitmap image

$$\Leftrightarrow \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Equivalent binary matrix

While dealing with color or gray-scale images that have more than just 2 colors, we first decompose the gray coded image into a series of binary images. This is called bit-plane decomposition. The gray-levels of an m-bit gray-scale image can be represented in the form of the base 2- polynomial:

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \ldots + a_1 2^1 + a_0 2^0$$

The coefficients of the polynomial (a_i) can only have values 0 and 1

$$g_i = a_i \oplus a_{i+1} \quad 0 \leq i \leq m-2$$
$$g_{m-1} = a_{m-1}$$

Thus a pixel with gray-level 244 can be represented in the binary form as:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	0	0	0	0

The corresponding gray code is calculated ass follows:

$g_{8-1} = a_{8-1}$ i.e. $g_7 = a_7 = 1$
$g_6 = a_6 \quad a_7 = 1 \quad 0 = 1$
$g_5 = a_5 \quad a_6 = 0 \quad 0 = 0$
$g_4 = a_4 \quad a_5 = 1 \quad 0 = 1$
$g_3 = a_3 \quad a_4 = 0 \quad 1 = 1$
$g_2 = a_2 \quad a_3 = 0 \quad 0 = 0$
$g_1 = a_1 \quad a_2 = 0 \quad 0 = 0$
$g_0 = a_0 \quad a_1 = 0 \quad 0 = 0$

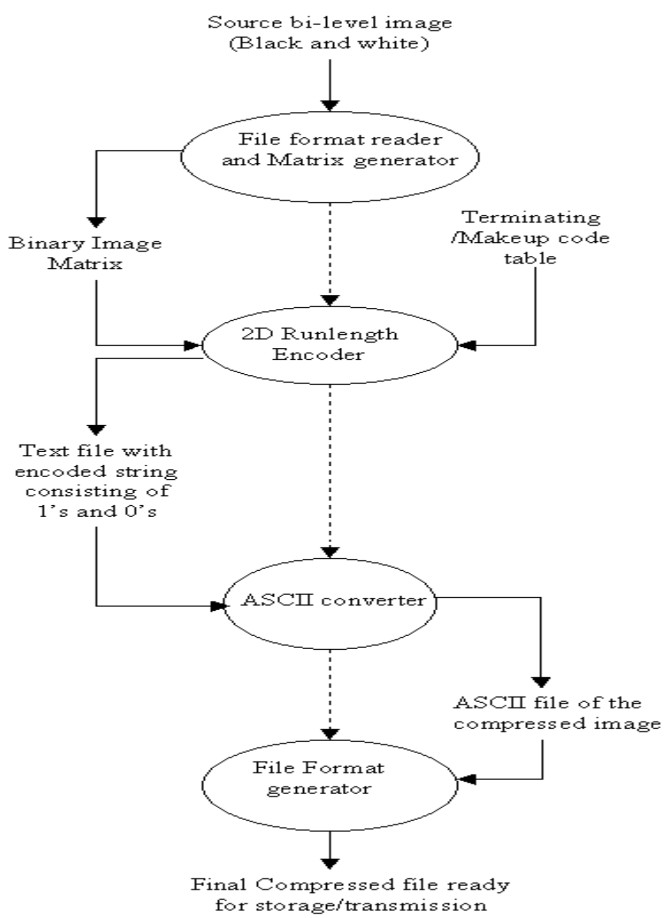
Hence gray code corresponding to the pixel with gray-level 244 is 11011000. The gray coded image is now decomposed into m 1-bit planes. This is done to avoid complexity in the bit planes that is created due to small changes in gray levels of the individual pixels.
Thus for an 8-bit image 8 separate binary matrices are generated from the gray codes and the matrices are passed as input to the next module.

2D runlength encoder: This module is responsible for the encoding. Each row from the matrix that is input from the previous module is encoded with the help of some predetermined code constraints. The 2D runlength-encoding algorithm is discussed shortly in the next section. The matrix is converted into a string of 0's and 1's and the string is written into a temporary file, which is passed to the next module for further processing.

ASCII converter: It accepts the file containing the string of 0's and 1's and taking 8 characters at a time, it converts it into an equivalent ASCII characters and the stream of ASCII characters are further written into a file (which is 1/8th of the previous intermediate file) and is being passed to the final module for last rites.

File formatter: This is the ultimate module of the encoder software, which generates a header from the file information previously fetched by the file format reader, and adds this header to the intermediate ASCII file. This file is now ready for future storage/transmission and decompression process.

Schematic Diagram of the Encoder:

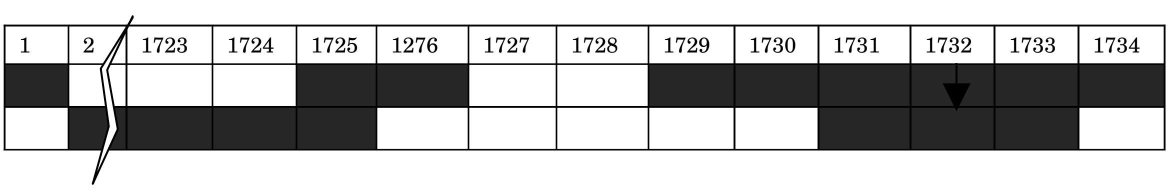


Description of the Algorithm used

The 2-dimensional runlength coding approach adopted for both the CCITT Group 3 and 4 standards is a line-by-line method in which the position of each black-to-white or white-to-black run transition is coded with respect to the position of a reference element a0 that is situated on the current coding line; the reference line for the first line of each new image is an imaginary white line. The basic coding process for a single scan line is shown in the flowchart that follows this section. A changing element is defined as a pixel whose value is different from that of the previous pixel on the same line. The most important changing element is a0 (the reference element), which is either set to the location of an imaginary white "changing element" to the left of the first pixel of each new coding line or determined from the previous coding mode. After a0 is located, a1 is identified as the location of the next changing element to the right of a0 on the current coding line, a2 as the next changing element to the right of a1 on the same coding line, b1 as the changing element of the opposite value of (a0) and to the right of a0 on the reference (or previous) line, and b2 as the next changing element to the right of b1 on reference line. If any of these changing elements are not detected, they are set to the location of an imaginary pixel to the right of the last pixel on the appropriate line.

After identification of the current reference element and associated changing elements, two simple tests are performed to select one of the three possible coding modes: pass mode, horizontal mode or vertical mode. The initial test, which corresponds to the first branch point of the following flow-chart, compares the location of b2 to that of b1. The second test, which corresponds to the second branch-point of the flow chart, computes the distance (in pixels) between the locations of a1 and b1 and compares it against 3. Depending on the outcome of these tests, one of the three outlined coding blocks of the flow chart is entered and the appropriate coding process is executed .A new reference element is then established, as per the flow chart, in preparation for the next coding iteration.

Specific codes are defined, that are to be used for each of the three possible coding modes. In pass mode, which specifically excludes the case in which b2 is directly above a1, only the pass mode codeword 0001 is needed. This mode identifies white or black reference line runs that do not overlap the current white or black coding line runs. In horizontal coding mode, the distance from a0 to a1 and a1 to a2 must be coded in accordance with the terminating and makeup codes defined by CCITT and then appended to the horizontal mode code word 001. This is indicated by the notation 001 + M (a0 ~ a1) + M (a1 ~a2), where a0 ~ a1 and a1 ~ a2 both denote the distances from a0 to a1 and a1 to a2, respectively.
For example, consider the image line



Here we consider a pair of adjacent rows from an image. The first changing element for each new line i.e. a0 is set to the left of the first pixel of the line so a0 = 0, a1 = 1 etc. Likewise in the second iteration we have the next changing element is 1725 because at 1725 a transition from a continuous white run to black occurs. Finally, in vertical coding mode, one of the six special variable-length codes is assigned to the distance between a1 and b1.

To summarize the various coding modes can be shown as:

Mode	Code Word
Pass	0001
Horizontal	001 + M (a0 ~ a1) + M (a1 ~ a2)
Vertical	
A1 below b1 (a1 ~ b1 = 0)	1
A1one to the right of b1 (a1 ~ b1 = -1)	011
A1 two to the right of b1 (a1 ~ b1 = -2)	000011
A1 three to the right of b1 (a1 ~ b1 = -3)	0000011
A1 one to the left of b1 (a1 ~ b1 = 1)	010
A1 two to the left of b1 (a1 ~ b1 = 2)	000010
A1 three to the left of b1 (a1 ~ b1 = 3)	0000010
Extension	0000001xxx

The extension mode code word at the bottom of the table is used to enter an optional facsimile-coding mode. For example, the 0000001111 codeword is used to initiate an uncompressed mode of transmission.

The runlengths can be obtained from the CCITT terminating and make up code tables that are given as follows.

CCITT Terminating Codes

Run Length	**White Code Word**	**Black Code Word**	**Run Length**	**White Code Word**	**Black Code Word**	**Run Length**	**White Code Word**	**Black Code Word**
0	110101	110111	22	11	110111	44	101101	1010100
1	111	10	23	100	101000	45	100	1010101
2	111	11	24	101000	10111	46	101	1010110
3	1000	10	25	101011	11000	47	1010	1010111
4	1011	11	26	10011	11001010	48	1011	1100100
5	1100	11	27	100100	11001011	49	1010010	1100101
6	1110	10	28	11000	11001100	50	1010011	1010010
7	1111	11	29	10	11001101	51	1010100	1010011
8	10011	101	30	11	1101000	52	1010101	100100
9	10100	100	31	11010	1101001	53	100100	110111
10	111	100	32	11011	1101010	54	100101	111000
11	1000	101	33	10010	1101011	55	1011000	100111
12	1000	111	34	10011	11010010	56	1011001	101000
13	11	100	35	10100	11010011	57	1011010	1011000
14	110100	111	36	10101	11010100	58	1011011	1011001
15	110101	11000	37	10110	11010101	59	1001010	101011
16	101010	10111	38	10111	11010110	60	1001011	101100
17	101011	11000	39	101000	11010111	61	110010	1011010
18	100111	1000	40	101001	1101100	62	110011	1100110
19	1100	1100111	41	101010	1101101	63	110100	1100111
20	1000	1101000	42	101011	11011010			
21	10111	1101100	43	101100	11011011			

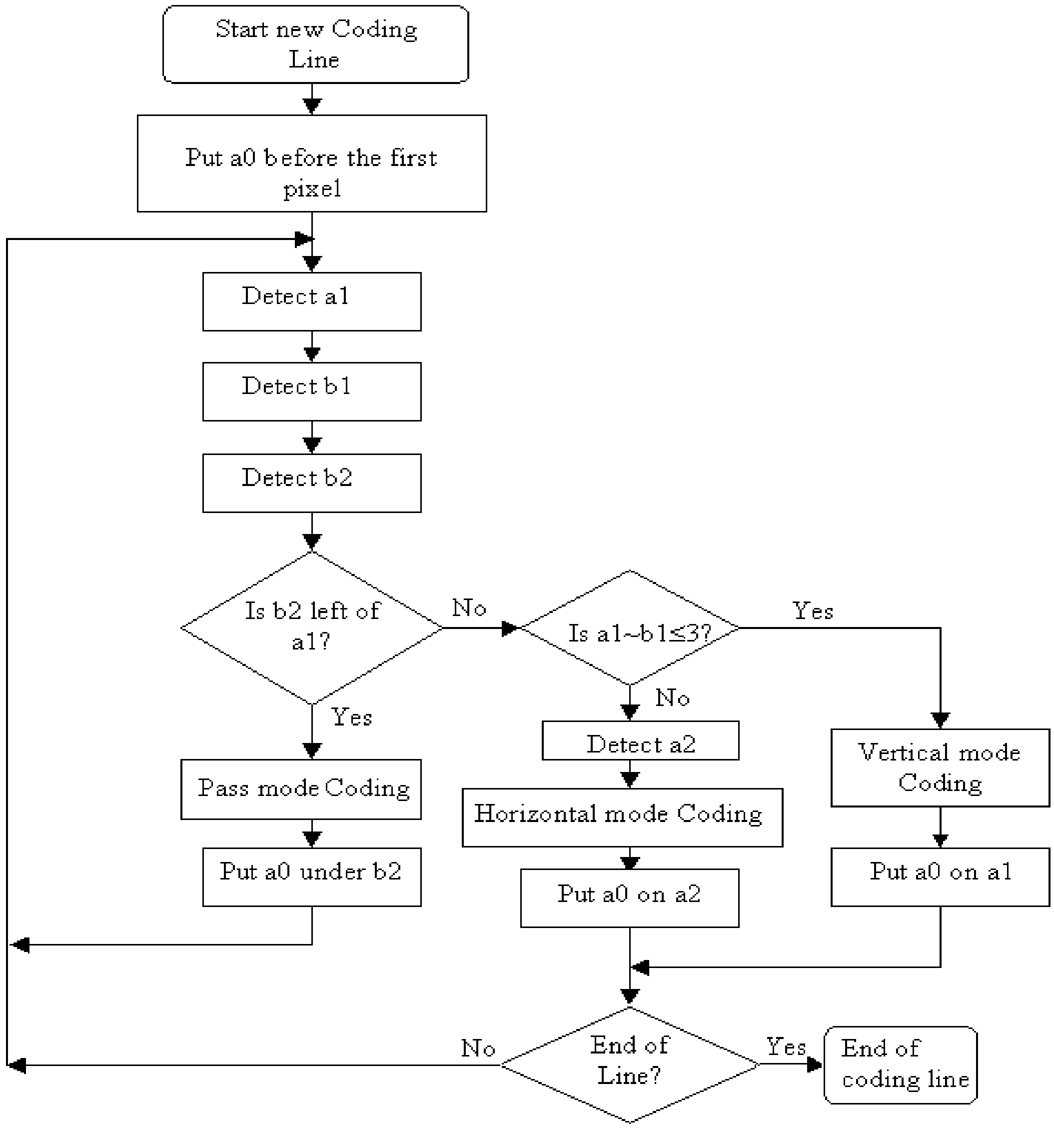
CCITT Makeup Codes

Run Length	White Code Word	Black Code Word		Run Length	White Code Word	Black Code Word
64	11011	0000001111		960	011010100	0000001110011
128	10010	000011001000		1024	011010101	0000001110011
192	010111	000011001001		1088	011010110	0000001110101
256	0110111	000001011011		1152	011010111	0000001110110
320	00110110	000000110011		1216	011011000	0000001110111
384	00110111	000000110100		1280	011011001	0000001010010
448	01100100	000000110101		1344	011011010	0000001010011
512	01100101	0000001101100		1408	011011011	0000001010100
576	01101000	0000001101101		1472	010011000	0000001010101
640	01100111	0000001001010		1536	010011001	0000001011010
704	011001100	0000001001011		1600	010011010	0000001011011
768	011001101	0000001001100		1664	011000	0000001100100
832	011010010	0000001001101		1728	010011011	0000001100101
896	011010011	0000001110010				

Run Length	Code Word		Run Length	Code Word
1792	00000001000		2240	000000010110
1856	00000001100		2304	000000010111
1920	00000001101		2368	000000011100
1984	000000010011		2432	000000011101
2048	000000010011		2496	000000011110
2112	000000010100		2560	000000011111
2176	000000010101			

And finally the unique codeword 000000000001 is used to indicate the end of each coding line and is concatenated with each line of coded strings. This end-of-line indicator is used to signal the first line of each new image. Thus with the number of end-of-line indicators we can easily evaluate the number of rows in the image.

Flow-Chart depicting the 2D runlength coding process



Decoder

The decoder has the following modules: File header reader, Information extractor, ASCII to binary converter Line decoder, Matrix generator, Matrix to image file converter

<u>File header reader</u>: This module is programmed to read the file header. It validates the input (in this case, the file) to be used by the subsequent modules. It also sends the raw header data to the next module for further processing and usage.

<u>Information extractor</u>: It accepts the raw-header that is being passed to it and converts it to useful information that is required by subsequent modules in the entire decoding process. Numbers of columns in the image, colors used are being passed to the modules working in the next stage.

ASCII to binary converter: This module accepts raw ASCII data stream without the header. Each ASCII character is converted into its binary equivalent (a cluster of eight 1's and 0's). Thus a stream of 1's and 0's is generated which starts and ends with an end-of-line indicator, 000000000001. The end-of-line count gives the number of rows in the image.

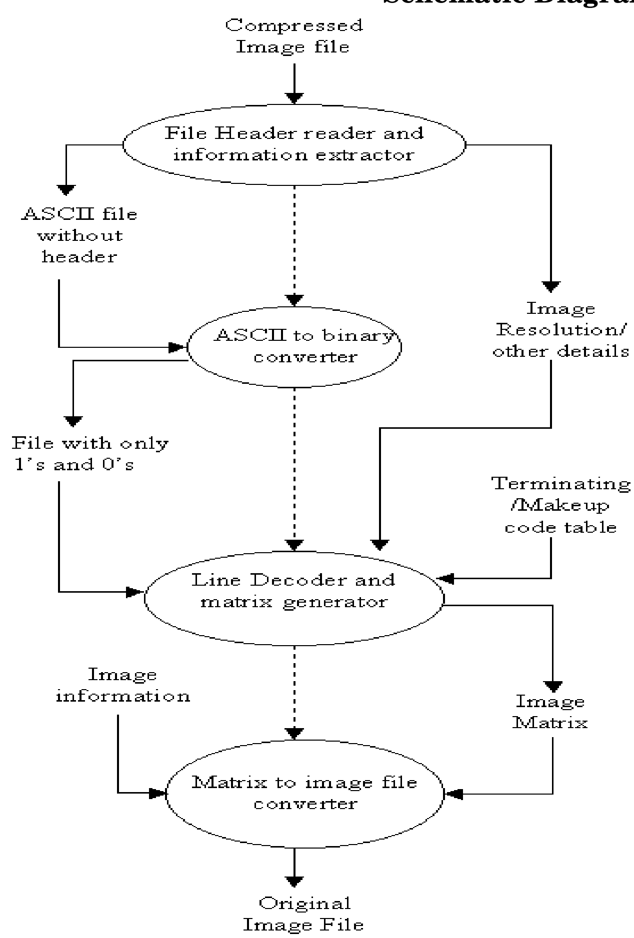
Line decoder: This module is responsible for extracting an encoded string delimited by two end-of-line indicators and decoding it. The code words are separated from each other by exhaustive look-up in the code-tables through efficient search mechanism and appropriate values of a0, a1 and a2 are calculated from them. These values are passed to the next module.

Matrix generator: It accepts the value of a0, a1 and a2 as being passed by the previous module and develops individual rows of the matrix. For example, for each row a0 is initially set to 0. a1 is calculated from the mode of coding as depicted from the code word by the Line decoder. So starting from a0 to the position just left of a1 are filled with white. a1 onwards will be filled with the color opposite of a0 i.e. black and so on.

The above two modules need to work simultaneously to build the matrix.

Matrix to image file converter: The matrix that is generated is being passed to the ultimate stage of the decoder, which is responsible for the converting the matrix into a bitmap file format or a format that is easily readable by any image-viewer software.

Schematic Diagram of the Decoder



Statistical Comparison

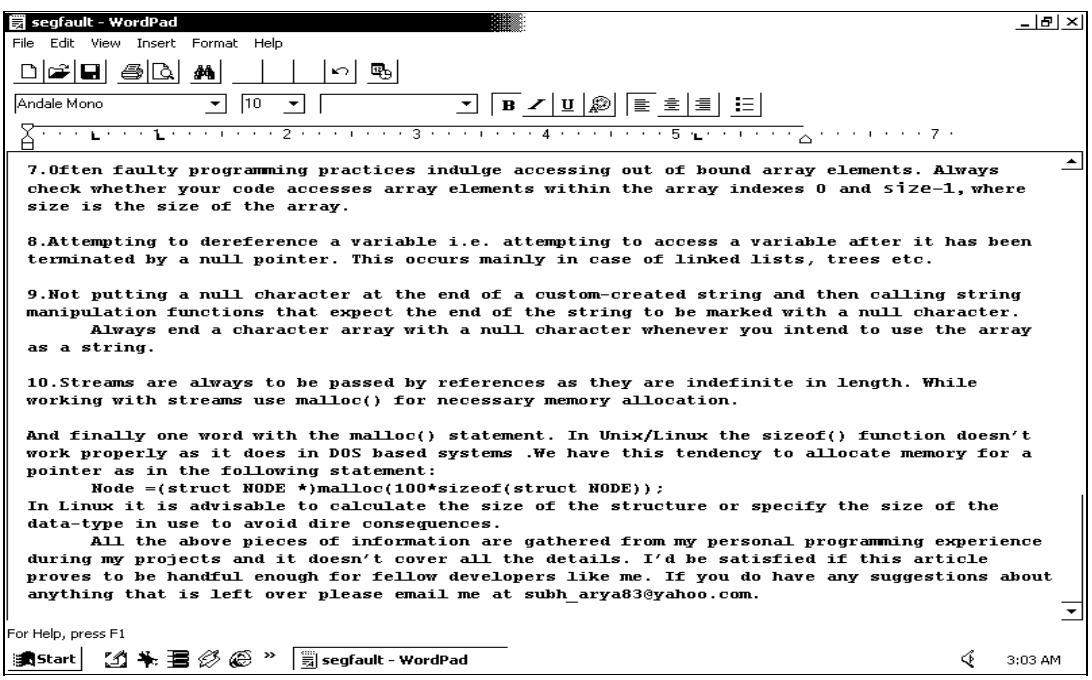
Here a standard Windows bitmap image of resolution 800 x 600, saved under the Windows monochrome bitmap scheme is saved using other popular encoding algorithm and the results are given as follows:

Algorithm/File format used	Size in KB	Type of Compression
Original Image	58.594	----------
BMP (Windows Bitmap)	58.654	Loss less
JPEG	126.51	Lossy
GIF (Graphics Interchange Format)	16.025	Loss less
PNG (Portable Network Graphics)	9.067	Loss less
TIFF (Tagged Image File Format)	58.871	Loss less
PBM (Portable Bit Map)	58.627	Loss less
TDR	13.969	Loss less

The study reveals that the compression achieved in the "TDR" file format (as generated by the encoder software), implemented through 2-dimensional runlength coding algorithm is significantly higher than most of the popular encoding techniques.
The same procedure was applied with 10 other monochrome bitmap files and an average compression ratio of 5:1 was observed.

The technique described in the above context can prove to be highly effective to store and transfer documents. Large documents that are saved as image files like this can be easily transmitted over a network because of their less file size.

Image samples on which the algorithm is tested:
a) Figure 1 (doc1.bmp)


segfault - WordPad

File Edit View Insert Format Help

Andale Mono 10

7.Often faulty programming practices indulge accessing out of bound array elements. Always check whether your code accesses array elements within the array indexes 0 and size-1, where size is the size of the array.

8.Attempting to dereference a variable i.e. attempting to access a variable after it has been terminated by a null pointer. This occurs mainly in case of linked lists, trees etc.

9.Not putting a null character at the end of a custom-created string and then calling string manipulation functions that expect the end of the string to be marked with a null character.
Always end a character array with a null character whenever you intend to use the array as a string.

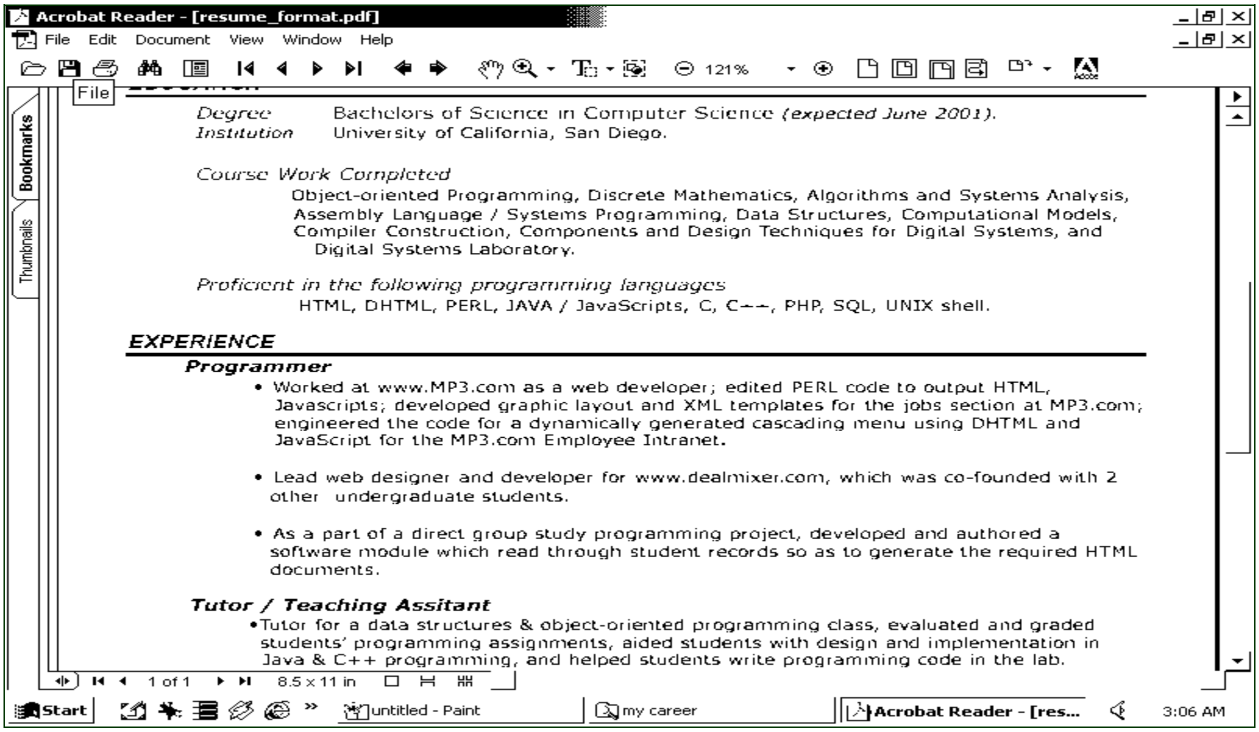
10.Streams are always to be passed by references as they are indefinite in length. While working with streams use malloc() for necessary memory allocation.

And finally one word with the malloc() statement. In Unix/Linux the sizeof() function doesn't work properly as it does in DOS based systems .We have this tendency to allocate memory for a pointer as in the following statement:
Node =(struct NODE *)malloc(100*sizeof(struct NODE));
In Linux it is advisable to calculate the size of the structure or specify the size of the data-type in use to avoid dire consequences.
All the above pieces of information are gathered from my personal programming experience during my projects and it doesn't cover all the details. I'd be satisfied if this article proves to be handful enough for fellow developers like me. If you do have any suggestions about anything that is left over please email me at subh_arya83@yahoo.com.

For Help, press F1

Start segfault - WordPad 3:03 AM

b) Figure 2 (doc2.bmp)



Acrobat Reader - [resume_format.pdf]

Degree Bachelors of Science in Computer Science *(expected June 2001).*
Institution University of California, San Diego.

Course Work Completed
Object-oriented Programming, Discrete Mathematics, Algorithms and Systems Analysis, Assembly Language / Systems Programming, Data Structures, Computational Models, Compiler Construction, Components and Design Techniques for Digital Systems, and Digital Systems Laboratory.

Proficient in the following programming languages
HTML, DHTML, PERL, JAVA / JavaScripts, C, C++, PHP, SQL, UNIX shell.

EXPERIENCE

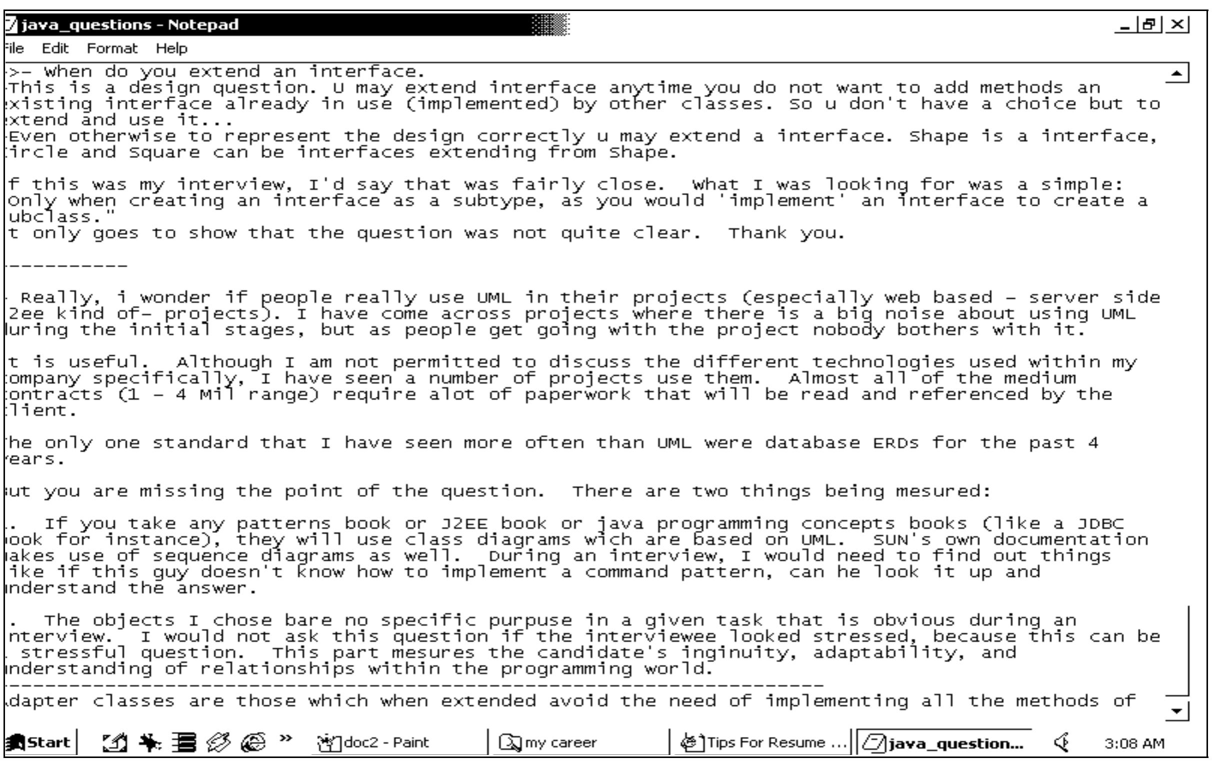
Programmer

- Worked at www.MP3.com as a web developer; edited PERL code to output HTML, Javascripts; developed graphic layout and XML templates for the jobs section at MP3.com; engineered the code for a dynamically generated cascading menu using DHTML and JavaScript for the MP3.com Employee Intranet.
- Lead web designer and developer for www.dealmixer.com, which was co-founded with 2 other undergraduate students.
- As a part of a direct group study programming project, developed and authored a software module which read through student records so as to generate the required HTML documents.

Tutor / Teaching Assitant

- Tutor for a data structures & object-oriented programming class, evaluated and graded students' programming assignments, aided students with design and implementation in Java & C++ programming, and helped students write programming code in the lab.

c) Figure 3 (doc3.bmp)

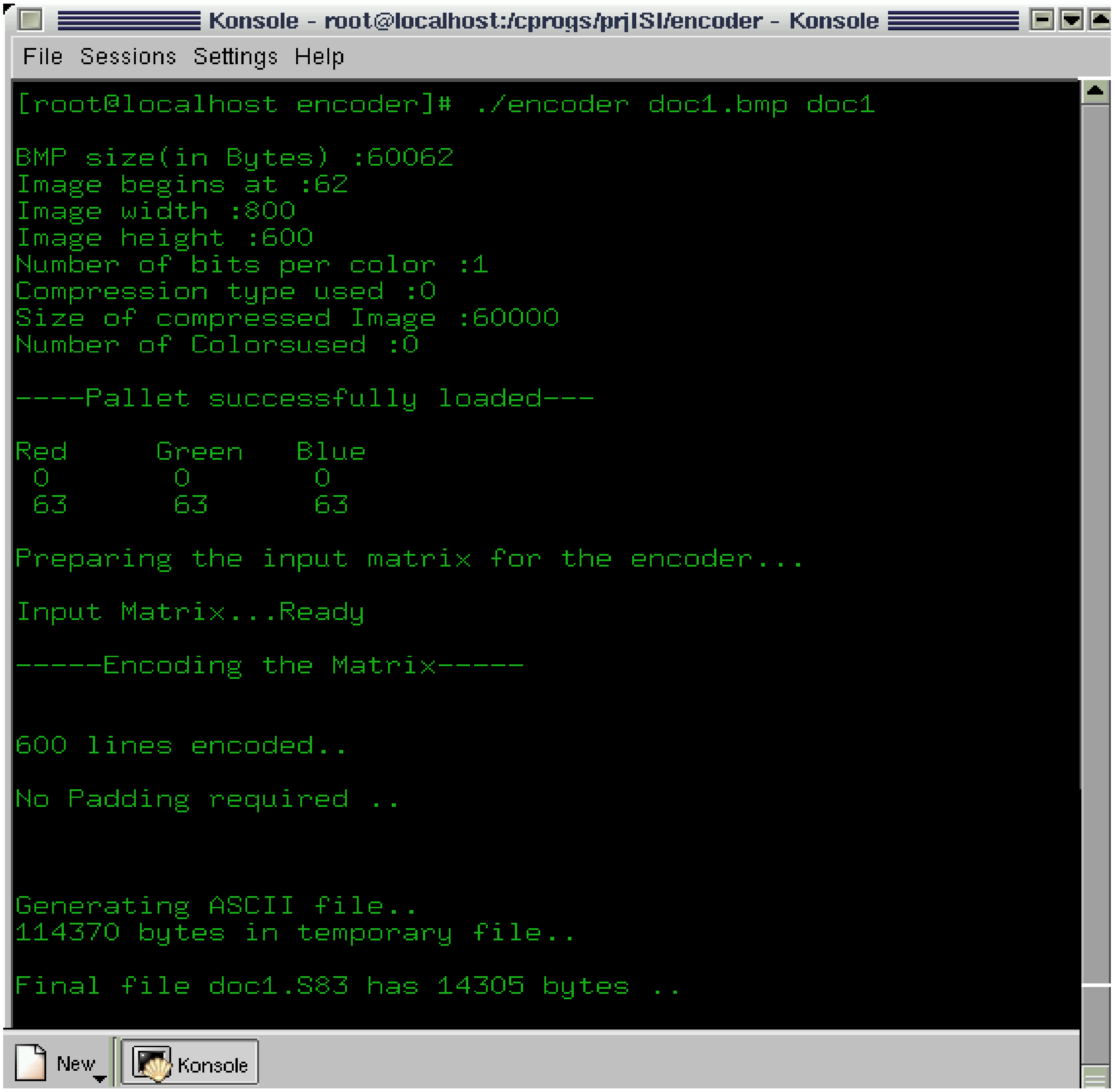


java_questions - Notepad

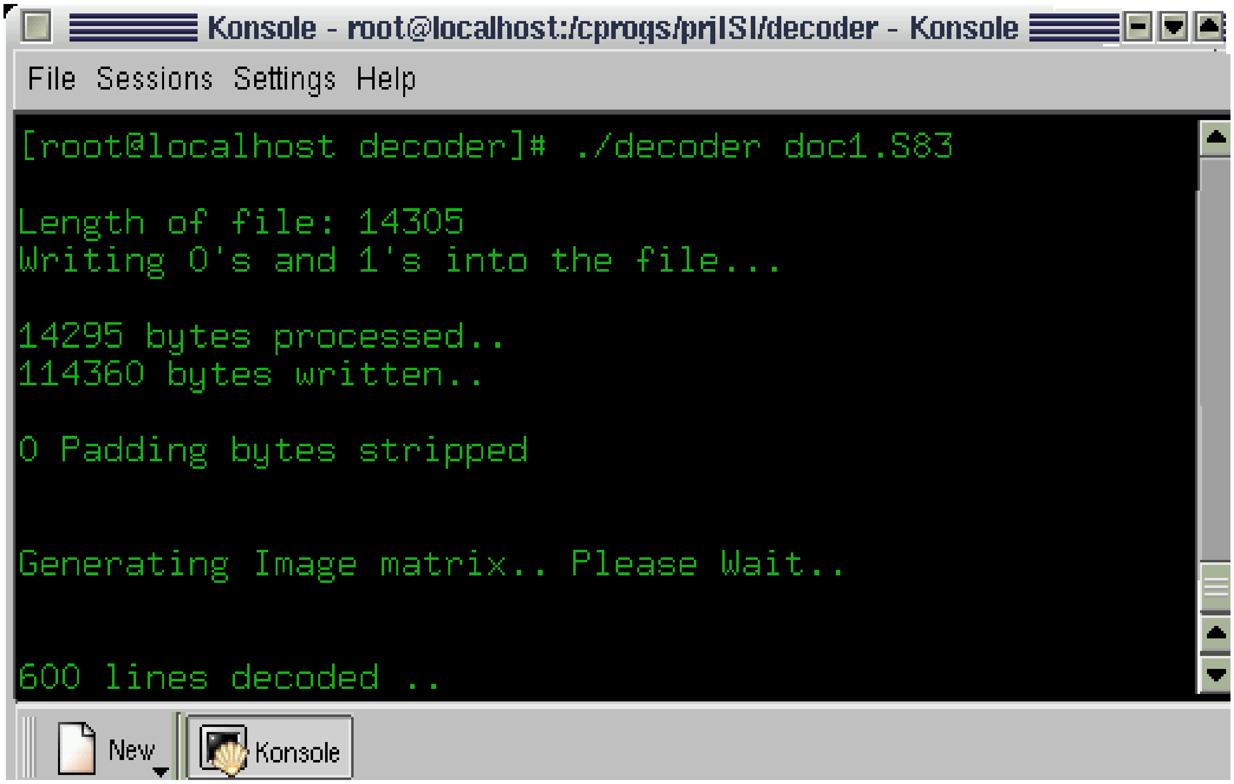
```
>- when do you extend an interface.
This is a design question. U may extend interface anytime you do not want to add methods an
xisting interface already in use (implemented) by other classes. So u don't have a choice but to
xtend and use it...
Even otherwise to represent the design correctly u may extend a interface. Shape is a interface,
ircle and Square can be interfaces extending from Shape.

f this was my interview, I'd say that was fairly close.  What I was looking for was a simple:
Only when creating an interface as a subtype, as you would 'implement' an interface to create a
ubclass."
t only goes to show that the question was not quite clear.  Thank you.

----------

 Really, i wonder if people really use UML in their projects (especially web based - server side
2ee kind of- projects). I have come across projects where there is a big noise about using UML
uring the initial stages, but as people get going with the project nobody bothers with it.

t is useful.  Although I am not permitted to discuss the different technologies used within my
ompany specifically, I have seen a number of projects use them.  Almost all of the medium
ontracts (1 - 4 Mil range) require alot of paperwork that will be read and referenced by the
lient.

he only one standard that I have seen more often than UML were database ERDs for the past 4
ears.

ut you are missing the point of the question.  There are two things being mesured:

.  If you take any patterns book or J2EE book or java programming concepts books (like a JDBC
ook for instance), they will use class diagrams wich are based on UML.  SUN's own documentation
akes use of sequence diagrams as well.  During an interview, I would need to find out things
ike if this guy doesn't know how to implement a command pattern, can he look it up and
nderstand the answer.

.  The objects I chose bare no specific purpuse in a given task that is obvious during an
nterview.  I would not ask this question if the interviewee looked stressed, because this can be
 stressful question.  This part mesures the candidate's inginuity, adaptability, and
nderstanding of relationships within the programming world.
-------------------------------------------------------------------
dapter classes are those which when extended avoid the need of implementing all the methods of
```

Related Screenshots

a) The Encoder



b) Decoder



Scope of improvement

Apart from the constraints that are being discussed earlier there are a few areas where performance of the software can be improved. Since the algorithm implemented through the software is not an optimal one, there is always a scope of increased compression by use of general data compression algorithms over it.