

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

PRESENTED BY:

Harsh Agarwal - B004

Amit Birajdar - B014

Manan Bolia - B015

Vedang Gupte - B019

Abstract

Images are among the most common and popular representations of data. Digital images are used for professional and personal use ranging from official documents to social media. Thus any Organisation or individual needs to store and share a large number of images. One of the most common issues associated with using images is the potentially large file size of image. Advancements in image acquisition technology and an increase in the popularity of digital content means that images now have very high resolutions and high quality, inevitably leading to an increase in size. Storage limitations on any device mean that only a small number of such images can be stored. Moreover, Transmitting and uploading larger images takes up more time and bandwidth. Image compression has thus emerged as a vital part of image processing. The need of the hour is to reduce the size of the image as much as possible, while maintaining a high level of quality and preserving all the details in the image. Compressed images can also be transmitted faster and require less bandwidth. In this paper we will be discussing two compression techniques- Lempel–Ziv–Welch (LZW), and Run length Encoding and how to implement them. We will then compare them based on certain parameters.

Introduction

Image compression is the process of performing operations on an image in order to reduce its file size. Any good compression algorithm should try to achieve a balance of preserving an acceptable level of detail, while reducing size as much as possible. The meaning of acceptable detail is defined by the usage and type of image. Different tasks require different amounts of details to be preserved. The best possible result is achieved by using lossless compression. This is a class of operations that allow the compressed image to be perfectly reconstructed to get back the exact original image. The image in compressed form does not look like the original image, and requires a series of operations to reconstruct it. The image in its compressed form can be transmitted much faster to the target receiver because of its small size. Then it can be

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

reconstructed at the receiver's end to get back the original image with all details intact. Two techniques that we have used for this are Lempel–Ziv–Welch (LZW), and Run length Encoding (RLE). LZW achieves this using a code table to store codes for the intensity values in the image. RLE is a very simple form of lossless data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original sequence.

Run Length Encoding Method:

Run Length Encoding is a type of lossless compression method. We have used it to reduce the size of the image matrix by removing redundancy of repeated intensity values in an image by counting the runs of repeated values in the image. Then the code matrix is created where each intensity value is followed by the frequency of its sequential occurrence. This is most useful when data contains multiple runs of repeated values. Text documents are compressed easily and efficiently using this technique.

The **algorithm** for compressing an image using RLE is as follows-

- Step 1.** Read input image
- Step 2.** If image has more than one plane (e.g. rgb), convert it to grey image
- Step 3.** Set suitable threshold values and convert image to binary
- Step 4.** Encode the image by scanning each row until end of row is reached-
 - 5.1.** For every element in image matrix, add an entry in encoded matrix at i^{th} position
 - 5.2.** Count the number of consecutive instances of element, enter this value in $(i+1)$ position
 - 5.3.** If end of row is reached, Stop
- Step 5.** Repeat for every row

When the encoded image is to be decoded-

- Step 1.** Accept the encoded image and calculate the size of the matrix
- Step 2.** Repeat for every row element by element-
 - 2.1** For every odd position i of element, store intensity as n
 - 2.2** Read next position $i+1$, which has frequency of element in i , store value as m
 - 2.3** Copy intensity value n into decoded matrix m number of times
 - 2.4** If row ends, Stop
- Step 3.** Display decoded image

The problem with this is that it will apply the same method on the images with landscape orientation as well as portrait orientation, it might work on the images with landscape orientation

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

and give an optimal output but it will not give optimal output on the images with portrait orientation. Thus we need to apply the column RLE method on the portrait image.

Now, how will we decide that which method of RLE to apply when and on which image? Thus, We have devised a smarter algorithm which will do it for us! Further referenced as ORLE Algorithm

- Step 1.** Read input image
- Step 2.** If image has more than one plane(e.g. rgb), convert it to grey image
- Step 3.** Set suitable threshold values and convert image to binary
- Step 4.** If number of rows \geq number of columns(portrait image), then take transpose of image matrix to convert it to landscape and set flag
- Step 5.** Encode the image by scanning each row until end of row is reached-
 - 5.1.** For every element in image matrix, add an entry in encoded matrix at i^{th} position
 - 5.2.** Count the number of consecutive instances of element, enter this value in $(i+1)$ position
 - 5.3.** If end of row is reached, Stop
- Step 6.** Repeat for every row

When the encoded image is to be decoded-

- Step 1.** Accept the encoded image and calculate the size of the matrix
- Step 2.** Repeat for every row element by element-
 - 2.1** For every odd position i of element , store intensity as n
 - 2.2** Read next position $i+1$, which has frequency of element in i , store value as m
 - 2.3** Copy intensity value n into decoded matrix m number of times
 - 2.4** If row ends, Stop
- Step 3.** If flag is not set, display decoded image. Else Transpose decoded image then display

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPERL-ZIV-WELCH METHOD

* Original dimension of images may be different. Below images are for just to display the images irrespective of their original dimensions

Input Images * :

Image no	Image
1	
2	

PRESENTED BY : Harsh Agarwal - B004 Amit Birajdar - B014 Manan Bolia - B015 Vedang Gupte - B019

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPERL-ZIV-WELCH METHOD

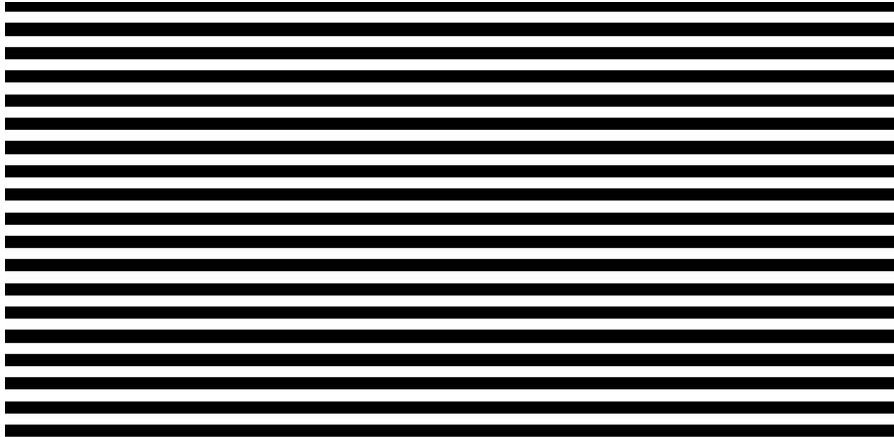
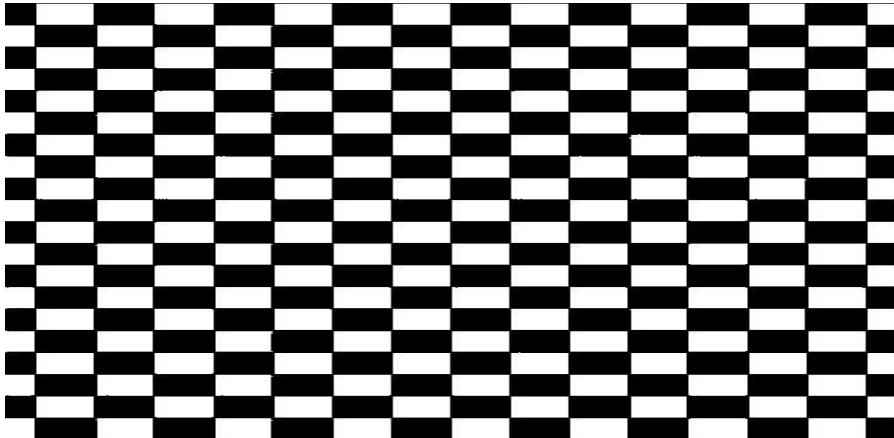
3	
4	

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPER-ZIV-WELCH METHOD

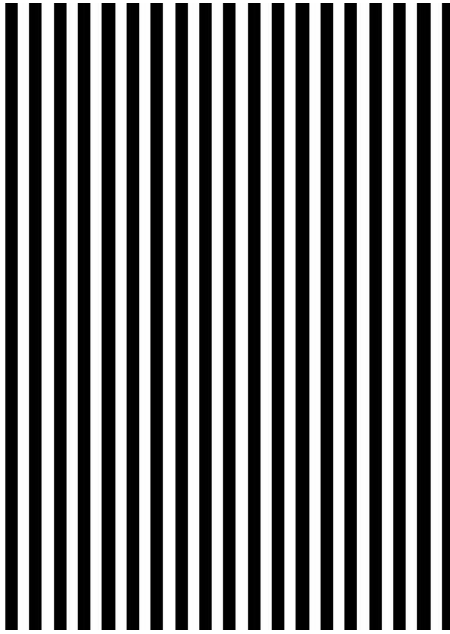
5	<p style="text-align: center;">2018 STEAM INNOVATORS CAMP Email: dr.sundarmathcenter@gmail.com</p> <p style="text-align: center;">I. Registration Form</p> <p>Site Selected: <input type="checkbox"/> MODESTO (1:30 pm to 4:30 pm) <input type="checkbox"/> TURLOCK (9:30 am to 12:30 pm)</p> <p>Circle the Camps Selected:</p> <table> <tr> <td>Week 1: June 19th - 22nd</td> <td>\$30.00</td> </tr> <tr> <td>Week 2: July 16th - 19th</td> <td>FREE</td> </tr> <tr> <td>Week 3: July 17th - 20th</td> <td>\$30.00</td> </tr> <tr> <td>Week 4: July 24th - 27th</td> <td>\$30.00</td> </tr> </table> <p>Student Name: _____ Gender: ___F___M Parent/Guardian Name: _____ Mailing Address: _____ Home Phone: _____ Emergency or Cell Phone: _____ Email Address: _____ Student's Age: _____ Grade Level as of Fall 2018: _____ School Name: _____ City: _____</p> <p style="text-align: center;">II. Consent to Participate in the 2018 STEAM Innovators Camp</p> <p>➤ (Student's Name) _____ has my consent to participate in the 2018 STEAM Innovators Camp offered through Sundar Academy.com. Any tape, photos, and comments off from participants while engaged in the program may be used only for the publicity, education, and other training purposes benefiting the program.</p> <p>➤ I understand that my child must abide by the rules in order to participate in the program.</p> <p>➤ Parent/Guardian Signature: _____ Date: _____ <small>(Required)</small></p> <p>➤ As a participant of the 2018 STEAM Innovators Camp I will abide by all the rules.</p> <p>➤ Student Signature: _____ Date: _____ <small>(Required)</small></p> <ul style="list-style-type: none"> • Please bring the completed Application and the check/money order to the Hobby Lobby on the first day of the session. • Make Checks Payable to: "Dr. Sundar Math Center Inc." • Send an email to Dr.sundarmathcenter@gmail.com confirming your child attendance camp # and site 	Week 1: June 19 th - 22 nd	\$30.00	Week 2: July 16 th - 19 th	FREE	Week 3: July 17 th - 20 th	\$30.00	Week 4: July 24 th - 27 th	\$30.00
Week 1: June 19 th - 22 nd	\$30.00								
Week 2: July 16 th - 19 th	FREE								
Week 3: July 17 th - 20 th	\$30.00								
Week 4: July 24 th - 27 th	\$30.00								
6									

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPERL-ZIV-WELCH METHOD

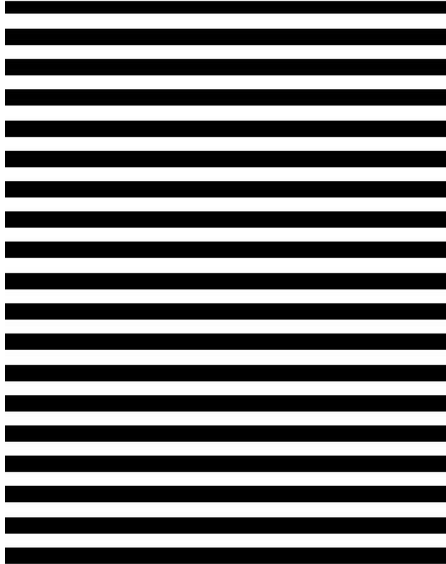
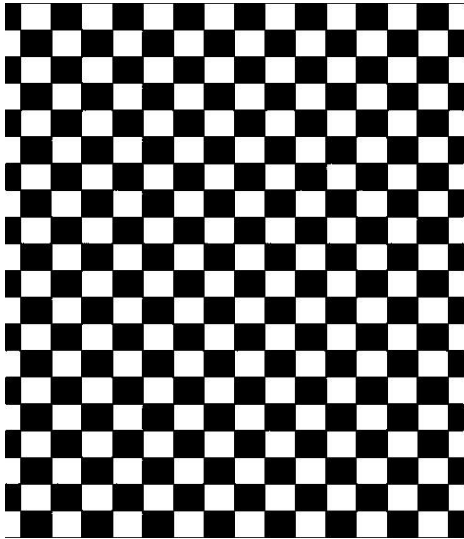
7	
8	

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPERL-ZIV-WELCH METHOD

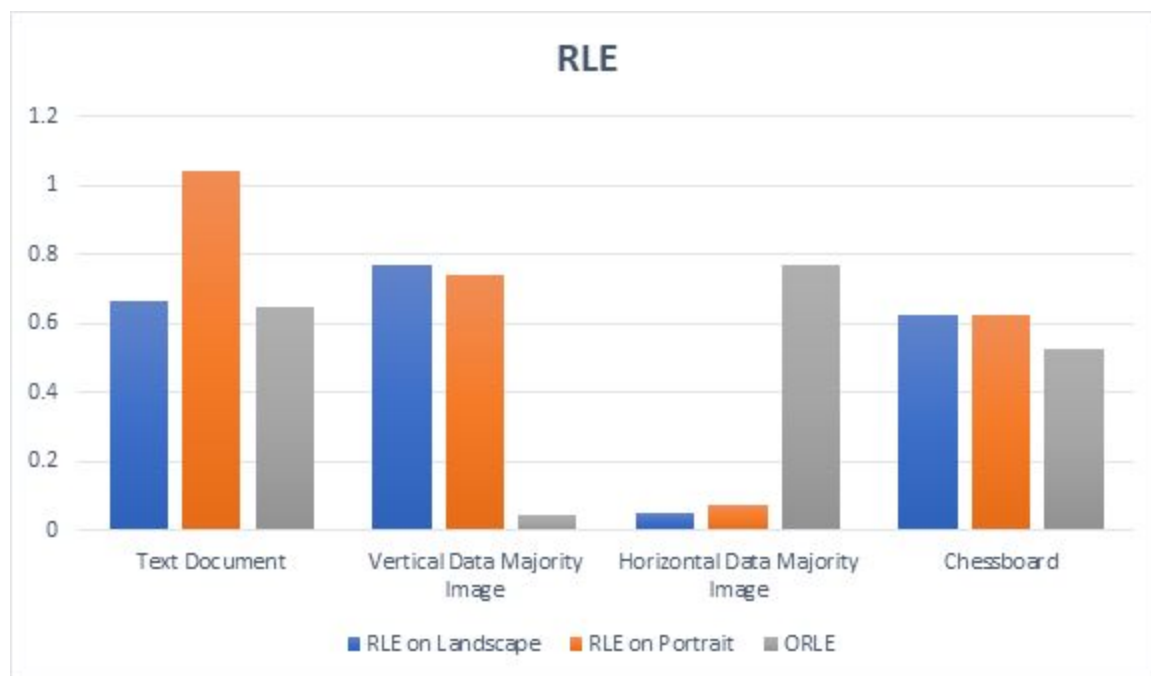
Observations:

Image Orientation	Method used	Image	Image No	Input image size(kb)	Encoded image size(kb)	Compression Ratio
LANDSCAPE	Normal RLE	Text Document	1	27	18	0.667
		Data with majority vertical lines	2	26	20	0.769
		Data with majority horizontal lines	3	39	2	0.051
		Chessboard	4	40	25	0.625
PORTRAIT	Normal RLE	Text Document	5	151	157	1.0397
		Data with majority vertical lines	6	42	31	0.738
		Data with majority horizontal lines	7	26	2	0.0769
		Chessboard	8	40	25	0.625
PORTRAIT	Optimised RLE	Text Document	5	151	98	0.649

PRESENTED BY : Harsh Agarwal - B004 Amit Birajdar - B014 Manan Bolia - B015 Vedang Gupte - B019

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPLE-ZIV-WELCH METHOD

	Data with majority vertical lines	6	42	2	0.047
	Data with majority horizontal lines	7	26	20	0.769
	Chessboard	8	40	21	0.525



Graph and table Observation:

1. From the above chart, we can see that for portrait text document, the encoded file is greater than the input file and so it is not useful but if we use the ORLE algorithm then the compression ratio decreases, inferring that image has been compressed that is encoded file size is less than input image size.
2. For portrait vertical image, use of ORLE compared to RLE gives a drastic drop in compression ratio almost to 6.66% of the encoded image using RLE

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

3. For portrait horizontal image , use of RLE is better than ORLE as RLE output is 10% of the ORLE encoded image.
4. For chessboard like pattern we can infer that use of ORLE and RLE algorithm on the image of this type does not provide much difference in the size of the encoded images generated with the respective algorithms.

Lempel-Ziv-Welch Method:

LZW compression works by reading a sequence of intensity values, grouping the values into strings, and converting the strings into codes. Because the codes take up less space than the strings they replace, we get compression. Characteristic features of LZW include,

- LZW compression uses a code table, with 4096 as a common choice for the number of table entries. This can be implemented using a dictionary. Codes 0-255 in the code table are always assigned to represent single bytes from the input file.
- When encoding begins the code table contains only the first 256 entries, with the remainder of the table being blanks. Compression is achieved by using codes 256 through 4095 to represent sequences of bytes.
- As the encoding continues, LZW identifies repeated sequences in the data, and adds them to the code table.
- Decoding is achieved by taking each code from the compressed file and translating it through the code table to find what character or characters it represents.

LZW can be used in a wide variety of file formats, but the most common ones are-

- PDF(Portable document format)
- TIFF(Tagged image file format)
- GIF(Graphic interchange format).

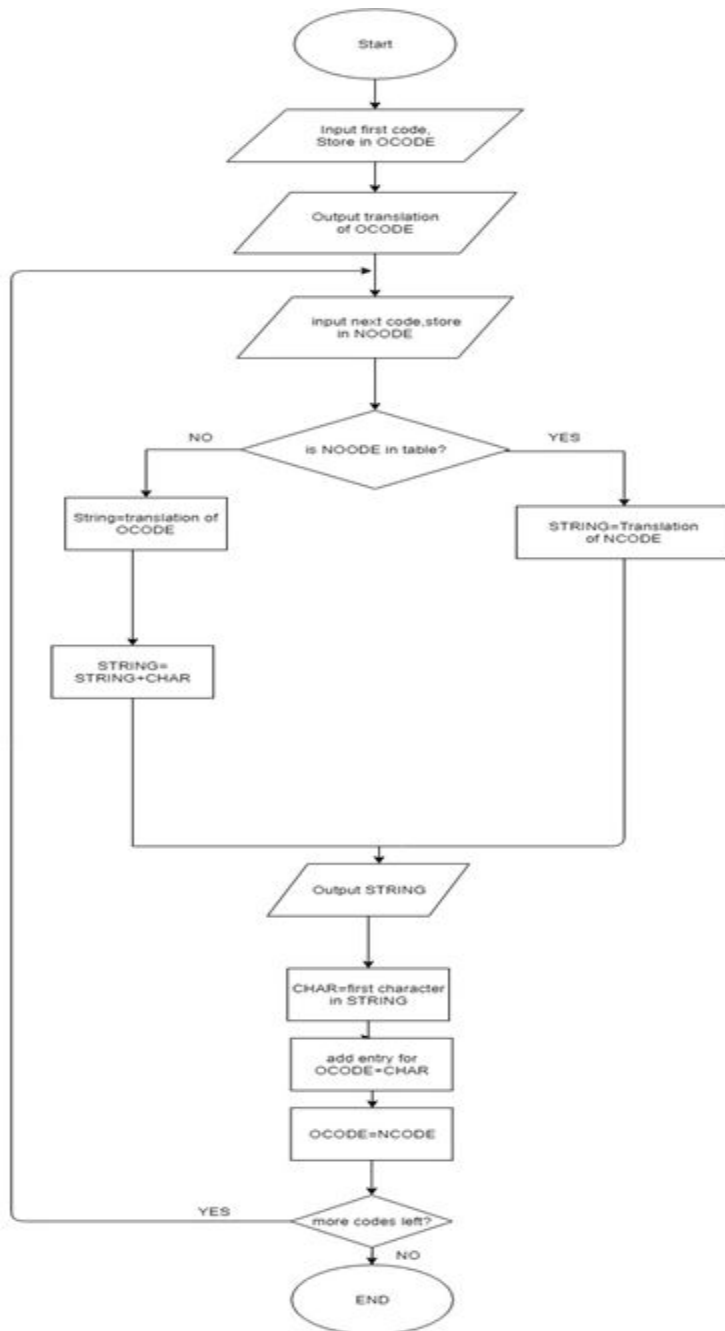
A simple flowchart can be used to show how it works. CHAR represents a single byte and STRING represents a sequence of bytes

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD



Decompression works by reading the encoded stream and comparing with the code table to see if there are any matches. If a match is found, the corresponding sequence is the output. The following flowchart demonstrates the working. OCODE is old code and NCODE is new code

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD



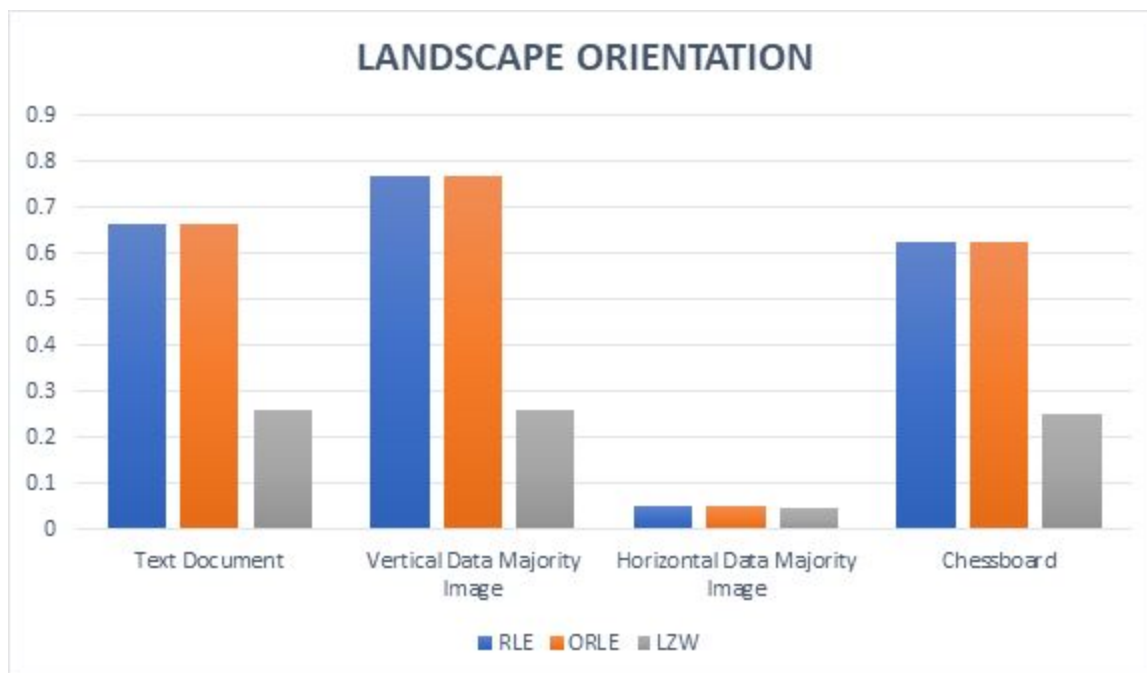
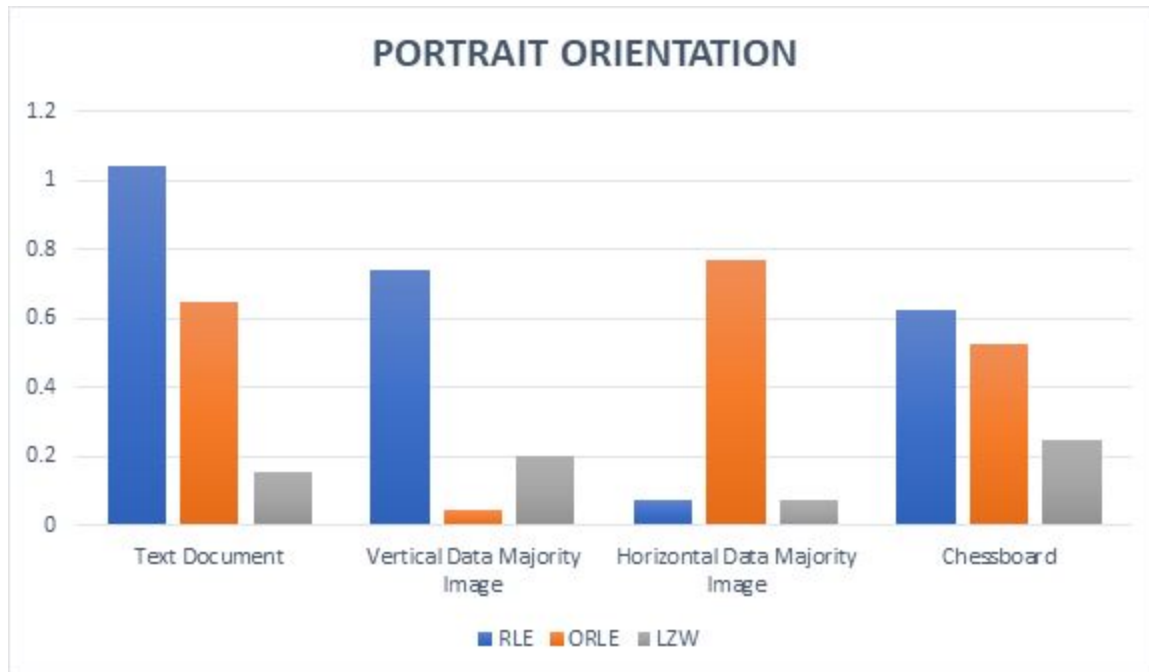
Observations:

PRESENTED BY : Harsh Agarwal - B004 Amit Birajdar - B014 Manan Bolia - B015 Vedang Gupte - B019

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPLE-ZIV-WELCH METHOD

Image Orientation	Method used	Image	Image	Input image size(kb)	Encoded image size(kb)	Compression Ratio
LANDSCAPE	LZW	Text Document	1	27	7	0.259
		Data with majority vertical lines	2	27	7	0.259
		Data with majority horizontal lines	3	41	2	0.048
		Chessboard	4	44	11	0.25
PORTRAIT	LZW	Text Document	5	147	23	0.156
		Data with majority vertical lines	6	40	8	0.2
		Data with majority horizontal lines	7	27	2	0.074
		Chessboard	8	44	11	0.25

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD



Graph and table Observations:

PRESENTED BY : Harsh Agarwal - B004 Amit Birajdar - B014 Manan Bolia - B015 Vedang Gupte - B019

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

We observe that if the image is not an ideal image like the landscape and portrait pattern or if the image is of the chessboard pattern, the best compression is obtained by LZW method. The optimised RLE method will give better output for with portrait images which are not ideal such as a text document and will give the same result as RLE for a landscape image.

What we also notice is that we are getting the optimal out for the Horizontal majority data lines in landscape orientation for RLE as well as ORLE and optimal output for portrait orientation with vertical majority data lines, but we are getting very high compression ratio for vertical majority data lines in landscape orientation and for horizontal majority data lines in portrait orientation which is not optimal.

Drawback: So we can say that this the drawback of the Optimised RLE algorithm as it was not able to give optimal compression ratio to compress those two cases. It could have given us better results if it applied column RLE on the landscape image with vertical majority data lines and row RLE on portrait image with horizontal majority data lines.

Future scope:

Using number of sign changes to determine the algorithm to apply:

As we know that the optimized RLE algorithm will automatically select the method to use for the encoding of the image by comparing the length of rows to the length of columns and then apply RLE, it basically points out that applying the column RLE to a portrait image of the file(form, receipt, etc) and applying the row RLE to a landscape image gives us the best results but then how do we decide which algorithm to use if the image is a square image with equal number of rows and columns?

Here, we can apply a new technique which will compliment the RLE encoding technique by deciding which method to apply on the image in case the image is a square image.

If we look at an image matrix then we can find that after converting the image to binary it only has two numbers, i.e. 0 and 1, which tell us that the pixel is a dark one or a bright one, we can take the 1 as a positive sign and the 0 as a negative sign and then, we count the number of sign changes or the number of times a 0 has a 1 as the next element in the column or vice versa while traversing a row. Once we get the number of sign changes for a row, we calculate that for every row and once we get the number of sign changes for every row, then we add them and store the result, now we do that same for every column, we count the number of sign changes while traversing a column from top to bottom and then add the number of sign changes for all the columns and store them.

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

Now, we compare the number of sign changes in rows to the number of sign changes in columns and if the number of sign changes in column is more than the number of sign changes in columns, then we go with **row RLE** otherwise we go with **column RLE**.

Hence, we are able to decide which method or approach to apply on a square image for getting the encoded image by RLE method.

Applying sign change approach to images which are almost square shaped:

We know that the images can be of 3 types- portrait, landscape and square shaped, we know what algorithm will be best suitable for proper portrait image(700x300) or proper landscape image(300x700) and we know what approach to follow if we have a square image, but what about the images which are neither properly portrait nor perfectly square or neither properly landscape nor perfectly square?(eg: [700x699],[300x302],[500,457]) As we have discussed above about the sign change algorithm, we count the number of sign changes in individual rows and columns and compare to decide the approach for square images, we can apply that technique on these images also as this method uses the data present inside the images to decide which method to apply, so if the the images are not properly portrait or properly landscape we can use the data contained in the image and calculate the sign changes in rows and columns and decide the approach just like we have done in square images.

Automatic thresholding usage:

We know that the RLE method used above only takes in a binary image as input and we have compared the binary image formed by the inbuilt binarize function with the binary image formed by thresholding on the threshold value set as 200 by hit and try method which still might not be a optimal threshold value, so we can use the automatic threshold finding technique which uses the valley points in histogram to determine the threshold value which will be optimal for the image and give better results when convert the image to binary with that threshold value.

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

Modification in RLE, SVC(Selective Value Count) method:

This is a technique which is a modification of the Row Length Encoding method to give better compression. Now, the problem with RLE is that it produces two elements to store the information of successive same bits, be it one bit or be it hundred, so if the image has alternate zeros and ones as values in every row and every column then our RLE technique will not be able to give compression and instead it will give us an image with twice the size of the original image.

So here what we can do to prevent the problem is to not generate dual entries or frequency entry for the elements who have a different element as the next element and only generate frequency entry for the elements who have the next element same as them. Eg:-

Optimised RLE: 1 1 1 1 0 1 → 1 4 0 1 1 1

SVC Method: 1 1 1 1 0 1 → 1 4 0 1

Here we have 4 consecutive 1's hence, we store 1 and frequency of 1 as 4 in the next bit. Next we have a single appearance of 0 and hence we do not store its frequency and similarly for the next 1.

Basic concept behind this being, store frequency for only those bits that consecutively appear more than once. Frequency of the bits appearing only once need not be stored.

While decoding, the decoder checks for each bit of the code. It will consider the first element as the intensity value. Since the image is binary, therefore any data greater than 1 will be considered as frequency. Then, it will check if the next element is greater than 1. If yes, then the current element will be repeated n number of times (n = value in the next bit which is greater than 1). Else if the value is either 1 or 0 it will just be copied to the decoded code as it is.

Conclusion

This paper studies two different image compression algorithms-LZW and RLE. Implementations of both were discussed. We took a set of input images with different orientation and data ,and tested the amount of compression achieved by each algorithm on each image. The results from both methods were then compared based on compression ratio for every type of image. The Optimized RLE was noted to be significantly better than RLE when portrait images were used. LZW has better compression than RLE but the time taken to execute is significantly more. We have also noted how we can improve these algorithms in the future to potentially achieve higher compression ratio, but it needs to be tested for confirmation.

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPERL-ZIV-WELCH METHOD

CODE FILE :

LZW

LZW_test :

```
clear all;
close all;
clc;
Input = imread('1.jpg');

Binary = imbinarize(Input(:,:,1));
[row col] = size(Binary);

Encoded = logical(LZW_img_enc(Binary));

Ratio = numel(Encoded)/numel(Binary);
Ratio

Decoded =
=
vec2mat(LZW_img_dec(Encoded),numel(Binary(1,:)));

imshow(Binary)
figure
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

```
imshow(Decoded)
rmse = 0 ;
for i =1: row
for j= 1:col
    rmse = rmse + (Binary(i,j)-Decoded(i,j)).^2;
end

end
rmse
imwrite(Binary,'C:\Users\HARSH\Desktop\notes\ip\project\
LZW_image_codec-master\LZW_image_codec-master\output\bin
ary1.jpeg')
imwrite(Decoded,'C:\Users\HARSH\Desktop\notes\ip\project
\LZW_image_codec-master\LZW_image_codec-master\output\de
coded1.jpeg')
imwrite(Encoded,'C:\Users\HARSH\Desktop\notes\ip\project
\LZW_image_codec-master\LZW_image_codec-master\output\en
coded1.gif')
```

LZW_img_enc :

```
function [enc_img,dictionary] = LZW_img_enc(img)
dictionary{1} = 0;
dictionary{2} = 1;
dictionary = dictionary';

n_img = numel(img);
enc_bits = numel(de2bi(numel(dictionary) - 1));

n = 1;
enc_img = [];

while ( n+1 <= n_img )
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

```
current_element = img(n);
next_element = img(n+1);

[r,code] =
check_dict(current_element,next_element,dictionary);

if r

temp_current_element = current_element;
temp_next_element = next_element;

while r

    current_element = temp_current_element;
    next_element = temp_next_element;

    temp_current_element = [temp_current_element ;
temp_next_element];

    if n+1 < n_img
        n = n + 1;
    else
        [r,code] =
check_dict(temp_current_element,[],dictionary);

        enc_img = [ enc_img ;
logical(de2bi(code,enc_bits,'left-msb')) ];
    return;
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

```
end

temp_next_element = img(n+1);

[r,code] =
check_dict(temp_current_element,temp_next_element,diction
nary);

end

enc_img = [ enc_img ;
logical(de2bi(code,enc_bits,'left-msb')) ];

new_element = [ temp_current_element ;
temp_next_element ];
dictionary = update_dict(new_element,dictionary);
enc_bits = numel(de2bi(numel(dictionary) - 1));

if n+1 < n_img
    n = n + 1;
else
    [r,code] =
check_dict(temp_next_element,[],dictionary);

    enc_img = [ enc_img ;
logical(de2bi(code,enc_bits,'left-msb')) ];
    return;
end
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPERL-ZIV-WELCH METHOD

```
else

    enc_img          =          [          enc_img          ;
logical(de2bi(code,enc_bits,'left-msb')) ];

    new_element = [ current_element ; next_element ];
    dictionary = update_dict(new_element,dictionary);
    enc_bits = numel(de2bi(numel(dictionary) - 1));

    if n+1 < n_img
        n = n + 1;
    else
        [r,code]          =
check_dict(next_element,[],dictionary);
        enc_img          =          [          enc_img          ;
logical(de2bi(code,enc_bits,'left-msb')) ];
        return;
    end

end

end

end
```

LZW_img_dec :

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPERL-ZIV-WELCH METHOD

```
function [dec_img,dictionary] = LZW_img_dec(enc_img)

dictionary{1} = 0;
dictionary{2} = 1;
dictionary = dictionary';

n_enc_img = numel(enc_img);
enc_bits = 2;

n = 2;
dec_img = [];

current_code = bi2de(enc_img(1));
next_code = bi2de(enc_img(2:3)', 'left-msb');

while( n+enc_bits-1 <= n_enc_img )

    current_element = dictionary{current_code+1};

    if next_code+1 <= numel(dictionary)
        next_element = dictionary{next_code+1};

        new_element = [ current_element ; next_element(1)
];
        [dictionary,new_code] =
update_dict(new_element,dictionary);
    else
        next_element = [ current_element ;
current_element(1) ];
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

```
[dictionary,new_code] =  
update_dict(next_element,dictionary);  
end  
  
dec_img = [ dec_img ; current_element ];  
  
if n+enc_bits-1 < n_enc_img  
n = n + enc_bits;  
else  
dec_img = [ dec_img ; next_element ];  
return;  
end  
  
enc_bits = numel(de2bi(new_code+1));  
  
current_code = next_code;  
next_code =  
bi2de(enc_img(n:n+enc_bits-1)', 'left-msb');  
end  
  
end
```

check_dict :

```
function [r,code] = check_dict(current,next,dictionary)  
n_dict = numel(dictionary);  
concat = [current ; next];  
r = 0;  
code = -1;
```


IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

```
for i=1:n_dict

    e = dictionary{i};

    if numel(concat) == numel(e)
        if concat == e
            code = i-1;
            r = 1;
            return;
        end
    elseif numel(current) == numel(e)
        if current == e
            code = i-1;
        end
    end
end

end
```

update_dict :

```
function [d_new,n] = update_dict(element,dictionary)

n = numel(dictionary);
d_new = [dictionary ; element];
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPERL-ZIV-WELCH METHOD

end

RLE

Test :

```
clc;
clear all;
Input = imread('pchess.jpg');
[row col dep ] = size(Input);
if(dep>1)
Input = rgb2gray(Input);
end

%Binary = imbinarize(Input(:,:,1));

for i = 1:row
    for j = 1:col
        if(Input(i,j)>200)
            Binary(i,j) = 1;
        else
            Binary(i,j) = 0;
        end
    end
end
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPLE-ZIV-WELCH METHOD

```
imshow(Binary);

for i = 1:row
    column = Binary(i,:);
    Encoded = rle(column(:));
    [encl encc] = size(Encoded);
    eco(i,1:encc) = Encoded(1,:);
    lencc(i,1) = encc;

end

for i = 1:row
    column = eco(i,1:lencc(i,1));
    Decoded(i,:) = irle(column(:));

end

figure
imshow(Decoded)
csvwrite('C:\Users\HARSH\Desktop\notes\ip\project\RLE___IRLE_coding\potrait\rle\encodedlenp chess.csv',lencc)
imwrite(Binary,'C:\Users\HARSH\Desktop\notes\ip\project\RLE___IRLE_coding\potrait\rle\inputp chess.jpeg')
imwrite(Decoded,'C:\Users\HARSH\Desktop\notes\ip\project\RLE___IRLE_coding\potrait\rle\decodedp chess.jpeg')
imwrite(uint16(eco),'C:\Users\HARSH\Desktop\notes\ip\project\RLE___IRLE_coding\potrait\rle\encodedp chess.jpeg','BitDepth',12);
Ratio = numel(eco)/numel(Binary);
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

```
Ratio
rmse = 0 ;
for i =1: row
for j= 1:col
    rmse = rmse + (Binary(i,j)-Decoded(i,j)).^2;
end

end

rmse
```

rle :

```
function Output=rle(Input)
L=length(Input);
j=1;
k=1;
i=1;
while i<2*L
    comp=1;
    for j=j:L
        if j==L
            break
        end;
        if Input(j)==Input(j+1)
            comp=comp+1;
        else
            break
        end;
    end;
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

```
Output(k+1)=comp;
Output(k)=Input(j);
if j==L && Input(j-1)==Input(j)
    break
end;
i=i+1;
k=k+2;
j=j+1;
if j==L
    if mod(L,2)==0
        Output(k+1)=1;
        Output(k)=Input(j);
    else
        Output(k+1)=1;
        Output(k)=Input(j);
    end;
    break
end;
end;
```

irle :

```
function Output=irle(Input)
L=length(Input);
s=1;
k=1;
i=1;
while i<=L
    while s<=Input(i+1)
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPEL-ZIV-WELCH METHOD

```
Output(k)=Input(i);  
s=s+1;  
k=k+1;  
end;  
i=i+2;  
s=1;  
end;
```

Optimized RLE :

Test :

```
clc;  
clear all;  
Input = imread('1.jpg');  
[row col dep] = size(Input);  
if(dep>1)  
Input = rgb2gray(Input);  
end  
  
%Binary = imbinarize(Input(:,:,1));  
  
for i = 1:row  
    for j = 1:col  
        if(Input(i,j)>200)  
            Binary(i,j) = 1;
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPERL-ZIV-WELCH METHOD

```
        else
            Binary(i,j) = 0;
        end
    end
end

imshow(Binary);
% encoding
c = 0;
if(row>=col)
    Binary = Binary';
    [row col ] = size(Binary);
    c = 1;
end
for i = 1:row
    column = Binary(i,:);
    Encoded = rle(column(:));
    [encr encc] = size(Encoded);
    eco(i,1:encc) = Encoded(1,:);
    lencc(i,1) = encc;

end

for i = 1:row
    column = eco(i,1:lencc(i,1));
    Decoded(i,:) = irle(column(:));

end

if( c==1)
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPLE-ZIV-WELCH METHOD

```
Binary = Binary';
Decoded = Decoded';
[row col ] = size(Binary);

end
figure
imshow(Decoded)
csvwrite('C:\Users\HARSH\Desktop\notes\ip\project\RLE___
IRLE_coding\potrait\rle2\encodedlenp chess.csv',lenc)
imwrite(Binary,'C:\Users\HARSH\Desktop\notes\ip\project\
RLE___IRLE_coding\potrait\rle2\inputp chess.jpeg')
imwrite(Decoded,'C:\Users\HARSH\Desktop\notes\ip\project
\RLE___IRLE_coding\potrait\rle2\decodedp chess.jpeg')
imwrite(uint16(echo),'C:\Users\HARSH\Desktop\notes\ip\pro
ject\RLE___IRLE_coding\potrait\rle2\encodedp chess.jpeg',
'BitDepth',12);
Ratio = numel(echo)/numel(Binary);

Ratio
rmse = 0 ;
for i =1: row
for j= 1:col
    rmse = rmse + (Binary(i,j)-Decoded(i,j)).^2;
end
end

rmse
```

rle :

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPERL-ZIV-WELCH METHOD

```
function Output=rle(Input)
L=length(Input);
j=1;
k=1;
i=1;
while i<2*L
    comp=1;
    for j=j:L
        if j==L
            break
        end;
        if Input(j)==Input(j+1)
            comp=comp+1;
        else
            break
        end;
    end;
    Output(k+1)=comp;
    Output(k)=Input(j);
    if j==L && Input(j-1)==Input(j)
        break
    end;
    i=i+1;
    k=k+2;
    j=j+1;
    if j==L
        if mod(L,2)==0
            Output(k+1)=1;
            Output(k)=Input(j);
        else
            Output(k+1)=1;
            Output(k)=Input(j);
        end;
        break
    end;
end;
```

IMAGE COMPRESSION USING RUN LENGTH ENCODING METHOD AND LEMPERL-ZIV-WELCH METHOD

```
end;  
end;
```

irle :

```
function Output=irle(Input)  
L=length(Input);  
s=1;  
k=1;  
i=1;  
while i<=L  
    while s<=Input(i+1)  
        Output(k)=Input(i);  
        s=s+1;  
        k=k+1;  
    end;  
    i=i+2;  
    s=1;  
end;
```