

Okay, here are concise answers to the questions about Run-Length Encoding (RLE) in English:

1. What is RLE?

- A simple, lossless data compression technique that replaces consecutive identical data values (runs) with a count and the single value.

2. What are the benefits of using RLE?

- Simple to implement, fast, lossless, and effective at compressing data with many repeating sequences.

3. What are the drawbacks of using RLE?

- Inefficient (can even increase file size) for data without long runs of identical values. Less effective than more complex algorithms.

4. In what situations is RLE useful?

- Best for data containing long sequences of repeated values, like simple graphic images (icons, bitmaps), faxes, or certain types of indexed images.
-

Huffman Code Summary:

Advantages:

- **Variable-length:** Uses shorter codes for more frequent characters.
- **Optimal:** Achieves the best possible compression for this type of coding.

Disadvantages:

- **Computationally intensive:** Can require significant processing power.
 - **Requires tree storage:** The decoding tree must be stored, adding to data size.
 - **Inefficient for diverse/non-repetitive data:** Performs poorly when character frequencies are similar or there's little repetition.
-

Summary of RLE Drawbacks and Use Cases:

- **Drawback:** RLE is inefficient and can even increase file size for data that doesn't have many repeating sequences.
- **Useful For:** Compressing simple images (like icons or line drawings) and in some data storage situations.

Example

Use SRS TO code the string "8940000000000000000000000000000000"

Solution

We can replace with: 894f32 /where f is the flag for zero.

Example

Replace all occurrences of pattern of characters `and` with the predefined code `&`.

Solution

and you and I **becomes:** & you & I

Example

Use RLE TO code the string "AAABBCDDD"

Solution

"AAABBCDDD" would be encoded as "A3B2C1D3".

3. What are the drawbacks of using Run-Length Encoding?

Solution

Inefficient for data without many repeating sequences. Can actually increase the data size in some cases.

4. In what situations is Run-Length Encoding useful?

Solution:

RLE is commonly used in image compression (especially for simple images like icons or line drawings) and in certain data storage scenarios..

Lossless Compression Algorithms

- 1) Run-length Encoding (RLE)
- 2) Simple Repetition Suppression
- 3) Pattern Substitution
- 4) Shanon-Fano Method
- 5) Huffman Code

Example

If you know that original image is **256×256** pixel, single band (**gray scale**), **8-bit per pixel**. After compression the image file is **6,554 byte**.

Calculate the compression ratio

Solution

Size U = Size of image = $256 * 256 * 1 = 65,536$ bytes

$$\text{Compression ratio} = \frac{\text{Size U}}{\text{Size C}} = \frac{65536}{6554} = 9.999 = 10$$

This is called a “10 to 1” compression

Note: repeat the previous question for image with three bands.

Example :

Suppose you have the following sub of image. Perform the following operation

1. Stretch histogram.
2. Shrink (compress) histogram
3. Slide the histogram

7	12	8	7
11	9	6	4
10	5	1	5

Where: Image with 6 bit for each pixel

$\text{Shrink}_{\max} = 30$, $\text{Shrink}_{\min} = 10$

OFFSET values = 50

Solution :

1. Stretch histogram

$I(r,c)_{\max} = 12$, $I(r,c)_{\min} = 1$, $\text{Max} = 63$, $\text{Min} = 0$

$$\text{Stretch } (I(r, c)) = \left[\frac{I(r, c) - I(r, c)_{\min}}{I(r, c)_{\max} - I(r, c)_{\min}} \right] [\text{MAX-MIN}] + \text{MIN}.$$

$$I(r, c) = ((7-1)/(12-1)) * (63-0)+0 \\ = (6/11)*63 = \mathbf{34.36}$$

34.36			

12

2. Shrink histogram

$I(r,c)_{\max} = 12$, $I(r,c)_{\min} = 1$,
 $\text{Shrink}_{\max} = 30$, $\text{Shrink}_{\min} = 10$

7	12	8	7
11	9	6	4
10	5	1	5

$$\text{knirhS } (r(I, c)) = \left[\frac{\text{Shrink}_{\max} - \text{Shrink}_{\min}}{I(r, c)_{\max} - I(r, c)_{\min}} \right] [I(r, c) - I(r, c)_{\min}] \text{knirhS} + \text{min}$$

$$I(r, c) = ((30 - 10) / (12-1)) * (7 - 1) + 10 \\ 10 + 6*(20/11) = \mathbf{20.86}$$

20.86			

3. Slide the histogram

OFFSET values = 50

$$\text{Slide } (I(r, c)) = [I(r, c)] + \text{OFFSET}$$

$$I(r, c) = 7 + 50 \\ = \mathbf{57}$$

57	62	58	

Example

Let's assume we have a simple grayscale image with the following pixel values and their corresponding frequencies:

Use the Shannon-Fano method To compress the image

Pixel Value	Frequency
0	50
1	100
2	12
3	25
4	6
5	3
6	1

solve this using the **Shannon-Fano algorithm**, we follow these steps:

Step 1: Sort the Pixel Values by Frequency (Descending)

Pixel Value Frequency

1 100

0 50

3 25

2 12

4 6

5 3

6 1

Step 2: Apply Shannon-Fano Algorithm

Goal: Assign binary codes by dividing the list into two parts where the total frequencies are as equal as possible, then assign 0 to one part and 1 to the other.

Step 2.1: First Split

Group A	Group B
---------	---------

1 (100) + 0 (50) = 150	3 (25) + ... = 47
------------------------	-------------------

Assign:

- Group A: 0
 - Group B: 1
-

Step 2.2: Split Group A [1, 0]

Group A1	Group A2
----------	----------

1 (100)	0 (50)
---------	--------

Assign:

- Pixel 1: 00
 - Pixel 0: 01
-

Step 2.3: Split Group B [3, 2, 4, 5, 6]

Group B1	Group B2
----------	----------

3 (25) + 2 (12) = 37	4 (6) + 5 (3) + 6(1) = 10
----------------------	---------------------------

Assign:

- B1: 10
 - B2: 11
-

Step 2.4: Split B1 [3, 2]

Group B1a Group B1b

3 (25) 2 (12)

Assign:

- Pixel 3: 100
- Pixel 2: 101

Step 2.5: Split B2 [4, 5, 6]

Group B2a Group B2b

4 (6) + 5 (3) 6 (1)

Assign:

- B2a: 110
- B2b: 111

Now split B2a:

- 4 (6): 1100
- 5 (3): 1101

Pixel 6: 111

Final Shannon-Fano Codes:

Pixel Value Frequency Code

1	100	00
0	50	01
3	25	100
2	12	101

Pixel Value Frequency Code

4	6	1100
5	3	1101
6	1	111

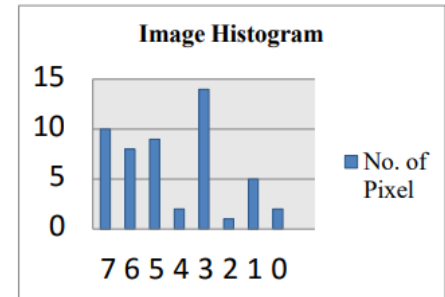
Let me know if you'd like to calculate the **average code length**, compression ratio, or visualize the Shannon-Fano tree!

Example- :

We have an image with 3 bit /pixel, so the possible range of values is 0 to . 7

We have an image with the following histogram

Gray level value	0	1	2	3	4	5	6	7
No. of Pixel histogram value	10	8	9	2	14	1	5	2



Answer:

Step :1

Great a running sum of histogram values. This means that the **first values** is 10, The **second** is **10+8=18**, next is **10+8+9=27**, and soon. Here we get

.10,18,29,43,44,49,51

Gray level value	0	1	2	3	4	5	6	7
No. of Pixel	10	8	9	2	14	1	5	2
Run sum	10	18	27	29	43	44	49	51

Step :2

Normalize by dividing by total number of pixels. The total number of pixels is **10+8+9+2+14+1+5+2=.51**

Gray level value	0	1	2	3	4	5	6	7
No. of Pixel	10	8	9	2	14	1	5	2
Run sum	10	18	27	29	43	44	49	51
Normalize	10/51	18/51	27/51	29/51	43/51	44/51	49/51	51/51

Step 3:

Multiply these values by the maximum gray level values in this case 7 , and then round the result to the closet integer. After this is done we obtain **1,2,4,4,6,6,7,7.**

Gray level value	0	1	2	3	4	5	6	7
No. of Pixel	10	8	9	2	14	1	5	2
Run sum	10	18	27	29	43	44	49	51
Normalize	10/51	18/51	27/51	29/51	43/51	44/51	49/51	51/51
Multiply by 7	1	2	4	4	6	6	7	7

Step 4:

Map the original values to the results from step 3 by a one-to-one correspondence.

Old	0	1	2	3	4	5	6	7
New	1	2	4	4	6	6	7	7