Step 1 Implement Run Length Encoding

1. see RunLengthString.java on the github repo

Step 2 Use the RunLength.java implementation that works with the binary input and output libraries provided

<u>Binary Compression</u>

1. Begin by first outputting the number of bits in the binary file ‘4runs.bin’

4runs.bin contains 40 bits

2. Now let’s try to compress this file with Run Length Encoding and see what we get (we’ll combine RunLength with BinaryDump to see how much compression we achieve)

when compressed, 4runs.bin contains 32 bits

the compression ratio is 32/40 = 0.8

<u>ASCII Compression</u>

1. Let’s run through some of the same steps but with a text file
   Use the command: java BinaryDump 8 < abra.txt How many bits do you get?

abra.txt contains 96 bits

2. Let’s see what we can get to with compression
   java RunLength - < abra.txt | java BinaryDump 8 How many bits did you get?
   What is the compression ratio?

when compressed, abra.txt contains 416 bits

the compression ratio is 416/96 = 4.33

Why do you think you got this? What is happening?

ascii character encoding does not naturally produce many runs of consecutive 0's or 1's. therefore, runlength encoding does not work vey well with ascii encoded text. since the runs are so short, it is less space-efficient to use runlength encoding that to simply leave the file uncompressed.

3. Create your own text file that does lend itself to RunLength Encoding and perform the same steps as above, reporting your compression ratio.

Using a simple C program, I created a simple ascii text file containing only null characters (numeric value 0x00). These are non-printed, and have a binary value 00000000. This makes it so that the file will be very well-suited to run-length encoding.

my sample file contained 8,000 bits

after compression it contained 5,016 bits

the compression ratio was 5,016/8,000 = 0.627

<u>Bitmap Compression</u>
Run Length encoding is widely used for bitmaps because this input data is more likely to have long runs of repeated data (i.e. pixels).

Step 1: Use BinaryDump to find out how many bits the bitmap file q32x48.bin has

q32x48.bin contains 1536 bits

Step 2: Use Run Length function to compress the bitmap file q32x48.bin

Now use the BinaryDump function to count the bits in the compress file (‘q32x48rle.bin’).

q32x48rle.bin contains 1144 bits

Step 3: Calculate the compression ratio

the compression ratio was 1144/1536 = 0.74

Step 4: Perform the Steps 1 and 2 on the higher resolution bitmap file q64x96.bin
Note down the original bits and compressed bits.
Calculate the compression ratio?

q64x96.bin contains 6144 bits

when compressed q64x96.bin contains 2296 bits

the compression ratio is 2296/6144 = 0.37

Step 4: Compare the compression ratio of the first bitmap image to this second compressed bitmap image. What do you think is the reason for this difference?

Since runlength encoding relies on locally similar bits in order to achieve a high compression ratio, images must contain large blocks of identical colours in order to be compressed efficiently. Unfortunately I can't view the bitmaps locally, however I would think that perhaps q32x48.bin has a random arrangement of pixels, whereas q64x96.bin has a more "blocky" arrangement of pixels.