Algorithms Practical 9 Compression

Binary Compression

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

Use the command: java **BinaryDump** 40 < 4runs.bin
Note down the bits.

```
0000000000000001111111000000011111111111
```

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)

Use the command: java RunLength - < 4runs.bin | java BinaryDump
Note down the bits.

```
0000111100000111
0000011100001011
```

32 Bits

Calculate the compression ratio: compressed bits / original bits

80%

3. Next we’ll output this compressed file to a new binary file and check we have the same compression ratio.

Use the command: java RunLength - < 4runs.bin > 4runsrle.bin
Use Binary Dump to check the bits: you can create this command yourself now

```
00001111000001110000011100001011
```

32 Bits

ASCII Compression

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?

```
010000010100001001010010010000010100001101000001010001000100000101000010
0
10100100100000100100001
```

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? 416 bits

What is the compression ratio? 433.33%

Why do you think you got this? What is happening?
We can see that RunLength compression is not efficient for ASCII characters as the compression ratio is very high and the number of compressed bits is bigger than the original bits meaning that using RunLength compression, it doesn't compress files that contain ASCII characters effectively and therefore no space is saved.

3. Create your own text file that does lend itself to RunLength Encoding and perform the same steps as above, reporting your compression ratio.
My text file : cat.txt
Number of Original Bits : 3813 bits
Number of compressed bits : 10342 bits
Compression Ratio : 270.87%

Bitmap Compression
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
Note down your answer: 1536 bits

Step 2: Use Run Length function to compress the bitmap file q32x48.bin
Use the command to compress and output the compressed file to a new file:
Java RunLength - < q32x48.bin > q32x48rle.bin

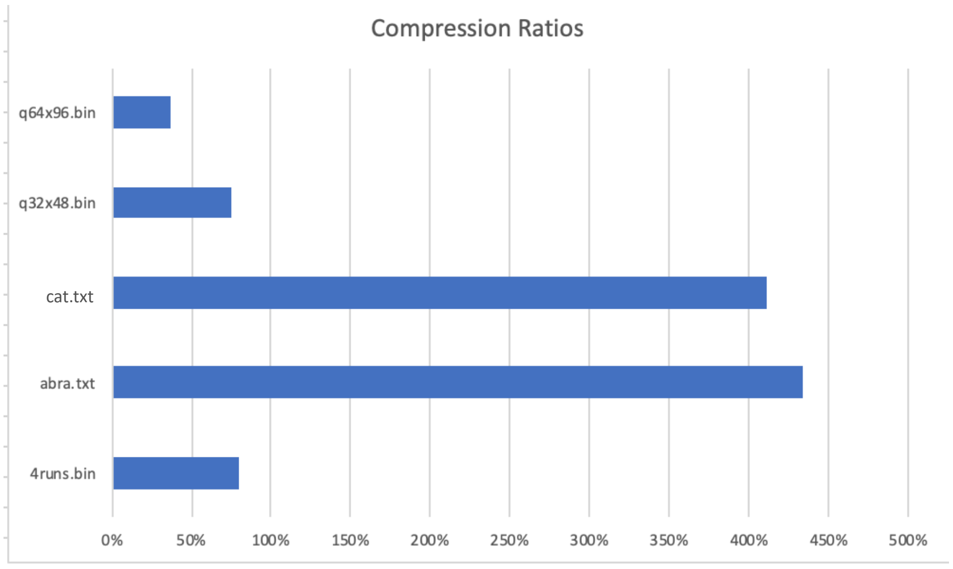
Now use the BinaryDump function to count the bits in the compress file ('q32x48rle.bin'): 1144 bits

Step 3: Calculate the compression ratio
74.48%

Step 4: Perform the Steps 1 and 2 on the higher resolution bitmap file q64x96.bin
Note down the original bits and compressed bits:
Original bits : 6144 bits
Compressed bits : 2296 bits
Calculate the compression ratio? 37.37%

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? As we can see, RunLength works far better on .bin files compared to files that contain ASCII characters. q64x96 has a better compression ration compared to q32x48.bin. the reason I think this is the case as q64x96.bin has a higher number of original bits compared to q32x48.bin, so more of that data is compressed. So the bigger the original number of bits the lower the compression ratio and therefore more spaced is saved.

q32x48.bin
q32x48rle.bin



As we can see from the graphs the two text files have the highest compression ratios.
This clearly shows that RunLength compression is ineffective on ASCII characters.
 It works better on .bin files as it gives lower compression rations.
This indicates that the files have been compressed and so space has been saved which is what we want.