ANSWERS

F8.9 Compression

Data representation · GCSE · OCR J277 1.2.5, AQA 8525 3.3.8, Edexcel 1CP2 2.3.2 · about 20 min

What this lesson is about

Lossy and lossless; run length encoding and Huffman coding.

- Lossy and lossless
- Run length encoding
- Huffman coding
- Which to use

Questions

5 marks in all

1. Which kind of compression must be used for a program file? [1 mark]

- ☐ **A** Lossless, because every character must be kept exactly
- ☐ **B** Lossy, because it is smaller
- ☐ **C** Either
- ☐ **D** Neither

> **Answer:** A. A single changed character could break the program.

2. What does lossy compression do? [1 mark]

- ☐ **A** Permanently removes detail people are unlikely to notice
- ☐ **B** Keeps every bit of the original
- ☐ **C** Makes a file larger
- ☐ **D** Encrypts the file

> **Answer:** A. The original cannot be rebuilt, but the file is much smaller.

3. Run length encode `WWWWBBW` in the form count then character. [1 mark]

> **Answer:** 4W2B1W. 4 W, 2 B, 1 W.

4. On which data does RLE work worst? [1 mark]

- ☐ **A** Data where the value changes every character
- ☐ **B** Data with long runs
- ☐ **C** Black and white images
- ☐ **D** Blank rows

> **Answer:** A. With no runs, every character needs a count too, so the result is longer.

5. In Huffman coding, which characters get the shortest codes? [1 mark]

- ☐ **A** The most frequent ones
- ☐ **B** The least frequent ones
- ☐ **C** Capital letters
- ☐ **D** The first characters in the message

> **Answer:** A. Short codes for common characters save the most bits.

The task: compress a row

Write `rle_encode(text)` and `rle_decode(code)` yourself. Encode the row `........######....######........` and print it as `encoded: <code>`, then print `original: <n> characters, encoded: <m> characters`, and finally `decoded matches: True`, using your decode function.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

row = "........######....######........"
```

The hint students can ask for: Walk the row counting how long the current run is. When the character changes, write the count and that character, then start counting again. To decode, gather the digits into a number, then repeat the character that follows it.

A solution

```
from bugbot import *
connect()
def rle_encode(text):
    out = ""
    count = 1
    for i in range(1, len(text)):
        if text[i] == text[i - 1]:
            count = count + 1
        else:
            out = out + str(count) + text[i - 1]
            count = 1
    return out + str(count) + text[-1]

def rle_decode(code):
    out = ""
    number = ""
    for ch in code:
        if ch.isdigit():
            number = number + ch
        else:
            out = out + ch * int(number)
            number = ""
    return out
row = "........######....######........"
code = rle_encode(row)
print("encoded:", code)
print("original:", len(row), "characters, encoded:", len(code), "characters")
print("decoded matches:", rle_decode(code) == row)
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.