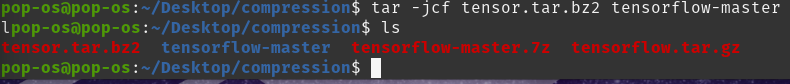
**Practice commands**

A computer screen shot of a computer program

Description automatically generatedA screenshot of a computer code

Description automatically generatedA screen shot of a computer

Description automatically generatedA screen shot of a computer program

Description automatically generated

**Optimal coding for each character**

A black text on a white background

Description automatically generated

This question lead me down a rabbit hole on the internet for how to compute them as I was getting many different answers. The one that seemed the most effective was to calculate the huffman codes in python, so that is the approach that I took to solve this problem.

This was the python code I used to calculate the codes:

A screenshot of a computer program

Description automatically generated

So the codes for each character are as follows:

‘c’ = ‘0’

‘a’ = ‘100’

‘b’ = ’110‘

‘d’ = ‘111’

‘e’ = ‘101’

**Flowchart according to the diagram for delta compression**

I wasn’t exactly sure how to do this so I generated a delta compression graph from python using graphviz.

A diagram of a server

Description automatically generated

A computer screen shot of a program code

Description automatically generated

**LZW algorithm diagram for BABAABAAA**

I was already using graphviz for the rest of the diagrams so I used it for this one too.

A diagram of a program

Description automatically generated

For LZW It is a dictionary based compression method, it builds a dictionary of input sequences, starting by initialising a dictionary, that contains all single character strings from the inputs alphabet. In this case we only deal with ‘A’ and ‘B’, it adds them to the dictionary, then reads the input characters to match the longest string in the dictionary, when a new string is encountered it adds it to the dictionary and outputs the code of the longest known string.