Calum Brown – Assignment 2

In this assignment I wrote 3 segments of code

One for original coding

One fore LZW

And one for arithmetic coding

All the codes have their own comments explaining them and I will talk about each of them aswell

Sidenotes:

text.txt is the CHARTER OF FUNDAMENTAL RIGHTS OF THE EUROPEAN UNION

And capital letters and other characters are all taken into consideration

I used python to code and I used repl online to do so aswell, if you wish to see the decoded message you can copy code I've given in txt files and you'll just to add the text.txt file and name is the same aswell and the rest will appear once run, or you just trust me it decodes fine which it does

Here is the **original** coding

I merely use this to calculate the total amount of bits need to be sent to be able to compare it with LZW and arithmetic

The code comprises reading the fundamental rights file and adding all of its characters into a dictionary and then counting how much each character occurs

Then after calculating the total I look at the minimum number bits to send a prefix free code where all codewords have the same number of bits since some shouldn't be more important than others in an original code

And then lastly I reread the file to find the total number of characters and then find the corresponding number of bits which is just 7 by the number of characters

```
import math

file = open("text.txt", "r")
#here are the two dictionaries
#the one for the sender which is updated as it reads the txt file
#and the one for the counter which updates aswell but with 0 instead of the
character
```

```
dictionary = []

dictionary_counter = []

dictionary_codes = []


x = "a" #start with arbitrary value, will get overwritten
total = 0

#this loop counts how characters are in the txt file
while(1):

  if x != "": #continues if we are not at the end of the txt file
    x = file.read(1) #reads the next character

    #this just checks for certain character not yet in the orinal alphabet
    #moreso unique icons like '/', '>' and most importanly ''' which i couldnt
add
    if (x not in dictionary):
      dictionary.append(x)
      total = total + 1
  else:


    break


#there are 75 total character in the text file
#using boring orignal coding we can have each character be 7 bits since
#2^7 envelopes 77 (log(2)(77) = 7 rounding up)
print("total character: " + str(total))
n = math.ceil((math.log(77, (2))))
print("bits per char: " + str(n))



#so since every character will be 7 bits long
#i wont bother creating a unique codeword for everyone and unstead send 7 bits
arbitrary
#since im only doing this to find the length of the code


x = "a" #start with arbitrary value, will get overwritten
total_char = 0
file.seek(0)

#this loop rereads the txt file to count the amount of character in are in it
while(1):
```

```
  if x != "": #continues if we are not at the end of the txt file
    x = file.read(1) #reads the next character
    total_char = total_char + 1 #adds to total char
  else:
    break

print("total characters sent: " + str(total_char))
print("total bits sent: " + str(total_char*n)) #this is the total bits sent
assuming all char have the same codeword length of 7 bits
```

output:

```
total character: 75
bits per char: 7
total characters sent: 24218
total bits sent: 169526
```

Next we have the **LZW** coding

In this code I make my own LZW based on the example given “Comparison of Entropy and Dictionary Based Text Compression in English, German, French, Italian, Czech, Hungarian, Finnish, and Croatian”.

I have 2 dictionaries, one for the encoder and the decoder, the encoder updates before the message is sent and the decoders dictionary updates after the message is sent just like a real LZW

Then is the main loop of the LZW

The part in red is the heart of the LZW updating the dictionaries, and then doing two things:

Finding the code of the message which is its index in the dictionary_decoder and then converting that to binary and then printing that to a txt file, we don’t read the txt file, its used later to calculate the total bits sent

And the actual decider which receives the code and then finds the corresponding word using its index and then writes that to the decoded file

Then as stated I read the encoded file to find the total bits sent and compare to the orignal

```
file = open("text.txt", "r")
file_encoded = open("encoded.txt", "a")
file_decoded = open("decoded.txt", "a")


file_encoded.truncate(0)
file_decoded.truncate(0)
```

```
#here are the two dictionaries
#the one for the sender which is updated as it reads the txt file
#and the one for the decoder which updates after and reacts to whats being
sent in

#66 total initial in dictionary => 2^7
#a = 0000000
#67= 1000011

dictionary =
['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','
t','u','v','w','x','y','z','A','B','C','D','E','F','G','H','I','J','K','L','M'
,'N','O','P','Q','R','S','T','U','V','W','X','Y','Z',''
',','.','','\n','0','1','2','3','4','5','6','7','8','9']

dictionary_decoder =
['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','
t','u','v','w','x','y','z','A','B','C','D','E','F','G','H','I','J','K','L','M'
,'N','O','P','Q','R','S','T','U','V','W','X','Y','Z',''
',','.','','\n','0','1','2','3','4','5','6','7','8','9']

counter = 0


decoded_message = ""

w = file.read(1) # reads the first character which gives starting point
x = "a" #start with arbitrary value, will get overwritten


while(1):

  if x != "": #contniues if we are not at the end of the txt file
    x = file.read(1) #reads the next character

    #this just checks for certain character not yet in the orinal alphabet
    #moreso unique icons like '/', '>' and most importanly ''' which i couldnt
add
    if (x not in dictionary):
      print
      dictionary.append(x)
      dictionary_decoder.append(x)


    if (w + x not in dictionary): #each times checks if w  and x are yet in
the alphabet
      #if they are not, we add them to the dictionary, and then send their
iteration from before
      dictionary.append(w + x)
```

```
      #this segment here is the decoder, which adds the new codeword to their
dictionary also
      b = dictionary[dictionary.index(w)] #retireves the sent word

      decoded_message = decoded_message + b
      c = dictionary[dictionary.index(x)] #and retreives the next character

      code = (bin(dictionary_decoder.index(w))[2:]) #the[2:] removes he '0b'
at front so we can see the real number of bits being sent later on
      code_dec = (bin(dictionary_decoder.index(w))) #we are converting to
inary so the LZW can dictionary decoder can recorrect it the binary to an
index place and find the corresponf character


      file_encoded.write(str(code))# print all the encoded words into a txt
file to be able to read the length of the total message sent

      file_decoded.write(str(dictionary_decoder[int(code_dec, 2)]))  #it then
checks the code it recieved which is the index of the code sent by encoder and
since the dictionaries are almost the same except a step behind it will be
able to make a match
      #it prints it into anoter text file
      #file_decoder.write(str(dictionary_decoder.index(w)))
      dictionary_decoder.append(b + c)  #and adds the new codeword to its own
dictionary after recieving the message codeword

      counter = counter + 1

      w = x #the starts from the last character
    else:
      w = w + x #if in dictionary, we add to it
  else:
    #here we are at the end of the file but still yet too send the last
remaaing character which we do now
    #file_decoder.write(str(dictionary_decoder.index(w)))
    file_decoded.write(str(dictionary_decoder[dictionary.index(w)]))
    file_decoded.close()
    break


file_encoded.close()
file_decoded.close()

file_encoded = open("encoded.txt", "r")

#this part is used to calculate the total amount of bits in the encoded txt
file to be compared with the orininal encoding of 169526 bits
x = "a" #start with arbitrary value, will get overwritten
```

```
total_bits = 0
file_encoded.seek(0)
characters = 0

for line in file_encoded:
    characters = characters + len(line)


print("total bits sent: " + str(characters))
print("LZW/original coding: " + str(100*(characters/169526)) + "%")
```

output:

```
total bits sent: 72510
LZW/original coding: 42.77220013449264%
```

As you can see it sent less bits than the original and is far more effective, it only uses 43% of the bits original used to send the same code

And **arithmetic** coding:

This is the longest code being 150 lines

I did all of the steps within one segment of code, which includes:

the probability of all the characters coming up

calculate the entropy

and then assigning them their own appendix free code word via arithmetic coding,

and the encoding the text file and putting it into another txt file,

then read the encoded file bit by bit seeing if it matches with any code word if it doesn't I add the next one, if it does I'd write the word into an alternative decoded file to see the results

and then compare the amount of bits by arithmetic to the original

```
import math

file = open("text.txt", "r")
file_encoder = open("encoded.txt", "a")


file_encoder.truncate(0)

#here are the two dictionaries
#the one for the sender which is updated as it reads the txt file
#and the one for the counter which updates aswell but with 0 instead of the
character
```

```
dictionary = []

dictionary_counter = []

dictionary_codes = []

x = "a" #start with arbitrary value, will get overwritten

#this loop counts how many of each character are in the txt file
while(1):

  if x != "": #continues if we are not at the end of the txt file
    x = file.read(1) #reads the next character

    #this just checks for certain character not yet in the orinal alphabet
    #moreso unique icons like '/', '>' and most importanly ''' which i couldnt
add
    if (x not in dictionary):
      dictionary.append(x)
      dictionary_counter.append(int(0))


    dictionary_counter[dictionary.index(x)] =
dictionary_counter[dictionary.index(x)] + 1
  else:
    #here we are at the end of the file but still yet too read the last
remaning character which we do now
    dictionary_counter[dictionary.index(x)] =
dictionary_counter[dictionary.index(x)] + 1

    break


total = 0

#the total is used for the probability of a word in the document
for i in range(len(dictionary_counter)):
  total = total + dictionary_counter[i]


#prints out all the words and the number of them occuring and their
probability
for i in range(len(dictionary_counter)):
  print(dictionary[i] + ": " + str(dictionary_counter[i]) + ": " +
str(dictionary_counter[i]/total))


entropy = 0
#this caluclates the entropy
```

```
#it uses the simple entropy equation given in the notes
for i in range(len(dictionary_counter)):
  entropy = entropy +
(dictionary_counter[i]/total)*math.log(1/(dictionary_counter[i]/total), (2))

print("entropy: " + str(entropy))


#arithmetic coding
a = 0
np = 0
n = 0
P = 0
P_div = 0
power_counter = 0
#in this loop we cycle through each charcter in the array and by looking at
its probability we build a code word for it prefix free
for i in range(len(dictionary)):
  P = dictionary_counter[i]/total
  P_div = 1/P

  while(1):
    if (2**power_counter > P_div):
      n = power_counter + 1
      break
    else:
      power_counter = power_counter + 1

  #this is the equation "c-1<2^n(a)<c" and then converted to binary with the
front 0s intact
  code = (bin(round((2**n)*(a)))[2:].zfill(n))
  #new dictionary to store all the codes
  dictionary_codes.append(code)
  print(dictionary[i] + ": " + str(code) + ", length: " + str(n))
  #dictionary_codes.append(bin(round(code)))

  a = a + P

  n = 0
  P = 0
  P_div = 0
  power_counter = 0



#this part here re reads the file and applies each character its code word and
then writes it into the decoder.txt file
x = 'a'
file.seek(0)
```

```
while(1):

  if x != "": #continues if we are not at the end of the txt file
    x = file.read(1) #reads the next character


    file_encoder.write(str(dictionary_codes[dictionary.index(x)]))


  else:
    #here we are at the end of the file but still yet too read the last
remaning character which we do now
    file_encoder.write(str(dictionary_codes[dictionary.index(x)]))

    break


#finally this loop reads teh encoded txt file and then decodes it printing it
into decoded.txt
file_encoder.close()
file_encoder = open("encoded.txt", "r")

file_decoded = open("decoded.txt", "a")
file_decoded.truncate(0)

x = 'a'
file.seek(0)

for i in range(total):
  if x != "": #contniues if we are not at the end of the txt file
    x = file.read(1) #reads the next character

    #checks if its a codeword
    if (x in dictionary):
      #if it it converts it back and writes to decoded.txt
      file_decoded.write(x)
      x = "0"
    else:
      #if not adds the next bit
      x = x + file.read(1)

file_decoded.close()

#this part is used to calculate the total amount of bits in the encoded txt
file to be compared with the orininal encoding of 169526 bits
x = "a" #start with arbitrary value, will get overwritten
total_bits = 0
file_encoder.seek(0)
```

```
characters = 0

for line in file_encoder:
    characters = characters + len(line)


print("total bits sent:" + str(characters))
print("arithmetic/original coding: " + str(100*(characters/169526)) + "%")
```

the output for this is quite long so ill sample the different segments

this part prints out ALL the characters, how many appeared and then their probability of occurring

```
C: 50: 0.0020644948181180066
H: 24: 0.0009909575126966432
A: 94: 0.0038812502580618524
R: 57: 0.0023535240926545273
T: 120: 0.004954787563483215
E: 114: 0.004707048185309055
```

The entropy

```
entropy: 4.483593586537159
```

Then all the encoded words, stored in the dictionary the order they appear

```
C: 0000000000, length: 10
H: 00000000100, length: 11
A: 0000000011, length: 10
R: 0000000111, length: 10
T: 000000101, length: 9
E: 000000111, length: 9
```

And then comparing it to the original and total bits sent

```
total bits sent:146499
arithmetic/original coding: 86.41683281620519
```

Fyi the decoded file looks is an exact copy of the text.txt for both LZW and arithmetic

In both LZW and arithmetic I was able to encode the text file effectively and reduce the total amount of bits being sent, I was able to have it read binary and have the binary hold the 0's at the front for the code also

There are flaws with my code design

Arithmetic, requires the txt file to be read first before being designed, though this is not ideal, I do think that this is the only way to it as its impossible to find probability without knowing the whole scope

The main problem was with LZW coding not being prefix free, I feel though that LZW would be used to send messages one code word at a time and if done this is way to will be done very effectively

Else without knowing the total amount of codewords you're going to have in your dictionary before sending the file it would be very hard to design a prefix free LZW consistently since it depends on the file being sent