Faculty of Engineering & Technology

Department of Electrical and Computer Engineering

Information and Coding Theory - ENEE5304

Course Assignment

Prepared by:

Zeina Odeh 1190083

Alaa Sehwail 1191741

Supervised by:

Dr. Wael Hashlamon

Section: 1

Dec | 2023

Table of Contents

Table of Figures

1. Introduction

In this assignment, we will learn how to implement Huffman code using the Python framework (at pyCharm) and test it on a large document, also learn how to find the frequency and probability for each character, and to find the entropy for the alphabet, and find the total number of bits needed to encode the entire document using Huffman code and compare this number if the ASCII code is used, in addition to learning how to calculate the average number of bits/character for whole document using Huffman code. Finally, find the percentage of compression accomplished using the Huffman encoding compared to ASCII code.

2. Theoretical Background

Huffman coding is a lossless technique for data compression. In essence, it assigns variable-length codes to input characters according to the frequency of the corresponding characters. Prefix Codes are the codes supplied to input characters that make a character's code (bit sequence) never become the prefix for another character's code. This feature of Huffman Coding ensures that the resultant bitstream may be decoded without any confusion.

How to construct a Huffman Tree [1] :

A Huffman Tree is the output, while the input is an array of distinct characters together with how often each character appears.

1. For every distinct letter, create a leaf node, and then construct a min heap from all of the leaf nodes. The min heap serves as a priority queue. The two nodes in the min heap are compared using the value of the frequency field. At root, at first, is the least common character.

2. Take two nodes out of the min heap that have the lowest frequency.

3. Assign the frequency of the newly created internal node to the total of the frequencies of the two nodes. Assign the left child status to the first extracted node and the right child status to the second extracted node. To the min heap, add this node.

4. Continue steps #2 and #3 until there is just one node left in the heap. The tree is finished when the root node is the last one to remain.

3. Results

Firstly, we calculated the total number of characters and unique symbols without repetition as follows:

```
************************ Total Number of Characters in Document:  37706
```

Figure 1: Total Number of Characters

Then, we calculate the number of occurrences for each character in the document, and calculated the probability for each character as:

$$\text{Probability/Character} = \frac{\text{Frequency/Character}}{Total\ Number\ of\ Characters}$$

After that, we calculated the Codeword for each character using Huffman code based on the Huffman tree built into the code.

Finally, we calculate the length for each Codeword.

Symbol	Frequency	Probability	Codeword	Length of Codeword
	7049	0.186946	111	3
!	3	0.000080	00011111011011	14
"	2	0.000053	00011111011001	14
'	20	0.000530	00011111001	11
,	436	0.011563	000110	6
-	89	0.002360	000111111	9
.	414	0.010980	000100	6
:	2	0.000053	00011111011010	14
;	26	0.000690	0001111000	10
?	1	0.000027	00011111011000	14
a	2264	0.060043	1001	4
b	484	0.012836	100000	6
c	779	0.020660	110110	6
d	1515	0.040179	11010	5
e	3887	0.103087	010	3
f	794	0.021058	00000	5
g	620	0.016443	100001	6
h	2278	0.060415	1010	4
i	1983	0.052591	0110	4
j	20	0.000530	00011111010	11
k	304	0.008062	1011000	7
l	1127	0.029889	10001	5
m	678	0.017981	101101	6
n	2077	0.055084	0111	4
o	1971	0.052273	0011	4
p	421	0.011165	000101	6
q	17	0.000451	00011111000	11
r	1481	0.039278	10111	5
s	1795	0.047605	0010	4
t	2937	0.077892	1100	4
u	800	0.021217	00001	5
v	179	0.004747	0001110	7
w	788	0.020899	110111	6
x	34	0.000902	0001111001	10
y	356	0.009441	1011001	7
z	61	0.001618	000111101	9
–	14	0.000371	000111110111	12

Now, we calculated the Entropy of the characters in the document as the following equation:

$$Entropy = \sum Probability * \log_2 \frac{1}{Probability}$$

And the result for the Entropy given in the figure below:

```
************************ Entropy of Characters in the Document: ************************
Entropy: 4.172002 bits
```

Figure 2: Entropy

After that, calculate the average bits / character using Huffman code as the following equation:

$$Average\ bits\ per\ character = \sum Codeword\ Length * Probability$$

And the result for the average bits / character given in the figure below:

```
************************ Average Bits/Character using Huffman Coding: ************************
Average Bits/Character: 4.218506 bits
```

Figure 3: Average bits / Character

As we noticed, average bits / character are greater than the value of entropy, based on Shannon source coding theorem: (average bits/character never goes below H(X)) and this emphasize that the implementation of the Huffman code is correct.

Then, we calculated the number of bits if the ASCII code is used as the following equation:

$$Number\ of\ bits_{ASCII} = 8 * Total\ Number\ of\ Character$$

And the result for number of bits if the ASCII code is used given in the figure below:

```
************************ Number of Bits for ASCII Encoding (N_ASCII): ************************
N_ASCII: 301648 bits
```

Figure 4: Number of bits using ASCII code

Then, we calculated the number of bits if the Huffman code is used as the following equation:

$$Number\ of\ bits_{Huffman} = \sum Codeword\ Length\ for\ each\ Character\ *\ Character\ Frequency$$

The result for a number of bits if the Huffman code is used is given in the figure below:

```
************************ Total Number of Bits Needed To Encode the Entire Story Using Huffman Code (N_Huffman): ************************
Total Number of Bits Using Huffman: 159063 bits
```

Figure 5: Number of bits using Huffman code

Then, we calculated the percentage of compression accomplished by using the Huffman encoding as compared to ASCII code using following equation:

$$Percentage\ of\ Compression = \frac{Number\ of\ bits\ using\ Huffman}{Number\ of\ bits\ using\ ASCII}$$

The result for percentage of compression accomplished by using the Huffman encoding as compared to ASCII code is given in the figure below:

```
********** Percentage Huffman-ASCII : **********
Percentage of Compression: 52.731329
```

Figure 6: Percentage Huffman-ASCII

We noticed that when use ASCII for encoding the number of bits is greater than number of bits when used Huffman encoding, so; using Huffman code is more efficient (Optimal Code). Also noticed that Huffman code works well for text transmissions, since it constructs a code with small average Codeword length.

4. Conclusion

As Conclusion, we learned how to implement Huffman code using the Python framework (at pyCharm) and tested it on a large document, also learned how to find the frequency and probability for each character, and found the entropy for the alphabet, and found the total number of bits needed to encode the entire document using Huffman code and compared this number if the ASCII code is used, in addition, calculated the average number of bits/character for whole document using Huffman code. Finally, found the percentage of compression accomplished using the Huffman encoding compared to ASCII code. And conclude that using Huffman code for encoding gives less number of bits than use ASCII code.

5. References

[1] *Huffman Coding Greedy Algo 3*. (2023, September 11). GeeksforGeeks. https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/

6. Appendix

```
import heapq
import math
import pandas as pd
from docx import Document


doc_path = 'Story.docx'
def build_huffman_tree(freq):
    heap = [[prob, [char, ""]] for char, prob in freq.items()] #for each char has an empty text for the code according to prob.
    heapq.heapify(heap)  #sort heap descending way
    while len(heap) > 1:
        # line 13 + 14 pop the lowest prob. for char in heap
        lo = heapq.heappop(heap)
        hi = heapq.heappop(heap)
        # assign 0 and 1
        for pair in lo[1:]:
            pair[1] = '0' + pair[1]
        for pair in hi[1:]:
            pair[1] = '1' + pair[1]
        heapq.heappush(heap, [lo[0] + hi[0]] + lo[1:] + hi[1:])  #combine char with its codeword
    return heap[0][1:]


def huffman_codes(tree):
    codes = {}
    for char, code in tree:
        codes[char] = code
    return codes
```

```
doc = Document(doc_path)
character_counts = {}     #number of ocureences for each char.
characters = set()       #unique char.
total_characters = 0

for paragraph in doc.paragraphs:
    text = paragraph.text.strip().lower()
    for char in text:
        if char.isalpha() or char in ':,.;?!\'"-— ':  #check char.
            characters.add(char)
            character_counts[char] = character_counts.get(char, 0) + 1
            total_characters += 1

for table in doc.tables:
    for row in table.rows:
        for cell in row.cells:
            text = cell.text.strip().lower()
            for char in text:
                if char.isalpha() or char in ':,.;?!\'"-— ':
                    characters.add(char)
                    character_counts[char] = character_counts.get(char, 0) + 1
                    total_characters +=1

unique_characters = sorted(characters) #sort char.



character_probabilities = {}
for char, count in character_counts.items():
    character_probabilities[char] = count / total_characters #calculate the prob.
```

```
entropy = 0.0
for char, probability in character_probabilities.items():
    entropy += probability * math.log2(1 / probability)  #calculate the entropy

tree = build_huffman_tree(character_probabilities)

codes = huffman_codes(tree)

bits_per_ascii_character = 8 #ASCII value for each char.
NASCII = total_characters * bits_per_ascii_character #number of bits needed if using ASCII to encode the whole document

print('\n')
print(f"********** Total Number of Characters in Document: {total_characters}")

data = {
    'Symbol': unique_characters,
    'Frequency': [character_counts[char] for char in unique_characters],
    'Probability': [character_probabilities[char] for char in unique_characters],
    'Codeword': [codes[char] for char in unique_characters],
    'Length of Codeword': [len(codes[char]) for char in unique_characters]
}

df = pd.DataFrame(data)
print('\n')
print(df.to_string(index=False))
print('\n')

print('********** Entropy of Characters in the Document: **********')
```

```
print(f'Entropy: {entropy:.6f} bits')

print('\n')


print(f"************************ Number of Bits for ASCII Encoding (N_ASCII): ************************")

print(f"N_ASCII: {NASCII} bits")

print('\n')

#calculate average bits/char.

average_bits_per_character = sum(len(code) * character_probabilities[char] for char, code in codes.items())


print('************************ Average Bits/Character using Huffman Coding: ************************')

print(f'Average Bits/Character: {average_bits_per_character:.6f} bits')

print('\n')

#number of bits needed if using Huffman code to encode the whole document

Nhuffman = sum(len(code) * character_counts[char] for char, code in codes.items())


print('************************ Total Number of Bits Needed To Encode the Entire Story Using Huffman Code (N_Huffman): ************************')

print(f'Total Number of Bits Using Huffman: {Nhuffman} bits')

print('\n')


percentage_Huffman_ASCII = Nhuffman / NASCII * 100

print('************************ Percentage Huffman-ASCII : ************************')

print(f'Percentage of Compression: {percentage_Huffman_ASCII:.6f} ')

print('\n')
```