Birzeit University

Faculty of Engineering and Technology

Electrical and Computer Engineering Department

ENEE5304
INFORMATION AND CODING THEORY

"Course Project on Source Coding"

Prepared by: Jenin Mansour 1200540

Instructor: Dr. Wael Hashlamoun

Date of submission: 9/1/2023

2023-2024

Introduction:

This project includes implementation of a python program that uses the Huffman code to encode an English short story “To Build A Fire by Jack London” to find the code-words for each characters. The program will display a summary of the encoding process by printing various statistics such as the number of characters in the story along with their frequency of occurrence and their probability(We did not distinguish between capital and small letters, also we skip the “enter" character), also we find the average number of bits needed per character using Huffman, and the Entropy of the alphabet the we found the Compression ratio for Entropy to Huffman bits, also the total number of bits needed using ASCII encoding, percentage of compression Total number of bits needed using Huffman code compared to ASCII code, and code-word lengths of specific characters.

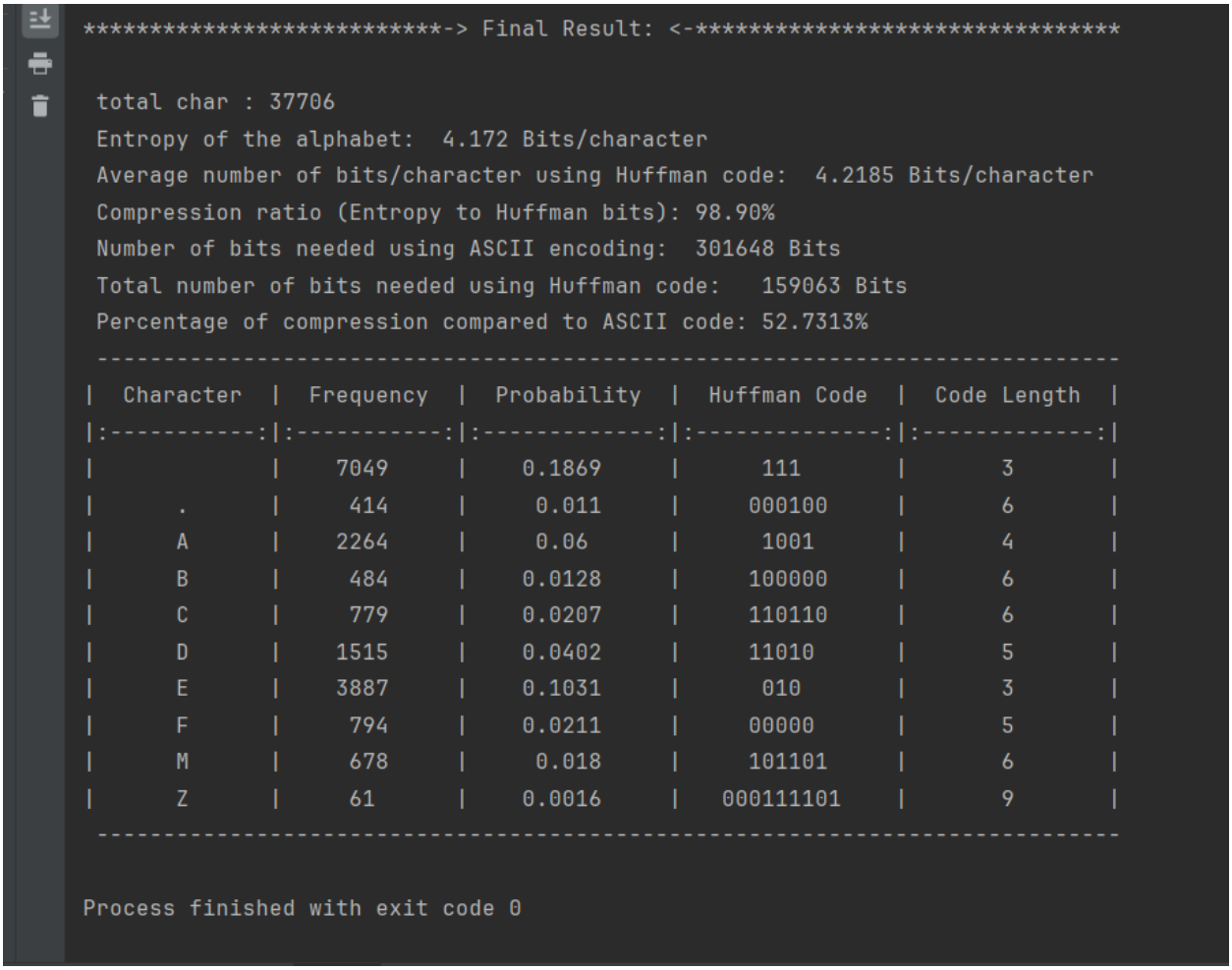
Theoretical Background:

Source coding is a technique used to organize data to avoid unnecessary length during transmission. It involves using Variable Length Code (VLC) [1]. Compressed data length is achieved by representing frequently occurring messages with shorter codes and less frequently occurring messages with longer codes. The main problem in source coding is to minimize the weighted average code-word length within the bounds of Kraft's inequality. The most widely used methods for ensuring this is Huffman coding [2].

Huffman coding is a data compression technique that scans all data and calculates the frequency of occurrence for each symbol. It then sorts symbols in descending order based on their probabilities of occurrence, marking the two with the lowest probabilities as 0 and 1. The probabilities of the two marked symbols are combined, resulting in a new probability. This process is repeated iteratively until only two symbols remain [2].

Entropy is a measure of uncertainty it represents the amount of information by an event. When the coding efficiency is 1, it means the encoding method has been optimized. So the average number of bits per symbol using Huffman must approach the entropy limit, and The total number of bits using Huffman-code must be equal the Average number of bit per symbol using Huffman multiply to the total number of character.

Results:



```
**************************-> Final Result: <-********************************

 total char : 37706
 Entropy of the alphabet:  4.172 Bits/character
 Average number of bits/character using Huffman code:  4.2185 Bits/character
 Compression ratio (Entropy to Huffman bits): 98.90%
 Number of bits needed using ASCII encoding:  301648 Bits
 Total number of bits needed using Huffman code:   159063 Bits
 Percentage of compression compared to ASCII code: 52.7313%
 ------------------------------------------------------------------------------
|  Character  |  Frequency  |  Probability  |  Huffman Code  |  Code Length  |
|:-----------:|:-----------:|:-------------:|:--------------:|:-------------:|
|             |    7049     |    0.1869     |      111       |       3       |
|      .      |     414     |     0.011     |     000100     |       6       |
|      A      |    2264     |     0.06      |      1001      |       4       |
|      B      |     484     |    0.0128     |     100000     |       6       |
|      C      |     779     |    0.0207     |     110110     |       6       |
|      D      |    1515     |    0.0402     |     11010      |       5       |
|      E      |    3887     |    0.1031     |      010       |       3       |
|      F      |     794     |    0.0211     |     00000      |       5       |
|      M      |     678     |     0.018     |     101101     |       6       |
|      Z      |     61      |    0.0016     |   000111101    |       9       |
 ------------------------------------------------------------------------------

Process finished with exit code 0
```

Conclusions:

We can notice that the (Average number of bit per symbol using Huffman * Total number of character) = total number of bits using Huffman-code ➔ 4.2185 * 37706 = 159062.7. Also, the average number of bits per symbol approaches the entropy limit ($4.17 \approx 4.21$) and the coding efficiency $\approx 100\%$; there is some errors according to the reading of file, we can also notice the huge difference between ASCII coding and Huffman coding, and the code-words for each symbol have different lengths according to its frequency so the symbols with high probability have shorter code words, and the codes are prefix-free so the code is uniquely decodable.

References:

1] https://medium.com/@acamvproducingstudio/an-introduction-to-source-coding-huffman-coding-and-coding-efficiency-32042a6b244f#:~:text=Source%20coding%20is%20a%20technique,Variable%20Length%20Code%20(VLC).

2] https://en.wikipedia.org/wiki/Shannon%27s_source_coding_theorem

Appendix:

```
import heapq
from collections import defaultdict
import math
from tabulate import tabulate



def huffman_tree(frequency_of_occurrence):
    h = [[weight, [char, ""]] for char, weight in frequency of occurrence.items()]
    heapq.heapify(h)
    while len(h) > 1:
        lo = heapq.heappop(h)
        hi = heapq.heappop(h)
        for pair in lo[1:]:
            pair[1] = '0' + pair[1]
        for pair in hi[1:]:
            pair[1] = '1' + pair[1]
        heapq.heappush(h, [lo[0] + hi[0]] + lo[1:] + hi[1:])
    return h[0][1:]
#
**********************************************************DONE**********************************
**************

def huffman coding(story):
    # Creating a defaultdict with a default value of 0 for nonexistent keys
    frequency of occurrence = defaultdict(int)

    # Updating the count for each character in the input text
    for char in story.upper():
# if char in valid characters:
          frequency_of_occurrence[char] += 1
    # value = frequency counts (how many times each character appears in the text).
    total_chars = sum(freq for freq in frequency_of_occurrence.values())

    probabilities = {}

    for char, count in frequency of occurrence.items():
        probability = count / total_chars
        probabilities[char] = probability#adds a key-value pair to the probabilities dictionary.
The key is the character (char), and the value is the calculated probability.

    huffmantree = huffman_tree(frequency_of_occurrence)
    huffman codes = {}
    for char, code in huffmantree:
        huffman_codes[char] = code

    return frequency of occurrence, total chars, probabilities, huffman codes


def entropy_calc(probabilities):
    entropy = 0
    for prob in probabilities:
        entropy += prob * (-(math.log2(prob)))
    return entropy


def huffman_bits_perchar(probabilities, huffman_codes, frequency_of_occurrence):

    total = 0
    for char in frequency of occurrence.keys():
        total += probabilities[char] * len(huffman_codes[char])
    return total


def total_huffman_bitsfunction( huffman_codes, frequency_of_occurrence):
    total = 0
    for char in frequency_of_occurrence.keys():
        total += len(huffman_codes[char]) * frequency_of_occurrence[char]
    return total

def bits needed(total chars):
    return total_chars * 8
```

```
        # Read the story
with open("story2.txt", "r", encoding="utf-8") as file:
    story = file.read().upper().replace('\n' , '')

    # perform Huffman coding
    #frequency_of_occurrence, total_chars, probabilities, huffman_codes = huffman_coding(story)
    result = huffman coding(story)
    frequency_of_occurrence = result[0]
    total_chars = result[1]
    probabilities = result[2]
    huffman codes = result[3]




    print("**************************-> Final Result: <-*******************************")

    print("\n total char :", total chars)

    entropy = entropy_calc(probabilities.values())
    print(" Entropy of the alphabet: ",round( entropy,4), "Bits/character ")

    total_huffman_bits_per_char = huffman_bits_perchar(probabilities, huffman_codes,
frequency of occurrence)
    print(" Average number of bits/character using Huffman code: ",
round(total_huffman_bits_per_char, 4), "Bits/character ")


    print(f" Compression ratio (Entropy to Huffman bits): {(entropy /
total huffman bits per char)*100:.2f}%")



    nascci_bits = bits_needed(total_chars)
    print(" Number of bits needed using ASCII encoding: ", nascci_bits,"Bits ")

    Nhuffman =total_huffman_bitsfunction( huffman_codes, frequency_of_occurrence)
    print( " Total number of bits needed using Huffman code:  ", round(Nhuffman,4), "Bits ")

    percentage_of_compression = (Nhuffman / nascci_bits)
    print(" Percentage of compression compared to ASCII code: " +
str(round(percentage of compression*100, 4))+"%")


    print(" ---------------------------------------------------------------------------")


    scharacters = {'A', 'B', 'C', 'D', 'E', 'F', 'M', 'Z', ' ', '.'}
    chtable = sorted(
        [(char, freq, round(probabilities[char], 4), huffman_codes[char],
len(huffman_codes[char])) for char, freq in frequency_of_occurrence.items()if char in
scharacters])
    print(tabulate(chtable, headers=["Character", "Frequency", "Probability", "Huffman Code",
"Code Length"],
                   tablefmt="pipe", colalign=("center", "center", "center", "center", "center")))

    print(" ---------------------------------------------------------------------------")
```