Федеральное государственное бюджетное образовательное учреждение высшего образования  
«Сибирский государственный университет телекоммуникаций и информатики»  
(СибГУТИ)

Институт информатики и вычислительной техники  
Кафедра прикладной математики и кибернетики

**Практическая работа №2  
 по дисциплине «Теория информации»  
на тему «Побуквенное кодирование текстов»**

Выполнили:  
студенты гр.ИП-014

Обухов А.И.

Проверила:  
Старший преподаватель каф. ПМиК  
Дементьева Кристина Игоревна

Новосибирск 2024 г.

**Цель работы:** Экспериментальное изучение избыточности сжатия текстового файла.

**Язык программирования:** С, С++, С#, Python

**Результат:** программа, тестовые примеры, отчет.

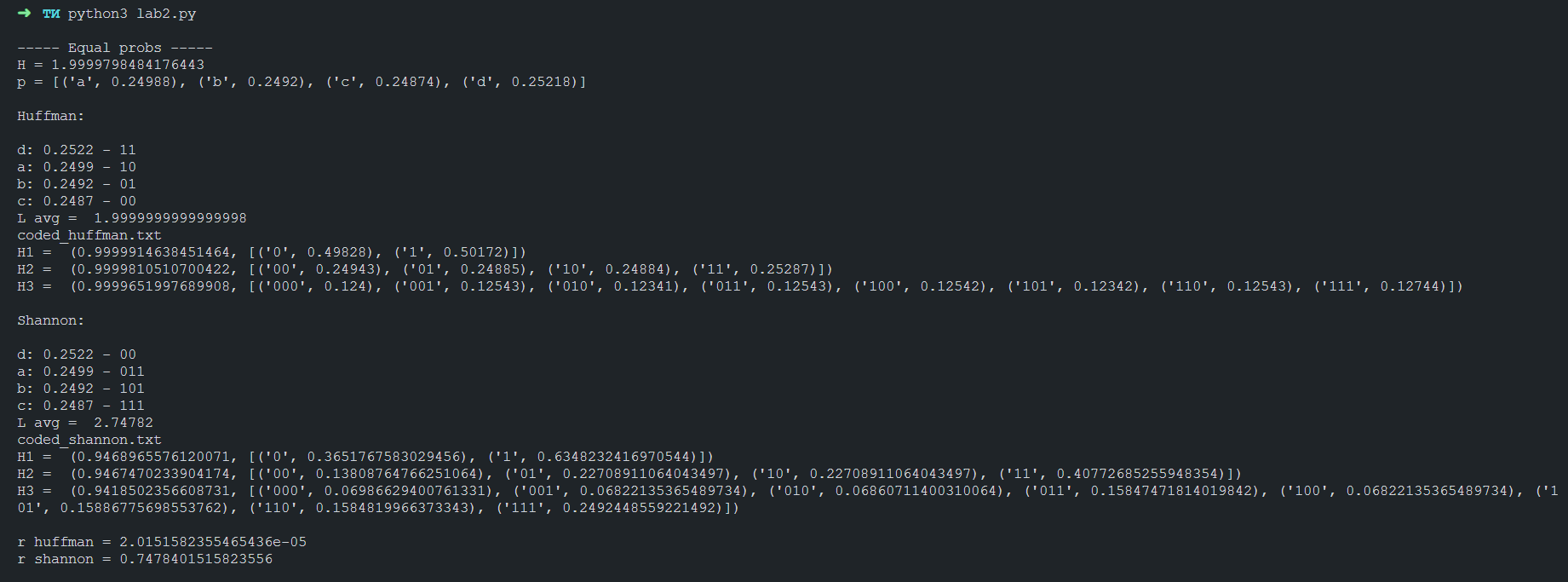
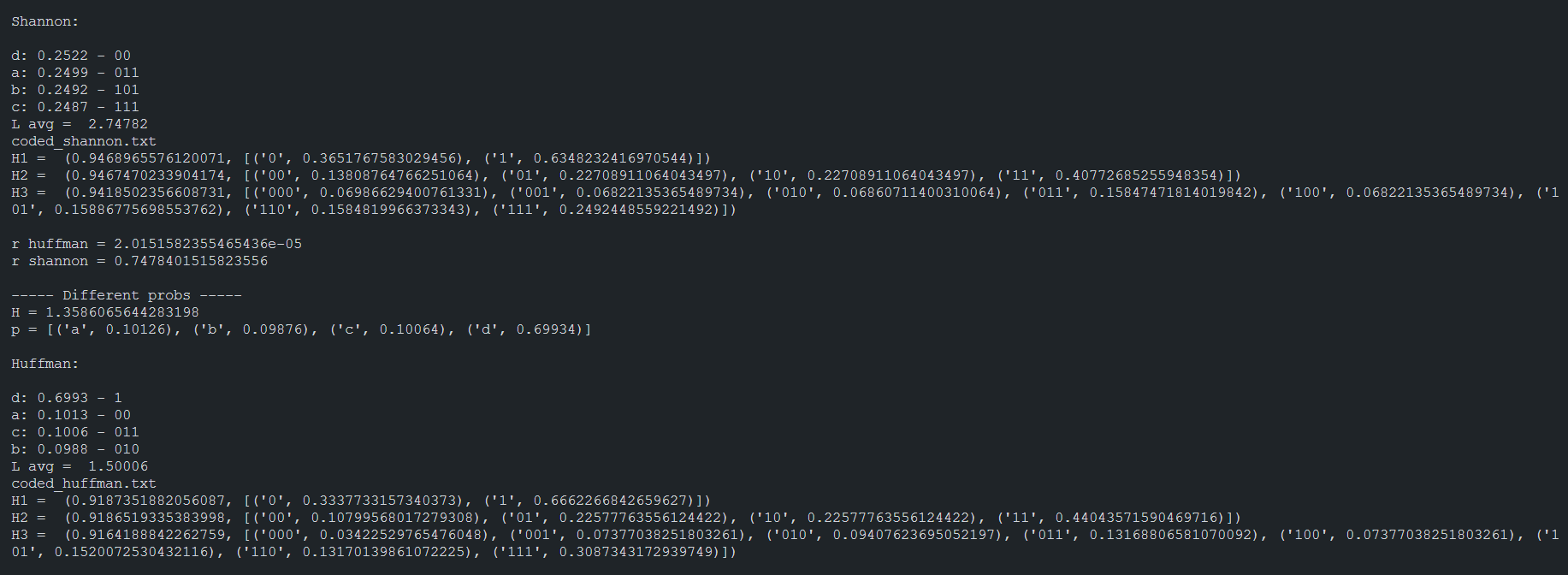
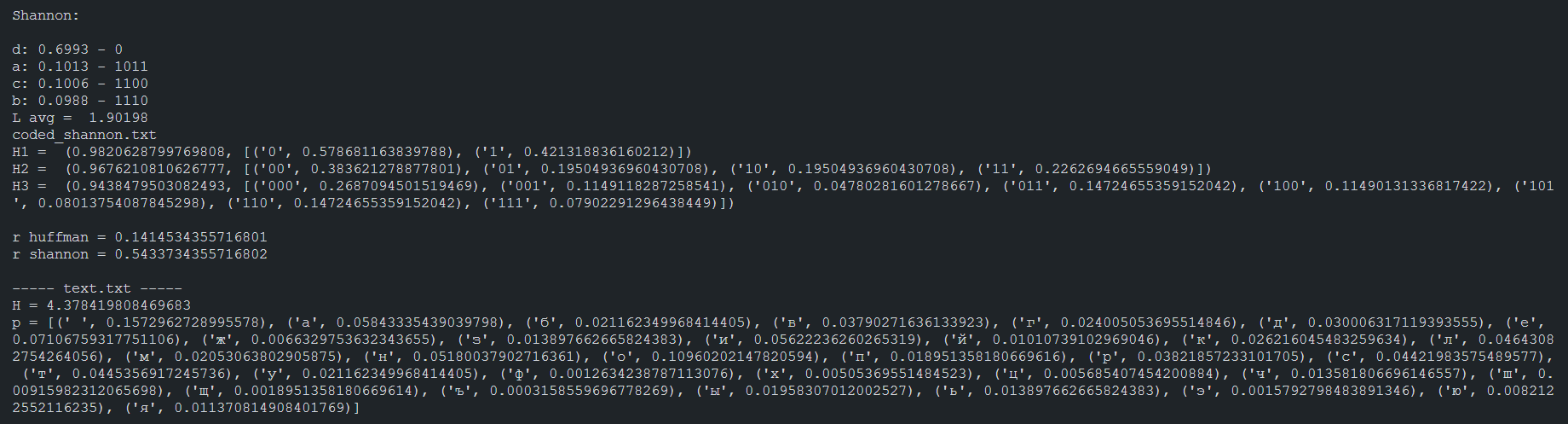
**Задание:**

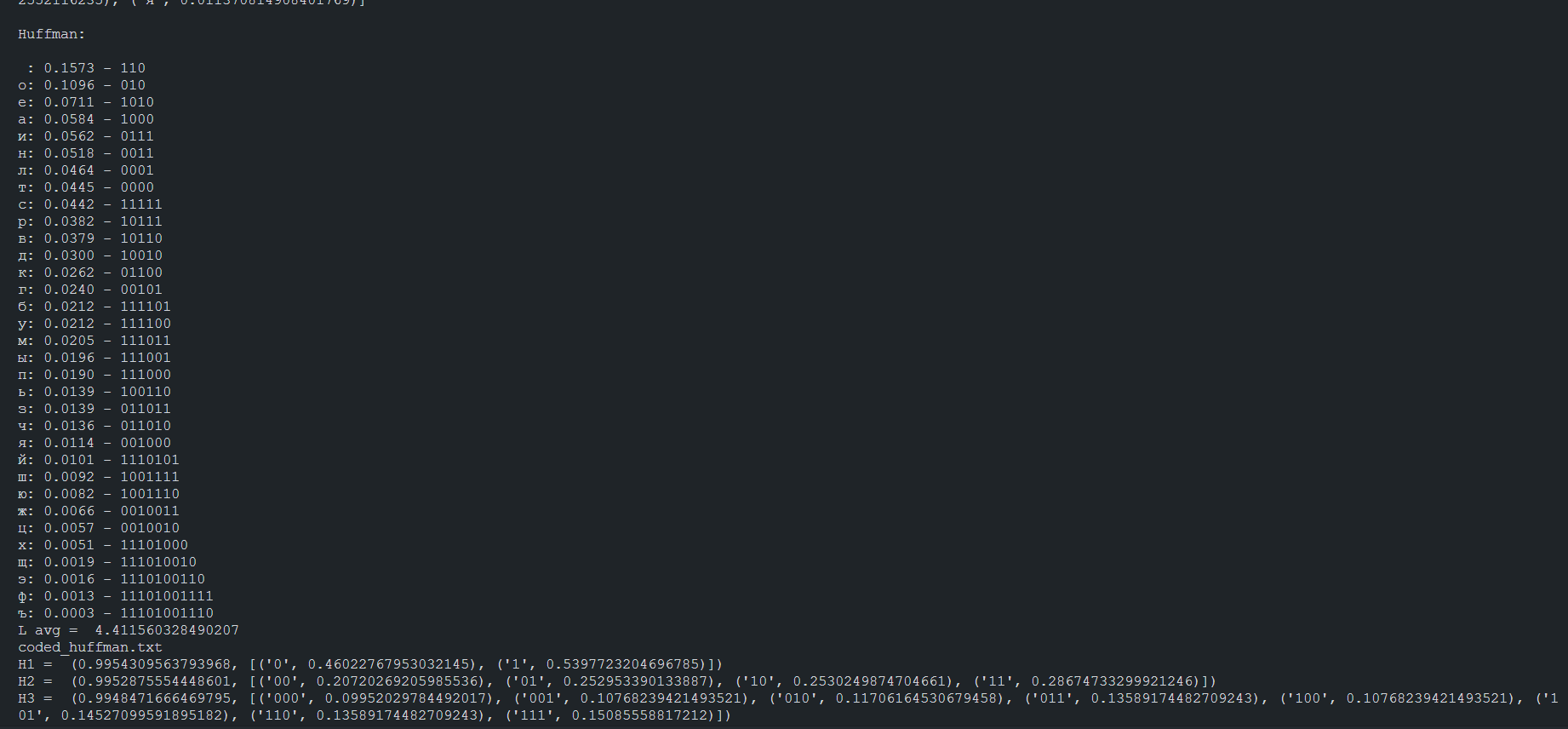
1. Запрограммировать процедуру двоичного кодирования текстового файла побуквенным кодом. В качестве методов сжатия использовать метод Хаффмана и метод Шеннона (или метод Фано). Текстовые файлы использовать те же, что и в практической работе 1.

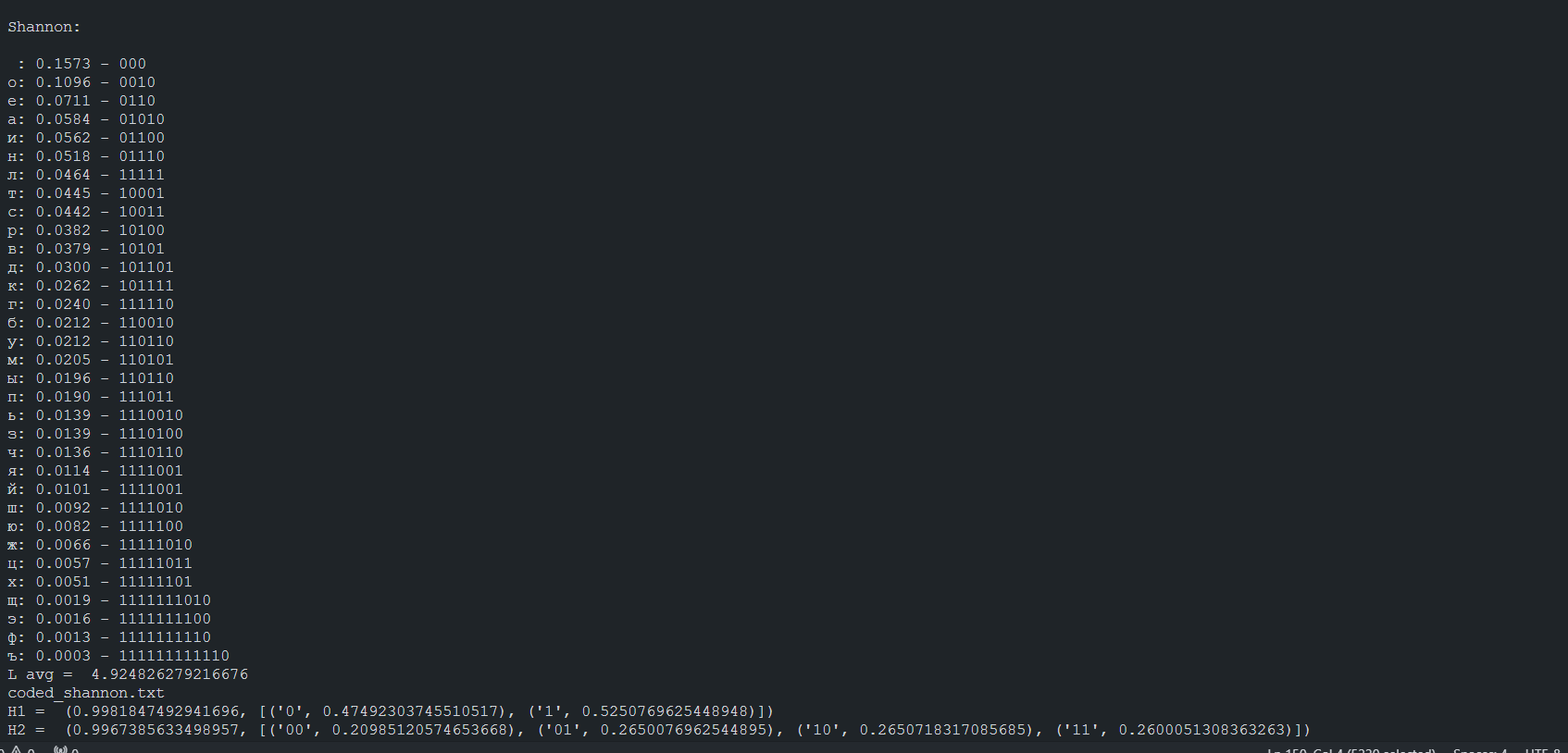
2. Вычислить среднюю длину кодовых слов и оценить избыточность кодирования для каждого построенного побуквенного кода.

3. После кодирования текстового файла вычислить оценки энтропии файла с закодированным текстом*H*1**,** *H*2**,** *H*3(после кодирования последовательность содержит 0 и 1) и заполнить таблицу.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод кодирования | Название текста | Оценка  избыточности кодирования |  |  |  |
| Метод Хаффмана | text.txt | 0.02595 | 0.99669 | 0.99669 | 0.99664 |
| Метод Шеннона | text.txt | 0.4421 | 0.99001 | 0.98942 | 0.98796 |

**Скриншоты работы программы:**  
****





**Анализ результатов работы программы**

В ходе выполнения практической работы были проведены эксперименты по вычислению оценок энтропии Шеннона текстов с использованием различных методов побуквенного кодирования. Для этого были использованы три текстовых файла с разными свойствами, включая последовательность символов с равными вероятностями, последовательность символов с различными вероятностями и художественный текст на русском языке.

Были реализованы три метода оценки энтропии Шеннона:   
1. Первый метод (H1) оценивает энтропию по отдельным символам файла.  
2. Второй метод (H2) оценивает энтропию по парам символов.  
3. Третий метод (H3) оценивает энтропию по тройкам символов.

Для каждого текстового файла были вычислены указанные оценки энтропии для каждого метода. Результаты показали, что значения оценок энтропии различаются в зависимости от метода и свойств текста. В частности, метод Хаффмана обеспечивает более оптимальное сжатие по сравнению с методом Шеннона для данного текста "text.txt".

Также были рассмотрены значения избыточности кодирования для обоих методов. Полученные результаты позволяют сделать вывод о более эффективном использовании битовых ресурсов при кодировании с использованием метода Хаффмана.

Таким образом, выполнение практической работы позволило изучить основные свойства энтропии Шеннона и сравнить различные методы ее оценки. Полученные результаты могут быть полезны для анализа эффективности и качества методов побуквенного кодирования при работе с текстовыми данными.

**Листинг программы**

import heapq

from math import ceil

from lab1 import \*

class Node:

    def \_\_init\_\_(self, char, freq):

        self.char = char

        self.freq = freq

        self.left = None

        self.right = None

    def \_\_lt\_\_(self, other):

        return self.freq < other.freq

def huffman\_encode(line: str) -> float:

    print("\nHuffman:\n")

    def build\_huffman\_tree(text):

        char\_freq = Counter(text)

        heap = [Node(char, freq) for char, freq in char\_freq.items()]

        heapq.heapify(heap)

        while len(heap) > 1:

            left = heapq.heappop(heap)

            right = heapq.heappop(heap)

            merged = Node(None, left.freq + right.freq)

            merged.left = left

            merged.right = right

            heapq.heappush(heap, merged)

        return heap[0]

    def build\_huffman\_codes(node, prefix="", codes={}):

        if node:

            if node.char is not None:

                codes[node.char] = prefix

            build\_huffman\_codes(node.left, prefix + "0", codes)

            build\_huffman\_codes(node.right, prefix + "1", codes)

    split\_line = list(line[i: i + 1] for i in range(len(line)))

    probabilities = {k: v / len(split\_line) for k, v in Counter(split\_line).items()}

    probabilities = dict(sorted(probabilities.items(), key=lambda item: item[1], reverse=True))

    root = build\_huffman\_tree(line)

    codes = {}

    build\_huffman\_codes(root, "", codes)

    for i in probabilities.keys():

        print(f"{i}: {probabilities[i]:.4f} - {codes[i]}")

    l\_average = sum(probabilities[i] \* len(codes[i]) for i in probabilities.keys())

    print("L avg = ", l\_average)

    print("coded\_huffman.txt")

    with open("./output/coded\_huffman.txt", "w") as f:

        for i in line:

            f.write(codes[i])

    with open("./output/coded\_huffman.txt", "r") as f:

        text = f.readline()

    for i in range(1, 4):

        print(f'H{i} = ', calc\_entropy(text, i))

    return l\_average

def shannon\_encode(line: str) -> float:

    print("\nShannon:\n")

    def decimal\_converter(num):

        if num == 0.0:

            return 0.0

        while num > 1:

            num /= 10

        return num

    def float\_bin(number: float, places: int):

        whole, dec = str(number).split(".")

        whole = int(whole)

        dec = int(dec)

        res = bin(whole).strip("0b") + "."

        for x in range(places):

            whole, dec = str((decimal\_converter(dec)) \* 2).split(".")

            dec = int(dec)

            res += whole

        return res

    split\_line = list(line[i: i + 1] for i in range(len(line)))

    probabilities = {k: v / len(split\_line) for k, v in Counter(split\_line).items()}

    probabilities = dict(sorted(probabilities.items(), key=lambda item: item[1], reverse=True))

    code\_length = [ceil(-log2(i)) for i in probabilities.values()]

    cumulative\_probs = [float(0) for \_ in range(len(probabilities))]

    for i in range(1, len(probabilities)):

        cumulative\_probs[i] = cumulative\_probs[i - 1] + list(probabilities.values())[i - 1]

    codes = list()

    for i in range(len(cumulative\_probs)):

        codes.append(float\_bin(cumulative\_probs[i], code\_length[i])[1:])

        print(f"{list(probabilities.keys())[i]}: {list(probabilities.values())[i]:.4f} - {codes[i]}")

    l\_average = sum(list(probabilities.values())[i] \* code\_length[i] for i in range(len(probabilities.items())))

    print("L avg = ", l\_average)

    print("coded\_shannon.txt")

    with open("./output/coded\_shannon.txt", "w") as f:

        for i in line:

            index = list(probabilities.keys()).index(i)

            f.write(codes[index])

    with open("./output/coded\_shannon.txt", "r") as f:

        text = f.readline()

    for i in range(1, 4):

        print(f'H{i} = ', calc\_entropy(text, i))

    return l\_average

def main():

    print("\n----- Equal probs -----")

    input\_text = preprocess\_file('./input/equal\_prob.txt', 'en')

    orig\_entropy = calc\_entropy(input\_text, 1)

    print(f"H = {orig\_entropy[0]}\np = {orig\_entropy[1]}")

    l\_avg\_huf = huffman\_encode(input\_text)

    l\_avg\_shan = shannon\_encode(input\_text)

    print(f"\nr huffman = {l\_avg\_huf - orig\_entropy[0]}\nr shannon = {l\_avg\_shan - orig\_entropy[0]}")

    print("\n----- Different probs -----")

    input\_text = preprocess\_file('./input/diff\_prob.txt', 'en')

    orig\_entropy = calc\_entropy(input\_text, 1)

    print(f"H = {orig\_entropy[0]}\np = {orig\_entropy[1]}")

    l\_avg\_huf = huffman\_encode(input\_text)

    l\_avg\_shan = shannon\_encode(input\_text)

    print(f"\nr huffman = {l\_avg\_huf - orig\_entropy[0]}\nr shannon = {l\_avg\_shan - orig\_entropy[0]}")

    print("\n----- text.txt -----")

    input\_text = preprocess\_file('./input/text.txt', 'ru')

    orig\_entropy = calc\_entropy(input\_text, 1)

    print(f"H = {orig\_entropy[0]}\np = {orig\_entropy[1]}")

    l\_avg\_huf = huffman\_encode(input\_text)

    l\_avg\_shan = shannon\_encode(input\_text)

    print(f"\nr huffman = {l\_avg\_huf - orig\_entropy[0]}\nr shannon = {l\_avg\_shan - orig\_entropy[0]}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

*```*

    split\_line = list(line[i: i + 1] for i in range(len(line)))

    probabilities = {k: v / len(split\_line) for k, v in Counter(split\_line).items()}

    probabilities = dict(

        sorted(probabilities.items(), key=lambda item: item[1], reverse=True)

    )

    # print(probabilities)

    code\_length = [ceil(-log2(i)) for i in probabilities.values()]

    # print(code\_length)

    cumulative\_probs = [float(0) for \_ in range(len(probabilities))]

    for i in range(1, len(probabilities)):

        cumulative\_probs[i] = (

            cumulative\_probs[i - 1] + list(probabilities.values())[i - 1]

        )

    # print(cumulative\_probs)

    codes = list()

    for i in range(len(cumulative\_probs)):

        codes.append(float\_bin(cumulative\_probs[i], code\_length[i])[1:])

        print(

            f"{list(probabilities.keys())[i]}: {list(probabilities.values())[i]:.4f} - {codes[i]}"

        )

    # print(codes)

    l\_average = sum(

        list(probabilities.values())[i] \* code\_length[i]

        for i in range(len(probabilities.items()))

    )

    print("L avg = ", l\_average)

    print("coded\_shannon.txt")

    with open("./output/coded\_shannon.txt", "w") as f:

        for i in line:

            index = list(probabilities.keys()).index(i)

            f.write(codes[index])

    with open("./output/coded\_shannon.txt", "r") as f:

        text = f.readline()

    for i in range(1, 4):

        print(f'H{i} = ', calc\_entropy(text, i))

    return l\_average

def main():

    print("\n~~~text.txt~~~")

    input\_text = preprocess\_file('./input/text.txt', 'ru')

    orig\_entropy = calc\_entropy(input\_text, 1)

    print(f"H = {orig\_entropy[0]}\np = {orig\_entropy[1]}")

    l\_avg\_huf = huffman\_encode(input\_text)

    l\_avg\_shan = shannon\_encode(input\_text)

    print(f"\nr huffman = {round(l\_avg\_huf - orig\_entropy[0], 5)}\nr shannon = {round(l\_avg\_shan - orig\_entropy[0], 5)}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()