Course_Project_1B

专业班级：**提高2201班**

姓名：　　**王翎羽**

学号：　　**U202213806**

实验环境

编程语言：***Python***

实验工具：***PyCharm 2023.1.3、Jupyter Notebook***

实验任务

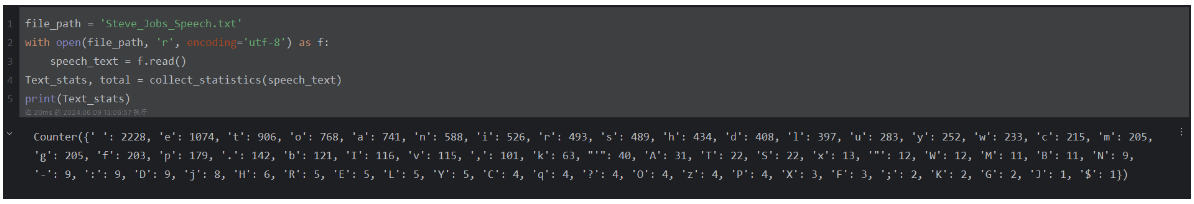
- *Read Steve Jobs’ 2005 Stanford Commencement Address “You’ve got to find what you love”as in the file ”Steve Jobs Speech.txt”.*
- *Collect the statistics of the letters, punctuation, space in “Steve Jobs Speech.doc”.*
- *Compute the entropy of “Steve Jobs Speech.doc”.*
- *Apply the Huffman coding method and Shannon coding method for “Steve Jobs Speech.doc”.Output the letters/punctuation/space and their Huffman codewords and Shannon codes,respectively.*
- *Compute the average code length of “Steve Jobs Speech.doc” using your Huffman codes andShannon codes, respectively.*

Note: For the Huffman code, please develop your code to generate a Q-ary Huffman code,where Q is an input variable which can be random chosen.

实验结果

熵的计算

这个任务与之前的任务相同，所以直接Copy过来。得到结果如图所示。

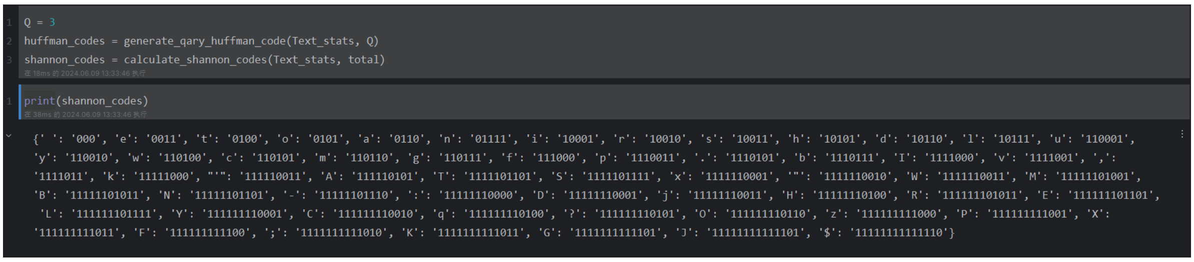


```python
file_path = 'Steve_Jobs_Speech.txt'
with open(file_path, 'r', encoding='utf-8') as f:
    speech_text = f.read()
Text_stats, total = collect_statistics(speech_text)
print(Text_stats)
```

在 20ms 的 2024.06.09 13:06:57 执行

```
Counter({' ': 2228, 'e': 1074, 't': 906, 'o': 768, 'a': 741, 'n': 588, 'i': 526, 'r': 493, 's': 489, 'h': 434, 'd': 408, 'l': 397, 'u': 283, 'y': 252, 'w': 233, 'c': 215, 'm': 205,
'g': 205, 'f': 203, 'p': 179, '.': 142, 'b': 121, 'I': 116, 'v': 115, ',': 101, 'k': 63, "’": 40, 'A': 31, 'T': 22, 'S': 22, 'x': 13, '”': 12, 'W': 12, 'M': 11, 'B': 11, 'N': 9,
'-': 9, ':': 9, 'D': 9, 'j': 8, 'H': 6, 'R': 5, 'E': 5, 'L': 5, 'Y': 5, 'C': 4, 'q': 4, '?': 4, 'O': 4, 'z': 4, 'P': 4, 'X': 3, 'F': 3, ';': 2, 'K': 2, 'G': 2, 'J': 1, '$': 1})
```

Shannon 编码



```
Q = 3
huffman_codes = generate_qary_huffman_code(Text_stats, Q)
shannon_codes = calculate_shannon_codes(Text_stats, total)
```

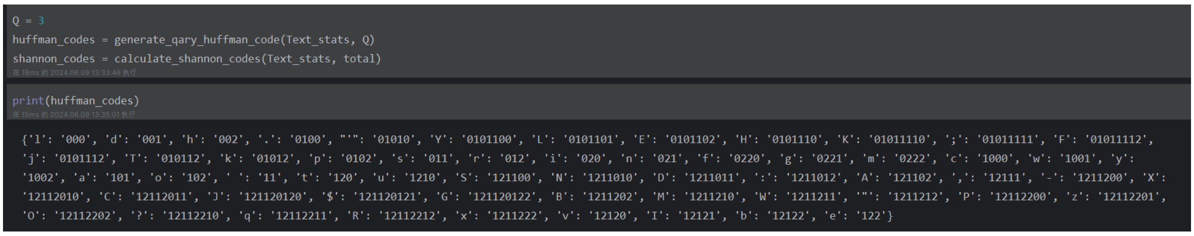
在 18ms 的 2024.06.09 13:33:46 执行

```
print(shannon_codes)
```

在 38ms 的 2024.06.09 13:33:46 执行

```
{' ': '000', 'e': '0011', 't': '0100', 'o': '0101', 'a': '0110', 'n': '01111', 'i': '10001', 'r': '10010', 's': '10011', 'h': '10101', 'd': '10110', 'l': '10111', 'u': '110001',
'y': '110010', 'w': '110100', 'c': '110101', 'm': '110110', 'g': '110111', 'f': '111000', 'p': '1110011', '.': '1110101', 'b': '1110111', 'I': '1111000', 'v': '1111001', ',':
'1111011', 'k': '11111000', "'": '111110011', 'A': '111110101', 'T': '1111101101', 'S': '1111101111', 'x': '1111110001', '"': '1111110010', 'W': '1111110011', 'M': '11111101001',
'B': '11111101011', 'N': '11111101101', '-': '11111101110', ':': '11111110000', 'D': '11111110001', 'j': '11111110011', 'H': '11111110100', 'R': '111111101011', 'E': '111111101101',
 'L': '111111101111', 'Y': '111111110001', 'C': '111111110010', 'q': '111111110100', '?': '111111110101', 'O': '111111110110', 'z': '111111111000', 'P': '111111111001', 'X':
'111111111011', 'F': '111111111100', ';': '1111111111010', 'K': '1111111111011', 'G': '1111111111101', 'J': '11111111111101', '$': '11111111111110'}
```

Huffman 编码



```
Q = 3
huffman_codes = generate_qary_huffman_code(Text_stats, Q)
shannon_codes = calculate_shannon_codes(Text_stats, total)
```

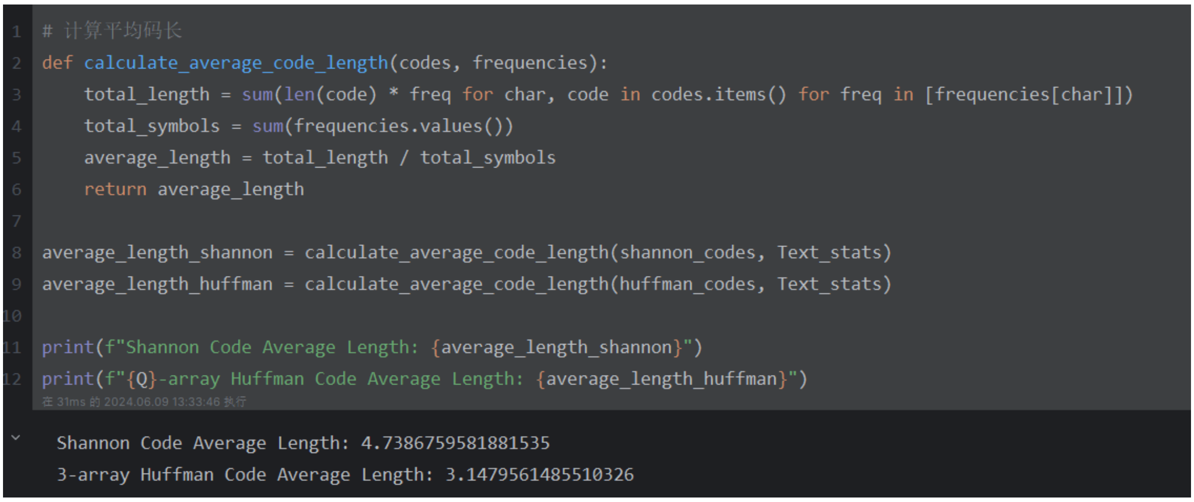
在 18ms 的 2024.06.09 13:33:46 执行

```
print(huffman_codes)
```

在 15ms 的 2024.06.09 13:35:01 执行

```
{'l': '000', 'd': '001', 'h': '002', '.': '0100', "'": '01010', 'Y': '0101100', 'L': '0101101', 'E': '0101102', 'H': '0101110', 'K': '01011110', ';': '01011111', 'F': '01011112',
'j': '0101112', 'T': '010112', 'k': '01012', 'p': '0102', 's': '011', 'r': '012', 'i': '020', 'n': '021', 'f': '0220', 'g': '0221', 'm': '0222', 'c': '1000', 'w': '1001', 'y':
'1002', 'a': '101', 'o': '102', ' ': '11', 't': '120', 'u': '1210', 'S': '121100', 'N': '1211010', 'D': '1211011', ':': '1211012', 'A': '121102', ',': '12111', '-': '1211200', 'X':
'12112010', 'C': '12112011', 'J': '121120120', '$': '121120121', 'G': '121120122', 'B': '1211202', 'M': '1211210', 'W': '1211211', '"': '1211212', 'P': '12112200', 'z': '12112201',
'O': '12112202', '?': '12112210', 'q': '12112211', 'R': '12112212', 'x': '1211222', 'v': '12120', 'I': '12121', 'b': '12122', 'e': '122'}
```

平均码长



```
# 计算平均码长
def calculate_average_code_length(codes, frequencies):
    total_length = sum(len(code) * freq for char, code in codes.items() for freq in [frequencies[char]])
    total_symbols = sum(frequencies.values())
    average_length = total_length / total_symbols
    return average_length

average_length_shannon = calculate_average_code_length(shannon_codes, Text_stats)
average_length_huffman = calculate_average_code_length(huffman_codes, Text_stats)

print(f"Shannon Code Average Length: {average_length_shannon}")
print(f"{Q}-array Huffman Code Average Length: {average_length_huffman}")
```

在 31ms 的 2024.06.09 13:33:46 执行

```
Shannon Code Average Length: 4.7386759581881535
3-array Huffman Code Average Length: 3.1479561485510326
```

结果分析

程序完成了任务要求，可以读取文本后为其生成Q-Huffman编码和Shannon编码。

附：实验源码

```
import math
from collections import Counter, defaultdict
import string
import heapq


```

```python
# 统计字母、标点符号和空格的频率
def collect_statistics(text):
    letters_and_punctuation = string.ascii_letters + string.punctuation + ' '
    filtered_text = [char for char in text if char in letters_and_punctuation]
    char_count = Counter(filtered_text)
    total_chars = sum(char_count.values())
    return char_count, total_chars

file_path = 'Steve_Jobs_Speech.txt'
with open(file_path, 'r', encoding='utf-8') as f:
    speech_text = f.read()
Text_stats, total = collect_statistics(speech_text)
print(Text_stats)


def calculate_entropy(filtered_text, order=0):

    if order == 0:
        # 0阶马尔可夫模型：每个字符独立选择
        char_count = Counter(filtered_text)
        total_chars = sum(char_count.values())
        entropy = 0
        for count in char_count.values():
            probability = count / total_chars
            entropy -= probability * math.log2(probability)
        return entropy
    else:
        # 高阶马尔可夫模型
        storage = defaultdict(Counter)
        total_ngrams = 0

        for i in range(len(filtered_text) - order):
            prefix = filtered_text[i:i + order]
            next_char = filtered_text[i + order]
            storage[prefix][next_char] += 1
            total_ngrams += 1

        entropy = 0
        for prefix, suffix_counts in storage.items():
            prefix_total = sum(suffix_counts.values())
            for count in suffix_counts.values():
                probability = count / prefix_total
                entropy -= (prefix_total / total_ngrams) * probability * math.log2(probability)

        return entropy


Text_entropy_0 = calculate_entropy(speech_text, 0)
Text_entropy_3 = calculate_entropy(speech_text, 3)
Text_entropy_5 = calculate_entropy(speech_text, 5)

print(f"Text Entropy (0th order): {Text_entropy_0}")
print(f"Text Entropy (3rd order): {Text_entropy_3}")
print(f"Text Entropy (5th order): {Text_entropy_5}")


# Shannon编码
def calculate_shannon_codes(frequencies, total_chars):
    codes = {}
    probabilities = {char: freq / total_chars for char, freq in frequencies.items()}
    sorted_chars = sorted(probabilities.items(), key=lambda item: item[1], reverse=True)
    cumulative_prob = 0.0
    for char, prob in sorted_chars:
        code_length = math.ceil(-math.log2(prob))
        cumulative_prob_bin = bin(int(cumulative_prob * (1 << code_length)))[2:].zfill(code_length)
        codes[char] = cumulative_prob_bin[:code_length]
        cumulative_prob += prob
    return codes


# Huffman编码
class QaryHuffmanNode:
    # 定义节点
    def __init__(self, symbol=None, frequency=0):
        self.symbol = symbol
        self.frequency = frequency
        self.children = []

    def __lt__(self, other):
        return self.frequency < other.frequency

def qary_huffman_code(symbols, frequencies, Q):
    # 将所有节点压入堆中
    heap = [QaryHuffmanNode(symbol=symbol, frequency=freq) for symbol, freq in zip(symbols, frequencies)]
    heapq.heapify(heap)

    # 转换最小堆
    while len(heap) > 1:
        children = [heapq.heappop(heap) for _ in range(min(Q, len(heap)))]
        parent_frequency = sum(child.frequency for child in children)
        parent_node = QaryHuffmanNode(frequency=parent_frequency)
        parent_node.children.extend(children)
        heapq.heappush(heap, parent_node)

    # 递归编码
    def build_code(node, prefix, code):
        if node.symbol is not None:
            code[node.symbol] = prefix
        else:
            for i, child in enumerate(node.children):
                build_code(child, prefix + str(i), code)

    root = heap[0]
    code = {}
    build_code(root, "", code)
    return code

def generate_qary_huffman_code(counter, Q):
    symbols = list(counter.keys())
    frequencies = list(counter.values())
    return qary_huffman_code(symbols, frequencies, Q)

Q = 3
huffman_codes = generate_qary_huffman_code(Text_stats, Q)
shannon_codes = calculate_shannon_codes(Text_stats, total)

```

```python
print(huffman_codes)


# 计算平均码长
def calculate_average_code_length(codes, frequencies):
    total_length = sum(len(code) * freq for char, code in codes.items() for freq in [frequencies[char]])
    total_symbols = sum(frequencies.values())
    average_length = total_length / total_symbols
    return average_length

average_length_shannon = calculate_average_code_length(shannon_codes, Text_stats)
average_length_huffman = calculate_average_code_length(huffman_codes, Text_stats)

print(f"Shannon Code Average Length: {average_length_shannon}")
print(f"{Q}-array Huffman Code Average Length: {average_length_huffman}")


# 保存
def save_codes_to_file(filename, codes):
    with open(filename, 'w', encoding='utf-8') as f:
        for char, code in codes.items():
            f.write(f"{char}: {code}\n")

save_codes_to_file('shannon_codes.txt', shannon_codes)
save_codes_to_file('huffman_codes.txt', huffman_codes)
```