Word Frequency using MultiThreading

Nguyen The Duy Khanh

005019181

MSCS-630 - Advanced Operating System

Primus Vekuh

Jan 17, 2025

Table of Contents

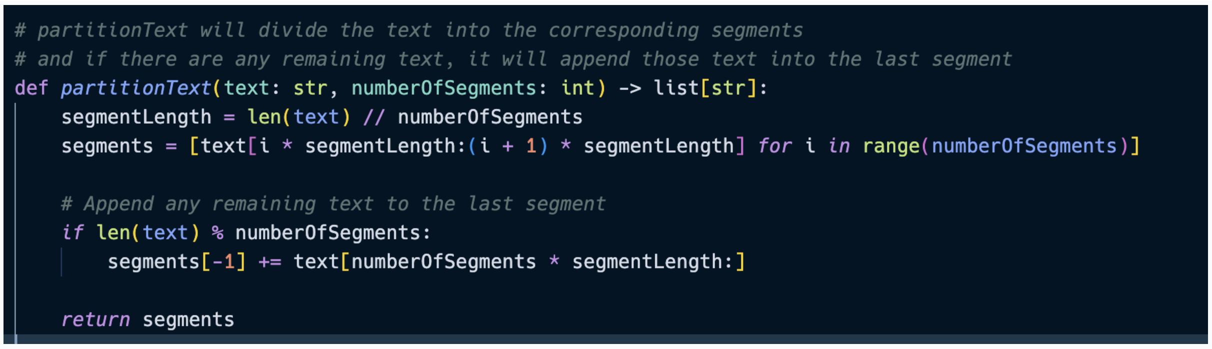
Word Frequency Counter

Core functionality

The core functionalities needs to have the following arguments:

- The file path that contains the text to be segmented
- Number of segment to be divided

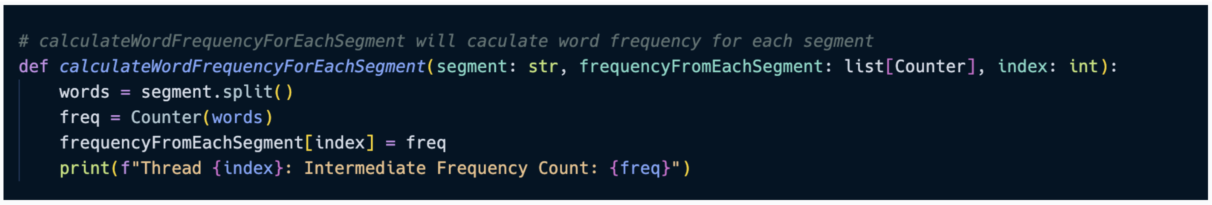
However, before calculating the segment frequency, we would need to divide the text equally to each segment. Therefore, the partition text will handle that. If there are any remaining text, we will add those remaining one to the final segment



```python
# partitionText will divide the text into the corresponding segments
# and if there are any remaining text, it will append those text into the last segment
def partitionText(text: str, numberOfSegments: int) -> list[str]:
    segmentLength = len(text) // numberOfSegments
    segments = [text[i * segmentLength:(i + 1) * segmentLength] for i in range(numberOfSegments)]

    # Append any remaining text to the last segment
    if len(text) % numberOfSegments:
        segments[-1] += text[numberOfSegments * segmentLength:]

    return segments
```

Figure 1: Dividing the text into each segment equally

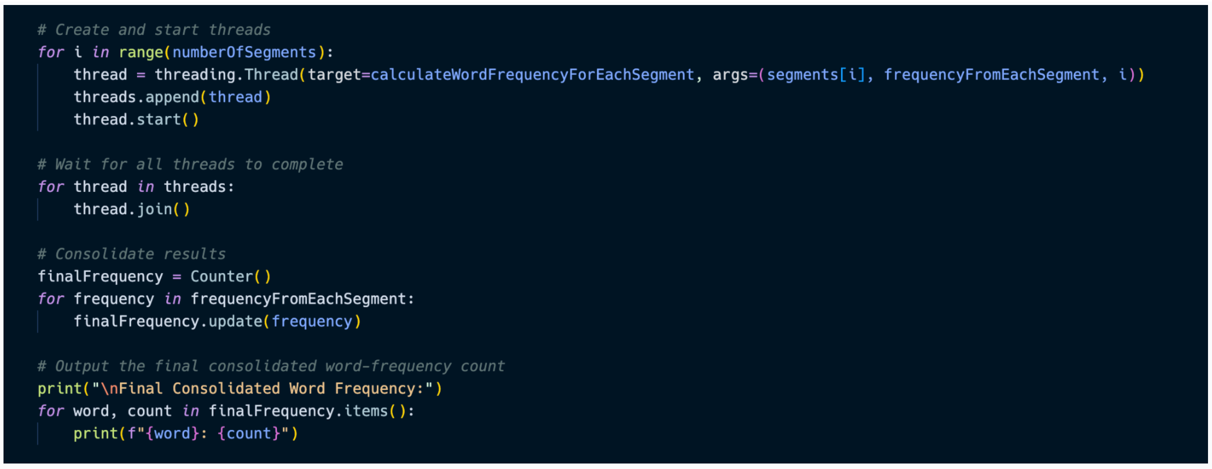
After dividing the text into each segment, we will use the Counter data structures to count the word frequency and record the frequency in the hash map with each thread will handle each segment.



```python
# calculateWordFrequencyForEachSegment will caculate word frequency for each segment
def calculateWordFrequencyForEachSegment(segment: str, frequencyFromEachSegment: list[Counter], index: int):
    words = segment.split()
    freq = Counter(words)
    frequencyFromEachSegment[index] = freq
    print(f"Thread {index}: Intermediate Frequency Count: {freq}")
```

Figure 2: Count the word frequency for each segment

Afterwards, we will consolidated the results by looping each of the segment word's frequency and update the frequency to the final result.

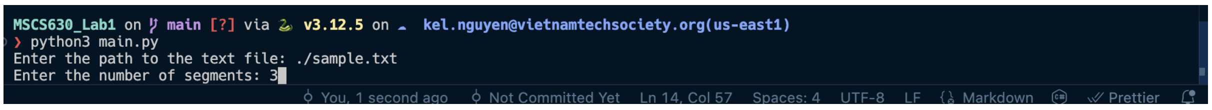


```python
# Create and start threads
for i in range(numberOfSegments):
    thread = threading.Thread(target=calculateWordFrequencyForEachSegment, args=(segments[i], frequencyFromEachSegment, i))
    threads.append(thread)
    thread.start()

# Wait for all threads to complete
for thread in threads:
    thread.join()

# Consolidate results
finalFrequency = Counter()
for frequency in frequencyFromEachSegment:
    finalFrequency.update(frequency)

# Output the final consolidated word-frequency count
print("\nFinal Consolidated Word Frequency:")
for word, count in finalFrequency.items():
    print(f"{word}: {count}")
```

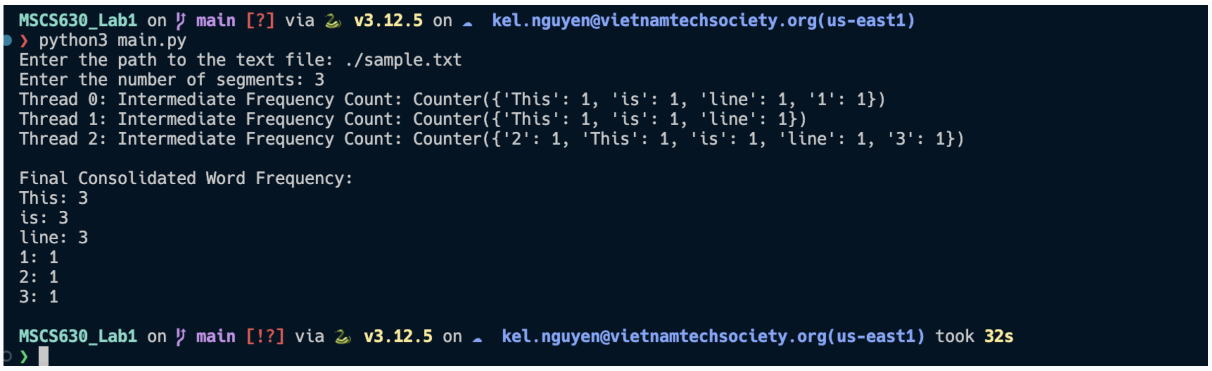
Figure 3: Consolidate each segment word's frequency into one

Final Output

When supplying the final arguments to the main application, the final consolidated word frequency would be:



```
MSCS630_Lab1 on main [?] via v3.12.5 on   kel.nguyen@vietnamtechsociety.org(us-east1)
❯ python3 main.py
Enter the path to the text file: ./sample.txt
Enter the number of segments: 3
```



```
MSCS630_Lab1 on main [?] via v3.12.5 on   kel.nguyen@vietnamtechsociety.org(us-east1)
❯ python3 main.py
Enter the path to the text file: ./sample.txt
Enter the number of segments: 3
Thread 0: Intermediate Frequency Count: Counter({'This': 1, 'is': 1, 'line': 1, '1': 1})
Thread 1: Intermediate Frequency Count: Counter({'This': 1, 'is': 1, 'line': 1})
Thread 2: Intermediate Frequency Count: Counter({'2': 1, 'This': 1, 'is': 1, 'line': 1, '3': 1})

Final Consolidated Word Frequency:
This: 3
is: 3
line: 3
1: 1
2: 1
3: 1

MSCS630_Lab1 on main [!?] via v3.12.5 on   kel.nguyen@vietnamtechsociety.org(us-east1) took 32s
❯
```