Algorithm HW1

60547079S 陳奕寧 [ianchen14@gmail.com](mailto:ianchen14@gmail.com)

Pseudo code：

1.

a = f.open()

regeTokened = Regetokenizer.tokenize(a)

for word in regeTokened:

tokenLen = len(word)

i = 0

while(i != len(regeTokened)):

if(ord(regeTokened[i][0]) not in range(97, 122)):

regeTokened.remove(regeTokened[i])

i = i - 1

i = i + 1

for i in range(0, len(tokenLen)):

j = j + 1

while(j < len(regeTokened)):

if(ord(regeTokened[i][0]) > ord(regeTokened[j][0])):

score = score + 1

j = j + 1

print(time, score)

Input: 文章 txt 檔

Output: 該文章的亂度

# 開啟文字檔

# 使用 NLTK 抓取文字且做 token

# 把抓取到的每個單字轉成數字陣列

# 檢查 list 裡的字是否都是在 ASCII 97~122 間的字元

# 從頭開始訪問陣列，且分別把每一個對之後的索引計算亂度，

# 印出結果與時間

2.

f.open()

regeTokened = Regetokenizer.tokenize(a)

for i in range(0, len(regetokened)):

regetokened[i] = regetokened[i].lower()

i = 0

while(i != len(regeTokened)):

if(ord(regeTokened[i][0]) not in range(97, 122)):

regeTokened.remove(regeTokened[i])

i = i - 1

i = i + 1

for i in range(0, len(regetokened)):

if(regeTokened[i] not in result):

result[regeTokened[i]] = 1

else:

result[regeTokened[i]] += 1

for key, value in result.items():

if(value > biggest):

biggest = value

while(checknum >= 1):

print(checknum, ':', end=' ')

for key, value in result.items():

if(value == checknum):

print(key, end=' ')

print()

checknum -= 1

print(time.time() - start)

Input: 文章 txt 檔

Output: 該文章其使用的字，且印出其分佈的結果

# 開啟文字檔

# 使用 NLTK 抓取文字且做 token

# 轉換成 lowercase

# 檢查 list 裡的字是否都是在 ASCII 97~122 間的字元

# 統計出現的單字與次數且放到 dictionary result

# 找出現最多次數的單字，方便以後做統整

# 循序檢查 dictionary 中每個單字的 value 是否一樣然後印出該單字

# 印出程式執行時間

Time Complexity：

1.

假設整篇文章有 n 個字，分開文章的單字大概花 O(n) 個時間。

regeTokened = Regetokenizer.tokenize(a) ---> O(n)

因為後來說不分大小寫，所以再花 O(n) 的時間把剛剛分完的字詞全部轉成小寫。

for i in range(0, len(regeTokened)):

regeTokened[i] = regeTokened[i].lower()

----> O(n)

檢查 list 裡的字是否都是在 ASCII 97~122 間的字元，又再檢視了一次 list，花了 O(n)。

i = 0

while(i != len(regeTokened)):

if(ord(regeTokened[i][0]) not in range(97, 122)):

regeTokened.remove(regeTokened[i])

i = i - 1

i = i + 1 ---->O(n)

主要計算文章亂度的地方，使用一個迴圈巡迴整個陣列，裡面包含一個 while 判斷 for 跑到得值後面剩下的其字母順序是否大於目前的值，是就在 score 上加一。

for i in range(0, len(regeTokened)):

j = i + 1

print(ord(regeTokened[i][0]))

while(j < len(regeTokened)):

if(ord(regeTokened[i][0]) > ord(regeTokened[j][0])):

score = score + 1

j = j + 1

----> O(n^2)

所以總共的 time complexity 是 f(n + n + n + n^2) 是 O(n^2)。

2.

跟第一題一樣先處理字詞分割，花了 O(n) 的時間。

regeTokened = Regetokenizer.tokenize(a) ---> O(n)

與轉換大小寫，一樣花了 O(n) 的時間。

for i in range(0, len(regeTokened)):

regeTokened[i] = regeTokened[i].lower()

----> O(n)

跟第一題一樣檢查一次 list 找出不該出現的字元。

i = 0

while(i != len(regeTokened)):

if(ord(regeTokened[i][0]) not in range(97, 122)):

regeTokened.remove(regeTokened[i])

i = i - 1

i = i + 1 ---->O(n)

方便程式執行，我先把陣列轉換成 dictionary 且同時計算出現的字詞與其次數，像 apple: 10, to: 210, …

for i in range(0, len(regetokened)):

if(regeTokened[i] not in result):

result[regeTokened[i]] = 1

else:

result[regeTokened[i]] += 1

----> O(n)

一樣方便之後程式執行，我先跑過一次 dictionary 找出出現最多次的次數。

for key, value in result.items():

if(value > biggest):

biggest = value

----> O(n)

再依照出現的順序印出是該次數的詞，這邊的時間複雜度我覺得很難說，因為最壞的情況應該是有一個詞出現次數最多，我認為應該是 O(n^2)，但應該也不是很緊湊的。while 從剛剛找出的最大次數開始遞減， for 跑整個迴圈。

while(checknum >= 1):

print(checknum, ':', end=' ')

for key, value in result.items():

if(value == checknum):

print(key, end=' ')

print()

checknum -= 1

-----> O(n^2)

所以總共的時間複雜度是 f(n + n + n + n + n + n^2) 是 O(n^2)。

但在最後的這段程式我認為 n^2 真的不是緊湊的，因為我把字都放到 dictionary 裡，試想最壞的情況每個字都各自佔一個位置，但這對於文章應該是很難發生的。

Space Complixity：

1.

因為只有使用一個陣列，所以只用了 n 個 String 的單位大小，O(n)。

2.

也是先使用了一個陣列 n 個單位大小，但額外使用了一個 dictionary ，dictionary 的大小比較難說但最大應該可以預期是 n 個單位大小，所以推測最後空間複雜度應該是 O(n)。

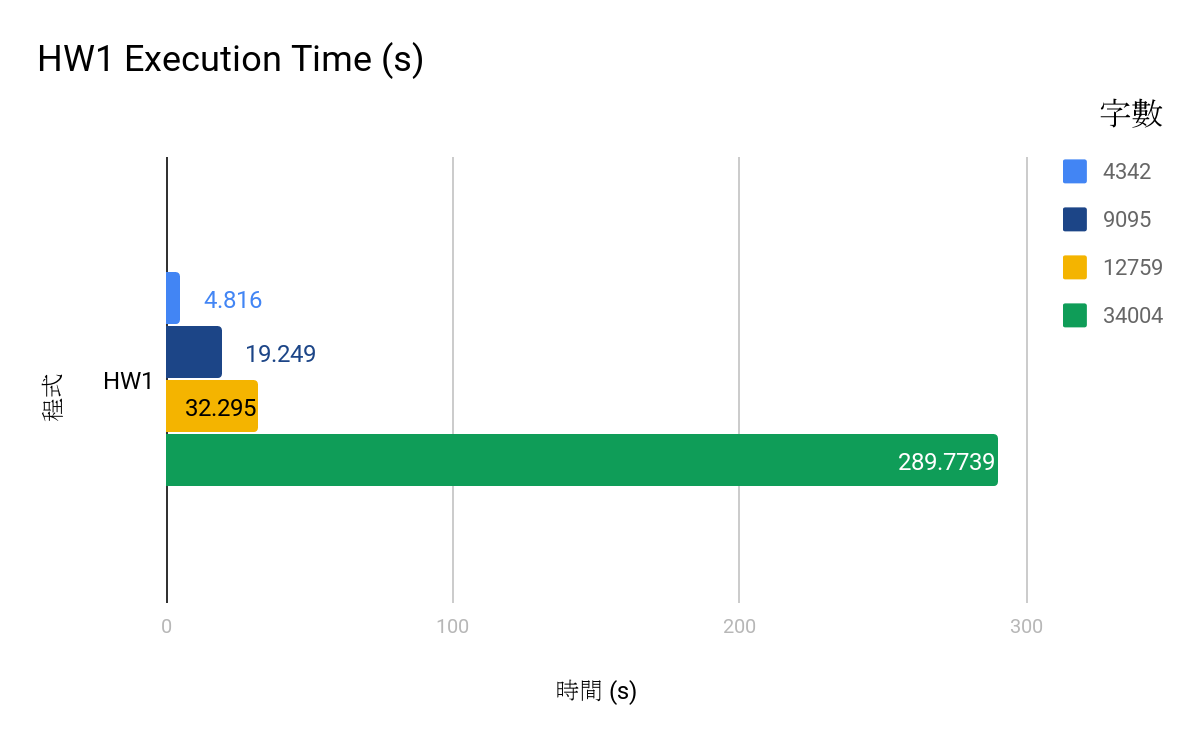
實驗：

電腦設備：



程式語言： Python 3.5.2

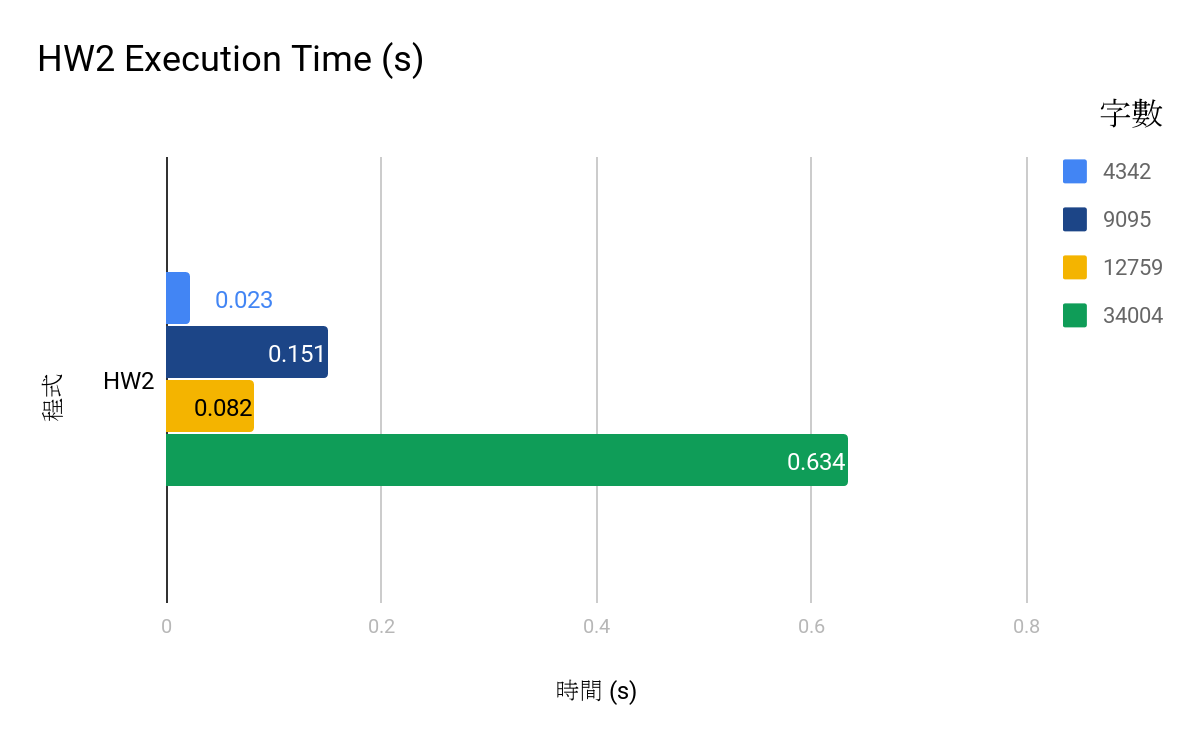
1.

分別用範例文章 4342 、 9095、12759 、34004 字來做實驗。  


我的觀察是時間大概就是符合上述評估的時間複雜度 O(n^2) 的成長，且因為不像程式二可能因為文章內使用字的問題導致 9000 字的與 12000 字的時間更長(也有跟兩個任務類型有關)。

2.

分別用範例文章 4342 、 9095、12759 、34004 字來做實驗。



不知道別人的時間如何，但程式二的速度還蠻快的，可能是因為我利用 dictionary 的特性來做所以速度很快，但觀察實驗發現， 9000 字的比 12000 字的時間要長，大致看了一下兩個出來的結果，可能是因為 9000 字在文字上的使用(比較分散)，比 12000 字的使用更多，所以可能導致這樣的結果。