* 開發平台

Window 10

* 使用開發環境

VScode

* 使用的程式語言

Python

* 功能

讀入一個文字檔，裡面有使用者隨機產生的數字 範圍有0~1w, 10w,50w,100w 程式根據使用者所選擇的方法, 將隨機亂數的值 依序排列

並且輸出CPU執行的時間以及完成方法時當下的時間(時區為UTC+8 )

流程

讀取檔案完成後，先根據檔案內容的第一個元素來判斷欲採用何種方法。

* 方法一，將N個數目字直接進行Bubble Sort，並顯示CPU執行之時間。
* 方法二，將N個數目字切成k份，由k個threads分別進行Bubble Sort後，再由k-1個threads進行Merge Sort，並顯示CPU執行之時間。
* 方法三，將N個數目字切成k份，並由k個Processes分別進行Bubble Sort之後，再由k-1個Processes進行Merge Sort，並顯示CPU執行之時間。
* 方法四，將N個數目字切成k份，在一個Process內對k份資料進行Bubble Sort，再用同一個Process作Merge Sort，同時顯示CPU執行之時間。
  + 實作方法
  + 創建一個list()用來儲存需要處理的資料，message 為使用者介面，
* if \_\_name\_\_ == "\_\_main\_\_":
* message()
* data = list()  # 建立一個串列
* case = ""
* filename = input("輸入檔名:") + ".txt"
* #    filename = "input1\_2.txt"
* case = readfile(data, case, filename)

讀檔方式以換行做為切割檔案的依據並且輸出資料的長度

* def readfile(data, case, filename):
* case = input("請輸入要哪個case: ")
* f = open(filename, "r")
* for line in f.readlines():
* line = line.strip("\n")
* data.append(int(line))
* print(data.\_\_len\_\_())

在每個任務前, 都先做檔案分割處理 K為使用者想要切割的份數 , K\_thd 每份資料裡面包含多少的資料個數, dividelist 為 儲存資料之陣列 下方的if 用來處理是否有餘數 有餘數就將餘數的部分加到最後一筆資料進行處理

    K\_thd = int(len(data) / K)

    dividelist = [data[i : i + K\_thd] for i in range(0, K\_thd \* K, K\_thd)]

    if len(data) % K != 0:  # 是否有餘數

        Lastlist += data[K\_thd \* K + 1 : len(data)]

        dividelist[K - 1] += Lastlist

執行緒 :

Threads 為將剛剛dividelist切個的資料進行 K份bubblesort 的處理 將處理完的結果放置queue裡頭 在對K-1份資料進行 mergesort 之處理 處理方式為一次丟兩筆佇列資料進行mergesort 處理完之結果放回佇列最後 依此類推 最後得到最後一筆結果

    for i in range(K):

        threads.append(td.Thread(target=mission2\_bubble, args=(dividelist[i], q)))

        threads[i].start()

    for i in range(K):

        threads[i].join()

多處理程序:

使用到multiprocessing 下的 Pool進行多處理程序之運作

pool.apply\_async()用來向程序池提交目標請求，pool.join()是用來等待程序池中的worker程序執行完畢，防止主程序在worker程序結束前結束。但必pool.join()必須使用在pool.close()或者pool.terminate()之後

利用multiprocessing下的Pool可以很方便的同時自動處理幾百或者上千個並行操作，指令碼的複雜性也大大降低。

其餘方式與做執行緒大同小異

    manager = mp.Manager()

    q = manager.Queue(K)

    for i in range(K):

        processes = pool.apply\_async(mission3\_bubble, args=(dividelist[i], q))

        # processes.start()

        m\_processes.append(processes)

。

* 分析

由下列圖表可以得知:

我們切割檔案的份數都是以 1000份去做切割

第一題的執行時間成長幅度最高，因為第一題是把所有資料用一個Bubble Sort排序並且沒有用到分割，為效率最差的一種方法。

第二題是將資料切成k份，由k個thread對k份資料排序，再由k-1個thread將k份資料Merge。而第三題一樣會把資料切成k份，由k個process進行排序，再由k-1個process將資料Merge。第二三題差在一個使用執行緒一個使用處理程序 可以看到threads的1w筆與10w筆的執行效率是比process還要來的好的 但是隨者資料的數量越多 threads的執行效率遠高於process 而process 則發揮出cpu的效率 因為threads 是共用同一個地址空間且更多的執行緒需要更多的記憶體空間若有大量的執行緒，就會影響其效能，因為 OS 需要在它們之間做環境切換 而 process 因為有自己的各自的記憶體空間提高 CPU 的使用效率所以我們但不是process越多執行效率就越好

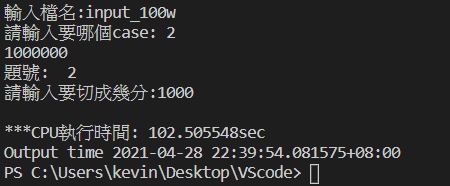
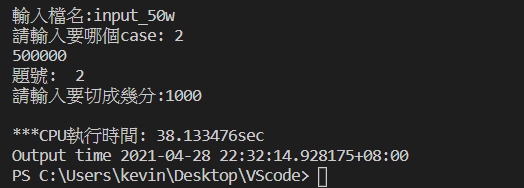
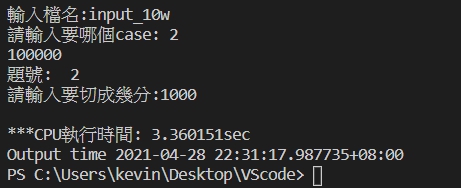
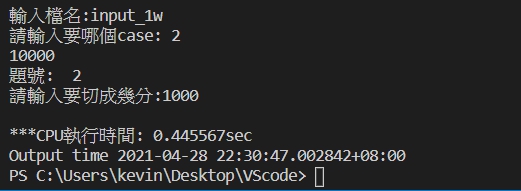
而第四題也會將資料切成k份，但是只用了一個process對k份資料排序，而main 本身就算是一個process 並用相同的process將資料Merge，效率跟第二題與第三題相比,較第三題要差但略勝第二題，但依成長幅度，其效率還是遠大於第一題的方法。

我有試者將切割的數量變為50 份 我發現在 切割50份時 thread 與process 在與1000份比較時沒有明顯的差別但在50w 100w比資料時發現其速度 竟然差了9倍甚至10倍, 我認為還是要根據檔案的多寡來決定 我們使用的threads, process 避免造成不必要的資源浪費 或者導致 執行效率變差 由此證明，將大問題分解成小問題，當問題越小執行速度會呈現正相關。當前BubbleSort時間複雜度約為n2更能看出其效果。

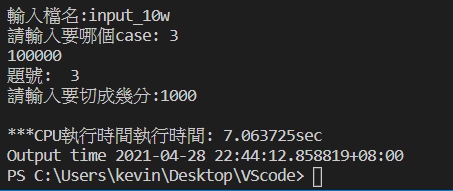
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 一萬筆 | 十萬筆 | 五十萬筆 | 一百萬筆 |
| 第一題 | 8.29 | 918.54 | 24119.67 | 95019.17 |
| 第二題 | 0.445 | 3.36 | 38.13 | 102.50 |
| 第三題 | 6.63 | 7.06 | 11.02 | 30.38 |
| 第四題 | 0.58 | 1.84 | 23.30 | 88.28 |

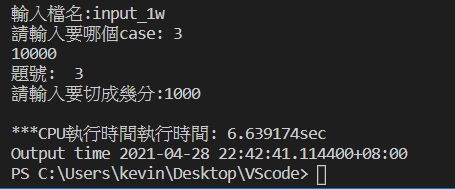
單位：秒

任務2

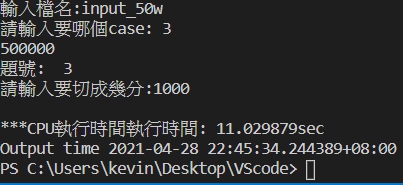


任務三









任務四

