學號:F74074122

姓名:歐禮寬

系級:資訊111

開發環境:

OS: Windows Subsystem for Linux Ubuntu 18.04.1

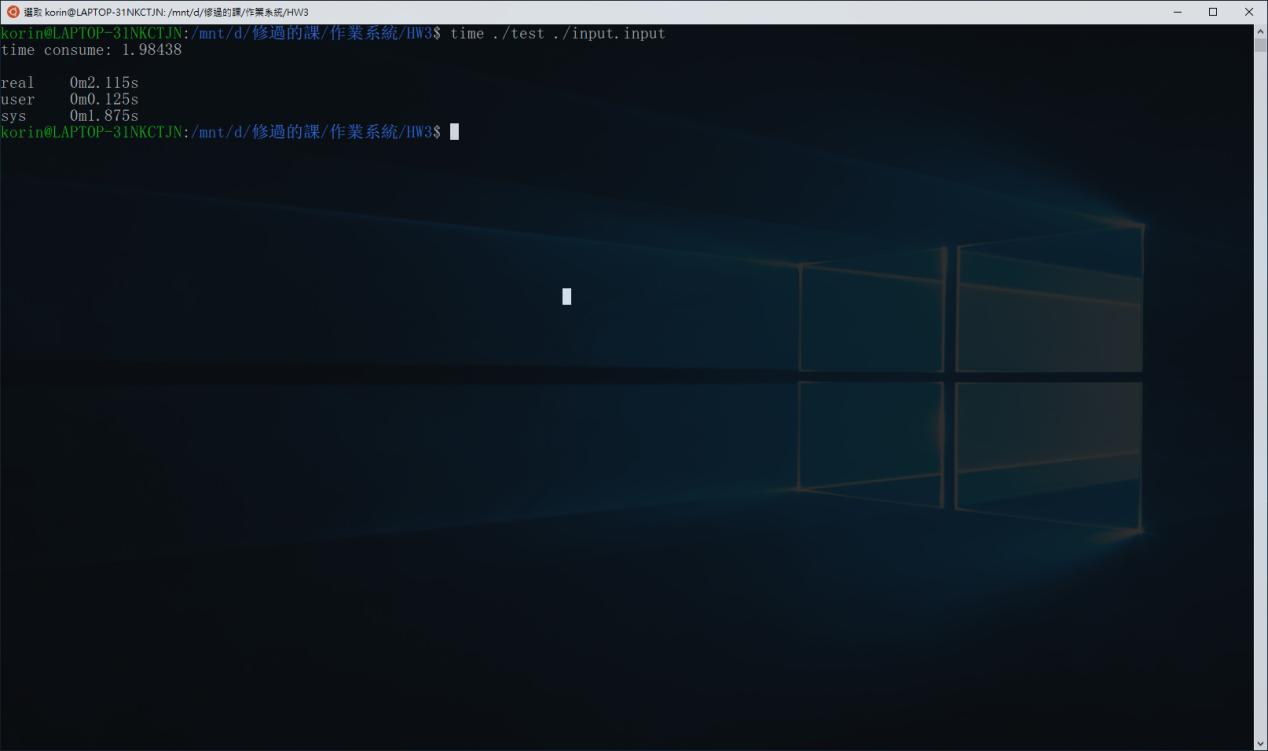
CPU: Intel® Core™ i5-8250U CPU @ 1.60GHz

Memory: 8GB

Programming Language: C++ gcc version 7.4.0

程式執行時間:

跑助教給的測資約2秒



程式開發與使用說明:

程式會依序執行每行指令(PUT、GET、SCAN)。PUT指令會檢查memory當中有沒有相同的key，有的話就更新value，沒有的話將key-value存進第一塊memory(struct keyvalue)，當memory到一數值時，就將資料存進disk(檔案)，並排序檔案內的資料，使得後續對disk資料操作能更有效率；GET指令會先去memory找資料，沒有的話再去disk找資料，找到的資料我會存在第二塊memory，後續要是要用到就可以直接從這裡拿，不須讀檔；SCAN指令一樣先去memory資料，沒有的話再去disk找資料，找到的資料存在第三塊memeory，因為SCAN找的資料都是連續的，而我的檔案都排序過，所以讀出來的資料也都是連續的，所以SCAN的後幾筆資料都只要一直找第三塊memory的下一筆資料。

執行完指令後，程式會將memory尚未寫入disk的資料寫進去，好讓下一次執行能取得上一次的資料在結束。

編譯: g++ keyvaluestore.cpp

執行: ./a.out ./1.input

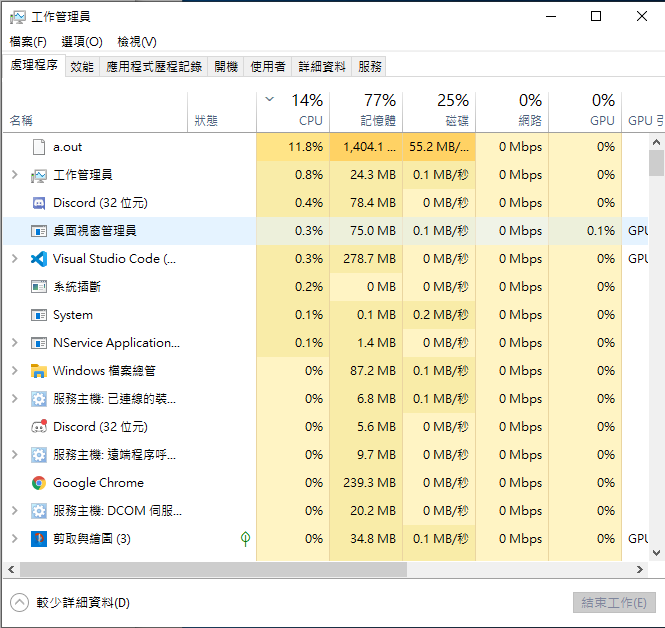
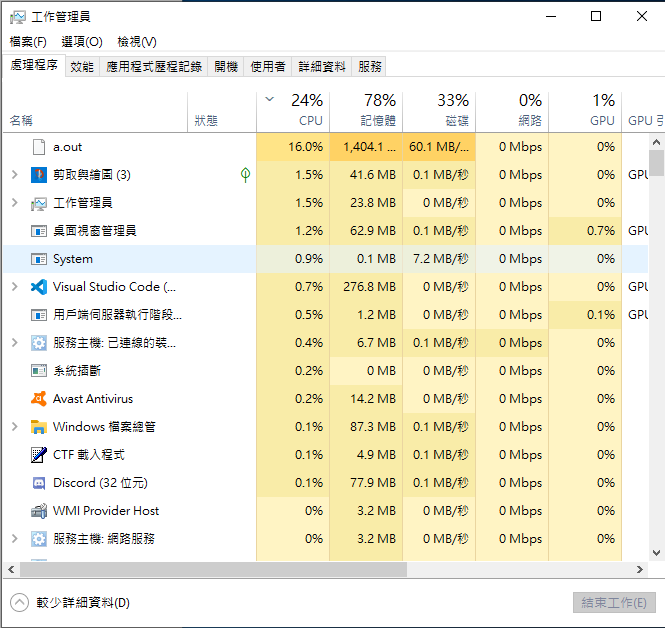
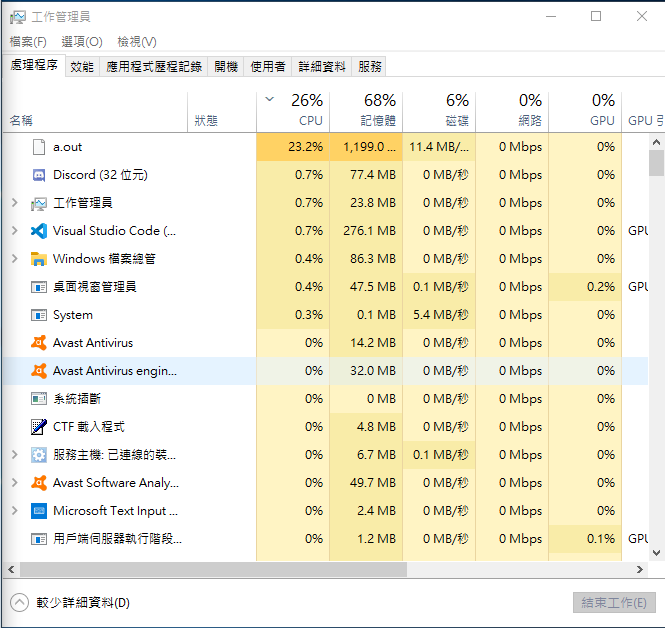
效能分析報告:

整理資料作法及分析:

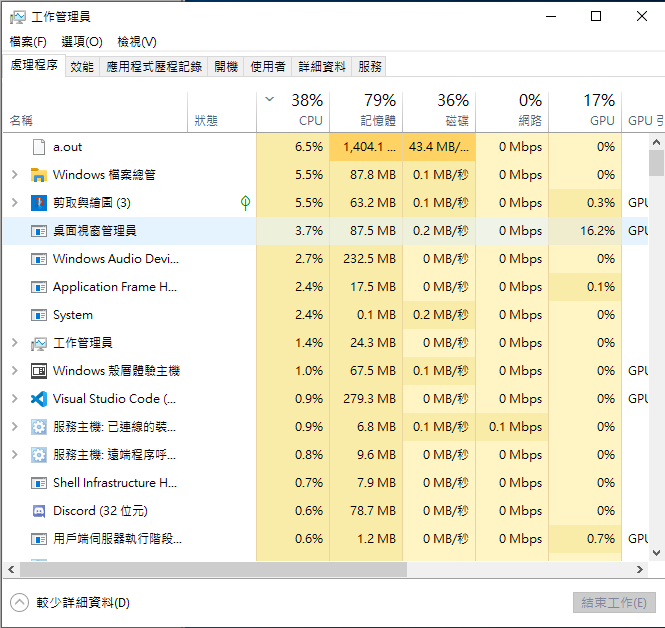
我整理資料的做法是依據數字的最高位數(0~9)，各有一塊相對應的memory空間去存取，並用C++的map輔助以增加更新某個key的速度，而存入disk前，我會用HW1用過的external sort對資料做排序，再存入disk，雖然會增加時間，卻能對後續資料的取得有幫助，像是能使用binary search快速的找出某一筆key；再來就是排序後的資料會有連續的特性，而SCAN就是GET很多連續的資料，這時我只要找到第一筆資料，之後的資料就只要看我找到的那一筆資料的下面幾筆就好了。

程式執行下:

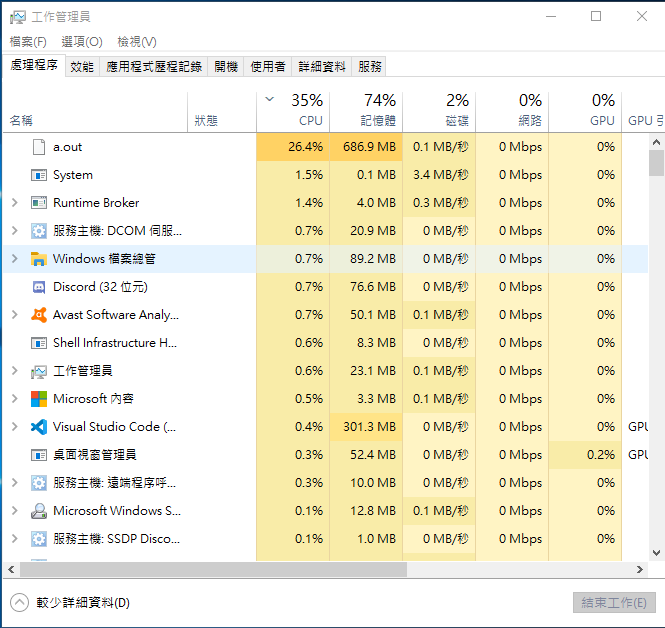
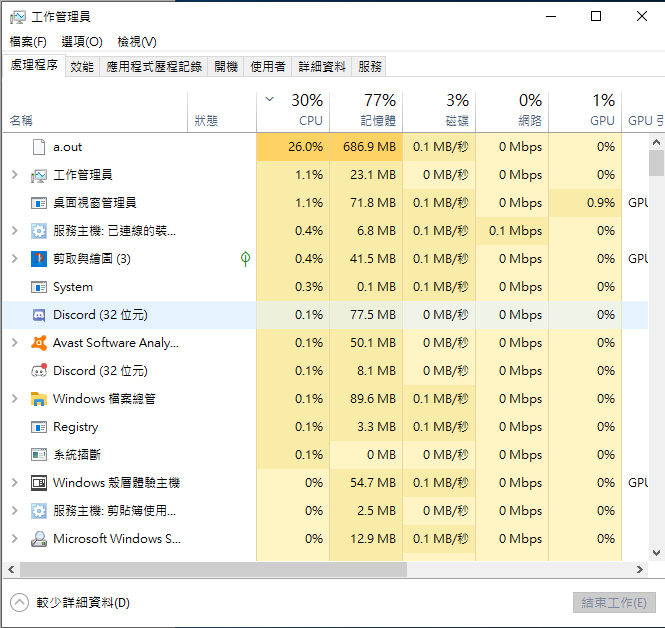
以下4張圖為執行PUT的情況，CPU會忽高忽低我覺得應該是要寫入disk時，CPU會idle的原因，而磁碟使用料也是忽高忽低，因為只有memory到一定量才會去寫檔，而從記憶體也可以看出，程式一開始melloc的記憶體好像要等到用到時，作業系統才會真的給我用。



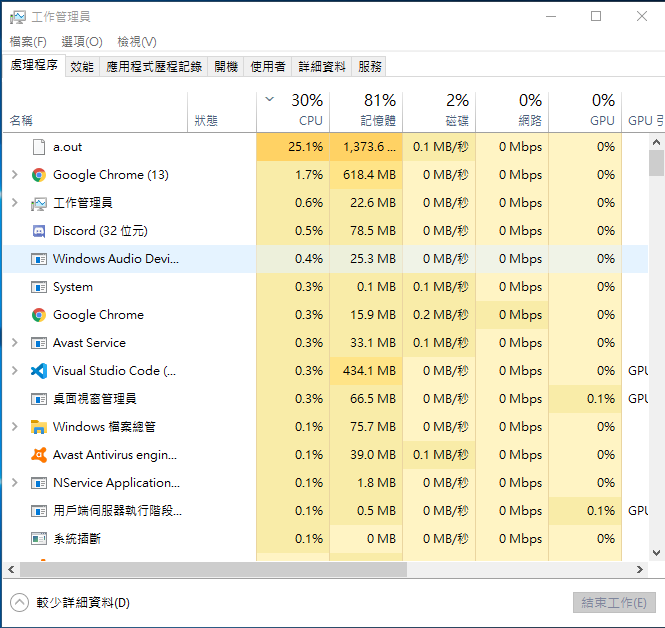
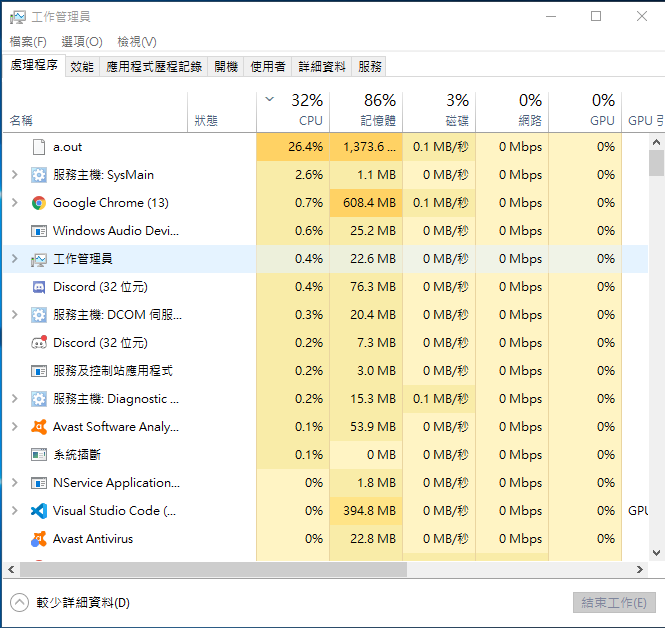
‘



以下兩張圖為GET的情況，CPU使用量很高，磁碟使用量很低，因為我GET是用binary search 整的檔案讀出一筆資料，所以我覺得正好符合這種情況。



以下兩張圖為SCAN的情況CPU使用率一樣很高，而我原本以為磁碟使用率應該會很，因為我SCAN的時候是寫成讓他讀disk裡的一段資料而非GET的一筆，但從圖中可以看出其實並沒有很高。



改進程式的過程:

PUT資料到memory時，如果key存在要跟新資料，而如判斷key存不存在，我是用遍歷的方式判斷，發隨著memory的增大，速度會下降許多，所以後來新增一個C++的map來記錄key是否存在，藉此提升查詢的速度。

GET資料時，如果資料不再memory而要去找disk，我最一開始是從檔案開頭一直讀一段很大的資料到記憶體，再用資料的最小值及最大值判斷key是否存在，存在的話再用binary search的方式找key，但當檔案很大的時且資料又在檔案後半部，最非常的耗時間，因為要一直讀取，所以後來改成對disk做binary search，所以每次就只要讀檔案中的一筆資料且每次都能保證找完檔案的一半，這樣就能減少讀取花費的時間及找尋的次數。

SCAN資料時我最一開始是就直接當作多次GET，然後每次去找memory及disk，但因為我檔案裡的key都排序過，所以當我找到SCAN的第一筆資料，且將他後面的一段資料也都讀出來，下次就不用再讀一次檔案，只要記住上一筆資料的位置，然後判斷他的下一筆資料是否是要找的key值就好了。