基本資訊
學號:E94074029
姓名:江羿賢
系級:資訊系 111 級

開發環境:
OS: Ubuntu 20.04 (虛擬機)
CPU: Intel® Core™ i7-8750H CPU @ 2.20GHz (6CPU)
Memory: 4GB
Programming Language: C++ 11

程式執行時間:
執行 4.3GB 的 PUT 共 921 秒
```
It costs : 921 secs
```
執行 1000 萬筆的 GET 共 3088 秒
```
It costs : 3088 secs
```

程式開發與使用說明:
這個程式一開始會先建立好 buffer，如果是以 4GB 的 memory 來說的話，會以輸入以及輸出 700 萬次作為輸入的 buffer 以及輸出的 buffer 的上限容量，以及將檔案用取餘數的方式來分檔案存放，一共分成 70 萬個檔案，接著來講一些比較主要的結構，首先分成 70 萬個檔的檔名利用一個 vector 來放，而 input_buffer 是利用 map 的 array 來存的，而該 array 的 index 就是應該進去哪個 temp 檔的意思，利用取餘數來算說進來的資料要放入哪個 map，同時也建立一個 iterator 來方便資料提取，而 output 的 buffer 利用一個 vector 來存，並且有兩個 function，一個是 release_input_buffer，一個是 releas_output_buffer，用來等到 buffer 到一定數量的時候 release，避免寫爆 memory，而詳細等最後再來說，接著來看一下程式的架構，一開始先開一個 storage 資料夾以便我們存 temp 檔，而利用 argv 讀檔後，建立起 70 萬個檔名的名稱，之後對 input 檔名作完處理後，也決定了 output 的檔名。之後開始讀檔了，利用 while 迴圈來看第一個字是 PUT GET SCAN。而如果是 PUT 的話，先看有沒有太多的 buffer，如果有就先 release，如果沒有就讀進來 key 跟 value，並且讓 key 去對 70 萬取餘數，算出來後看要放進哪個 map 當中，放入之後把 buffer 的 counter++後就結束了。而如果是 GET 的話一樣先看有沒有太多的 buffer，如果有就先 release，之後就先讀入 key，並且對 70 萬取餘數，看在哪個 map 中，並在那個 map 中找，如果有找到就更新要輸出的 value，如果在裡面沒找到的話就到 temp 裡面找，一樣先看要去哪一個 temp 裡面找，並且讀檔，從頭讀到尾，如果有相同的 key 就更新，再來看要輸出的東西是不是空的，不是的話就輸出裡面的東

西，是的話就輸出 EMPTY。再來 SCAN 也跟 GET 一樣，不一樣的地方在於是讀進要 SCAN 頭尾的值，再利用 for 迴圈去做 GET 在做的事情就好了。結束之後就把檔關掉，並且 release 掉兩個 buffer，而如果 output_buffer 是空的話就不做 release 了，這就是程式的結構。再來講剩下的兩個 function，release_input_buffer 的 function 是利用從 0 讀到 70 萬來全部 release，如果裡面是空的就不用 release，如果是空的就遍歷該編號的 map 並且放進同樣標號的 temp 檔當中寫入，放完之後就把全部 clear 掉，並把 input_buffer_counter 清空成 0。而 release_output_buffer 的部分比較容易，就從頭把值 output 出去檔案就好了，結束後就把全部 clear 掉。

使用說明的部分，基本上就是先編譯，再執行，並且在執行時的後面要加入檔案的參數

Ex:

g++ E94074029.cpp -o E94074029

./E94074029 /home/richard/test_file/1.input

或者如下圖

```
g++ E94074029.cpp -o E94074029
./E94074029 /home/richard/test_file/1.input
```

分析報告:

先來提供一些數據

在 4.3GB 的 PUT 中共花了 921 秒

```
It costs : 921 secs
```

在 4.3GB 的 PUT 讀入 buffer 中的狀態

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
15471 richard   20   0  804224 790728   3184 R  99.7  19.6   0:05.16 E94074029
```

```
/dev/sda:
 Timing buffered disk reads: 366 MB in  3.01 seconds = 121.40 MB/sec
```

在 4.3GB 的 PUT 寫入 storage 的狀態

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
15471 richard   20   0 1515308   1.4g   3184 D  26.7  37.3   1:08.33 E94074029
```

```
/dev/sda:
 Timing buffered disk reads: 298 MB in  3.01 seconds =  99.15 MB/sec
```

在 1000 萬筆 GET 中共花了 3088 秒

```
It costs : 3088 secs
```

在 1000 萬筆 GET 讀入 buffer 中的狀態

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
15567 richard   20   0  111608  97540   3264 D  24.3   2.4   0:08.71 E94074029
```

```
/dev/sda:
 Timing buffered disk reads: 248 MB in  3.02 seconds =  82.05 MB/sec
```

在 1000 萬筆 GET 寫入 output 中的狀態

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
15567 richard   20   0 1230544   1.1g   3264 R  97.0  29.1   8:00.50 E94074029
```

```
/dev/sda:
 Timing buffered disk reads: 240 MB in  3.01 seconds =  79.62 MB/sec
```

先來看一下 PUT 的數據，因為在讀入 buffer 時，可以從同一個檔案按照順序直接讀進來所以這邊會有很高的 99%CPU 使用率以及 121MB/sec 的 I/O 速度，而這邊 MEM 的部分還沒寫到滿所以只有 20%，而在寫入 storage 時因為要開檔關檔所以會造成 CPU 要等 I/O 做好，所以只剩下 27%的 CPU 使用率，而這邊 I/O 的速度要稍微等一下開關檔所以也會降低一點變成 99.15MB/sec，而 MEM 因為已經寫很多了所以到了 37%，而這邊因為為了要減輕 PUT 找資料的遍歷的負擔，所以這邊分了 70 萬個檔案，會造成較多次的開關檔，所以才會需要 921 秒。接著來看 GET 的數據，在讀入 buffer 時，因為要從 temp 檔中找尋資料，而因為需要遍歷，也需要開關檔，所以要耗費非常久的時間在 I/O 上，所以這邊的 CPU 執行只有 24%，MEM 這邊也還剛開始寫而已只有 2.4%，而 I/O 的部分因為在讀取，所以有 82.05MB/sec，等到寫滿或是程式將要執行完畢時，會從 vector 中 output 出去，而這時候因為只開一個檔並按照順序寫入，所以這邊的 CPU 使用率達到了 97%，MEM 也寫得很滿到了 29%，而因為這邊也需要寫入，所以也有約 80MB/sec 的速度。這邊作業系統在對於檔案的開寫的等待上應該是有幫我們做事情的，在等待 I/O 時候，CPU 並不一定會降低到某種非常低的程度而是繼續保持運作，有可能有幫我們做一些 forward 之類等操作，不會讓 CPU 整個空轉在那裡，而是會先幫我們做某些事情，而在 MEM 的部分應該也有做一些保護，所以在寫的時候不能夠算的太剛好，否則可能會造成程式不能夠繼續執行下去，這也是保護系統的一個機制。

最佳化程式的過程

一開始看到時，我對 input 跟 output 都只是利用單一檔案下去存，一開始我沒有使用 buffer 單單只使用了一個檔案，所以在每次 GET 時都必定要開檔，而 PUT 也是直接放進檔案的最尾端，所以 GET 時每次都要遍歷，所以要很久。

第二個版本是我對 temp 檔利用取餘數做出拆解，這樣子雖然 PUT 會慢一些，但是可以增快 GET，不過總結來說還是太慢了。

第三個版本是我對 input 跟 output 都增加一個 vector 的 buffer，可以讓 PUT 進來的可以先存著，以及 output 也可以先存著，讓如果還在 buffer 中的 input 可以直接 get 到 output 中，減少開關檔的次數。但還是要花費很多時間。

第四個版本是改用一個 map 來做為我 input 的 buffer，因為這個是利用紅黑樹

的結構下去做的容器，所以 search 很快，不需要再遍歷整個 vector 來找到所需 get 的 key，速度增加了滿多的。
最後使用的程式結構:
建立一個 map 的 array 作為 input_buffer，共有 70 萬個，以 key 取餘數來做為要存在哪一個的依據，而 output_buffer 繼續使用 vector，而在 PUT 時可以快速的找到哪個 map 並且插入，讓每個 map 都不會有過多的東西，而在要 GET 的 key 如果還在 input_buffer 中的話，也可以很快的找到值，而如果在 input_buffer 中沒有的話就要進入 temp 檔找，不過因為切了很多 temp 檔，所以相對的每個 temp 檔遍歷非常快，而 output 一樣就是放進 vector 中，這個吐出值到.output 檔中已經超級快了，所以也不需要做調整，而這樣分起來最大的進步在於，在 map 中有依據取餘數來分的話，在要 release 寫入 storage 時會快很多，因為本來是遍歷 map 並且再做取餘數看要放進哪個檔案中，要開關檔非常多次，而最後的版本因為已經分好了，所以可以利用看取餘數的 0~70 萬來做，並且裡面加條件如果是 empty 就不做了，這樣可以非常大幅減少了開關檔的時間，最多也只需要開關檔 70 萬次，比之前是有幾筆要開幾次還要好的多，而 PUT 的執行速度也比上一個版本快了 6~8 倍，非常之快，而在這樣分的情況下，GET 也有略快一些。