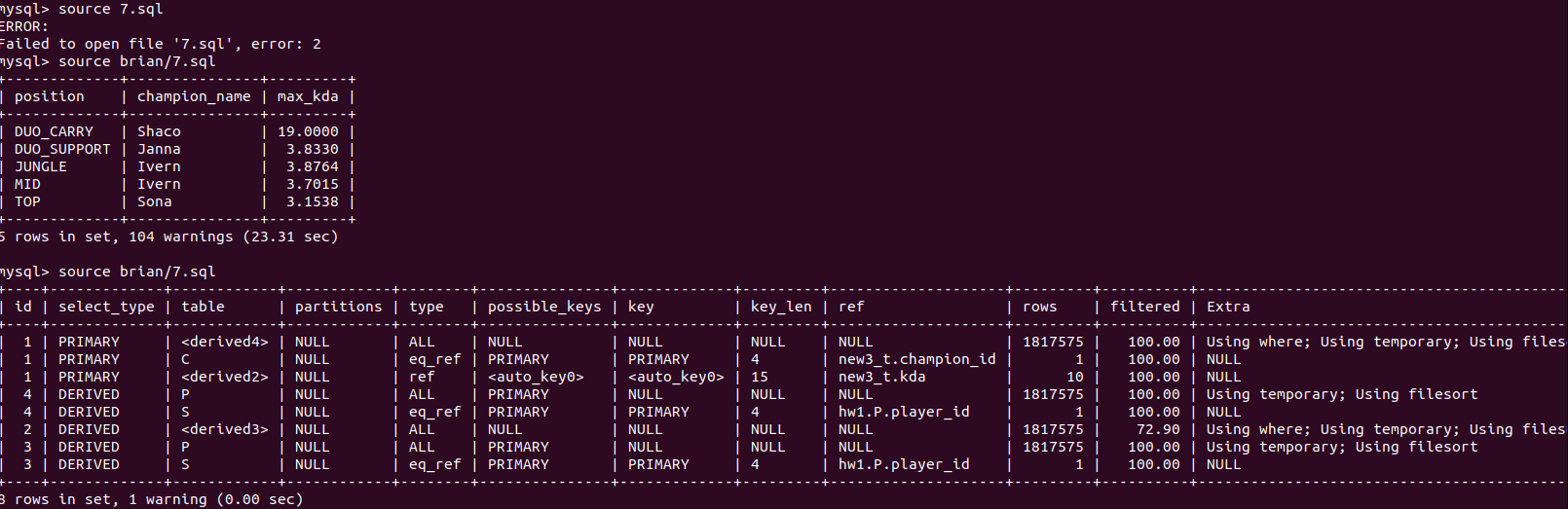
Database HW3 report

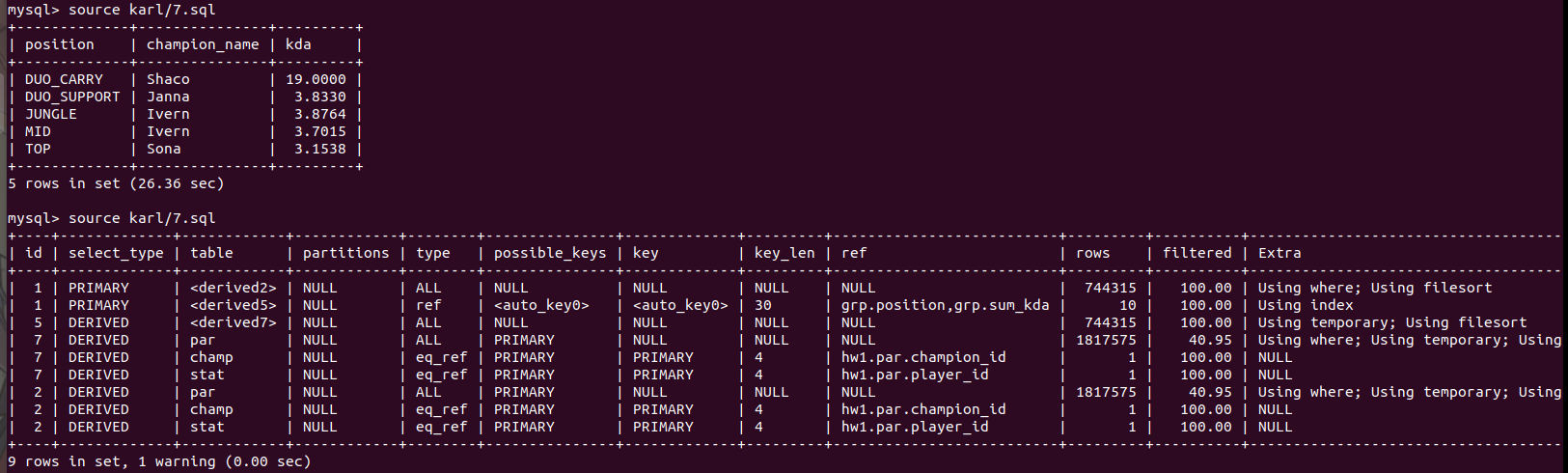
2.1.1

Reference : <https://segmentfault.com/a/1190000008131735>

我的 7.sql



Karl 7.sql



Yao 7.sql



我的跟karl助教的explain差異不大 但我只用8個select而karl助教用9 select。

Yao助教則是只用了4個select完成。

Type性能比較:ALL < index < range ~ index\_merge < ref < eq\_ref < const < system

從上面可以知道，用越多ALL type會越慢。

並且在extra用到Using temporary,using filesort也很耗時

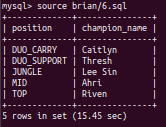
還有如果filter比例越低 這個select規模越小越快。

以上決定了yao的query效率較高。

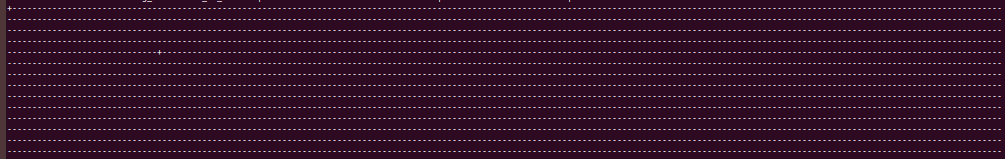
2.1.2

我的optimizer\_trace:

首先先看時間

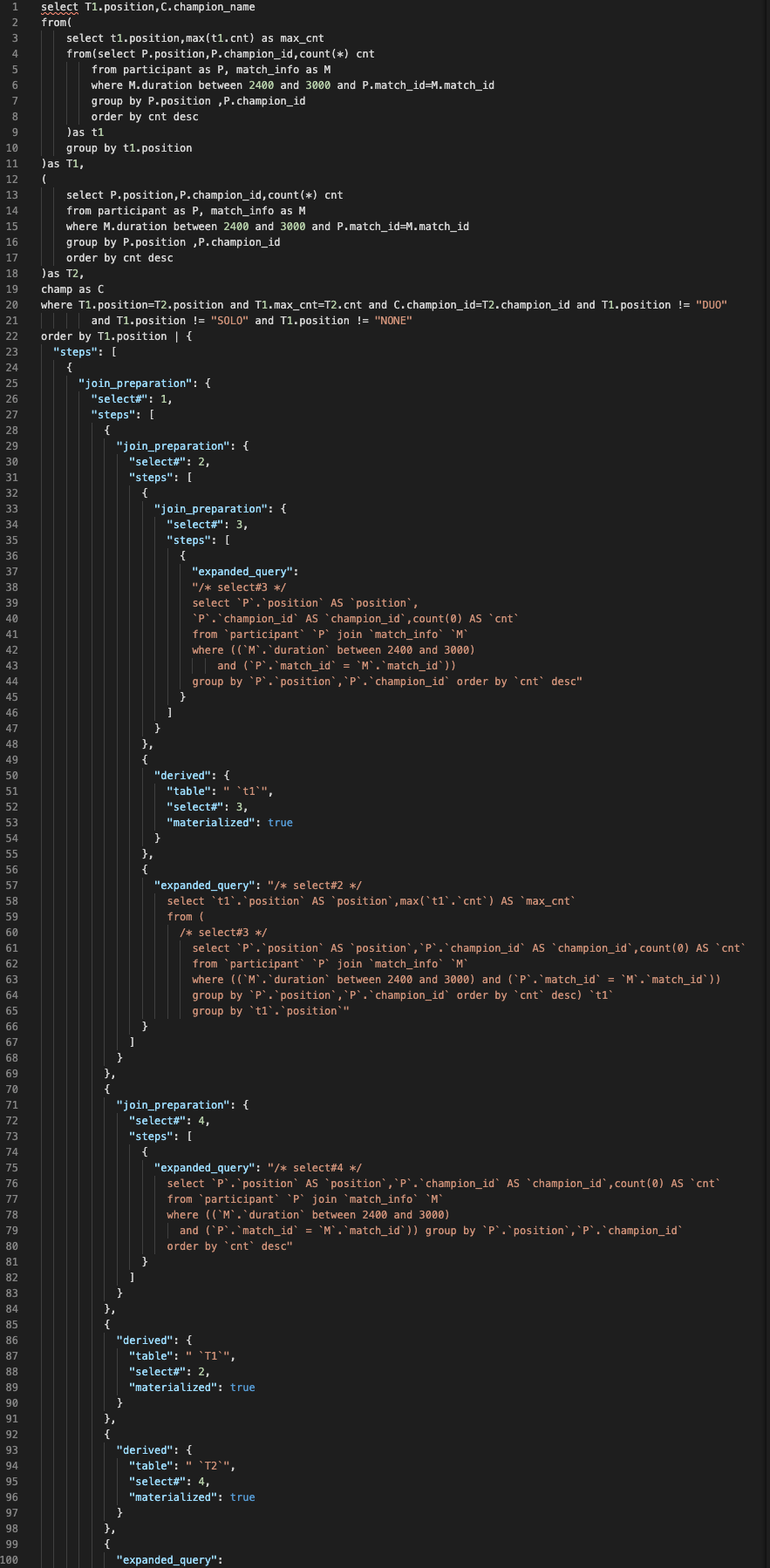


然後把optimizier\_trace的json打開發現有點長，所以複製到mac OS存成json檔再用VScode開啟，然後因為要符合直立式螢幕，所以有些行我手動縮排。



無意義的行數我先不複製

這些都弄好後，json如下









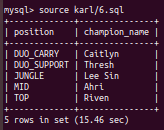


可以看到一個select下面還有subselect，並且select #4,5,8,9各重複兩次，這就是造成時間為17sec的原因

karl:

助教的json看起來很短

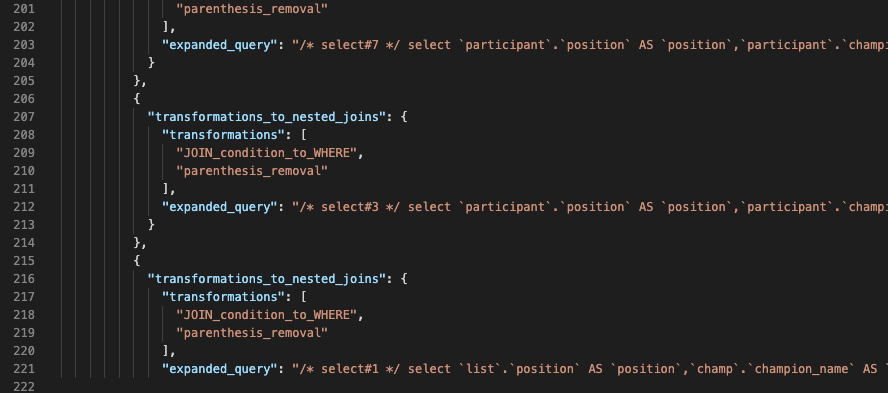
先來看看執行時間



Json:



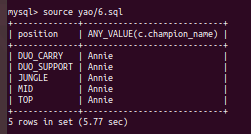




可以看到其實select #4,5,8,9也都重複兩次 寫法跟我的很像

所以得出來的時間才會差不多

Yao:

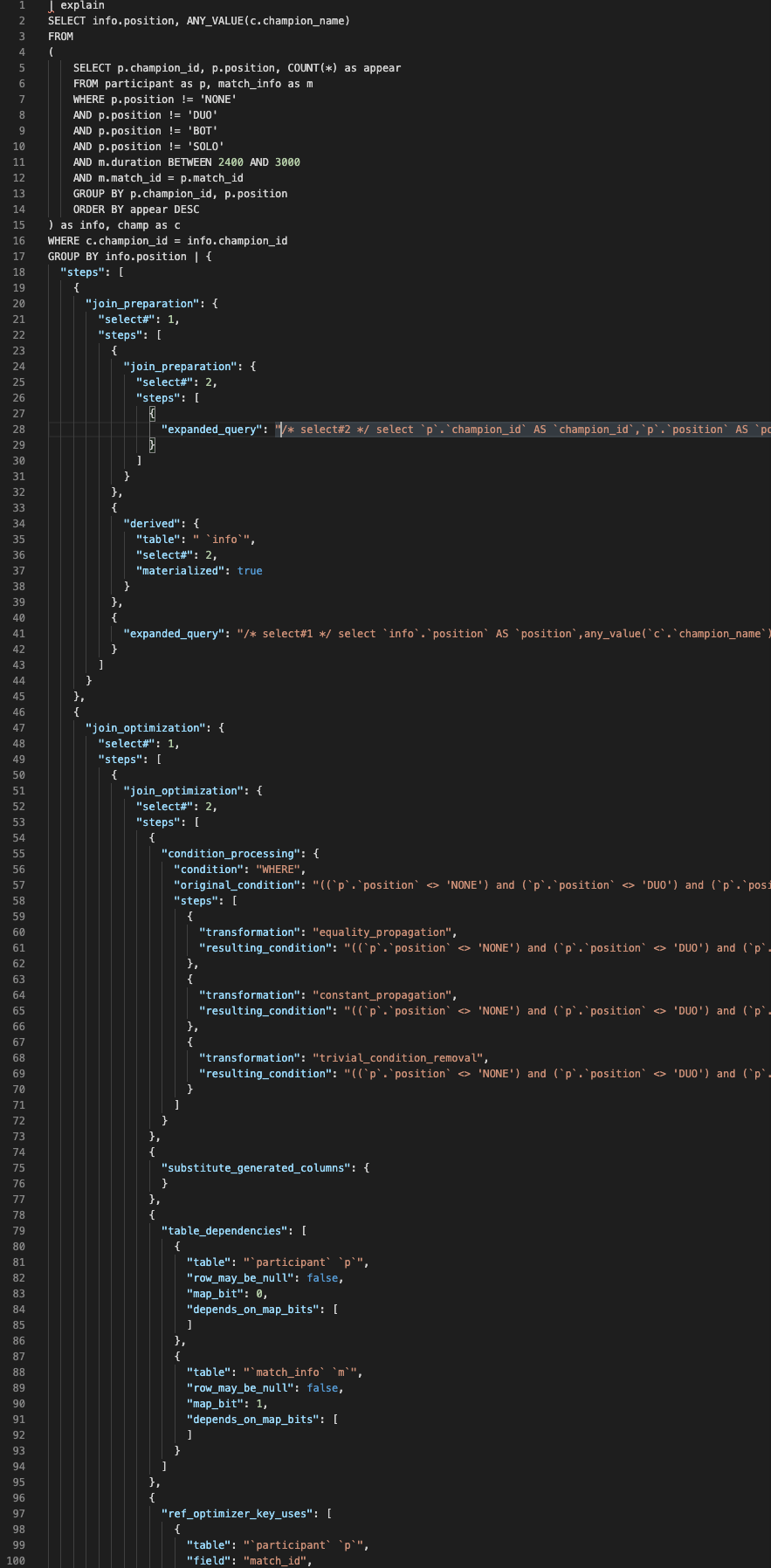


時間只需1/3

但答案在﻿Ver 14.14 Distrib 5.7.30, for Linux (x86\_64) using是錯的

應該是any\_value()的問題

Query也不能在Ver 14.14 Distrib 5.6.21, for osx10.8 (x86\_64)環境下跑













Phase 1 growing phase在18~68行

Phase 2 shrinking phase在70~78行

在main內讀入write target regs, read target regs，用vector<int> regs一次存起在，write target index放在regs[0]。

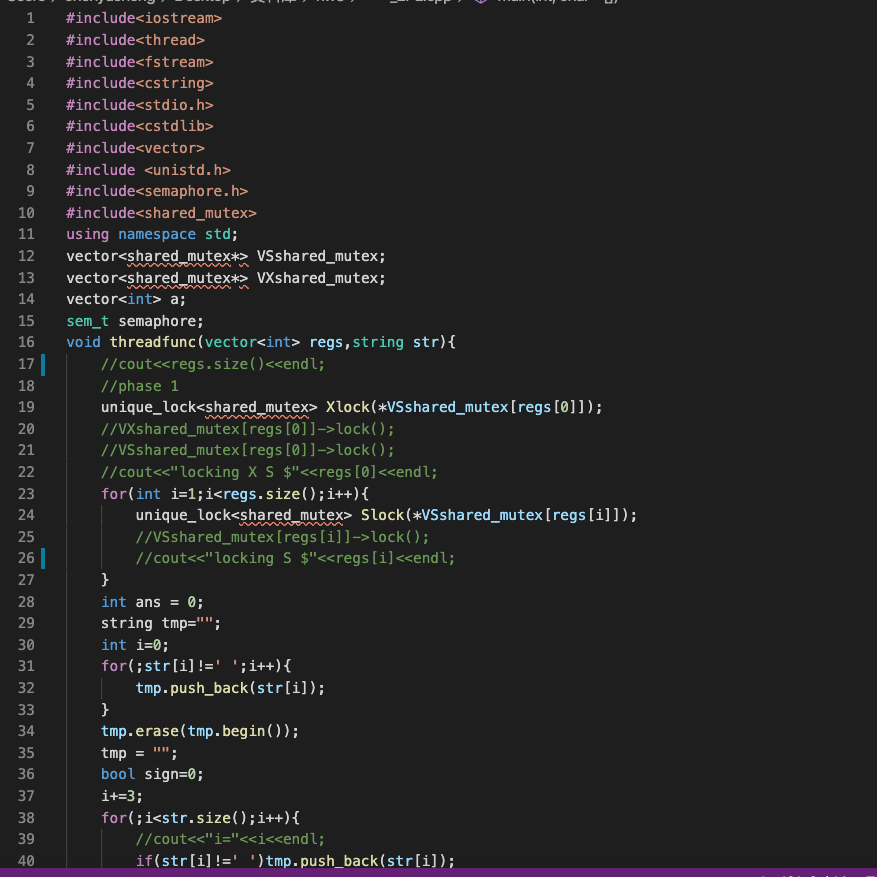
再把regs與整行line讀入threadfunc(reg, str).

Threadfunc是用 VSmutex(type為shared\_mutex) 的vector進行上鎖。

並在函數(threadfunc)內用unique\_lock與shared\_lock分別鎖住write,read target

算完後，assign給write target，最後進入shrinking phase

我沒有用semaphore，可能就是因為這樣data2 才會出錯吧



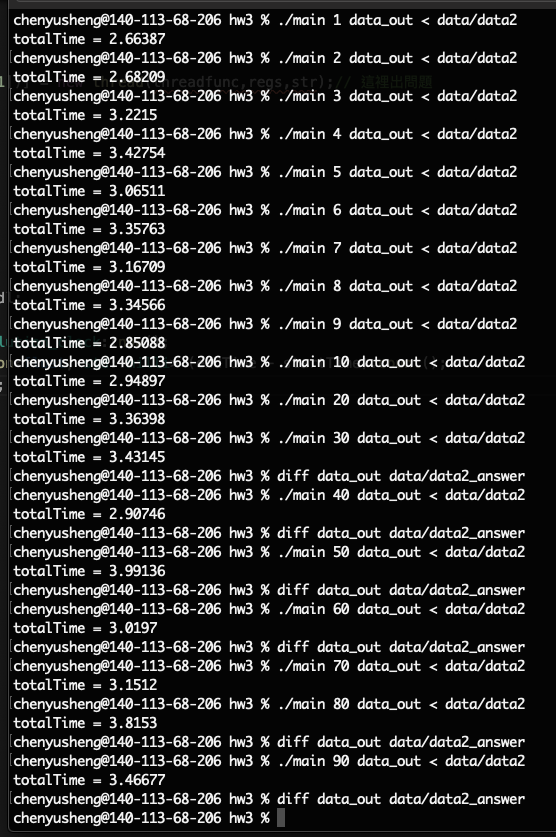


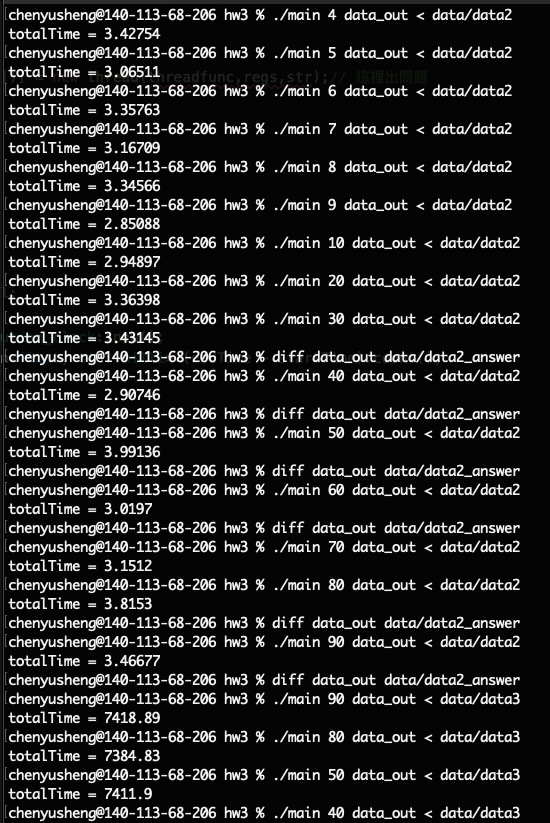


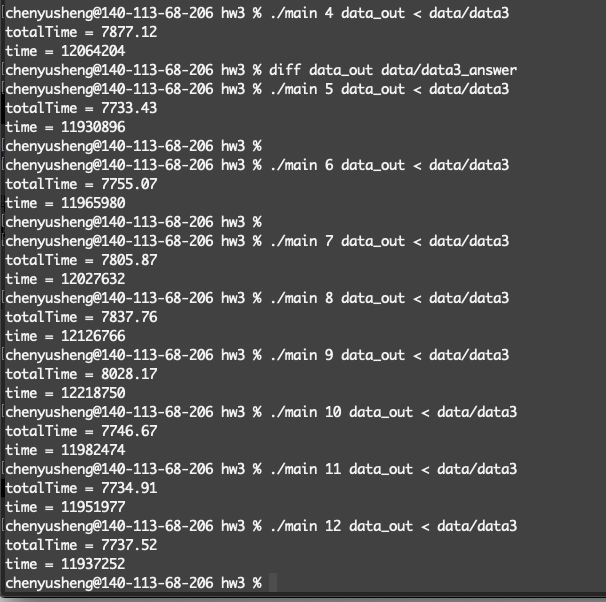


2-2-2

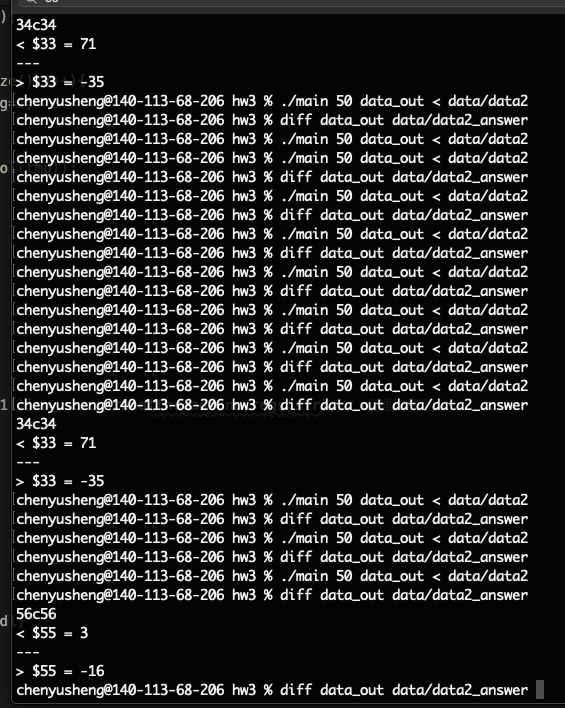
不同的Thread數時間都差不多







2-2-3



Data2有非常低的機率大概(10次出現一次)會出現一兩行不一樣，但只要不快速連續執行就比較沒有錯誤，推測是前面的行數的read被卡住了，被後面行數的write搶先X lock，導致data錯誤。

Solve method:

1. 加一個vector< queue<int> > QV; vector大小跟regs個數一樣
2. 加入line\_num紀錄目前是哪一行，一行代表1 job.
3. threadfunc多一個int line\_num的input參數。

這個QV存每一個regs下一次要被哪一行讀，每個regs都有自己的queue，這個queue紀錄該regs需要被些行數使用，而且queue順序由小到大。

當進到threadfunc內後(，會去看queue的front是否等於line\_num，如果等於就代表現在這個job可以馬上把該regs的mutex lock住;若不等於，則代表前面行數的job還沒有把queue的front pop，要等前面pop完我才能lock。

等到threadfunc算完，並把值賦給write target regs後，pop。

這麼做可確保有平行度且不會順序錯亂，比如說$1 被1, 4, 7,50行所使用。

那麼在讀到第7行後，如果queue內有1,4就要等，而讀到第50行，因為跟前面差太多行，就很有可能queue第一個是50，就能直接進行job.

以上都是在OS X跑，如果在linux跑會因為不明原因永遠無法pop導致queue front 永遠!= line\_num。所以希望助教測試時高抬貴手在OS X跑。我還有準備另一份code不知道是否能同時傳兩個code上去

這兩份檔案分別是\_2PL.cpp, \_2PL\_queue.cpp, 這兩份只差queue的部分。

main 預設是\_2PL\_queue.cpp，如果真的無限迴圈了就請助教跑一下\_2PL.cpp。