資料工程 project 1 report

資工四 劉昌澈

405410075

1. Project 實作概論
   1. 資料處理

因為檔案當中有多筆資料與大部分的資料是不對齊的（例如：少數資料含有@Time，但多數資料沒有）因此將不對齊的部分刪除，以利後面的資料處理。

* 1. 將檔案分割成多個已排序好暫存檔

首先，將資料一筆一筆的讀入一個資料結構為data\_handler的buffer中（大小先猜，不夠再用doubling technique來增加），直到實際讀入資料量（byte為單位）大於預期的讀取資料量，而預期資料量是將整體資料量除以暫存檔的數量（這邊暫存檔我訂四個使後面winner tree修改時間降低）來求得，並在當次讀取把目前的讀取到的buffer以qsort進行排序寫道暫存檔，一直重複直到資料讀取完畢

* 1. 合併暫存檔

依照我上述寫暫存檔的方式此時我會有4暫存檔，我在使用winner tree將他合併而結果就是一個排序好的檔案。Winner tree的實作我以arrary來實作tree的部分，這樣我的access time可以O(1)來調整整顆winner tree，每次調整我都會記住node跟node比較之間的贏家，以利計算下一個需調整的目標，並在找到贏家後寫入最終的sorted好的檔案。

1. Project 實作過程
   1. 原形

原先想要使用read一次讀入檔案的一大部分在進行排序，認為這樣的方法可以大幅降低執行時間，然而因為記憶體管理的不佳，一直遇到一些老朋友（segmentation fault/ abort trap/ bus error），所以重新將code整理並妥善管理記憶體

* 1. 最終版

改良版我以getline一行一行抓，雖然速度慢勒點，但對於我debug方便很多，不用擔心到pointer亂跳的問題，而我記憶體管理也小心很多有malloc到的一定都會free掉，漸少memory leak使得程式不會因memory而crash。

1. 測試環境
   1. OS: macOS Mojave version 10.14.6
   2. CPU: 2.3GHZ intel i5
   3. Memory: 8GB 2133 MHZ LPDDR3
   4. 撰寫語言：C
2. 檔案說明
   1. main.c: 處理使用者輸入的參數並呼叫對應的function
   2. utility.c: sorting中所需的function都在這個檔案底下
   3. utility.h: 標頭檔和prototype define
   4. driver.c: 用來測試tournament tree的 program
3. 資料結構
   1. Data\_handler: 暫存讀取資料，並將其依subject分類
   2. File\_handler: 記錄此次讀取的資料個數與資料量
   3. Queue: 一個處理structure queue的資料結構
4. 核心函式說明
5. int read\_file：將一大檔分批讀入，測試時使用4個檔案分批讀。
6. int convert：將資料讀入memory暫存，如果原先開的data\_handler buffer空間不夠時運用doubling technique將從新malloc memory成2倍大，並在讀取完後將最大資料個數回傳以供下一趟讀取開較大的buffer節省找到適合大小的時間。
7. void assign：用於兩個data\_handler型態做assign。
8. void write\_file：data\_handler型態的寫黨函數。
9. void myqsort：將暫存buffer 已qsort sorting，並output結果到暫存檔。
10. void mergeFile：將暫存檔合併起來，底層採用的是 winner tree來實作，最終

output到sorted.rec

1. 程式編譯與執行
   1. Compile: gcc utility.c main.c -o rsort
   2. Execute: rsort [-rb/-s] [/-r] filename
2. 參數解釋

-rb: 指定資料中的哪比subject為output，並且依照此subject sorting

-s: 依檔案大小排序此時的rb設定為url

-r: 沒下-r預設值是由小到大排序，下-r後變成由大到小

1. 測試結果
   1. rsort -s -r final.rec

real: 1m 50.328s

user: 0m 35.204s

sys: 1m 6.831s

* 1. rsort -rb @title final.rec

real: 1m 17.071s

user: 0m 33.574s

sys: 0m 40.407s

* 1. rsort -rb @title -r final.rec

real: 1m 18.953s

user: 0m 33.553s

sys: 0m 42.037s

* 1. rsort -s final.rec

real: 1m 51.626s

user: 0m 34.155s

sys: 1m11.012s

1. 結果討論

整個時間花比較多的部分是在system call的部分，我在猜應該是在write的那部分，因為我是一筆一筆資料寫出去的，如果要改善我程式的執行效率的話，可以把資料先暫存到buffer中再一次寫出，如此可以減少user mode to kernel mode的次數，達到執行時間下降的效益。

1. 實作過程中的困難與解決方法
   1. Strcpy overlap處理: 實作過程時strcpy會發出exception後來發現是因為malloc給的位置的很相近導致overlap的問題，而這個問題可以用memmove解決，他會依照overlap的情況來處理，例如string1 assign 給string2 ，而overlap在string2的頭，那就是從string1的尾開始讀，這樣就能避免資料不等的情況。
2. 附錄

Github: <https://github.com/ianbig/DE_project2>