DSA Final Project Report
Bank account management system

指導老師：林軒田、張智星
組別：中華愛國同心會
林日能 b03902005
林昱龍 b01502006
顏毓均 b01502069

大綱：

一、工作分配

姓名	工作
林日能	account, history, main program, avltree DS, makefile, final report
林昱龍	find, suggest, memory pool, sharelog, log DS, hash DS, final report
顏毓均	input/output, main, vector DS, final report

二、 比較不同資料結構

(1) Result of mini-competition:

資料結構	上傳時間	分數
STL vector	2015-06-28 16:56:02	284432.000000
avl tree	2015-06-21 22:13:11	356808.000000
rb tree	2015-06-30 17:51:52	341211.000000
bst tree	2015-06-30 17:51:29	337342.000000
STL unordered_map	2015-07-01 01:28:18	406917.000000
sharedlog + rb tree	2015-06-30 17:50:41	391490.000000

(2) 時間複雜度的漸進分析

共同使用的函式
deposit: O(1)
withdraw: O(1)
search: O(t) t: 登入 id 至今擁有的交易筆數
transfer id recommend: **Θ**(N) N:總用戶數

a. main_vector: STL vector

login: O(N) N:總用戶數
create: O(N) N:總用戶數
delete: O(N) N:總用戶數
transfer:O(N) N:總用戶數

merge: O(t1+t2)+t2*O(N)*O(log(ti))
t1,t2: id1,id2 的交易筆數 ti: id2 的第 i 筆交易的對方 擁有的交易筆數
find: O(N)+O(mlogm)
N:總用戶數 m:符合規則的用戶數

b. main_avl, main_bst, main_rb, main_log: avl tree, bst tree, rb tree

login: O(N) N:總用戶數
create: O(N) N:總用戶數
delete: O(N) N:總用戶數
transfer:O(N) N:總用戶數
不使用 shared transfer log 時：
merge: O(t1+t2)+t2*O(N)*O(log(ti))
t1,t2: id1,id2 的交易筆數 ti: id2 的第 i 筆交易的對方 擁有的交易筆數
使用 shared transfer log 時：
merge: O(t1+t2) t1,t2: id1,id2 的交易筆數

find: O(N) N:總用戶數

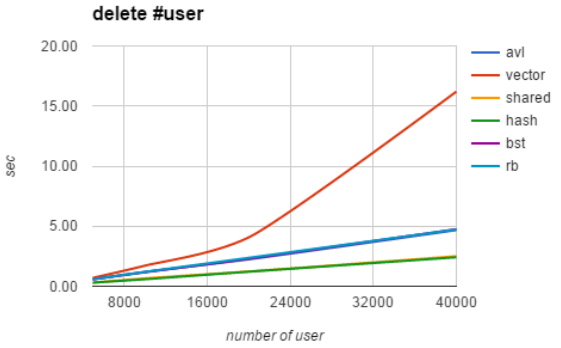
c. main_hash: STL unordered_map

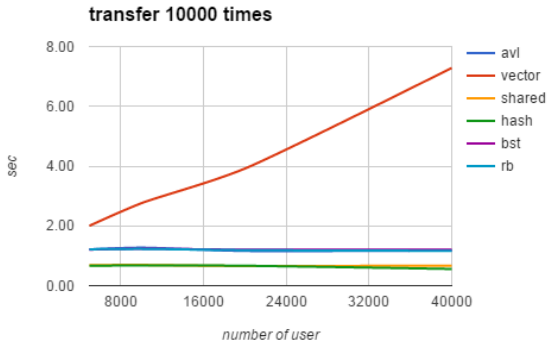
login: O(a) a: load factor, 在記憶體空間足夠時<0.5
create: O(a)
delete: O(a)
transfer:O(a)
merge: O(t1+t2) t1,t2: id1,id2 的交易筆數
transfer:O(a)
find: O(N)+O(mlogm) N:總用戶數 m:符合規則的用戶數

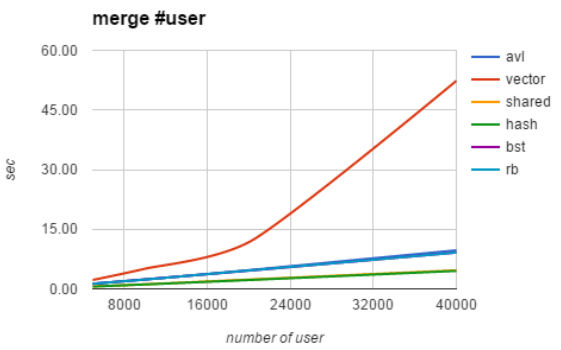
(3) 利用自己做的測資產生器(test_data/generator) 進行不同動作的測試

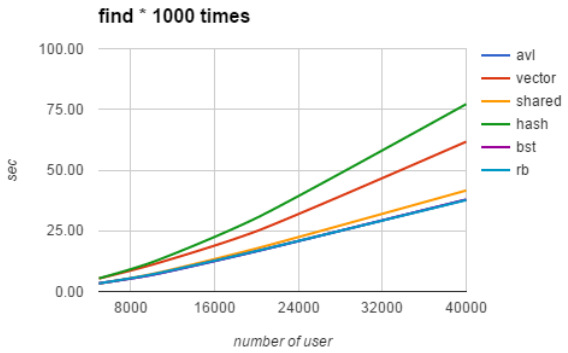
create 一百萬個帳戶所需時間比較(單位：秒)

	test_set1	test_set2	test_set3	test_set4	test_set5	test_set6	test_set7	test_set8	test_set9	test_set10	average	best	worst
main	9.69	10.47	10.18	13.80	14.89	9.45	14.88	14.00	11.26	9.52	11.81	9.45	14.89
main_vector	too long	too long	too long	too long	too long	too long	too long	too long	too long	too long	too long	too long	too long
main_log	11.81	15.42	16.87	11.71	11.84	14.93	10.41	13.25	13.76	14.21	13.42	10.41	16.87
main_hash	10.36	9.52	7.56	10.71	7.00	6.83	12.42	7.81	7.12	9.95	8.93	6.83	12.42
main_bst	13.49	10.58	12.15	11.33	13.91	14.46	12.18	12.18	15.60	10.95	12.68	10.58	15.60
main_rb	10.01	11.43	12.25	9.64	11.06	9.82	14.13	14.19	10.59	8.58	11.17	8.58	14.19









三、推薦資料結構

超級比一比

	Tree	Hash	Vector
空間			勝
create		勝	
delete		勝	
merge		勝	
find	勝		
transfer		勝	

綜合以上評比，本組選擇以 Hash 做為主要結構。

(1)優點：

a. 查詢，插入，刪除快速
b. 不用進行 logN 次的字串比較
   對於一個字串長度為 L 的字串，經過 hash function 運算會花 **Θ**(L)的旋轉和乘法運算
   進到 bucket 中會進行 m 次(bucket 中的帳戶數量)的字串比對 O(L)
   相較於 binary search tree 就要進行 O(logN)次的字串比對 O(L)
   對於一個良好的 hash 可以期待 m < logN

(2)缺點：

a. 在 find wild 回傳結果時必須排序，非常耗時，相較之下 tree 可以用 inorder traversal 可以直接得到排序好的結果
b. hash table 所佔的記憶體非常多
c. 被知道 hash function 後容易被碰撞攻擊(註 ：stl:unordered_map 在 gcc 中所使用的預設 hash funciton 為 MurmurHash 2.0)

四、使用方式

於 final_project 使用 makefile 編譯系統

用 make　　編譯 以下六個執行檔

用 make main 編譯 AVL tree data structure

用 make main_vector 編譯 vector data structure

用 make main_log 編譯 red-black tree + shared log data structure

用 make main_hash 編譯 hash data structure

用 make main_bst 編譯 binary search tree data structure

用 make main_rb 編譯 red-black tree data structure

用 make cleanall 清除所有.o 檔與執行檔
於 test_data 使用 makefile 產生測資
用 make 編譯 generator
用 make run 產生 test_in.txt

五、Bonus Feature

(1) 新功能：log

打入 log 可以顯示從系統開始以來到現在所有的交易紀錄，以 From $A To $B $money 的方式一行行的呈現，並且依照時間順序排列。
因為使用 shared log：中央共同轉帳紀錄的關係，所以每個帳戶要進行 merge 時只要從帳戶儲存的轉帳紀錄指標找到那筆轉帳紀錄，再把屬於自己那方的舊名字換成新名字就好了，因此所需要時間只和需要更名的紀錄數量成正比，並且銀行那方也保有永存的全部轉帳紀錄。

(2) find 的實作

原本是想利用前綴字元來縮小搜尋範圍，但考慮到可能會有*abc..等以星號或問號開頭的測資，以及我們這次所用的資料結構和前綴字元的相關性不大，加入額外的判斷條件反而會影響效能，所以我們就採用了 inorder traversal 或暴力搜尋每一個在資料庫裡的帳戶字串，加上一點長度和萬用字元數目的判斷，再利用 match_wild 這個函式進行萬有字元字串的比對。
match_wild(string str, string wild)
因為 find 所用到的 regular expression 只有?和*，沒有()等其他萬用字元，所以我們利用 NFA（非決定性有限狀態機）的概念，把每一個查詢字串 wildcard 的字元視為一個 state，每次走到下一個 state 時就消耗 str 的一個字元，當無路可走或走完了還沒到終點就換一個分支。
state 可以分為四種：

a. '\0'：如果比較的字元都是結束符號，則代表字串符合 wildcard 的模式(Accept)，reutrn ture
b. 英文數字：如果比較的字串 str[i]和它相同則到下一個 state，如果不是則 Reject 此條分支，檢驗其他可能的分支。
c. '?'：消耗任一個字元後往下一個 state 前進。
d. '*'：這裡會產生兩支分支：不消耗任何字元，往下一個 state，或停在這個 state 消耗一個字元。在這裡我們利用 stack 來記錄還沒有被選擇的分支，等到現在分支被 Reject 的時候就從 statck 中拿到其他未探索的分支的資訊。如果 stack 空的話，就代表已經沒有其它的分支了，此時可知比較字串不符合模式，return false。

當然 NFA 可以轉換成 DFA，但是太複雜所以就沒有實行了。

(3) Recommand System 相似 id 推薦系統

在 create id 時，如果該 id 已被使用就要推薦前十個可用 id。我們的做法是按分數及字典序由小到大生成可能的 id，再去檢查該 id 是否存在，如果存在就繼續找，直到找齊十個為止。首先，我們準備了一個叫做 Rank 的容器，它可以保持一個前十名的排行榜，並且告知新加入的 id 是否入榜。接著我們按照分數->字串長->可變字元的方式來分門別類生成字串：先決定分數（由小到大），再依照分數決定字串的長度（由短到長），以及和原字串字元不同的位置（由後至前，以回朔方式產生），這樣子我們就可以得到一類長度固定，但部分字元尚未決定的字串。接著再依照字典序由小到大產生字串，如果字串符合條件就插入至 Rank 中，如果無法上榜就代表現在這個字串比榜尾的字串字典序還要大，因此就可以跳出，處理下一類的字串。當某次分數輪完發現 Rank 已經滿了，就可以結束並回傳 Rank。

至於 transfer id 的推薦，因為 62^101/(62-1)的可能性太多了，要用上面的方法找到在資料庫的十個 id 怕是不容易，因此改採用暴力法搜尋全部的 id 並用 Rank 取前十名 id。

(4) Memory Pool　記憶體池的檢討

記憶體池，就是先預留一塊記憶體空間，然後自己進行記憶體拿取／歸還的管理，這樣子在動態規劃記憶體時就可以和記憶體池要空間而不必透過 malloc/free 的 system call。
在這次的作業中，因為動態規劃所使用的資料結構單位的大小都已經固定了(Account，Node)，因此我們可以先拿一塊固定大小的記憶體再依照資料結構單位的大小切成很多塊，然後用一個 stack(用 array 來實作)來紀錄每一塊可以用的記憶體位址，每次拿取空間時讓 stack pop 一個位置，當記憶體池被用完時再呼叫 malloc 從系統拿空間。而歸還時，則先判定歸還的位址 p 是否屬於記憶體池，如果是則用 stack push 紀錄新的可用位址，如果不是則使用 free(p)歸還。

實作結果：在開啟 O2 優化之後，有沒有使用記憶體池的影響變的微乎其微，在本機上幾乎看不出差別。而在 judge system 上，用太大的記憶體池反而會使分數降低。猜想是因為 malloc 所消耗的時間和所需要的空間大小成正相關，因此用的記憶體池越大反而越耗時。

六、引用資源

md5：http://www.zedwood.com/article/cpp-md5-function
avl, bst, rb：http://adtinfo.org/ (From hw6)