

Data Structures and Algorithms, Spring 2015

台大資訊系 - 資料結構與演算法

Final Project Report

Bank Account Management System

Team Name

田神與他的小夥伴們

Professor

Prof. 林軒田(Hsuan-Tien, Lin)

Prof. 張智星(Jyh-Shing. Jang)

Team Member

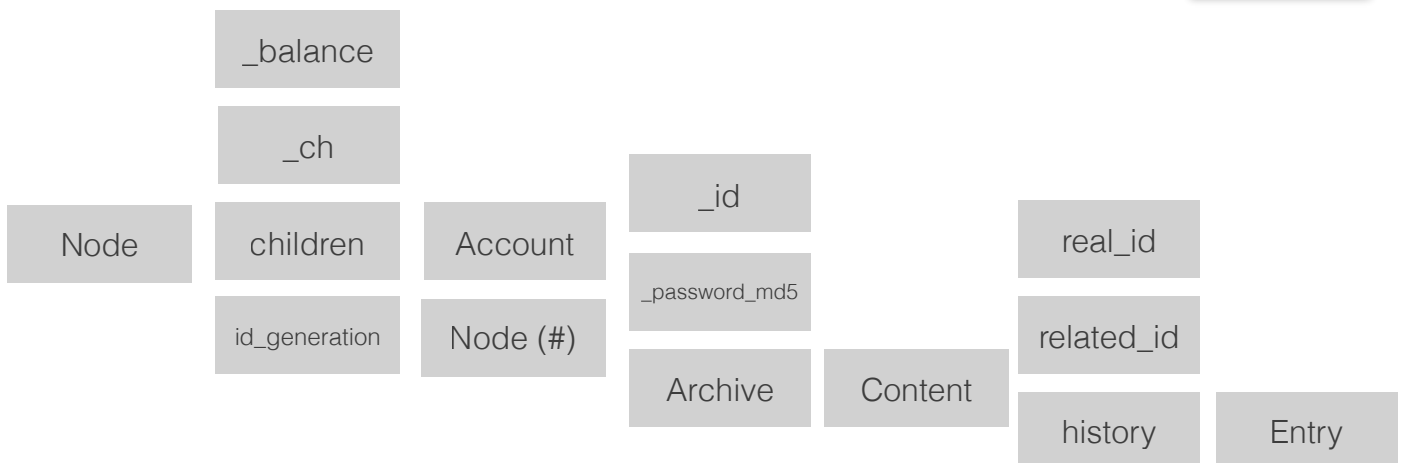
B03902049 陳力宇

B03502040 劉君猷

B03901023 許秉鈞

【主要架構】 - (4) the data structure you recommend

Trie



Trie是一個樹狀結構，裝載Node；Root為無_ch的Node。

Node內包含一個char “_ch”、現有金額“_balance”、一個裝載Account*的List “children”、一個裝載Node*(#)的List “id_generation”， (#)代表被刪除的Account。

Account內包含_id（若已經被刪除，則在string後端加上 # (編號) ），_password_md5。

Archive為內含Content*的vector，代表每個Account與其他帳號的所有交易紀錄。

Content為一紀錄“擁有此Archive的Account”與“read_id”之間的交易紀錄，即history。

Entry為最小單位，_value紀錄交易(transfer)時的交易金額，_fot紀錄是收入或支出。

另外，在main中有一個current_account，為全域變數，用以紀錄現在登入的帳號。

【設計順序】

1. 先設計account.h，將所有的子集合寫進標頭檔。
2. 接著設計trie.h，處理Node之間的insert與find_child，其中trie不具有delete功能，因為刪除的Account會成為id_generation的殘餘，並在id上顯示#(編號)。
3. 設計function.h，在function中不處理真正的記憶體配置，只處理外部的stdout。
4. 設計auxiliary.h：先完成較簡單的function，如login、create、deposit、withdraw。
5. 設計delete function，其中比較重要的是，需traverse過“Archive”有接觸過的Account，並將id修改為在string後端加上 # (編號) ，註名 _deleted = true。
6. 設計search function。
7. 最後處理需要score的merge、transfer與find。

【優勢】 - (5) the advantages of the recommendation

1. login: based on a ID and a password. Note that the password cannot be stored in your system in its original form.

Time Complexity為 $O(\log n)$ ，即find_id呼叫find_child，



從children的List中從頭開始找字母，找到後進入下一層。

2. create: create an account with some demanded ID. If the ID already exists, some alternative IDs must be recommended.

create_id(id, md5(password));

$O(\log n)$ ，呼叫find_child與insert，從children的List中從頭開始找字母，找到後進入下一層。

recommend_unused(id);

此function為推薦score最小的相近但不存在的Account，並印出最接近的10個。

我們的做法是，先從目標Node向上一層找parent，即先把id pop_back最後一個字元，再重新find_id，然後從parent的children中尋找score = 1的結果，若traverse到自己的node時，則traverse自己的children。

因此，全部走一次共 $62 + 62 = 124$ 種可能，我們「假設」所有的id都是隨機取名，因此不太可能在124種中卻找不到10個不存在的Account。在這個情況下，我們可以把時間壓得很快，約 $O(1)$ 。

另外，我們設計了“-”號，並加在related_id後，以方便排序並讓系統知道這個帳號是id_generation中的一代。

3. delete: delete an account.

我們的做法不刪除帳號，而是修改id名稱，並用bool_deleted註名已被刪除。比較重要的是，需traverse過"Archive"有接觸過的Account，並將id修改為在string後端加上#(編號)，並在archive上的related_id加上“-”號方便辨識，註名_deleted = true，但因為只需find_id Archive上的Account，目標是特定對象，不必把整個樹traverse一次，因此約為 $O(\log n)$ 。

4. deposit: deposit money into an account.

$O(\log n)$ ，找到自己的Node並修改_balance。

5. withdraw: withdraw money from an account.

$O(\log n)$ ，找到自己的Node並修改_balance。

6. transfer: transfer money to another account

呼叫enroll() function，並在user == nullptr的時候呼叫recommend()。

enroll()做的事情是，將一筆交易紀錄，存到當下的account的archive中。

然而，recommend()需要將附近最佳score的子樹都traverse一次，最終挑出10個，因此需要 $O(n)$ 。

7. merge: merge two accounts into one.

呼叫find_id()與combine_id(),



combine_id()在做的事情是，更新user->archive這個vector裡面的Content，為了加速使用Binary Search，總時間為 $O(n \log n)$ 。

8. search: search the history of transfer.

Bottle-neck為show_history()這個function，它在做的事情是，在當下的account中的archive，先用binary search找到history中的目標，然後印出From 或 To，以及Entry中原本所紀錄的金額，時間應為 $O(n \log n)$ 。

9. find:

find為 $O(n)$ ，因為要全部都找才知道全部符合的有哪些，另外，我們在這個function引用了開源碼，以解決並加速wildcard的計算。

ref: <http://www.codeproject.com/Articles/1088/Wildcard-string-compare-globbing>

【劣勢】 - (6) the disadvantages of the recommendation

各別在insert,search的時候不會是最快的，相較於unordered map的insert或是vector的search。但因為amortize的關係，它對於每種操作的複雜度很平均。因此，在太小的測資上，均攤的效果不明顯，如果本次的judge system是更多的秒數，trie將是很棒的選擇。

【三種資料結構】 - (3) the data structures you compared, including the results submitted to the mini-competition site

2.0 unordered_map + vector Version :

與ordered_map的差異在於，此資料結構為無序，因此在做插入與搜尋時會比較快，但是想要有順序的使用的話就要先sort過、或是預先處理過M所以如有需要這部分的函式會拖慢速度，但經由在judge system上測試的結果，同樣時間內可處理的行數比3.0版多一點，於是可猜測測資裡面呼叫插入搜尋的函式應比推薦有序的函式多。

3.0 ordered_map + vector Version :

1. login: based on a ID and a password. Note that the password cannot be stored in your system in its original form.

$O(n)$ ，在整個map中find這個Account（此為 $O(\log n)$ ），即出現在map裡面find的使用。

2. create: create an account with some demanded ID. If the ID already exists, some alternative IDs must be recommended.

在一開始時，需要access到某id，所以需要先做find，為 $O(\log n)$ 。然後在有此ID的狀況，又會需要 $O(\log n)$ ，所以共為 $O(\log n)$ ；但若找不到此ID，則會推薦數個比較適合的ID，此操作的複雜度為一常數C乘上 $O(\log n) \rightarrow O(\log n)$ ，所以總共也是 $O(\log n)$ 。

3. delete: delete an account.

delete()也需要找到id，所以複雜度為 $O(\log n)$ 。

4. deposit: deposit money into an account.

因為此操作為已登入current_id的情況下，所以複雜度為 $O(1)$ 。

5. withdraw: withdraw money from an account.

因為此操作為已登入current_id的情況下，所以複雜度為 $O(1)$ 。

6. transfer: transfer money to another account

因為要transfer所有Graph上和Account有連結的點，所以利用iterator跑一遍，時間複雜度理論上worst case是 $O(n)$ ，平均case會比較快。

7. merge: merge two accounts into one.

直接將兩個Database做merge，只要考慮Database的edges的size()，因此為 $O(n)$ 將之全部檢查一遍即可。

8. search: search the history of transfer.

這是map結構中最吃時間的function，因為必須考慮整個current_account裡面的edges的size()，裡面還有許多巢狀結構的判斷式，結果約為 $O(n)$ ，但太多的if判斷式也會拖慢速度。

9. find

find為 $O(n)$ ，因為要全部都找才知道全部符合的有哪些，我們在這個function引用了開源碼，以解決並加速wildcard的計算，同1.0版本。

(關於md5的時間複雜度，因為此為一個hash function，因此為 $O(1)$ 。)

3.0版的優勢與劣勢：

3.0版的優勢是，因為map的order特性與期stl裡面的函式可以使用，所以寫起來可讀性很高，程式碼相較1.0版本短很多，但缺點就是執行插入搜尋時因為都要走 $O(\log n)$ 下去，所以在修

改插入大量資料時，整體速度會慢下來，可由judge system知道在小筆測資時，1.0的trie寫法跟這方法所花的時間不會差很多，但是越大筆速度就會拉開，以致於到judge system上面測試時能處理的行數會相差5倍多。

【分工】 - (2) how you divide the responsibilities of the team members

1.0 (trie version), 所有人一起寫

2.0 by 陳力宇、許秉鈞

3.0 by 劉君猷

優化：劉君猷、陳力宇

report by 許秉鈞

【如何編譯】 - (7) how to compile your code and use the system

我們將3個版本的資料結構分成3個資料夾，針對一個資料結構的寫法是：將所需要的功能劃分好，切成許多分開的檔案寫，然後要使用到互相的標頭檔時互相include並extern外部變數，在編譯時使用make，對於整個target裡的檔案都編成.o檔，再用linker的方式連結成一個./final_project的執行檔

【額外加分】 - (8) the bonus features you implement and why you think they deserve the bonus

1. 我們將trie以最「精簡」的方式從零到一code出來，沒有多餘的記憶體浪費，在空間上省了非常多的資源，而且若考慮均攤的時間複雜度，我們也有很好的成效。
2. 我們有很清楚的class relationship，每個object的分工都非常完整。
3. 我們在1.0 版 和 3.0版之間，不同的資料結構下（map與trie）有重新設計結構，並學以致用、把離散數學圖論中的Graph與Edges的想法應用到3.0版本的edge中。
4. 總結而之，雖然我們在時間上並不非常突出，但若同時考慮空間與時間，我們的程式將是最適合實際資料的存儲資料結構。

【心得】 - (1126) 感恩田神，讚嘆田神

謹以此期末報告，紀念大一下的資料結構與演算法課程。

當初只是抱著「DSA段考很難、作業很重、可以學到很多」，想要衝一發的心情，就填了這個課，看到班上這麼多同學、甚至是一大堆加簽，真的讓我很驚訝！最後，我必須說，這堂課真的學到了很多很多！

這是DSA最後一次的作業，回想這學期，有太多次TA課纏著助教不放、為了debug壓縮到其他科的讀書時間，還有無數次的作業討論、期中期末瘋狂Coding訓練手感…，從高中從未碰過程式，到今天能夠輕鬆完成Class，甚至是熟稔Heap、Red-Black Tree，還自己課外學了一些BFS, DFS, Greeding演算法，真的覺得收穫良多。

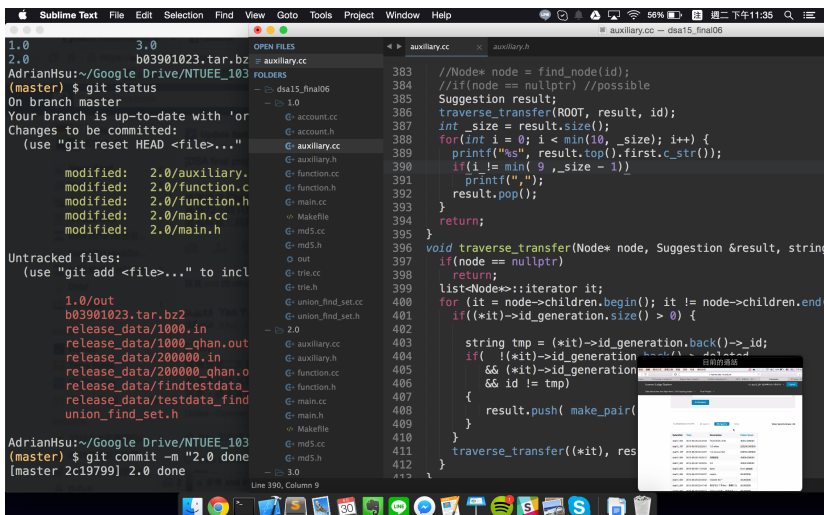
這次的銀行系統，從資料結構設計、區分檔案、到debug都是從零到一，因此特別棘手、但也學到非常非常多，這也是我們第一次寫超過1000行的程式！比較可惜的是，我們在最初就選擇了Trie這個資料結構，卻沒有知情測試方法是限制5秒比較行數，因此比較吃虧。如果將主力投入在unordered_map或map，我們應該會有更好的成績。

一連五天的資結，從超慘的看不懂別人寫的code、我的code別人看不懂，git可怕的conflict與merge、少判nullptr與忘記防呆的segmentation fault、上傳時超緊張與期待、然後result出來一個99.5%的Error，真的是非常刺激又好玩，哈哈！

老實說，第一個晚上設計整個架構時，還有最後一晚瘋狂的優化，真的覺得設計code超好玩、很有成就感。但是，其他的夜晚就別提了吧zzz

感謝田神，從第一堂課起就努力跟上老師的速度，從一開始跟不上，到了期末竟然能夠超前進度、還多自修了map, radix tree, trie…，也許這就是發現對於一個學科的熱情吧！為了追上進度，希望「將寫程式學好」的一個單純的意念，讓我不斷擴長知識。甚至，這堂課某方面點燃了我對於Computer Science的熱情，雖然早已耳聞是重達十學分的超重必修，但，這些是值得的。

很高興DSA讓我們擁有這麼充實的資料結構與演算法課程，感謝 天翼、彥頡兩位助教，尤其是彥頡TA，記得有一次問了很久但沒有答案，助教竟然寫了一封兩三頁的回答、並附上References給我，覺得感激不盡！謝謝軒田、智星教授，謝謝助教！



Submit實況轉播