2012 Final Project Report

(1) Team Name：DSAGG（Don't Submit A Goddamn Garbage）
Team Member：B00902062 陳俊瑋

(2) The Evolution of Our Code & how we compile our code：
I write a Makefile which allows us to run the code on linux.
All the phases of my code：
Phase 1：
我的 class email 裏頭有 string 型態的 from, to 的對象，還有 vector<int>id number，做為記錄現在已存過的信，還有將 Date 轉成一個相對時間(int 型態的 time), string 型態的文章內容，且用 vector 包住(即為 vector<string>content), 而整個 class 也是由一個 vector 包住(即為 vector<email>mail)。 另外，我起初就將 id 對應到 vector 的第幾項，因為好處在於我的以如果單用 push_back 整個 class 的方式，有可能 vector class 對應的 id 為 2,3,5,7,1 共 5 個，但若今天要 remove mail1 的時候，我必須藉由一個 iterator 去跑整個 vector class，顯然 worst case 的 time complexity 為 $\boldsymbol{O}$**(n)**，但我今所採用的方式，可以明確的知道 mail1 即在 vector class 的第一項，因此我在 remove 上所須的時間必為 $\boldsymbol{O}$**(1)**，不過壞處在於我一開始就得定義 vector class 的 size。在 search 部分，我跑一個 idnumber 的 for 迴圈，每一次對應到的 id，即為對應到的信，因而我再去跑一個對應信件內容的 for 迴圈，去確認是否有出現過這個 keyword，所以在 search 時,我的 complexity 為 $\boldsymbol{O}(\boldsymbol{k}) * \boldsymbol{O}(\boldsymbol{s}) = \boldsymbol{O}(\boldsymbol{k} * \boldsymbol{s})$ k 為現在已加入的 mail 數, s 為 mail 中 subject 跟 content 的 words。
運用此作法做出的 code score 在 **7000~8000** 之間

Phase 2：
我將上個階段存在 class 裏頭的 content 改成用字典樹的方式去 implement，節點開了以 128(ascii 數量)的指標陣列，將每個單字都存在 trie 裡，並在 trie 的節點裡開一個 set 存這個單字出現在哪些 mail 裡，也就是把這些 mail 的 id 存在這個單字的結尾字元的節點裡的 set。會選用 set 的原因是 set 會處理重複 id 只會存一個，也就是當同一個文章出現兩次相同單字， 我只會存一次 id，搜尋一樣是從現在有 add 多少的 mail 從第一個搜尋到最後一個，檢查有沒有 keyword 從 class 的 mail 的 content 找變成直接從 trie 的字典樹搜尋 keyword，再檢查 keyword 的結尾字元有沒有儲存這個 id，檢查的部分可以使用 set 內建的 find，那花的時間是$O(\log n)$, n 為該 node 存的 id 數，對每個 keyword 都相同的檢查，如果都有找到，那就把這個 id push back 到 ans 的 vector, sort ans 再 cout 出 search 之後的結果。因此這個 phase 我減少了每次跑 mail 裡面的所有 content 的時間，也就是$\boldsymbol{O}(\boldsymbol{k}) * \boldsymbol{O}(\boldsymbol{s}) = \boldsymbol{O}(\boldsymbol{k} * \boldsymbol{s})$, k 為 addmail 數, s 為每

個 mail 含的內容，變成是直接從跑$\boldsymbol{O(n)}$ *keyword 數* $\boldsymbol{O(k)}$, n 為 keyword 長度 k=addmail 數，就可以求得 ans。
經過此修正的 code score 在 **13000~15000** 之間。

Phase3：
我將上個階段的 set 改成 unordered set，這樣可以省去 set 去做 sort 的時間，另外在 search 的部分，我加了一個 word filter 的功能，就是將每個不同的 word 在該封信出現過做一個紀錄，舉例來說，若 apple 在 mail1, 3, 6 出現過，我就會將 apple 這個單字紀錄 id 1, 3, 6。而我會將第一個 keyword 的 filter，跑一個 nested for 迴圈，外層為 filter id number，內層去檢查該封信是否有其他的 keyword, worst case 的 time complexity 為$O(k) \cdot O(s) = O(k * s)$, k 為第一個 keyword 對應的 id number 數, s 為 mail 中 subject 跟 content 的 words。為了讓我外層的 for 迴圈跑的次數減少，我根據猜測性的想法會先將原先的所有 keyword 用 STL sort 的動作，將**最長**的字改為第一個 keyword (猜測的原因：我當初是想說像是 I, a, an, is, me, the 之類較短的單字出現比率相對一些較長單字如 wikipedia, administration 來的高)
經過此修正的 code score 在 **30000~36500** 之間

Phase 4：
我開始修改一些小細節，像是 function 的部分，我把 Date 轉換 int 的部分做了更優化的處理； 而在主函式裏頭，我把許多之前宣告瑣碎的變數做更簡潔的處理，能不必要存的東西就不存以節省 space。在印出方面，我把 cout 改成 printf, fputs 或是 puts，讓速度變得更快。在輸入方面，因為我是用 cin，我補上 cin.sync_with_stdio(false);使其加速。最後在 Makefile 我也補上 -O3 優化編譯。
經過此修正的 code score 在 **37000~44000** 之間

Phase5：
我把之前 word filter 的第一個 keyword 做修正，因為最長的單字不代表說一定對應到最少的 id number 數，因此，為了確保第一個 keyword 有最少的 id number 數，我去記錄每次 search 的所有 keyword 我的 id number 數，把最少的把換成第一個 keyword，如此可以才可以完全確保上述提到的 nested for 迴圈的外層都是跑最少的 id number 數。
經過此修正的 code score 在 **45000~52063** 之間
/* 對我來說, 52063 是一個很印象深刻的數，因為我卡在 52063 12 次，不尋常地卡在同一筆測資讓我感到相當困擾卻又不知道錯誤何在，於是在寄信給 TA 後，我才得知原來我 subject 的部分沒處理到 empty 的情況，因此之前的 code

會發生 segmentation fault 的問題，經過小幅修正，讓我突破此障礙。
經過此修正的 code score 在 **53000~56000** 之間 */

Phase 6：

這次我把 trie 的節點多開了一個 vector, norder 同樣是存 id，因此現在的 trie 節點有兩個存 id 的 STL，一個是 vector,一個是 unordered_set，因為 vector 是拜訪比較快，也就是跑迴圈從 i=0 到 i=vector.size()的速度比 unordered_set 跑 iterator 快很多。而 unordered_set 是 find 比較快，大概是 O(logn)的時間，因此我在找第一個 id 數最小的 keyword 存了哪些 id 用 vector 拜訪取值，再丟入第二個，第三個，到最後一個的 word 都用 unordered_set 的 find 去檢查有沒第一個 Word 的 id，沒有的話就提早 return，有的話就 push back 到 ans 裡。輸出的部分，從原本的一個動作 printf 一次，變成累積五萬個字元 fwrite 到 output 一次。因此這個方法我分別利用了兩種不同 STL 的好處, vector 拜訪快, unordered_set 搜尋快，這樣比原本只有 unordered_set 的拜訪和搜尋快。然而這樣犧牲了多餘的空間儲存相同的數據。而輸出的部分減少了每次都要 printf 的時間， 而是可以再 fwrite 一次。
經過此修正的 code score 在 **60000~62000** 之間。

Phase 7：

由於我之前用了很多 c++的東西，包括 fstream, sstream, cin, string, getline 有鑑於 C++可能包裝過多的東西使得速度拖慢這次決定大幅度的修改。首先把 fstream 搭配 getline 的部分改成 fopen 和 fread 和 fscanf。getline 動作的部分，改成 fgets 一次抓一行。,string 的部分，其實原本很多 function 在傳入的部分都有還是有做 string.c_str()的轉型動作，因此我只要把輸入輸出的部分都由 string 換成 char 就好。char 雖快，但是要注意 size 的問題，在此就因為我分割 conent 的 word size 開太小，碰到第 53055 的測資有一個測資就卡住，最後在助教提示下才知道有 Verwaltungsgemeinschaft 這種超長單字，把 word size 開更大之後，順利取得新高分。因此從 c++換成 c，用底層的方法速度會更快。
經過此修正的 code score 在 **101000~103000** 之間。

Phase 8：

這次再處理一些小東西。包括 itoa 數字轉 char 的函式自己 implement，取代 fstream 轉換型態動作。還有把部分 if else 判斷式改成 switch case，然後把 i++ 的部分改成++i。還有原本一次 add mail 需要好多次的 fscanf 和 fread 改成 fread 一次到一個超大的陣列，再進行切割的動作。因此, itoa 自己 implement 可以減少某些型態判斷，而 switch case 再判斷時只需要取一次值做比對，比 if else 更快。而++i 比 i++快是因為 i++還需要有 temp 存值的動作，因此沒必要

的話用++i 也會快一些。而原本一次讀檔需要多的 fscanf 和 fread 改成一次的 fread 也節省時間。
經過此修正的 code score 在 **118000~120000** 之間。

Phase 9：

這次我把 STL 的 sort vector<int>ans 的部分, sort 自己 implement, 採用的是 counting sort, 而 trie 上的節點存的 unorderer_set 換成是 vector<bool>, 把 id 存在 index 裡, 像是 vector[id]=true, 碰到 vector[id]的 size>原本的 size 時, 就做 resize 的動作。這樣 find 的部分只要直接取 vector[id]看他是否=true 來判斷他存不存在。因此，counting sort 的效率是$O(n+k)$, k 為最大數字,n 為個數, 比原本內建的 sort 快。而利用 vector<bool>做 find 它是不是存在只需要$O(1)$的時間, 比原本 unordered_set 的$O(\log n)$更快。
經過此修正的 code score 在 **138000~140000** 之間。

Phase10：

我修正 add 的部分, 把每次 add 進來的 path(例如：add ./data/mail278)存在 trinode 裡面, 而同時我會在 node 多開一個 int 型態存加進來的 mail id, 因此我若將此一封信刪除後又加進來時, 我只需去看 node 上這個 path 是否有記錄下來, 若有的話我就可以直接印出結果不需考慮加信的動作, 因為我的 trinode 在 remove 的時候並不會真的刪掉原先該信存進來的 word；若這個 path 沒有記錄過, 則我才須將這封信的內容跟這個 path 存進 trinode 裡。另外, 每個 word 在 trie 的位置我用一個 trinode* where 存起來, 這樣我在找這個 keyword 的時候我不需從 trinode 的 root 開始跑起, 只需檢查 where 所對應到的位置只有那些 id number。
經過此修正的 code score 在 **188000~189500** 之間

Phase 11：

這次我加入了 from trie, to trie, 也就是另外建兩個以 from 名字所構成的字典樹和以 to 名字所構成的字典樹, 同樣的 add mail 的時候把每次要加 from name 和 to name 分別存到 from trie 和 to trie 裡, 把 id push_back 到節點上。因此每個名字上都有這個名字出現在哪些 id 的 vector。現在的 filter 就分成四個部分，沒 from to 限制的 search 一樣是 word filter 第一個取最小 id 數量的 word, 然後只有 from 的限制是取此 from name 在 from trie 上有哪些 id 再做 search 檢查的動作，寫在 from filter 而只有 to 的限制是取此 to name 在 to trie 上有哪些 id 再做 search 檢查的動作, 寫在 to filter。最後有 from 限制又有 to 限制的 search 我跑 from trie 也跑 to trie 再判斷哪個的 id 總數少, 就用他來做

filter 去 search 檢查其他的 keyword,是寫在 fromto filter。因此 from trie 和 to trie 的 id 數量有時會比最少 id 數量的 word 還少，比原單純的 word filter 更快。經過此修正的 code score 在 **192000~193000** 之間。

Phase12：

原先我存需印出的 id 是用一個 vector<int>ans 存起，然後再經由 counting sort 去調整大小順序，但在 counting sort 的過程中，免不了要跑一層 mail 數量的最大值的 for 迴圈。但我把 counting sort 的部分刪掉，改而採用兩個 int 型態的變數 maxnum, minnum 分別去紀錄要印出來的 id number 的上下界，因此我跑 for 迴圈的時候只要跑 [minnum, maxnum] 即可，此做法只有在 worst case 的時候, time complexity 才為 mail 數量的最大值。

最終的 code score 達 **203150**

(3) Bonus

A. 比較三個 STL：set、unordered_set、vector 存 subject 跟 content 的優缺點
   1. set 的根基是 binary search；unordered_set 的根基是 hash
   2. set 以及 unordered_set 的優點在於不會存 subject、content 重複的 word，因此若整篇文章有大量的資料是重複的，用 set 跟 unordered_set 去 implement 會省下不少空間；至於 vector 對於重複資料若不做些許修正，會重複存到，故對於大量重複的資料, vector 會浪費許多不必要的 space。
   3. vector 在 insert 時為$O(1)$，而 set 和 unordered_set 在 insert 時,會去檢查此 element 是否已存在來決定是否要加入，而且 set 還會在 insert 同時做 sort 的動作，故時間不會是$O(1)$；在 find 的時候，因為 set 的根基是 binary bearch tree，所以 find 的 average time 為 $O(\log n)$，而 vector 若用 iterator 去 find 所花上的 time complexity 為$O(k)$ (k 為 iterator 所跑的個數)，而 unordered_set 在 find 方面, average speed 為 constant, worst case 為 linear of container size

B. 比較第一個演算法的字典樹 trie 和第二個演算法的字典樹 Ternary search tree 的優缺點
   1. trie 是每一層有 77 個(以 ascii 碼表示的)，也就是所每當有更長的單字插入時，又占了更多的 77 個 node，無論是否有其他的字元存在,浪費相當多空間。好處是我在搜尋單字時就是$O(n)$, n 為字串長度，速度非常的快。而第二個的 Ternary search tree 是往下分成三個分枝，而每個節點會有一個字元，大於這個字元(也就是 ascii 碼的比較)就插入左子樹，小於插入右子樹，等於就插到中子樹，因此基本上 Ternary search tree 每個 node 最多只會有三個 child，而且是有插入碰到沒有的字元才會往下，而不像是 trie 每碰到更長的單字就必須新開 77 個

node，使得有的是空 node，節省相當多的空間。

2. 但是在速度上就比不上 trie 了，並不是$O(n)$, n 為字串長度，而有可能會更長，因為每一層並不含有 77 個 node，而是一直往下直到等於才換下一個字元。總而言之，trie 搜尋快但是佔記憶體空間過多，而 Ternary search tree 則是搜尋慢但是節省相當多的記憶體空間。
3. 我推薦的還是 trie，這是完全私心的想要追求速度的極致，顯然地浪費了相當多的空間，而其實 Ternary search tree 也是會有 height 過長的問題，所以還是選擇快速 height 短的 trie。

C. 比較 addmail filter, word filter, longfisrt wordfilter, hasidminfirst filter, 和 from filter, to filter 的優缺點

1. 最早的 addmail filter，是將每個信的內容存在 class mail 裡的 content 裡，在將每個 keyword 都去搜尋是否存在於 class mail 的 content 裡，速度非常緩慢。而第二種 wordfilter，加入了 trie 的概念，搜尋不再是從 class mail 裡找而是去從一個巨大的 trie 節點找尋是否存在 id，沒有就提早 return，減少暴搜每封 addmail 信的時間。第三種 longfirst word，則是將 key word 做越長的單字擺越前面，我猜測最長的單字含有的 id 會越少，在把第一個 keyword 當做基底有哪些 id，去搜尋其他剩下的 keyword 有沒有這些 id，沒有也提早 return。這樣過濾的速度又更進化了，不再管現在的 addmail 總數，直接把最長的字串當基底去檢查其他 keyword。第四種的 hasidminword filter，也就是直接把最少 id 數量的 word 擺在第一個， 第二少第二個...，也就是做以 id 數量為 compare 的 sorting，這樣比前一個 filter 又更實際，真的按照 id 數量做排序，排序後第一個 keyword 會有最少的 id 數量，再去搜尋第二少的 keyword，也更容易提早 return，是目前為止最快的 filter。最後一個 from filter，和 to filter 是考慮有可能會有很少出現的 from 人名，或者 to 人名，這樣他的 id 數量有機會比 hasidminword 更少，這樣以他們做基底去從第 1 個第二個 word...到最後一個 word 去搜尋，也就是在 search 的時候分成四個 case，沒 from 沒 to 限制的依然用 hasidminfirstword filter，只有 from 限制就用 from filter 並判斷和第一個 keyword 比誰 id 數量少就以誰為基底，只有 from 限制就用 to filter 並判斷和第一個 keyword 比誰 id 數量少就以誰為基底，而有 from 限制也有 to 限制，就判斷 from filter 和 to filter 和 第一個 keyword 比誰 id 數量少就以誰為基底，能夠更精準的求出最少 id 數量做 filter，速度也相對的提升了。
2. 第一個 addmail filter 的好處就是他依然保留最一開始的哪些 mail 存了哪些 content，保留最原始的資料，沒使用 trie 也保存了相當的空間(trie 會有空 node)。第二個 wordfilter 則是少了 sorting word 的時間，假使 word 的 size 不大或者排序不亂，那第二個有可能是更好的演算法。

而第四個完全是第三個加強版，直接以 id 數量作 sorting，在 keyword 很多而且很亂的情況下 sorting 後能增益相當快的時間。缺點是第二個,第三個,第五個都需要另外開一個 trie 存 content,稍嫌浪費空間(有空 node 的問題)。最後一個 from filter 和 to filter，在 from name 和 to name 重複性很少的情況下會有最大的效益，但是同樣的我另外開了兩個 from trie 和 to trie，也就犧牲了更多的空間。

3. 我推薦的是 hasidminfirstword filter，因為他只要經過簡單的 hasidminfirst 的 sorting 以後就可以達到速度極快的效果，而加上 from filter 和 to filter 進步不是很明顯，也犧牲了相當多的空間，因此我首推只有 hasidminfirstword filter。

D. 額外的功能

1. Linebytime 的功能

比照一般的信箱，越晚寄過來的信會被排在越前頭，也就是說，信的排序是由距離現在時刻越近排到距離現在時刻越遠，因此我要模擬的正是此功能。我指令若讀到 "linebytime"，則我會把當下有 add 進來的信，依他們的時間先後順序排序，印出他們對應的 id，若信件被 remove，則不會印出。

例如：

Mail1836　Date: 23 March 2012 at 09:47

Mail5932　Date: 5 June 2012 at 16:42

Mail7811　Date: 26 March 2012 at 12:13

若今此三封信都 add 進來

Sample Input：

linebytime

remove 5932

linebytime

Sample Output：

5932 7811 1836

Mail 1 removed, you have 1 mails

7811 1836

2. Blacklist 的功能

比照黑名單的功能，可以將同一個人寄的所有信都刪掉。可以決定是 from 人，還是 to 人，將這個人的信件做刪除的動作。那麼以後的 search 就無法再 search 到這個人的信。找不到人名的話就 output"Cannot find XXX"，黑名單成功的話就 output"Blacklist from: XXX successful!"或"Blacklist to:　XXX successful!"。

Sample1:

Mail903,mail3241,mail8562 都是 from Diana 的信。

(in DSA's sample_huge)

input:

add ./data/mail903

add ./data/mail3241

add ./data/mail8562

search From Diana To - Before - After - his in the

blacklist from Diana

blacklist from Andy

search From Diana To - Before - After - his in the

blacklist from Diana

output:

Mail 903 added, you have 1 mails

Mail 3241 added, you have 2 mails

Mail 8562 added, you have 3 mails

903 3241 8562

Blacklist from: Diana successful!

Cannot find Andy

-1

Cannot find Diana

Sample2:

Mail444,mail3746,mail3992,mail4923,mail7414,mail8327,mail8810 都是 to Iris 的信。

(in DSA's sample_huge)

Input:

add ./data/mail444

add ./data/mail3746

add ./data/mail3992

add ./data/mail4923

add ./data/mail7414

add ./data/mail8327

add ./data/mail8810

search From - To Iris Before - After - a

blacklist to Diana

blacklist to Iris

search From - To Iris Before - After - a

blacklist to Shelley

output:

Mail 444 added, you have 1 mails
Mail 3746 added, you have 2 mails
Mail 3992 added, you have 3 mails
Mail 4923 added, you have 4 mails
Mail 7414 added, you have 5 mails
Mail 8327 added, you have 6 mails
Mail 8810 added, you have 7 mails
444 3746 3992 4923 7414 8327 8810
Cannot find Diana
Blacklist to: Iris successful!
-1
Cannot find Shelley

3. lineby from& lineby to 的功能

可以將同一個 from 人或者 to 人的信，按照 id 順序 output 出來，被刪掉或者黑名單的信件就不會出現，會 output”Cannot find XXX”。輸入的形式為”lineby from XXX”或者”lineby to XXX”。

Sample1:

Mail903,mail3241,mail8562 都是 from Diana 的信。
(in DSA's sample_huge)
Input:
add ./data/mail903
add ./data/mail3241
add ./data/mail8562
lineby from Andy
lineby from Diana
remove 903
lineby from Diana
blacklist from Diana
lineby from Diana
Output:
Mail 903 added, you have 1 mails
Mail 3241 added, you have 2 mails
Mail 8562 added, you have 3 mails
Cannot find Andy
903 3241 8562
Mail 903 removed, you have 2 mails
3241 8562
Blacklist from: Diana successful!

Cannot find Diana

Sample2:

Mail444,mail3746,mail3992,mail4923,mail7414,都是 to Iris 的信。

(in DSA's sample_huge)

Input:

add ./data/mail444

add ./data/mail3746

add ./data/mail3992

add ./data/mail4923

add ./data/mail7414

lineby to Jacky

lineby to Iris

remove 3746

lineby to Iris

blacklist to Iris

lineby to Iris

Output:

Mail 444 added, you have 1 mails

Mail 3746 added, you have 2 mails

Mail 3992 added, you have 3 mails

Mail 4923 added, you have 4 mails

Mail 7414 added, you have 5 mails

Cannot find Jacky

444 3746 3992 4923 7414

Mail 3746 removed, you have 4 mails

444 3992 4923 7414

Blacklist to: Iris successful!

Cannot find Iris