Wordfilter 測試執行與分析

此為 C++撰寫的文字過濾器，以一個名為”filter.txt”文件檔案作為需過慮之關鍵字文庫進行文字過濾之功能，並在 exe 執行檔中以輸入 1~3 來選擇操作功能，1 為新增過濾字進入到”filter.txt”文件中，2 為刪除”filter.txt”文件中的過濾字，3 為執行輸入需進行過濾之句子。

Filter.txt 文件為題目所要求之 test case，其內涵網路所搜索之大陸敏感字、髒話、自訂義之文字以及約 139K 個 20 萬-40 萬隨機之數字串，約 140K 個不完全重複之過濾字，由於在字串變數宣告時不得大於 1M 大小之限制，使得文件最大僅能容下 1M 之容量，最大包含約 150K 之過濾字，故僅放入 140K 過濾字以免程式崩潰。

測試用句子.txt 是用來測試程式執行所使用的句子文件，僅含 10 句涉及一些敏感字與特定文字的文件，不同句子分別測試連續敏感字，中英混雜，純英句子不同狀況下其文字過濾器的效能狀況。

- 測試執行

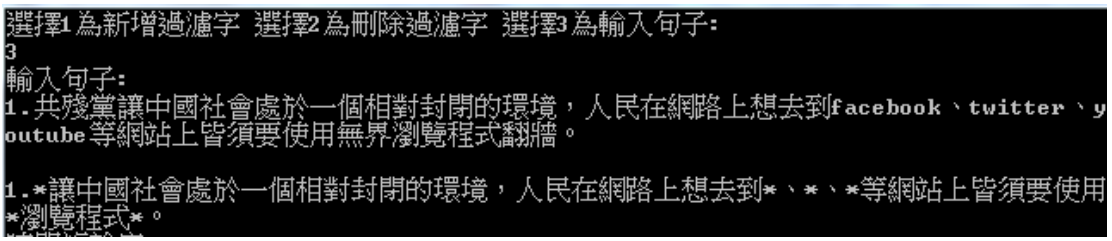

圖 1 文字過濾執行

如圖 1 所示在程式執行時，會先要求使用者選擇執行之動作，選擇 3 後便可以輸入自己想要輸入之文字，之後程式便會過濾使用者所輸入句子中的敏感字，並以[*]來顯示，最後輸出為敏感字已被過濾的句子。

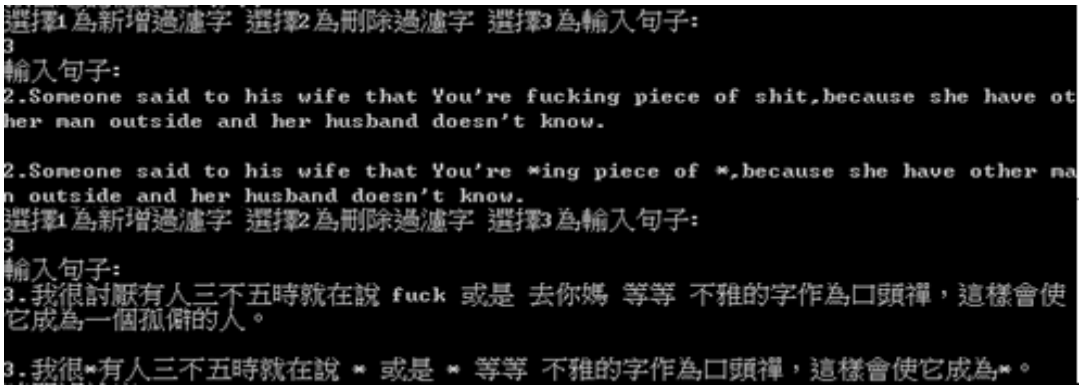

圖 2 純英與中英混雜句子過濾執行

在圖 2 中分別連續輸出一個純英文句子以及中英混雜之句子進行文字過濾器的測試，從結果可以得知即使在英文型態變化後依然可以把關鍵敏感字給過濾掉，以及能夠在中英混雜句子中把所有敏感字過濾，以此驗證其文字過濾器能夠在中英環境中執行。

- 效能分析與測試結果

```
wstring wsInput = input;                                          //將字串使用wstring進行處理
wstring::size_type uPos = 0;                                      //宣告欲過濾字之定位
vector<wstring> foo;
foo.reserve(150000);
if(wsInput.size()!=0 && wcslen(filter)!=0 && pch!=NULL){
    pch = wcstok(filter,L" ");                                    //將過濾文庫之字串以空白進行切割
    while(pch !=NULL){
        foo.push_back(pch);                                       //將切割後的過濾字放入記憶體陣列中
        pch = wcstok(NULL,L" ");
        m++;
    }
    vector<wstring>::iterator irr;
    for (irr = foo.begin();irr!=foo.end();irr++){                 //過濾字陣列掃描
        if(wsInput.length() >= (*irr).length()){                  //當輸入之句子長度大於過濾字長度時執行下列動作
            uPos = wsInput.find(*irr);                            //將輸入之句子中第一個與過濾字相同的文字位子紀錄並作為定位點
            while(uPos != wstring::npos){                         //在定位點不等於尾端位置時持續執行
                wsInput.replace(uPos,(*irr).length(),repl);       //利用定位點將欲過濾之句子過濾掉
                uPos = wsInput.find(*irr);                        //在句子中持續尋找重複之欲過濾字
                m++;
            }
            m++;                                                  //將搜尋之步驟累計作為時間複雜度之評估
        }
    }
}
```

圖 3 文字過濾部分程式碼

在設計此文字過濾器時為了處理包含大量文字之文件，利用 C++語法中 STL 型態字串去處理，在處理過程中必定需要將一份文件分割成單一過濾字並分別存進記憶體個別位置來形成一個陣列，並透過搜索該陣列去尋找句子中的敏感字，如圖 3 所示；此時的時間複雜度為將訊息分割時所需的次數加上每個關鍵字比對的次數再加上取代過濾文字時的次數之總和，為了驗證其想法使用一個 m 進行累計，並將其分別放進個別迴圈中，並在程式過濾後顯示其計算結果，如圖 4 所示。

```
選擇1為新增過濾字 選擇2為刪除過濾字 選擇3為輸入句子:
3
輸入句子:
1.共殘黨讓中國社會處於一個相對封閉的環境，人民在網路上想去到facebook、twitter、y
outube等網站上皆須要使用無界瀏覽程式翻牆。

1.*讓中國社會處於一個相對封閉的環境，人民在網路上想去到*、*、*等網站上皆須要使用
*瀏覽程式*。
時間複雜度282776

進行運算所花費的時間：0.073 S
實際花費記憶體空間141385
預留之記憶體空間大小150000
```

圖 4 文字過濾執行與效能顯示

在圖 4 中所顯示句子的輸入與過濾後結果，並顯示其時間之複雜度與整個文字過濾所需花費之時間以及該文字過濾器所花費之記憶體大小，接下來會連續執行 10 次不同句子輸入，來觀察其各項數據之變化來驗證。

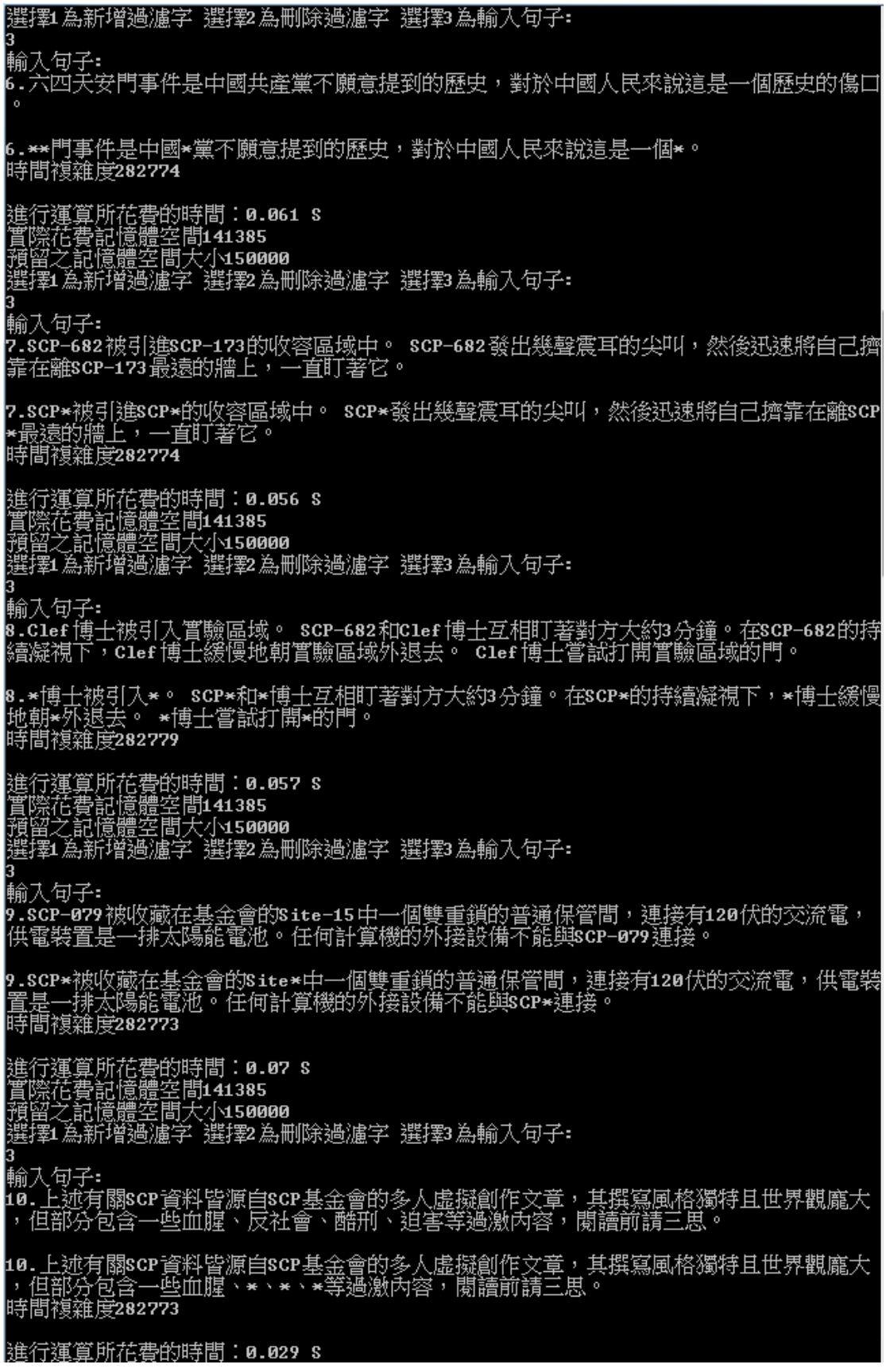

圖 5 連續執行文字過濾之部分圖

圖 5 為輸入 10 次不同句子後分別顯示之結果，由於將結果全數放入會使圖片過大，因此只放部分結果，在後面會把個別的時間複雜度與花費時間統整並以表格所示。

表 1　10 次不同句子執行之效能表單

	時間複雜度	運算花費時間	時間複雜度-文件切割	時間複雜度-過濾字搜索	時間複雜度-文字取代
第一次	282776	0.073	141385	141385	6
第二次	282772	0.071	141385	141385	2
第三次	282774	0.038	141385	141385	4
第四次	282777	0.052	141385	141385	7
第五次	282787	0.028	141385	141385	17
第六次	282774	0.061	141385	141385	4
第七次	282774	0.056	141385	141385	4
第八次	282779	0.057	141385	141385	9
第九次	282773	0.070	141385	141385	3
第十次	282773	0.029	141385	141385	3

由表 1 可以看到其時間複雜度大部分都是在切割資料與過濾文字上，而其餘的部分則在於文字取代上，也可觀察到基本上其複雜度的大小取決於 filter.txt 的過濾字的多寡，而運算花費時間也可以在表中觀察到，每次執行時間並不冗長，而是非常快速的進行，從此可以看出將資料以 STL 形態處理時的速度。

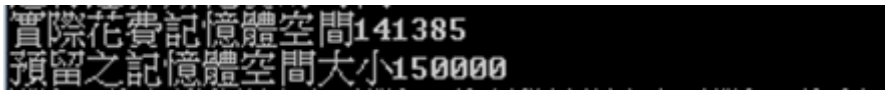

圖 6　vector 花費之記憶體大小

為了存取大量資料不僅使用了長度為 1M 的字串，並且使用 vector(STL)之陣列儲存資料，而圖 6 所示，即為 vector 所實際使用的空間與預留空間的大小，整個文字過濾器的使用都以該陣列為基礎進行運算，故為主要消耗之記憶體空間。

- 新增與移除過濾字

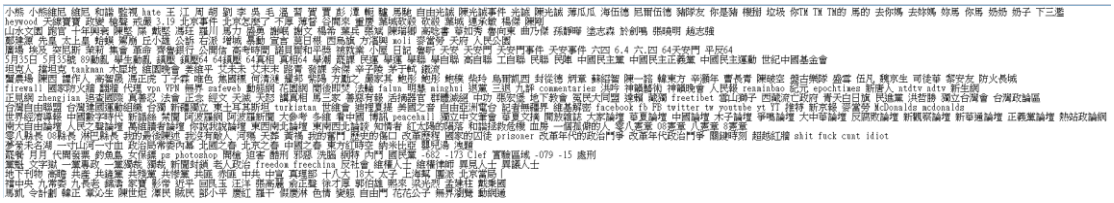

圖 7　filter.txt 內容

圖 7 為 filter.txt 之部分內容，新增與移除之功能便是處理 filter.txt 的動作。

圖 8 加入過濾字之執行

在圖 8 中選擇要增加過濾字之後會將輸入之文字加入 filter.txt 文件中之最尾端如圖 9 所示。

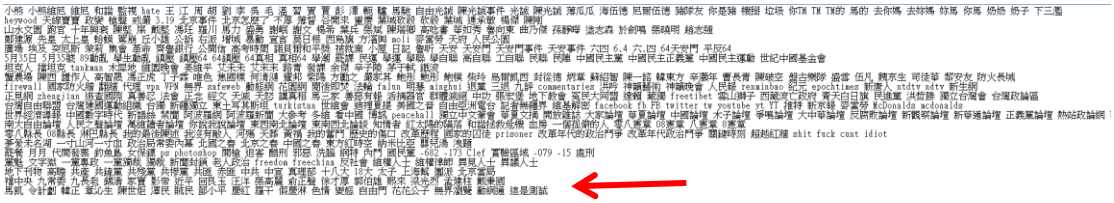

圖 9 加入新過濾字後之 filter.txt 內容

之後進行測試新加入之過濾字後的狀態，如圖 10 所示。

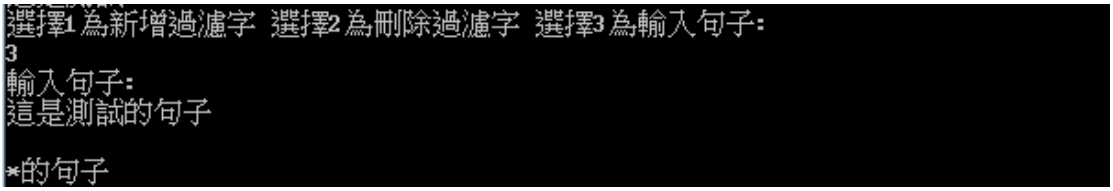

圖 10 測試新過濾字之執行

之後欲刪除過濾字可以選擇刪除過濾字來進行，如圖 11 所示。

圖 11 刪除過濾字之執行

在圖 11 中輸入欲刪除的過濾字可以使 filter.txt 文件中的過濾字刪除掉，如圖 12 所示。

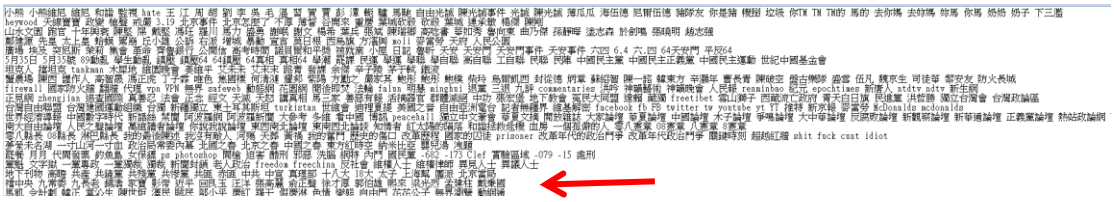

圖 12 刪除過濾字後之 filter.txt 內容

- 結論

此文字過濾器設計是以 filter.txt 作為文字過濾的基礎，將輸入的句子裡所有敏感字進行過濾，來獲取一個處理過的句子；其中使用了中文、英文、中英混雜之句子來測試其結果都沒問題，並連續使用 10 句不同的句子來分析其效能差異並列表顯示，最後介紹其新增與刪除過濾字之功能使其更為容易設定 filter.txt 的內容。