國立臺南大學資訊工程學系

資工三「演算法」課程
第四次作業

題目:Edit Distance

班級 ： 資工三

姓名 ： 呂益銓

學號 ： S10659013

老師：陳宗禧

中華民國 108 年 12 月 24 日

目錄

(一) 簡介及問題描述

1. 簡介與問題

[Goal]: 將一篇文章作一編輯後更正

Input: 一個字典(說明格式)及一篇文章

Output: i. BK-tree 字典 (Dic1, Dic2, Dic3)

ii. 根據 BK-tree 字典更正後的文章

iii. 分析

i. 根據自行收集的英文單字建立字典(至少 200 英文單字) 如 : 動 物 名 稱 (https://tw.blog.voicetube.com/archives/44177) 、 國 家 城 市 名 稱 (https://tw.blog.voicetube.com/archives/39547)，可自行設定主題。撰寫建構 BK-tree 作 search

ii. Edit distance：實作三種 Edit Distance a. 傳統 Edit Distance: Substitution 設為 1 b. Levenshtein Edit Distance: Substitution 設為 2 c. Damerau-Levenshtein Distance (https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance)

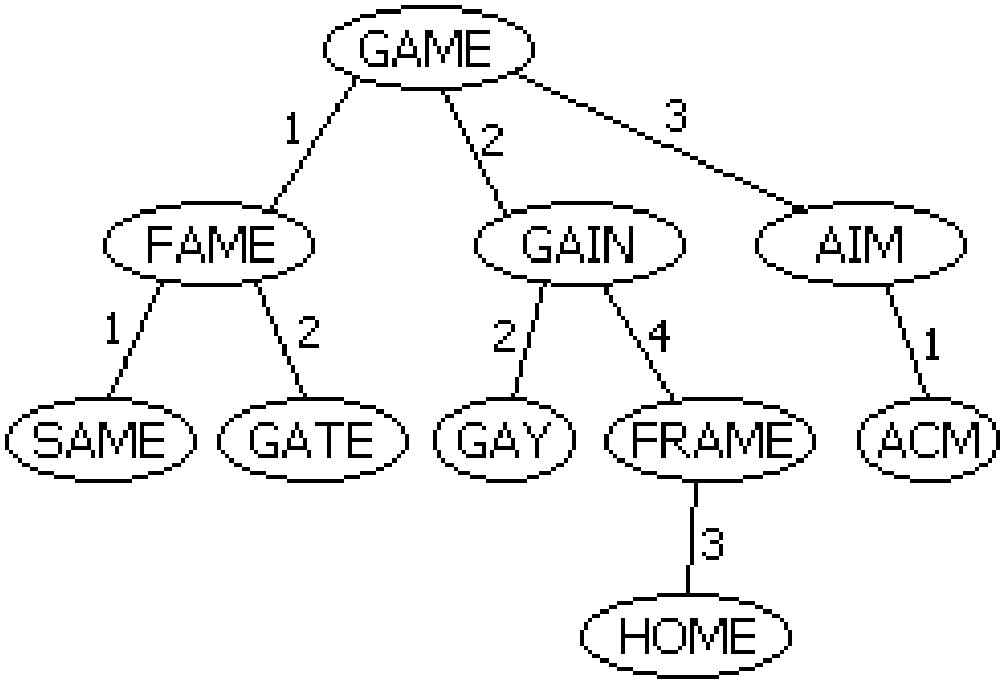
iii. 比較三種編輯距離的差異

iv. 比較三種編輯距離所建構的 BK-tree (要說明 BK-tee 存在檔案的結構)以及搜尋 字串的比較次數(錯誤容忍值) v. 輸出更正錯誤的字數(Word)與英文字(Character)的數量，以及整篇文章字數 (Words)和字元數(Characters)的錯誤率

(二) 理論分析

i. BKtree

每一個 node 都有一個 word(string)和多個 child 每一個 child 為一個 struct 存有 distnace 和一個 node 的指針，利用這種方法建立出一個節點有多個 child 的 tree，多個 child 利用紅黑樹來存以便搜尋時可以快一點。
建立完成後先設定 root 的 word，之後再依序將剩下來的字 insert 進去，如果有一樣的 distance 就往 child node 往下搜尋直到找到 external node，如果沒有一樣的 distance 就 insert 的 child 的 distance 是利用 edit distance 下去計算出來的，node 的指針則新增一個 nod 裡面的 child 為空 word 為剛剛 insert 的字，



ii. 傳統 Edit Distance

傳統的 edit distance 是利用 insertions, deletions 和 substitution，每個 insertions 都增加 1，每個 deletions 都增加 1，每個 substitution 都增加 1，再利用 dynamic programming 的方法來實作，如果是用傳統 recursive 的方法時間複雜度會到指數的等級，明顯不切實際利用填表格的方式來實作，時間複雜度則到 O(mn) 快很多。

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

iii. Levenshtein Distance

Levenshtein Distance和剛剛的distance基本上一樣只差在substitution的operation是+2而不是+1。

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 2_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

iv. Damerau–Levenshtein distance

Damerau–Levenshtein distance 和傳統 edit distance 也是一樣，只差別在多出一個 operation 為 transposition，當有兩個字母一樣時就可以交換，如:ac 和 ca，這兩個的 distance 為 1，因為只要 ac 和 ca 交換就一樣了。

$$d_{a,b}(i,j) = \min \begin{cases} 0 & \text{if } i = j = 0 \\ d_{a,b}(i-1,j) + 1 & \text{if } i > 0 \\ d_{a,b}(i,j-1) + 1 & \text{if } j > 0 \\ d_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} & \text{if } i,j > 0 \\ d_{a,b}(i-2,j-2) + 1 & \text{if } i,j > 1 \text{ and } a[i] = b[j-1] \text{ and } a[i-1] = b[j] \end{cases}$$

(三) 演算法則

1. Edit Distance

Algorithm

```
input: strings a[1..length(a)], b[1..length(b)]
output: distance, integer

let d[0..length(a), 0..length(b)] be a 2-d array of integers, dimensions length(a)+1,
length(b)+1
// note that d is zero-indexed, while a and b are one-indexed.

for i := 0 to length(a) inclusive do
      d[i, 0] := i
for j := 0 to length(b) inclusive do
      d[0, j] := j

for i := 1 to length(a) inclusive do
      for j := 1 to length(b) inclusive do
            if a[i] = b[j] then
                  cost := 0
            else
                  cost := 1
            d[i, j] := minimum(d[i-1, j] + 1,          // deletion
                               d[i, j-1] + 1,          // insertion
                               d[i-1, j-1] + cost)     // substitution
  return d[length(a), length(b)]
```

時間複雜度(time complexity)

O(mn)

空間複雜度(space complexity)

O(mn)

2. Levenshtein Distance

Algorithm

```
input: strings a[1..length(a)], b[1..length(b)]
output: distance, integer

let d[0..length(a), 0..length(b)] be a 2-d array of integers, dimensions length(a)+1,
length(b)+1
// note that d is zero-indexed, while a and b are one-indexed.

for i := 0 to length(a) inclusive do
    d[i, 0] := i
for j := 0 to length(b) inclusive do
    d[0, j] := j

for i := 1 to length(a) inclusive do
    for j := 1 to length(b) inclusive do
        if a[i] = b[j] then
            cost := 0
        else
            cost := 2
        d[i, j] := minimum(d[i-1, j] + 1,          // deletion
                           d[i, j-1] + 1,          // insertion
                           d[i-1, j-1] + cost)     // substitution
  return d[length(a), length(b)]
```

時間複雜度(time complexity)

O(mn)

空間複雜度(space complexity)

O(mn)

3. Damerau–Levenshtein distance

```
input: strings a[1..length(a)], b[1..length(b)]
    output: distance, integer

    da := new array of |Σ| integers
    for i := 1 to |Σ| inclusive do
        da[i] := 0
```

```
    let d[−1..length(a), −1..length(b)] be a 2-d array of integers, dimensions length(a)+2,
length(b)+2
    // note that d has indices starting at −1, while a, b and da are one-indexed.

    maxdist := length(a) + length(b)
    d[−1, −1] := maxdist
    for i := 0 to length(a) inclusive do
        d[i, −1] := maxdist
        d[i, 0] := i
    for j := 0 to length(b) inclusive do
        d[−1, j] := maxdist
        d[0, j] := j

    for i := 1 to length(a) inclusive do
        db := 0
        for j := 1 to length(b) inclusive do
            k := da[b[j]]
            ℓ := db
            if a[i] = b[j] then
                cost := 0
                db := j
            else
                cost := 1
            d[i, j] := minimum(d[i−1, j−1] + cost,    //substitution
                               d[i,     j−1] + 1,         //insertion
                               d[i−1, j    ] + 1,         //deletion
                               d[k−1, ℓ−1] + (i−k−1) + 1 + (j-ℓ−1)) //transposition
        da[a[i]] := i
  return d[length(a), length(b)]
```

時間複雜度(time complexity)

O(mn)

空間複雜度(space complexity)

O(mn)

(四) 程式設計環境架構

程式設計語言、工具、環境與電腦硬體等規格說明...

1. 程式語言

C in MS Windows

2. 程式開發工具

Visual Studio Code + MinGw64

3. 電腦硬體

CPU: Intel i5-9400f

Main Memory: 16GB

Disk:1T HDD

500G SSD

250G SSD

OS: W10 x64

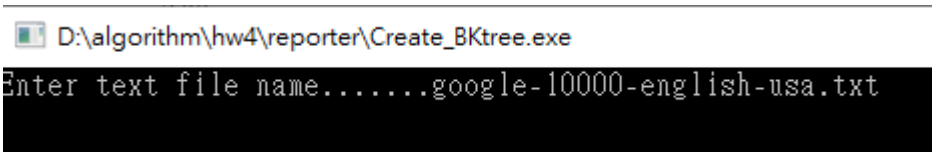
(五) 程式 (含 source code, input code, and output code)

程式含 source code, input code, and output code 等…

1. 主程式

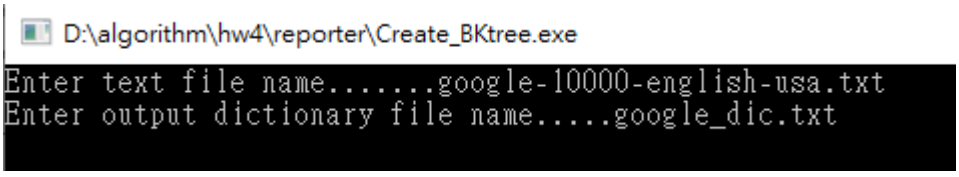
Create BKtree

1.先輸入檔名

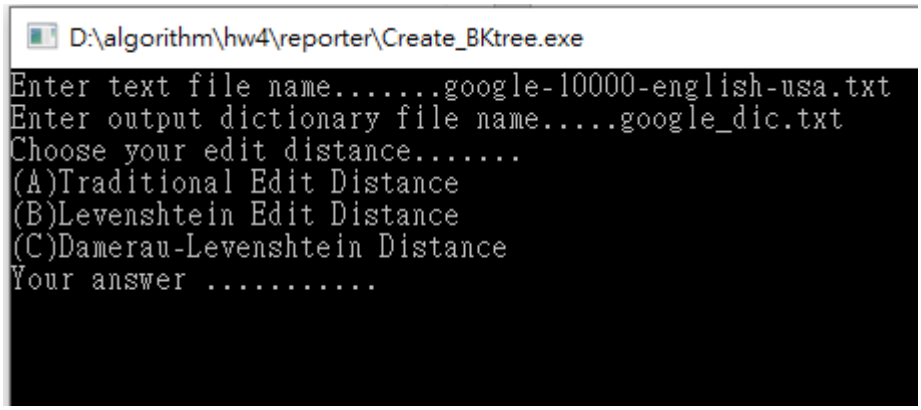


檔案格式

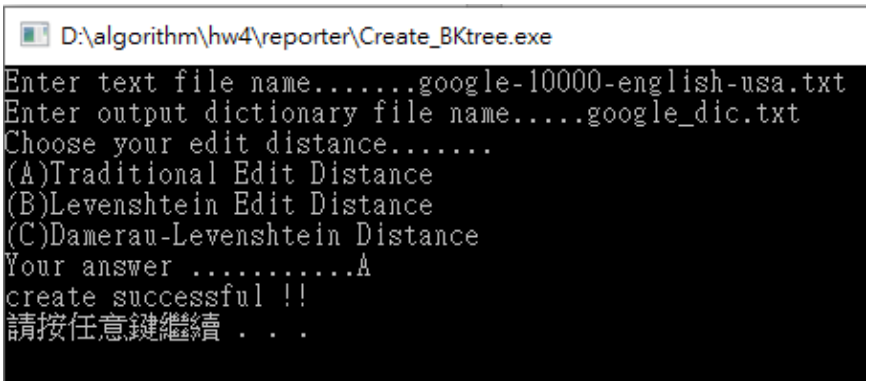
2.再輸入要儲存的字典名稱



3.在選擇建立樹的時候要使用的 distance 種類



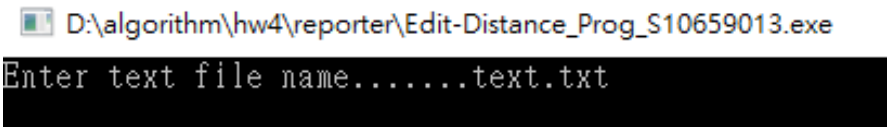
4.建立成功



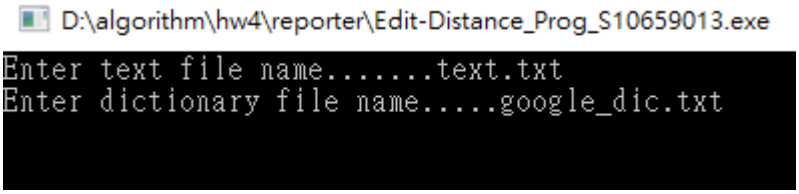
輸出後的檔案格式

Edit-Distance_Prog_S10659013

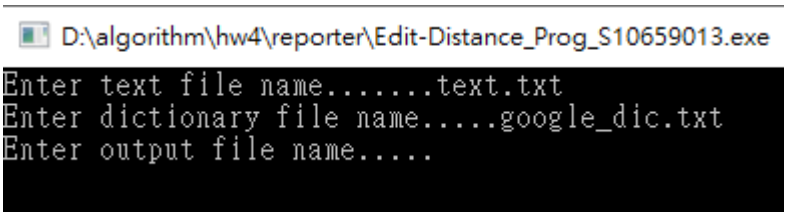
1.先輸入檔名



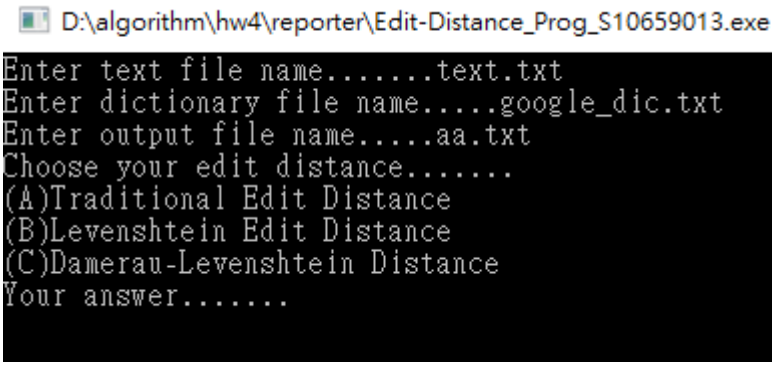
2. 再輸入字典的 file name



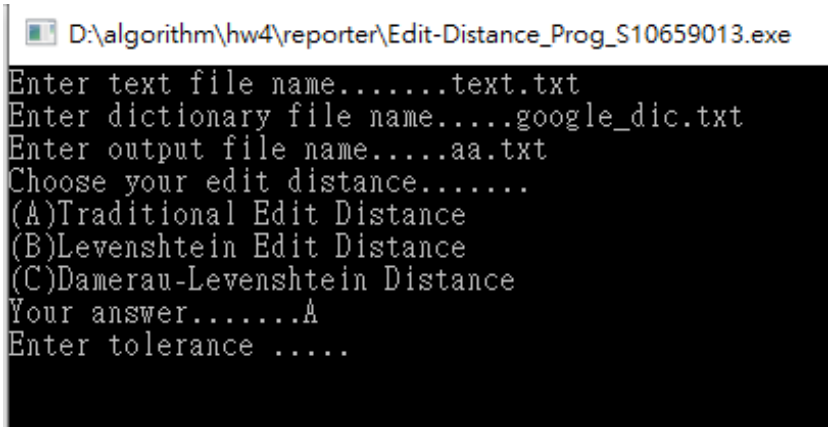
3. 再輸入要輸出更正後的 filename



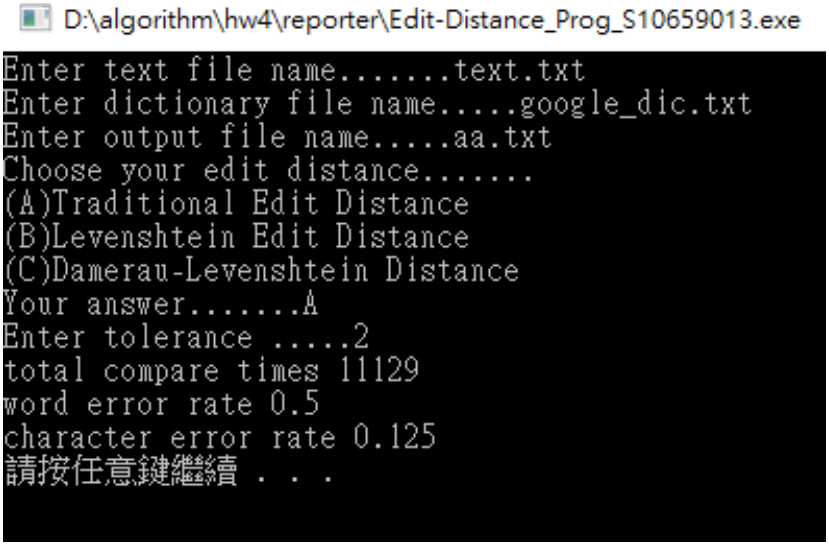
3.要使用的 distance 種類(要和剛剛建立的字典 distance 種類一樣)



4. 再輸入容忍值如果找到容忍值以下的最小值(不為 0)則會將單字修改成找到的字



5. 輸出總共比較的次數與 word error rate 與 character error rate



(五) 執行結果、討論與心得

1. 比較三種編輯距離的差異

同一個字串傳統的 distance 比 Levenshtein Distance 算出來的距離還要短因為 Levenshtein Distance 的 substitution 為 2 而傳統的為 1，而 Damerau－Levenshtein distance 可以會比剛剛傳統 distance 還要短或是一樣，因為 Damerau－Levenshtein 會增加一種 operation 為 transposition(交換)所以出來的距離才會小於或等於傳統的 distance

Example:ca 和 abc Levenshtein Distance 為 3 ca->a->ab->c

Damerau－Levenshtein 為 2 ca->ac->abc

同一字串比較 Damerau－Levenshtein <= 傳統的 distance <= Levenshtein Distance

2. 比較三種編輯距離所建構的 BK-tree（要說明 BK-tee 存在檔案的結構)以及搜尋字串的比較次數(錯誤容忍值)

BK-tee 存在檔案的結構:

由 node 所組成的 multiway search tree ，child 是可變的所以可以一直加下去。

使用 BFS 來儲存進 file

```
typedef struct node
{
        string word;
        map<int, node*> child;
} node;
```

```
the 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
they 1 2
to 1 2 3 4
of 1 2 3 4 5 6
from 1 2 3 4 5 6 7
about 2 3 4 5 6 7 8
search 2 3 4 5 6 7 8 9
business 4 5 6 7 8 9 10
products 2 3 4 5 6 7 8 9 10 11
university 5 6 7 8 9 10 11 12
information 4 5 6 7 8 9 10 11 12
professional 6 7 8 9 10 11 12 13
international 3 4 5 6 7 8 9 10 11 12 13
administration 3 7 8 9 10 12 13 14
responsibility 3 10 11 12 13 14
sublimedirectory
telecommunications
them 1 2
he 1 2
top 1 2
```

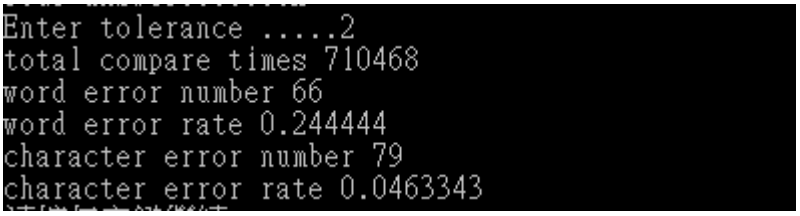
root Node 的 word
和 child 的 distance

剛剛 root 第一個 child 的 word
和他所包含 child 的 distance

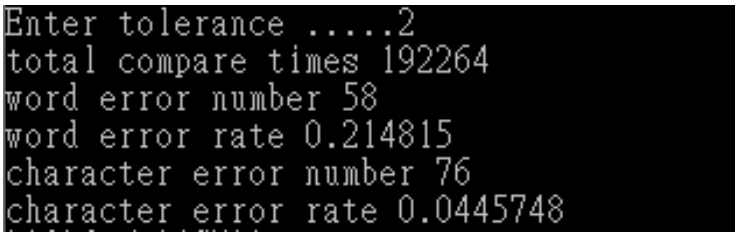
第二個 child 的 word 和他所包含 child 的 distance

三種 BKtree 來更正同一文章:

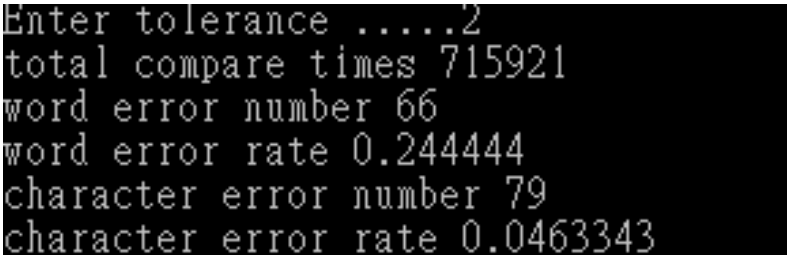
1.traditional:



2. Levenshtein Distance



3. Damerau–Levenshtein



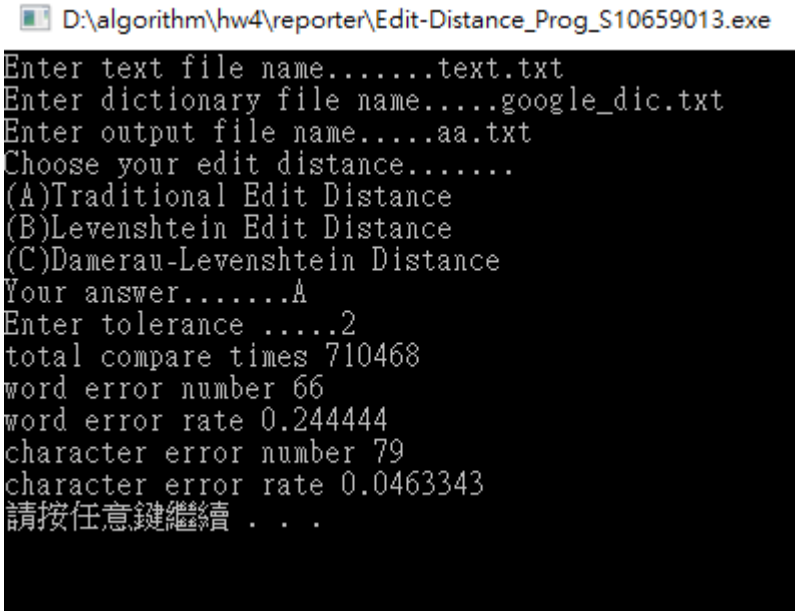
Levenshtein Distance 因為距離最大所以找出來的字最少所以比較次數最少，其他的蒐集出來的字差不多所以比較次數差不多

容忍錯誤值:當容忍錯誤值增加時所搜尋的字數變多所以 total compare times 會增加還有可能因為搜尋的字數增加所以原本搜不到的字也被蒐集出來了所以導致錯誤率增加。

3. 輸出更正錯誤的字數(Word)與英文字(Character)的數量，以及整篇文章字數 (Words)和字元數(Characters)的錯誤率

字元錯誤率算法:(total changed word number)/total word number

字數錯誤率算法:(total edit distance)/ total characters


D:\algorithm\hw4\reporter\Edit-Distance_Prog_S10659013.exe
Enter text file name.....text.txt
Enter dictionary file name.....google_dic.txt
Enter output file name.....aa.txt
Choose your edit distance.....
(A)Traditional Edit Distance
(B)Levenshtein Edit Distance
(C)Damerau-Levenshtein Distance
Your answer.....A
Enter tolerance 2
total compare times 710468
word error number 66
word error rate 0.244444
character error number 79
character error rate 0.0463343
請按任意鍵繼續 . . .

參考文獻

[1]

[1] 台南大學 E-course 演算法課程 Algorithms Report Format