國立臺南大學資訊工程學系

資工三「演算法」課程

第4次作業

**題目: Plagiarism Detection based on Edit Distance**

|  |  |  |
| --- | --- | --- |
| 班級 | ： | 資工三 |
| 姓名 | ： | 周巧晴 |
| 學號 | ： | S10959042 |

老師：陳宗禧

中華民國 111年12月28日

# 目錄

1. **簡介及問題描述摘要….……………..…………………………………………3**
   1. **簡介…………….…………………………………………………………………………3**
   2. **問題………………………….……………………………………………………………3**
2. **理論分析….………………………..………………………………………5**
3. **演算法則….…………………………..……………………………………8**
4. **程式設計環境架構.………………………..…………………………………17**
5. **程式.…………………………………………..………………………………18**
6. **執行結果、討論與心得.………………………..……………………………31**

參考文獻……………………………………………………………………………38

**(一) 簡介及問題描述摘要**

設計與實作五個演算法，計算Fibonacci Number?….

1. 簡介

利用Edit Distance 實作一抄襲偵測(Plagiarism Detection)系統。

2. 問題

**Functions:**

i. 利用Dynamic Programming 實作Minimum Edit Distance 演算法，包含Minimum Edit Distances (一為Insertion, Deletion, Substitution 皆各為一個operation)以及Levenshtein Distance (Insertion, Deletion,皆各為一個operation，Substitution 為兩個operations)兩種距離定義。

a. Input: Strings S1 and S2 (利用Keyboard 輸入)

b. Output: Minimum Edit Distance (number) between *S*1 and *S*2

ii. 利用(i)實作一抄襲偵測(Plagiarism Detection)系統，包含兩種Minimum Edit Distances。

a. Input: 原始檔案(S) and 比對檔案(T) (利用讀檔，宜考慮空白、Tab、Enter 等值)

b. Output: 抄襲程度𝑑𝑆,𝑇(全部檔案以及各段抄襲程度)、段抄襲程度的變異數、段抄襲程度的標準差。

|  |  |  |
| --- | --- | --- |
| 方法 | **Input** | **Output** |
| (1) | 原始檔案(S) 比對檔案(T)  …  …  …  … | **兩篇文章抄襲程度：**  Where and are the number of characters in and , respectively, and is the edit distance between and . |
| (2) | TP2  TP3  d1m  d13  d12  d11  共m段  共n段  SPn  SP4  SP3  SP2  SP1  比對檔案(T)  原始檔案(S) | |  | | --- | | **兩段𝑺𝑷𝒊 and 𝑻𝑷𝒋文字抄襲程度：** |   For and .  TP1   |  | | --- | | **兩篇文章抄襲程度：** |   TPm   |  | | --- | | where 𝑘𝑖 = arg MAX1≤𝑗≤𝑚𝑑𝑖,𝑗 , for 1 ≤ 𝑖 ≤ 𝑛.  **Notes:** 要排除空白段，連續Tab or 空白。  Outputs:  a. 兩篇文章抄襲程度：  b. 輸出段抄襲程度的變異數   1. c. 輸出段抄襲程度的標準差 | |

**(二) 理論分析**

1. Minimum Edit Distance

i. 利用Dynamic Programming 實作Minimum Edit Distance 演算法，包含Minimum Edit Distances (一為Insertion, Deletion, Substitution 皆各為一個operation)以及Levenshtein Distance (Insertion, Deletion,皆各為一個operation，Substitution 為兩個operations)兩種距離定義。

a. Input: Strings S1 and S2 (利用Keyboard 輸入)

b. Output: Minimum Edit Distance (number) between *S*1 and *S*2

Input：Strings S1 and S2

Output：Minimum Edit Distance (number) between *S*1 and *S*2

假設S1的長度是n，S2的長度是m。

S1變成S2有三種方式：

1. Deletion
2. Insertion
3. Substitution

而S1變成S2需要做d次的Deletion或Insertion或Substitution，則稱S1和S2之間的edit distance是d。

D(i,j)被定義為字串S1[1..i]和S2[1..j]之間的edit distance。

而S1和S2之間的edit distance就是D(n,m)。

Minimum Edit Distances (一為Insertion, Deletion, Substitution 皆各為一個operation)：

D(0,j)=j，D(i,0)，

Levenshtein Distance (Insertion, Deletion,皆各為一個operation，Substitution 為兩個operations)：

D(0,j)=j，D(i,0)，

Time Complexity?

需計算填滿D(1..n,1..m)的n\*m陣列方能得到D(n,m)值，所以是O(nm)。

ii. 利用(i)實作一抄襲偵測(Plagiarism Detection)系統，包含兩種Minimum Edit Distances。

a. Input: 原始檔案(S) and 比對檔案(T) (利用讀檔，宜考慮空白、Tab、Enter 等值)

b. Output: 抄襲程度𝑑𝑆,𝑇(全部檔案以及各段抄襲程度)、段抄襲程度的變異數、段抄襲程度的標準差。

|  |  |  |
| --- | --- | --- |
| 方法 | **Input** | **Output** |
| (1) | 原始檔案(S) 比對檔案(T)  …  …  …  … | **兩篇文章抄襲程度：**  Where and are the number of characters in and , respectively, and is the edit distance between and . |
| (2) | TP2  TP3  d1m  d13  d12  d11  共m段  共n段  SPn  SP4  SP3  SP2  SP1  比對檔案(T)  原始檔案(S) | |  | | --- | | **兩段𝑺𝑷𝒊 and 𝑻𝑷𝒋文字抄襲程度：** |   For and .  TP1   |  | | --- | | **兩篇文章抄襲程度：** |   TPm   |  | | --- | | where 𝑘𝑖 = arg MAX1≤𝑗≤𝑚𝑑𝑖,𝑗 , for 1 ≤ 𝑖 ≤ 𝑛.  **Notes:** 要排除空白段，連續Tab or 空白。  Outputs:  a. 兩篇文章抄襲程度：  b. 輸出段抄襲程度的變異數   1. c. 輸出段抄襲程度的標準差 | |

方法(1)

Input：兩篇文章的txt檔，比對檔案S1(長度為n) and原始檔案S2(長度為m)

Output：兩篇文章的抄襲程度

將兩篇文章分別作為S1和S2建立D(0..n,0..m)，得知為D(n,m)，為比較長的文章的長度。

再用上述公式算出兩篇文章的抄襲程度。

Time Complexity?

需計算填滿D(1..n,1..m)的n\*m陣列方能得到D(n,m)值，所以是O(nm)

方法(2)

Input：兩篇文章的txt檔，原始檔案(長度為) and比對檔案(長度為)

Output：兩篇文章的段抄襲程度、兩篇文章抄襲程度、段抄襲程度的變異數、輸出段抄襲程度的標準差

將文章每一段(以enter作分隔)的開頭結尾空白或tab鍵都排除、把空白段排除、把連續空白和tab只留第一個。

原始檔案文章共a段，每段有ni個字元，，比對檔案文章共b段，每段有mj個字元，。

將比對檔案文章每段跟原始檔案文章每段作為TPi和SPj建立D(0.. mj, 0.. ni)，並得知所有=D(mj,ni)，為比較長的文章段的長度，以下面公式做計算。

找出所有i相等的最大值sum起來除以比對總文章段數量(如同下面公式)，就算出兩篇文章的抄襲程度了。

where 𝑘𝑖 = arg MAX1≤𝑗≤𝑚𝑑𝑖,𝑗 , for 1 ≤ 𝑖 ≤ 𝑛.

Time Complexity?

O()

**(三) 演算法則**

1. Minimum Edit Distance

i. 利用Dynamic Programming 實作Minimum Edit Distance 演算法，包含Minimum Edit Distances (一為Insertion, Deletion, Substitution 皆各為一個operation)以及Levenshtein Distance (Insertion, Deletion,皆各為一個operation，Substitution 為兩個operations)兩種距離定義。

a. Input: Strings S1(長度為n) and S2(長度為m) (利用Keyboard 輸入)

b. Output: Minimum Edit Distance (number) between *S*1 and *S*2

Input: Strings S1 and S2

Output: Minimum Edit Distance and Levenshtein Distance between *S*1 and *S*2

for i = 0 to s1.length() do

d1[i][0] = i

d2[i][0] = i

for i = 1 to s2.length() do

d1[0][i] = i

d2[0][i] = i

for i = 1 to s1.length() do

for j = 1 to s2.length() do

if (s1[i - 1] != s2[j - 1])

if (d1[i - 1][j - 1] <= d1[i][j - 1] && d1[i - 1][j - 1] <= d1[i - 1][j])

d1[i][j] = d1[i - 1][j - 1] + 1

else if (d1[i - 1][j] <= d1[i][j - 1] && d1[i - 1][j] <= d1[i - 1][j - 1])

d1[i][j] = d1[i - 1][j] + 1

else

d1[i][j] = d1[i][j - 1] + 1

if (d2[i - 1][j - 1] + 1 <= d2[i][j - 1] && d2[i - 1][j - 1] + 1 <= d2[i - 1][j])

d2[i][j] = d2[i - 1][j - 1] + 2

else if (d2[i - 1][j] <= d2[i][j - 1] && d2[i - 1][j] <= d2[i - 1][j - 1] + 1)

d2[i][j] = d2[i - 1][j] + 1

else

d2[i][j] = d2[i][j - 1] + 1

else

d1[i][j] = d1[i - 1][j - 1]

d2[i][j] = d2[i - 1][j - 1]

cout << "Minimum Edit Distances is " << d1[s1.length()][s2.length()] << endl

cout << "Levenshtein Distance is " << d2[s1.length()][s2.length()] << endl

1. 演算法時間複雜度(time complexity)

需計算填滿D(1..n,1..m)的n\*m陣列方能得到D(n,m)值，所以是O(nm)。

1. 演算法空間複雜度(space complexity)

需要存d(0..n,0..m)的陣列，所以是O(nm)。

ii. 利用(i)實作一抄襲偵測(Plagiarism Detection)系統，包含兩種Minimum Edit Distances。

a. Input: 原始檔案(S) and 比對檔案(T) (利用讀檔，宜考慮空白、Tab、Enter 等值)

b. Output: 抄襲程度𝑑𝑆,𝑇(全部檔案以及各段抄襲程度)、段抄襲程度的變異數、段抄襲程度的標準差。

|  |  |  |
| --- | --- | --- |
| 方法 | **Input** | **Output** |
| (1) | 原始檔案(S) 比對檔案(T)  …  …  …  … | **兩篇文章抄襲程度：**  Where and are the number of characters in and , respectively, and is the edit distance between and . |
| (2) | TP2  TP3  d1m  d13  d12  d11  共m段  共n段  SPn  SP4  SP3  SP2  SP1  比對檔案(T)  原始檔案(S) | |  | | --- | | **兩段𝑺𝑷𝒊 and 𝑻𝑷𝒋文字抄襲程度：** |   For and .  TP1   |  | | --- | | **兩篇文章抄襲程度：** |   TPm   |  | | --- | | where 𝑘𝑖 = arg MAX1≤𝑗≤𝑚𝑑𝑖,𝑗 , for 1 ≤ 𝑖 ≤ 𝑛.  **Notes:** 要排除空白段，連續Tab or 空白。  Outputs:  a. 兩篇文章抄襲程度：  b. 輸出段抄襲程度的變異數   1. c. 輸出段抄襲程度的標準差 | |

方法(1)

Input：兩篇文章的txt檔，比對檔案S1(長度為n) and 原始檔案S2(長度為m)

Output：兩篇文章的抄襲程度

for i = 0 to s1.length() do

d1[i][0] = i

d2[i][0] = i

for i = 1 to s2.length() do

d1[0][i] = i

d2[0][i] = i

for i = 1 to s1.length() do

for j = 1 to s2.length() do

if (s1[i - 1] != s2[j - 1])

if (d1[i - 1][j - 1] <= d1[i][j - 1] && d1[i - 1][j - 1] <= d1[i - 1][j])

d1[i][j] = d1[i - 1][j - 1] + 1

else if (d1[i - 1][j] <= d1[i][j - 1] && d1[i - 1][j] <= d1[i - 1][j - 1])

d1[i][j] = d1[i - 1][j] + 1

else

d1[i][j] = d1[i][j - 1] + 1

if (d2[i - 1][j - 1] + 1 <= d2[i][j - 1] && d2[i - 1][j - 1] + 1 <= d2[i - 1][j])

d2[i][j] = d2[i - 1][j - 1] + 2

else if (d2[i - 1][j] <= d2[i][j - 1] && d2[i - 1][j] <= d2[i - 1][j - 1] + 1)

d2[i][j] = d2[i - 1][j] + 1

else

d2[i][j] = d2[i][j - 1] + 1

else

d1[i][j] = d1[i - 1][j - 1]

d2[i][j] = d2[i - 1][j - 1]

double d11 = 1 - (double)d1[s1.length()][s2.length()] / (double)max(s1.length(), s2.length())

double d22 = 1 - (double)d2[s1.length()][s2.length()] / (double)max(s1.length(), s2.length())

if (d11 < 0)

d11 = 0

if (d22 < 0)

d22 = 0

cout << "Minimum Edit Distances 的抄襲程度是 " << d11 << endl

cout << "Levenshtein Distance 的抄襲程度是 " << d22 << endl

* + 1. 演算法時間複雜度(time complexity)

需計算填滿D(1..n,1..m)的n\*m陣列方能得到D(n,m)值，所以是O(nm)。

* + 1. 演算法空間複雜度(space complexity)

需要存d(0..n,0..m)的陣列，所以是O(nm)。

方法(2)

Input：兩篇文章的txt檔，原始檔案(長度為) and比對檔案(長度為)

Output：兩篇文章的段抄襲程度、兩篇文章抄襲程度、段抄襲程度的變異數、輸出段抄襲程度的標準差

a=0

for i = 0 to s1.length() – 1 do

if (s1[i] == '\n')

if (a == 1)

s1.erase(i, 1)

i--

else if (a == 2)

s1.erase(i - 1, 1)

i--

a= 1

else if (s1[i] == ' ' || s1[i] == '\t')

if (a)

s1.erase(i, 1)

i--

a= 2

else

a = 0

part1 = 1, part2 = 1

for i = 0 to s1.length() – 1 do

if (s1[i] == '\n')

part1++

a=0

for i = 0 to s2.length() – 1 do

if (s2[i] == '\n')

if (a == 1)

s2.erase(i, 1)

i--

else if (a == 2)

s2.erase(i - 1, 1)

i--

a = 1

else if (s2[i] == ' ' || s2[i] == '\t')

if (j)

s2.erase(i, 1);

i--;

j = 2;

else

j = 0

for i = 0 to s2.length() – 1 do

if (s2[i] == '\n')

part2++

index1 = 0

index2 = 0

for p2 = 0 to part2 – 1 do

partlength2 = 0

index1 = 0

for i = index2 to s2.length() – 1 do

if (s2[index2] == '\n')

break

partlength2++

index2++

for p1 = 0 to part1 – 1 do

partlength1 = 0

for i = index1 to s1.length() – 1 do

if (s1[index1] == '\n')

break

partlength1++

index1++

for i = 0 to partlength1 do

sd1[i][0] = i

sd2[i][0] = i

for i = 1 to partlength2 do

sd1[0][i] = i

sd2[0][i] = i

for i = 1 to partlength1 do

for j = 1 to partlength2 do

if (s1[index1 - 2 - partlength1 + i] != s2[index2 - 2 - partlength2 + j])

if (sd1[i - 1][j - 1] <= sd1[i][j - 1] && sd1[i - 1][j - 1] <= sd1[i - 1][j])

sd1[i][j] = sd1[i - 1][j - 1] + 1

else if (sd1[i - 1][j] <= sd1[i][j - 1] && sd1[i - 1][j] <= sd1[i - 1][j - 1])

sd1[i][j] = sd1[i - 1][j] + 1

else

sd1[i][j] = sd1[i][j - 1] + 1

if (sd2[i - 1][j - 1] + 1 <= sd2[i][j - 1] && sd2[i - 1][j - 1] + 1 <= sd2[i - 1][j])

sd2[i][j] = sd2[i - 1][j - 1] + 2

else if (sd2[i - 1][j] <= sd2[i][j - 1] && sd2[i - 1][j] <= sd2[i - 1][j - 1] + 1)

sd2[i][j] = sd2[i - 1][j] + 1

else

sd2[i][j] = sd2[i][j - 1] + 1

else

sd1[i][j] = sd1[i - 1][j - 1]

sd2[i][j] = sd2[i - 1][j - 1]

if (1 - (double)sd1[partlength1][partlength2] / (double)partlength1 >= 0)

d1[p2][p1]=1-(double)sd1[partlength1][partlength2]/(double)max(partlength1,partlength2)

else

d1[p2][p1] = 0

if (1 - (double)sd2[partlength1][partlength2] / (double)partlength1 >= 0)

d2[p2][p1]=1-(double)sd2[partlength1][partlength2]/(double)max(partlength1, partlength2)

else

d2[p2][p1] = 0

d11 = 0

d22 = 0

for i = 0 to part2 – 1 do

max1 = d1[i][0]

max2 = d2[i][0]

for j = 1 to part1 – 1 do

if (max1 < d1[i][j])

max1 = d1[i][j]

if (max2 < d2[i][j])

max2 = d2[i][j]

d11 += max1

d22 += max2

cout << "原始檔案第" << i + 1 << "段和比對檔案的段抄襲程度(Minimum Edit Distances)是 " << max1 << endl

cout << "原始檔案第" << i + 1 << "段和比對檔案的段抄襲程度(Levenshtein Distance)是 " << max2 << endl

d11 /= part2

d22 /= part2

ds1 = 0

ds2 = 0

for i = 0 to part2 – 1 do

max1 = d1[i][0]

max2 = d2[i][0]

for j = 1 to part1 – 1 do

if (max1 < d1[i][j])

max1 = d1[i][j]

if (max2 < d2[i][j])

max2 = d2[i][j]

ds1 += pow(max1 - d11, 2)

ds2 += pow(max2 - d22, 2)

ds1 /= part2

ds2 /= part2

cout << "Minimum Edit Distances :" << endl << "兩篇文章抄襲程度 : " << d11 << endl << "輸出段抄襲程度的變異數 : " << ds1 << endl << "輸出段抄襲程度的標準差 : " << sqrt(ds1) << endl;

cout << "Levenshtein Distance :" << endl << "兩篇文章抄襲程度 : " << d22 << endl << "輸出段抄襲程度的變異數 : " << ds2 << endl << "輸出段抄襲程度的標準差 : " << sqrt(ds2) << endl;

1. 演算法時間複雜度(time complexity)

O()

1. 演算法空間複雜度(space complexity)

需要存d(0.. mj,0.. ni)的陣列，所以是O()。

**(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 Windows 10

2. 程式開發工具

Visual Studio 2022

3. 電腦硬體

處理器 Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz

已安裝記憶體(RAM) 8.00 GB (7.81 GB 可用)

系統類型 64 位元作業系統，x64 型處理器

手寫筆與觸控 手寫筆支援

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

i. 利用Dynamic Programming 實作Minimum Edit Distance 演算法，包含Minimum Edit Distances (一為Insertion, Deletion, Substitution 皆各為一個operation)以及Levenshtein Distance (Insertion, Deletion,皆各為一個operation，Substitution 為兩個operations)兩種距離定義。

a. Input: Strings S1 and S2 (利用Keyboard 輸入)

b. Output: Minimum Edit Distance (number) between *S*1 and *S*2

1. 主程式

#include <iostream>

#include <string>

#include <time.h>

using namespace std;

int main(void) {

string s1, s2;

cout << "請輸入第一個字串(不能有空格)>>";

cin >> s1;

cout << "請輸入第二個字串(不能有空格)>>";

cin >> s2;

double START, END;

START = clock();

int\*\* d1 = new int\* [s1.length() + 1], \*\* d2 = new int\* [s1.length() + 1];

for (int i = 0; i < s1.length() + 1; i++) {

d1[i] = new int[s2.length() + 1];

d2[i] = new int[s2.length() + 1];

}

for (int i = 0; i < s1.length() + 1; i++) {

d1[i][0] = i;

d2[i][0] = i;

}

for (int i = 1; i < s2.length() + 1; i++) {

d1[0][i] = i;

d2[0][i] = i;

}

for (int i = 1; i < s1.length() + 1; i++) {

for (int j = 1; j < s2.length() + 1; j++) {

if (s1[i - 1] != s2[j - 1]) {

if (d1[i - 1][j - 1] <= d1[i][j - 1] && d1[i - 1][j - 1] <= d1[i - 1][j]) {

d1[i][j] = d1[i - 1][j - 1] + 1;

}

else if (d1[i - 1][j] <= d1[i][j - 1] && d1[i - 1][j] <= d1[i - 1][j - 1]) {

d1[i][j] = d1[i - 1][j] + 1;

}

else {

d1[i][j] = d1[i][j - 1] + 1;

}

if (d2[i - 1][j - 1] + 1 <= d2[i][j - 1] && d2[i - 1][j - 1] + 1 <= d2[i - 1][j]) {

d2[i][j] = d2[i - 1][j - 1] + 2;

}

else if (d2[i - 1][j] <= d2[i][j - 1] && d2[i - 1][j] <= d2[i - 1][j - 1] + 1) {

d2[i][j] = d2[i - 1][j] + 1;

}

else {

d2[i][j] = d2[i][j - 1] + 1;

}

}

else {

d1[i][j] = d1[i - 1][j - 1];

d2[i][j] = d2[i - 1][j - 1];

}

}

}

END = clock();

cout << "Minimum Edit Distances is " << d1[s1.length()][s2.length()] << endl;

cout << "Levenshtein Distance is " << d2[s1.length()][s2.length()] << endl;

cout << "Time: " << END - START << "ms";

for (int i = 0; i < s1.length() + 1; i++) {

delete[] d1[i];

delete[] d2[i];

}

delete[]d1;

delete[]d2;

system("pause");

return 0;

}

2. Input Code Format

Three of examples for input use are in below….

1. S1: execution

S2:intention

1. S1: penpineappleapplepen

S2: penapple

1. S1:123456789

S2:4567891230

3. Output Code Format

Three of examples for output use are in below….

(1)

Minimum Edit Distances is 5

Levenshtein Distance is 8

Time: 0ms

(2)

Minimum Edit Distances is 12

Levenshtein Distance is 12

Time: 0ms

(3)

Minimum Edit Distances is 7

Levenshtein Distance is 7

Time: 0ms

ii. 利用(i)實作一抄襲偵測(Plagiarism Detection)系統，包含兩種Minimum Edit Distances。

a. Input: 原始檔案(S) and 比對檔案(T) (利用讀檔，宜考慮空白、Tab、Enter 等值)

b. Output: 抄襲程度𝑑𝑆,𝑇(全部檔案以及各段抄襲程度)、段抄襲程度的變異數、段抄襲程度的標準差。

|  |  |  |
| --- | --- | --- |
| 方法 | **Input** | **Output** |
| (1) | 原始檔案(S) 比對檔案(T)  …  …  …  … | **兩篇文章抄襲程度：**  Where and are the number of characters in and , respectively, and is the edit distance between and . |
| (2) | TP2  TP3  d1m  d13  d12  d11  共m段  共n段  SPn  SP4  SP3  SP2  SP1  比對檔案(T)  原始檔案(S) | |  | | --- | | **兩段𝑺𝑷𝒊 and 𝑻𝑷𝒋文字抄襲程度：** |   For and .  TP1   |  | | --- | | **兩篇文章抄襲程度：** |   TPm   |  | | --- | | where 𝑘𝑖 = arg MAX1≤𝑗≤𝑚𝑑𝑖,𝑗 , for 1 ≤ 𝑖 ≤ 𝑛.  **Notes:** 要排除空白段，連續Tab or 空白。  Outputs:  a. 兩篇文章抄襲程度：  b. 輸出段抄襲程度的變異數   1. c. 輸出段抄襲程度的標準差 | |

**方法(1)**

1. 主程式

#include <fstream>

#include <iostream>

#include <string>

#include<math.h>

#include<time.h>

using namespace std;

int main(void) {

string s1, s2, temps;

ifstream in1, in2;

string f1, f2;

cout << "請輸入原始檔案檔名(15位元以內)>>";

cin >> f1;

in1.open(f1);

cout << "請輸入比對檔案檔名(15位元以內)>>";

cin >> f2;

in2.open(f2);

double START, END;

START = clock();

getline(in2, s1);

getline(in1, s2);

while (getline(in2, temps)) {

s1 += '\n';

s1 += temps;

}

while (getline(in1, temps)) {

s2 += '\n';

s2 += temps;

}

int\*\* d1 = new int\* [s1.length() + 1], \*\* d2 = new int\* [s1.length() + 1];

for (int i = 0; i < s1.length() + 1; i++) {

d1[i] = new int[s2.length() + 1];

d2[i] = new int[s2.length() + 1];

}

for (int i = 0; i < s1.length() + 1; i++) {

d1[i][0] = i;

d2[i][0] = i;

}

for (int i = 1; i < s2.length() + 1; i++) {

d1[0][i] = i;

d2[0][i] = i;

}

for (int i = 1; i < s1.length() + 1; i++) {

for (int j = 1; j < s2.length() + 1; j++) {

if (s1[i - 1] != s2[j - 1]) {

if (d1[i - 1][j - 1] <= d1[i][j - 1] && d1[i - 1][j - 1] <= d1[i - 1][j]) {

d1[i][j] = d1[i - 1][j - 1] + 1;

}

else if (d1[i - 1][j] <= d1[i][j - 1] && d1[i - 1][j] <= d1[i - 1][j - 1]) {

d1[i][j] = d1[i - 1][j] + 1;

}

else {

d1[i][j] = d1[i][j - 1] + 1;

}

if (d2[i - 1][j - 1] + 1 <= d2[i][j - 1] && d2[i - 1][j - 1] + 1 <= d2[i - 1][j]) {

d2[i][j] = d2[i - 1][j - 1] + 2;

}

else if (d2[i - 1][j] <= d2[i][j - 1] && d2[i - 1][j] <= d2[i - 1][j - 1] + 1) {

d2[i][j] = d2[i - 1][j] + 1;

}

else {

d2[i][j] = d2[i][j - 1] + 1;

}

}

else {

d1[i][j] = d1[i - 1][j - 1];

d2[i][j] = d2[i - 1][j - 1];

}

}

}

double d11 = 1 - (double)d1[s1.length()][s2.length()] / (double)max(s1.length(), s2.length());

double d22 = 1 - (double)d2[s1.length()][s2.length()] / (double)max(s1.length(), s2.length());

if (d11 < 0) {

d11 = 0;

}

if (d22 < 0) {

d22 = 0;

}

END = clock();

cout << "Minimum Edit Distances 的抄襲程度是 " << d11 << endl;

cout << "Levenshtein Distance 的抄襲程度是 " << d22 << endl;

cout << "Time: " << END - START << "ms";

for (int i = 0; i < s1.length() + 1; i++) {

delete[] d1[i];

delete[] d2[i];

}

delete[]d1;

delete[]d2;

in1.close();

in2.close();

system("pause");

return 0;

}

2. Input Code Format

Three of examples for input use are in below….

(1) 原始檔案:TestFile1.txt比對檔案:TestFile11.txt

(2) 原始檔案:TestFile1.txt比對檔案:TestFile12.txt

(3) 原始檔案:TestFile2.txt比對檔案:TestFile21.txt

(4) 原始檔案:TestFile2.txt比對檔案:TestFile22.txt

3. Output Code Format

Three of examples for output use are in below….

(1) o1\_1.txt

(2) o1\_2.txt

(3) o2\_1.txt

(4) o2\_2.txt

**方法(2)**

1. 主程式

#include <fstream>

#include <iostream>

#include <string>

#include<math.h>

#include <time.h>

using namespace std;

int main(void) {

string s1, s2, temps;

ifstream in1, in2;

string f1, f2;

cout << "請輸入原始檔案檔名(15位元以內)>>";

cin >> f1;

in1.open(f1);

cout << "請輸入比對檔案檔名(15位元以內)>>";

cin >> f2;

in2.open(f2);

double START, END;

START = clock();

getline(in2, s1);

getline(in1, s2);

while (getline(in2, temps)) {

s1 += '\n';

s1 += temps;

}

while (getline(in1, temps)) {

s2 += '\n';

s2 += temps;

}

for (int i = 0, j = 0; i < s1.length(); i++) {

if (s1[i] == '\n') {

if (j == 1) {

s1.erase(i, 1);

i--;

}

else if (j == 2) {

s1.erase(i - 1, 1);

i--;

}

j = 1;

}

else if (s1[i] == ' ' || s1[i] == '\t') {

if (j) {

s1.erase(i, 1);

i--;

}

j = 2;

}

else {

j = 0;

}

}

int part1 = 1, part2 = 1;

for (int i = 0; i < s1.length(); i++) {

if (s1[i] == '\n') {

part1++;

}

}

for (int i = 0, j = 0; i < s2.length(); i++) {

if (s2[i] == '\n') {

if (j == 1) {

s2.erase(i, 1);

i--;

}

else if (j == 2) {

s2.erase(i - 1, 1);

i--;

}

j = 1;

}

else if (s2[i] == ' ' || s2[i] == '\t') {

if (j) {

s2.erase(i, 1);

i--;

}

j = 2;

}

else {

j = 0;

}

}

for (int i = 0; i < s2.length(); i++) {

if (s2[i] == '\n') {

part2++;

}

}

double\*\* d1 = new double\* [part2], \*\* d2 = new double\* [part2];

for (int i = 0; i < part2; i++) {

d1[i] = new double[part1];

d2[i] = new double[part1];

}

int index1 = 0, index2 = 0;

for (int p2 = 0; p2 < part2; p2++) {

int partlength2 = 0;

index1 = 0;

for (index2; index2 < s2.length(); index2++) {

if (s2[index2] == '\n') {

break;

}

partlength2++;

}

index2++;

for (int p1 = 0; p1 < part1; p1++) {

int partlength1 = 0;

for (index1; index1 < s1.length(); index1++) {

if (s1[index1] == '\n') {

break;

}

partlength1++;

}

index1++;

int\*\* sd1 = new int\* [partlength1 + 1], \*\* sd2 = new int\* [partlength1 + 1];

for (int i = 0; i < partlength1 + 1; i++) {

sd1[i] = new int[partlength2 + 1];

sd2[i] = new int[partlength2 + 1];

}

for (int i = 0; i < partlength1 + 1; i++) {

sd1[i][0] = i;

sd2[i][0] = i;

}

for (int i = 1; i < partlength2 + 1; i++) {

sd1[0][i] = i;

sd2[0][i] = i;

}

for (int i = 1; i < partlength1 + 1; i++) {

for (int j = 1; j < partlength2 + 1; j++) {

if (s1[index1 - 2 - partlength1 + i] != s2[index2 - 2 - partlength2 + j]) {

if (sd1[i - 1][j - 1] <= sd1[i][j - 1] && sd1[i - 1][j - 1] <= sd1[i - 1][j]) {

sd1[i][j] = sd1[i - 1][j - 1] + 1;

}

else if (sd1[i - 1][j] <= sd1[i][j - 1] && sd1[i - 1][j] <= sd1[i - 1][j - 1]) {

sd1[i][j] = sd1[i - 1][j] + 1;

}

else {

sd1[i][j] = sd1[i][j - 1] + 1;

}

if (sd2[i - 1][j - 1] + 1 <= sd2[i][j - 1] && sd2[i - 1][j - 1] + 1 <= sd2[i - 1][j]) {

sd2[i][j] = sd2[i - 1][j - 1] + 2;

}

else if (sd2[i - 1][j] <= sd2[i][j - 1] && sd2[i - 1][j] <= sd2[i - 1][j - 1] + 1) {

sd2[i][j] = sd2[i - 1][j] + 1;

}

else {

sd2[i][j] = sd2[i][j - 1] + 1;

}

}

else {

sd1[i][j] = sd1[i - 1][j - 1];

sd2[i][j] = sd2[i - 1][j - 1];

}

}

}

if (1 - (double)sd1[partlength1][partlength2] / (double)partlength1 >= 0) {

d1[p2][p1] = 1 - (double)sd1[partlength1][partlength2] / (double)max(partlength1,partlength2);

}

else {

d1[p2][p1] = 0;

}

if (1 - (double)sd2[partlength1][partlength2] / (double)partlength1 >= 0) {

d2[p2][p1] = 1 - (double)sd2[partlength1][partlength2] / (double)max(partlength1, partlength2);

}

else {

d2[p2][p1] = 0;

}

for (int i = 0; i < partlength1 + 1; i++) {

delete[] sd1[i];

delete[] sd2[i];

}

delete[]sd1;

delete[]sd2;

}

}

double d11 = 0, d22 = 0;

for (int i = 0; i < part2; i++) {

double max1 = d1[i][0], max2 = d2[i][0];

for (int j = 1; j < part1; j++) {

if (max1 < d1[i][j]) {

max1 = d1[i][j];

}

if (max2 < d2[i][j]) {

max2 = d2[i][j];

}

}

d11 += max1;

d22 += max2;

cout << "原始檔案第" << i + 1 << "段和比對檔案的段抄襲程度(Minimum Edit Distances)是 " << max1 << endl;

cout << "原始檔案第" << i + 1 << "段和比對檔案的段抄襲程度(Levenshtein Distance)是 " << max2 << endl;

}

d11 /= part2;

d22 /= part2;

double ds1 = 0, ds2 = 0;

for (int i = 0; i < part2; i++) {

double max1 = d1[i][0], max2 = d2[i][0];

for (int j = 1; j < part1; j++) {

if (max1 < d1[i][j]) {

max1 = d1[i][j];

}

if (max2 < d2[i][j]) {

max2 = d2[i][j];

}

}

ds1 += pow(max1 - d11, 2);

ds2 += pow(max2 - d22, 2);

}

ds1 /= part2;

ds2 /= part2;

END = clock();

cout << "Minimum Edit Distances :" << endl << "兩篇文章抄襲程度 : " << d11 << endl << "輸出段抄襲程度的變異數 : " << ds1 << endl << "輸出段抄襲程度的標準差 : " << sqrt(ds1) << endl;

cout << "Levenshtein Distance :" << endl << "兩篇文章抄襲程度 : " << d22 << endl << "輸出段抄襲程度的變異數 : " << ds2 << endl << "輸出段抄襲程度的標準差 : " << sqrt(ds2) << endl;

cout << "Time: " << END - START << "ms";

for (int i = 0; i < part2; i++) {

delete[] d1[i];

delete[] d2[i];

}

delete[]d1;

delete[]d2;

in1.close();

in2.close();

system("pause");

return 0;

}

2. Input Code Format

Three of examples for input use are in below….

(1) 原始檔案:TestFile1.txt比對檔案:TestFile11.txt

(2) 原始檔案:TestFile1.txt比對檔案:TestFile12.txt

(3) 原始檔案:TestFile2.txt比對檔案:TestFile21.txt

(4) 原始檔案:TestFile2.txt比對檔案:TestFile22.txt

3. Output Code Format

Three of examples for output use are in below….

(1) o1\_1.txt

(2) o1\_2.txt

(3) o2\_1.txt

(4) o2\_2.txt

**(五) 執行結果、討論與心得**

i. 利用Dynamic Programming 實作Minimum Edit Distance 演算法，包含Minimum Edit Distances (一為Insertion, Deletion, Substitution 皆各為一個operation)以及Levenshtein Distance (Insertion, Deletion,皆各為一個operation，Substitution 為兩個operations)兩種距離定義。

a. Input: Strings S1 and S2 (利用Keyboard 輸入)

b. Output: Minimum Edit Distance (number) between *S*1 and *S*2

1. 執行結果

1. o1.txt
2. o2.txt
3. o3.txt
4. o4.txt
5. o5.txt
6. o6.txt
7. o7.txt
8. o8.txt
9. o9.txt
10. o10.txt

2. 討論

執行時間、問題大小等問題討論

1. Running Time
2. 57ms
3. 198ms
4. 473ms
5. 794ms
6. 835ms
7. 866ms
8. 870ms
9. 838ms
10. 843ms
11. 871ms

(2) Problem size *n*

1. 原始檔案：TestFile1.txt比對檔案：TestFile11.txt 長度都是1000字元
2. 原始檔案：TestFile2.txt比對檔案：TestFile21.txt長度都是2000字元
3. 原始檔案：TestFile3.txt比對檔案：TestFile31.txt長度都是3000字元
4. 原始檔案：TestFile4.txt比對檔案：TestFile41.txt長度都是4000字元
5. 原始檔案：TestFile5.txt比對檔案：TestFile51.txt長度都是5000字元
6. 原始檔案：TestFile6.txt比對檔案：TestFile61.txt長度都是6000字元
7. 原始檔案：TestFile7.txt比對檔案：TestFile71.txt長度都是7000字元
8. 原始檔案：TestFile8.txt比對檔案：TestFile81.txt長度都是8000字元
9. 原始檔案：TestFile9.txt比對檔案：TestFile91.txt長度都是9000字元
10. 原始檔案：TestFile10.txt比對檔案：TestFile101.txt長度都是10000字元

跟預期的O(nm)指數圖形不一樣，而是呈對數圖形。

ii. 利用(i)實作一抄襲偵測(Plagiarism Detection)系統，包含兩種Minimum Edit Distances。

a. Input: 原始檔案(S) and 比對檔案(T) (利用讀檔，宜考慮空白、Tab、Enter 等值)

b. Output: 抄襲程度𝑑𝑆,𝑇(全部檔案以及各段抄襲程度)、段抄襲程度的變異數、段抄襲程度的標準差。

|  |  |  |
| --- | --- | --- |
| 方法 | **Input** | **Output** |
| (1) | 原始檔案(S) 比對檔案(T)  …  …  …  … | **兩篇文章抄襲程度：**  Where and are the number of characters in and , respectively, and is the edit distance between and . |
| (2) | TP2  TP3  d1m  d13  d12  d11  共m段  共n段  SPn  SP4  SP3  SP2  SP1  比對檔案(T)  原始檔案(S) | |  | | --- | | **兩段𝑺𝑷𝒊 and 𝑻𝑷𝒋文字抄襲程度：** |   For and .  TP1   |  | | --- | | **兩篇文章抄襲程度：** |   TPm   |  | | --- | | where 𝑘𝑖 = arg MAX1≤𝑗≤𝑚𝑑𝑖,𝑗 , for 1 ≤ 𝑖 ≤ 𝑛.  **Notes:** 要排除空白段，連續Tab or 空白。  Outputs:  a. 兩篇文章抄襲程度：  b. 輸出段抄襲程度的變異數   1. c. 輸出段抄襲程度的標準差 | |

方法(1)

1. 執行結果

1. o3.txt
2. o4.txt
3. o5.txt
4. o6.txt
5. o7.txt
6. o8.txt
7. o9.txt
8. o10.txt
9. o11.txt
10. o12.txt

2. 討論

執行時間、問題大小等問題討論

1. Running Time
2. 79ms
3. 294ms
4. 697ms
5. 1389ms
6. 2375ms
7. 2787ms
8. 3784ms
9. 4316ms
10. 6108ms
11. 8196ms

(2) Problem size *n*

1. 原始檔案：TestFile3.txt比對檔案：TestFile31.txt 長度都是1000字元
2. 原始檔案：TestFile4.txt比對檔案：TestFile41.txt長度都是2000字元
3. 原始檔案：TestFile5.txt比對檔案：TestFile51.txt長度都是3000字元
4. 原始檔案：TestFile6.txt比對檔案：TestFile61.txt長度都是4000字元
5. 原始檔案：TestFile7.txt比對檔案：TestFile71.txt長度都是5000字元
6. 原始檔案：TestFile8.txt比對檔案：TestFile81.txt長度都是6000字元
7. 原始檔案：TestFile9.txt比對檔案：TestFile91.txt長度都是7000字元
8. 原始檔案：TestFile10.txt比對檔案：TestFile101.txt長度都是8000字元
9. 原始檔案：TestFile110.txt比對檔案：TestFile111.txt長度都是9000字元
10. 原始檔案：TestFile120.txt比對檔案：TestFile121.txt長度都是10000字元

跟預期的O(nm)指數圖形一樣。

方法(2)

1. 執行結果

1. o3.txt
2. o4.txt
3. o5.txt
4. o6.txt
5. o7.txt
6. o8.txt
7. o9.txt
8. o10.txt
9. o11.txt
10. o12.txt
11. o13.txt
12. o14.txt
13. o15.txt
14. o16.txt
15. o17.txt
16. o18.txt
17. o19.txt
18. o20.txt
19. o21.txt
20. o22.txt

2. 討論

執行時間、問題大小等問題討論

1. Running Time
2. 60ms
3. 273ms
4. 651ms
5. 906ms
6. 1987ms
7. 2324ms
8. 3030ms
9. 3830ms
10. 4565ms
11. 5410ms
12. 637ms
13. 1360ms
14. 2132ms
15. 1916ms
16. 1994ms
17. 2337ms
18. 3993ms
19. 3250ms
20. 3596ms
21. 4136ms

(2) Problem size *n*

1. 原始檔案：TestFile3.txt比對檔案：TestFile31.txt 長度都是1000字元，段數都是1
2. 原始檔案：TestFile4.txt比對檔案：TestFile41.txt長度都是2000字元，段數都是1
3. 原始檔案：TestFile5.txt比對檔案：TestFile51.txt長度都是3000字元，段數都是1
4. 原始檔案：TestFile6.txt比對檔案：TestFile61.txt長度都是4000字元，段數都是1
5. 原始檔案：TestFile7.txt比對檔案：TestFile71.txt長度都是5000字元，段數都是1
6. 原始檔案：TestFile8.txt比對檔案：TestFile81.txt長度都是6000字元，段數都是1
7. 原始檔案：TestFile9.txt比對檔案：TestFile91.txt長度都是7000字元，段數都是1
8. 原始檔案：TestFile10.txt比對檔案：TestFile101.txt長度都是8000字元，段數都是1
9. 原始檔案：TestFile110.txt比對檔案：TestFile111.txt長度都是9000字元，段數都是1
10. 原始檔案：TestFile120.txt比對檔案：TestFile121.txt長度都是10000字元，段數都是1
11. 原始檔案：TestFile130.txt比對檔案：TestFile131.txt 長度都是1000字元，段數都是100
12. 原始檔案：TestFile140.txt比對檔案：TestFile141.txt 長度都是1000字元，段數都是200
13. 原始檔案：TestFile150.txt比對檔案：TestFile151.txt 長度都是1000字元，段數都是300
14. 原始檔案：TestFile160.txt比對檔案：TestFile161.txt 長度都是1000字元，段數都是400
15. 原始檔案：TestFile170.txt比對檔案：TestFile171.txt 長度都是1000字元，段數都是500
16. 原始檔案：TestFile180.txt比對檔案：TestFile181.txt 長度都是1000字元，段數都是600
17. 原始檔案：TestFile190.txt比對檔案：TestFile191.txt 長度都是1000字元，段數都是700
18. 原始檔案：TestFile200.txt比對檔案：TestFile201.txt 長度都是1000字元，段數都是800
19. 原始檔案：TestFile210.txt比對檔案：TestFile211.txt 長度都是1000字元，段數都是900
20. 原始檔案：TestFile220.txt比對檔案：TestFile221.txt 長度都是1000字元，段數都是1000

O()當a=b=1，呈指數圖形。

O()，i=floor(1000/a)，j=floor(1000/b)，

(1000-a\*floor(1000/a))\*(1000-b\*floor(1000/b))+

(1000-a\*floor(1000/a))\*floor(1000/b)+

(1000b\*floor(1000/b))\*floor(1000/a)+999\*999\*1000

3. 心得

比較熟悉Minimum Edit Distance的用法了，釐清了一些細節，實作過程有遇到一些問題，像是沒看清題目是原始檔案1對多比對檔案中的最大值為段抄襲程度，而是用比對檔案1對多原始檔案中的最大值為段抄襲程度，花了一些時間de這些很遜的bug。題目要求要排除空白段，連續Tab或空白讓我比較猶豫，最後決定刪除enter前後的所有空白和Tab並且將連續空白或Tab留剩一個，私以為這是最好的結果。藉由這次複習了下C++的文法，之前一直避免用C++來寫，這次因為有字串，所以才用C++寫，不算太難，也比較省時間，否則會花很多時間處理字串的申請空間大小和寫一些小function，這次作業比較簡單，也沒那麼傷身體，算是一點小確幸。

**參考文獻**

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "Introduction to Algorithms," Third Edition, The MIT Press, 2009.
2. R.C.T. Lee, S.S. Tseng, R.C. Chang, and Y.T.Tsai, "Introduction to the Design and Analysis of Algorithms," McGraw-Hill, 2005.
3. Anany V. Levitin, "Introduction to the Design and Analysis of Algorithms," 2nd Edition, Addison Wesley, 2007.
4. Richard Neapolitan and Kumarss Naimipour, "Foundations of Algorithms," Fourth Edition, Jones and Bartlett Publishers, 2010.
5. …