The String-to-String Correction Problem

I. Introduction

字串到字串的校正問題是確定兩個字串之間的距離，該距離由將一個字串轉換為另一個字串所需的“編輯操作”的最小成本順序來衡量。編輯操作允許將字串的一個字元更改為另一個字元(Change Operation)，從字串中刪除一個字元(Delete Operation)，或將單個字元插入字串(Insert Operation)。 本篇論文中，作者提出了一種算法，該算法解決問題時間與兩個字串的長度乘積成比例。

II. Problem Description

給定 X 和 Y 兩個任意字串，限定操作只可能為替換(change)、刪除(delete)與插入(insert)，將所進行的操作記為 S 序列。若要將 A 字串編輯至 B 字串，找最小成本的操作序列 S。

III. Definition

i. 字串

由數個字元組成的序列，稱為字串，以 A[i]表示字串 A 中第 i 個字元，A[i:j]表示 A[i]A[i+1]⋯A[j]，其中|A|表示字串長度且當 i>j 時，A[i:j]為空字串。

ii. 編輯操作

常見的電腦編輯操作有以下三種

(1)change operation，表示將一字元由另一字元替代

(2)delete operation，表示刪除一字元

(3)insert operation，表示插入一字元

若對字串 A 進行一系列的編輯操作後變成 B，可以表記每次的編輯操作為 S_i，則編輯序列可以記為

$$S = S_1S_2\cdots S_m,\ 1 \le i \le m \text{ for all } i$$

iii. 編輯操作成本

每當電腦執行一編輯操作所花費的成本為編輯操作成本

本篇論文為每個編輯操作賦予一個非負實數作為成本，表示為 γ

配合編輯序列，我們可以統計出所有的編輯成本，記為

$$\gamma(S) = \gamma(S_1) + \gamma(S_2) + \cdots + \gamma(S_m)$$

若 $m = 0$， $\gamma(S) = 0$

iv. 編輯距離

回到問題定義，求最小編輯距離可以改為求最小編輯成本，即

$$\text{Distance}(A,B) = \min\{\gamma(S) \mid S \in \text{所有 A 到 B 編輯序列}\}$$

IV. Solving Process

動態規劃常常適用於有重疊子問題和最佳子結構性質的問題，動態規劃方法所耗時間往往遠少於遞迴解法。動態規劃背後的基本思想非常簡單。大致上，若要解一個給定問題，我們需要解其不同部分（即子問題），再根據子問題的解以得出原問題的解。

根據問題敘述，給定有限長度字串 A 和 B，並計算其最小編輯成本，可以採用 DP 先解小問題後再求解。針對此問題，可以寫出以下遞迴式

$$D(i,j) = \min\begin{cases} D(i-1, j-1) + \gamma\,(\text{change operation}) \\ D(i-1, j) + \gamma\,(\text{delete operation}) \\ D(i, j-1) + \gamma\,(\text{insert operation}) \end{cases}$$

其中 D(i , j)表示 A[1: i] 與 B[1: j]的最小編輯成本

V. Example

設 A = “DESIGN” ， B = “ALGORITHM” ，並設定三種編輯操作成本為

(1)change operation cost = 3

(2)delete operation cost = 2

(3)insert operation cost = 1

結果如下圖所示，最小編輯成本為 7，其時間複雜度為 O(|A|*|B|)

		A	L	G	O	R	I	T	H	M
	0	1	2	3	4	5	6	7	8	9
D	1	2	3	4	5	6	7	8	9	10
E	2	3	4	5	6	7	8	9	10	11
S	3	4	5	6	7	8	9	10	11	12
I	4	5	6	7	8	9	0	1	2	3
G	5	6	7	0	1	2	2	3	4	5
N	6	7	8	2	3	4	4	5	6	7

VI. Implementation

```python
import numpy as np

A = "DESIGN"
B = "ALGORITHM"
change_cost = 3
delete_cost = 2
insert_cost = 1
D = np.zeros((len(A)+1,len(B)+1))

for i in range(len(A)+1):
    D[i,0] = i
for i in range(len(B)+1):
    D[0,i] = i
for i in range(1,len(A)+1):
    for j in range(1,len(B)+1):
        if A[i-1] == B[j-1]:
            D[i,j] == D[i-1,j-1]
        else:
            D[i,j] = min(D[i-1,j-1]+change_cost, D[i-1,j]+delete_cost, D[i,j-1]+insert_cost)

print(D)
```

VII. Application

編輯距離是針對二個字串（例如英文字）的差異程度的量化量測，量測方式是看至少需要多少次的處理才能將一個字串變成另一個字串。編輯距離可以用在自然語言處理中，例如拼寫檢查可以根據一個拼錯的字和其他正確的字的編輯距離，判斷哪一個（或哪幾個）是比較可能的字。DNA 也可以視為用 A、C、G 和 T 組成的字串，因此編輯距離也用在生物信息學中，判斷二個 DNA 的類似程度。Unix 下的 diff 及 patch 即是利用編輯距離來進行文本編輯對比的例子。

VIII. Feedback

Dynamic programming 為一個解決遞迴型態問題的策略，有別於 Divide-and-conquer 設計方式為直接呼叫遞迴式，DP 傾向於先從子問題著手，及解原問題前將所有可能會參考到的子問題之解全數算出。此篇論文主要的核心在於利用動態規劃方法求解編輯距離問題，其實背後的本質與最長共同子序列(Longest Common Subsequence, LCS)十分類似，這類問題並不會使用窮舉搜索，因為單就一個長度為 n 的序列便會有 2^n 個子序列，使用窮舉的時間複雜度為指數階，因此必須借助動態規劃的方法進行求解。