R05546030 **彭新翔**

一、 簡介

俄羅斯科學家 Vladimir Levenshtein 於 1965 提出了 Levenshtein Distance 的概念，屬於編輯距離(Edit Distance)的一種，指在兩個字串之間相互轉換所需要的最少操作次數，其中允許的操作包括替換(change)、刪除(delete)與插入(insert)，而在實際上的問題處理，經常會使用動態規劃(Dynamic Programming)手段進行處理。這類的問題可以被應用於自然語言處理和生物遺傳學上，如：拼寫檢查、文檔改動、基因序列比較…等。

二、 問題定義

此篇論文所要處理的問題為：

給定 A 和 B 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。

為了更清楚地理解問題與解法敘述，在此必須先給出一些定義：

A. 字串 (string)

以大寫英文字母代表一個由有限個數的字元(character)所組成的字串，並以 $A\langle i\rangle$ 表示其中第 i 個字元，即：

$$A\langle i:j\rangle = A\langle i\rangle A\langle i+1\rangle \cdots A\langle j\rangle$$

其中：

- $|A|$ 表示字串長度
- 當 $i > j$ 時，$A\langle i:j\rangle = \Lambda$ 為一空字串

B. 編輯操作 (edit operation)

編輯操作以數對 $(a,b) \neq (\Lambda,\Lambda)$ 表示，一般記做 $a \to b$，承上所述共有三種可能的編輯操作方式：

(1) 替換(change operation)：其中 $a \neq \Lambda$ 且 $b \neq \Lambda$

(2) 刪除(delete operation)：其中 $b \neq \Lambda$

(3) 插入(insert operation)：其中 $a \neq \Lambda$

回到問題本身，若要將 A 字串編輯至 B 字串通常需要進行一系列的編輯操作，為了便於撰寫我們將每一次的編輯操作都記為 s_i 並定義編輯序列(edit sequence)為：

$$S = s_1 s_2 \cdots s_m, \forall i, 1 \leq i \leq m$$

C. 成本函數 (cost function)

為了評估操作的成本，我們有以下定義：

(1) 對於每一個編輯操作(edit operation)，賦予一個非負的實數作為成本，即：$\gamma(s_i)$

(2) 對於一系列編輯操作所組成的編輯序列(edit sequence)，定義其成本函數為每一個操作的成本相加，即：

$$\gamma(S) = \sum_{i=1}^{m} \gamma(s_i) = \gamma(s_1) + \gamma(s_2) + \cdots + \gamma(s_m)$$

(3) 上述定義中，若 $m = 0$ 則定義有 $\gamma(S) = 0$

D. 編輯距離 (edit distance)

根據上述成本函數，我們可以將編輯距離定義為編輯序列中成本函數最小者，即：

$$\delta(A,B) = \min\{\gamma(S) \mid S \text{ is an edit sequence taking } A \text{ to } B\}$$

除此之外，論文原文中為了簡化問題的敘述方式，還額外定義了軌跡(Traces)，但我認為在後續的解法敘述與實踐中並沒有非提及不可的需求，這邊省略不贅述。

R05546030 **彭新翔**

三、 解法敘述

根據問題要求，給定有限長度字串 A 和 B 並計算其編輯距離(edit distance)，採用動態規劃方法將原問題切分成小問題後求解，針對字串最後的字元進行分類討論，可以寫出以下的遞迴式：

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + \gamma(A\langle i\rangle \to B\langle j\rangle) \\ D(i-1,j) + \gamma(A\langle i\rangle \to \Lambda) \\ D(i,j-1) + \gamma(\lambda \to B\langle j\rangle) \end{cases}$$

其中：

- $A(i) = A\langle 1:i\rangle = A\langle 1\rangle A\langle 2\rangle \cdots B\langle i\rangle$
- $B(j) = B\langle 1:j\rangle = B\langle 1\rangle B\langle 2\rangle \cdots B\langle j\rangle$
- $D(i,j) = \delta(A(i), B(j))$ 其中 $0 \le i \le |A|$ 且 $0 \le j \le |B|$

舉例來說，倘若 $A =$ "math" 而 $B =$ "mouth"，此處針對三種編輯操作給定成本：

(1) 替換(change operation)：$\text{Cost} = 2$

(2) 刪除(delete operation)：$\text{Cost} = 1$

(3) 插入(insert operation)：$\text{Cost} = 1$

針對上述簡單的例子，利用動態規劃求解的過程即為完成下列矩陣：

	Λ	m	o	u	t	h
Λ	0	1	2	3	4	5
m	1	0	1	2	3	4
a	2	1	1	2	3	4
t	3	2	2	2	2	3
h	4	3	3	3	3	2

如上所述，由於求解的過程必須使用動態規劃將成本矩陣填滿，因此其時間複雜度為 $O(|A| \times |B|)$

四、 程式實踐

以下程式碼為使用 Python 進行 Wagner-Fischer Algorithm 的實踐：

```python
def edit_distance(string1, string2):
    m = len(string1)
    n = len(string2)
    d = [[0 for j in range(n + 1)] for i in range(m + 1)]
    for i in range(m + 1):
        d[i][0] = i
    for j in range(n + 1):
        d[0][j] = j

    for i in range(m):
        for j in range(n):
            if string1[i] == string2[j]:
                d[i+1][j+1] = d[i][j]
            else:
                d[i+1][j+1] = min(d[i][j+1] + 1, d[i+1][j] + 1, d[i][j] + 1)

    print ("Edit Distance between \"{0}\" and \"{1}\" is {2}".format(string1, string2, d[m][n]))
```

R05546030 **彭新翔**

五、 閱讀心得

在工業工程相關系所的課程中，作業研究(Operation Research)的內容其實與演算法(Algorithm)有許多雷同的地方，後者的內容與發展過程中囊括了近幾年來計算機科學所發展與被提出的解決方法，而前者目的在於替各式各樣的最佳化問題建立數學模型並給予適當的解法，根據問題的性質通常可以再細分為：線性規劃、非線性規劃、動態規劃、整數組合規劃、凸優化…等。而在現實生活中或者狹義地僅只看計算機科學中，最佳化的問題通常就是希望可以找到適當的演算法降低複雜度。

此篇論文主要的核心在於利用動態規劃(Dynamic Programing)方法求解編輯距離(Edit Distance)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。一般來說動態規劃經常用來處理多階段(multistage)的決策問題，將一個龐大的問題分解成一系列前後相關的小問題，先求解子問題再經由子問題所得到的解得到原問題的解，其模式組成通常包含有以下三部分：

(1) 遞迴關係式(recursive relation)：用以關聯求解時的不同階段，如序列與子序列間的關係。
(2) 最佳值函數(optimal value function)：所要用於決策的目標，如使複雜度最低。
(3) 邊界條件(boundary condition)：限制條件。

在多數的狀態，動態規劃本身並不太像是一個演算法(algorithm)，而是提供了一種解決問題的思考方式與過程(solution procedure)，並且在不同的學科之間總是往往能夠看到一些共同性，在閱讀這篇論文的過程中最有趣的事情莫過於發現 Wagner 與 Fisher 於 1970 年代提出這篇論文的同時，分子生物學家 Needleman 和 Wunsch 也以動態規劃方法分析了胺基酸序列的相似程度，頗有牛頓與萊布尼茲在同時代下發展微積分的既視感（當然以某些角度來看，胺基酸序列的分析要進行，應當將資料以字串形式存於計算機中再進行，因此 Wagner 兩人的方法更具有前瞻性）。

雖然在閱讀過程中發現整體的架構與演算法並不會十分難以理解，但能夠被引用許多次（以撰寫報告當天而言，在 Google 學術搜尋上所顯示的引用次數為 3389 次）代表具備十分重大的意義，就如上面所提及的這篇算法具備十分高的**通用性**，比如說在撰寫過程中便有閱讀到許多相關應用，包括但不限於：

- 搜尋引擎進行關鍵字比對與模糊搜尋
- 文字自動校正與查詢（像是 Word 撰寫文章時的紅色藍色小波浪、Google 拼寫錯誤提醒與建議）
- 基因序對的分析比較
- 文字語言翻譯
- Shell 更新緩衝區內容，刷新前後的差異比較
- Command: diff, git diff, rename

此外，四十多年來在計算機科學相關領域中有許多學者致力於找出複雜度更加低的演算法，但在 2015 年於奧地利舉行的 47th Annual Symposium on the Theory of Computing 中麻省理工學院學者 Indyk 和 Arturs Backurs 所發表的研究成果中，指出從數學的角度發現 Wagner-Fischer Algorithm 無法再降低複雜度使其低於 $O(n^2)$。就通用性與正確性而言，此篇論文的價值的確並不能夠僅僅以內容難度來作為評斷依據。

六、 參考資料

[1] Wikipedia – Wagner–Fischer algorithm, https://en.wikipedia.org/wiki/Wagner–Fischer_algorithm
[2] Wikipedia – Edit distance, https://en.wikipedia.org/wiki/Edit_distance

[3] Wikipedia – Approximate string matching, https://en.wikipedia.org/wiki/Approximate_string_matching

[4] 演算法筆記 – Approximate String Matching, http://www.csie.ntnu.edu.tw/~u91029/ApproximateStringMatching.html

[5] Chet Ramey, The Bourne–Again Shell, http://www.aosabook.org/en/bash.html

[6] Morris, UVa 12747 Back to Edit Distance, http://morris821028.github.io/2014/10/03/oj/uva/uva-12747/

[7] Arturs Backurs, Piotr Indyk, Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false), https://arxiv.org/abs/1412.0348

[8] Kevin Hartnett, For 40 years, computer scientists looked for a solution that doesn't exist, https://www.bostonglobe.com/ideas/2015/08/10/computer-scientists-have-looked-for-solution-that-doesn-exist/tXO0qNRnbKrClfUPmavifK/story.html

[9] 说说单词智能纠错算法, http://www.it610.com/article/4796569.htm

[10] Shi-Yao Jow, Lecture hangout: Sequence Alignment, http://math.cts.nthu.edu.tw/Mathematics/2012Summer-mb/Lecture%205.pdf