

演算法設計方法論 (Design Strategies for Computer Algorithms)

Homework 1

DUE DATE: OCTOBER 30, 2017

學號: b03902129 系級: 資工四 姓名: 陳鵬宇

1 問題定義

The string-to-string correction problem 是要解決以下的問題:

輸入: 字串 $A = a_1a_2, \dots, a_{|A|}$ 和字串 $B = b_1b_2, \dots, b_{|B|}$, 其中 $|A|$ 和 $|B|$ 分別為字串 A 和 B 的長度。

輸出: 將字串 A edit 至字串 B 花費最少的 sequence S , 其中 $S = s_1s_2, \dots, s_{|S|}$, $|S|$ 為 S 的長度, S 有以下三種 edit 方式:

1. 改變 (*change*): 將字串 A 的一個字元改變成另一個字元。
2. 刪除 (*delete*): 將字串 A 的一個字元刪除。
3. 插入 (*insert*): 在字串 A 中插入一個字元。

2 名詞、性質及定義

2.1 Edit Distance

2.1.1 A 's properties and notations

先給定以下 A 的一些性質及表示法, 方便讀者理解:

- A 為一個長度有限的字串
- a_i 為 A 的第 i 個字元
- $A_{i..j} = a_ia_{i+1}, \dots, a_j$
- 當 $i > j$ 時, $A_{i..j}$ 為空字串 (null string)
- $|A|$ 為字串 A 的長度
- $A_0 = A$

2.1.2 *edit operation*

定義:

edit operation: $a \rightarrow b$, 若 $A = XaY$ 可透過 *operation* $a \rightarrow b$ 變成 $B = XbY$ (X 和 Y 為字串)。

我們用 $A \Rightarrow B$ via $a \rightarrow b$ 表示之, 其中 $0 \leq |a| \leq 1, 0 \leq |b| \leq 1$.

根據問題定義所敘述, 同樣地, *edit operation*: $a \rightarrow b$ 也有三種 *edit* 方式:

1. 改變 (*change*): change a to b , 其中 $a \neq \emptyset$ 且 $b \neq \emptyset$
2. 刪除 (*delete*): delete a , 其中 $b = \emptyset$
3. 插入 (*insert*): insert b , 其中 $a = \emptyset$

2.1.3 edit sequence S

定義:

$$\text{edit sequence } S = s_1 s_2, \dots, s_m \text{ 且 } AS = B,$$

s_i 是一個 edit operation: $a \rightarrow b, \forall i, 1 \leq i \leq m$; 而 $AS = B$ 代表: 透過 edit sequence S , A 會被 edit 成 B , 如下所示:

$$\begin{aligned} A &= A_0 \\ A_0 s_1 &= A_1 \\ A_0 s_1 s_2 &= A_1 s_2 = A_2 \\ &\vdots \\ A_0 s_1 s_2, \dots, s_{m-1} &= A_{m-2} s_{m-1} = A_{m-1} \\ A_0 s_1 s_2, \dots, s_{m-1} s_m &= A_{m-1} s_m = A_m = B \end{aligned}$$

此外, 我們說 S takes A to B , 如果存在一個 S 可以將 A edit 成 B

2.1.4 cost function γ and edit distance $\delta(A, B)$

定義:

cost function $\gamma(a \rightarrow b)$: 對每一個 edit operation $a \rightarrow b$, 我們賦與它一個非負的實數。

因為 edit sequence $S = s_1 s_2, \dots, s_m$ 是由 m 個 edit operations 所組成的, 所以我們定義:

$$\text{cost function } \gamma(S) = \sum_{i=1}^m \gamma(s_i) = \gamma(s_1) + \gamma(s_2) + \dots + \gamma(s_m).$$

但這樣的結果我們仍不滿意, 我們希望能找到最小的 cost function, 所以我們定義:

$$\text{edit distance } \delta(A, B) = \min\{\gamma(S) \mid S \text{ 為可以將 } A \text{ edit 成 } B \text{ 的某一 edit sequence}\}.$$

我們題目的輸出就是希望能夠使上述的 edit distance $\delta(A, B)$ 擁有最小 cost 的 S 。

2.2 Traces T

2.2.1 cost function of T

定義:

$$\text{cost function } \gamma(T_{A \rightarrow B}),$$

其中 $T_{A \rightarrow B}$ 為由 A 連到 B 的軌跡 (trace), 它有以下兩種性質:

(性質 1) 對所有 $T_{A \rightarrow B}$, 存在一個 edit sequence S taking A to B , 使得:

$$\gamma(S) = \gamma(T_{A \rightarrow B}).$$

(性質 2) 對所有 edit sequence S taking A to B , 存在一個 $T_{A \rightarrow B}$, 使得:

$$\gamma(T_{A \rightarrow B}) \leq \gamma(S).$$

所以我們知道

$$\delta(A, B) = \min(\gamma(T_{A \rightarrow B})).$$

3 解法敘述

3.1 Computation of Edit Distance

3.1.1 Dynamic Programming

透過觀察，我們可以發現其實 *change* 等於 *delete* 後再 *insert*，所以

$$\gamma(a \rightarrow b) = \gamma(a \rightarrow \emptyset) + \gamma(\emptyset \rightarrow b) \quad (*)$$

現在要來尋找將 A edit 成 B 花費最少的 S 。由式 $(*)$ ，我們給定以下的花費函數 (cost function):

- 改變 (*change*): $\gamma(a \rightarrow b) = \2 , 其中 $a \neq b$
- 刪除 (*delete*): $\gamma(a \rightarrow \emptyset) = \1
- 插入 (*insert*): $\gamma(\emptyset \rightarrow b) = \1
- 不變 (*doNothing*): $\gamma(a \rightarrow b) = \0 , 其中 $a = b$

將一個字串 $A = a_1a_2 \dots a_i$ edit 成字串 $B = b_1b_2 \dots b_j$ 的 *edit distance* $\delta(A, B)$ 用動態規劃的方式寫成:

$$\begin{aligned} \delta(A, B) = \delta(A_{1..i}, B_{1..j}) = \min\{ & \delta(A_{1..i-1}, B_{1..j-1}) + \delta(a_i \rightarrow b_j), \\ & \delta(A_{1..i-1}, B_{1..j}) + \delta(a_i \rightarrow \emptyset), \\ & \delta(A_{1..i}, B_{1..j-1}) + \delta(\emptyset \rightarrow b_j)\}. \end{aligned}$$

舉例來說，假設我們有兩字串 $A = \text{"alignment"}$ 和 $B = \text{"algorithm"}$ ，我們想要把 A edit 成 B ，可以畫出如下的花費表格 (*doNothing*: ↖, *delete*: ↑, *insert*: ←):

(因為 *change* 是由 *delete* 和 *insert* 所組成的，所以我們省去此 *edit operation*)

$A \backslash B$	\emptyset	a	l	g	o	r	i	t	h	m
\emptyset	0, ↖	1, ←	2, ←	3, ←	4, ←	5, ←	6, ←	7, ←	8, ←	9, ←
a	1, ↑	0, ↖	1, ←	2, ←	3, ←	4, ←	5, ←	6, ←	7, ←	8, ←
l	2, ↑	1, ↑	0, ↖	1, ←	2, ←	3, ←	4, ←	5, ←	6, ←	7, ←
i	3, ↑	2, ↑	1, ↑	2, ←	3, ←	4, ←	3, ↖	4, ←	5, ←	6, ←
g	4, ↑	3, ↑	2, ↑	1, ↖	2, ←	3, ←	4, ←	5, ←	6, ←	7, ←
n	5, ↑	4, ↑	3, ↑	2, ↑	3, ←	4, ←	5, ←	6, ←	7, ←	8, ←
m	6, ↑	5, ↑	4, ↑	3, ↑	4, ←	5, ←	6, ←	7, ←	8, ←	7, ↖
e	7, ↑	6, ↑	5, ↑	4, ↑	5, ←	6, ←	7, ←	8, ←	9, ←	8, ↑
n	8, ↑	7, ↑	6, ↑	5, ↑	6, ←	7, ←	8, ←	9, ←	10, ←	9, ↑
t	9, ↑	8, ↑	7, ↑	6, ↑	7, ←	8, ←	9, ←	8, ↖	9, ←	10, ←

由表格知 $\delta(A, B) = \delta(\text{"alignment"}, \text{"algorithm"}) = 10$; 透過 backtracking 可寫出由 A edit 到 B 的過程:

$$\text{alignment.delete}(i) = \text{alignment} \quad (1)$$

$$\text{alignement.delete}(n) = \text{alignment} \quad (2)$$

$$\text{algment.delete}(m) = \text{algment} \quad (3)$$

$$\text{algment.delete}(e) = \text{algnt} \quad (4)$$

$$\text{algnt.delete}(n) = \text{algt} \quad (5)$$

$$\text{algt.insert}(o) = \text{algot} \quad (6)$$

$$\text{algot.insert}(r) = \text{algorit} \quad (7)$$

$$\text{algorit.insert}(i) = \text{algorit} \quad (8)$$

$$\text{algorit.insert}(h) = \text{algorithm} \quad (9)$$

$$\text{algorithm.insert}(m) = \text{algorithm} \quad (10)$$

當然這不是唯一解，在我的演算法中在計算 δ 的 min 要取哪一項若發生等於的情況，會優先選擇 *insert operation*，但不論選擇哪一項符合 min 的 *edit operation*， $\delta(A, B)$ 都會等於 10。

在這個例子中， $S = d_i d_n d_m d_e d_n i_o i_r i_i i_h i_m$ ， $|S| = 10$ ，其中 d 和 i 分別代表刪除 (*delete*) 和插入 (*insert*)，下標的英文字母代表為對哪個字元作操作。

因為此演算法要將整個動態規劃的表格填滿，故時間複雜度為 $O(|A| \times |B|)$ 。

4 讀後心得

在閱讀完〈The String-to-String Correction Problem〉後，我對於字串之間的操作有了更進一步的了解，很佩服原作者 Robert A. Wagner 和 Michael J. Fischer 能夠在 1974 年這個資訊還不是很普及的時候，提出了這樣一篇強而有力的論文，他們提出 *edit distance* 這個概念，讓字串間的操作有了新的理解，並且定義了花費函數 (cost function)，讓問題變得更加明確「找出 $\delta(A, B)$ 和對應的 S 」。

在寫這份報告同時，也上網查了很多有關的例子，發現許多有趣的應用，像是生物的基因比對、git 上面的 diff 指令、經典的 *Longest Common Subsequence(LCS)* 問題等等。

此篇論文中有提到 $\delta(A, B)$ 的概念，說的白話一點就是我們不只希望能找到由 A edit 到 B 的過程 (S)，同時我們還希望這個 *edit distance* 是 *minimum* 的，假設當我們看到以下三個基因序列：

1. AGCCT
2. AACCT
3. ATCT

我們可能可以很直覺的說「2. 和 1. 的相似程度比 3. 和 1. 還要高」，但能夠這樣用肉眼看出是因為我們舉的例子還很小，「肉眼」還感覺的出差異性，但一旦我們的序列變得更長，不再是屬於同一個數量級別時，如何有一個有效的量化標準就變得很重要了，正好 *minimum edit distance* $\delta(A, B)$ 就能完成這件事。若要比較兩個基因序列 $A = \text{"CTTA ACT"}$ 和 $B = \text{"CGGATCAT"}$ 的相似程度，我們可以透過計算 *edit distance* $\delta(A, B) = \delta(\text{"CTTA ACT"}, \text{"CGGATCAT"})$ ，來做為一個標準，當 $\delta(A, B)$ 較小時，這代表這兩個基因序列相似程度高；反之，代表相似程度低。

edit distance 演算法的時間複雜度是 $O(|A||B|) = O(n^2)$ ，但人總是希望能夠做的更好，而 2015 年 MIT 的一篇論文〈Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)*〉就有提到這個時間複雜度是無法再被下降的。

在這個資訊蓬勃發展的二十一世紀，*edit distance* 或許已經稱不上什麼太高深的演算法，但是它歷久而彌新，在現今很多的應用中都能看見它的影子，並且默默的推動著 Computer Science 的發展。

5 參考資料

- [1] 〈The String-to-String Correction Problem〉
- [2] 〈Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)*〉
- [3] Edit distance - Wikipedia
- [4] Sequence alignment - Wikipedia
- [5] Levenshtein distance - Wikipedia
- [6] String-to-string correction problem - Wikipedia
- [7] The Myers diff algorithm: part 1 –The If Works
- [8] 自然語言處理 – Minimum Edit Distance