動態規畫技巧簡介
(Dynamic Programming)

趙坤茂

陽明大學生命科學系

Email: kmchao@ym.edu.tw

WWW: http://www.ym.edu.tw/~kmchao

The Heaviest Non-decreasing Subsequence Problem

- Let *S* be a sequence of integers.
- A heaviest non-decreasing subsequence of *S* is a non-decreasing subsequence with the maximum sum.

(這是2000年全國大專軟體設計競賽大學甲組的試題)

動態規畫技巧與序列分析

- 費氏數(Fibonacci number)
- 最長共同子序列
- 兩個序列的分析
- 多重序列分析

費氏數(Fibonacci number)

費氏數(Fibonacci number)

費氏數(Fibonacci number)可用下列的遞迴關係(recurrence)來描述：

$$F_0 = 0$$

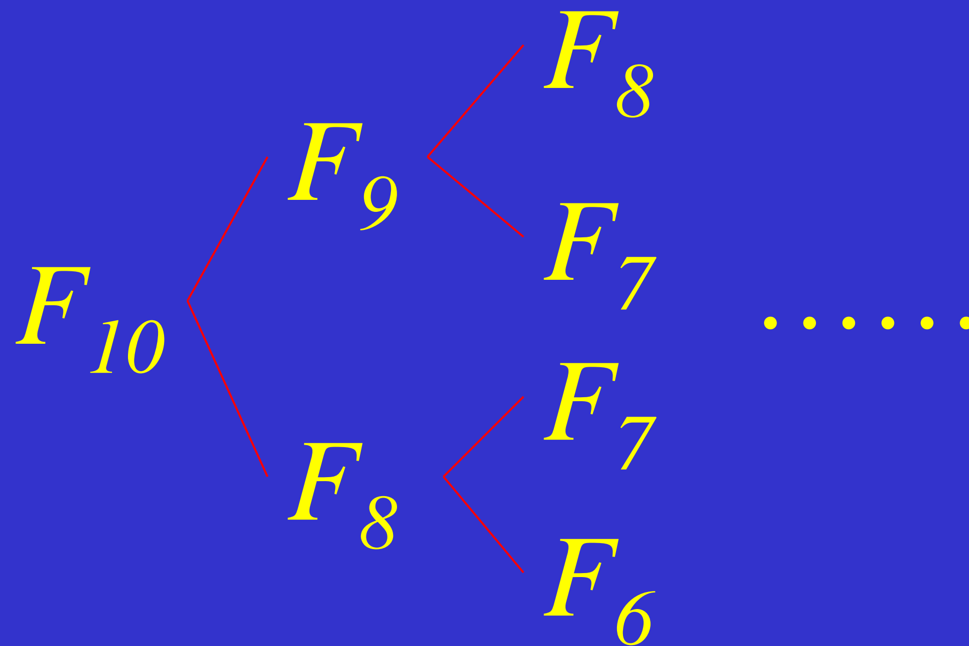
$$F_1 = 1$$

$$F_i = F_{i-1} + F_{i-2} \quad \text{for } i \geq 2.$$

How to compute F_{20}

- 如果想知道F_{20} 的值是多少，有人可能會以程式語言中的遞迴呼叫(recursive call)這麼做：先試著去求得F_{19} ，然後再設法求F_{18} ，最後再將兩個加起來。
- 而要如何求得F_{19} 呢？那這還不簡單嗎？將F_{18} 及 F_{17} 算出來就可以了呀！Wait a minute ! F_{18} 不是已經算過了嗎？為何現在又要重算了呢？實際上，以遞迴呼叫來處理這樣的問題，重算的次數還真嚇人呢！

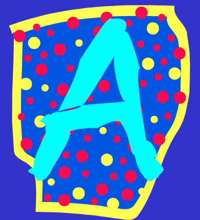
請問F_{10}是多少？


F_{10}
F_9
F_8
F_8
F_7
F_7
F_6
……

列表式計算

- 如果我們從F_0、F_1、...往F_{10}邁進，很快我們就可以算出F_{10}

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
0	1	1	2	3	5	8	13	21	34	55



好漢做事好漢當

上代數的時候，小明同學在後面..z...z...z.... Z 。

後來，老師發現了，就生氣地說：

「旁邊的同學，把睡覺的叫起來！」

語畢， 不知道是哪個活得不耐煩的同學答道：

「是你自己把他弄睡著的，你自己去把他叫醒...」

最長共同子序列

動態規畫(dynamic programming)

基本上，動態規畫技巧有三個主要部份：

- 遞迴關係(recurrence relation)
- 列表式運算(tabular computation)
- 路徑迴溯(traceback)

「最長共同子序列」(LCS, Longest Common Subsequence)問題

- 首先我們先解釋什麼是子序列(subsequence)，所謂子序列就是將一個序列中的一些(可能是零個)字元去掉所得到的序列，例如：pred、sdn、predent等都是”president”的子序列。
- 給定兩序列，最長共同子序列(LCS)問題是決定一個子序列，使得 (1) 該子序列是這兩序列的子序列；(2) 它的長度是最長的。

LCS

例如：

序列一：president

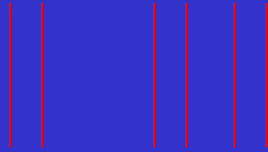
序列二：providence

它的一個LCS為

LCS

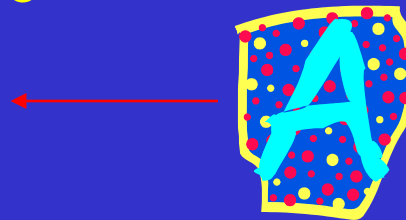
例如：

序列一：president



序列二：providence

它的一個LCS為 priden



(PResIDENt

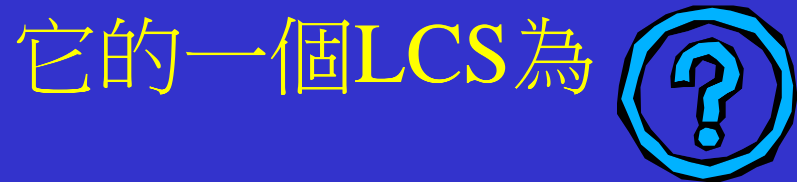
PRovIDENce)

LCS

又例如：

序列一：algorithm

序列二：alignment

它的一個LCS為 

LCS

又例如：

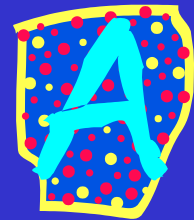
序列一：algorithm

序列二：alignment

它的一個LCS為 algm or algt

(ALGorithM

ALiGnMent)



How to compute LCS?

- 給定兩序列及，令*len(i, j)*表示與的LCS之長度，則下列遞迴關係可用來計算*len(i, j)*：

$$len(i,j)=\begin{cases}0 & \text{if } i=0 \text{ or } j=0,\\ len(i-1,j-1)+1 & \text{if } i,j>0 \text{ and } a_i=b_j,\\ \max(len(i,j-1),len(i-1,j)) & \text{if } i,j>0 \text{ and } a_i\neq b_j.\end{cases}$$

procedure *LCS-Length(A, B)*

1. **for** $i \leftarrow 0$ **to** m **do** $len(i,0) = 0$
2. **for** $j \leftarrow 1$ **to** n **do** $len(0,j) = 0$
3. **for** $i \leftarrow 1$ **to** m **do**
4. **for** $j \leftarrow 1$ **to** n **do**
5. **if** $a_i = b_j$ **then** $\left[\begin{array}{l} len(i,j) = len(i-1,j-1)+1 \\ prev(i,j) = \text{"}\nwarrow\text{"} \end{array}\right.$
6. **else if** $len(i-1,j) \geq len(i,j-1)$
7. **then** $\left[\begin{array}{l} len(i,j) = len(i-1,j) \\ prev(i,j) = \text{"}\uparrow\text{"} \end{array}\right.$
8. **else** $\left[\begin{array}{l} len(i,j) = len(i,j-1) \\ prev(i,j) = \text{"}\leftarrow\text{"} \end{array}\right.$
9. **return** *len* and *prev*

i	*j* *0*	*1* *p*	*2* *r*	*3* *o*	*4* *v*	*5* *i*	*6* *d*	*7* *e*	*8* *n*	*9* *c*	*10* *e*
0	0	0	0	0	0	0	0	0	0	0	0
1 *p*	0	↖ 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
2 *r*	0	↑ 1	↖ 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2
3 *e*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↖ 3	← 3	← 3	↖ 3
4 *s*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↑ 3	↑ 3	↑ 3	↑ 3
5 *i*	0	↑ 1	↑ 2	↑ 2	↑ 2	↖ 3	← 3	↑ 3	↑ 3	↑ 3	↑ 3
6 *d*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3					
7 *e*	0										
8 *n*	0										
9 *t*	0										

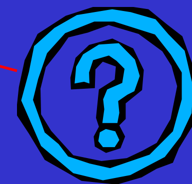


圖: 以*LCS-Length*計算president與providence的LCS。

i	*j* *0*	*1* *p*	*2* *r*	*3* *o*	*4* *v*	*5* *i*	*6* *d*	*7* *e*	*8* *n*	*9* *c*	*10* *e*
0	0	0	0	0	0	0	0	0	0	0	0
1 *p*	0	↖ 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
2 *r*	0	↑ 1	↖ 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2
3 *e*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↖ 3	← 3	← 3	↖ 3
4 *s*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↑ 3	↑ 3	↑ 3	↑ 3
5 *i*	0	↑ 1	↑ 2	↑ 2	↑ 2	↖ 3	← 3	↑ 3	↑ 3	↑ 3	↑ 3
6 *d*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↖ 4	← 4	← 4	← 4	← 4
7 *e*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↖ 5	← 5	← 5	↖ 5
8 *n*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↑ 5	↖ 6	← 6	← 6
9 *t*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↑ 5	↑ 6	↑ 6	↑ 6

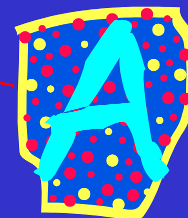


圖: 以*LCS-Length*計算president與providence的LCS。

```
procedure Output-LCS(A, prev, i, j)
1   if i = 0 or j = 0 then return
2   if prev(i, j)=" ↖ " then [ Output-LCS(A, prev, i-1, j-1)
                               print a_i
3   else if prev(i, j)=" ↑ " then Output-LCS(A, prev, i-1, j)
4   else Output-LCS(A, prev, i, j-1)
```

i	*j* *0*	*1* *p*	*2* *r*	*3* *o*	*4* *v*	*5* *i*	*6* *d*	*7* *e*	*8* *n*	*9* *c*	*10* *e*
0	0	0	0	0	0	0	0	0	0	0	0
1 *p*	0	↖ 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
2 *r*	0	↑ 1	↖ 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2
3 *e*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↖ 3	← 3	← 3	↖ 3
4 *s*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↑ 3	↑ 3	↑ 3	↑ 3
5 *i*	0	↑ 1	↑ 2	↑ 2	↑ 2	↖ 3	← 3	↑ 3	↑ 3	↑ 3	↑ 3
6 *d*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↖ 4	← 4	← 4	← 4	← 4
7 *e*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↖ 5	← 5	← 5	↖ 5
8 *n*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↑ 5	↖ 6	← 6	← 6
9 *t*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↑ 5	↑ 6	↑ 6	↑ 6

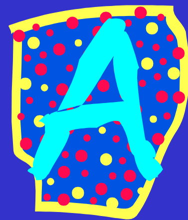
圖： *Output-LCS*的回溯路線，深色陰影(priden)為LCS所在。

極樂世界

- 老師：「 小明，你的毛病就是用詞不當，現在考考你 ... 。用一句成語來形容老師很開心，而且成語中要包含數字。」
- 小明：「 」

極樂世界

- 老師：「 小明，你的毛病就是用詞不當，現在考考你 ... 。用一句成語來形容老師很開心，而且成語中要包含數字。」
- 小明：「含笑九泉」



兩個序列的分析

兩個序列的分析

- 在1970年代，分子生物學家Needleman 及Wunsch [15]以動態程式設計技巧(dynamic programming)分析了氨基酸序列的相似程度；
- 有趣的是，在同一時期，計算機科學家Wagner及Fisher [22]也以極相似的方式來計算兩序列間的編輯距離(edit distance)，而這兩個重要創作當初是在互不知情下獨立完成的。
- 雖然分子生物學家看的是兩序列的相似程度，而計算機科學家看的是兩序列的差異，但這兩個問題已被證明是對偶問題(dual problem)，它們的值是可藉由公式相互轉換的。

三種常用的序列分析方法

目前序列分析工具可說是五花八門 ，儘管如此，有三種構想是較受大家所青睞的：

- 第一種是**Smith-Waterman**的方法，這種方法很精細地計算兩序列間最好的*k* 個區域排比(local alignment)，雖然這個方法很精確，但因耗時較久，所以多半應用在較短序列間的比較，然而，也有一些學者試著去改善它的一些計算複雜度，使它在長序列的比較上也有一些實際的應用。

三種常用的序列分析方法(續)

- 第二種是Pearson的FASTA方法，這種方法先以較快方式找到一些有趣的區域，然後再以Smith-Waterman的方法應用在那些區域中。如此一來，它的計算速度就比Smith-Waterman快，而且在很多情況下，它的精細程度也不差。
- 第三種是Altschul等人所製作的BLAST ，它的最初版本完全沒有考慮間隔(gap)，所以在計算上比其他方式快了許多。雖然它不夠經細，但它的計算速度使得它在生物序列資料庫的搜尋上有很大的優勢，也因此它可說是目前最受歡迎的序列分析工具。此外，1997年剛出爐的Gapped BLAST已針對精細程度做了很大的改進，且在計算速度上仍維持相當的優勢。

為什麼要用排比(alignment)？

- 早期的序列分析通常是以點矩陣(dot matrix)方法來進行的，這種方法是以二維平面將兩序列間相同的地方點出來，從而藉由目視的方式看看兩序列有那些相似的地方。這種方法最大的優點是一目了然且計算簡單；
- 然而，當序列較長的時候，藉由目視方法去分析它們是一種很沒有效率的方式
- 況且有些生物序列(如蛋白質序列)並不是只有相同字符才相似，這時候點矩陣方法就無法看出整體的相似程度。
- 於是有人建議以排比(alignment)來顯示兩序列的相似程度

排比(alignment)

- 給定兩序列，它們的整體排比(global alignment)是在兩序列內加入破折號(dash)，使得兩序列變得等長且相對的位置不會同時都是破折號。
- 例如：假設兩序列為ＣＴＴＧＡＣＴＡＧＡ及ＣＴＡＣＴＧＴＧＡ，下圖列出了它們的一種排比。

ＣＴＴＧＡＣＴ－ＡＧＡ
ＣＴ－－ＡＣＴＧＴＧＡ

圖: ＣＴＴＧＡＣＴＡＧＡ及ＣＴＡＣＴＧＴＧＡ的一種可能排比。

排比的評分方式

- 有那麼多種不同的排比組合，到底要挑那一個排比呢？為了要挑出較理想的排比，通常我們需要一些評分方式來做篩選工作。
- 最簡單的評分方式是將每一個配對基底(aligned pair)都給一個分數，再看看那一種排比的總分最高。令$w(a,b)$代表a與b配對所得到的分數(通常$w(*,-)$及$w(-,*)$是負值；mismatch也是負值；只有match是正值，而蛋白質序列分析則採用PAM矩陣或BLOSUM矩陣來決定這些值)
- 在上述的簡單評分原則下，前圖的排比所得到的分數為$w(C,C) + w(T,T) + w(T,-) + \ldots + w(A,A)$

最佳排比演算法

我們希望找的是一個分數最高的排比 **(optimal alignment)**，而這要如何達成呢？令$S(i, j)$代表$a_1a_2...a_i$ 與$b_1b_2...b_j$ 最好的排比分數，在適度設定初始值後，它的值可用下列的遞迴關係**(recurrence relation)**來計算：

$$S(i,j)=\max\begin{cases}S(i-1,j-1)+w(a_i,b_j)\\S(i-1,j)+w(a_i,-)\\S(i,j-1)+w(-,b_j)\end{cases}$$

「同盟線性評分法」(affine gap penalties)

- 我們可以用動態規畫技巧由小到大依序將*S(i, j)*算出，並且記錄最佳值的由來，如此一來，在計算完了之後，我們也能一舉將最佳排比回溯出來。
- 在比較生物序列時，我們通常會對每個破折號區域另外扣一個懲罰分數(令其為α)，破折號區域也就是我們常說的「間隔」(gap)，如果破折號發生在第一個序列我們稱之為「插入間隔」(insertion gap)；如果發生在第二個序列我們稱之為「刪除間隔」(deletion gap)。
- 例如在前圖的排比中，我們有一個長度為*2*的刪除間隔及一個長度為*1*的插入間隔，所以在排比分數上還要扣去兩個間隔的分數(2α)。我們通常稱這樣的評分方式為「同盟線性評分法」(affine gap penalties)

最佳排比演算法

我們要如何計算最佳的排比呢？令$D(i, j)$代表$a_1a_2 \ldots a_i$與$b_1b_2 \ldots b_j$間以刪除間隔結束的排比之最佳分數；令$I(i, j)$代表$a_1a_2 \ldots a_i$與$b_1b_2 \ldots b_j$間以插入間隔結束的排比之最佳分數；令$S(i, j)$代表$a_1a_2 \ldots a_i$與$b_1b_2 \ldots b_j$間任何排比之最佳分數，則在適度的初始值設定後，下列的遞迴關係可用來計算這三個值：

最佳排比演算法(續)

$$D(i,j)=\max\begin{cases}D(i-1,j)+w(a_i,-)\\ S(i-1,j)+w(a_i,-)-\alpha\end{cases}$$

$$I(i,j)=\max\begin{cases}I(i,j-1)+w(-,b_j)\\ S(i,j-1)+w(-,b_j)-\alpha\end{cases}$$

$$S(i,j)=\max\begin{cases}S(i-1,j-1)+w(a_i,b_j)\\ D(i,j)\\ I(i,j)\end{cases}$$

遞迴關係的白話解說

- 雖然上面的遞迴關係有些複雜，但在計算上也是很直截了當的。我們利用矩陣*D*來記錄以刪除間隔結束的最佳分數，如果緊接著又是刪除動作時，我們就不必再扣懲罰分數了；
- 同樣地，我們利用矩陣*I*來記錄以插入間隔結束的最佳分數，如果緊接著又是插入動作時，我們就不必再扣懲罰分數了。
- 我們可以用動態規畫技巧由小到大依序將$D(i, j)$、$I(i, j)$及$S(i, j)$算出，並且記錄最佳值的由來，如此一來，在計算完了之後，我們也能一舉將最佳排比回溯出來。

區域排比(local alignment)

- 在做生物序列排比時，有時更有趣的是找出局部區域的相似程度，此時我們考慮的是所謂的區域排比(local alignment)，也就是我們不必從頭到尾排比整個序列，而只要找出序列一的某個區段和序列二的某個區段之最佳排比即可。
- 我們在此以最簡單的評分方式(也就是以每一個配對基底(aligned pair)分數的總和為排比分數)來說明如何計算最佳區域排比。

最佳區域排比的演算法

令$S(i, j)$代表$a_1a_2...a_i$ 與$b_1b_2...b_j$ 最好的區域排比分數，在適度設定初始值後，它的值可用下列的遞迴關係(recurrence relation)來計算：

$$S(i,j) = \max \begin{cases} 0 \\ S(i-1, j-1) + w(a_i, b_j) \\ S(i-1, j) + w(a_i, -) \\ S(i, j-1) + w(-, b_j) \end{cases}$$

為什麼要加個0？

- 和整體排比(global alignment)的遞迴關係相比較，你會發現這裡的遞迴關係只多了*0*這一項，原因是整體排比要從序列前端開始排起，而區域排比卻是任一個地方都可能是個起點，如果往前連接分數小於*0*，我們就不該往前串聯，而以此點做為一個起點($S(i, j)=0$)試試看。

多個最佳區域排比

- 有些人感興趣的是找出*k* 個最好的區域排比或是分數至少有設定值那麼高的所有區域排比，這樣的計算在你熟悉動態規畫技巧後應不至於難倒你的。
- 上述的方式也就是一般俗稱的Smith-Waterman方法 (實際上，整體排比問題是由**Needleman**及**Wunsch**[15]所提出；而區域排比問題則是由**Smith**及**Waterman**[21]所提出)，它基本上需要與兩序列長度乘積成常數正比的時間與空間。
- 在序列很長時，這種計算時間及空間都是很難令人接受的！

FASTA

- 在分析兩個序列時，Smith-Waterman的方法也許勉強可以接受，但如果以它做為資料庫搜尋的引擎，那就有些龜速了，因為這將會耗費不少寶貴的時光。
- FASTA(唸成”fast-AY”，而不是”FAST-uh”)及BLAST(Basic Local Alignment Search Tool，唸成BLAST)就這樣應運而生了。
- 這兩種方法都設法不去計算動態規畫設計表(dynamic programming table)的每一點，而是以近似方法(heuristic mathod)來做排比，因此在計算速度上勝過Smith-Waterman方法很多，非常適合做資料庫搜尋。

FASTA(續)

- 首先，讓我們約略地看一下FASTA這個方法，這個方法讓使用者選定參數*ktup* (k-tuple，在DNA序列分析時這個值通常設成 *6* 到 *8* ；蛋白質序列則常使用 *1* 到 *2* 的值)，FASTA只考慮那些長度至少為 *ktup* 的那些相似子序列，試著藉由它們找出一些可能有相似性的對角區域(diagonal band)，然後再將Smith-Waterman的方法套用在這些小區域上。在BLAST橫掃千軍前，FASTA曾是最常被用來分析序列的工具。
- 在FASTA中，因為它試著去串聯那些 *ktup*序列，所以耗費了不少時間。

BLAST

- 初版的BLAST則以長度至少為 *w* 的相似區段著眼，只往對角線(diagonal)方向試著去延伸，直到分數的降低程度超過使用者所給定的範圍為止，因為它完全不考慮間隔(gap)，所以非常地有效率，但缺點是有時不夠敏銳(sensitive)。
- 不過在第二版的BLAST中，已針對這樣的缺點加以修正，它在延伸對角線時採用的策略是跳著延伸(註：延伸對角線耗去了大部份初版BLAST執行的時間)，這個立論基礎是如果它真是分數很高的相似區段，跳著延伸也不會錯過。

BLAST(續)

- 利用延伸對角線所省下的時間，新的BLAST也試著去考慮串聯一些分數高的相似區段，也就是說某些有間隔的排比在新的BLAST中也可被發現。新版的BLAST已幾乎取代初版的BLAST。

龜兔賽跑

阿強上課總是打瞌睡，
老師忍無可忍地把昏睡中的阿強叫醒，並問他
"龜兔賽跑中，你知道兔子為什麼會輸嗎？"
"不知道"阿強睡眼惺忪地回答。
"因為兔子在打瞌睡"老師生氣的說....
"喔！我明白了"阿強若有所悟
"原來沒打瞌睡的全是烏龜啊！"

多重序列分析

多重序列排比

- 多序列的分析一直是計算生物學上很重要的課題，但是它的問題複雜度卻令人沮喪。粗略地說，比較兩個長度皆為n的序列所需的時間(也就是動態規畫矩陣點)是和n的平方成常數正比的；而比較k個長度皆為n的序列所需的時間則和n的k次方成常數正比。
- 試想如果我們要同時比較10個長度只有200的序列所需的時間是多少呢？它基本上會和200的10次方成常數正比，而這卻是很龐大的數目。

多重序列排比的計算方法

- 因此，在計算方法上有兩種不同的流派：一種是**Lipman**等人提出的方式，他們做的是同時比較多個序列，但試著去降低計算時所用的動態規畫矩陣點，據他們的論文指出，這種方式比較10個長度為200的序列也不會遭遇太大的問題；
- 另一種方式是**Feng**及**Doolittle**所採用的，它根據序列遠近程度的演化樹來做序列排比，一旦**gap**在某個比較中出現後，它就會被保留到最後，這種方法用來比較k個長度皆為n的序列所需時間約略與成常數正比，所以非常廣受歡迎。

多重序列排比的評分方式

- 最廣為接受的一種方式稱為*SP* (*Sum-of –Pairs*)分數，這種方式將多重序列排比投影到每一對序列上所得的排比分數總和起來，做為該多重排比的分數。
- 這種方式若要直接採用「同盟線性評分方式」，則會滋生非常多的動態規畫設計表，但若稍稍放鬆一下，「類似同盟線性評分方式」(quasi affine gap penalties)雖然不夠精準，但卻可較有效率地計算多重排比的分數，是最常被用到的變形評分方式。
- 此外，有些人建議某些序列組合應加權計分；也有人根據演化樹來計算分數

都是老師惹的禍

- 在經過了數天的苦讀熬夜，甚至通宵之後，三個學生總算通過了期末考，考完當天：
- 甲生說：這樣熬下來，我看起來真是憔悴，你們看看我的臉，彷彿老了十年呢.....
- 乙生說：你那樣還好，我看起來才糟呢，我看起來像只剩十年可活了...嗚~~~~
- 丙生嘆了一口氣說：唉~~~~我看起來像是已經死了十年了~~~~