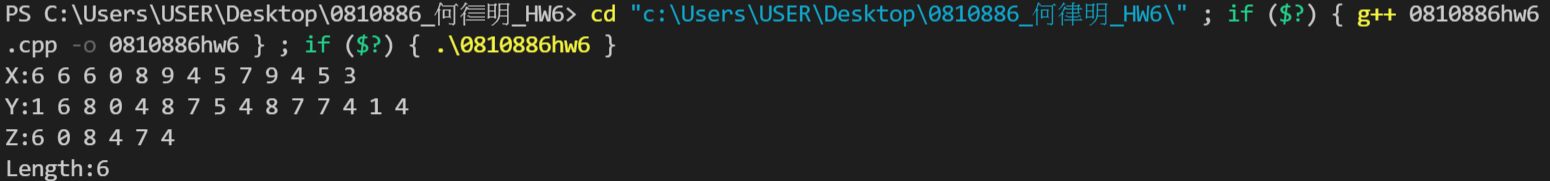
【作業要求】

1.

Result:



結果為6, 0, 8, 4, 7, 4，長度為6。

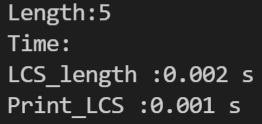
2.

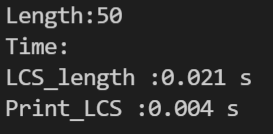
Result:

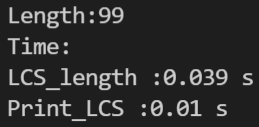
LCS\_length,

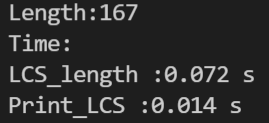


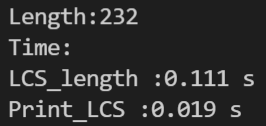
Z的結果如下

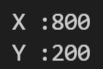
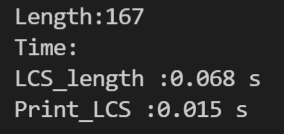
🡪 

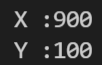
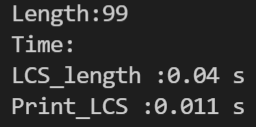
🡪 

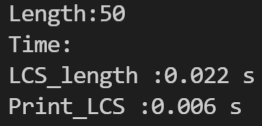
🡪 

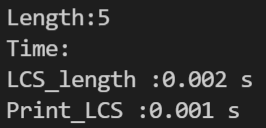
🡪 

🡪 

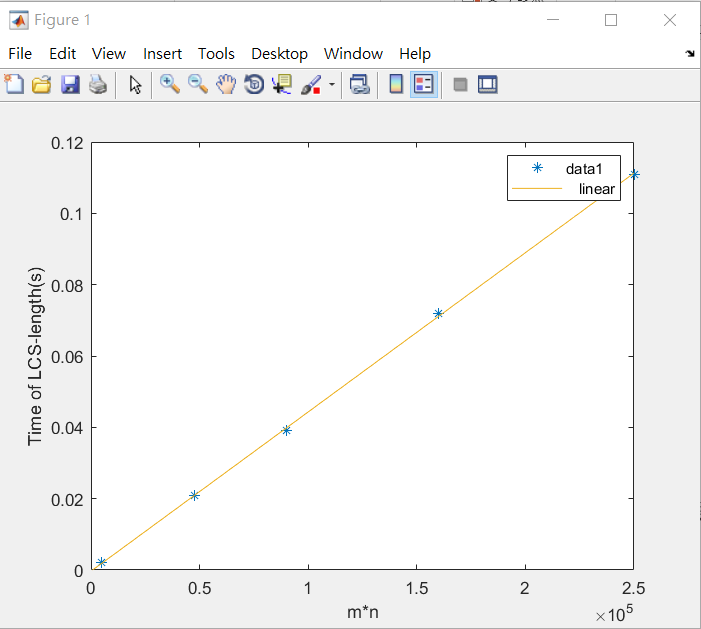
🡪 

🡪 

🡪 

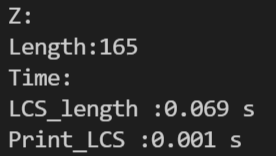
🡪 

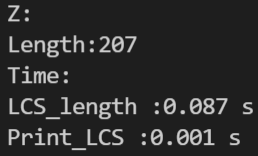
m\*n 對 LCS\_length time的圖(using matlab):

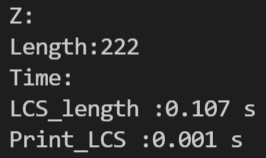


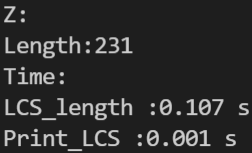
很明顯可以看出LCS\_length的執行時間(複雜度)為，因為m\*n與執行時間幾乎呈一直線。也符合老師上課所說的複雜度。

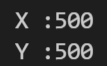
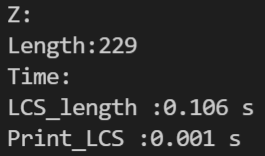
至於老師所說的Print\_LCS複雜度為，可是上面附圖中明明m+n都固定為1000，Print\_LCS的時間卻差很多，照理來說，應該要時間都差不多才對啊，是否是複雜度錯誤了? 其實不然，我發現上述的時間不準是因為Print\_LCS包含了程式印出Z的時間，於是我把真正印出數據的部分拿掉，只讓Print\_LCS執行，但不列印出Z的內容發現如下:

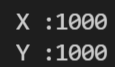
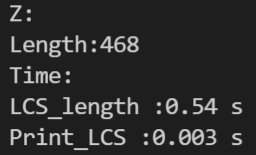
🡪

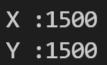
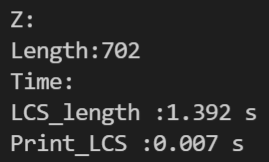
🡪

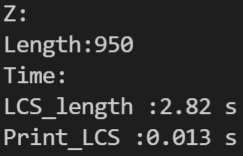
🡪

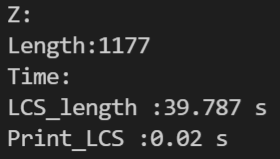
🡪 ，此時Print\_LCS的執行時間都為0.001秒，也很明顯的Print\_LCS複雜度的確為。但這個結論還有另外一個條件就是，當X<<Y，或者是說X>>Y，的時候，Z的大小會偏向其中一個，此時b table的斜向元素會很多走接近m或者是說接近n就走到了0的位置，所以說並部會走到m+n那麼多步。這也是造成上面其中幾張圖其中一個原因，所以說我們盡量取X跟Y的大小不要差太多的情況去測試，如下:

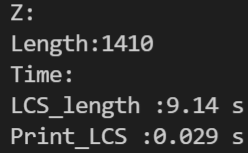
🡪 

🡪 

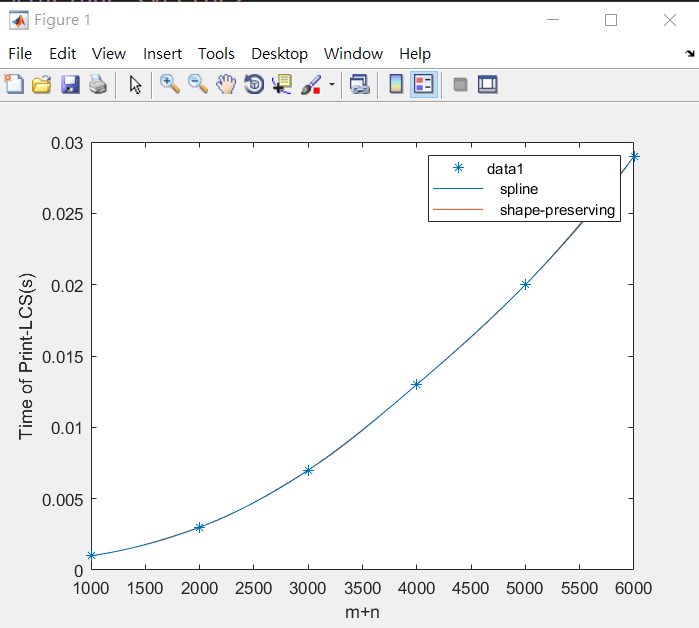
🡪 

🡪 

🡪 

🡪 

m+n 對 time of Print\_LCS的圖(using matlab):

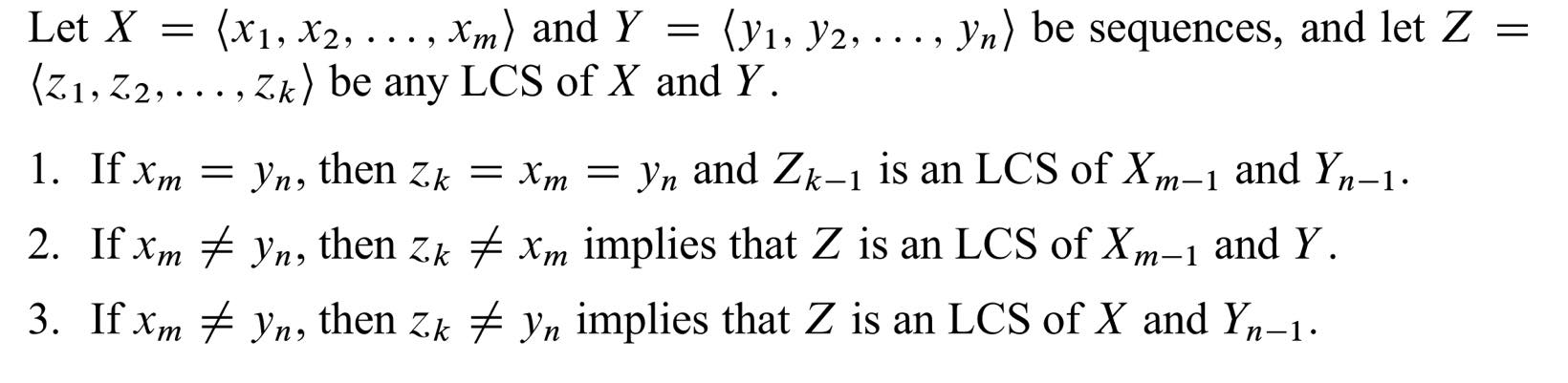
可以發現當m+n夠大(超過3000)，time 會趨近於一條線。

【討論】

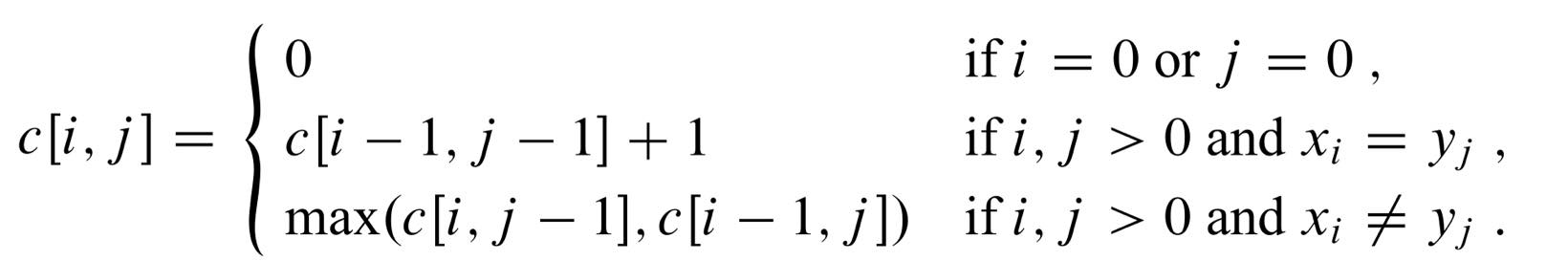
窮舉法:

假設X的長度為m，代表說我們要取X的subsequence，每一個元素可以有取跟不取兩種，所以說X總共有種不同的subsequences，而長度為n的則有種不同的subsequences，去比較X每一種的subsequence跟Y每一種subsequence總共有種不同的X跟Y配對，所以說複雜度為big theta of 。

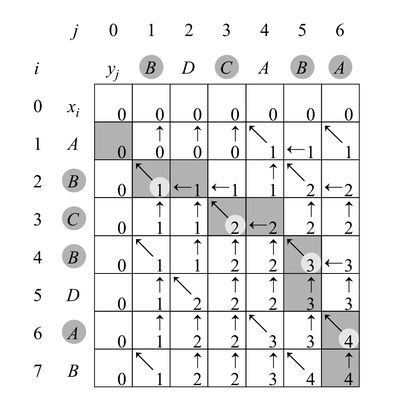
Dynamic programming:



意味著(Xm, Yn)的LCS一定可以從(Xm-1, Yn)、(Xm, Yn-1)、(Xm-1, Yn-1)得知，我們可以把各m跟n的值所賦予的(Xm, Yn)LCS儲存在一個名為C的table(2-d array)中，並有著下的關係:



儲存在表中的同時，我們也可以知道(Xm, Yn)的是從(Xm-1, Yn)、(Xm, Yn-1)、(Xm-1, Yn-1)中的哪一個得知的，所以可以得到以下的表:

從最右下的格子(代表(Xm, Yn)的長度) ，照著箭頭走經過朝向左上方的箭頭代表該(Xi, Yj)相同，所以說LCS一定包含Xi，一直走到0一路上經過的左上方的箭頭Xi就是LCS中的各個元素。但要注意的為一路上經過的為Reverse order，所以在輸出的時候可以用recursive的方法把他再次反過來，即為正確的答案LCS。至於複雜度的部分因為這個表高度為m，寬度為n，所以說走最多m+n步一定可以走到原點(0)，正因為如此複雜度為。遠遠好過窮舉法。也正因為有著optimal substructure的特性所以說能有效地解決問題。