Dynamic Programming
動態規劃

林劭原老師

林劭原 老師

前言

- Three useful algorithm design techniques:
- divide-and-conquer approach
- dynamic programming
- greedy method

Longest common subsequence(LCS)最長共同子字串

- A subsequence 子字串 keeps the original order, but it may not be consecutive.
- The longest-common-subsequence(LCS) problem is given two sequences

$$X =< x_1, x_2, \dots, x_m > and \ \mathrm{Y} =< y_1, y_2, \dots, y_n >,$$

Find a maximum-length common subsequence of X and Y.

- 最佳解可能不只1個，求出1個即可。

林劭原 老師

Longest common subsequence(LCS)最長共同子字串

- DNA sequences are formed by A.C.G.T.
- If $s_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA$ and $s_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA$

  then $s_3 = GTCGTCGGAAGCCGGCCGAA$ is an LCS of s_1 and s_2

Longest common subsequence(LCS)最長共同子字串

- Brute force 暴力法
- Find all subsequences of X and check each subsequence to see whether it is also a subsequence of Y, keeping track of the longest subsequence we find.
  Since X has 2^m subsequences, this method takes exponential time.指數型時間

Longest common subsequence(LCS)最長共同子字串

- Dynamic programming
  「下面的推導非常重要，dynamic programming 的推導模式幾乎都是這樣的」
- Let LCS(X,Y) = a maximum-length common subsequence of X and Y, and let LLCS(X,Y) = length of LCS(X,Y).
- Suppose X and Y end in the same letter.
  For example, X is ……..A and Y is ……..A.
  Then, LCS(X,Y) must end in A.
  So, LLCS(X,Y) = LLCS($< x_1, x_2, \ldots, x_{m-1} >$, $< y_1, y_2, \ldots, y_{n-1} >$)+1

Longest common subsequence(LCS)最長共同子字串

- Suppose X and Y do not end in the same letter.
  For example, X is ………A and Y is ………B.
  Let Z = LCS(X,Y).
  Then Z does not end in A or Z does not end in B
  Thus, Z = LCS $(< x_1, x_2, \dots, x_{m-1} >, < y_1, y_2, \dots, y_n >)$ or
  Z = LCS $(< x_1, x_2, \dots, x_m >, < y_1, y_2, \dots, y_{n-1} >)$.
- We don't know Z, but we can test both cases to see which one is better.

Longest common subsequence(LCS)最長共同子字串

- Let $X_i =< x_1, x_2, \ldots, x_i >, Y_j =< y_1, y_2, \ldots, y_j >$, and $Z_k =< z_1, z_2, \ldots, z_k >$.
- Theorem : Optimal substructure of an LCS
- If $x_m = y_n$, then $z_k = x_m = y_n$ and z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
- If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y_n.
- If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X_m and Y_{n-1}.

Longest common subsequence(LCS)最長共同子字串

- 具有 optimal substructure是指:
  最佳解答是由「子問題的解」構成，則這些「子問題的解」也必須是最佳解
- LCS problem has the property that an optimal solution contains within it optimal solutions to sub-problems.
  最佳解 包含了 子問題的最佳解
- This property is called optimal substructure.
- Problems solvable by dynamic programming must have optimal substructure.

Longest common subsequence(LCS)最長共同子字串

- For convenience, let c[i,j] = |LLCS(X_i, Y_j)|. Then:
- $c[i,j] = \begin{cases} 0 & if\ i = 0\ or\ j = 0 \\ c[i-1, j-1] + 1 & if\ i,j > 0\ and\ x_i = y_j \\ \max\{c[i, j-1], c[i-1], j\} & if\ i,j > 0\ and\ x_i \neq y_j \end{cases}$
- 由列式來看，會以為要用到遞迴程式，但其實不然，可以直接由最底層開始算.
- Dynamic programming often uses optimal substructure in a bottom-up fashion. Actually, no recursion is required.

dynamic programm

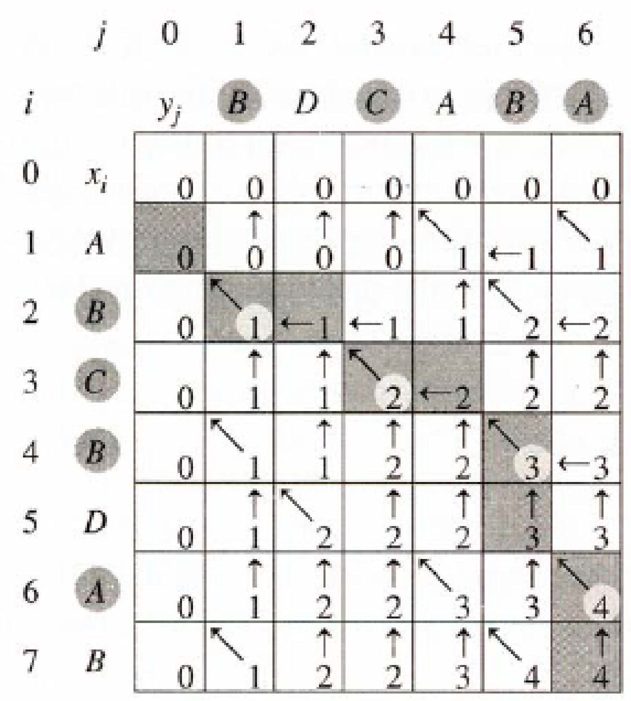
Longe

林劭原 老師

共同子字串

例如: $X = <A, B, C, B, D, A, B>$, $Y = <B, D, C, A, B, A>$

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1					
3	C	0	1					
4	B	0	1		↖	↑		
5	D	0	1		←	?		
6	A	0	1					
7	B	0	1					opt. sol.



$c[7,6]$ is the answer.

$c[i, j]$ is determined by $c[i-1, j-1]$, $c[i-1, j]$, and $c[i, j-1]]$.

↖左上 ↑上 ←左

We can use another table (say, table b) to store ↖, ↑, or ←.

A dynamic programming algorithm usually uses two tables.

First, fill in the initial values into the table. 先填初值

Then, fill in the remaining entries of the table. 再填其他

For LCS, the table can be filled in row-by-row or column-by-column.

林劭原 老師

- LCS-LENGTH(X,Y)
- m = X.length

```
n = Y.length
let b = [1..m,1..n] and c[0..m,0..n] be new tables
for i = 1 to m
        c[i,0] = 0
for j = 0 to n
        c[0,j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i,j] = c[i-1,j-1]+1
                        b[i,j] = "↖"
                elseif c[i-1,j]≥c[i,j-1]
                        c[i,j] = c[i-1,j]
                        b[i,j] = "↑"
                else c[i,j] = c[i,j-1]
                        b[i,j] = "←"
return c and b
```

```
Print-LCS(b,X,i,j)
if i == 0 or j == 0
        return
If b[i,j] == "↖"
        Print-LCS(b,X,i-1,j-1)
        print x_i
elseif b[i,j] == "↑"
        Print-LCS(b,X,i-1,j)
else Print-LCS(b,X,i,j-1)
```

Matrix-chain multiplication

- 在乎矩陣相乘時，「乘」用了幾次，不在乎「加、減」用了幾次，因為「乘」比較花時間。
- If A is a p*q matrix and B is a q*r matrix, then computing AB requires p*q*r multiplications.
- 例:$\mathrm{A}_1, \mathrm{A}_2, \mathrm{A}_3$ are 10*100,100*5, and 5*50 respectively.
  Want to compute $\mathrm{A}_1\mathrm{A}_2\mathrm{A}_3$.
  If use $\left((A_1A_2)A_3\right)$, then 10*100*5+10*5*50 = 7500 multiplications.
  If use $(A_1(A_2A_3))$, then 10*5*50+10*100*50 = 75000 multiplications.

Matrix-chain multiplication

- A product of matrices is fully parenthesized 矩陣相乘被完全加括號 if it is either a single matrix or the product of two fully parenthesized matrix products surrounded by parentheses.

Matrix-chain multiplication

- The matrix-chain multiplication problem 矩陣相乘問題 is:
- Given a chain <$A_1, A_2, \ldots, A_n$> of n matrices, where for $i = 1,2,\ldots,n$, matrix A_i has dimensions $p_{i-1} \times p_i$, fully parenthesize the product $A_1 A_2 \ldots A_n$ 完全被加了括號 in a way that minimizes the number of multiplications.
- A fully parenthesized product is optimal if it requires the minimum number of multiplications. 成的個數最少

林劭原 老師

Matrix-chain multiplication

Method 1. Brute force 暴力法

Exhaustively check all possible parenthesizations.

Analysis:

Let $P(n)$ denote the number of possible parenthesizations for n matrices.加括號的方法數

Then,

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \end{cases}.$$

M

Method 2. Dynamic programming

Consider $A_1A_2A_3A_4A_5A_6A_7$.

Suppose you know the top level split in an optimal parenthesization is $((A_1A_2A_3)(A_4A_5A_6A_7))$.

Just find the cost of an optimal parenthesization of $A_1A_2A_3 = r$.

Just find the cost of an optimal parenthesization of $A_4A_5A_6A_7 = s$.

Then the cost of an optimal parenthesization of $A_1A_2A_3A_4A_5A_6A_7 = r + s + p_0 \times p_3 \times p_7$.

More generally, if you have $A_iA_{i+1} \cdots A_j$ and someone tells you that

the top level split in an optimal parenthesization is between A_k and A_{k+1}.

Suppose $m[i, j] =$ cost of an optimal parenthesization of $A_iA_{i+1} \cdots A_j$.

Then $m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} \times p_k \times p_j$.

This problem also has optimal substructure property.

事實上沒有人會告訴我們 k 是多少，但我們可以去試所有可能的 k .

$$m[i,j]=\begin{cases}0 & \text{if } i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}\times p_k\times p_j\} & \text{if } i<j\end{cases}$$

The minimum number of multiplications for the product $A_1A_2\cdots A_n$ is therefore $m[1,n]$.

There may be more than one optimal solution.

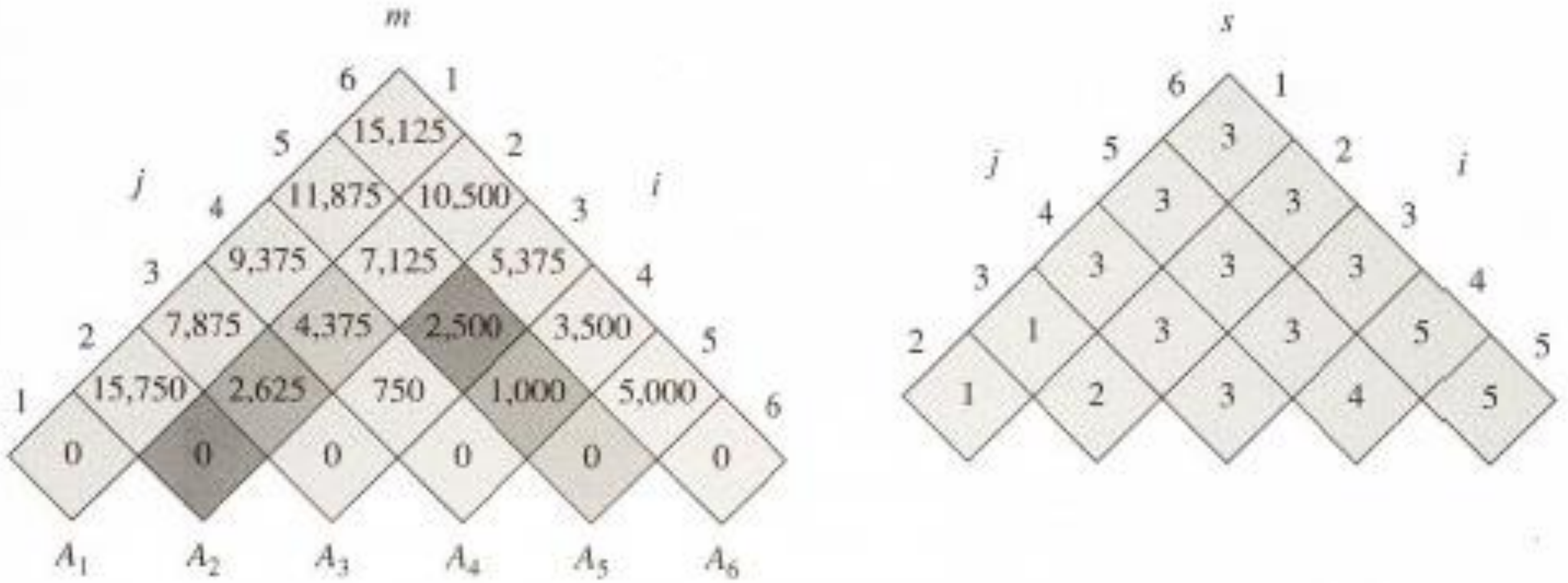
If we only want to know the minimum number of multiplications in an optimal solution, then table m is sufficient.

If we also want to know the way to multiple the matrices, then an extra table is required; suppose this extra table is called s .

Mat

A dynamic programming algorithm usually uses two tables.

例:	A_1	A_2	A_3	A_4	A_5	A_6
	30×35	35×15	15×5	5×10	10×20	20×25



$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \end{cases}$$
$$= 7125 .$$

tion

```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

以下為印出最佳的括弧給法的 pseudocode:

```
PRINT-OPTIMAL-PARENS(s, i, j)
if i == j
    print "A"_i
else print "("
    PRINT-OPTIMAL-PARENS(s, i, s[i, j])
    PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
    print ")"
```

This algorithm takes $O(n^3)$ time –
it needs to construct a table of size $\Theta(n^2)$ and
it takes $O(n)$ time to fill in an entry of the table.

若希望也能得出an optimal parenthesization的括弧給法,
則可以多使用a table s of size $\Theta(n^2)$ 來記錄每個k是多少.
得出an optimal parenthesization的括弧給法花 $O(n^2)$ time.
上例的 table s 可得: opt. paren. is $((A_1(A_2A_3))((A_4A_5)A_6))$.

HW5:高鐵搭乘問題

- 高鐵票分段買比較便宜? - UniMath (google.com) https://sites.google.com/a/g2.nctu.edu.tw/unimath/2017-01/HSR?fbclid=IwAR1zUbVWOfKJuOFkpSx6vBuJWJtTkojrd_QGuFeoeaKWMogt35hIURTvSh0
- 請用動態規劃做出文章中結論的表格。(100分)
- 郭君逸教授用了 Acyclic Dijkstra’s Algorithm ，我不確定單純用動態規劃可不可以，我直覺覺得可以，但我沒寫過，大家可以試試看，做不出來沒關係。