89

Algorithm Midterm (2016 Spring)

ID: B10315001 Name: 宋世民

$f(n) = \Theta(g(n))$

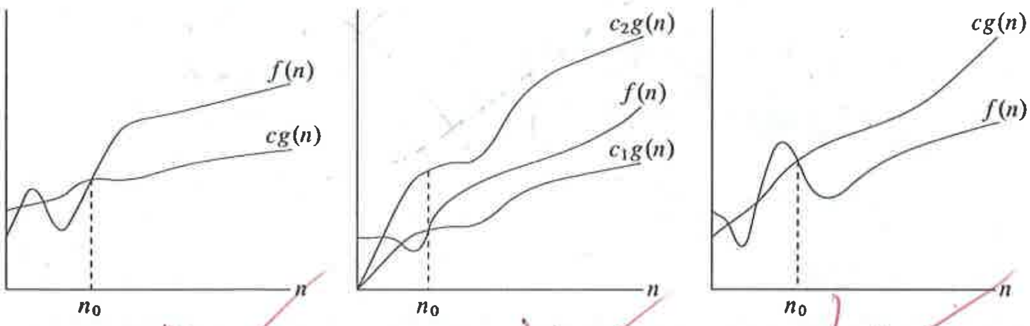
$\exists c_1 g(n) \le f(n) \le c_2 g(n)$

1. Please select the appropriate notation for the following figures: (6%)

(1) $f(n) = \Theta(g(n))$ (2) $g(n) = \Theta(f(n))$ (3) $f(n) = O(g(n))$ (4) $g(n) = O(f(n))$

(5) $f(n) = \Omega(g(n))$ (6) $g(n) = \Omega(f(n))$ (7) $f(n) = o(g(n))$ (8) $g(n) = o(f(n))$

(9) $f(n) = \omega(g(n))$ (10) $g(n) = \omega(f(n))$



(a) 5 (b) 1 (c) 3

$f(n) = O(g(n))$

$0 < f(n) \le cg(n)$

$f(n) = o(g(n))$

$0 < f(n) < cg(n)$

2. (d) For the following statements, which one is incorrect? (2%)

(a) $2n = O(n^2)$ (b) $2n^2 = O(n^2)$ (c) $2n = o(n^2)$ (d) $2n^2 = o(n^2)$

3. (a) What does "programming" mean in "dynamic programming"? (2%)

dynamic programming: 动態規劃

programming: tabular method，即作成表格以儲存資料

(b) What is the major difference between "divide-and-conquer" and "dynamic programming"? (3%)

divide-and-conquer:
1、Partition the problem into disjoint subproblems.
2. solve the subproblems recursively
3. Combine them ~~to~~ solve the original problem.

dynamic programming:
1、Apply when the subproblems overlap.
2. Solve each subproblem just once and save them in a table.

Major difference: when subproblems overlap, dynamic programming is better

(c) What is the major difference between "dynamic programming" and "greedy algorithm"? (3%)

dynamic programming:
1、Solve the subproblems before making first choice
2. Typically in bottom-up manner (or memoization + top-down)

greedy:
1、Make 1st decision before solve subproblems.
2. In a top-down manner.

4. Recall in Chapter 4, we talk about the problem of buying one unit of stock one time and then selling it at a later date to maximize your profit. Please explain the basic idea of how to using divide-and-conquer to solve the problem. (6%)

−2

low … high

-1	1	3	-2	1	5	-6	2

mid | mid+1

將一陣列切割，分成左、中、右三部分

左：由 mid → low 去找最大子序列

右：由 mid+1 → high 去找最大子序列

中：由包含中間部分的陣列去找最大子序列

如此遞迴，最後合併各子結果，

即可得最佳解。

Algorithmi

```
FIND-MAX-SUBARRAY
left-sum = -∞
sum = 0
for i = mid down to low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = -∞
sum = 0
for j = mid+1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, max-left + max-right)
```

5. Given a chain <A_1, A_2, ..., A_n> of n matrices, where for i = 1, 2, ..., n, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $A_1 A_2 \dots A_n$ in a way that minimizes the number of scalar multiplications.

(a) What does "fully parenthesize" mean? (3%)

−1

parenthesize all the matrices to make multiplication order,

ex. $A_1A_2A_3A_4A_5A_6 \xrightarrow{\text{F.P.}} ((A_1(A_2A_3))(A_4(A_5A_6)))$

(b) Suppose that to optimally parenthesize $A_i A_{i+1} \dots A_j$, we split the product between A_k and A_{k+1}. Let $m[i, j]$ be the minimum number of scalar multiplications needed to compute the matrix $A_{i..j}$; for the full problem, the lowest-cost way to compute $A_{1..n}$ would thus be $m[1, n]$. Please define $m[i, j]$. (3%)

$$m[i,j] = \begin{cases} 0 & , \text{if } i=j \\ \min\limits_{i\le k<j} \{ m[i,k] + m[k+1,j] + P_{i-1}P_kP_j \} & , \text{if } i \ne j \end{cases}$$

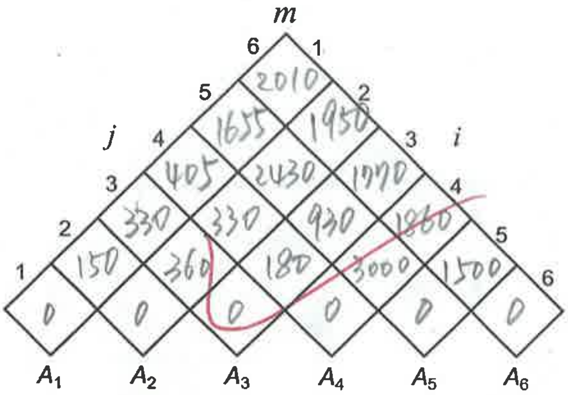
(c) Let $s[i, j]$ be a value of k at which we split the product $A_i A_{i+1} \dots A_j$ in an optimal parenthesization. Please define $s[i, j]$. (3%)

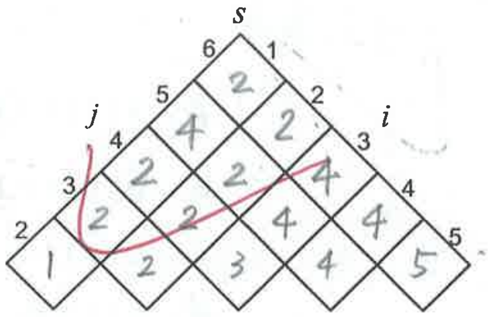
$s[i,j] = k$, if $m[i,k] + m[k+1,j] + P_{i-1}P_kP_j$ 有 minimum.

A_1	A_2	A_3	A_4	A_5	A_6
5×10	10×3	3×12	12×5	5×50	50×6

$P_0\ P_1\ P_2\ P_3\ P_4\ P_5\ P_6$

(d) Suppose the sequence of dimensions $p = \langle p_0, p_1, \ldots, p_n\rangle$ is <5, 10, 3, 12, 5, 50, 6>. Please complete the following tables. (12%)





(e) The initial call `PRINT-OPTIMAL-PARENS(s,1,n)` prints an optimal parenthesization of $\langle A_1, A_2, \ldots, A_n\rangle$. Please complete the following function. (4%)

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 ")"
```

(s, 1, 6)

([1,2] [3,6])

(([1,1][2,2])([3,4][5,6]))

((A1A2)((A3A4)(A5A6)))

(f) Following (d) and (e), what would be printed when we call `PRINT-OPTIMAL-PARENS(s,1,6)`? (3%)

((A1A2)((A3A4)(A5A6)))#

6. (a) Please define "prefix codes". (3%)

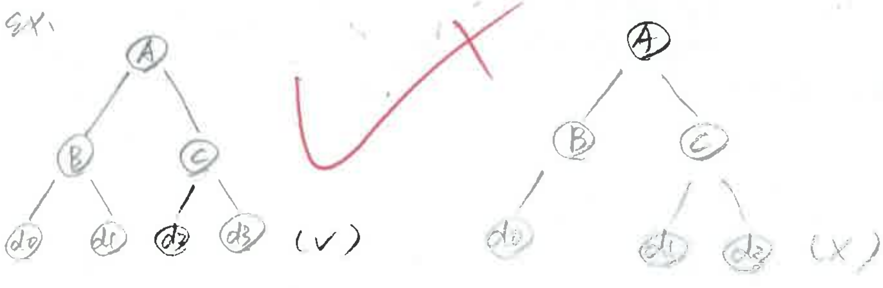
-3

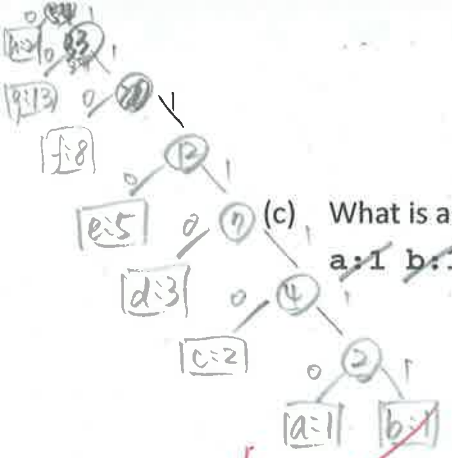
縮寫碼，~~即將字元~~表成 0 1 序列

(b) Please define "full binary tree". (3%)

All nodes except leaf nodes have children.

Ex:





(c) What is an optimal Huffman code for the following set of frequencies? (6%)

`a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21`

a: 1111110 g: 10

b: 1111111 h: 0

c: 111110 #

d: 11110

e: 1110

f: 110

7. Determine the cost and structure of an optimal binary search tree for a set of $n = 7$ keys with the following probabilities:

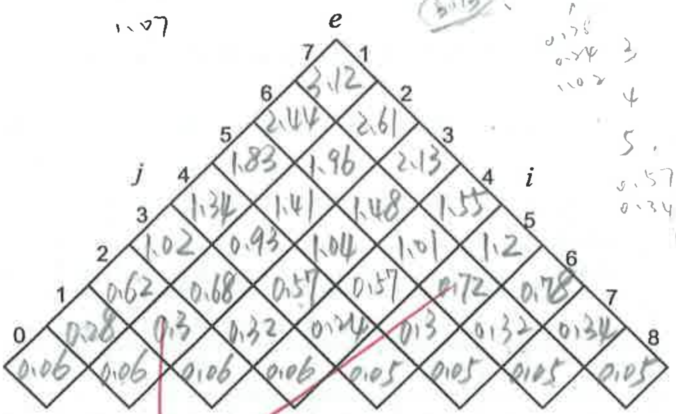
i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.02	0.10	0.12	0.14
q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

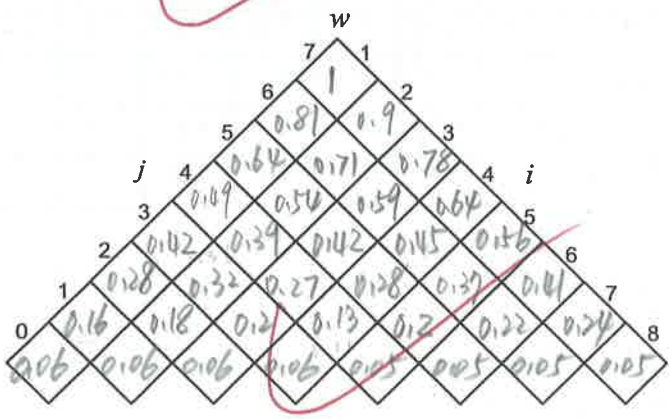
(a) The following pseudocode has some bugs. Please find them out. (8%)

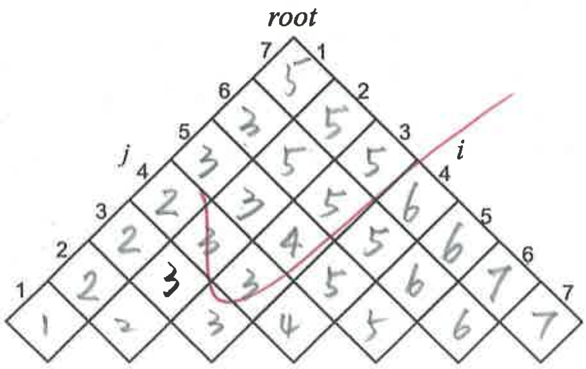
−4

```
OPTIMAL-BST(p, q, n)
let e[1..n+1, 0..n], w[1..n+1, 0..n],
        and root[1..n, 1..n] be new tables
for i = 1 to n+1
    e[i, i-1] = q_{i-1}
    w[i, i-1] = q_{i-1}
for l = 1 to n
    for i = 1 to n - l  -> n-l+1
        j = i + l
        e[i, j] = ∞
        w[i, j] = w[i, j-1] + p_i + q_j
        for r = i to j
            t = e[i, r-1] + e[r-1, j] + w[i, j]        e[r+1, j]
            if t < e[i, j]
                e[i, j] = t
                root[i, j] = r
return e and root
```

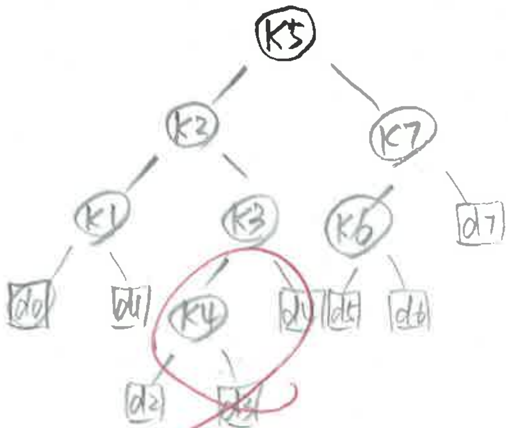
(b) Please complete the following tables. (16%)

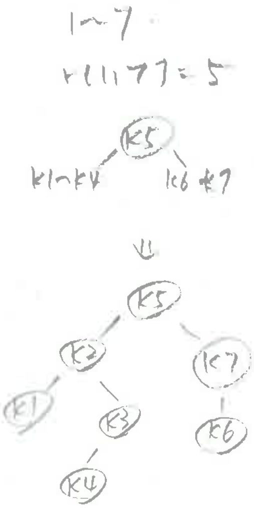






(c) What is the resulting optimal binary search tree? (4%)





8. Following the rules we discussed in class, please:

(a) Complete the following table. (8%)

(b) Determine an LCS of <1, 0, 0, 1, 0, 1, 0, 1> and <0, 1, 0, 1, 1, 0, 1, 1, 0>. (2%)

(a)

				✓	✓		✓	✓	✓	✓	
	j	0	1	2	3	4	5	6	7	8	9
i	y_j		0	1	0	1	1	0	1	1	0
0	x_i	0	0	0	0	0	0	0	0	0	0
✓ 1	1	0	0 ↑	1 ↖	1 ←	1 ↖	1 ↖	1 ←	1 ↖	1 ↖	1 ←
✓ 2	0	0	1 ↖	1 ↑	2 ↖	2 ←	2 ←	2 ↖	2 ←	2 ←	2 ↖
3	0	0	1 ↖	1 ↑	2 ↖	2 ↑	2 ↑	3 ↖	3 ←	3 ←	3 ↖
✓ 4	1	0	1 ↑	2 ↖	2 ↑	3 ↖	3 ↖	3 ↑	4 ↖	4 ↖	4 ←
✓ 5	0	0	1 ↖	2 ↑	3 ↖	3 ↑	3 ↑	4 ↖	4 ↑	4 ↑	5 ↖
✓ 6	1	0	1 ↑	2 ↖	3 ↑	4 ↖	4 ↖	4 ↑	5 ↖	5 ↖	5 ↑
7	0	0	1 ↖	2 ↑	3 ↖	4 ↑	4 ↑	5 ↖	5 ↑	5 ↑	6 ↖
✓ 8	1	0	1 ↑	2 ↖	3 ↑	4 ↖	5 ↖	5 ↑	6 ↖	6 ↖	6 ↑

(b)

LCS : 101011

if $x_i = y_i$
    $c[i, j] = c[i-1, j-1] + 1$
    $b[i, j] =$ "↖"

else if $c[i-1, j] \geq c[i, j-1]$
    $c[i, j] = c[i-1, j]$
    $b[i, j] =$ "↑"

else $c[i, j] = c[i, j-1]$
    $b[i, j] =$ "←"