ADA hw1

B08902065 資工二 洪易

特別感謝B08902127 林歆凱, B08902072 黃國銘, B08202029 楊欣翰
B08902063 陳羿穎 提供我寫這份作業時的幫助與靈感。

5. Time Complexity & Recurrence

(1)

(a)

(1)(a) disprove. 反證法.

設 $\sqrt{n} = O(n^{\sin(n)})$ $\Rightarrow$ $n^{\frac{1}{2}} = O(n^{\sin(n)})$ 即 $n^{\frac{1}{2}} \le c \cdot n^{\sin(n)}$

即 當 $n > n_0$ 時. 所有 $\sin(n)$ 均 $\ge \frac{1}{2}$. 但 $-1 \le \sin(n) \le 1$, 故必存在 $n > n_0$ 且 $\sin(n) < \frac{1}{2}$. 故矛盾 $\Rightarrow$ $\sqrt{n} \ne O(n^{\sin(n)})$. □

(b)

(b) prove.

已知 $\Theta(g(n)) = f(n)$ 故 $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$.

$\because$ $f(n) = \Omega(g(n))$. 可取得 $c_1 > 0$, 及 n_0 使得 $\forall n \ge n_0$, $f(n) \ge c_1 \cdot g(n)$

同取 log. 故 $\log(f(n)) \ge \log c_1 + \log(g(n))$ $\Rightarrow \log f(n) = \Omega(\log(g(n)))$.

又 $\because$ $f(n) = O(g(n))$, 可取得 c_2, n_0, 使得 $\forall n \ge n_0$, $f(n) \le c_2 \cdot g(n)$.

同取 log $\Rightarrow \log(f(n)) \le \log(c_2) + \log g(n))$ $\Rightarrow \log(f(n)) = O(\log(g(n)))$

故. $\log(f(n)) = O(\log(g(n)))$, $\log(f(n)) = \Omega(\log(g(n)))$ $\therefore$ $\log(f(n)) = \Theta(\log(g(n)))$ □

(c)(d)

(c) disprove.

取 $f_1(n) = n^n$, $f_2(n) = 2n$, $g_1(n) = n^n$, $g_2(n) = n$

$\Rightarrow f_1(n) = O(g_1(n))$ 且 $f_2(n) = O(g_2(n))$

但 $f_1 \circ f_2(n) = 2n^{2n} = (4n^2)^n$, $g_1 \circ g_2(n) = n^n$

$\Rightarrow f_1 \circ f_2(n) = (4n^2)^n \neq O(n^n) = O(g_1 \circ g_2(n))$

$\forall\, n > n_0$ 時，找不出任何 c 使得 $(4n^2)^n \leq c \cdot n^n$. □.

(d) prove.

先證 $(n+a)^b = O(n^b)$，取 $c_1 = \max(a^b, 2^b)$，$n_0 = \max(a, 2)$ 當 $n \geq n_0 = a$ 時

若 $a > 2$，則 $(n+a)^b < a^b \cdot n^b = (an)^b$，若 $a \leq 2$，則

$(n+a)^b \leq (n+2)^b < (2n)^b$（在 $n > 2$ 時恆成立）故 $(n+a)^b = O(n^b)$

再證 $(n+a)^b = \Omega(n^b)$，取若 $a > 0$，$c_2 = 1$，$n_0 = 0$ 若 $a \leq 0$，$c_2 = -2^b$，$n_0 = -2a$

若 $a > 0$，則 $\forall\, n \geq 0$，$(n+a)^b > n^b$ $(a > 0)$ 成立.

若 $a \leq 0$，則 $\forall\, n \geq -2a$，$(n+a)^b > 2^{-b} \cdot n^b = (\frac{n}{2})^b$

$(n > -2a)$ 亦成立.

故 $(n+a)^b = \Omega(n^b)$

故 $(n+a)^b = O(n^b) = \Omega(n^b) = \Theta(n^b)$ □.

(2)

(a)

Q5

(2)(a) $T(n)=\Theta\left(\frac{n}{\log(n)}\right)$. 皆以 $\log_2$ 為 log 作計算 (constant 不影響).

pf: 取一足夠大 k，使得 $n-127k \le 2 \Rightarrow k \ge \frac{n-2}{127}$ …①

Recursion-tree method.

$$T(n) = T(n-127) + \frac{127}{\log(n)} = T(n-127\times 2) + \frac{127}{\log(n-127)} + \frac{127}{\log(n)}$$

$$\cdots = T(n-127\times k) + 127\times\sum_{i=0}^{k-1}\frac{1}{\log(n-127\times i)}$$

$$= 1 + 127\sum_{i=0}^{k-1}\frac{1}{\log(n-127i)} \ge 1 + 127k\cdot\frac{1}{\log(n)}$$

$$\ge 1 + 127\cdot\left(\frac{n-2}{127}\right)\cdot\frac{1}{\log(n)} = 1+\frac{n-2}{\log(n)} = \Omega\left(\frac{n}{\log(n)}\right)$$

$$T(n) = 1 + 127\sum_{i=0}^{k-1}\frac{1}{\log(n-127i)} \le 1 + 127\int_{2}^{n}\frac{dx}{\log(x)}$$

$$= 1 + 127 \left.\ln(2)\, li(x)\right|_{2}^{n} = \underbrace{127\ln(2)}_{\text{constant}}\, li(n) + \underbrace{1-\ln(2)\, li(2)}_{\text{costant}}$$

又 $li(n) = O\left(\frac{n}{\log(n)}\right)$ 故 $T(n) = O\left(\frac{n}{\log(n)}\right)$.

Ref: 維基百科上有證明.

故 $T(n) = \Omega\left(\frac{n}{\log(n)}\right) = O\left(\frac{n}{\log(n)}\right) = \Theta\left(\frac{n}{\log(n)}\right)$ #.

(b)

(b) $T(n) = \Theta(n \log(n))$.

$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n\log(n) > n\log(n) = \Omega(n\log(n))$

故 $T(n) = \Omega(n\log(n))$.

再若. 設 $T(n) \le a \cdot n\log(n) + bn$.

if $n \le 2$, trivial.

if $n > 2$, $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n\log n$.

$$= a\frac{n}{2}\log(\frac{n}{2}) + b\cdot\frac{n}{2} + a\frac{n}{4}\log(\frac{n}{4}) + b\cdot\frac{n}{4} + a\frac{n}{8}\log(\frac{n}{8}) + b\cdot\frac{n}{8} + n\log n.$$

$$= (\frac{7}{8}a + 1)\, n\log(n) + (\frac{7}{8}b - \frac{11}{8}a\cdot\log(2))\, n.$$

要使 $a\log n \cdot n + bn$ 大於 $T(n)$ 可取 $a = 16$, $b = -200\log(2)$

=> 此時 $T(n) = O(n\log n) + O(n) = O(n\log(n))$

=> $T(n) = O(n\log(n)) = \Omega(n\log(n) = \Theta(n\log(n))$. □.

(c)(d)

(c) $T(n) = \Theta(n^2)$

Master Theorem => $f(n) = n\log(n) = O(n^{\log_2 4 - \epsilon}) = O(n^{2-\epsilon})$.

可取 $\epsilon = 0.5$, 使得 $n^{2-\epsilon} > n\log(n)$. (在 $n \to \infty$)

=> $\lim_{n\to\infty} \frac{n^{1.5}}{n\log(n)} = \lim_{n\to\infty} \frac{\sqrt{n}}{\log(n)} \overset{l'hospital}{=} \lim_{n\to\infty} \frac{x}{2\sqrt{x}} = \infty$,

$\because \epsilon > 0 \therefore T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$ □.

(d) $T(n) = \Theta(n\cdot\log(\log(n)))$

Let $n = e^k$ 代入, $T(e^k) = e^{\frac{k}{2}}\cdot T(e^{\frac{k}{2}}) + e^k$ => $\frac{T(e^k)}{e^k} = \frac{T(e^{\frac{k}{2}})}{e^{\frac{k}{2}}} + 1$

=> Let $U(k) = \frac{T(e^k)}{e^k}$. 則 $U(k) = U(\frac{k}{2}) + 1$ => master theorm

$f(k) = \Theta(k^{\log_2 1}) = \Theta(1)$ => $U(k) = \Theta(n^{\log_2 1}\cdot\log k) = \Theta(\log k)$, 故代回 $T(n)$

=> $T(n) = e^k\cdot\Theta(\log k) = \Theta(e^k\log k) = \Theta(n\cdot\log(\log(n)))$

6. Viennese Waltz

(1) 使用 Dynamic Programming 的作法

步驟一：

建立一個 n * n 的表 sum[n][n]去計算所有位置以整個圖中最左上角為左上點，該位置為右下點所構成的實心矩形的總和，分為四種狀況進行討論。

一是最左上點，答案也就是第(0, 0)的值，所費時間為 constant。

二是最左排所有點，答案皆為其上方的位置之值加上自身格子的值，所費時間亦為 constant。

三是最上列所有點，答案皆為其左方的位置之值加上自身格子的值，所費時間亦為 constant。

四是其他所有點，答案為其左邊位置之值加上上方位置之值在減掉左上格位置的值，故所費時間亦為 constant。

跑過所有位置，且在每個位置所花之時間均為 constant，故此步驟之時間複雜度為 $0(n^2)$ 。

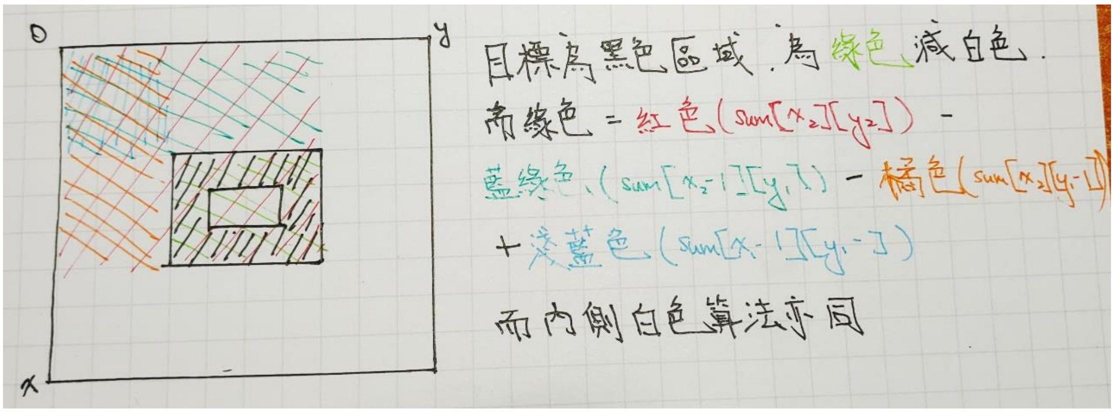
步驟二：

計算每個要求的矩形的值，設輸入值的左上點與右下點分別為(x_1, y_1)及(x_2, y_2)。可知每個要求矩形的 weight 為外圍矩形值減掉$(x_1 + 1, y_1 + 1)$與$(x_2 - 1, y_2 - 1)$構成的小矩形，分為兩種狀況考慮。

一是 $x_2 - x_1 <= 1$ 或 $y_2 - y_1 <= 1$，則此時矩形為實心，可直接取步驟一所建的表中，$sum[x_2][y_2] - sum[x_1][y_2] - sum[x_2][y_1] + sum[x_1][y_1]$，即為正解，其所費時間為 constant。

二則是其他情況，需考慮兩個矩形，使用外圍矩形去減掉裡面的小矩形，其算法為大矩形：$(sum[x_2][y_2] - sum[x_1][y_2 - 1] - sum[x_2 - 1][y_1] + sum[x_1 - 1][y_1 - 1])$ - 小矩形 $(sum[x_2 - 1][y_2 - 1] - sum[x_1][y_2 - 1] - sum[x_2 - 1][y_1] + sum[x_1][y_1])$，所費時間亦為 constant。

故此步驟走過每個要求矩形，而每個矩形所花時間是 constant，故此步驟所費時間為 0(K)



總和兩步驟之總和，所花時間為 $0(n^2) + 0(k) = 0(n^2 + k)$

(2)

步驟一：

承(1)中的步驟一建表 sum[n][n]，所花時間為 $O(n^2)$

步驟二：

以所有位置分別作為左上和右下，進行 brute force 搜尋並利用(1)中的做法去計算每個矩形的 weight，對於所有可能矩形均需花 constant 的時間去計算 weight。並維持一個 max 去紀錄跑過的組合中 perimeter 小於等於 L 且 weight 最大的矩形，最後求得目標矩形。此步驟需跑過所有位置為左上時，右下的所有可能點，也就是需要花費 $O(n^2 * n^2) = O(n^4)$ 的時間。

總和兩步驟共需花費 $O(n^4) + O(n^2) = O(n^4)$ 的時間

(3)

步驟一：

建立 row_sum[n][n]及 col_sum[n][n]分別記錄每個位置以左同列及以上同行的總和，而建表方式為類似 DP，row_sum[x][y] = row_sum[x - 1][y]，col_sum[x][y] = row_sum[x][y - 1]，故所需花費 $O(n^2)$的時間。

步驟二：

設立兩個 col_pointer，i 及 j，分別代表矩形的左邊與右邊行的位置，而 j 從 1 跑到 n - 1，i 則是從 0 跑到 j - 1，故此雙層迴圈需花費 $O(n^2)$的時間。利用 row_sum 的 dp 表去建立一個新的 dp 表：q 來實作一個單調隊列(monotonic queue)。其中 q 會紀錄在(i, j)範圍內每列的 weight 以及是第幾個 row。在一開始設隊列的頭位置為 head = 0，隊列尾的位置為 tail = 1 以及一個 current 從 1 跑到 N 去尋找最佳解。而對於每個 current，進行以下操作：

利用步驟一中的 DP 表以 O(1)時間計算當時由(i, j)為左右行，(head, current)為上下列所構成的矩形 weight。若大於 max_weight 則取代掉。

每次進入迴圈時，比較 cur_weight = row_sum[current][j + 1] - row_sum[current][i] 與 tail_weight = q[tail].weight + (col_sum[current + 1][i] - col_sum[tail][i]) + (col_sum[current + 1][j] - col_sum[tail][j])。若 tail_weight < cur_weight，則將 tail -= 1，並持續計算到 tail_weight > cur_weight 或 tail < head 為止。此時將 current 及 cur_weight 放入單調隊列的尾端並將 tail + 1。

在此步驟中，i, j, current 均須從 0 走到 n - 1，因此時間複雜度為 $O(n^3)$。

總和兩步驟，共需花費 $O(n^2) + O(n^3) = O(n^3)$ 的時間。

(4)

步驟一：

與(3)中的步驟一相同，建立兩個 DP 表，所費時間為 $O(n^2)$

步驟二：

設立兩個 col_pointer，i 及 j，分別代表矩形的左邊與右邊行的位置，而 j 從 1 跑到 n – 1，i 則是從 0 跑到 j – 1，故此雙層迴圈需花費 $O(n^2)$的時間。利用 row_sum 的 dp 表去建立一個新的 dp 表：q 來實作一個單調隊列(monotonic queue)。其中 q 會紀錄在(i, j)範圍內每列的 weight 以及是第幾個 row。在一開始設隊列的頭位置為 head = 0，隊列尾的位置為 tail = 1 以及一個 current 從 1 跑到 N 去尋找最佳解。而對於每個 current，進行以下操作：

當 current – head > (L – 2 * (j - i)) / 2 時，head += 1，並利用步驟一中的 DP 表以 O(1)時間計算當時由(i, j)為左右行，(head, current)為上下列所構成的矩形 weight。若大於 max_weight 則取代掉。

每次進入迴圈時，比較 cur_weight = row_sum[current][j + 1] - row_sum[current][i] 與 tail_weight = q[tail].weight + (col_sum[current + 1][i] - col_sum[tail][i]) + (col_sum[current + 1][j] - col_sum[tail][j])。若 tail_weight < cur_weight，則將 tail -= 1，並持續計算到 tail_weight > cur_weight 或 tail < head 為止。此時將 current 及 cur_weight 放入單調隊列的尾端並將 tail + 1。

在此步驟中，i, j, current 均須從 0 走到 n – 1，因此時間複雜度為 $O(n^3)$。

總和兩步驟，共需花費 $O(n^2) + O(n^3) = O(n^3)$ 的時間。

<Prove of correctness>

此題目標是找到最大 perimeter 的矩形，因此確定可以找到最大值的左右行與上下列便可證明找到的答案必定是對的。

左右行：

由於跑過所有行的組合(i, j)，故必定可以經過最大值所存在的行組合。

上下列：

因為在單調隊列的計算方式會使其選出的 head 永遠是最大的，因此上列一定會是最大的，下列的部分，則因為 current 會跑過所有列，因此也必定會經過最大的。

因此此演算法不會錯過任何有可能是最大的組合，且在遇到最大組合時會儲存下來，故能找到最佳解。