Problem 24-2

(a)

設 box X nests within box Y，則根據定義，存在一排列 π

$x_{\pi(1)} < y_1, x_{\pi(2)} < y_2, \ldots, x_{\pi(d)} < y_d$。

設 box Y nests within box Z，則根據定義，存在一排列 π'

$y_{\pi'(1)} < z_1, y_{\pi'(2)} < z_2, \ldots, y_{\pi'(d)} < z_d$。

因爲排列 π 是一對一的對應，故必存在另一排列 π''

$x_{\pi''(1)} < y_{\pi'(1)}, x_{\pi''(2)} < y_{\pi'(2)}, \ldots, x_{\pi''(d)} < y_{\pi'(d)}$。

則

$x_{\pi''(1)} < y_{\pi'(1)} < z_1$

$x_{\pi''(2)} < y_{\pi'(2)} < z_2$

.

.

$x_{\pi''(d)} < y_{\pi'(d)} < z_d$。

所以 box X nests within box Z，nesting relation 爲 transitive。

(b)

若有兩個 box A, B，要判斷是否 A nests within B，則：

Step 1. 將 A, B 的 dimension value 做 sort，由小到大，設 sort 好的 A 爲 S_A，sort 好的 B 爲 S_B。

Step 2. 若 $S_A[i] < S_B[i]$, for all $1 \le i \le d$，則 A nests within B，若不是，則 A can not nests within B。

Proof：

sort 的動作並不影響結果，A nests within B if and only if S_A nests within S_B。

若都是 S_A 的值比較小，則我們已經找到一組符合定義的 permutation。

若不是，我們令 p 爲 $S_A[i] \ge S_B[i]$中最小的 index

則 $S_A[1] < S_B[1], \ldots, S_A[p-1] < S_B[p-1], S_A[p] \ge S_B[p], \ldots$

S_A 中小於 $S_B[p]$的值只有 p-1 個，而 S_B 中小於等於 $S_B[p]$的值有 p 個，所以 S_A 中不可能找到 p 個值可以各別小於 $S_B[1],\ldots,S_B[p]$，換句話說，就是不存在符合定義的 permutation。

Time : $O(d \log d) + O(d) = O(d \log d)$

(c)

Step 1. 先對這 n 個 box 的 dimension value 全部都各別 sort 一次。

Step 2. 兩兩比較 nesting 的關係，並作成一個 weighted directed graph，每個 box 爲 graph 中的一個 vertex，若 box i nests within box j，則 $w(i, j) = -1$。

Step 3. 新增一個起點並指到全部的 vertices，這些 edges 的 weight 全爲 0。

Step 4. 由新增的起點開始做 DAG-SHORTEST-PATHS（課本 592 頁），則找出來的 path 就是最長的 nesting relation sequence。

這個 algorithm 的正確性是很明顯的，因爲 nesting 的關係不會形成 cycle，故可以套用 DAG-SHORTEST-PATHS algorithm，而找出來的 path 會是圖中包含最多 edge 的 path，path 中每一條 weight 爲-1 的 edge 就是一個 nests within 的關係。

Time：

Step 1. $O(n) * O(d \log d) = O(nd \log d)$

Step 2. $O(n^2) * O(d) = O(dn^2)$

Step 3. $O(n)$

Step 4. $O(n + n^2) = O(n^2)$

Total = $O(nd \log d + dn^2)$

Exec 25.2-4

整個程式只改了一行，原本是分成 n 個 n^2 空間的 Array 來儲存，當要作第 k 次的時候會從第 k-1 次的 Array 拿取 d_{ij} 和 $d_{ik}+d_{kj}$ 的值來比較大小並將比較大值的存入這第 k 個 Array，現在改過後的新程式變成每次都用同一個 Array 放資料，原本拿上一個 Array 的 d_{ij} 和 $d_{ik}+d_{kj}$ 值來做比較，改成直接就拿現在這個 Array 內的 d_{ij} 和 $d_{ik}+d_{kj}$ 來比較，這樣會不會因爲拿到的值被修改過而發生問題呢?

我們可以很容易就知道，在做這第 k 次的時候，還沒被更改到的部份一定是第 k-1 次的資訊，因爲用同一個 Array 存，從 code 我們可以看出每個 Array 中的 d_{ij} 都只會被改一次，所以在更改 d_{ij} 的瞬間，我們可以確定 Array 中的 d_{ij} 值一定還是和 k-1 次的狀態一樣，因此問題只剩下後面的 d_{ik} 以及 d_{kj} 這兩個值會不會因爲在算到 d_{ij} 這格之前就被修改過而不是 k-1 次的 d_{ik} 以及 d_{kj} 值，如果能確定值依然還是 k-1 次的值，那這個改過的 code，即使每次都用同個 Array 存，依然會有正確的結果

稍微觀察我們不難發現，不管 d_{ik} 或是 d_{kj} 在做 d_{ij} 之前有沒有被跑過都沒關係，因爲就算被跑過值也不會改變，可由下面三式表示：

$i \neq k, j = k$ $d_{ik} = \min(d_{ik}, d_{ik}+d_{kk}) = d_{ik}$

$i = k, j \neq k$ $d_{kj} = \min(d_{kj}, d_{kk}+d_{kj}) = d_{kj}$

$i = j = k$ $d_{kk} = \min(d_{kk}, d_{kk}+d_{kk}) = d_{kk}$

(因爲 $d_{kk} \geq 0$ 不然有自己到自己有負的 cycle 問題本來就不能作)

從上面可知當 d_{ij} 其中有至少一邊和 k 有關的時候，跑過值也不會變，因此共用同個空間存並不會發生問題，第 k 次運算需要用到的 d_{ik} 或是 d_{kj} 值依然會是第 k-1 次時留下來的值，不管在第 k 次那格是否已經被算過

因此得知修改過後的 code 只花 $O(n^2)$的空間依然可以得到正確的結果