# 104 年公務人員高等考試三級考試試題

類 科：資訊處理
科 目：資料結構

一、有位程式設計師在撰寫程式時遇到了一個難解的問題，後來發現有兩個演算法可以解這個難題：演算法A的時間複雜度為$O(n^2\log(n!))$，演算法B的時間複雜度為$O(n^2((\log n)!))$。假設輸入資料的個數$n$通常都很大，他應該選擇那個演算法比較好，原因何在？

【擬答】：

演算法 A 為 $O(n^2\log(n!))$
log(n!) = log (1*2*3…*n) = log 1+log 2+log 3+…+log n-----------(1)

演算法 B 為 $O(n^2((\log n)!))$
(log n)!=1*2*3*…*log n------------(2)

在 n 很大之時，比較(1)(2)兩式，(2)的成長幅度比起(1)來的大上許多，同時乘上正數 $n^2$ 之後，(2)的成長幅度還是比起(1)來的大，因此選擇演算法 A 比較理想。

二、樹（tree）是一個很常用的資料結構。一個樹是指一個沒有迴圈（cycle）的聯通圖（connected graph）。

(一)證明：每個具有 n 個節點（node）的樹，n＞1，至少有 2 個分支度（degree）為 1 的節點。（分支度就是指有多少邊以此節點為端點。）
(二)用前項結果證明：每個具有 n 個節點的樹，n＞1，恰好有 n－1 個邊（edge）。

【擬答】：

(一)就題意而言，一個樹是指一個沒有迴圈（cycle）的聯通圖（connected graph）。沒有迴圈，代表任兩個節點皆只有一條路徑可到達，在 n>1 的狀況下，就是至少會有兩個節點的分支度為 1。

(二)因為聯通，所以 n 節點(V)的邊(E)可以得到 |E|>= |V| -1 -----(1)
以數學歸納法來證明|E|<= |V| -1
當 n >1 時，由(一)可得到 E < = 2 -1 = 1 。
如果 n<= k 時，假設 |E|<= |V| -1 成立
當 n = k+1 時，如果取消一條邊，則會分解為兩個圖型，分別有 n1 與 n2 的頂點，而且 n1 與 n2 都不會大於 k，因此總邊數會等於 (n1-1+n2-1)+1 =n1+n2-1=n-1，也就是 |E|<= |V| -1 成立----(2)
結合 (1)(2)，可以得到|E|= |V| -1

三、給定一個權重圖（weighted graph），$G=(V, E, w)$，其中每個邊（edge）$e$的權重$w(e)$都是正整數，為了簡單，假設$V=\{1, 2, ..., n\}$。任意點$v$與起始點$s$的距離可以用一個矩陣$d[1..n]$來表示。

(一)設計一個只需 $O(n)$空間的方法來記錄從 $s$ 出發，到達每個點的最短路徑。
(二)說明計算與印出從起始點 $s$ 到任意點 $t\in V$ 的最短路徑的演算法。（解此小題時可參考 Dijkstra 或其他演算法來設計，且不須將 Dijkstra 或別的演算法做詳細的描述。）

【擬答】：

(一)先利用一個一維陣列 d_rev，在計算有向圖中的所有最短路徑時，記錄每一次計算時，所使用的前一節點，最後再將此陣列反向輸出至 d 即可獲得。

(二)利用 Dijstra 演算法即可
假設 S 是那些最短路徑已被找到的頂點之集合，dist[ ]為 n 個位置的陣列，用來儲存某一

頂點到其他頂點的最短距離，dist[u]表示從頂點 v 到 u 的最短路徑，cost[i][j]表示邊 <i, j>的長度

*1.*如果下一條最短路徑是到 u 點的，那這條路徑開始在 v 點，結束在 u 點，且經過的點全在 S 裡。

*2.*下一條路徑的目的 u 必須是所有不在 S 中的頂點裡，且有最小長度的最短路徑 dist[u]。

*3.*在步驟 2 中選到的點 u 變成 S 的一部份，且從選擇過程中可以得到 v 到 u 的最短路徑。在 u 加入 S 之後，任何一條從頂點 v 開始，只經過在 S 中的頂點，並且結束於不在 S 的頂點 w 的最短路徑可能會變短，換言之，dist(w) =min{dist(w), dist(u)+cost(u, w)}

四、有個矩陣$A[1..n]$，$n$的值很大。在矩陣$A$中存有$n$個正整數，且從小到大排列。給定某個整數$x$，二分搜尋法（binary search）可以在$O(\log n)$的時間內找出$x$在矩陣$A[1..n]$的位置，或宣告在$A[1..n]$中沒有x。在某個應用中，已知絕大部分的$x$都會出現在矩陣$a[1..n]$的前面$m$個元素，且$m$的值遠小於$n$，但是無法預知$m$的範圍。設計一個演算法，可以在$O(\log m)$的時間內完成搜尋。

【擬答】：

先找到 m 的範圍，將搜尋範圍縮小至 m 後，再使用 binary search 即可獲得 O(log m)的時間。演算法參考如下：

```
m:=1;
while (m < n and A[m] < x)
m=m*2;
first = 1;
last = m;
middle = (first+last)/2;
while (first <= last) {
  if (A[middle] < x)
     first = middle + 1;
  else if (A[middle] == x) {
     printf("%d found at location %d.\n", search, middle+1);
     break;
  }
  else
     last = middle - 1;
     middle = (first + last)/2;
}
if (first > last)
  printf("Not found! %d is not present in the list.\n", search);
```

五、假設有個矩陣 $A[1..n]$儲存$n$個整數。Quick sort是一個排序演算法。假設有個副程式partition（$A, l, r$）其輸入參數$A$是一個矩陣，$l, r, l<r<n$，是兩個指標。其回傳的值$m$也是一個指標。這個副程式可將矩陣中從$l$到$r$的這一段資料 $A[l..r]$區分成兩段： $A[l..m]$和 $A[m+1..r]$，使得在 $A[l..m]$中的元素都小於或等於$x$，而在 $A[m+1..r]$中的元素都大於或等於$x$，其中$x$是從 $A[l..r]$中隨機選擇的一個整數。接下來要在此兩段資料遞迴執行partition。避免這些遞迴計算可以利用一個推疊（stack）來處理。假設partition（A, $l$, r）回傳$m$，則執行：

**If**（$l<m$）push（$l,m$）into stack

**If**（$m+1<r$）push（$m+1,r$）into stack

一開始，堆疊中只有一組資料，（1,n）表示 $A[1..n]$需要排序。如此反覆將堆疊最上面的資料（$l,r$）移出，執行 partition（$A,l,r$），直到堆疊沒有資料為止。

㈠證明在最糟情況下，堆疊的高度可以達到 $n/2$。
㈡設計一個好的演算法以降低 stack 的高度，並證明堆疊的高度最多只需要 $\log n+1$。

【擬答】：

㈠首先可以分成兩堆，最小與最大，每次只將最小的資料放置入堆疊，而最大的資料組則繼續執行 partition 的動作。而每次放入堆疊皆有一筆資料，再從最大資料組裡者出找出兩個資料來繼續做排序。
㈡先分成幾個群組，再使用分組排序後之各組中位數，取其整體之中位數當作控制項。