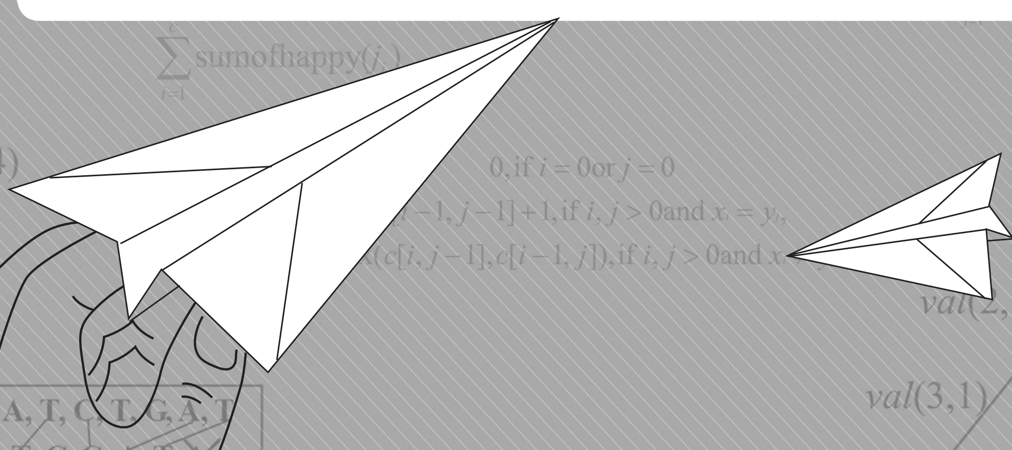
11 逼近演算法

章節大綱

$$\pi = \cfrac{4}{1 + \cfrac{1^2}{3 + \cfrac{2^2}{5 + \cfrac{3^2}{7 + \cfrac{4^2}{9 + \ddots}}}}}$$

圓周率 π 的前一百個數字是 3.14159 26535 89793 23846 26433 83279 50288 41971 69399 37510 58209 74944 59230 78164 06286 20899 86280 34825 34211 70679，不過這也只是逼近的數字而已。



11.1 何謂逼近演算法？

「什麼是逼近演算法(approximation algorithm)？」

簡言之：「設計一個有效率的演算法，找到與最佳答案差距有限的解。」

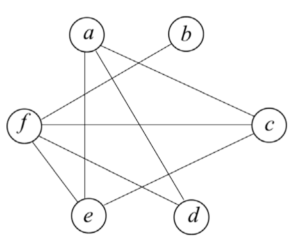
當碰到 NP-complete 問題或其他難題時，想要找出最佳解，常需要花費十分冗長的執行時間；有時，花很長時間找到最佳解，反而不是一個適當的解法。退而求其次，倘若有一個演算法可以很快地找到一個還可以被接受的解，雖然不是最佳的，但是與最佳解差距不多，有時反而可以被接受。

一般而言，設計一個逼近演算法，需說明其解和最佳解的差距有多大，以提供解答的品質保證。

11.2 最小點覆蓋問題

第一個例子是***最小點覆蓋問題***(the minimum vertex cover problem)。

表 11.1 最小點覆蓋問題

問題	給定一個圖 $G=(V, E)$，請找出圖中最小的點覆蓋(vertex cover)。所謂點覆蓋為 V 的子集合 S，使得每一條 E 中的線的其中一個端點(endpoint)，需落在 S 中
輸入	一個圖 $G=(V, E)$ $G=(V=\{a, b, c, d, e, f\}, E=\{(a, c), (a, d), (a, e), (b, f), (c, e), (c, f), (d, f), (e, f)\})$ 
輸出	一個 V 的子集合 S，並符合以下兩個條件： (1)每一條 E 中的線的其中一個端點，需落在 S 中 (2) $\vert S\vert$ (即 S 擁有的元素個數)需最小 $S=\{a, c, f\}$

最小點覆蓋問題(表 11.1)，和上一章提及的點覆蓋問題(表 10.4)，十分相似。其差別在於最小點覆蓋問題，需找到最小的點覆蓋集合並輸出此集合的點；而點覆蓋問題，只需判斷是否存在一個大小為 k 的點覆蓋(輸出是或否)。但是，目前已知點覆蓋問題是 NP-complete 問題；因此，最小點覆蓋問題想必(目前)也不容易有效地被解決，本節將為此問題設計一個逼近演算法。

「什麼是最小點覆蓋問題？」

「找到最少的點，黏到所有的線。」

「怎樣的點可以黏到所有的線？」

「每條線都需被所挑選的其中一點黏到。」

「每一條線有兩個端點，應該挑哪一個呢？」

「不清楚！」

「每一條線的兩端點，可以都不挑嗎？」

「不可以。若是如此，則連接該兩點的線，將無任何點可照顧到。」

「需要兩點都挑嗎？」

「兩點都挑的話，好像又太浪費了。」

以下介紹的逼近演算法，希望找到足夠少的點，來黏住所有的線。以表 11.1 中的圖為例(圖 11.1(a))，首先，考慮圖中任意一條線(a, c)。將其兩端點 a, c 設定為選取，並將所有連到此兩點的線全部除去(圖 11.1(b))。接著，重複以上的動作，直到所的的線被刪除完畢為止。令下一條考慮的線為(b, f)，將 b, f 設定為選取，並將所有連到 b, f 的線全部除去後，得到圖 11.1(c)。此時圖中並無任何線，結束此演算法。

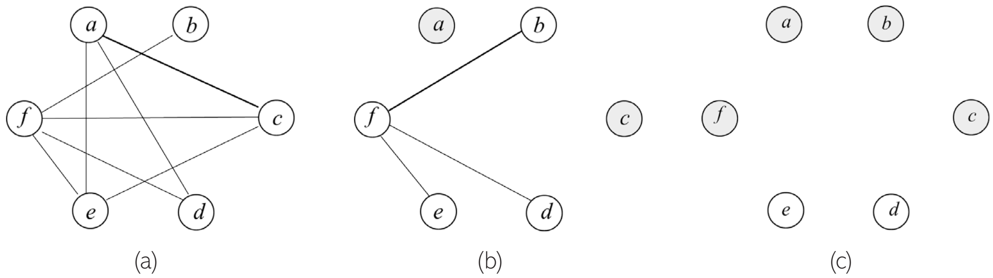
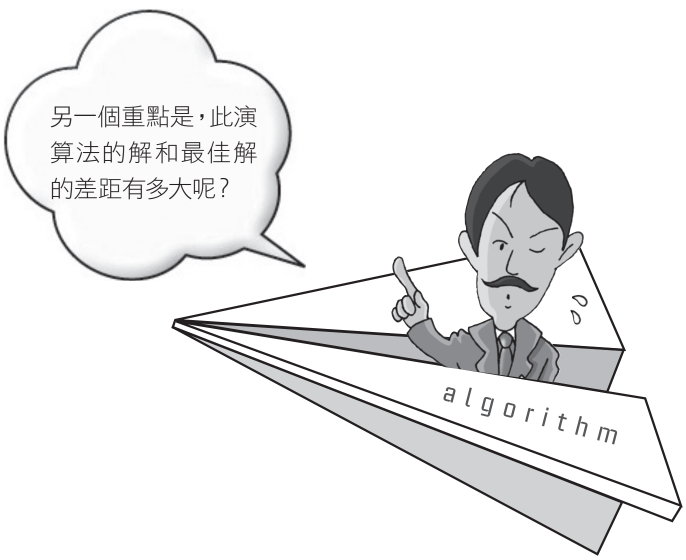


圖 11.1 最小點覆蓋逼近演算法之範例

在整個過程，我們選取$\{a, b, c, f\}$為點覆蓋。雖然比最佳解(如$\{a, c, f\}$)所需的點多，此演算法顯然十分簡單且速度快。



此演算法所得到的點覆蓋數目，不會超過最小點覆蓋的兩倍。原因是，每一條被考慮的線中，這個逼近演算法會選取兩個點，而最佳解需在此兩點中，至少選取一點；否則此被考慮的線，不會被最佳解中任何點黏住(出現矛盾)。上述的逼近演算法，請見表 11.2。

表 11.2 最小點覆蓋逼近演算法

輸入	一個圖 $G=(V, E)$
輸出	點覆蓋 S，使得｜S｜≦ $2\times S^*$，此處 S^*為最小點覆蓋

步驟	Algorithm vertex_cover_approx ()
	{
	/*集合 S 儲存選取的點，集合 C 紀錄的是剩下的線*/
	Step 1: S=φ;C=E;
	
	/*當集合 C 中有線時，執行下列步驟*/
	Step 2: while C≠φ do
	{
	自 C 中任意選出一線(x, y);
	將點 x、y 加入 S 中;
	刪除在 C 中和點 x、y 相連的線;
	}
	Step 3: 輸出 S;
	}

11.3 裝箱問題

下一個例子是裝箱問題(bin packing problem)。

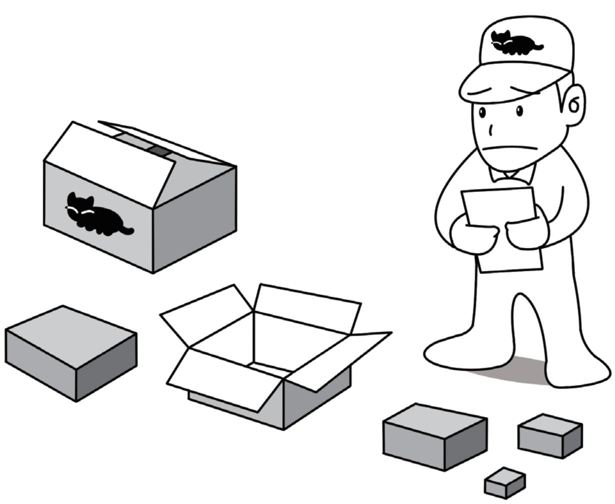


表 11.3 裝箱問題

問題	老王任職在一個貨運公司，擔任包裝工作。目前有一批不同重量的貨物，需要被打包於同樣規格的箱子中。此箱子的最大承受重量是固定單位，而且打包的過程只需要考慮重量因素 請問老王應該如何將貨物分開裝箱，才能使用最少的箱子以節省包裝成本
輸入	一批貨物其重量為$\{w_1, w_2, \cdots, w_n\}$。每只箱子的最大承受重量為單位重量(即 1)，且每個貨物的重量不超過單位重量(即 $1<i\leqq n$，$0<w_i\leqq 1$) {0.2, 0.1, 0.5, 0.7, 0.4, 0.2, 0.3}
輸出	最少使用箱子數目 3 (即{0.2, 0.1, 0.5, 0.2}, {0.7, 0.3}, { 0.4})

裝箱問題也是 NP-hard，因此我們將專注於設計一個逼近演算法。最直覺簡單的作法是，利用首次合適(first fit)的概念：所有箱子按照由小到大的編號固定排列，每個貨物依次按照此排列，尋找第一個可以置入的箱子並直接放入。當輸入為{0.2, 0.1, 0.5, 0.7, 0.4, 0.2, 0.3}時，則需要三個箱子，且其裝箱方式為{0.2, 0.1, 0.5, 0.2}, {0.7, 0.3}, {0.4}。表 11.4 將說明，首次合適演算法的解不會超出最佳解的兩倍。

表 11.4 首次合適演算法的解不會超出最佳解的兩倍

假設執行首次合適演算法後裝箱的狀況為 $B_1, B_2, \cdots, B_m$(B_i 代表編號 i(此處 $1\leqq i\leqq k$)箱子的總重量且 $B_i>0$)，共需要 m 個箱子。令任意相鄰的兩個箱子其編號為 B_i 及 B_{i+1}，則其和 $B_i+B_{i+1}>1$；否則，$B_i+B_{i+1}\leqq 1$，依照此演算法的作法，編號 $i+1$ 箱子的貨物，會全部裝入編號 i 箱子中(因為比較早被考慮到)。又因為 $B_1+B_2>1, B_3+B_4>1, \ldots, B_{\lfloor m/2\rfloor\times 2-1}+B_{\lfloor m/2\rfloor\times 2}>1$，最佳解使用箱子的數目 n，最少需 $\lfloor m/2\rfloor+1$ (大於 $m/2$)個箱子，故 $n>m/2$。推導可得 $2n>m$；因此，首次合適演算法的解不會超出最佳解的兩倍。

11.4 平面上的旅行推銷員問題

最後一個例子是平面上的旅行推銷員問題(traveling-salesman problem)。一般的旅行推銷員問題是，在一個圖中，找出一條最小成本的路徑，使得此路徑經過所有的點剛好一次，且繞回原起始點。本節討論的平面上的旅行推銷員問題稍有不同，需符合三角不等式(triangle inequality)之性質。

表 11.5 平面上的旅行推銷員問題

問題	老王是一個勤快的推銷員，每天需要拜訪每一位客戶剛好一次；並且，最後需回到原來出發地點。但是為了節省時間，老王每每利用最短的直線矩離前往下一個客戶處。請設計一個演算法，幫他找到一個可以拜訪所有客戶一圈的最短旅程
輸入	平面上 n 個點 $P=\{p_1, p_2, \cdots, p_n\}$ {(1, 2), (1, 4), (3, 3), (3, 5), (4, 1), (5, 4) }
輸出	一個旅程 $p_{o(1)} \to p_{o(2)} \to \cdots \to p_{o(n)} \to p_{o(1)}$ 其整個旅程的距離總和最短。此處旅程中的兩點 $p_1=(x_1, y_1)$，$p_2=(x_2, y_2)$ 距離是指***歐幾里得距離***(Euclidean distance)，即 $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$ (1, 2)→(1, 4)→(3, 5)→(5, 4)→(3, 3)→(4, 1)→(1, 2)

平面上的旅行推銷員問題上的點皆置於平面上，而且任兩點的距離(成本)為連接兩者的直線距離；因為幾何上的三角形有兩邊長的和大於第三邊的特性，故點 a 直接連到點 b 的距離(成本)，小於或等於點 a 繞過點 c 再連接點 b 的距離和(圖 11.2)。所以平面上的旅行推銷員問題，符合三角不等式之性質。儘管如此，這個問題還是 NP-hard；接下來，本節將為此問題設計一個逼近演算法。

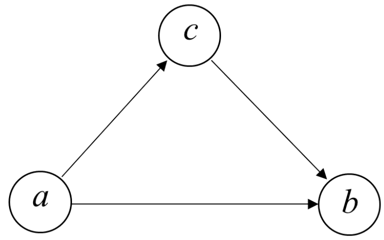


圖 11.2　平面上的旅行推銷員問題符合三角不等式之性質

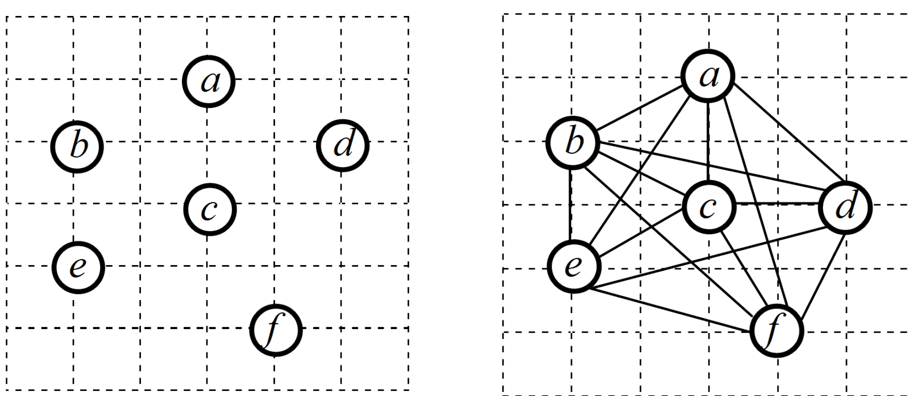
這個逼近演算法先利用一棵樹(tree)將所有的點連接後，再自此樹上找尋一個拜訪每一個點的旅程。此演算法找到的解的距離總和不會超過最佳解的兩倍，其演算法請見表 11.6 及圖 11.3。

表 11.6 平面上的旅行推銷員逼近演算法

輸入	平面上 n 個點 $P=\{p_1, p_2, \cdots, p_n\}$
輸出	一個旅程 $p_{o(1)} \rightarrow p_{o(2)} \rightarrow \cdots \rightarrow p_{o(n)} \rightarrow p_{o(1)}$ 其整個旅程的距離總和，不會超過最佳解的兩倍

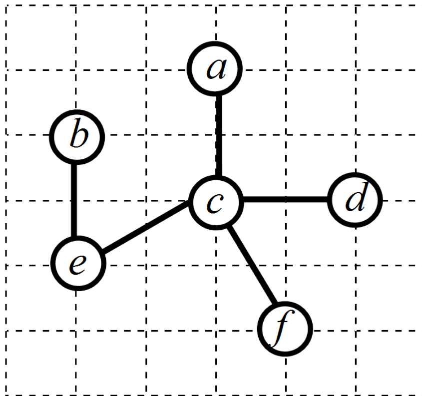
步驟	Algorithm TSP_approx ()
	{
	/*將平面上的點轉成一個圖*/
	Step 1: 建造一個圖 $G=(V, E)$，使得 V 中的每一個點 v_i 代表平面上的一個點 p_i $(1 \leqq i \leqq n)$，而且任意兩點 v_1 和 v_2 存在一條線 (v_1, v_2) 落入 E 中，並令其權重為 p_1 到 p_2 的距離
	Step 2: 在圖 G 上執行 Kruskal 最小成本生成樹演算法，並產生最小成本生成樹 T
	Step 3: 在 T 上任選一點 v 當作樹根(root)，執行前序拜訪並記錄每個點被拜訪的順序 $H=v_{o(1)} \rightarrow v_{o(2)} \rightarrow \cdots \rightarrow v_{o(n)}$，此處的 $v_{o(1)}$ 為 v
	Step 4: 輸出旅程 $p_{o(1)} \rightarrow p_{o(2)} \rightarrow \cdots \rightarrow p_{o(n)} \rightarrow p_{o(1)}$
	}

上述演算法中所謂的***前序拜訪***(preorder traversal)是指，在一棵樹中遞迴地拜訪一個點後，才開始拜訪其兒子們。以下是上述演算法的一個範例。

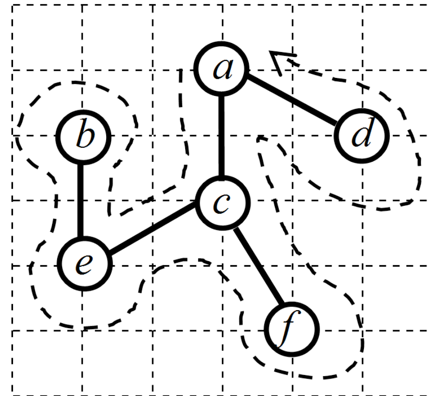


(a) 平面上的點集合

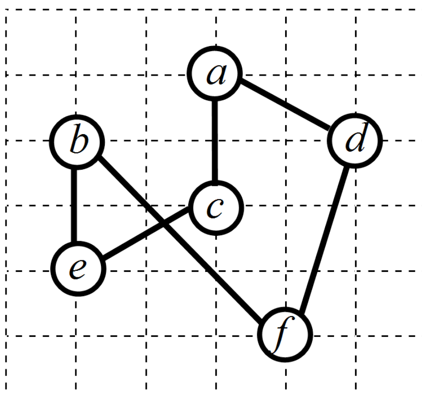
(b) 將平面上的點轉成一個圖，且設定線的權重為連接兩點的直線距離



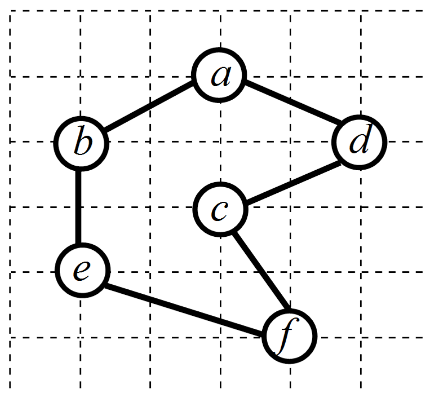
(c) 找出此圖的最小成本生成樹 *T*



(d) 以 *a* 為起始點，列出 *T* 的前序拜訪順序



(e) 將 *T* 的前序拜訪順序，最後連接回始點 *a*，形成一個完整的旅程



(f) 此範例之最佳旅程

圖 11.3　平面上的旅行推銷員逼近演算法之範例

最後，表 11.7 說明了平面上的旅行推銷員逼近演算法，找到的旅程的距離和，不會超過最佳解的兩倍。

表 11.6 平面上的旅行推銷員逼近演算法找到的旅程的距離和，不會超過最佳解的兩倍

令最短的旅程為 TSP^*。若將此旅程扣除任一個線，則成為一棵生成樹；因此，其距離和 $C(TSP^*)$ 必大於或等於最小成本生成樹 T 的距離和 $C(T)$，即 $C(T) \leqq C(TSP^*)$。

以下說明此逼近演算法找到的旅程 TSP 的距離和 $C(TSP)$，小於或等於最小成本生成樹 T 所有線權重(距離)和的兩倍，即 $C(TSP) \leqq 2 \times C(T)$。

假想最小成本生成樹 T 是一道圍牆，自樹根繞此牆一周回到原點，將形成一個路徑 E。此路徑有時會經過同一點多次；例如，在圖 11.3 (d) 找到的路徑 U 為 $a \to c \to e \to b \to e \to c \to f \to c \to a \to d \to a$。注意路徑 U 中所有線的距離和 $C(U)=2 \times C(T)$，因為 T 中的每一條線剛好被經過兩次。又因 $C(T) \leqq C(TSP^*)$，故 $C(U) \leqq 2 \times C(TSP^*)$。

旅程 TSP 其實是將路徑 U 中(除了頭尾外)，忽略重複經過的點後，所形成的旅程。例如，在圖 11.3 (e) 中 TSP 是將路徑 U：$a \to c \to e \to b \to e \to c \to f \to c \to a \to d \to a$，忽略(跳過) 重複點 e, c, c, a 後，得到路徑 $a \to c \to e \to b \to f \to d \to a$。其中，路徑的片段 $b \to e \to c \to f$ 是被 $b \to f$ 所取代。但是因為兩邊長的和大於第三邊，故 $C(b \to e \to c \to f) \geqq C(b \to c \to f) \geqq C(b \to f)$。因此 TSP 的距離和不大於路徑 U 的距離和；即 $C(TSP) \leqq C(U)$。

因為 $C(TSP) \leqq C(U)$ 且 $C(U) \leqq 2 \times C(TSP^*)$，故 $C(TSP) \leqq 2 \times C(TSP^*)$。即平面上的旅行推銷員逼近演算法找到的解，其距離和不會超過最佳解的兩倍。

11.5 逼近演算法的技巧

一個逼近演算法的解和最佳解之差距，需要被嚴格保證。若是您設計一個演算法後，雖然執行後所得的解，常常十分靠近最佳解。但是倘若您提不出嚴格的證明，來保證每一次執行的品質，則您的演算法暫時不適合稱為逼近演算法。當然，一個逼近演算法的解和最佳解之差距愈小，一般會覺得更有價值。

學習評量

1. **感測網路部署問題**：

輸入一個 $k \times k$ 正方形，代表一個感測網路部署的區域。請盡量佈署最少數目的感測器(半徑為 r 的圓)，來覆蓋(監控)整個區域。

輸入

10	(正方形的寬 k)
4	(感測器的半徑 r)

輸出

4	(部署感測器的個數)

2. 請寫一個程式來驗證，在裝箱問題中，利用首次合適(first fit)演算法的解，不會超出最佳解的兩倍。

輸入

7	(貨物的個數 n)
0.2	(以下是這批貨物的重量$\{w_1, w_2, \cdots, w_n\}$，$0<w_i \leqq 1$)
0.1	
0.5	
0.7	
0.4	
0.2	
0.3	

輸出

3	(使用箱子數目)

3. **顏色多項式**：

一間工廠製作 n 個化學產品。但是有些化學產品不相容，如果互相接觸可能產生爆炸。這間工廠準備建造 x 間儲藏室來存放這些化學產品，並且希望這些不相容的化學產品，被放置於不同的儲藏室。為了決定需要興建多少間儲藏室，希望知道當 n 個化學產品被安全置放在 x 間儲藏室中，會有幾種不同的配置方式。

注意每間儲藏室被視為不同，而且可以存放任意多的相容化學產品。如果有一些化學產品被安置在不同的儲藏室中，兩個配置方式將被視為不同。例如，針對三個化學產品 C_1、C_2、C_3 (n=3)和兩間儲藏室 P_1、P_2 (x=2)，其中 C_1，C_2 不相容，$C_1(C_2)$相容於 C_3。我們有以下四種配置方式：

(1) $P_1=\{C_1\}$，$P_2=\{C_2, C_3\}$。
(2) $P_1=\{C_2\}$，$P_2=\{C_1, C_3\}$。
(3) $P_1=\{C_2, C_3\}$，$P_2=\{C_1\}$。
(4) $P_1=\{C_1, C_3\}$，$P_2=\{C_2\}$。

一般而言，針對 n 個相容的化學產品和 x 間儲藏室，我們可以有 x^n 配置方式；x^n 就被稱為其顏色多項式(chromatic polynomial)。相對地，針對 n 個不相容的化學產品和 x 間儲藏室，我們可以有 $x(x-1)(x-2)\cdots(x-n+1)$ 配置方式(此處 $x \geqq n$)。同樣地，$x(x-1)(x-2)\cdots(x-n+1)$為其顏色多項式。前一段的範例所對應的顏色多項式為 x^3-x^2。因此當儲藏室的個數 x=2，我們可有 $2^3-2^2=8-4=4$ 配置方式。

輸入 n 個化學產品與其不相容的關係，請寫一個程式，輸出其對應的顏色多項式。

輸入

```
3        (化學產品的個數)
1        (化學產品間不相容關係的數目)
1 2      (以下是每一個不相容關係)
```

輸出

```
1 -1 0 0 (顏色多項式的係數)
```