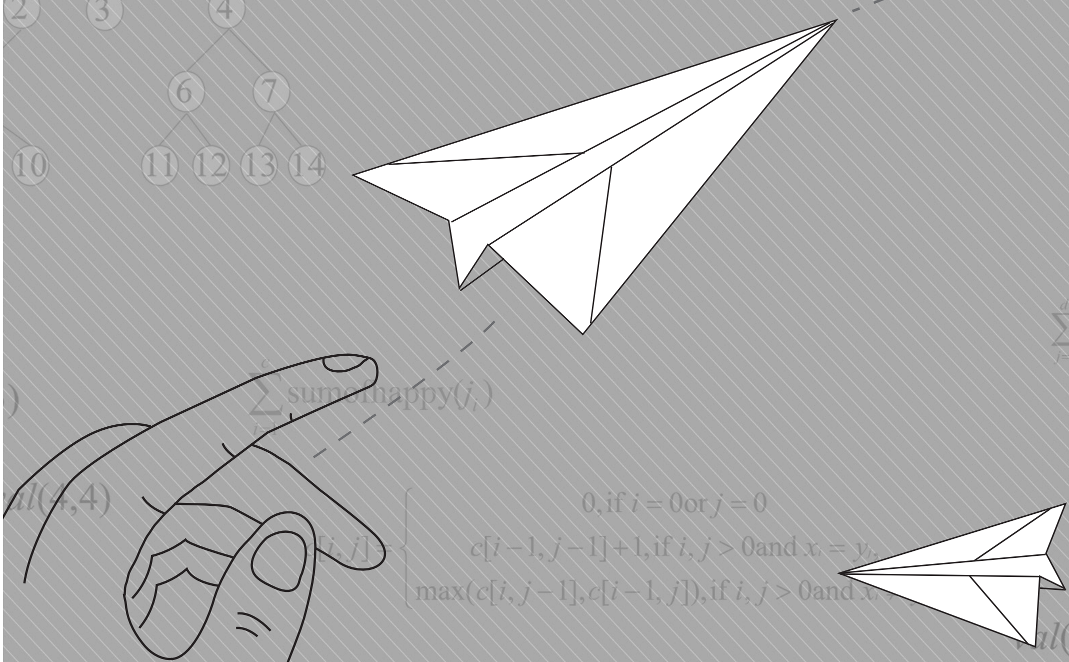
5

刪除搜尋法

章節大綱

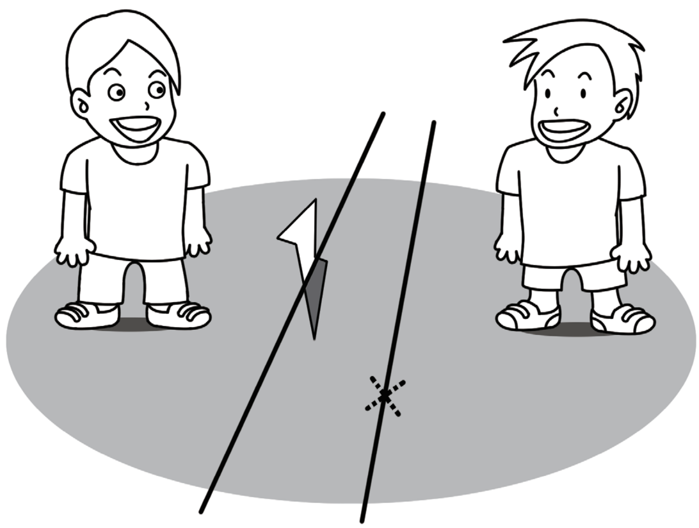


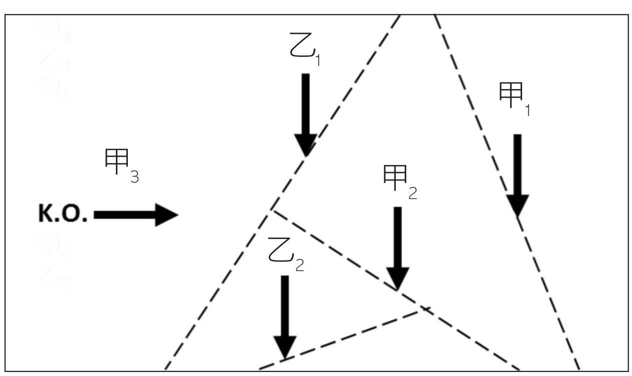
5.1 何謂刪除搜尋法？

「何謂刪除搜尋法(prune and search)？」

簡言之：「在尋找解的過程中，重覆地將解的可能範圍縮小，直到可以直接找到解為止」。

刪除搜尋法好像是一個古老鄉下小朋友的遊戲。





首先，用竹片在泥地上畫一個長方形。第一個小朋友將此竹片，向長方形中丟擲，並使之插立於泥地上，並任意劃一條直線經過此點，同時切割此長方形成兩半。接下來，另一個小朋友自此兩半中，擇一丟擲並使之插立於泥地上；同樣地，任意劃一條直線經過此點，並且將此選擇的區域切割成兩半。

兩方不斷輪流，直到一方丟到選擇的區域的外面或未插立於泥地上而宣告失敗。這個遊戲的特點是，可以選擇的區域不斷變小，使得將竹片丟中的難度愈來愈高。此遊戲和刪除搜尋法都是將解空間不斷縮小的一種方法。

5.2 壞蛋

第一個刪除搜尋法的例子，是如何從一窩蛋中找一顆壞掉的蛋。

表 5.1 找壞蛋問題

問題	有一位農夫想在一窩蛋中，將其中一顆壞了的蛋找出來。只知道壞了的蛋只有一顆，而且比正常的蛋輕。此時農夫家中只有天平，並無秤重的磅秤。請替他設計一個演算法解決此問題。在此假設所有的好蛋的重量都是一樣的
輸入	一窩蛋共 n 顆
輸出	找到唯一(較輕)的壞蛋

找到此顆壞蛋的方法得善用天平。因為壞蛋的重量較輕，故當天平兩邊有相同數量的蛋時，較高(輕)的一邊即為壞蛋所在的位置。以圖 5.1 為例，將一窩十顆蛋平分於天平的兩邊，觀察發現天平向右傾斜；壞蛋在左邊的五顆蛋中，因此可刪除壞蛋在右邊的可能(圖 5.1(a))。利用相同方法，再將五顆蛋平分於天平的兩邊，多餘的一棵暫時放在一旁。若天平是左右一般高，則放在旁邊的蛋是壞的(圖 5.1(b))。若有一邊較高則壞蛋落於高的那一邊(圖 5.1(c)及(圖 5.1(d))。

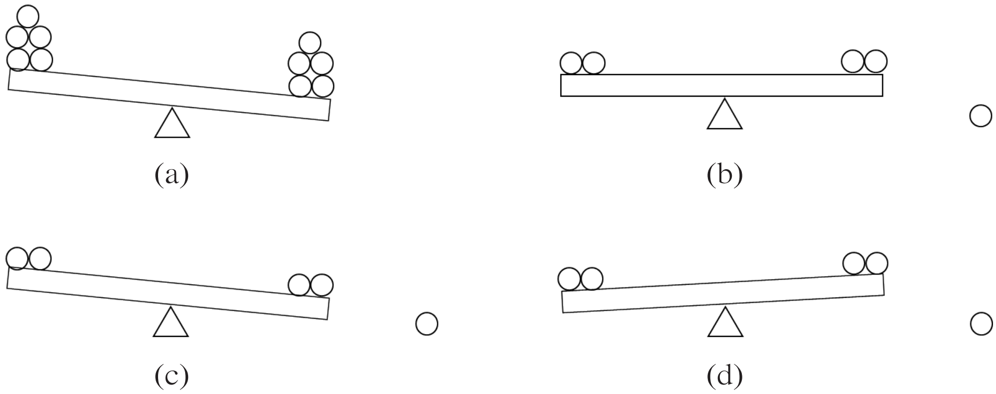


圖 5.1 壞蛋落於天平高的那一邊

以上作法的特性是：每秤一次天平，就將大約 1/2 的蛋排除。當剩下 2 個或 3 個蛋時，再秤最後一次天平就可以找到壞蛋了。顯然地，這個演算法的時間複雜度為 $O(\log_2 n)$，此處 n 為蛋的個數。

5.3 猜數字問題

第二個刪除搜尋法的簡單例子，是猜測事先預定好的一個正整數(小於 100)。當每次猜測的數字輸入時，會給「猜中」、「太高」、或「太低」的指示。

假設事先預定好的整數是 27，當猜測 45 時則會指示「太高」，而且此時解的範圍是在 1~44 之間。當猜測 18 時，則會指示「太低」，而且此時解的範圍縮小為 19~44 之間。當猜測 29 時，則會指示「太高」，而且此解的範圍再縮小為 19~28 之間。當猜測 26 時，則會指示「太低」，而且此解的範圍再縮小為 27~28 之間。顯然，最多再兩次猜測即可猜中，如下圖所示。

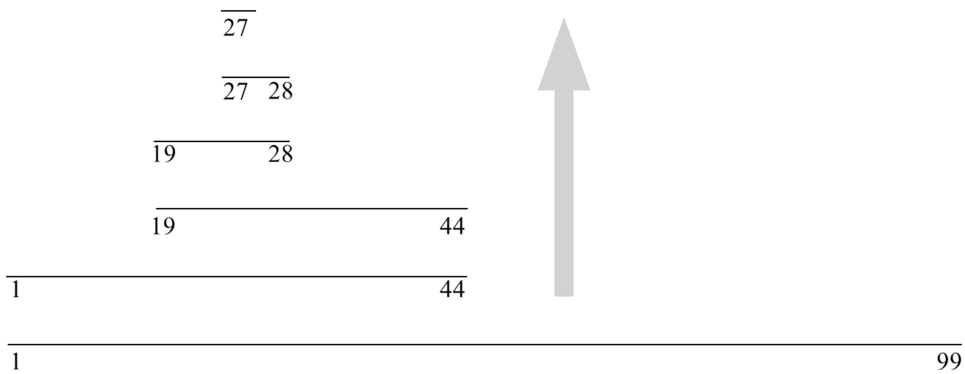


圖 5.2 解的範圍不斷地縮小

以上這個任意猜測的方法，不能保證所需的猜測次數。但是，經過幾次的嘗試後，可以知道，若是每次總是猜測可能解範圍的中間值，至少可消除一半的解範圍。這種猜測方式的演算法，請參考下表。

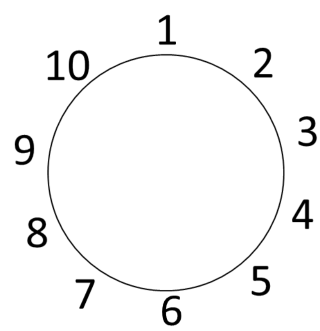
表 5.2 猜數字的演算法

輸入	1 到 100 之間的一個正整數 *x*
輸出	「猜中」、「太高」、或「太低」的指示
步驟	Step 1: 利用亂數函數產生 x，使得 $1 \leq x \leq 100$。 Step 2: 使用者輸入猜測的數字 z。 Step 3: 如果 z=k 則輸出「猜中」並停止此程式。如果 $z>x$ 則輸出「太高」。 如果 $z<k$ 則輸出「太低」。 Step 4: 跳至 Step 2。

5.4 喬瑟夫問題

接下來這個問題十分有趣，是一個利用數學拯救自己性命的一個歷史故事。

表 5.3 喬瑟夫問題

問題	西方有一個十分有數學天份，在一場戰爭戰敗後，他及其他人被敵人抓到監獄中。獄中的囚犯最後決定圍成一個圈圈，並採取以下的自裁行為：自 1 號開始，按照順時針方向，每隔一個人就要自裁，直到剩下一人為止。例如，十位囚犯圍成一個圈圈，如右圖：  則自裁的順序為 2, 4, 6, 8, 10, 3, 7, 1, 9, 最後 5 號囚犯存活下來。這個人因為十分聰明，知道最後存活者是幾號而存活下來。請寫一個程式，輸入囚犯個數 n，並輸出存活下來的囚犯編號 $J(n)$。
輸入	囚犯個數 n 10 13
輸出	最後存活下來的囚犯編號 $J(n)$ 5 11

「哪一個囚犯會存活下來？」

「不知道耶！」

「可以試幾個小例子，再看看是否有規律？」

「當 10 位囚犯時，自裁的順序是 2, 4, 6, 8, 10, 3, 7, 1, 9。最後 5 號囚犯存活下來。」

「當 13 位囚犯時，自裁的順序是 2, 4, 6, 8, 10, 12, 1, 5, 9, 13, 7, 3。最後 11 號囚犯存活下來。」

「有規律否？」

「好像偶數編號的囚犯都最先自裁。」

「奇數也有規律否？」

「好像沒有。」

以下就是前 16 個範例的紀錄。此地 $J(n)$ 代表，當有 n 個囚犯時，最後存活下來的囚犯編號。

表 5.4 喬瑟夫問題的前幾個範例

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$J(n)$	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15	1

從此表中可以得知，當囚犯個數 J=1, 2, 4, 8, 16 時，其輸出 $J(n)$ 都是 1。而且存活下來的囚犯編號，皆是奇數，因為偶數編號的囚犯在第一圈時，就最先自裁了。我們可先刪除所有偶數編號的囚犯會存活到最後的可能。

「哪一個奇數編號囚犯會存活下來？」

「不知道耶！」

「在第一圈後，還存活幾個奇數編號囚犯？」

「大約一半的囚犯。準確地說，如果所有囚犯個數是偶數 $2k$，則在第一圈後，還存活 k 個奇數編號囚犯；如果所有囚犯個數是奇數 $2k+1$，則在第一圈後，還存活 $k+1$ 個奇數編號囚犯。」

「知道還存活 k 個奇數編號囚犯時，哪一個囚犯會存活下來？」

「不知道耶！好像需要重新編號似地，一旦又被編到偶數的囚犯就要倒大楣了。」

「在第一圈殺戮偶數號囚犯後，如果重新編號後，知道哪一個囚犯會存活下來，則這個囚犯的舊編號是多少？」

「肯定是奇數！當新編號是 1, 2, 3, 4 其舊編號則是 1, 3, 5, 7, …。就看新編號是第幾位。如果新編號是第 i 位，則對應的舊編號就是第 $2i-1$ 位。等差級數啦！」

「可否用一個小例子，檢查上述的發現是否為真？」

「假設一開始共有 10 名囚犯。在第一圈自裁 5 名偶數號囚犯後，剩下 5 名奇數號囚犯。在此剩下 5 名奇數號囚犯中，誰會是最後存活者？簡單！查查表 5.4 就好了，$J(5)=3$。所以是順時針排第 3 名的囚犯，他的原先編號是，嗯，1, 3, 5($=2\times3-1$)。最後存活者是編號為 5 的囚犯。」

「如何利用符號紀錄上述的發現？」

「$J(10)=5$。換句話說，在第一圈自裁 5 名偶數號囚犯後，$J(10)=2\times J(5)-1=2\times3-1=5$。」

「$J(2k)=2J(k)-1$？」「另一方面，$J(2k+1)=$？」

「嗯…」

當所有囚犯個數是奇數 $2k+1$ 時，則在第一圈並刪除編號 1 後，還存活 k 個奇數編號囚犯。因此 $J(2k+1)=2J(k)+1$。若再加上先前的發現(即 $J(2k)=2J(k)-1$)，我們就可以設計出一個演算法來解決喬瑟夫問題。

表 5.5 喬瑟夫問題的演算法

輸入	囚犯個數 *n*(>=1 的正整數)
輸出	最後存活下來的囚犯編號 *J*(*n*)

步驟	`Algorithm J(n)`
	`{`
	`if n=1 then 輸出 J(1)=1。`
	`else {`
	`      if n=2k, then J(2k)=2J(k)-1;`
	`      else J(2k+1)=2J(k)+1;`
	`      }`
	`}`

公式 $J(2k)=2J(k)-1$ 也說明了「J=1, 2, 4, 8, 16 時，其輸出 $J(n)$都是 1」的原因；例如，$J(4)=2J(2)-1=2(2J(1)-1)-1=1$。

其實，還有一個更快更直接的計算方式。這個方式巧妙地隱藏於二進位表示法中。以下是將表 5.4 的數值化成二進位表示法，以方便進一步觀察。

表 5.6 神奇的規律隱藏於二進位表示法中

n	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111
J(*n*)	1	01	11	001	011	101	111	0001	0011	0101	0111	1001	1011	1101	1111

「*n* 和 *J*(*n*)之間是否有規律？」

「好像出現 1 的數目是相同。」

「排列順序有規律否？」

「好像沒有。不！好像是在繞圈圈。」

「繞圈圈？」「可以換個方式說清楚嗎？」

「將 n 的每個數字向左移一格，同時最左的 1 像繞圈圈一樣移到最右邊，就變成 $J(n)$。」

「你是正確的，但是為什麼呢？」

「嗯…」

「目前在甚麼狀況下，這個問題最容易被解決的？」

「當 J=1, 2, 4, 8, 16 或 2^k 時，最後存活的囚犯編號必為 1。」

「可以利用這個結果嗎？」「困難點在何處？」

「所有囚犯個數不會總是 2^k。除非囚犯人數可以減少。」

「甚麼方法可以讓囚犯人數減少？」

「如果可以，先讓幾個囚犯人數自裁。直到剩下 2^k 個囚犯，這樣最後自裁的囚犯的下一位，就是最後存活者。」

「需要裁多少囚犯？」

「多於 2^k 的囚犯就裁掉呀！。例如，J=10 時，需要裁到剩下 2^3 個囚犯；也就是 $10=2^3+2$，需要裁 2 位囚犯。」

「知道最後自裁的囚犯是誰就好了？」

「不難呀！因為要先裁 2 位囚犯，所以自裁的順序是從偶數的 2, 4, 6, …算起；2×2=4，最後自裁是編號 4 的囚犯。」

「那麼，最後存活的囚犯編號是多少？」

「最後自裁囚犯的下一位，就是編號 4 的下一位；自然地，4+1=5，編號 5 的囚犯就是最後存活者了。」

「這個方法和先前觀察的規律，有關係嗎？」

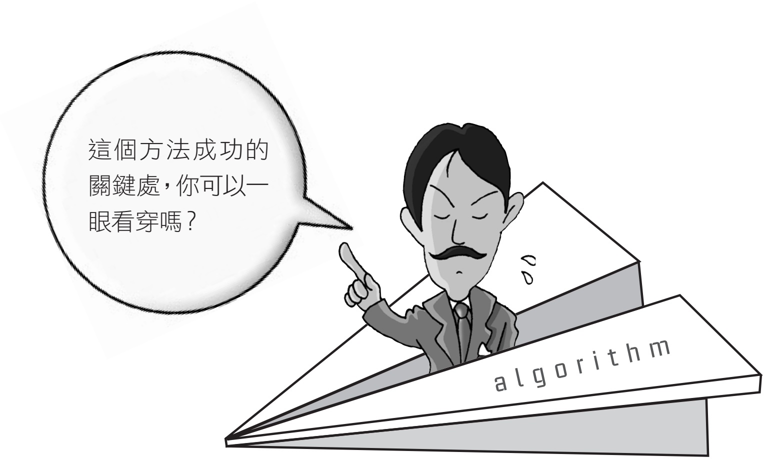
「當 $n=10=(1010)_2$ 時，將$(1010)_2$ 的每個數字向左移一格，得到$(0100)_2=4$，剛好是最後自裁是囚犯的編號。」「之後，再將最左的 1 移到最右邊，剛好得到最後存活者的編號 $J(n)=(0101)_2=5$。真是神奇呀！」

「將 $10=(1010)_2$ 的每個數字向左移一格，就得到最後自裁囚犯的編號？為什麼？」

「因為$(1010)_2-(010)_2=(1000)_2=8=2^3$，$(010)_2$ 就是需要減少的囚犯的數目。將$(1010)_2$ 的每個數字向左移一格，就是$(010)_2=2$ 乘以 2 變成$(0100)_2$，就是計算最後自裁囚犯的編號！」

「為何最左的 1 需要移到最右邊？」

「最後自裁囚犯的下一位，就是最後存活者；這個+1 的動作，只是在算下一囚犯的編號。」



下表是利用此方法所設計出的快速演算法。

表 5.7 喬瑟夫問題的快速演算法

輸入	囚犯個數 n
輸出	最後存活下來的囚犯編號 $J(n)$

步驟	Algorithm $J(n)$
	{
	Step 1: 將 n 表示成 2^k+m，使得 k 越大且 m 為正值。
	Step 2: 輸出 $2m+1$。
	}

5.5 簡化的線性規劃問題

線性規劃是一個應用廣泛的解題工具。因此，有效率地找到線性規劃問題的解，是一個程式設計的基本課題。首先，利用一個例子來引出線性規劃問題，再設計一個有效率的刪除搜尋演算法，來解決線性規劃中的一個簡化問題。這是一位農夫想養豬羊來賺錢的問題，請參考表 5.8。

表 5.8 農夫養豬羊問題

問題	某一位農夫想在自己的農舍上養豬羊來賺錢。他向朋友借了 200, 000 元當作創業資本。但是，養一頭豬或羊都需養 12 個月後，才可出售。購買一頭小豬的成本需 1,500 元，每個月飼養豬的成本是 400 元，一年後出售的價格是 22, 000 元。相對地，如果購買一頭小羊的成本需要 3, 200 元，飼養羊的每個月成本是 700 元，一年後出售的價格是 35, 000 元。請問這位農夫最少需要養幾頭豬幾頭羊，可以在一年後靠賣豬羊換得現金超過 200, 000 元
輸入	一頭豬的購買成本、飼養成本及出售價格 1500, 400, 22000 一頭羊的購買成本、飼養成本及出售價格 2200, 700, 35000
輸出	養 x 頭豬，y 頭羊 0, 6

讓 x 代表農夫飼養豬的數目，y 為飼養羊的數目。我們可以將上述的農夫養豬羊的問題，轉換成以下兩個變數的線性規劃問題。

表 5.9 農夫養豬羊所對應的線性規劃問題

目標式(objective function)：
極小化 $x+y$ 的值(此式在於希望養豬羊的總數最少)

限制式(constraints)：
此解需符合下列每一個條件
(1) $x \geq 0$ 而且 $x \in Z$(即整數)
(2) $y \geq 0$ 而且 $y \in Z$(即整數)
(3) $1500x+2200y+400 \times 12x+700 \times 12y \leq 200000$
(此式等同 $6300x+10600y \leq 200000$ 在於保證養豬羊的成本少於向朋友借的錢)
(4) $22000x+35000y \geq 200000$
(此式在於保證賣豬羊的錢超過 200,000 元)

圖解法常被用來解決簡單的線性規劃問題。此法先繪出其解空間後，再尋找解空間的解，使其符合目標式的要求。對應表 5.9 的解空間如下圖。

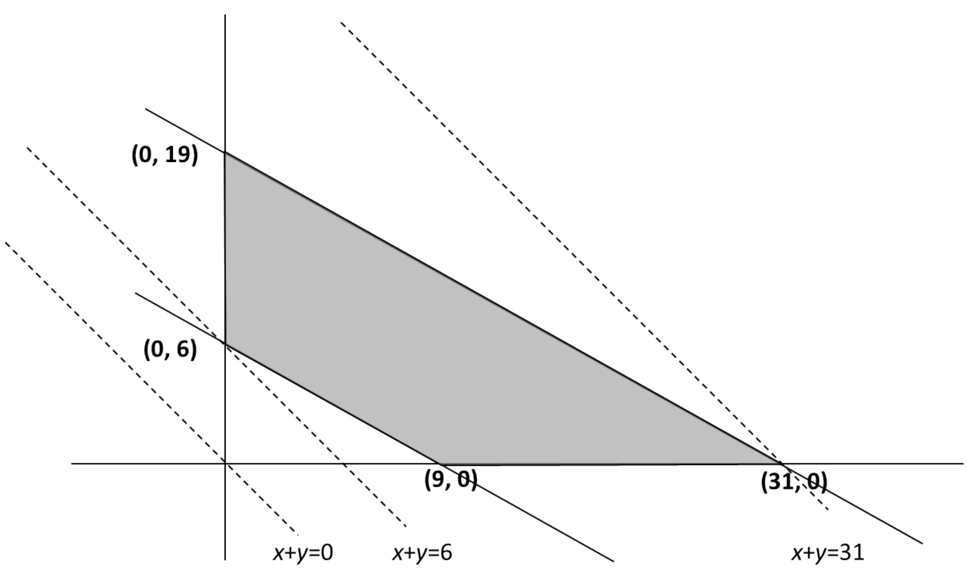


圖 5.3 表 5.9 所對應的解空間

圖 5.3 上任何落在解空間(灰色區域)的點都符合限制式。表 5.9 的問題就是在圖 5.3 的灰色區域中，找到使得 $x+y$ 為最小的點。因此，圖解法接下來尋找經過灰色區域，且和 $x+y=0$ 的平行線(即 $x+y=k$ 的直線)使得 k 為最小。從圖 5.3 得知 $x+y=6$ 為此解。因此此農夫需養 6 頭羊，就可以在一年後靠賣羊換得超過 200,000 元。

此問題的最佳解，必定落在直線的交點上，因此圖解法暗示著一個簡單的演算法。此法就是，先計算出所有任意兩個直線的交點後，過濾去除不符合限制式的交點，再從中找到讓目標式(objective function) $x+y$ 為最小的交點，即為最佳解。當 n 代表限制式的數目時，因為 n 條直線最多有 $\binom{n}{2}=n(n-1)/2$ 個交點，所以這個演算法恐怕最少需要 $O(n^2)$ 以上的時間來執行。

設計更快的演算法來解兩個變數的線性規劃問題，看起來並不容易。我們將介紹一個簡化後的線性規劃問題，並且為之設計一個 $O(n)$ 時間複雜度的演算法。有趣的是，這個演算法在稍加修改後，有機會解決一般的兩個變數的線性規劃問題，其時間複雜度依舊是 $O(n)$。

此簡化問題的目標式固定為 y，而其所有限制式皆為 $y\geq ax+b$ 這種類型(這裡的 a 及 b 是常數)。下表是一個範例及下圖是其對應的解空間。

表 5.10 簡化的的兩個變數線性規劃問題

目標式：找出 y 的最小值
限制式： 此解需符合下列每一個條件 (1) $0\geq -x$ (2) $y \geq -\frac{6}{5}x+6$ (3) $y \geq \frac{6}{5}x-15$ (4) $y\geq 4x-80$

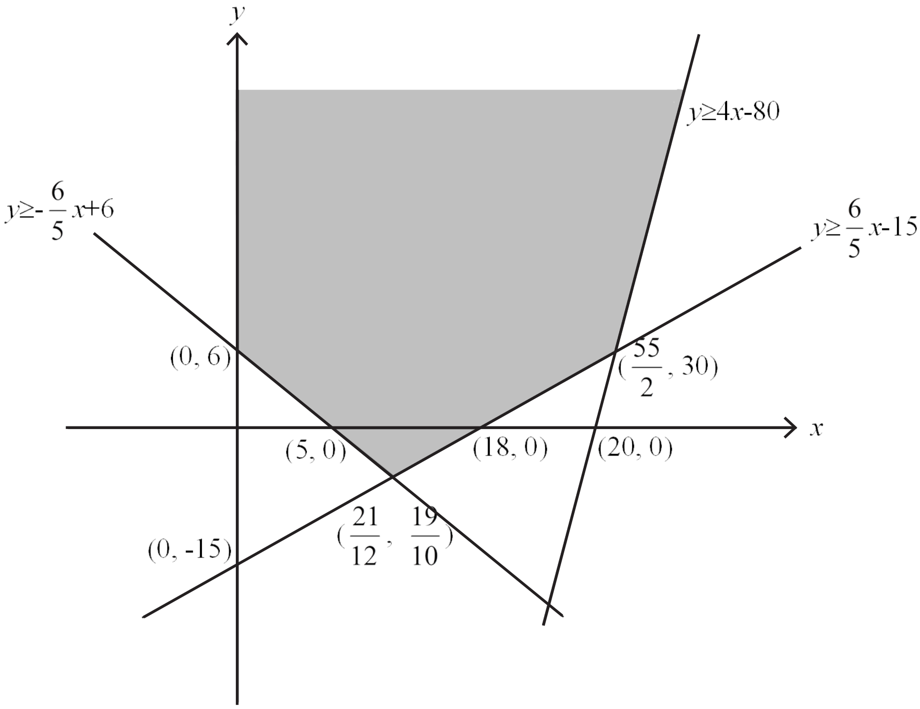


圖 5.4 表 5.10 所對應的解空間

「如何找到這簡化問題的最佳解？」

「嗯…」

「先想想此最佳解有何特性？」

「最佳解在解空間的最低點，此點也正好是兩條線的交點。另外，解空間像一個漏斗，其下方是縮小的。」

「可以利用這個特性來找到最佳解？」

「如果可以沿著解空間的邊界向下找漏斗底部，應可找到最佳解。」

「怎麼找到解空間的邊界？」

「嗯…」

「注視邊界！」「看到什麼特性？」

「邊界是由數條線段所連接組成的。」

「注視線段！」「這些線段有何特性？」

「線段的正上方就是解空間，好像沒有其他的線。解空間的邊界總是出現在最高點。」

「試試看是否可利用這個特性來找到邊界？」

「利用一條由上而下的直線，並計算與其他直線所有的交點。形成最高交點的線會幫助構成邊界。」

「找到邊界對尋找最佳解有何幫助？」

「最佳解是位於邊界最低的方向。也就是計算邊界的斜率可以決定最佳解的方向。」

「如何利用最佳解的方向來協助找到最佳解？」

「嗯…」

利用最佳解的方向來刪除多餘冗贅的限制式，以有效率地找到最佳解，是這一個演算法的核心。下列的範例說明其主要概念。在圖 5.5 中，當最佳解在虛線的左邊且 AB 兩線交點在右邊時，A 線必與最佳解無關。

原因是 AB 兩線在虛線左邊的範圍，B線永遠高於 A 線；因為 AB 兩線皆是 $y \geq ax+b$ 這種類型的限制式，符合 AB 兩線所圍的點都是向上的區域(如箭頭所示)。在虛線的左邊，A 線所圍的區域皆比 B 線所圍的區域大，故可行解(含最佳解)必落在 B 線的上端；因此對找到最佳解而言，A 線是多餘的，可被去除。

值得注意的是，任意兩條相交的直線必可刪除其中一條，而不影響找到最佳解的機會。

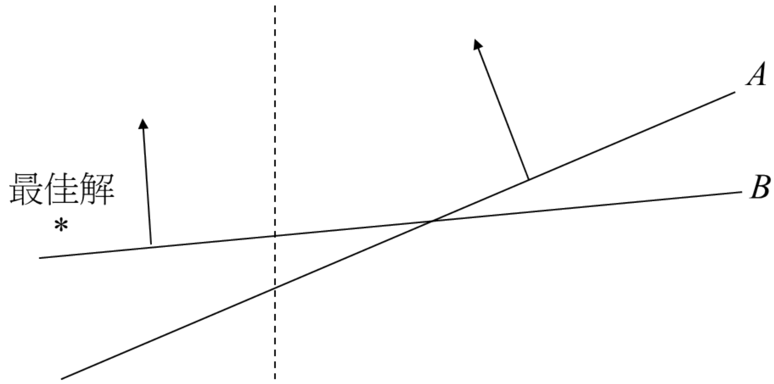


圖 5.5 當最佳解在虛線的左邊且 *AB* 兩線交點在右邊時，*A* 線必與最佳解無關

以下的演算法就是利用這樣刪除搜尋的概念，不斷地刪除一定比率的限制式，直到只剩下兩條限制式為止。

表 5.11 簡化的兩個變數線性規劃問題的演算法

輸入	目標式：找出 y 的最小值
	限制式：$y \geq a_i x_i + b_i$ $(i=1\ldots n)$
	(令同樣斜率的限制式，只保留最高的一條限制式。)
輸出	符合限制式的最佳(小)解

步驟	Algorithm simplified_linear_programming
	{
	Step 1: 如果未超過兩條限制式，則直接解開。
	Step 2: 將限制式兩兩任意配對並解開其交點。用 x_j 代表這些交點的 x 座標。
	Step 3: 找出 x_j 的中間值並令其為 x_m。
	Step 4: 找出與 $x=x_m$ 產生交點最高 $(y=y_m)$ 的線。若有兩條線以上交於此高點 (x_m, y_m)，則令 t_{max} 是斜率最大的線而 t_{min} 為斜率最小的線。
	Step 5: 如果 t_{max} 和 t_{min} 不同正負號則 (x_m, y_m) 即為最佳解。否則，當 t_{max} 為正號時，最佳解在 $x=x_m$ 的左方。當 t_{max} 為負號時，最佳解在 $x=x_m$ 的右方。
	Step 6: 當最佳解在 $x=x_m$ 的左方，針對 $x=x_m$ 右方的每一個交點，刪除(在 $x=x_m$ 左方)較低的那條限制式。當最佳解在 $x=x_m$ 的右方，針對 $x=x_m$ 左方的每一個交點，刪除(在 $x=x_m$ 右方)較低的那條限制式。
	Step 7: 跳至 Step 1。
	}

此演算法在每執行一次迴圈(Step 2 ~ Step 6)後，會刪除大約四分之一的限制式。因為當目前有 n 條限制式時，在 Step 2 中會找到大約 $n/2$ 個交點，而利用直線 $x=x_m$ 切成左右各大約 $n/4$ 個交點後，最後將會刪除大約 $n/4$ 條限制式。

令 $T(n)$ 是執行此演算法所需的時間時，可得 $T(n)=T(3n/4)+O(n)$；因為 Step 1 到 Step 7 都可以在 $O(n)$ 時間內完成(注意 Step 3 可利用第二章找尋第 k 小值的 $O(n)$ 演算法)。解開此遞迴式可知此演算法的時間複雜度為 $T(n)=O(n)$。

5.6 刪除搜尋法的技巧

刪除搜尋法總是需要執行多次的迴圈；並在每次的迴圈中，縮小固定比率的解範圍。如何有效率地縮小固定比率的解範圍，是使用刪除搜尋法的關鍵技巧。

最後，列出另一個可被刪除搜尋法解決的問題，即一個中心問題(the 1-center problem)：給定 n 個平面上的點，找一個最小的圓可以蓋住所有的點。

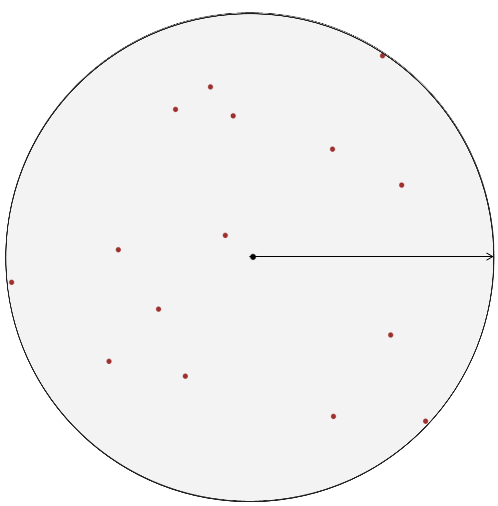


圖5-4 找一個最小的圓, 可以蓋住平面上所有的點。

學習評量

1. **喬瑟夫問題**：請寫一個程式，輸入囚犯個數 n，並輸出存活下來的囚犯編號 $J(n)$。

輸入
```
10                    (囚犯個數 n)
```
輸出
```
5                     (存活下來的囚犯編號 J(n))
```

2. 假設有 $2n$ 個人圍成一個圈圈。前面 n 個人是好人，而後面 n 個人是壞人。請寫一個程式找出一個 m 值，當沿著圈圈處決每第 m 人時，所有壞人會先被處決。

輸入
```
3                     (n值)
```
輸出
```
5                     (m值)
```

3. 輸入平面上 n 個點，請寫一個程式找出一個最小的圓(circle)可以包含所有的平面點。

輸入
```
 4                    (n值)
 1   0                (以下為 n 個平面點的座標)
 0   1
-1   0
 0  -1
```
輸出
```
0 0                   (圓心的座標)
1                     (圓的半徑)
```

Memo