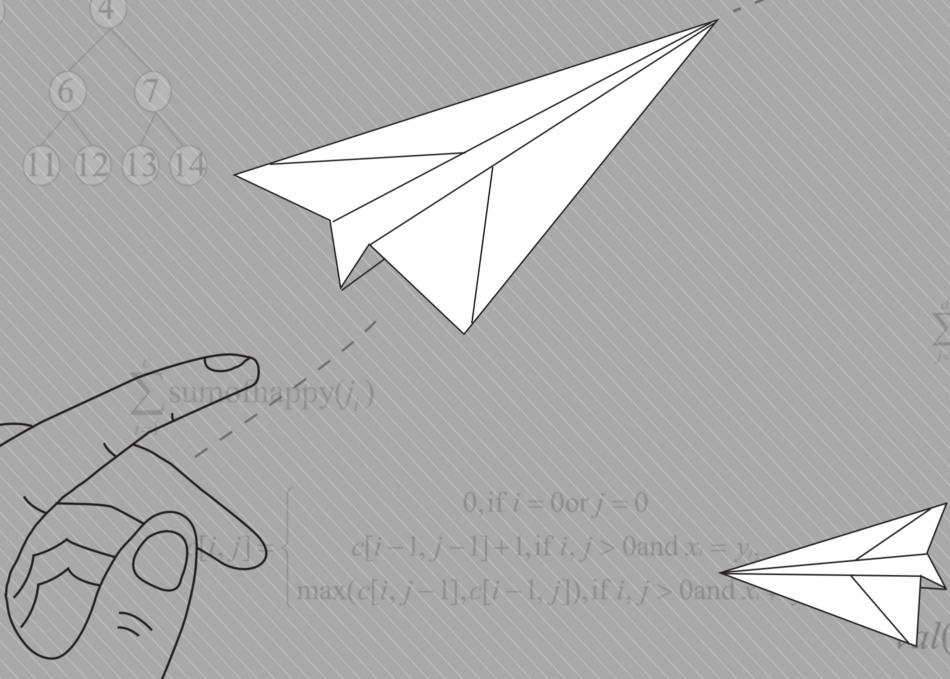
9 計算幾何

章節大綱



9.1 何謂計算幾何？

「什麼是計算幾何(computational geometry)？」

簡言之：「輸入幾何上的物件，並自這些物件中尋找解答的技巧。」

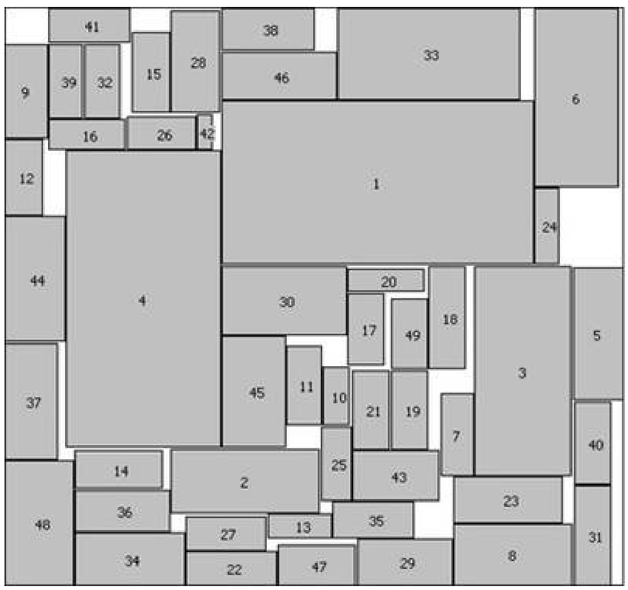


圖 9.1 減少整體電路的製造成本，需要考慮如何將不同大小的長方形擠入給定的空間中

當在設計積體電路時，減少整體電路所占用的面積，有助於降低其製造成本。降低占用的面積時，需要考慮如何將不同大小的長方形擠入一個給定的空間中。另外一個例子是，思考佈署一個無線感測器以監控所有的目標物。感測器的監控範圍可用一個圓代表，而每一個目標物可用一個點表示。如何找出一個最小的圓來涵蓋平面上所有的點，就是將此無線感測器的佈建，轉換成幾何計算的問題。

9.2 多邊形中的點

第一個例子是，判斷一個點是否在一個簡單多邊形內的問題。

表 9.1 判斷一個點是否在一個簡單多邊形內

問題	有一位外國旅客來到台北市旅遊。他想利用手中的衛星定位資訊，來查詢目前所在的區域(例如，是否在大安區？) 請替他設計一個演算法解決此問題
輸入	在平面上，由一連串相鄰的直線或橫線段所組成的一個簡單多邊形，代表一個有興趣的區域地圖 (105, 18)-(129, 18)-(129, 2)-(109, 2)-(109, 5)-(127, 5)-(127, 16)-(110, 16)-(110, 13)-(124, 13)-(124, 10)-(108, 10)-(108, 8)-(122, 8)-(122, 6)-(106, 6)-(106, 0)-(105, 0)-(105, 18) 一個平面上的查詢點，代表目前旅客所在的衛星定位資訊 (123, 9)
輸出	判斷此查詢點是否落入此簡單多邊形中？ 否

這個問題乍看之下以為十分簡單，因為只是判斷一個點是否落在一個多邊形中而已。但是，當這個多邊形變得比較複雜時，這個問題就需要思考一下。例如，在圖 9.2 中，一個黑點乍看之下好像落入此多邊形內，但是仔細一瞧才發覺其實是落在此多邊形外。

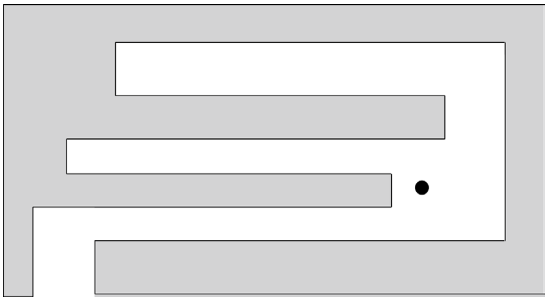


圖 9.2 如何判斷一個點是否落在一個多邊形中?

以下是一個簡單方法可以作此判斷。我們可以任選一個多邊形外的一點，並且將此點連接到查詢點，以形成一個線段(圖 9.3)。接下來，計算此線段和此多邊形的邊，所形成的相交數。若相交數為奇數，則此查詢點在此多邊形內；反之，若相交數為偶數，則此查詢點不在此多邊形內。

例如，在圖 9.3 中，此線段和多邊形的邊產生兩個相交，故查詢點落於多邊形外。

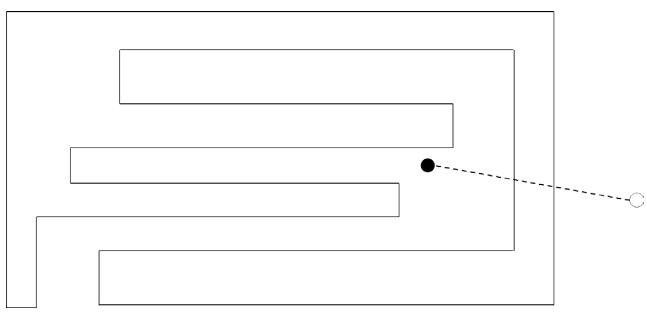


圖 9.3 多邊形外的一點(白點)連接到查詢點(黑點)形成一個線段，和多邊形的邊產生兩個相交，故查詢點落於多邊形外

接下來，考慮此方法的細節。即當此輸入的多邊形，是利用一連串的二維座標，來紀錄相鄰邊的轉角點時(圖 9.4)，我們該如何計算所需的相交數？

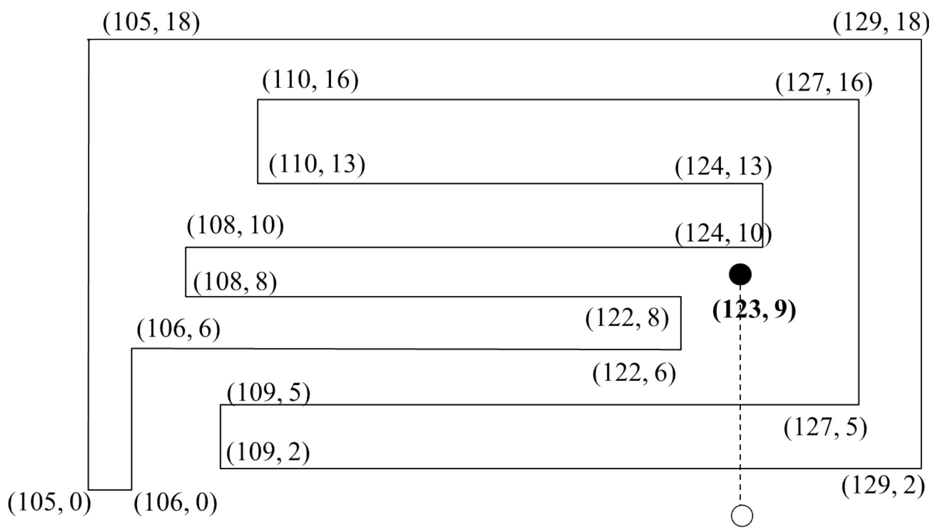


圖 9.4 利用一連串的二維座標，來代表輸入的多邊形。此時查詢點(黑點)的座標為(123, 9)

乍看之下，好像需要一些額外的計算才能完成。但是，若能適當選擇多邊形外的點，將有助於進一步簡化此判斷。

例如，在圖 9.4 中，當我們選擇一點，使得此點連接到查詢點(123, 9)的線段與 y 軸平行時，則此多邊形會與此線段相交的邊，只有兩個邊，即(109, 5)-(127, 5)及(109, 2)-(129, 2)。最後，只需要判斷線段(109, 5)-(127, 5)及(109, 2)-(129, 2)，和直線 x=123 是否相交？若有相交，最後判斷其交點的 y 座標是否小於或等於 9 即可。詳細的演算法如表 9.2 所示。

表 9.2 點在多邊形中演算法

輸入	平面上的一個簡單多邊形 P，其中 n 個點為 $p_1=(x_1, y_1), p_2=(x_2, y_2), \cdots, p_n=(x_n, y_n)$。其中 n-1 個邊 e_i 是由 $P_i=(x_i, y_i)$到 $P_{i+1}=(x_{i+1}, y_{i+1})$的直線或橫線段所組成($i$=1, ⋯, n-1)。最後的一個邊是由 $p_n=(x_n, y_n)$到 $p_1=(x_1, y_1)$的線段組成 一個平面上的查詢點 $q=(x_0, y_0)$
輸出	判斷此查詢點 q 是否落入此簡單多邊形 P 中

步驟	Algorithm point_in_polygon (p,q)
	{
	Step 1: number:=0; /*相交數初值為 0*/
	Step 2: for 所有多邊形 P 的邊 e_i do
	{
	if 線 $x=x_0$ 和 e_i 產生相交 then 令此交點為(x_0, y_k);
	if y_k <y_0 then number=number +1; /*增加一個相交次數*/
	}
	Step 3: if number 是奇數 then 輸出 "q 在多邊形 P 內";
	else 輸出 "q 在多邊形 P 外";
	}

上述演算法只需要 $O(n)$ 時間來執行(這裡 n 代表多邊形邊的個數)。

9.3 天空輪廓

下一個例子是，找出天空輪廓 (skyline) 問題。

表 9.3 天空輪廓問題

問題	輸入一個城市中建築物的位置及形狀(建築物皆為矩形)。請設計一個演算法，找出由這些建築物在天空中所形成的輪廓。此輪廓需去除所有隱藏線
輸入	建築物若干棟。所有建築物皆座落於同一個水平線上。每棟建築物由三個座標(L, H, R)代表之，其中 L 及 R 分別是建築物的左右 x 軸座標，而 H 是建築物的高度 (1, 5, 8), (5, 8, 10), (7, 3, 11), (12, 2, 24), (17, 11, 19), (18, 4, 22)
輸出	天空中的輪廓。此天空中的輪廓，是由一個人站在這些建築物旁朝遠方看去，所見到的外圍形狀；由一連串由左至右的 x 軸座標及高度交替所構成(以下大的粗黑數字代表高度) (1, **5**, 5, **8**, 10, **3**, 11, **0**, 12, **2**, 17, **11**, 19, **4**, 22, **2**, 24, **0**)

天空輪廓問題在表 9.3 中的輸入及輸出，可表示成圖 9.5 及圖 9.6。注意在圖 9.6 中，落在其他建築物(矩形)內的線都被刪除了，只保留最外圍的線。

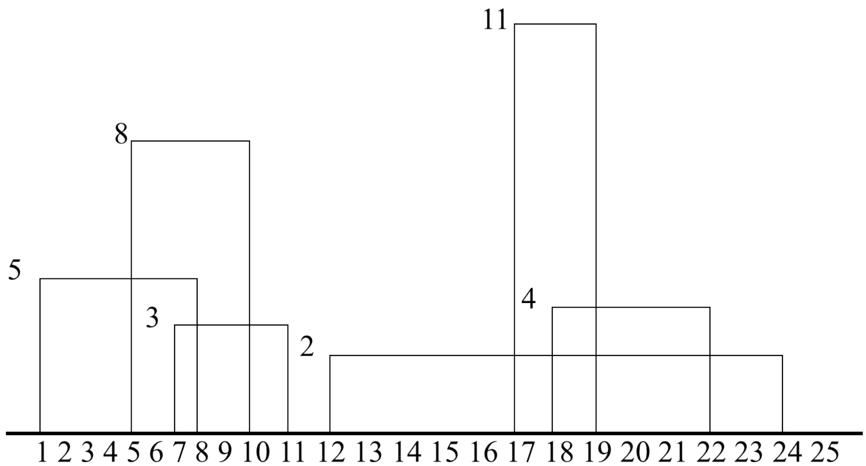


圖 9.5 表 9.3 中的天空輪廓問題之輸入範例

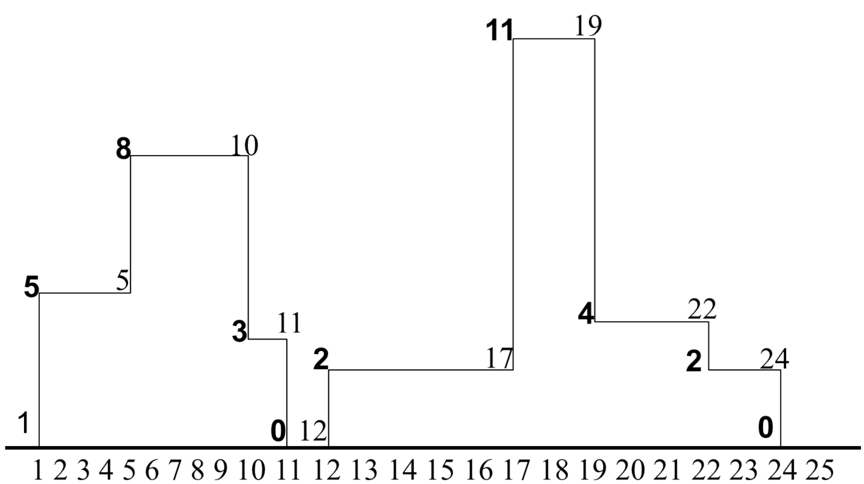


圖 9.6 表 9.3 中的天空輪廓問題之輸出範例(粗黑數字代表高度，非粗黑數字代表 x 軸座標)

解決天空輪廓問題的最簡單方法為：將建築物一棟一棟地加入，並且在每加入一棟建築物後，立刻調整其天空輪廓。如此在所有建築物處理完後，即可得到最後的天空輪廓。為了完成此演算法，假設在原來的天空輪廓上，加入一棟建築物，接下來觀察如何調整其天空輪廓。

例如，在圖 9.6 上加一棟建築物(7, **6**, 21)，如圖 9.7 所示。首先觀察原來的天空輪廓(1, **5**, 5, **8**, 10, **3**, 11, **0**, 12, **2**, 17, **11**, 19, **4**, 22, **2**, 24, **0**) 會作如何的變化？天空輪廓顯然被破壞了，尤其是 *x* 軸座標介於 7 到 21 之間 即新加入建築物的寬度範圍)的輪廓，若是低於 6 (即新加入建築物的高度) 就需要被隱藏起來，並且其高度需要被修正成 6。另外，建築物的左右兩道牆也有可能改變天空輪廓。

例如，在圖 9.7 中, 建築物的左牆被原先的輪廓隱藏了，但建築物的右牆形成新的天空輪廓。而建築物的中間部分被調整為至少大於等於 6 以上。

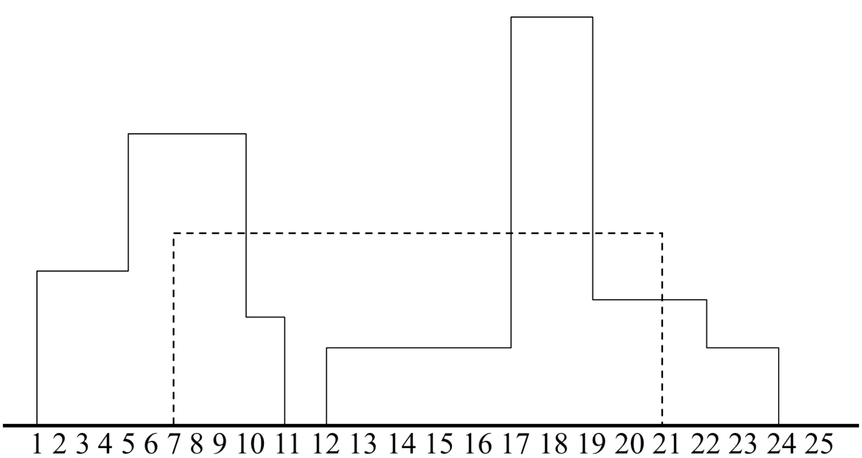


圖 9.7 在圖 9.5 上加一棟建築物(7, **6**, 21)後，天空輪廓被破壞了

我們可以設計一個演算法，由左至右掃描整個天空輪廓並進行調整。首先，找到新加入的建築物(7, **6**, 21)的左牆位置(即 x 軸座標為 7)；接著逐一調整 x 軸座標 7~21 之間的高度。

第一個碰到的平行線段為 5, **8**, 10。因為此線段自 x 軸座標 5 到 10，且其高度為 8，大於新加入建築的高度 6，故不需調整。

第二個平行線段為 10, **3**, 11，因為其高度低於新加入建築的高度 6，故需改成 10, **6**, 11。同樣的理由下兩個平行線段 11, **0**, 12 需改成 11, **6**, 12，而 12, **2**, 17 需改成 12, **6**, 17。此三個線段 10, **6**, 11、11, **6**, 12、12, **6**, 17 因為同一個高度，可合併成 10, **6**, 17。

再下一個平行線段 17, **11**, 19 其高度超過新加入建築的高度 6，故不必調整。

最後一個平行線段 19, **4**, 22 已經超過建築物右牆(其 x 軸座標為 21)，因為其高度仍低於新加入建築的高度 6，我們需將 19, **4**, 22(可看成 19, **4**, 21, **4**, 22)改成 19, **6**, 21, **4**, 22。

最後，剩下的天空輪廓，因為早已經超過建築物右牆，故不必處理。圖 9.9 繪出修正後的天空輪廓。

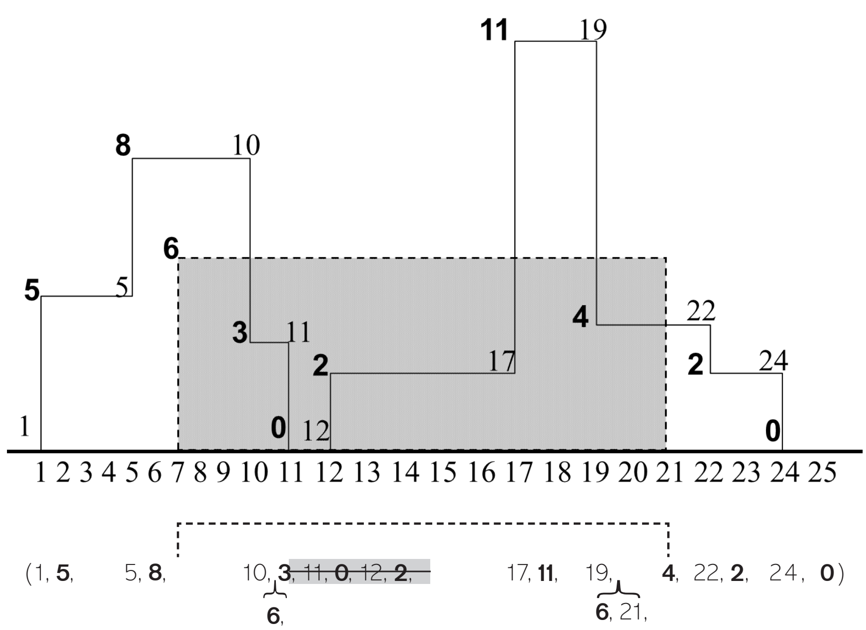


圖 9.8 天空輪廓的修正過程示意圖

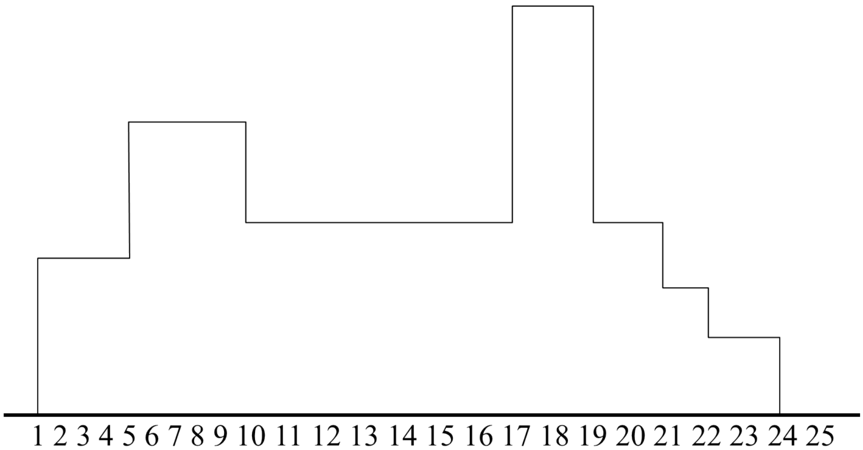


圖 9.9 修正後的天空輪廓 (1, **5**, 5, **8**, 10, **6**, 17, **11**, 19, **6**, 21, **4**, 22, **2**, 24, **0**)

詳細的天空輪廓演算法列於下表 9.4。

表 9.4 天空輪廓演算法

輸入	n 棟建築物：$B_1=(L_1, H_1, R_1)$，$B_2=(L_2, H_2, R_2)$，…，$B_n=(L_n, H_n, R_n)$，其中 L_i 及 R_i 分別是第 i 棟建築物的左右牆(x 軸座標)，而 H_i 是第 i 棟建築物的高度
輸出	天空中的輪廓 $S=(x_1, h_1, x_2, h_2, \cdots, x_z, h_z)$。其中 x_i 是 x 軸座標，h_i 是天空輪廓的高度 $(1≦i≦z)$

步驟	Algorithm skyline()
	{
	/*設定天空輪廓初值使得其涵蓋全部，以方便演算法設計*/
	/*此處 x_{max} 是最大 x 軸座標值*/
	Step1：if L_1=1時，令$S=(L_1, H_1, R_1, 0, x_{max}, \mathbf{0})$
	      else令$S=(1, \mathbf{0}, L_1, H_1, R_1, \mathbf{0}, x_{max}, \mathbf{0})$;
	
	/*依序將 $B_i=(L_i, H_i, R_i)$ 加入於目前的天空輪廓 S 中*/
	Step2：for i=2 到 n do
	      {
	      /*自左至右掃描 $S=(x_1, h_1, x_2, h_2, \cdots, x_z, h_z)$ 並找到左牆的位置*/
	      自左至右讀取的天空輪廓 S 中一個片段 x_j, $\boldsymbol{h}_j$, x_{j+1} 使得 $x_j≦L_i<x_{j+1}$;
	      /*左牆產生新的輪廓*/
	      if $\boldsymbol{h}_j<\boldsymbol{H}_i$ 則在天空輪廓 S 的 $\boldsymbol{h}_j$ 後，插入一段新輪廓的"L_i, $\boldsymbol{H}_i$";
	      /*修正中間的部分*/
	      讀取的天空輪廓 S 連續多個片段到 $x_k≦R_i<x_{k+1}$ do
	next

```
        {
        令其中的天空輪廓為(x_{j+1}, h_{j+1}, x_{j+2}, h_{j+2}, …, x_k);
        若其中 h_{j+1}, h_{j+2}, …, h_{k-1} 有小於 H_i 者，皆調整為 H_i;
        /*右牆產生新的輪廓*/
        if h_k<H_i 則在天空輪廓 S 的 x_k 後，插入一段新輪廓的" H_i, R_i";
        將連續同樣高度的多條線段，合併成同一個線段;
        }
      }
Step3：輸出 S;
}
```

天空輪廓演算法需要 $O(n^2)$時間複雜度，來掃描天空輪廓並作適當的調整，此處的 n 是輸入建築物的個數。

9.4 凸殼

下一個問題是***凸殼***(convex hulls)，一個十分有名的計算幾何問題。

表 9.5 凸殼

問題	一位製作皮件工廠的老闆，希望將所有每張皮革上的瑕疵點全數剪除。但是為了節省皮料，希望被剪除的整塊面積(為凸多邊形)愈小愈好。請設計一個演算法，來協助老闆進行此剪裁工作
輸入	平面上的 n 個點 $P=\{p_1, p_2, \cdots, p_n\}$ P={(1, 5), (2, 12), (3, 8), (4, 4), (5, 6), (6, 11), (7, 1), (8, 10), (10, 7), (11, 13), (12, 3), (14, 9)}
輸出	一個包含所有輸入點的最小凸多邊形(convex polygon)，稱作凸殼(convex hull) C。並按照逆時鐘方向，將凸殼 C 上的點依序輸出 C: (7, 1)→(12, 3)→(14, 9)→(11, 13)→(2, 12)→(1, 5)

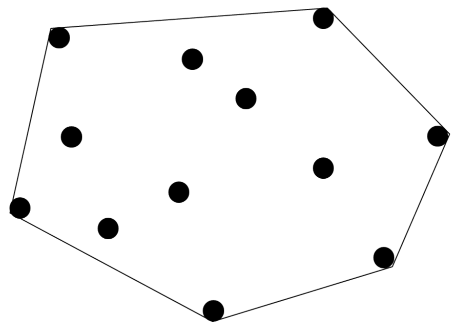


圖 9.10　平面上的點及其凸殼

平面上的 n 個點和其凸殼的關係，可以利用下面的比喻來解釋。彷彿是給了 n 個鐵釘，並且將它們釘在黑板上。接著找一條大橡皮筋，拉大到足以包覆所有的鐵釘，將橡皮筋放鬆後，橡皮筋會被最外圍的鐵釘圈住，所得到的多邊形就是凸殼。

「落在凸殼上的點，有什麼特性？」

「好像這些點都位於最外圍的地方。」

「為什麼凸殼上的點，都在最外圍呢？」

「凸殼需要包含所有輸入點，因此凸殼被撐到最外圍。」

「有些點也在蠻外面的地方，為何不在凸殼上？」

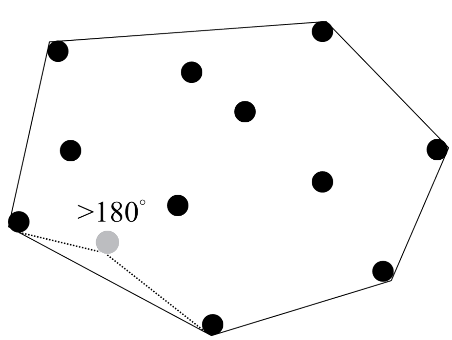


圖 9.11　淺色的圓點(左下方)為何不在凸殼上?

「因為會產生凹多邊形的關係吧！」

「怎樣判斷出現了凹多邊形？」

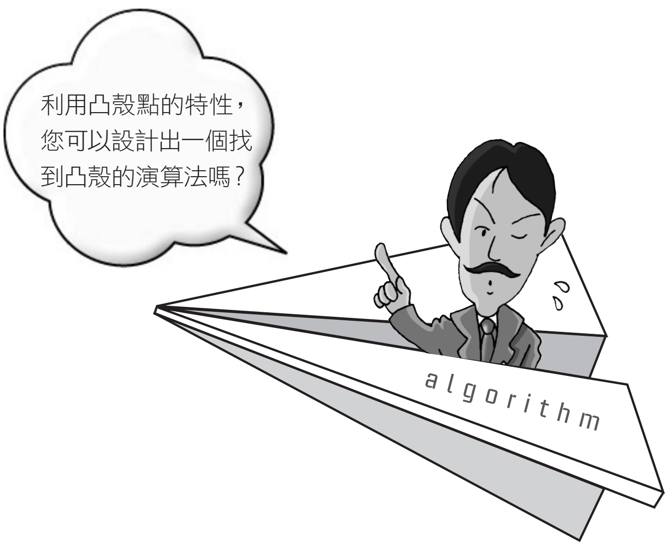
「看角度。自內部看來，有一個角度大於 180 就是凹多邊形。」

「反之，什麼是凸多邊形？」

「凸多邊形內的任兩點所形成的線段，需完全落入此多邊形中。」

「從角度上看，在凸殼上的點有何特性？」

「自外面看來，所有的角度(即任意相鄰 3 個點所形成的角度)都需小於 180 度。」



最直接的凸殼演算法是：先利用任意的三個點建立一個凸殼後，每加入一個新點，就調整成新的凸殼，直到所有點被加入且調整完畢。此類演算法包括天空輪廓演算法(表 9.4)，可視為***歸納設計法***(design by induction)，是一種常見的演算法設計策略。

若將上述的方法再稍微改變一下處理點的順序，就會成為一個知名的凸殼演算法，稱為***格雷漢掃描***(Graham's scan)。格雷漢掃描首先挑出一個最低的一點 p_1(若這樣的點有多個，則選擇其中最右邊的點)，計算此點和其他每一點連成直線的角度，並利用此角度將所有剩下的點排成逆時鐘的順序 $p_2, p_3, \cdots, p_n$ (圖 9.12)，以利後續處理。

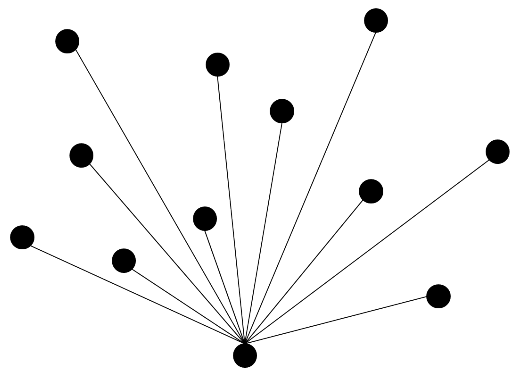
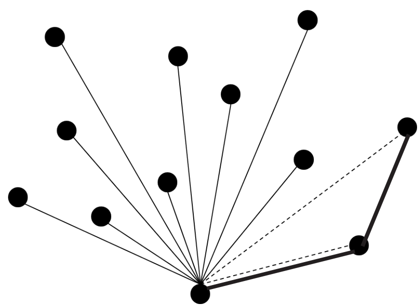


圖 9.12 平面上的每一點與最低點所形成的直線

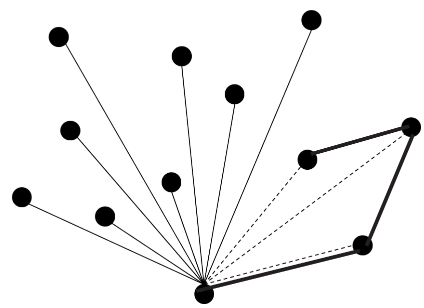
假設處理前面 i-1 個點 $p_1, p_2, \cdots, p_{i-1}$ 後，所得到的凸殼為 $c_1 \rightarrow c_2 \rightarrow \cdots \rightarrow c_{k-1} \rightarrow c_k$。

當格雷漢掃描處理下一個點 p_i 時，將會做以下的修正來找到新的凸殼。首先，計算 p_i 和最近凸殼上的兩點 c_k, c_{k-1} 所形成的角度，即$\angle p_i c_k c_{k-1}$(此角度需自凸殼內部量測)；如果$\angle p_i c_k c_{k-1}$ 小於 180 度，則將 p_i 加入原來的凸殼中，即形成新凸殼 $c_1 \rightarrow c_2 \rightarrow \cdots \rightarrow c_{k-1} \rightarrow c_k \rightarrow c_{k+1} = p_i$，並結束此點的處理。反之，如果$\angle p_i c_k c_{k-1}$ 大於或等於 180 度，將點 c_k 自凸殼中移出，並繼續對此凸殼作同樣的檢查，直到找到角度小於 180 度(並將 p_i 加入)為止。

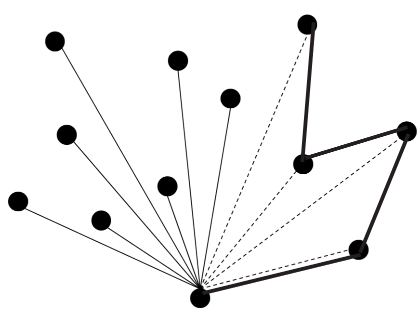
格雷漢掃描重複以上的步驟，直到所有點被處理完成為止。以下為格雷漢掃描的一個範例。



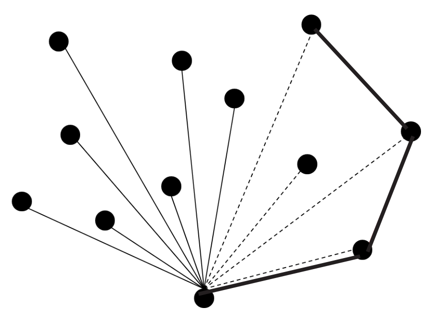
(a) 起始



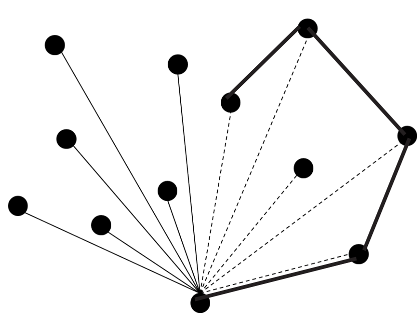
(b) 處理並加入下一點



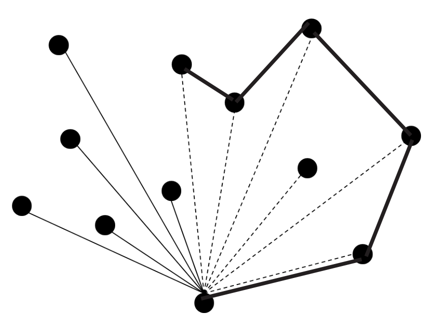
(c) 處理下一點，但發現大於 180 度



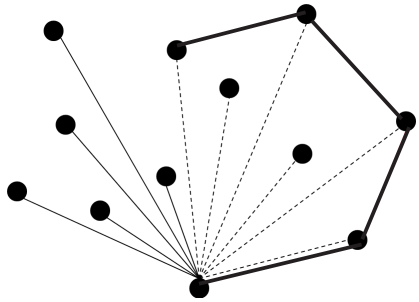
(d) 移除一點



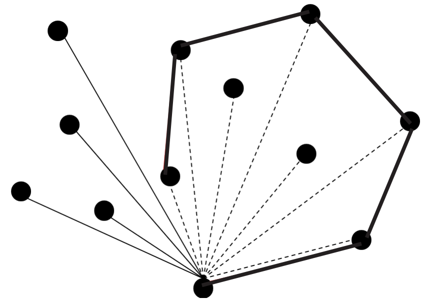
(e) 處理並加入下一點



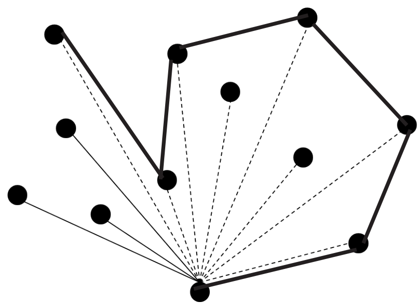
(f) 處理下一點，但發現大於 180 度



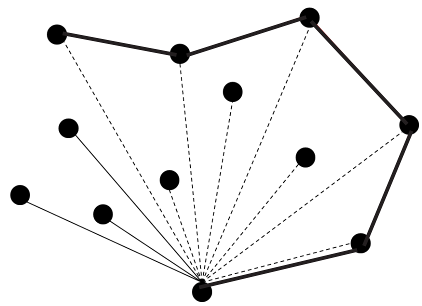
(g) 移除一點



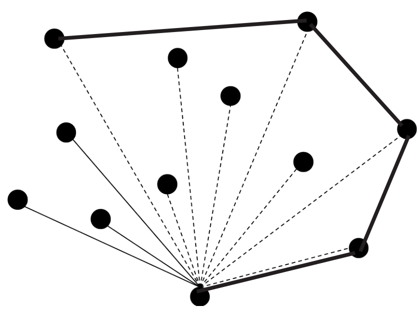
(h) 處理並加入下一點



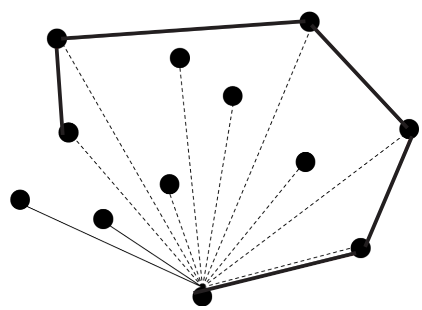
(i) 處理下一點，但發現大於 180 度



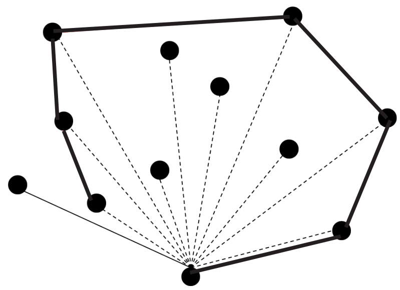
(j) 移除一點後，仍發現大於 180 度



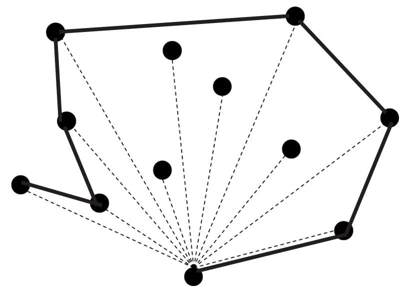
(k) 再移除一點



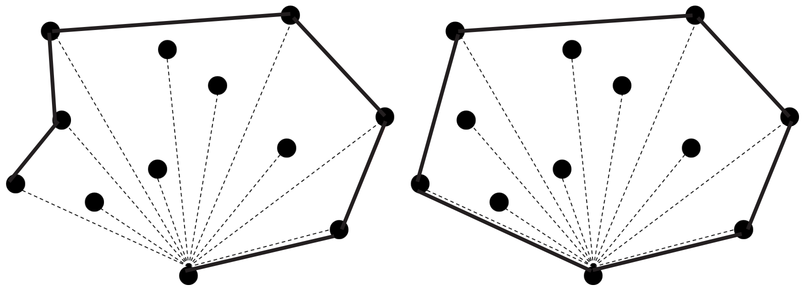
(l) 處理並加入下一點



(m) 處理並加入下一點



(n) 處理下一點，但發現大於 180 度



(o) 移除一點後，仍發現大於 180 度　　(p) 再移除一點，完成

圖 9.13　格雷漢掃描的過程

格雷漢掃描詳細的演算法請見表 9.6。

表 9.6 格雷漢掃描演算法

輸入	平面上的 n 個點 $P=\{p_1, p_2, \cdots, p_n\}$
輸出	按照逆時鐘方向，輸出凸殼上的點 C：$c_1 \to c_2 \to \cdots \to c_{k-1} \to c_k$

步驟	Algorithm Graham's_scan (P)
	{
	Step 1: 令 c_1 為最低的點(當這種點有多個，則選擇其中最右邊的點);
	Step 2: 計算 c_1 和其他每一點連成直線的角度，並利用此角度將其他點排成逆時鐘的順序: p_2, …, p_n;
	Step 3: 令 $c_2=p_2$，$c_3=p_3$，$k=3$; /*設定凸殼初值*/
	Step 4: for i=4 to n do /*將其餘 n-3 點一一處理*/
	{
	while $\angle p_i c_k c_{k-1} \geqq 180$ 度 do {$k=k-1$;}; /*刪除一點*/
	$k=k+1$;
	$c_k=p_i$; /*增加一點於凸殼中*/
	}
	Step 5: 輸出凸殼 $c_1 \to c_2 \to \cdots \to c_{k-1} \to c_k$;
	}

格雷漢掃描需要 $O(n \log n)$時間來執行(這裡 n 代表點的個數)，因為除了 Step 2 需要 $O(n \log n)$時間進行排序(sorting)外，其餘皆可於 $O(n)$時間內完成。

9.5 最近配對

最後的例子是最近配對(closest pair)問題。

表 9.7 最近配對問題

問題	海上航行時，若兩艘船不慎行駛太近，容易發生碰撞危險。假設我們可以及時收集到所有海上船隻的定位資訊。請設計一個演算法，快速找出最靠近的兩艘船，以方便即時提醒相關船員，以避免發生災害
輸入	平面上的 n 個點 $P=\{p_1, p_2, \cdots, p_n\}$ P={(1, 7), (2, 4), (5, 6), (6, 1), (7, 10), (8, 4), (9, 3), (9, 8), (12, 9), (13, 2), (14, 5), (14, 9)}
輸出	最近配對及其距離。當兩點 $p_1=(x_1, y_1)$, $p_2=(x_2, y_2)$, 這裡的距離是指歐幾里得距離(Euclidean distance)，即 $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$ 最近配對(8, 4), (9, 3)，距離 $\sqrt{2}$

自 n 個點中找到最近的兩個點，最簡單的方法是***暴力法***：比較所有的兩兩配對，並自其中找到最小距離的配對。此法需將所有兩兩配對(共 $\binom{n}{2}=\frac{n\times(n-1)}{2}$ 種配對)都考慮過，因此其時間複雜度為 $O(n^2)$。若使用各個擊破法，有機會設計出更快的演算法。

「如何設計出有效率的各個擊破演算法？」

「這個嘛...」

「什麼是各個擊破？」

「將一個問題切割成一些小問題，並且遞迴地解決後，再利用這些小問題的解，合併成原來大問題的解。」

「最近配對問題可以被切割成兩個小問題嗎？」

「應該可以。只要根據 x 軸座標將所有點排序後，自中間平分即可。」

「如何利用兩個小問題的解，合併成原來大問題的解？」

「分別找到一個最近配對後，從兩者中再選出更近者。」

「有沒有其他配對被遺漏了？」

「好像沒有…除非落在中間。除非，此配對的兩點正好落在切割線的不同端。」

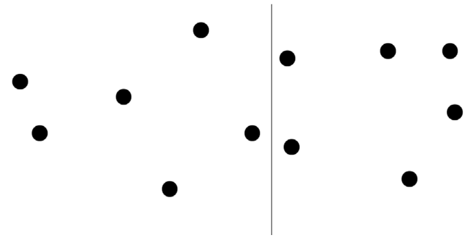


圖 9.14　利用各個擊破法尋找最近配對時，需考慮兩點落在切割線不同端的配對

「這樣的點配對有多少對？」

「好像有很多可能。」

「每一種可能的配對都要考慮嗎？」

「這個嘛...太遠的配對應該不需要考慮才對。」

「為什麼太遠的配對不需要考慮？」

「因為要找的是最近的配對。」

「多遠的配對不需要考慮？」

「只要距離超過兩邊選出的最小者，皆可不考慮，因為不會產生更近的配對。」

「跨越兩邊且距離小的配對個數多嗎？會不會影響到合併時的速度？」

「應該不會太多才對。因為這些點，若落在同一邊，其間隔仍需要大於一定距離，因此其分佈不會太密集。應該不會增加太多此各個擊破演算法的執行時間。」

設計一個各個擊破的最近配對演算法，其關鍵就在於如何合併及所需要的執行時間。根據以上的討論，此合併需檢查，是不是有落在切割線兩端且更靠近的配對。

當在左邊找到的最小配對的距離為 d_1，而在右邊找到的最小配對的距離為 d_2，則跨越兩邊的最小配對(如果存在的話)的距離，不可超過 $d=\min\{d_1, d_2\}$。

令此最小配對的其中一點 p_L 位於左邊，另一點 p_R 位於右邊。則必可找到兩個相鄰且分處兩邊的 $d\times d$ 方格包含此最小配對。否則，p_L 和 p_R 的距離會超過 d(圖 9.15)。

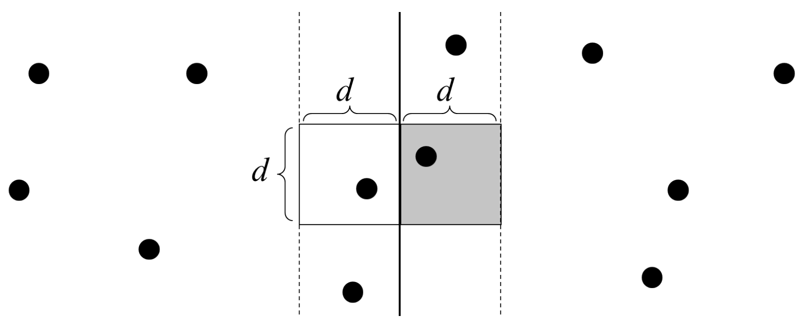


圖 9.15　跨越兩邊的最近配對的距離不可超過 $d=\min\{d_1, d_2\}$

落在同一方格中的點，必須距離等於或超過 d，否則在該邊所找到的最近配對是錯的。所以，在左邊同一方格中的點，最多只有四個，且被逼迫退到四個角上。另外在右邊相鄰的方格上，也有同樣的狀況；即最多只有四個點，且被逼迫退到四個角上(圖 9.16)。

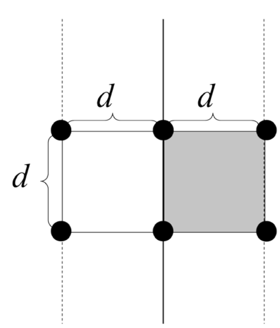


圖 9.16　跨越兩邊相鄰的 $d\times d$ 方格內的最近配對最多只有六個點

此各個擊破的最近配對演算法，在合併時，在以分割線為中線的寬長 $2d$ 帶狀範圍內，找尋可能存在的最近配對。找尋的方法是「在此範圍中的每一點 p，檢查是不是，在一個方格的距離內，有更靠近(距離比 d 小)的配對。如果有，則將此最近配對及距離回傳。」

詳細的最近配對演算法如表 9.8 所示。

表 9.8 最近配對演算法

輸入	平面上的 n 個點 $P=\{p_1, p_2, \cdots, p_n\}$
輸出	最近配對及其距離

步驟	Algorithm closest_pair (P)
	{
	/*排列平面上的點(前置作業)*/
	Step 1:將所有的 n 個點 P 按照 x 座標值排序，並存入陣列 P_x;將所有的 n 個點 P 按照 y 座標值排序，並存入陣列 P_y;
	
	/*divide step*/
	Step 2:將 P_x 所有點，切割成左右兩個均等且按照 x 座標值排序的陣列 L_x 及 R_x;將 P_y 切割成按照 y 座標值排序的陣列 L_y 及 R_y，使得 $L_x(R_x)$的點落入 $L_y(R_y)$中
	
	/*conquer step*/
	Step 3:遞迴地計算左、右兩集合的最近配對;令左邊找到的最小配對的距離為 d_1，令在右邊找到的最小配對的距離為 d_2;
	
	/*merge step*/
	Step 4:令 d=min{d_1, d_2}，即找出 d_1 和 d_2 的最小值
	Step 5:自 P_y 中刪除寬長 $2d$ 帶狀範圍(以分割線為中線)外的點後存入 P_y,;
	Step 6:按照 y 軸座標值順序掃描 P_y,中每一點，並計算此點和隨後其 y 軸座值之差小於 d 之點的距離，紀錄掃描過程中發現的最近配對:
	Step 7:自 Step 3 及 Step 6 所找到的配對中，選擇出最近配對(及其距離)回傳;
	}

上述演算法的 Step 2 可以利用原來已經在 Step 1 排列好的陣列 P_x 及 P_y，在 $O(n)$ 時間內完成。Step 6 需為 $P_{y'}$ 中的每一點，在 $P_{y'}$ 依照此點的 y 軸座標值增減 d 之範圍內，找尋可能的最近配對。還好因為這樣的點是有限的(如圖 9.16 所示)，所以 Step 6 可在 $O(n)$ 時間內完成。因此 Step 2 到 Step 7 的時間複雜度，可用此數學式表示：$T(n)=2T(n/2)+O(n)$ (此處 n 代表平面點的個數)。解開此數學式，可知 $T(n)$需要 $O(n \log n)$ 時間來執行。最後，整個演算法需要 $T(n)+O(n \log n)$(即 Step 1 的排序時間)$=O(n \log n)$時間來完成。

9.6 計算幾何的技巧

歸納法(induction)和各個擊破法(divide and conquer)都是計算幾何上常見的技巧。例如，本章中的天空輪廓演算法及格雷漢掃描演算法利用了歸納法，而最近配對演算法則利用各個擊破法。天空輪廓演算法，更利用由左至右掃描整個天空輪廓並進行高度調整的方法，也可視為一種重要的幾何演算法技巧，稱為***線掃描***(line sweep)。以下，列出一些其他常見的計算幾何問題：

1. **平行線段垂直線段之交點**(intersections of horizontal and vertical line segments)：輸入 n 條平行線段及 m 條垂直線段，找出所有的交點(intersection)。

2. **沃羅諾伊圖**(Voronoi diagram)：輸入 n 個平面上的點(稱為沃羅諾伊點)，請將平面切割成幾個區域，使得每一個區域，包含所有靠近其中一個沃羅諾伊點的平面點(如圖 9.17)。

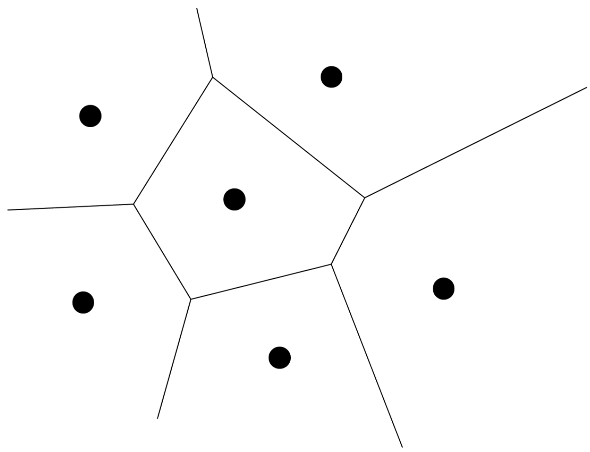
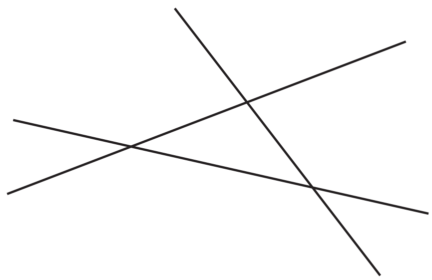


圖 9.17 沃羅諾伊圖

3. **最佳多邊形三角切割**(optimal polygon triangulation)：將一個多邊形切成多個三角形(切點須在頂點上且切割線不可相交)，使其所需的切割線段長度之總和為最小。

學習評量

1. **平面上的線**：一塊披薩被連續直切數次之後，最多會得到幾片呢？用數學的用語描述則是，一個平面可以被 n 條直線最多分割成幾個區域？當 n=1 時，可分割成 2 個區域；當 n=2 時，最多分割成 4 個區域；當 n=3 時，最多分割成 7 個區域(如下圖)。

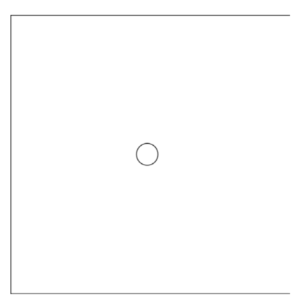


請寫一個程式計算當輸入 n 時, 計算最多的分割區域，並將此數輸出。

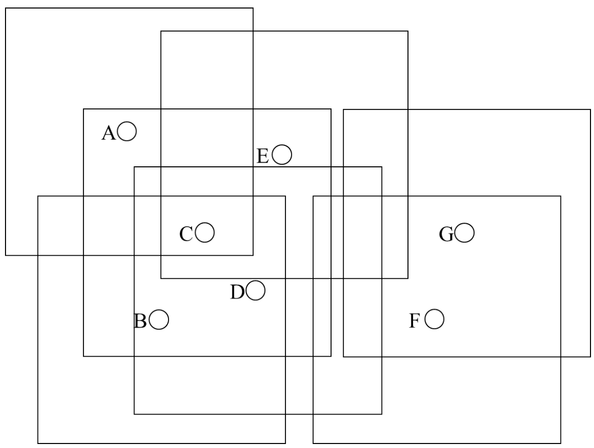
輸入
2
3
輸出
4
7

2. **決定冗餘感測器**：另一個無線感測網路的基本問題是，如何佈建感測網路，使得每一個地方至少被一個感測器所監控。倘若感測器沒有均勻分佈，則會導致若干地方未被監控。為了緩解此問題，可移動一些冗餘感測器，到未被監控的地方。

假設每一個感測器的位置可利用全球定位系統(GPS)取得。每一個感測器的通訊半徑(等同於其感測半徑)都一樣大。為了簡化計算，我們假設每一個感測器的通訊(感測)範圍是一個正方形，而其中心代表此感測器的位置(如右圖)。



每一個感測器隨時記錄他鄰近感測器的座標。例如在下圖中，感測器 C 可以知道感測器 A、B、D、E 因為 C 落在 A、B、D、E 的正方形中。反之，感測器 F 不是 C 的鄰居，因為 C 沒有落在 F 的正方形中，因此收不到 F 的位置資訊。在收到鄰近感測器的座標後，感測器 C 發現他自己是冗餘的，因為他的監控區域可以被鄰近感測器 A、B、D、E 完全覆蓋。



相反地，當接收到鄰居 C 和 D 的座標資訊之後，感測器 B 會發現他自己不是冗餘的。

請寫一個程式計算有多少個感測器，在收集鄰近感測器的座標後，會察覺自己是冗餘的。

輸入

```
4                (感測器的個數)
6                (感測器的通訊及監控半徑)
4 4              (以下為感測器的座標)
6 4
8 4
10 4
```

輸出

```
2                (冗餘感測器的個數)
```

3. **隱藏節點問題**：隨意網路(ad hoc networks)是一種隨意連接，並不需要基礎網路建設的無線網路。每一個設備(device)在此網路上的功能如同一個路由器(router)，可以尋找及維護路徑。假設每一個設備的通訊半徑為 R 單位長度。則當設備 A 和 B 的距離小於或等於 R (即$(x_1-x_2)^2+(y_1-y_2)^2 \leqq R^2$，此處 (x_1, y_1) 和(x_2, y_2) 是 A 和 B 的座標)時，設備 A 可以直接和設備 B 通訊。

在隨意網路中，隱藏節點問題(hidden-terminal problem)的出現，是因為兩個設備無法直接通訊，但是卻同時傳送給另一個相同設備。例如，設備 A 不能和設備 B 直接通訊 (即設備 A、B 之間的距離超過 R)，但是設備 A 可以和設備 C 直接通訊， 並且設備 B 可以和設備 C 直接通訊。此三設備就形成隱藏節點集合。又例如，四個設備 A、B、C、D 佈置在平面的 (0, 0)，(0, 1)，(1, 0)，(1, 1)座標上，並假設通訊半徑為 R=1 單位長度。如此，A 可以直接與 B(C)通訊，而且 D 可以直接與 B(C)通訊。但是，A 不能與 D 直接通訊，而且 B 不能與 C 直接通訊。在此平面上的隱藏節點集合共有四組(即{A, B, C}，{A, B, D}，{A, C, D}，和{B, C, D})。隱藏節點集合在隨意網路中嚴重地延長通訊時間，並且降低系統的效能。輸入平面上的 N 個設備，請計算出有多少個不同的隱藏節點集合。

輸入
```
4                    (佈署設備的個數)
1                    (佈署設備的通訊半徑 R)
0 0                  (以下是佈署設備的平面座標)
0 1
1 0
1 1
```

輸出
```
4                    (不同的隱藏節點集合數目)
```