中華民國第四十四屆中小學科學展覽會

作者說明書

高中組數學科

040403

國立臺中第一高級中學

指導老師姓名

黃金火

劉適存

作者姓名

呂文森

王冠穎

陳俊嘉

吳書雨

第四十四屆中小學科學展覽會

作品說明書

科別：數學科

組別：高中組

作品名稱：騎士與主教(方格填滿問題)

關鍵詞：格子點、模式、簡化點圖

編號：

目錄

壹、 摘要：

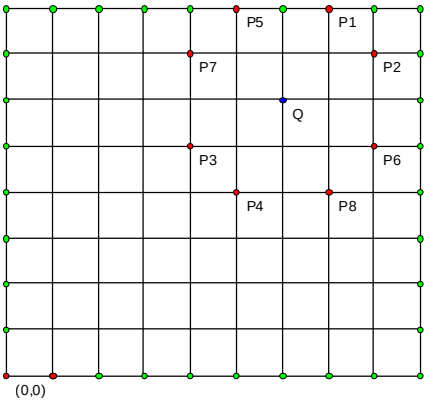
一、首先是我們這次研究所使用到的定義，如下：

(一)座標平面上，若點 P(x，y)符合 x∈N，y∈N 的條件，則稱點 P 爲一" 格子點"。

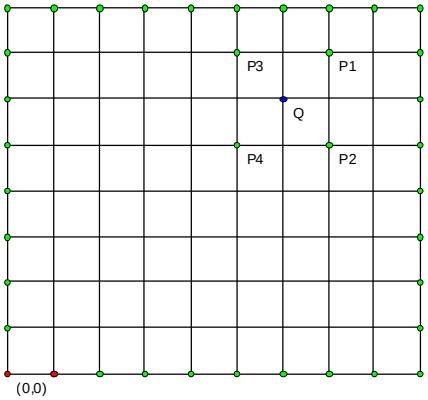
(二)稱 x=0，y=0，x=m，y=n(m，n>0)所圍成的方格中所有方格點(包含在線上的方格點)所成之集合爲"m*n 的方格"。

(三)接著定義一種走法，有點像象棋中"馬"走的形式，叫"騎士"(H)。

例子： 從圖 1 我們知道 Q(6，6)經過一次"騎士"後的落點可能爲 P_1*(7，8)或 P_2*(8，7)或 P_3*(4，5)或 P_4*(5，4)或 P_5*(5，8)或 P_6*(8，5)或 P_7*(4，7)或 P_8*(7，4)。



圖一



圖二

(四)再定義另一種走法，有點像象棋中"士"走的形式，叫"主教"(H)。

例子： 從圖二我們知道 Q(6，6)經過一次"主教"後 Q 的落點可能爲 P_1*(7，7)或 P_2*(5，7)或 P_3*(7，5)或 P_4*(5，5)

(五)接著定義

剩餘的格子數目爲 R(n*m)，其 n*m 指的是 n*m 的方格。

最少剩餘的格子數目爲 r(n*m)，其 n*m 指的是 n*m 的方格。

二、研究問題：

本研究問題，可說是融合了”騎士迷蹤”及”馬步棋”的靈感：從原點開始，在"n*m 的方格"中，依照一次"騎士"，一次"主教"，一次"騎士"，一次"主教".....的方式盡可能填滿 n*m 的方格中的格子點(且不可重複)，並求出"n*m 的方格"中，用這種方法走下去的最少剩餘的格子數目 r(n*m)=？雖然有前人做過類似的問題，不過我們所要達到的目標，是去算所剩的點數，與之前作品的所求是有很大差別的。

貳、 研究動機：

有次弟弟找我玩一種他在雜誌上看到的遊戲“馬步棋”，覺得頗富趣味，便想，要是能夠推廣出來的話，應該更爲有趣、更具挑戰性。

參、 研究目的：

找出 n*m 的方格，以我們定義的走法走完之 r(n*m) = ?

肆、 研究設備與器材：

稿紙(作方格紙用)、計算紙、鉛筆、橡皮擦、C++程式、Excel 程式、Sketchpad 繪圖軟體。

伍、研究過程或方法：

爲了熟悉這個問題，所以我們採取特殊化---先從簡單、數字小的開始處理。

接下來的 m、n、p、q、……、k 等代數，除特別聲明，一律爲正整數。

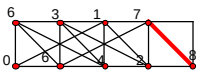
我們在”1*n 的方格”中嘗試使用窮舉法來處理 1*n 的問題。

首先，我們注意到：凡”騎士”的落點必在上面的橫列，而”主教”則必在下面的橫列。又因爲我們限制的走法，使得上面每一個”騎士”均恰對應到下面橫列中的一個”主教”。使得最後在填到 r(1*n)個剩餘(即最小)時，有上列的剩餘點，比下列的多一個(此時最後一步爲”主教”)，或一樣多(此時最後一步爲”騎士”)的情況。我們稱之爲”配對原理”。

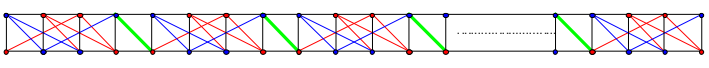
接著我們先討論應該採取怎樣的手法解決 1*n 的方格

第一種-部分前進法：即先將方格分爲許多個小循環，然後連接這些小循環後，用不斷重複的方式去填完。這其中我們得到了能符合電腦數據及最有可能的走法：

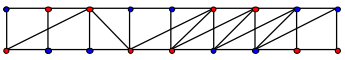
首先可以輕易得出 R(1*1)=3；R(2*1)=3，接著看到 3*1 圖形，(如圖 1-1)

圖 1-1 

它可使 3*1 達到 R(3*1)=0 的理想狀況，且最後停在右上角(3，1)(下一步是主教)，停在右上角代表若是格子不只 3*1 時我們仍能以此走法將前面的點數全部走完，類推下去可以將 n*1 的方格歸類成以四行爲一循環的簡易情形(因爲右邊有三行再加上左方起點一行所以是四行，如圖 1-2)，我們稱這時的 R(1*n)爲 R_1(1*n)。

圖 1-2 

故可得 R((4k-4)*1)=1，R((4k-3)*1)=3，R((4k-2)*1)=3，R((4k-1)*1)=0。

第二種-分段完成法：類似的，我們也把方格分爲許多個小循環，然後走到某個循環(留下些許空缺)，再開始不斷重複的填方格，最後再將其剩下的填滿。但顯而易見的，此種方法易在某處被阻擋，而無法填滿之前留下來的剩餘格子點。例 

第三種-來回填滿法：這裡，我們用一口氣走到底的方式。先留下比第二種方法還多的空缺，再回頭填滿一部分，到左方底端再向右走……重複幾次之後，試圖填滿方格。

第三種方法比較複雜，我們將用類似樹狀圖的方式來討論，而爲避免發生像法二遇到的窘境，我們用第一次佔用格子點最少的走法，以其能留給接下來的幾次路線有充分的空間調整，茲列如下：

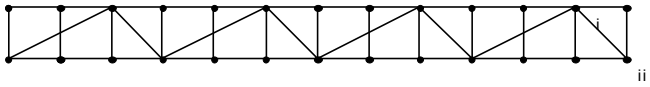
圖(法三) 

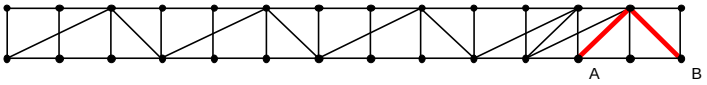
圖 i 

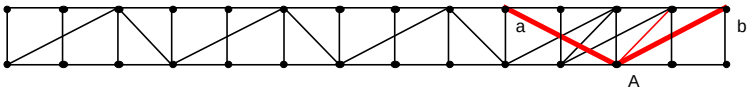
圖 i-A 

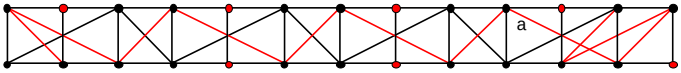
圖 i-A-a 

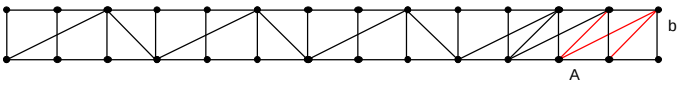
圖 i-A-b 

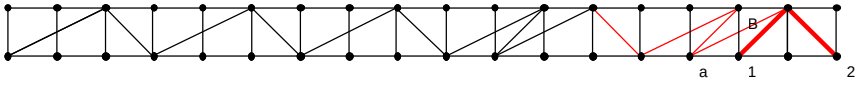
圖 i-B 

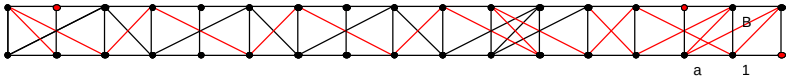
圖 i-B-a-1 

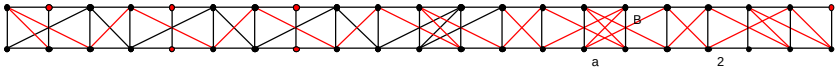
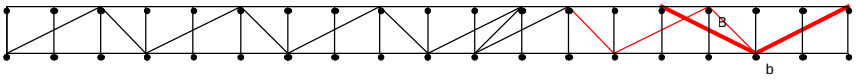
圖 i-B-a-1 

圖 i-B-b 

首先我們先將其分爲在第 n 個騎士往左回跳(圖 i)或在第 n 個主教往左回跳(圖 ii)。在 i 中，我們又可將其分成 A 及 B，而 A 中又可分成 a 和 b，A-a 的狀況又分爲 1 與 2，一樣的，B 中也可分成 a 與 b。

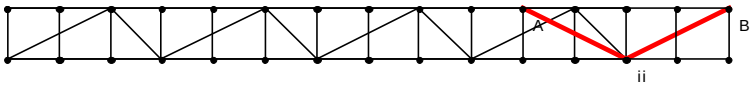
而在經過不斷試驗後我們發現 i 中之 R(1*n)均比 3 大，都不比 R_1(1*n)少。(有時低階的情況會有例外，不過都比那時方格的 R_1(1*n)多)

所以 i 並非最小的適合走法。

ii 中，我們同樣將其分爲 A 及 B，而整理之後亦得 R(1*n)比 3 大，都不比 R_1(1*n)少。(有時也有例外。不過相似的，都比那時方格的 R_1(1*n)多)

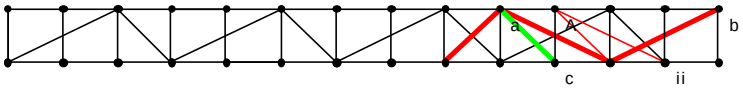
所以 ii 也不是最小的適合走法。

圖 ii



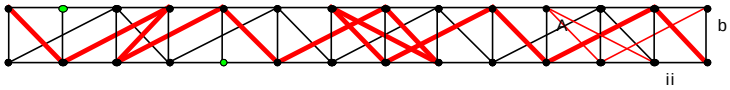
(ii-B 相當於 i 的狀況)

圖 ii-A



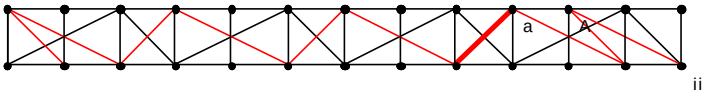
(若走 a 然後向左，此時相當於圖 i-A-a。若走 a 然後向右走 c，則可預見的會左方被擋住。)

圖 ii-A-b



(此時根據配對原理，因爲若要完成紅色粗線段，則必最後一步爲主教，所以右方必有一剩餘點，即 R(1*n)至少爲 3)

圖 ii-A-a



所以 ii 也非最小的適合走法。

∴得證：*r ((4k-4)*1)=1，r((4k-3)*1)=3，r((4k-2)*1)=3，r((4k-1)*1)=0。*

二、n*2：

這裡只證明 n*2 至少的剩餘格是 n-1 (因爲這是我們關心的重點)：總點數是 3*(n+1) ，若以是停在第二列則接下去都只能在二、一列，二、三列跑，故只有一次的機會能在第一列、第三列跑，其餘的都一定得取道第二列-----走最多的點數是 2*(n+1)點-----此時 1、3 列加起來走最多的點數是 2*(n+1)+2 點，可得最少剩餘點數爲 3*(n+1)-2*(n+1)-2= n-1。

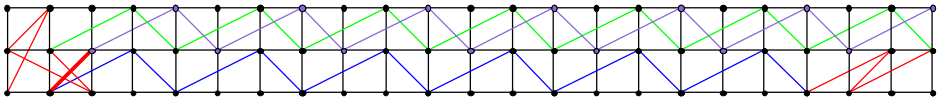
至此，我們推論---r(n*2)=n-1，現在來看看我們所找到符合的走法：

從以上幾張圖我們得到了一些手法：

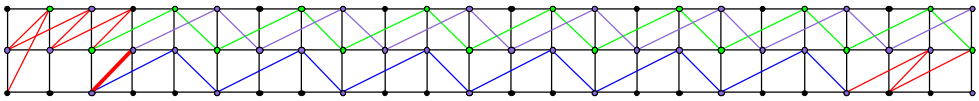
(1) 先將方格歸類爲(3k*4)、(3k*5)、(3k*6)三型

(2) 接著按照以下三張圖形個別處理：

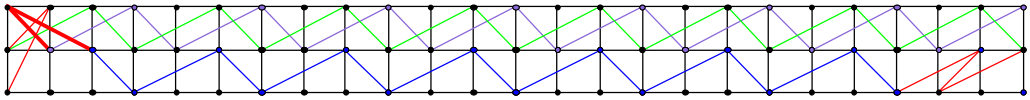
圖(3k*4)



圖(3k*5)



圖(3k*6)



因此我們可知：*r(n*2)=n-1*

前述 n*1、n*2 部分還算容易。因爲限於表格，所以”騎士”的走法，幾乎都侷限於一維。但接下來的證明，由於數字越來越大，所以我們面對的時候，只能採取一種先觀察電腦跑得的前幾項數據，再預估出接下來發展方向的方法，然後試圖尋找一種合理的規則，來解釋預測出的數字。我們經過不斷的嘗試之後，得到了一項特色：”在 m*n 的方格紙中，當 m=奇數 n=奇數時，此種方格紙能被填滿。”若想了解爲何會有這種情況，那就先讓我們看看一些特例：

三、n*3：

3*3 的方格中我們發現一種走法，可符合走完 3*3 方格的要求，即此時最小值(如圖 3-1)。

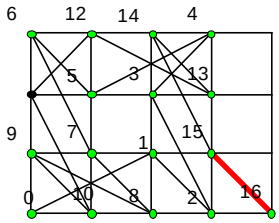
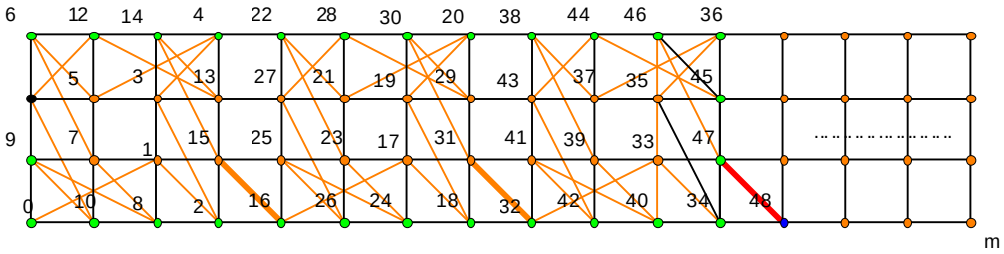


圖 3-1

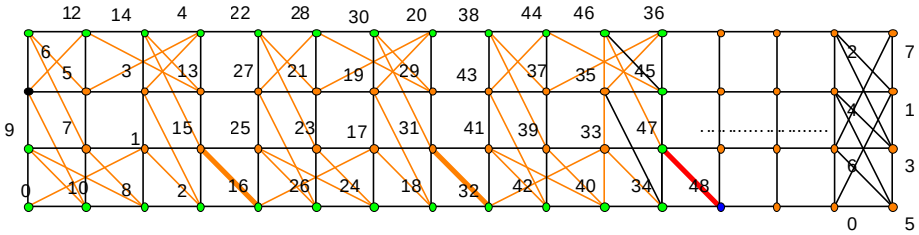
又，此種走法在最後一步時，能將下一步調到另一個起點(即將剩餘格子視作一新的 n*3 繼續往下走。所以我們猜想這應該也是以三個格子(四行)作一循環。而這又顯示出：若以先用填滿 3*3 方格爲目標的策略來走的話，符合 n=4k-1 的 n*3 方格，必能完全被填滿。 (如圖 3-2)

圖 3-2



而在符合 n=4k-3 的 n*3 方格中，若採取先用填滿 3*3 方格爲目標的策略來走的話，由於最後剩下的爲相當於 1*3 的方格，且起點亦爲左下角，所以也能填滿(如圖 3-3，爲最小值的可能走法)。

圖 3-3



四、n*5：

3*5 中我們也發現一種走法，可符合走完 3*5 方格的要求，即此時最小值，且此種走法同樣的在最後一步時，能將下一步調到另一個起點(即將剩餘格子點集合視作一新的”n*5 的方格”繼續走下去。(如圖 4-1))。

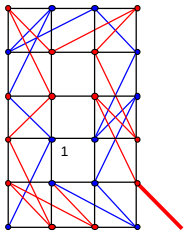


圖 4-1

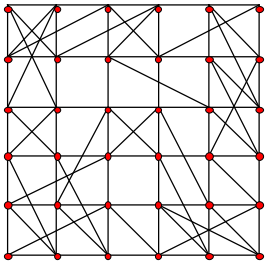
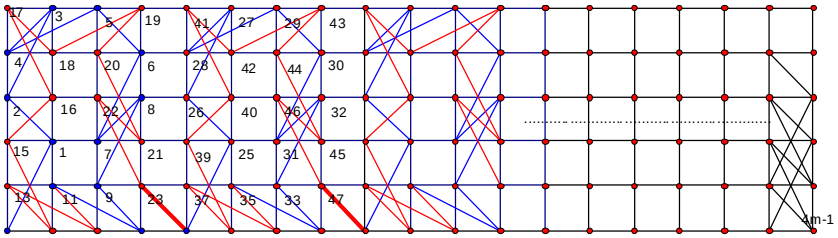


圖 4-2

故，符合 n=4k-1 的 n*5 方格，必能完全被填滿。

而在符合 n=4k-3 的 n*5 方格中，若採取先用填滿 3*5 方格爲目標的策略來走的話，反而就變得沒那麼順利，因爲最後相當於在走 1*5 的方格



，但從之前 1*n 的證明，我們知道當 1*5 時，r(1*5)=3。但我們爲求謹慎，仍繼續找尋更小的 R(n*5)。所以我們退而求其次，轉往空下 6 行---即以填滿 5*5 的方格爲目標努力。意即若我們能填滿 5*5，那符合(n-5)/3 爲整數的方格，也能被填滿。

果不其然，圖 4-2 中顯示 5*5 真的也能填滿，故符合 n=4k+1 的 n*5 方格亦可被填滿。

五、n*7：

而 3*7 中我們也幸運的(其實是花了許多時日，及累積了失敗經驗所得的)發現一種走法，可符合走完 3*7 方格的要求，即最小值(如圖 5-1) 。

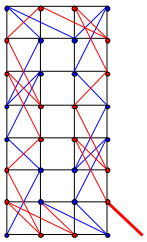


圖 5-1

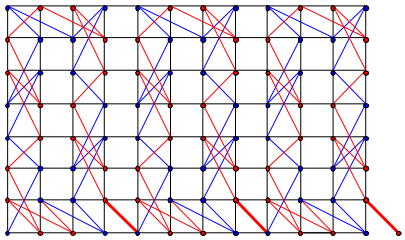


圖 5-2

而此種走法在最後一步時，也能將下一步調到另一個起點(即將剩餘格子點集合視作一新的 n*7 方格繼續往下走。所以我們猜想這應該也是以四行爲一單位作循環。這也暗示著：若以”先循環的填滿 3*7 方格”爲策略來走的話，符合 n=4k-1 的 n*7 方格，亦能完全填滿。(如圖 5-2)

而在符合 (n+3)/4 爲整數的 n*3 方格中，若採取先用填滿 3* 7 方格爲目標的策略來走的話，由於最後剩下的爲相當於 1*7 的方格，且起點亦爲左下角，所以也能填滿(如圖 5-3)，也爲最小值的可能走法。

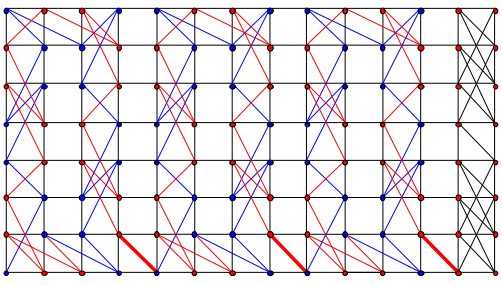


圖 5-3

看到這裡，相信各位獨到的眼光也發現了某種規律，沒錯，隱約中這些走法似乎可以歸納出些”*模式*”：

六、模式：

在模式中，我們先決定這些可填滿情況中填滿時的路徑：

藉 3*(2k-1)及 7*(2k-1)的證明中，我們發現：凡 (4n-1)* (2k-1) 均可先把這些方格紙劃分，從 7*3 的特例中，我們不難發現，在填藍色步驟的時候(如圖 6-1)

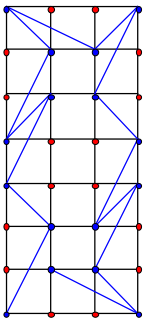


圖 6-1

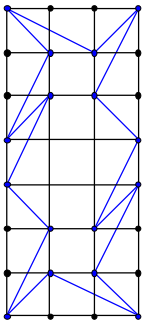


圖 6-1-1

，藍色步數的順序恰是沿某種"模式"(我們稱之爲<***模式一***>)，繞滿第一次途徑的格子。再看紅色步數，如圖 6-2

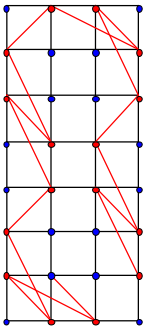


圖 6-2

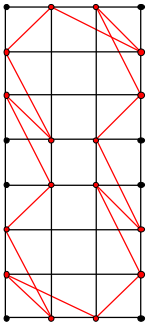


圖 6-2

，紅色部分的填法，和藍色的填法一樣，也有某種程度上的規則(我們稱之爲<***模式二***>)，所以只須將模式一的終點回到模式二的起點，便可再依樣畫葫蘆的填滿模式二，而我們把模式一和模式二合稱爲<總模式>。

接著只要再證明 (4n-1)*(2k-1) 部分均可將終點回到起點，那我們一直在執著的規則，便算能圓滿解釋(4n-1)*奇數的剩餘數問題。

額外訊息：在圖 6-1-1、6-2-1 中，我們注意到模式(一)、(二)中，若從任一點開始，其實都能填滿模式(一)或(二)。而這是否也暗示著：假使模式(一)及(二)能完整連接，那起點不在左下角時，方格亦能填滿呢？

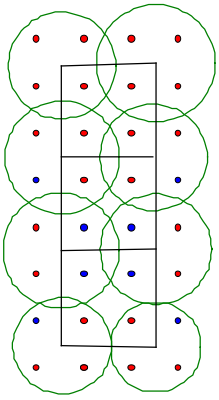
爲了避免問題在推廣時變得過於複雜，我們以一種較爲簡化、巨觀的方式---"簡化點圖"來看。

七、簡化點圖：

簡化點圖中，我們開始來決定路徑的方向，及處理模式一、二之間的連接方式：

仔細看 7*3 的圖中建構出來的”總模式”，不難看出順序大方向上是從左邊向上走，到底向右…向下…向左走回原點，而”主教”走法爲格子的對角線，所以我們想到將四個格子簡化分爲一點(見下頁圖 7-1)。而又每次主教走過簡化點中四個格子點的中的兩點，故我們重新佈置 7*3 的方格成兩張簡化圖(因爲有兩個模式)，可得另一張圖(如圖 7-2)。

圖 7-1　　圖 7-2



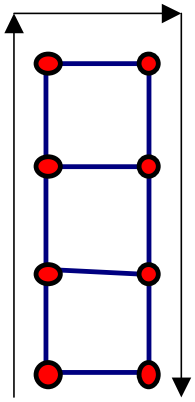
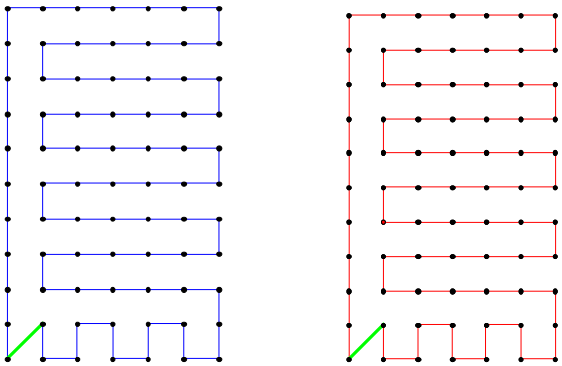


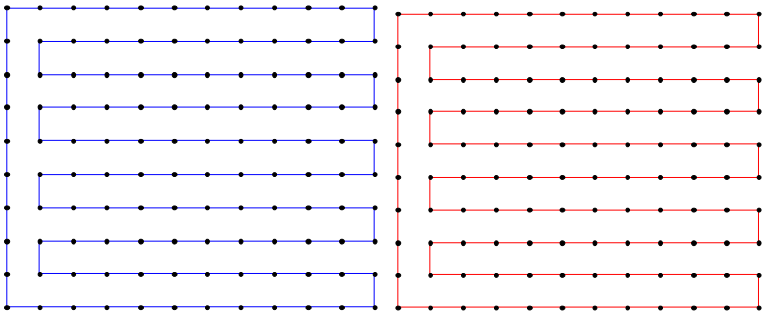
圖 7-2-1 及圖 7-2-2 中我們可以看到當兩邊均爲奇數或偶數個簡化點時模式(一)及(二)的走法。而顯而易見地，當一邊爲奇數個簡化點，一邊爲偶數個簡化點時，則適用於兩種走法。

圖 7-2-1　兩邊均爲奇數



(此時會出現的亮綠色部分是轉折的手法，稍後會提到。)

圖 7-2-2 兩邊均爲偶數



接著我們選 15*13 做例子：(如圖 7-3)

首先，我們做出它的兩張簡化點圖(圖 7-3-1、圖 7-3-2)出來

圖 7-3-1

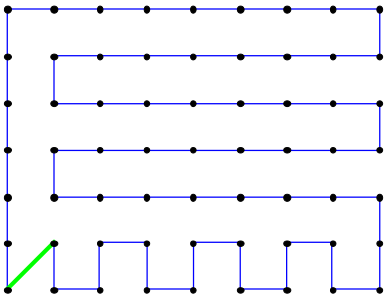
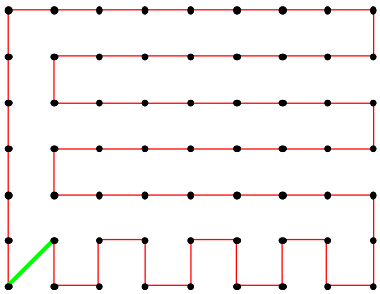


圖 7-3-2



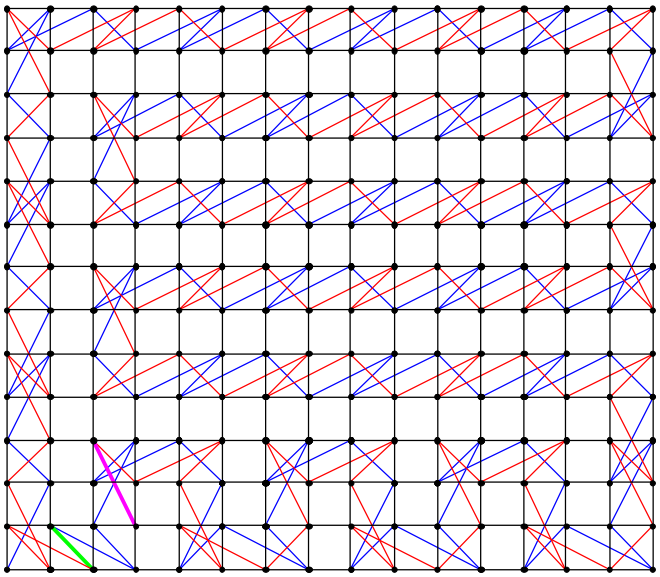


圖 7-3

仔細看亮綠色線段我們得到一種轉折手法可塗滿接合紅&藍部分，至於粉紅色線段我們看得到恰好填完 15*13。所以我們把這種手法應用到(4p-1)*(2k-1)部分，便可完全走滿。

額外推廣：至此，如果我們再整合一下之前的額外資訊，便能得到”只要此方格爲(2p+1)*(2q+1)的方格，其中 p、q 爲自然數，起點可從任一點開始，則必可依一次騎士、一次主教、一次騎士、一次主教…的方式將其填滿。”

八、預測表：

現在我們根據用同學所寫的程式得出的數據，加上之前的證明，可得到表(一)。表中的 m，n 指的是 m*n 的方格紙，相對應格子中的數表示剩下的點數，例如當 m=1，n=3 時剩下的便爲 0 個點，即可完全填滿。

表(一) ：

N \ M	0	1	2	3	4	5	6	7	8	9	10
0	0	1	2	3	4	5	6	7	8	9	10
1	1	3	3	0	1	3	3	0	1	3	3
2	2	3	4	2	3	4	5	6	7	8	9
3	3	0	2	0	2	0	2	0	2	0	2
4	4	1	3	2	5	4	7	6	9	8	11
5	5	3	4	0	4	0	4	0	4	0	4
6	6	3	5	2	7	4	9	6	11	8	13
7	7	0	6	0	6	0	6	0	6	0	6
8	8	1	7	2	9	4	11	6	13	8	15
9	9	3	8	0	8	0	8	0	8	0	8
10	10	3	9	2	11	4	13	6	15	8	17

仔細觀察表(一)，黑色部分是已證明或電腦數據，並確定爲最少剩餘數。我們不難發現除了少數 m，n 爲 1 時發生的特例外，其餘 ∀ (2k+1)*(2p+1) 的方格有 r((2k+1)*(2p+1))=0。(這是之前的證明：所有(2k+1)*(2p+1)的方格剩餘格子點數數均爲 0。)

再看表(一)中 m=4，m=6，m=8 那三行中 4*奇數，6*奇數，8*奇數 那幾行的前幾個的 r(n*m) (藍色部分)是 2，4，6，8---所以我們便直覺判斷接下來應是 10，12，14，…先用紅色數字填上，稍後再來驗證。再重新觀察 m=3 那一行，n=3 那一列---前幾個 r(n*m)是 3，0，2，0，2，0，2，0，… 預測接下來應是 2，0，2，0，…同樣先用紅色數字填上。再重新觀察 m=5 那行，n=5 那列---前幾個 r(n*m)是 5，3，4，0，4，0，4，0，… 預測接下來應是 4，0，4，0，… 先用紅色數字填上，其他奇數部分也如法炮製。

至此，我們僅存(2k)*(2p)的方格尚未做預測。仔細觀察以後我們不難發現：表中的藍色數字部分，在同一行中形成一種等差數列，例：在 m=4 那行中的綠色數字爲 5，7，9，11 所以我們猜想其餘的爲 13，15，17，19，21，23，… 用紅色數字填上，再用相同手法陸續完成附錄中的另一張表(二)<預測最少點剩餘點表>。現在便讓我們使用之前的”模式”嘗試解釋表(二)中紅色數字的部分：

九、

這裡，我們曾嘗試著用遞迴關係式去解。但由於能力、經驗不足，所以我們失敗了。然而，在嘗試使用模式去解之後，我們發現此時之 R(n*m)恰符合電腦所得之 r(n*m)數據。也因此，爲求完整我們提供了一個方法來解釋電腦所得之 r(n*m)數據：

(一)*4n*k*：

1.

若 k=4m-1，R(4n*(4m-1))=4m-2(即 k-1)：

先看 m=1n=1 時 4*3 的特例：我們空下最下面一列(手法一)，將其餘格子作成"模式"。發現經 0~2 步的調整(後手法二，如圖 9-1-1)

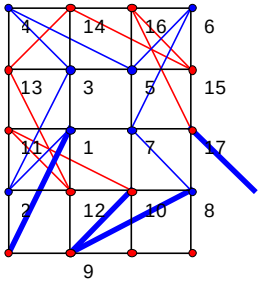


圖 9-1-1

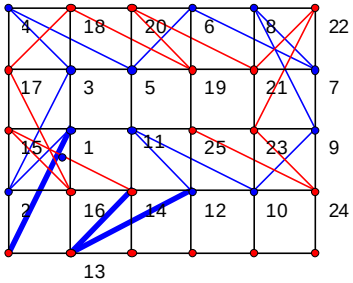


圖 9-1-2

可得到另一個規律：先利用簡化點圖填滿第一次的"模式一"(如圖 9-1-1 分成紅藍兩部分)，而在 7~10 步的調整後(手法三，如圖 9-1-1 藍色粗線段部分)又能走到紅色部分的模式，此時則留下最後一列的 3-1=2 個格子點。

同樣的，推廣後可得簡化點圖中走完(如圖中粉紅色線段所示)停留點的下一步恰可做另一次循環。所以，在模式及簡化點圖的循環下，若 k+1 能被 4 整除，則上方建構出來的模式將能被填滿。

∴此時有 R(4n*(4m-1))=4m-2(即 k-1)。

2.

若 k=4m+1，R(4n*(4m+1))=4m(即 k-1)：

此時亦可用 1.的三種手法，推得 R(4n*(4m+1))=4m (即 k-1)。 (如圖 9-1-2)。

3.

若 k=4m，R(4n*(4m))=4m+4n-3(即 4n+k-3)：

先讓我們來看看 m=1，n=1 時的特例 4*4：類似手法一的做法，不過，這次我們多空下最右方的一行然後再在 17~19 步時(如圖 9-1-3)作調整，便可得多空下的右方那一行餘 4-1=3 個格子點，而最下面一列則剩 4-2=2 個格子點。推廣後可知 R(4m*4n)=4m-1+4n-2=4m+4n-3。

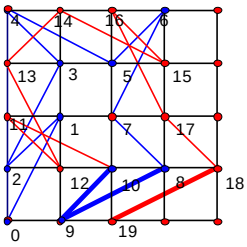


圖 9-1-3

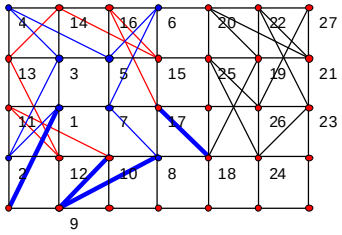


圖 9-1-4

4.

若 k=4m+2，R(4n*(4m+2))=4n+4m-1(即 4n+k-3)：

這次，我們多空下最右方的三行。從之前 4n*2 的證明我們知道此時 R(4n*2)=4n-1 此時留下最下方一列的格數則爲 4m，所以此時可知 R(4m*4n)=4m-1+4n-2=4m+4n-3。(如圖 9-1-4)。

(二)、*(4n+2)*k*：

1.

若 k=4m-1，R((4n+2)*(4m-1))=4m-2(即 k-1)：

先看 m=1n=1 時 6*3 的特例：同樣我們空下最下面一列，將其餘格子作成"模式"。再經手法一、二、三，又能走到第二次的模式，此時則留下最後一列的 3-1=2 個格子點，又推廣後，可得圖 9-2-1 中停留點的下一步恰可做另一次循環。

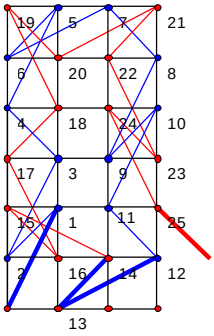


圖 9-2-1

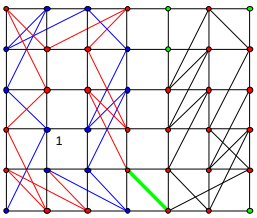


圖 9-2-2

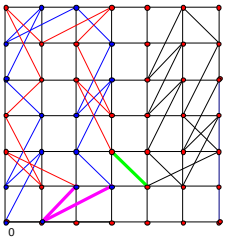


圖 9-2-3

所以，在簡化點圖及總模式一來一回的循環下，若 k=4m-1，則上半部建構出來的模式也能被填滿。此時 R((4n+2)*(4m-1))=4m-2(即 k-1)。

2.

若 k=4m+1，R((4n+2)*(4m+1))=4m(即 k-1)：

顯而易見地，若我們將此方格---(4n+2)* (4m+1) 的方格改成 (4m+1)*(4n+2)，對結果其實是沒有影響的。同理，我們在最右邊留下三行，從圖 9-2-2 中，我們看到停留點的下一步亦可做出另一次循環。從之前 2*n 的證明，我們知道此時 R((4n+2)*(4m+1))=4m(即 k-1)。

3.

若 k=4m+2，R((4n+2)*(4m+2))=4n+4m-1 (即 k-1)：

一樣地，空下最右邊三行，最下方一列後，我們得到一個(4n+1)*(4m-1)的方格，由手法三可得：在剩下右方三行時，最下面一列剩 4n+1 個點。由於此時停留的點，恰可作剩下三行(如圖 9-2-3，不包括最下面一列)的起點。2*n 的經驗告訴我們，r(2*(4m+2-1))=(4m+2-1)-1=4m，所以 R((4n+2)*(4m+2))=4n+4m-1(即(4n+2)+k-3)。

陸、研究結果：

綜上所述我們歸納出結論：n*m 的方格中，在 m,n>1 時

一、若 n/2,m/2 均非整數則無剩餘點數

二、若 n/2,m/2 均爲整數則剩下 m+n-3 個點

三、若 n/2 爲整數，m/2 非整數則剩下 m-1 個點

柒、結論：

目前雖尚未想到這研究的實質應用方向(唯一想到的方法，就是或許這研究能用來檢驗電子元件，或安置其通路，使之達到每個點都能形成通路。)，不過一如魔方陣，我們藉著這研究獲得許多樂趣與做研究的精神。或許這研究能替將來某些做研究的人提供綿薄之力，成就更璀璨的光芒，如此便不負我們所投入的時間了。

捌、參考資料及附錄：

(一) 1.作者：鄭國順，2.書名：趣味數學問題集，3.出版地：新竹市光復路 2 段 460 號，4.出版社：凡異出版社，5.頁數：P.37、P.189。

(二) 1.作者：郭騰元，2.出版年：1996 年，3.期刊名稱：小牛頓第 151 期，4.頁數：P.52~P.55

(三)1.作者：施信瑋、吳哲瑋、江孟烜、王澤瑋，2.來源：第 44 屆全國中小學科展，3.作品名稱：騎士迷蹤，4.網址：http://www.ntsec.gov.tw/activity/race-1/43/pdf/e/040414.pdf

附錄(一)：表(二)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	3	3	0	1	3	3	0	1	3	3	0	1	3	3	0	1	3	3	0	1
2	2	3	4	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
3	3	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2
4	4	1	3	2	5	4	7	6	9	8	11	10	13	12	15	14	17	16	19	18	21
5	5	3	4	0	4	0	4	0	4	0	4	0	4	0	4	0	4	0	4	0	4
6	6	3	5	2	7	4	9	6	11	8	13	10	15	12	17	14	19	16	21	18	23
7	7	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6
8	8	1	7	2	9	4	11	6	13	8	15	10	17	12	19	14	21	16	23	18	25
9	9	3	8	0	8	0	8	0	8	0	8	0	8	0	8	0	8	0	8	0	8
10	10	3	9	2	11	4	13	6	15	8	17	10	19	12	21	14	23	16	25	18	27
11	11	0	10	0	10	0	10	0	10	0	10	0	10	0	10	0	10	0	10	0	10
12	12	1	11	2	13	4	15	6	17	8	19	10	21	12	23	14	25	16	27	18	29
13	13	3	12	0	12	0	12	0	12	0	12	0	12	0	12	0	12	0	12	0	12
14	14	3	13	2	15	4	17	6	19	8	21	10	23	12	25	14	27	16	29	18	31
15	15	0	14	0	14	0	14	0	14	0	14	0	14	0	14	0	14	0	14	0	14
16	16	1	15	2	17	4	19	6	21	8	23	10	25	12	27	14	29	16	31	18	33
17	17	3	16	0	16	0	16	0	16	0	16	0	16	0	16	0	16	0	16	0	16
18	18	3	17	2	19	4	21	6	23	8	25	10	27	12	29	14	31	16	33	18	35
19	19	0	18	0	18	0	18	0	18	0	18	0	18	0	18	0	18	0	18	0	18
20	20	1	19	2	21	4	23	6	25	8	27	10	29	12	31	14	33	16	35	18	37
21	21	3	20	0	20	0	20	0	20	0	20	0	20	0	20	0	20	0	20	0	20
22	22	3	21	2	23	4	25	6	27	8	29	10	31	14	33	14	35	16	37	18	39
23	23	0	22	0	22	0	22	0	22	0	22	0	22	0	22	0	22	0	22	0	22
24	24	1	23	2	25	4	27	6	29	8	31	10	33	12	35	14	37	16	39	18	41
25	25	3	24	0	24	0	24	0	24	0	24	0	24	0	24	0	24	0	24	0	24
26	26	3	25	2	27	4	29	6	31	8	33	10	35	12	37	14	39	16	41	18	43
27	27	0	26	0	26	0	26	0	26	0	26	0	26	0	26	0	26	0	26	0	26
28	28	1	27	2	29	4	31	6	33	8	35	10	37	12	39	14	41	16	43	18	45
29	29	3	28	0	28	0	28	0	28	0	28	0	28	0	28	0	28	0	28	0	28
30	30	1	29	2	31	4	33	6	35	8	37	10	39	12	41	14	43	16	45	18	47
31	31	3	30	0	30	0	30	0	30	0	30	0	30	0	30	0	30	0	30	0	30
32	32	3	31	2	33	4	35	6	37	8	39	10	41	12	43	14	45	16	47	18	49
33	33	0	32	0	32	0	32	0	32	0	32	0	32	0	32	0	32	0	32	0	32
34	34	1	33	2	35	4	37	6	39	8	41	10	43	12	45	14	47	16	49	18	51
35	35	3	34	0	34	0	34	0	34	0	34	0	34	0	34	0	34	0	34	0	34

```
/*
031201
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define DATA(a,b) (*(data+b*(n[0]+1)+a))

struct point{
        int x ;
        int y ;
} *trace, *min_trace ;

int n[2], min ;
char *data ;

int go_1[8][2] = {   { 2, 1},   // 馬
                                                { 2,-1},
                                                { 1, 2},
                                                { 1,-2},
                                                {-2, 1},
                                                {-2,-1},
                                                {-1, 2},
                                                {-1,-2} } ;

int go_2[4][2] = {   { 1, 1},   // 士
                                                { 1,-1},
```

```
                                        {-1, 1},
                                        {-1,-1} } ;

////////////////////////////////////////////////////////////

int run(int step, int x, int y){   // step->已踏步數,    x,y  位置

        int p, q ;
        int i, done = 0    ;

        DATA(x,y) = 1 ;                         // 已走過標訂
        trace[step-1].x = x ;// 追蹤行程
        trace[step-1].y = y ;

        if(step%2 == 1){                // 若 馬
                for(i=0; i<8; i++){
                        p = x+go_1[i][0] ;
                        q = y+go_1[i][1] ;

                        if( p>=0 && p<=n[0]          && q>=0 && q<=n[1] )// 若
在範圍內

                        if( DATA(p,q) == 0){
                                done = run( step+1, p, q) ;
                                if( done == 0)
                                        return 0 ;
                        }
                }
        }
```

```
else{                                                   // 若 士
        for(i=0; i<4; i++){
                p = x+go_2[i][0] ;
                q = y+go_2[i][1] ;

                if( p>=0 && p<=n[0] && q>=0 && q<=n[1] )

                if( DATA(p,q) == 0){
                        done = run( step+1, p, q) ;
                        if( done == 0)
                                return 0 ;
                }
        }
}

DATA(x,y) = 0 ;                 // 去除已走過標訂

if(done==0){
        if( min>(n[0]+1)*(n[1]+1)-step){        // 皆未搜尋到
                min = (n[0]+1)*(n[1]+1)-step ;
                printf("%2d\n", min) ;
                memcpy( min_trace, trace, step*sizeof(struct point)) ;
        }
        if (min == 0)
                return 0 ;
}

return 1 ;
```

```
} // end run()

//////////////////////////////////////////////////////////////////

int main(void){

        int i ;
        char name[15] ;

        printf("輸入棋盤尺寸:\nM N\n") ;
        for(i=0; i<2; i++)                  // input size
                scanf("%d", &n[i]) ;

        printf("\n 剩餘:\n") ;

        min = (n[0]+1)*(n[1]+1) ;

        trace = (struct point*) malloc( min*sizeof(struct point)) ;
        min_trace = (struct point*) malloc        ( min*sizeof(struct point)) ;

        data = (char*) malloc( min *sizeof(char)) ;
        memset( data, 0x00, min*sizeof(char)) ;              //inil

        run( 1, 0, 0) ;

        sprintf( name, "%d_%d_%d_OUTPUT.txt", n[0], n[1], min) ;
        freopen( name, "w", stdout) ;

        printf("棋盤尺寸:    %d * %d\n", n[0], n[1]) ;
```

```
        printf("最少剩餘:    %d\n\n", min) ;
        printf("其中一解:\nx y\n\n") ;

        for(i=0; i<(n[0]+1)*(n[1]+1)-min; i++)
                printf("%d %d\n", min_trace[i].x, min_trace[i].y) ;

        freopen("con", "w", stdout) ;  //  接回螢幕
        printf("\n 最少剩餘:    %d\n\n", min) ;

        free(data) ;
        free(trace) ;
        free(min_trace) ;

        system("PAUSE") ; // end
        return 0 ;
}
```

評 語

040403 高中組數學科 佳作

騎士與主教

1. 本問題若能夠配合動態畫面之敘述，效果會更好。

2. 由第二天的表現，可以看出作者具有靈敏的反映能力。

3. 應該多努力於理論的探討。