台灣二〇〇二年國際科學展覽會

科　　別：數學科

作品名稱：Amazing Fairy Chess-討論多元方形鏈的數量

得獎獎項：數學科佳作

學　　校：國立臺南第一高級中學

作　　者：林逸侖　　施文智

作 者 簡 介

我是林逸侖，是台南一中三年級的學生。很幸運地，在去年參加第 41 屆全國中小學科學展覽時，以“Amazing Fairy Chess ---- 討論多元方形鏈的數量”獲得不錯的成績，而能夠被推薦參加這次的國際科展；我特別要感謝我們的指導教授----許瑞麟教授（任教於成功大學），在他的指導下，我從研究過程中學習到了很多經驗，也磨練了我撰寫報告的技巧，這對未來想走研究路線的我而言是十分珍貴的。

Abstract

Amazing Fairy Chess –
Discussing the amount of polyominoes

In this report, we discussed the amount of polyominoes, the graphs of a set of squares. “Polyominoes” has been brought up in 1960s, and later developed into a series of questions and games, such as a well-known video game – Tetrix, and the game of puzzle blocks. Both are the applications of polyominoes.

Among those questions, the toughest one is the amount of n-polyominoes. To solve this problem, we used a method which transforms the graphs into sequences. By looking into the properties of those sequences, we obtain a set of rules that can be used to determine the quantity of n-polyomines. The rules are implemented into computer codes in C language with proper modifications made to speed up the efficiency of our algorithm. The computational results show that the amount has been successfully calculated.

中文摘要

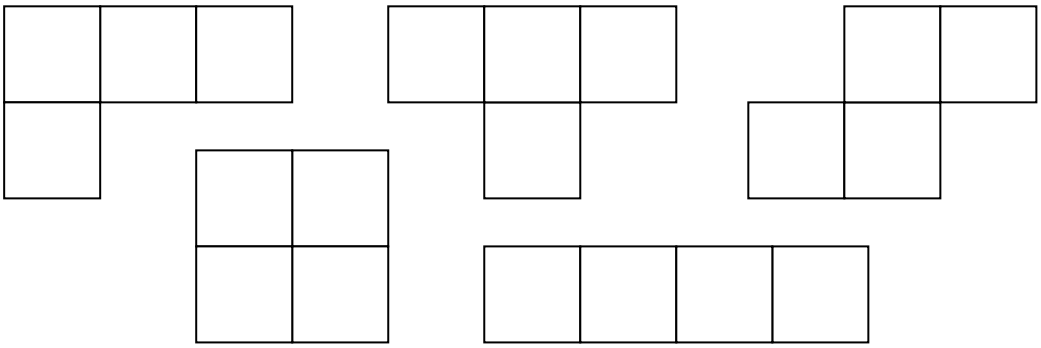
Amazing Fairy Chess –- 討論多元方形鏈的數量

在這篇研究報告中，我們討論的是一種方形集合圖形的數量。”多元方形鏈”約略在 60 年代被提出，衍生出一系列的問題和遊戲，例如熟知的電玩軟體『俄羅斯方塊』，或是『益智積木』的遊戲，都是多元方形鏈的應用。

在這些問題當中，最令人頭痛的難題就是 n 元方形鏈的圖形總數。爲了解決這道難題，我們採用一種轉換方法將圖形轉換成序組，並且給出序組的性質，再據此寫成 C 語言的程式；反覆地修改程式以增進執行效率及速度，最後利用該程式成功地統計出圖形總數。

一、研究背景

Polyominoes，譯作『多元方形鏈』，也有人把它翻譯作『多方塊』，是於 1954 年由 Solomon W. Golomb 發表於 American Mathematical Monthly 裡標題為“Checker Boards And Polyominoes”的論文裡介紹的。Solomon W. Golomb 定義“Polyominoes”為數個相同正方形以邊來相互連結之圖形。由於圖形多變化的特性，也讓它獲得了 Fairy Chess（天使棋）的美名，一般我們常見的『俄羅斯方塊』（圖一）就是 polyominoes 中的 tetrominoes（四元方形鏈）。



（圖一）俄羅斯方塊–四元方形鏈

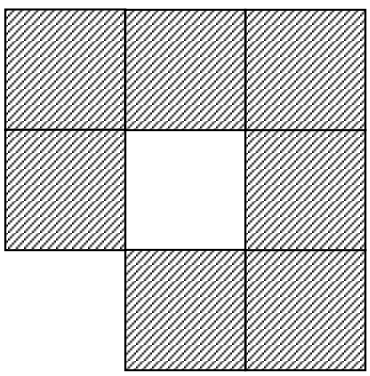
“明顯地，多方塊圖形數目是單位方形數的函數，但到目前為止，沒有一個人能成功的發現 n 元方形鏈對 n 的關係的公式；在計算高階多方塊的數目時，必陷入笨拙且浪費時間的過程。”（此段敘述引自（文獻一）Martin Gardner 編的 Mathematical Puzzles & Diversions）。

經過我們在網際網路上搜尋，找出一些相關文獻，關於這問題的探討可以參閱（文獻五～八）。這些文獻，多半是對某些特殊方形鏈來作探討（如中空、對稱、方塊數特定的圖形）。若要對一般性的方形鏈來計數，做法大致是加一個新方塊到(n-1)元方形鏈上，然後對發展出來的樹狀結構加以計數，再刪去重覆的部分。而在本篇研究報告中，我們嘗試提供另一種計數方法，把方形鏈的形狀『座標化』（我們稱之為序組）。使得我們可以利用計算機來產生並檢驗這些序組，把合法的保留儲存，並正確地計算出 n 元方形鏈的個數。

二、 研究動機

在一九九九年的高雄國際青少年盃數學邀請賽中，有一道題目是：『試問，以六塊正方形各邊之間完全連接成的圖形，共有幾種？（扣除鏡射、翻轉所形成的等價圖形）』在看到這題目的時候，突然回想起曾經在書上看到有關多方塊的概略介紹；後來，我們便利用閒暇之餘，進一步找尋相關資料。結果發現這是一道複雜且困難的問題，所以我們就嚐試用自己的想法進行研究。

在這些多方塊的圖形中，有一部分是中間具有空洞的（如圖二）。為簡化問題，在本研究中，我們不計算這些中空的圖形。



（圖二）中空的七元方形鏈

三、 研究目的

（一） 創造出一套方法，來計算非中空多方塊的數量。

（二） 利用該方法編寫程式，統計非中空多方塊的數量。

四、 研究過程

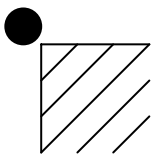
第一節 圖形的序組化

由於我們主要目的是編寫出可計算多方塊圖形數量的程式，然而，從二維的平面圖形不利於程式的編寫，我們希望能找出一種方法，使得這些圖形能夠用更簡潔的資料表示。

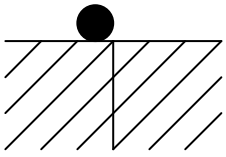
藉由觀察多方塊的圖形（請參考（圖一）四元多方塊之圖形），我們發現到圖形中具有數個以正方形的頂角所構成的交點，這些點分別是由一個、兩個、三個、或四個正方形接成的；於是我們考慮採取使用這些數字的組合來表示多方塊的圖形，易言之，就是從代數的角度解析幾何圖像。

多方塊圖形中，有許多由正方形頂點所構成的交點，這些交點可以分成下列四種情況：

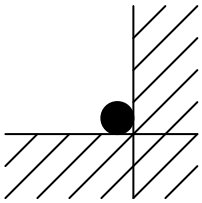
1. 恰由一正方形構成：這種點會出現在圖形的邊界上，恰巧落在凸起處，我們定義這點的數值是 1。



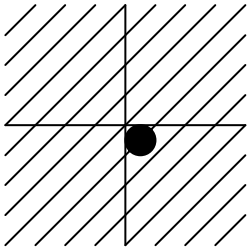
2. 恰由二正方形構成：這種點會出現在圖形的邊界，恰巧落在圖形的邊上，我們定義這點的數值是 2。



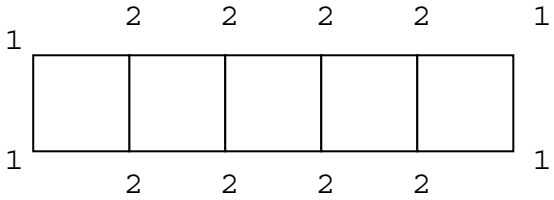
3．恰由三正方形構成：這種點會出現在圖形的邊界上，恰巧落在凹陷處，我們定義這點的數值是 3。



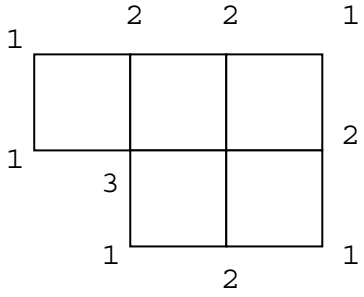
4．恰由四正方形構成：異於上述三者，這種點不會出現在圖形的邊界上，而僅出現於圖形內部，我們給這種點一個名稱：四角點。



以下面(圖三)和(圖四)為例，各交點的數值皆標明在該點旁邊。



（圖三）五元方形鏈之一，四角點數=0



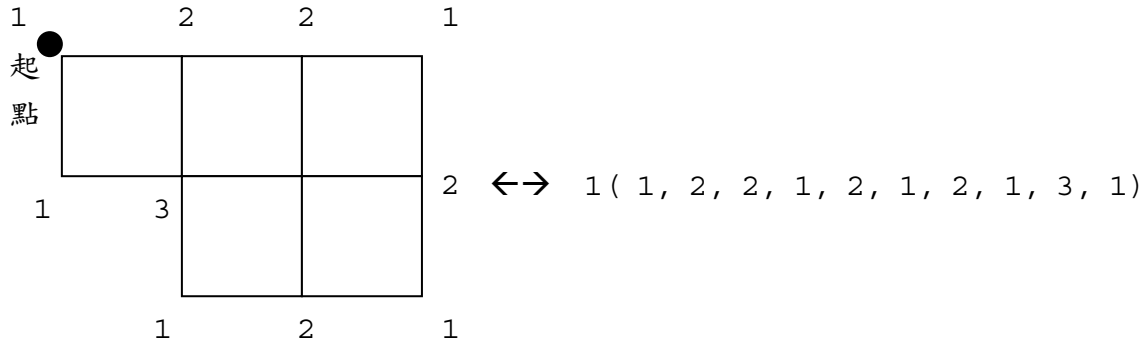
（圖四）五元方形鏈之一，四角點數=1

現在，將各交點數值依順時針的方式排成一個序組，由於每一個圖形的形狀都不一樣，因此對應的序組也不會相同。再用m代表四角點的數量，並在多邊形的邊界上任取一點作為起點，則圖形就可以表示成下列的方式：

$$m(A_1, A_2, A_3, \cdots\cdots A_n)$$

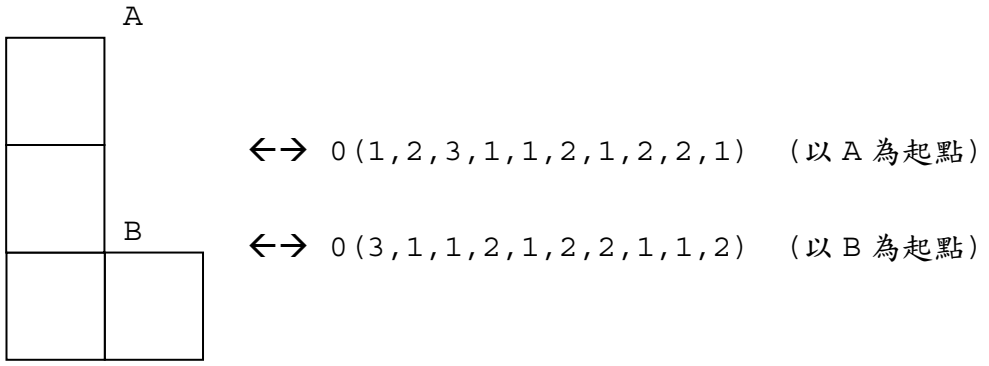
其中A_k代表從起點起第 k 個點的數值。

以五元方形鏈（如圖五）為例：



（圖五）五元方形鏈

明顯地，對於相同的一個序組，只能構成一種圖形。但若選取的“起點”不同，所形成的序組雖不一樣，描述的圖形卻是同一組。例如(圖六)：



（圖六）四元 L 形方形鏈

這兩個序組都可表示四元L形方形鏈的圖形，兩者差異僅僅是循環的起始位置不同而已。就純理論推導時，我們不區別起始點位置不同而造成序組的相異性，但在程式執行時，會將所有合法的序組比對後，將重覆者剔除。

第二節　序組性質的探討

鑒於我們的表示方法是利用序組構成的，為了討論這些序組的性質，需要研究這些數值的範圍及彼此間的關係。一個序組中，除四角點個數 m 之外，首重項數，其次是這些數的總和，因此我們再定義

T 表示序組長度；

S 表示序組中元素總和，

且令

n 代表單位方形的數量。

顯然 n 可以是任意自然數，因此需要討論的數值範圍只剩下 m、T、S。

<<序組中四角點 m 值的推導>>

為了從 n 值來估算 m 值的上限，我們分成下面兩種情形：

Case1 考慮 n 為完全平方數時：

此時當 n 個單位正方形組合成最大可能的正方形時，四角點的數量比其他 n 元方形鏈的四角點多。

<證明>

設 $n=k^2$

若排成 k*k 之正方形時，

$$m=(k-1)^2=k^2-2k+1=n-2k+1$$

反之，若排成 g*h 之長方形時，g<k<h，g,h 屬於自然數，則

$$m=(g-1)(h-1)=gh-g-h+1=n-(g+h)+1$$

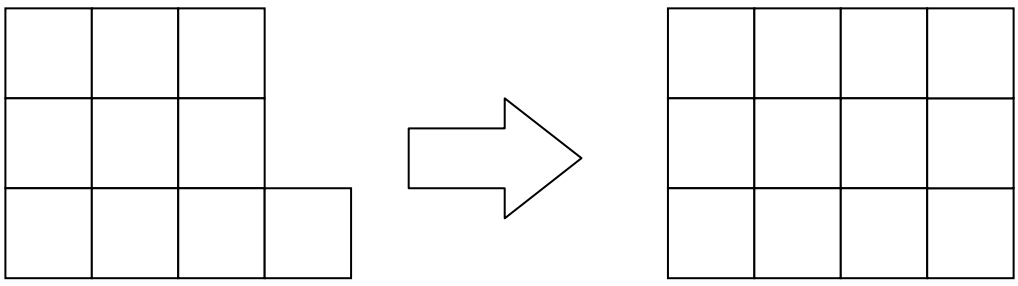
根據算幾不等式 $k=(g*h)^{1/2}<(g+h)/2$

從而 $2k<g+h \Rightarrow (k-1)^2>(g-1)(h-1)$

故得證

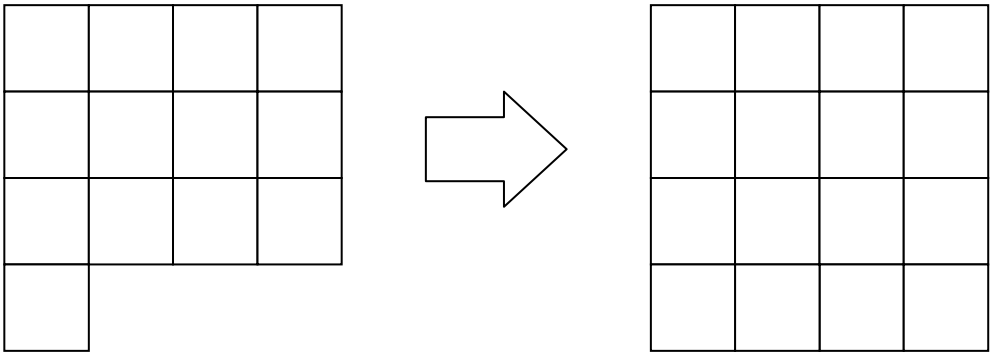
Case2 考慮 n 不為完全平方數時：

假設 n 介在兩個完全平方數 k^2 及 $(k+1)^2$ 之間。先組合出可能產生的最大正方形後，再將其餘的單位正方形附加至原圖任一側，自加入第二個後m值才逐次加 1，一直到該側排滿為止。因此，若 n≦ k*(k+1)時，m 的值會遞增直到排成 k*(k+1)之長方形時。此時 m=(k-1)*k。如(圖七)。



（圖七）當 n 介於 9 和 16 之間，於任一側一次連結單位正方形

若 $k(k+1)<n\leqq(k+1)^2$ 時，首先依上述方式排成 $k(k+1)$ 的長方形，再繼續添加單位正方形到邊長為 $(k+1)$ 的那一側，直到再度形成一個新的大正方形。此時。m 值會遞增到 $(k-1)(k-1)$，如（圖八）。



（圖八）當 n 介在 9 和 16 之間，其中一側排滿後，再從其鄰邊排。

綜合 Case1 & Case2

根據上述，可以歸納出以下公式：

取 $k=[n^{1/2}]$　其中 [] 代表高斯符號，m_{max} 代表四角點最大可能值

1. 若 $n=k^2$

   則 $m_{max}=(k-1)^2=n-2k+1$

2. 若 $k^2 < n \leqq k(k+1)$

   四角點數量會是 k*k 圖形四角點的數量，再加上剩下的 $(n-k^2)$ 個方塊所排成的四角點數。由於自加入第二個後，m 值才逐次加一，所以我們有以下公式：

   $m_{max}=(k-1)^2+(n-k^2)-1=k^2-2k+1+n-k^2-1=n-2k$

3. 若 $k(k+1) < n < (k+1)^2$

四角點數量會是 k*k 圖形四角點的數量$(k-1)^2$，再將剩下的方塊排至圖形任一側產生(k-1)個四角點，待排齊後將最後{n-k(k+1)}個正方形再排列至鄰側。

則 $m_{max}=(k-1)^2+(k-1)+\{n-k*(k+1)\}-1$

$=k^2-2k+1+k-1+n-k^2-k-1=n-2k-1$

<<序組長度 T 值的推導>>

考慮依序增加一個方形的情況，若與原圖形相接於一個邊時，會增加兩個新的端點(T 增加 2)；如果相接於兩個邊時，則不會增加新的端點(T 值不變)，但會創造一個四角點(m 值增加 1)再由起始值 T=4（單位方形）

得到下列公式

$$T = 4 + 2(n - 1) -2m = 2n - 2m + 2$$

<<序組中元素總和 S 值的推導>>

由於每一個方形會使 S 增加 4（四個角落各佔 1）而每一個四角點則會使四個頂點互相抵消，於是 S 會減少 4。

就可以得到

$$S = 4n - 4m$$

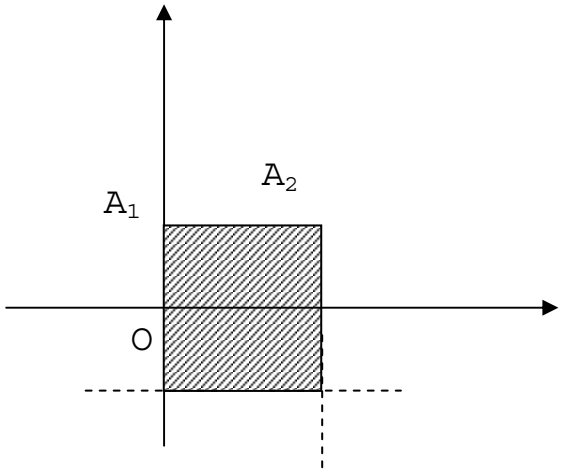
第三節　序組座標化

然而，任意序組就算滿足上述 m,T,S 的關係，也不見得可以轉換成正確圖形。因此我們尚需制定一套方法來剔除掉不符合條件的序組。

首先，設序組為 m(A_1, A_2, A_3,……, A_T)，並選定一直角座標系，其單位長為正方形邊長之一半，且原點 O 為 $\overline{A_T A_1}$ 之中點，正 Y 軸的方向為 $\overrightarrow{OA_1}$。此時，A_1 之座標為(0,1)。接著，要決定 A_2 的座標。

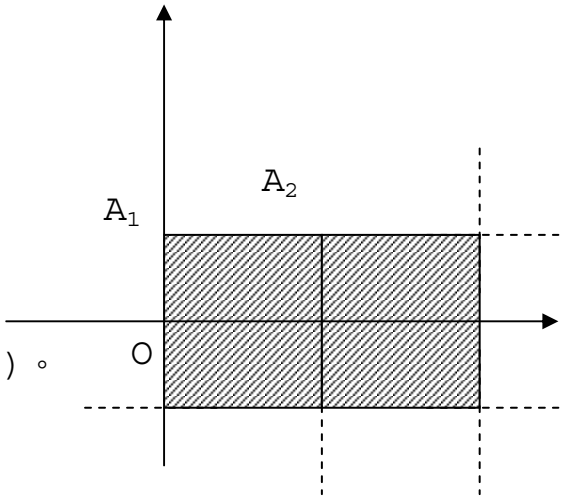
如果 A_1=1，則 $\overrightarrow{A_1A_2}$ 指向正 X 軸的方向，則 A_2=(2,1)。

（虛線表示該處可能連接有正方形）



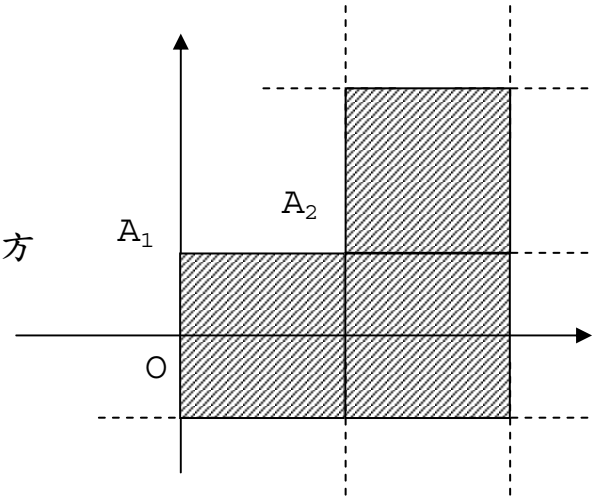
（圖九）

如果 A_1=2，則 $\overrightarrow{A_1A_2}$ 保持 $\overrightarrow{OA_1}$ 之方向（仍指向正 Y 軸），此時，A_2=(0,3)。



（圖十）

如果 $A_1=3$，則 $\overrightarrow{A_1A_2}$ 指向負 X 軸的方向，那麼 $A_2=(-2,1)$。



（圖十一）

很清楚地，A_2 的座標是由(1)A_1 之座標；(2)A_1 之數值；(3) $\overrightarrow{OA_1}$ 之方向來決定。這個事實可以用下列的代數式來表示：

令 $D_k \equiv 0,1,2,3 \pmod 4$ 分別代表正 Y 軸、正 X 軸、負 Y 軸、負 X 軸四個方向，其中 $D_k=\overrightarrow{A_kA_{k+1}}$〔k=1,2,3,……,(T-1)〕。設 $D_0=\overrightarrow{OA_1}=0$，$D_T=\overrightarrow{A_TA_1}$

且　　$A_k=1 \Rightarrow D_k=D_{k-1}+1$　（向順時針方向轉 90 度）

　　$A_k=2 \Rightarrow D_k=D_{k-1}$　　（維持 D_{k-1} 之方向性）

　　$A_k=3 \Rightarrow D_k=D_{k-1}-1$　（向順時針方向轉 270 度）

當 $D_k=\overrightarrow{A_kA_{k+1}}$〔k=1,2,3,……,(T-1)〕的方向性決定了，我們就可以依次決定 A_{k+1} 之座標如下：

如果　$D_k=0$，則 $A_{k+1}=(X_{k+1},Y_{k+1})=(X_k,Y_k+2)$

　　$D_k=1$，則 $A_{k+1}=(X_{k+1},Y_{k+1})=(X_k+2,Y_k)$

$D_k=2$，則 $A_{k+1}=(X_{k+1},Y_{k+1})=(X_k,Y_k-2)$

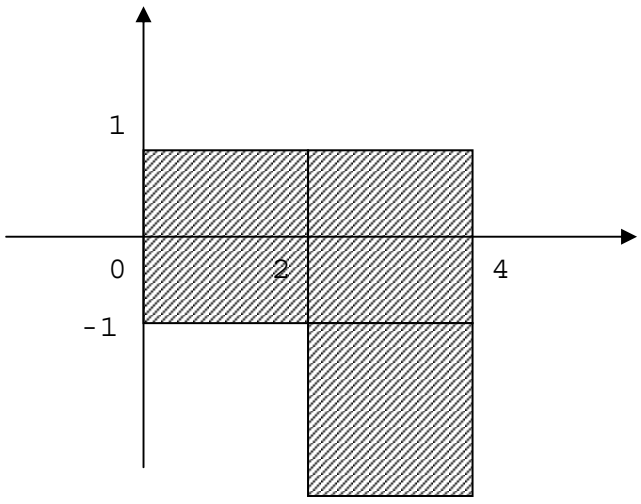
$D_k=3$，則 $A_{k+1}=(X_{k+1},Y_{k+1})=(X_k-2,Y_k)$

以序組 0(1,2,1,2,1,1,3,1)為例，如(表一)。

(表一)

元素值	方向性	點座標
	$D_0=0$	$A_1=(0,1)$
$A_1=1$	$D_1=D_0+1=1$	$A_2=(0,1)+(2,0)=(2,1)$
$A_2=2$	$D_2=D_1=1$	$A_3=(2,1)+(2,0)=(4,1)$
$A_3=1$	$D_3=D_2+1=2$	$A_4=(4,1)+(0,-2)=(4,-1)$
$A_4=2$	$D_4=D_3=2$	$A_5=(4,-1)+(0,-2)=(4,-3)$
$A_5=1$	$D_5=D_4+1\equiv 3$	$A_6=(4,-3)+(-2,0)=(2,-3)$
$A_6=1$	$D_6=D_5+1\equiv 0$	$A_7=(2,-3)+(0,2)=(2,-1)$
$A_7=3$	$D_7=D_6-1\equiv 3$	$A_8=(2,-1)+(-2,0)=(0,-1)$
$A_8=1$	$D_8=D_7+1\equiv 0$	$A_9=A_1=(0,-1)+(0,2)=(0,1)$

所對應的圖形如(圖十二)：



(圖十二)

那又如何利用這些座標來檢查序組的合法性呢？我們必須檢驗下列條件：

(1)起點和終點的座標是否相等？

(2)除了首尾外，是否有其他兩點的座標重合？

(3)計算面積，再跟 n 值核對，剔除錯誤的圖形。此步驟的必要性可由下例看出。

當 n=4，m=0 時，考慮序組 0(1,3,1,2,1,2,1,2,2,1)

(i)檢查序組長度 T 值是否正確？

T = 2n - 2m + 2 = 8 - 0 + 2 = 10

而這序組長度也為 10 (正確)

(ii)檢查序組元素和 S 是否正確？

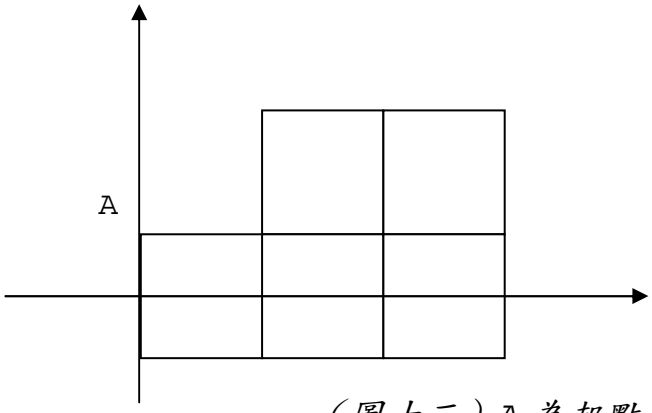
S = 4n - 4m = 16 - 0 = 16

而此序組元素和=1+3+1+2+1+2+1+2+2+1=16(正確)

(iii)寫出序組各點之直角座標，並檢驗前述條件(1)及條件(2)是否成立？ (正確)

元素值	方向性	點座標
	$D_0=0$	$A_1=(0,1)$
$A_1=1$	$D_1=1$	$A_2=(2,1)$
$A_2=3$	$D_2=0$	$A_3=(2,3)$
$A_3=1$	$D_3=1$	$A_4=(4,3)$
$A_4=2$	$D_4=1$	$A_5=(6,3)$
$A_5=1$	$D_5=2$	$A_6=(6,1)$
$A_6=2$	$D_6=2$	$A_7=(6,-1)$
$A_7=1$	$D_7=3$	$A_8=(4,-1)$
$A_8=2$	$D_8=3$	$A_9=(2,-1)$
$A_9=2$	$D_9=3$	$A_{10}=(0,-1)$
$A_{10}=1$	$D_{10}\equiv 0$	$A_{11}=A_1=(0,1)$

我們發現這序組通過目前所有的檢驗條件，但是在（圖一）俄羅斯方塊中並沒有圖形對應到這一個序組；可見這是錯誤的序組，但是利用座標，可以將這序組所對應的圖形描繪出來，如（圖十三）：



（圖十三）A 為起點

如果計算面積，我們會發現這序組對應的圖形面積是 5 而不是 n=4，因此，若將面積列為條件之一，則可以篩檢出類型這錯誤的序組。

第四節　演算法

Algorithm

Step 1:(initialization) input n; m=0;

Step 2:計算四角點個數上限 m_{max} 值。

參考第二節<<四角點個數 m 值的推導>>

$k=[n^{1/2}]$　　[]代表高斯符號

if $(n=k^2)$ $m_{max}=n-2k+1$;

if $(k^2 < n \leqq k(k+1))$ $m_{max}=n-2k$;

if $(k(k+1) < n < (k+1)^2)$ $m_{max}=n-2k-1$;

Step 3:創造序組

for(m=0;m≦m_{max};m++){

T=2n-2m+2;

S=4n-4m;　　（這兩條公式請參考第二節）

建構序組

m(A_1, A_2, A_3,……, A_T)

$\forall i \in N, i \le T \ni A_i \in \{1,2,3\}$

}

Step 4:檢驗元素總和

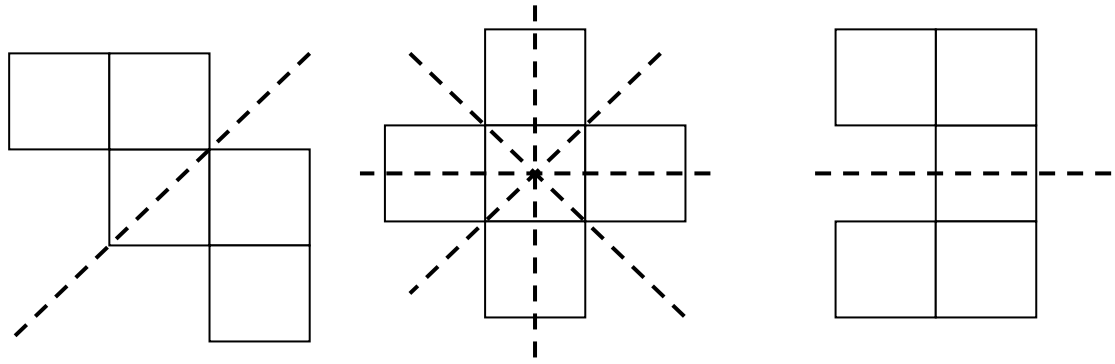
if($\sum_{i=1}^{T} A_i \neq S$) 剔除此不合的序組;

Step 5:計算序組對應的座標並檢驗，丟棄不合的序組

Step 6:印出合乎上述所有步驟的序組

第五節　對稱性的分析

對稱性一向廣為運用於組合圖形上，往往可以對原本的問題加以歸納簡化，而使得不論是程式編輯方面或者是運算速率上都大有幫助。在我們觀察一元方形至五元方形鏈的圖形後，發現到有蠻多圖形具有對稱性質（如圖十四），因此我們也嘗試著從這方向進行探討。

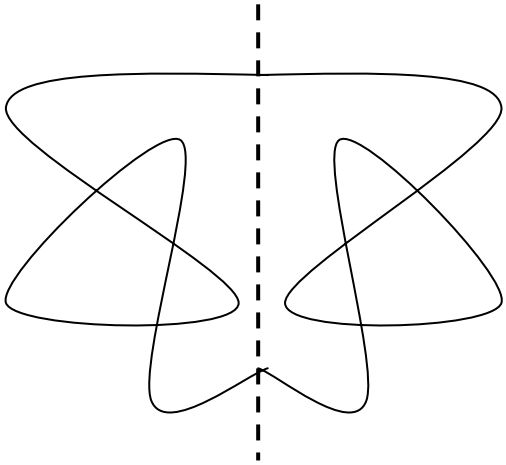


（圖十四）一些具有對稱性的五元方形鏈，虛線為其對稱軸

我們主要討論兩種對稱性質：1,線對稱　2,點對稱。

＜線對稱的定義＞

一個具線對稱性質的圖形，具有一條以上的對稱軸（直線），對於圖形上的任意點 A，必可在圖形上找到另一點 A′，使得線段 AA′ 的中垂線為該條對稱軸（如圖十五）。

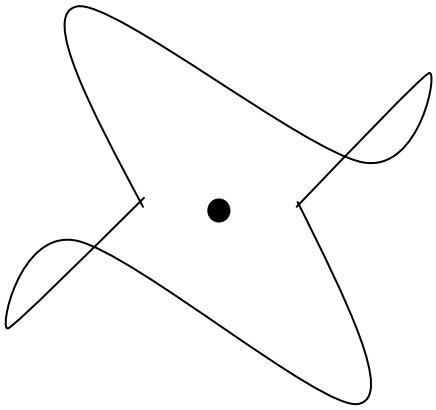


（圖十五）虛線兩側的圖形以虛線為對稱軸，具線對稱的性質。

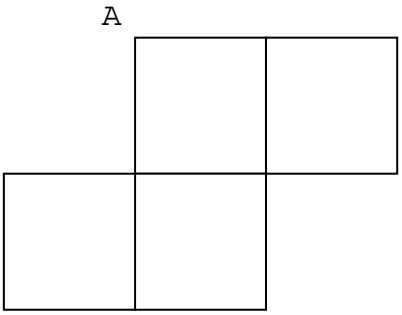
＜點對稱的定義＞

一個具點對稱性質的圖形，恰存在一點對稱點，對於圖形上的任意點 B,必可在圖形上找到另一點 B′,使得線段 BB′的中點為該對稱點(如圖十六)。

（圖十六）由於繞中央黑點旋轉 180 度後圖形仍然不變，因此右圖具有點對稱性質，對稱點為中央黑點。



再由觀察多方塊的圖形得知，具點對稱性質的圖形其序組至少有兩個以上相同的數列循環而成，以(圖十七)解釋如下：

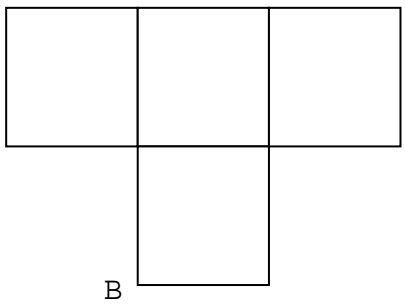


←→ 0(1,2,1,1,3,1,2,1,1,3)
^^^^^^^^^^出現循環
（以 A 點為起點）

（圖十七）四元方形鏈

這圖形祇具有點對稱的性質，恰可作為點對稱圖形的代表，而我們可以將序組的長度縮減為原長的一半，有助於程式的執行效率。

同樣的，具有線對稱性質的圖形其序組中的數列也會重複出現，只是呈現反向連結，以(圖十八)作解釋如下：



←→ 0(1,3,1,1,2,2,1,1,3,1)
^^^^^^^^^^出現循環
（以 B 點為起點）

（圖十八）四元方形鏈

此圖形恰有一條對稱軸。再者，我們進一步發現到，重複出現的數列恰為對稱軸同一側部分的圖形。

另外，尚有些圖形擁有兩條以上的對稱軸，或者是同時具有線對稱與點對稱的性質；我們針對這些圖形，分成下述五種類別：

1,[無任何對稱性]

2,[恰一條對稱軸]

3,[恰二條對稱軸]

4,[恰四條對稱軸]

5,[僅點對稱性質]

其序組之長度可依序分別簡化為：1(不變)，1/2，1/4，1/8，1/2。

由於探討的圖形是由單位正方形所組成的，產生的對稱軸只可能平行於單位正方形的邊，或者是平行於正方形的對角線，所以至多有四條對稱軸（2+2=4）。為了要證明所有的圖形恰可以分成這五類，我們必須要證明三件事情：

1 恰有兩條對稱軸的圖形，其對稱軸必互相垂直。

2 不存在恰有三條對稱軸的圖形。

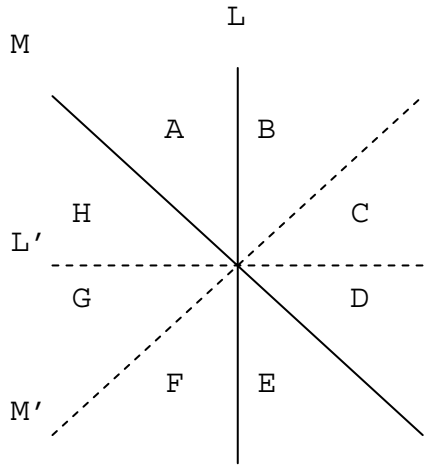
3 若一個圖形同時具備點對稱及線對稱的性質，則該圖形必存在兩條以上的對稱軸，反之亦然。

我們分下述三階段證明：

1,[恰二條對稱軸]的圖形，其對稱軸必互相垂直

利用反證法：

<證明>如右圖，若存在一個圖形恰有 L 及 M 兩條對稱軸，且 L 和 M 不垂直。因為對稱軸僅能平行於正方形的邊或對角線，所以 L 和 M 的夾角為 45°(135°)。現作兩條直線 L′和 M′分別垂直 L 和 M，並且通過 L 和 M 的交點。將原圖分成八個區域：A,B,C,D,E,F,G,H。為說明方便，以 A(L)B 表示 A 與 B 以 L 為對稱軸。



∵ A(M)H ， H(L)C ， C(M)F

∴ A(L′)F

同理，證得 B(L′)E

又∵ C(M)F ， F(L)E ， E(M)D

∴ C(L′)D

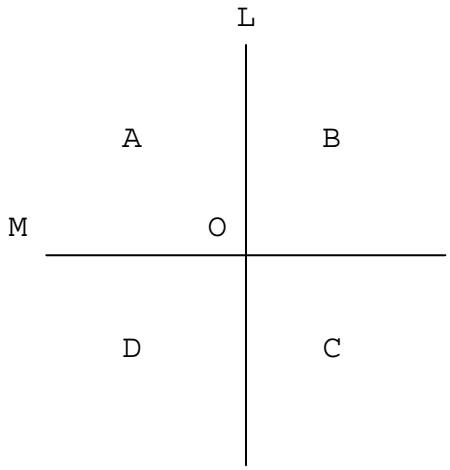
同理，H(L′)G

因此，L′為該圖形之一對稱軸。同理也可證得 M′也是一條對稱軸，與恰有兩條對稱軸的假設矛盾。

這也證明具有三條對稱軸的圖形，必具有第四條對稱軸。

2,含兩條以上對稱軸的圖形必具點對稱性質

<證明>如右圖，若存在一圖形具右圖垂直之對稱軸 L 及 M，為方便起見，以 A,B,C,D 表四個區域，O 代表 L 和 M 之交點。



∵ A(L)B ， B(M)C

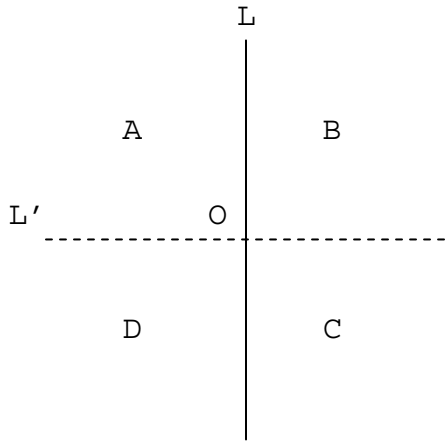
∴ A(O)C

同理證得 B(O)D

所以可以知道 O 為該圖形的對稱點，故得證。

3,若同時具有點對稱和線對稱性質之圖形，必存在有至少兩條的對稱軸

<證明>若存在一圖形具有右圖之對稱軸 L 和對稱點 O，為方便起見，過 O 作直線 L′垂直 L，並將圖形區分成 A,B,C,D 四個區域。



∵ A(L)B ， B(O)D

∴ A(L′)D

同理可得 B(L′)C

所以 L′也是原圖形之對稱軸，故得證

由對稱性的性質，我們了解到：一個具有對稱性的圖形，其序組可以大量縮減長度，這對運算有很大的幫助。(表二)是我們統計對稱圖形之數量及分類結果。

（表二）對稱圖形數量之統計：

單位方形數	*不具對稱性*	*1 條對稱軸*	*2 條對稱軸*	*4 條對稱軸*	*僅具點對稱*
1	0	0	0	1	0
2	0	0	1	0	0
3	0	1	1	0	0
4	1	1	1	1	1
5	5	4	1	1	1
6	20	8	2	0	5
7	85	15	4	0	3

根據我們統計的數值，我們預估當 n 值愈高時，polyominoes 愈趨向屬於高度不對稱的圖形，因此我們無法利用對稱性來輔助我們的程式，儘管對稱性可以縮短序組長度及驗證時間。

第六節　演算法執行結果及計算效益分析

就程式的觀點而言，對於這一方面的探討我們歸納有大約兩個方向，第一點是結論(即所需要的結果)。這一方面程式可以說是完成度達百分之百，因為不論是序組的類別以及完整序組的建構都已經能得到正確無誤的結果。

(表三)是 n=5 時的執行結果範例。

(表三)----------MS-DOS 環境----------

```
n=?
5

   n=5 m=0 accaababbaba
   n=5 m=0 acbbaabbbaba
   n=5 m=0 acbabaacbaba
   n=5 m=0 acbaacaacaba
   n=5 m=0 acacaabacaba
   n=5 m=0 acabbaabcaba
   n=5 m=0 acbaabcaabba
   n=5 m=0 abcbaabbabba
   n=5 m=0 acaacbaabbba
   n=5 m=0 abbbbaabbbba
   n=5 m=0 acaacaacaaca
   n=5 m=1 acababababba
 total :  12
  ------------------------------------------------
```

註：a 代表 1、b 代表 2、c 代表 3；total 是指 n=5 的所有圖形數。

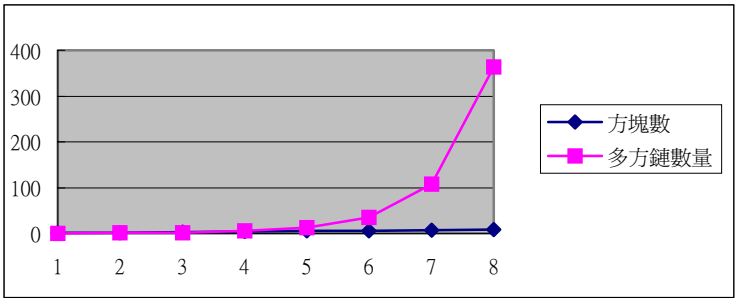
第二點是分析這一套計算多元方形鏈方法的數值效益。我們將第四節的演算法計算量劃分成兩部分來分析：<i>運算子（+，－，×，÷）；<ii>判斷式（if, for, switch, ……）的執行次數。

下面(表四)是我們對這兩種計算量的統計結果，同時也分別呈現在(圖十八～二十一)

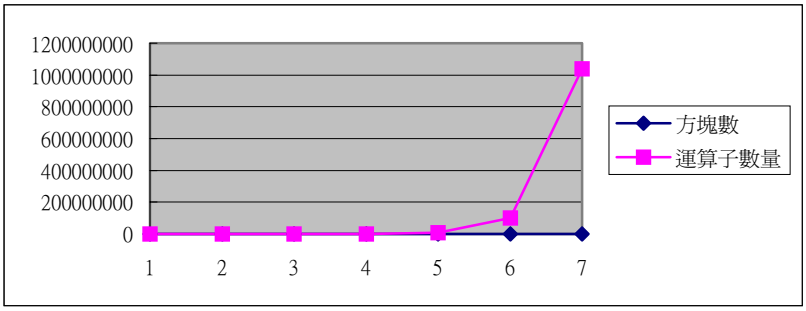
(表四)

方塊數 n 值	多方塊數量	運算子數量	判斷式數量	總運算量
1	1	570	83	653

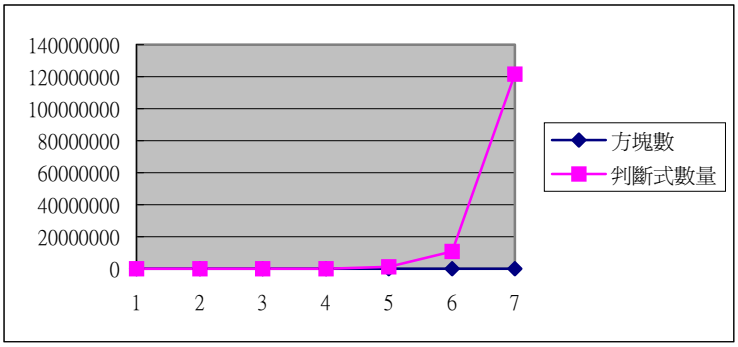
2	1	7,011	932	7,943
3	2	78,300	9,334	87,634
4	5	921,775	10,294	932,069
5	12	9,713,094	1,053,326	10,766,420
6	35	101,252,574	10,982,327	112,234,901
7	107	1,039,108,658	121,636,137	1,160,744,795



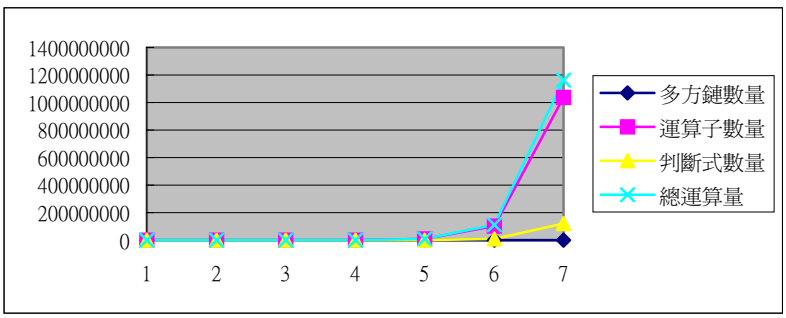
（圖十八）



（圖十九）



（圖二十）



（圖二十一）

我們發現，不管是運算子或判別式的執行次數，幾乎是以 10 為底的指數在成長，也就是 n 每增加 1，我們程式的計算量就增加約 10 倍，到 n=7 時，總計算量已經突破 10 億次，這也使得我們在目前的工作環境下(PentiumⅢ,CPU = 450MHz, RAM = 256MB)，最多僅可以計算到 n=8 的結果。如(表五)所列。

（表五）n 元方形鏈的數量（已和(文獻四)核對無誤）

方形數	圖形數
1	1
2	1
3	2
4	5
5	12
6	35
7	107
8	363

再仔細分析我們所有的子程式，我們發現計算量最繁重的地方發生在一個名為 gra 的副程式，其最主要的工作是計算序組座標的判別。如果想要增快程式執行效率，這一部份是首要改進的地方，所有的主／副程式我們皆收錄於附錄以供參考。

五、 總結

此研究的主題----polyominoes，是一種具高度變化性及趣味性的圖形，圖形的多樣性及組合性也常運用在一些益智類的遊戲上。這次我們並非研究其排列組合的多樣性（這是比較常看到的報告），

而是就圖形本身的數量加以討論，也提供一套解決方法。

這方法的主要價值在於簡化圖形成為序組，而序組又可以用來計算座標，並利用程式來篩檢計算 n 元非中空方形鏈的數量，得到了一些不錯的成果，也了解到當 n 值愈大，所生成的非對稱圖形也會愈多。

六、 參考文獻

(一) Martin Gardner ； 數學迷惘 Mathematical Puzzles & Diversions ； 初版 ； 台南市德光街 65-1 號 ； 復漢出版社 ； Page88~100 ； 1978

(二) 孫文先 ； 多方塊 Polyominoes ； 初版 ； 台北市信義路 3 段 147 巷 15 弄 5-1 號 7 樓 ；九章出版社 ；Page3~4；1999

(三) 柴田 望楊 ； 最新 c++學習講義 ； 初版 ； 台北縣汐止是新台五路一段 112 號 10 樓 A 棟 ； 2001

(四) George E. Martin；Polyominoes – A Guide to Puzzles and Problems in Tiling ； 10th edition ； U.S.A. ； MAA Spectrum ； Page1,13,23,39,55,101,123,178 ； 1996

(五) Kevin Gong；1991；Applying Parallel Programming to the Polyomino Problem ； http://kevingong.com/Polyominoes/ParallelPoly.html

(六) D. H. Redelmeier, W. F. Lunnon, Kevin Gong, Uwe Schult, Tomas Oliveira e Silva, and Tony Guttmann, Iwan Jensen and Ling Heng Wong；2000；Polyominoes

Enumeration ;
http://kevingong.com/Polyominoes/Enumeration.html

(七) Guttmann Anthony J., Jensen Iwan, Wong Ling Heng, Enting Ian G. ; 2000 ; Punctured polygons and polyominoes on the square lattice ; English summary ; No.9 ; Page1735~1764

(八) Barcucci Elena, Del Lungo Alberto, Pergola Elisa, Pinzani Renzo ; 1999 ; a methodology for the enumeration of combinatorial objects ; English summary ; No.4-5 ; Page435~490

七、 附錄（程式碼）

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define MAXSIZE 200
#define SIZE 200
int sum(char*str);
int max(int n,int k);
int check(char*str,int num);
int check2(char*str,int num);
int fucx(int num);
int fucy(int num);
int next(char*str,int num);
void clearafter(char*str,int index);
void initialize(char*str,int*n);
void reset(char*str,int num);
int shell(char*str,int num);
int gra(char*str,int n);
void boop(int*str,int max);
int top=0;
char STR[SIZE][SIZE];
int counter[SIZE];
```

```
int main()
{
   int i,n,s,numbers,k,M;
   int m,summation;
   char str[MAXSIZE];
   initialize(str,&n);
      k=(int)(sqrt(n));
      M=max(n,k);
      for(m=0;m<=M;m++){
         numbers=2*n+2-2*m;
         reset(str,numbers);
         str[numbers]='\0';
         while(1){
            if(next(str,numbers)==-1)
               break;
            summation=sum(str);
            if(summation!=4*n-4*m)
               continue;
            if(check(str,numbers)==-1)
               continue;
            if(check2(str,numbers)==-1)
               continue;
            top++;
            if(shell(str,numbers)==1){
               if(gra(str,n)==1){
               printf("\n   n=%d m=%d %s",n,m,str);
               counter[s]++;
               strcpy(STR[top],str);
               }
            else top--;
            }
         else top--;
         }
      }
   printf("\n total :  %d\n",top);
   getch();
   return(0);
}
```

```
int max(int n,int k)
{
   int M;
   if(n==k*k)M=(k-1)*(k-1);
   if(k*k<n&&n<=k*k+k)M=n-2*k;
   if(k*k+k+1<=n&&n<=k*k+2*k)M=n-2*k-1;
   return(M);
}

void initialize(char*str,int*n)
{
   int i;
       printf("n=?\n");
   scanf("%d",n);
   for(i=0;i<MAXSIZE;str[i++]='a');
   for(i=0;i<SIZE;counter[i++]=0);
}

void reset(char*str,int num)
{
   int i;
   for(i=0;i<num;str[i++]='a');
}

int check(char*str,int num)
{
   int i;
   int a=0,b=0,t;
   for(i=0;i<num;i++){
      switch(str[i]){
      case('c'):
         t=a;
         a=-b+1;b=t+1;
         break;
      case('b'):
         a=a+2;
         break;
```

```
      case('a'):
         t=a;
         a=b+1;b=-t-1;
         break;
      }
      if(i<num-1&&a==0&&b==0)
         return(-1);
   }
   if((a==0)&&(b==0))return(1);
   else return(-1);
}

int check2(char*str,int num)
{
   int x[SIZE],y[SIZE],z,i,j,k;
   x[0]=y[0]=z=0;
   for(i=0;i<num;i++){
      switch(str[i]){
      case('a'):
         x[i+1]=x[i]+fucx(z);
         y[i+1]=y[i]+fucy(z);
         z=z+1;
         break;
      case('b'):
         x[i+1]=x[i]+fucx(z);
         y[i+1]=y[i]+fucy(z);
         break;
      case('c'):
         x[i+1]=x[i]+fucx(z);
         y[i+1]=y[i]+fucy(z);
         z=z+3;
         break;
      }
   }
   if(x[num]>x[0]||x[num]<x[0]||y[num]>y[0]||y[nu
m]<y[0])return(1);
   for(j=0;j<num;j++){
      for(k=j+1;k<num;k++){
```

```
            if(x[j]==x[k]&&y[j]==y[k]){
               return(-1);
               break;
            }
         }
      }
      return(1);
}

int fucx(int num)
{
      if(num%4==1)return(1);
      if(num%4==3)return(-1);
      if(num%4==0||num%4==2)return(0);
}

int fucy(int num)
{
      if(num%4==0)return(1);
      if(num%4==2)return(-1);
      if(num%4==1||num%4==3)return(0);
}

int sum(char*str)
{
      int i,summation=0;
      for(i=0;i<strlen(str);i++){
         summation+=str[i]-'a'+1;
      }
      return summation;
}

int next(char*str,int num)
{
      int index=1;
      do{
         if(str[index]=='a'){
            str[index]='b';
```

```
            clearafter(str,index);
            return(1);
        }
        else if(str[index]=='b'){
            str[index]='c';
            clearafter(str,index);
            return(1);
        }
    }while(index++<num);
    return(-1);
}

void clearafter(char*str,int index)
{
    int i;
    for(i=0;i<index;i++)
    str[i]='a';
}

int shell(char*str,int num)
{
    int i,gap,k,len;
    char flag;
    for(i=0;i<top;i++){
        if(strlen(STR[i])!=num)continue;
        for(gap=0;gap<num;gap++){
            flag=0;
            for(k=0;k<num;k++){
                if(STR[i][k]!=str[(k+gap)%num]){
                    flag=1;
                    break;
                }
            }
            if(flag!=1)return(-1);
            flag=0;
            for(k=0;k<num;k++){
                if(STR[i][num-k-1]!=str[(k+gap)%num]){
                    flag=1;
```

```
                break;
            }
         }
         if(flag!=1)
            return(-1);
      }

   }
   return(1);
}

int gra(char*str,int n)
{
    int i=1,turn=0,q=0,j,k,o,l,h;
    int x[MAXSIZE],y[MAXSIZE],z[MAXSIZE];
    x[0]=0;y[0]=0;
    while(str[i]!='\0')
    switch(str[i-1]){
    case 'a':if(turn==0){
        x[i]=x[i-1]+1;
        y[i]=y[i-1]+1;}
        if(turn==1){
        x[i]=x[i-1]+1;
        y[i]=y[i-1]-1;}
        if(turn==2){
        x[i]=x[i-1]-1;
        y[i]=y[i-1]-1;}
        if(turn==3){
        x[i]=x[i-1]-1;
        y[i]=y[i-1]+1;}
        i++;turn++;if(turn==4)turn=0;break;
    case 'b':if(turn==0){
        x[i]=x[i-1];
        y[i]=y[i-1]+2;}
        if(turn==1){
        x[i]=x[i-1]+2;
        y[i]=y[i-1];}
        if(turn==2){
```

```
        x[i]=x[i-1];
        y[i]=y[i-1]-2;}
        if(turn==3){
        x[i]=x[i-1]-2;
        y[i]=y[i-1];}
        i++;break;
    case 'c':if(turn==0){
        x[i]=x[i-1]-1;
        y[i]=y[i-1]+1;}
        if(turn==1){
        x[i]=x[i-1]+1;
        y[i]=y[i-1]+1;}
        if(turn==2){
        x[i]=x[i-1]+1;
        y[i]=y[i-1]-1;}
        if(turn==3){
        x[i]=x[i-1]-1;
        y[i]=y[i-1]-1;}
        i++;turn--;if(turn==-1)turn=3;break;
    };
    o=0;
    for(j=(-2)*n+2;j<=2*n-2;j+=2){
      q=0;
      for(k=0;k<i;k++){
         if(y[k]==j){
            z[q]=x[k];
            q++;
         }
      }
      boop(z,q);
      if(q!=0)for(h=0;h<=q-1;h+=2){
      o+=(z[h+1]-z[h])/2;
      };
       }
    if(o==n) {return(1);}
    else return(-1);
}
```

```
void boop(int*str,int max)
{
     int flag=1,c,i;
     while(flag){
      flag=0;
      for(i=0;i<max-1;i++)if(str[i]>str[i+1]){
         c=str[i];str[i]=str[i+1];str[i+1]=c;
         flag=1;}
         }
}
```

評　　語

（1）本文探討 n 個方形組成多元方形鏈之組合數，其中以各頂點爲多少方形連接數之關係來計算其組合數。

（2）本文之結果並不完整，來扣除對稱個數且僅得 n≦8，且未能把列法與圖形配合