**國立臺灣師範大學附屬高級中學第46屆科學展覽會**

**作品說明書封面**

科　　別：資訊科

組　　別：高中組

作品名稱：郵票問題—n－方垛與排列組合

關 鍵 詞：郵票問題、排列組合、n－方垛

編 號：

摘　　要

　　本研究主要探討給定一頁m\*n的長方形郵票總數求從１開始，的可連續撕下的面值總額，的連續最大值及將其面值分配在一頁m\*n的長方形郵票中。

1. 研究動機

　　雖然郵票面值分配的問題不是什麼了不起的學問，卻是一種可以在生活中幫助人們的數學。在小小的一頁郵票上竟然有著許多耐人尋味的小細節。

貳、研究目的

1. 在任意n\*m的一頁郵票中都通用的最佳配置面額方法。
2. 探討在n\*m的一頁郵票中，若只有邊角相接也算連通的郵票撕法規則下的通用最佳配置方法。
3. 探討1\*m的一列郵票中，各個m值能達到的最佳配置方法及其最大連續面值。

參、研究設備及器材

A4筆記本一個、A4內頁一疊、筆等文具、可供計算之電腦

肆、研究過程與方法

一、 以下是達成研究目的可能可行的方法

(一)、解法一：由一個區域最佳解逐步調整各面值往上逼近可達到的最大的連續最大值。(只需邊角相接即算連通)

(二)、解法二：由最大值往下尋找可能的分配。(只需邊角相接即算連通)

(三)、解法三：從一個基準郵票開始，一面達成連續的最大值，一面增加。(只需邊角相接即算連通)

(四)、解法四：

二、解法一：由連續最大值往下尋找可能的分配。

為逼近所謂的連續最大值，首先必須要先求出其值。而連續最大值必小於等於其連通子集合個數。也就是特定的1到n\*m-方垛個數。

而我們討論的「總數為M\*N的郵票的連續子集合個數」是指sigma a=1~m\*n a-方垛的可能數、

而且就算旋轉後一樣，也會視為相異的形狀。所以我目前的目標會放在以n表示我們口中的n-方垛的可能數。

根據仔細的觀察後發現n+1-方垛的可能數就是所有可能的n-方垛的外圍在接上一個方塊的方法數剪掉其中重複的圖案再減掉一樣的。

不用減掉旋轉後一樣的，因為在我們的這個狀況裡，不同的方向的同一圖案會有可能會有

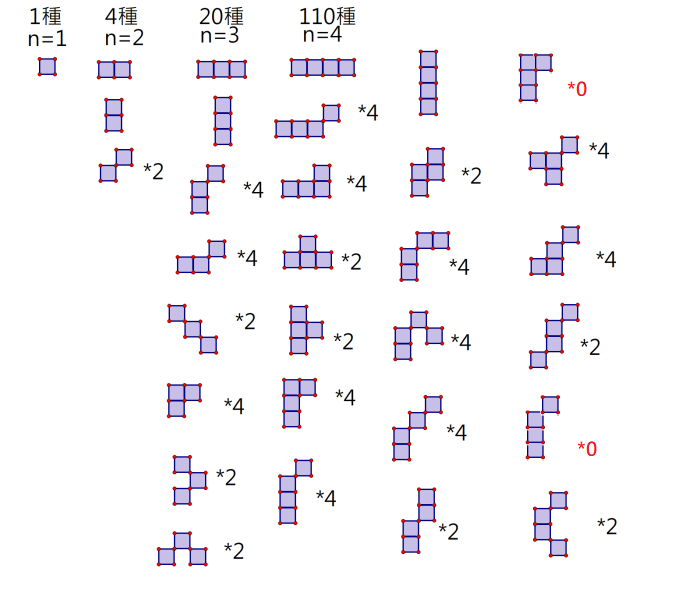
不同的寬和高。

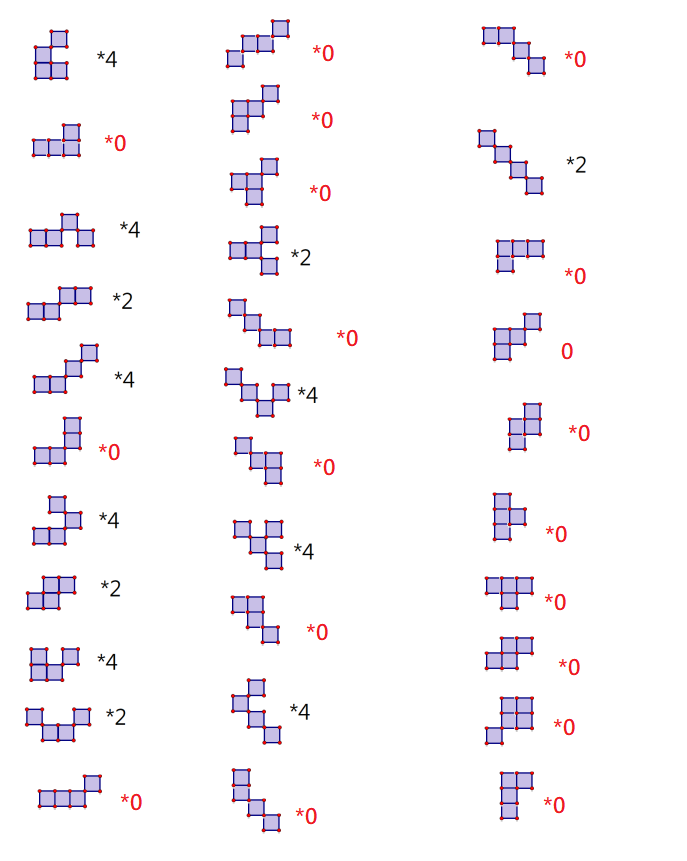
所以，目前先寫出可以算出n-方垛有幾種可能的程式吧。

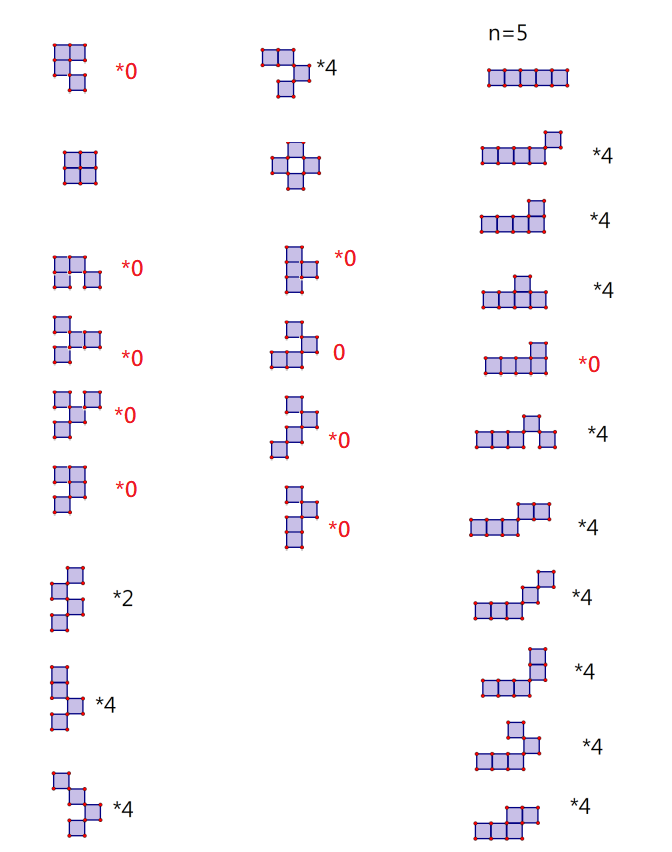
由n=1時，只有一種，而n+1-方垛的可能數就是所有可能的n-方垛的外圍在接上一個方塊的方法數剪掉其中重複的圖案再減掉一樣的。所以可以用遞迴的方式來寫，為了減低重複讀取相同結果會造成資源浪費，將會使用動態規劃的技巧。

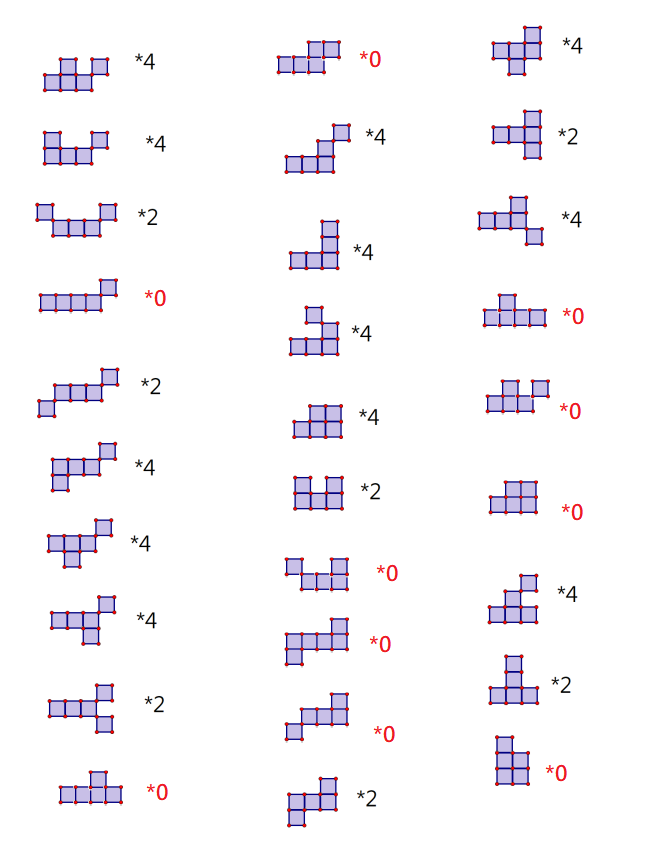
目前的程式碼，還沒有完全完成，目前可以執行但無實際功能，其中因為Python的基礎函式庫(library)好像不支援多維陣列，所以我使用numpy套件來實現多維陣列，目的是為了要儲存過程中計算出的每一個n-方垛的所有形狀，所以有四維[n, n-方垛的數量, 圖形的x, 圖形的y]

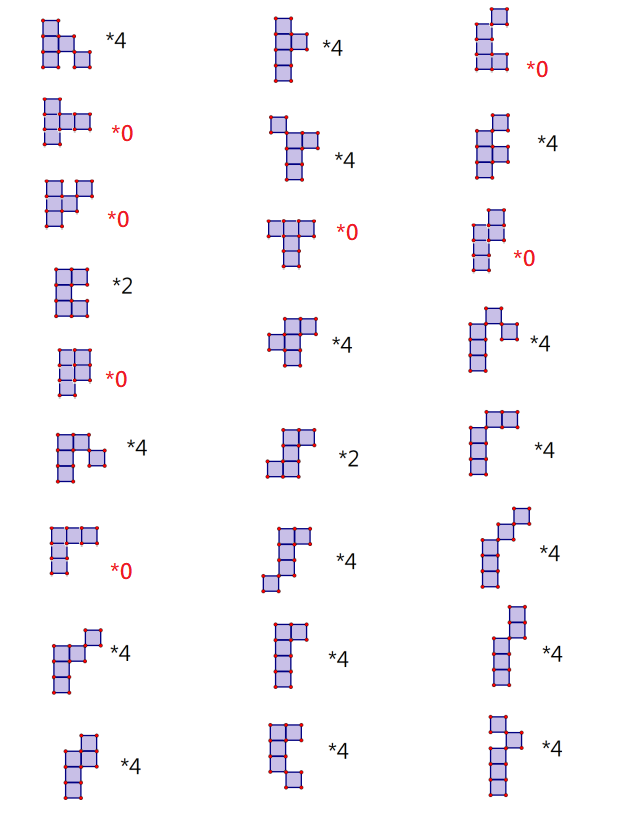
另一方面，我也試著從數字中找出規律。並試著把n=1~5 方垛列出來，結果5還沒列完，我就有了想法。





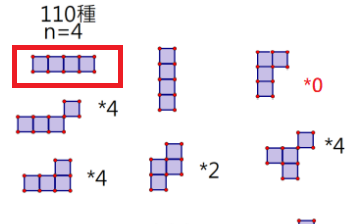


在

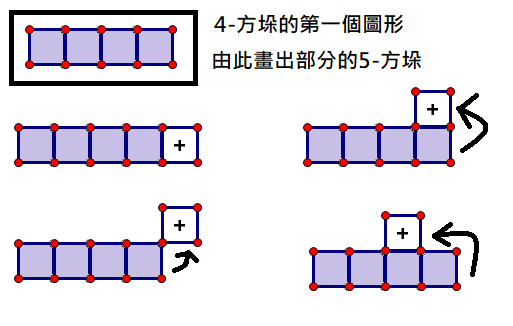


經過一系列的機械性動作後，我發現我重複著以下動作。

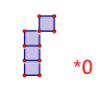
(一)、在n-1方垛中找還沒加工過的形狀



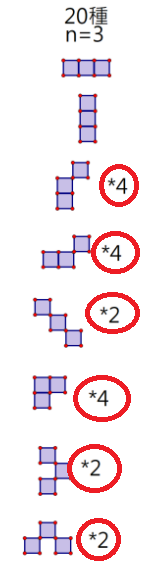
(二)、從最右下角開始加上新的方塊，然後依逆時針方向將n-1方垛的原圖都各加一個方塊。



(三)、翻轉過後如果發現這個形狀之前劃過，就標記\*0。代表之前已經算過了，不需要再算一次。



(四)、如果沒有畫過就看上下左右翻轉，不是旋轉，共有幾種不同的形狀。然後標記



(五)、最後再把所有數字加起來標記在n的旁邊

如右圖，所以當n=3時，只需邊角接壤的n-方垛共有

1+1+4+4+2+4+2+2 = 20個

所以我目前會想辦法找出一個方法讓電腦比較好判斷哪些形狀有畫過了，如果做到了，那麼我們就完成了一個小目標。也可以更快的算出之後n-方垛的個數。

我目前想的方法是，將圖形的數值轉變為數字，因為數字比較好互相比較。也就是把有視為1，沒有視為0，2視為換行，寫成一個三進位的數字，而這個數會受到上下左右翻轉的影響，不過沒關係，我們只取最小的數就好，

因為四維陣列太浪費空間，所以我會改成使用字串來記錄，其中”#”代表該處有方塊，” ”代表沒有方塊的地方，由”---”來分隔圖形，由”===”來分隔不同n的n-方垛。

下面是 1-方垛存在這個字串中的樣子。

#

===

目前的程式碼：

# import numpy as np

shape\_num = [1]

f = open('shape.txt', 'r')

shapes = f.read()

f.close()

# print(shape)

def access\_shape(n, i): # 方便存取n-方垛中的第i個圖形的函式

global shapes

return shapes.split('===\n')[n - 1].split('---\n')[i - 1]

def add\_shape(shape, uni\_code): # 方便將新的shape加進去shapes

global shapes

shapes = shapes + '---\n' + shape + '@' + str(uni\_code) + '\n'

return

def calculate\_shape\_num(n):

global shapes, shape\_num

if len(shape\_num) > n:

return shape\_num[n-1]

else:

old\_shape = ''

x = 0

y = 0

this\_shape\_num = 0

for i in range(0, calculate\_shape\_num(n-1)-1): # 第一步，找一個之前還沒被加工過的n-1-方垛

shape = access\_shape(n-1, i)

old\_shape = ' ' \* (n+1) + '\n'

for line in shape.split('\n'):

old\_shape = old\_shape + ' ' + line + ' \n' # 去頭去尾再多加一格空間

old\_shape = old\_shape + ' ' \* (n + 1) + '\n'

old\_shape = old\_shape.split('\n') # 之後比較方便換行繼續處理，也比較好編輯

for j in range(0, n-1): # 第二步，找到圖形最右下角的方塊，從他的右邊開始處理

for k in range(0, n-1):

if old\_shape[n-j-1][n-k-1] == '#':

y = j+1

x = k+2 # 紀錄位置

while True: # 使用while規避遞迴限制，因為整個函式最多只會呼叫自己一次，所以不會超出限制

new\_shape = old\_shape # 將 old\_shape 複製後再對其進行修改

if new\_shape[y][x] == '#': # 檢查此處是否有方塊

print('err, there is already a block here.', n, this\_shape\_num, x, y)

while True: # 讓我可以在出錯時檢查或是讓他繼續執行

try:

print(exec(input()))

except:

print('command error, type \'exit()\' to terminate') # 告訴不知道的人如何結束程式

new\_shape[y] = new\_shape[y][0:x] + '#' + new\_shape[y][x+1:len(new\_shape[y])] # 在該圖(x, y)處放置一個方塊

shapes = shapes + '===\n'

while True:

try: # 輸入n

n = int(input('n?'))

except: # 防呆機制

print('err, pls input positive int')

Continue

if n == 1:

print(1)

Continue

elif n < 1:

print('err, negative or zero get')

Continue

else: # 從0開始算，可以幫助之後規避python的遞迴限制

x = 0

while x < n:

x += 1

print(str(x) + str(calculate\_shape\_num(x)))

print('Program finish')

而另外，我也將前四個數字，1, 4, 20, 110丟進數列大全，OEIS 網站(參考資料一)查詢，發現已經有人把之後的個數都求了出來：1, 4, 20, 110, 638, 3832, 23592, 147941, 940982, 6053180, 39299408, 257105146, 1692931066, 11208974860, 74570549714, 498174818986, 3340366308393 …

標題為，Number of fixed n-celled polyominoes which need only touch at corners.

可知其目的與我非常相像。

三、解法二：由一個區域最佳解逐步調整各面值往上逼近可達到的最大的連續最大值。

首先可以發現，若是將n\*m個位置的面額都填上1，則可以輕易的達到連續的總額，其最大值為m\*n，若是將郵票1212的輪流填入，則也可以很輕易的達到從一開始，連續的總額，其最大值將近。

或甚至直接第n個位置就填n，那至少會有的連續最大值。

故原則上應該可以透過透過局部更動所有的面額來增加最大值。

四、解法三：從一個基準郵票開始，一面達成連續的最大值，一面增加。

如果要填的話，必定是從1開始填，之後，再填一個2，或一個1，來達到總額是2，再來想總額是3。如果整張圖只有一個1和2，或是只有兩個1。則這兩個數字必須連在一起。所以我想了一下各個正整數應該可以如何被組合出。

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 1 | 1+1, 2 | 1+1+1,1+2, 3 | 1+1+1+1, 1+1+2, 1+3, 4 |

我發現每一個正整數可以被組合出的方式就是1分別加上前一個正整數的所有方法再加上自己。所以在填的時候應該遵守以下原則：

(一)、從一開始填。

(二)、同一個數字盡量不要有一個以上的撕法。

(三)、填後的數字應該要是可以更動的，因為填完之前誰也不知道是不是最佳解。

五、限制n時的情形

(一)、n=1 時

首先，我們可以來討論1\*m的情況。也就是直直的一條線。這時可以明顯的發現，如果只撕1個，則有m種撕法，如果撕2個，則有m-1種撕法…如果撕m個，則有1種撕法。

所以其連通子集合個數有個。

而如果要達成最佳的面額分配。可以照著以下的步驟進行。

(一)、將每一種可能的值代入，從11…1到m m m … m。

(二)、檢查其可以達成連續的總值。

(三)、將其記錄下來，當出現更高的總值時，記錄下來並取代原本的最大值。

六、解法四：將1\*m的一列郵票所有可能跑遍。

1\*m的郵票，有m(m+1)/2種撕法，可以透過將要撕下的郵票的起點和終點的移動來聯想。所以理論上，透過把所有可能都檢查一遍來求值是可行的，在最糟情況下，大約只需要計算次，看來我們還有很大的進步空間。

程式碼：(還有問題尚待解決)

import math

import time

m = 1

record = open('1m-record.txt', 'a')

now\_max\_list = []

max\_dict = {}

max\_max = {}

while True:

m += 1

now\_list = []

max\_dict[m] = {}

max\_max[m] = 0

for i in range(m): # 初始化陣列

now\_list.append(1)

for i in range(int(math.pow(m\*(m+1)/2, m))): # 下一種面值分配

k = 0

for j in now\_list:

now\_list[k] += 1

if now\_list[k] == int(m\*(m+1)/2) + 1:

now\_list[k] = 1

k += 1

else:

break

# print(now\_list)

for j in range(m): # 每一種撕法

for k in range(j, m):

now\_max = 0

for p in range(j, k+1):

now\_max += now\_list[p] # 紀錄該撕法的總值

# now\_max\_list.append(now\_max)

max\_dict[m][now\_max] = now\_list + [j, k, now\_max] # 將總值以及撕法登陸

# now\_max\_list.sort()

for p in range(int(m\*(m+1)/2)): # 檢查連續最大值

try:

max\_dict[m][p+1]

except:

if max\_max[m] < p:

max\_max[m] = p

# print(str(p)+'23232322232')

# print(max\_dict[m][p])

break

print(max\_max)

try:

print(max\_dict[m][max\_max[m]])

except:

pass

七、未來展望

(一)、求出連續撕下x張郵票的所有方法。

(二)、給出通用的面值分配方式

伍、研究結果

一、一頁n\*m的郵票可達成的連續最大值必小於等於其連通子集合個數。

陸、討論

1. 目前正在討論如何更好且更快的求出n\*m郵票最好的分配方式。
2. 正在研究是否存在一種通用的填入方式。
3. 正在研究其連通子集合個數與其可達到的連續最大值的確切關係。
4. 正在考慮限制n的數字，將複雜度降低。
5. 正在研究如何加快程式演算速度。

柒、結論

一、一頁n\*m郵票，撕下的郵票可達到的連續最大值必小於等於其連通子集合個數。

捌、參考資料及其他

一、 OEIS 整數數列線上大全

n-方垛的個數（只需要直角接觸）OEIS:A006770 。取自oeis.org/A006770