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。

壹、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，十分耐人尋味。

貳、研究目的

一、在任意 n*m 的一頁郵票中都通用的最佳配置面額方法。

二、求出 n-方垛的個數與其關聯。

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)、方法一：使用 n-方垛求出可以達到的最大值。

(二)、方法二：由一個次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。

(三)、方法三：從一個基準郵票開始，一面達成連續的最大值，一面增加。

二、 已知資訊

總數為 M*N 的郵票可以達到的連續最大值為其連續子集合個數。

三、 相關名詞解釋

(一)、方塊，指的是在一頁郵票可以撕下的郵票

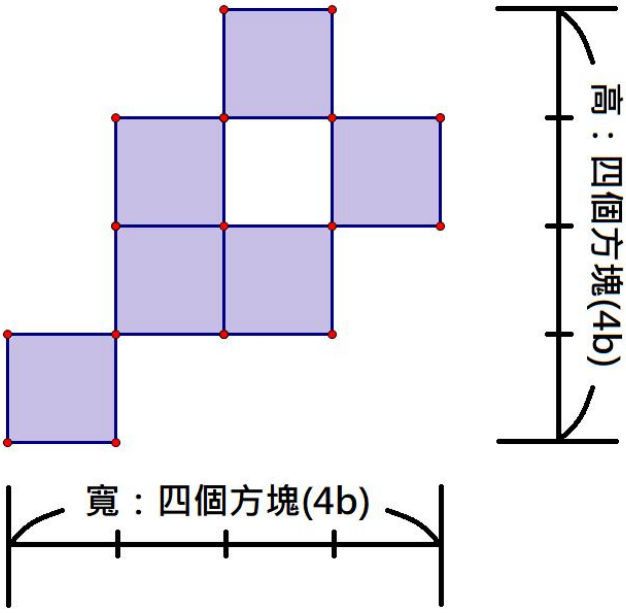
(二)、一頁郵票，指的是一頁長 m 張郵票寬 m 張郵票的長方形郵票集合，而且每一個郵票至少有兩個共用邊。

(三)、郵票，指的是正方形的紙，具有正整數面額。

(四)、寬和高，如右圖所示。

注意。就算方塊中間有空洞，依然是合格的郵票組合。

右圖所示為一合格的 6-方垛



四、方法一：使用 n-方垛求出可以達到的最大值。

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。

這句話經過斷句後，大致上呈現了上圖中右邊那張的意思

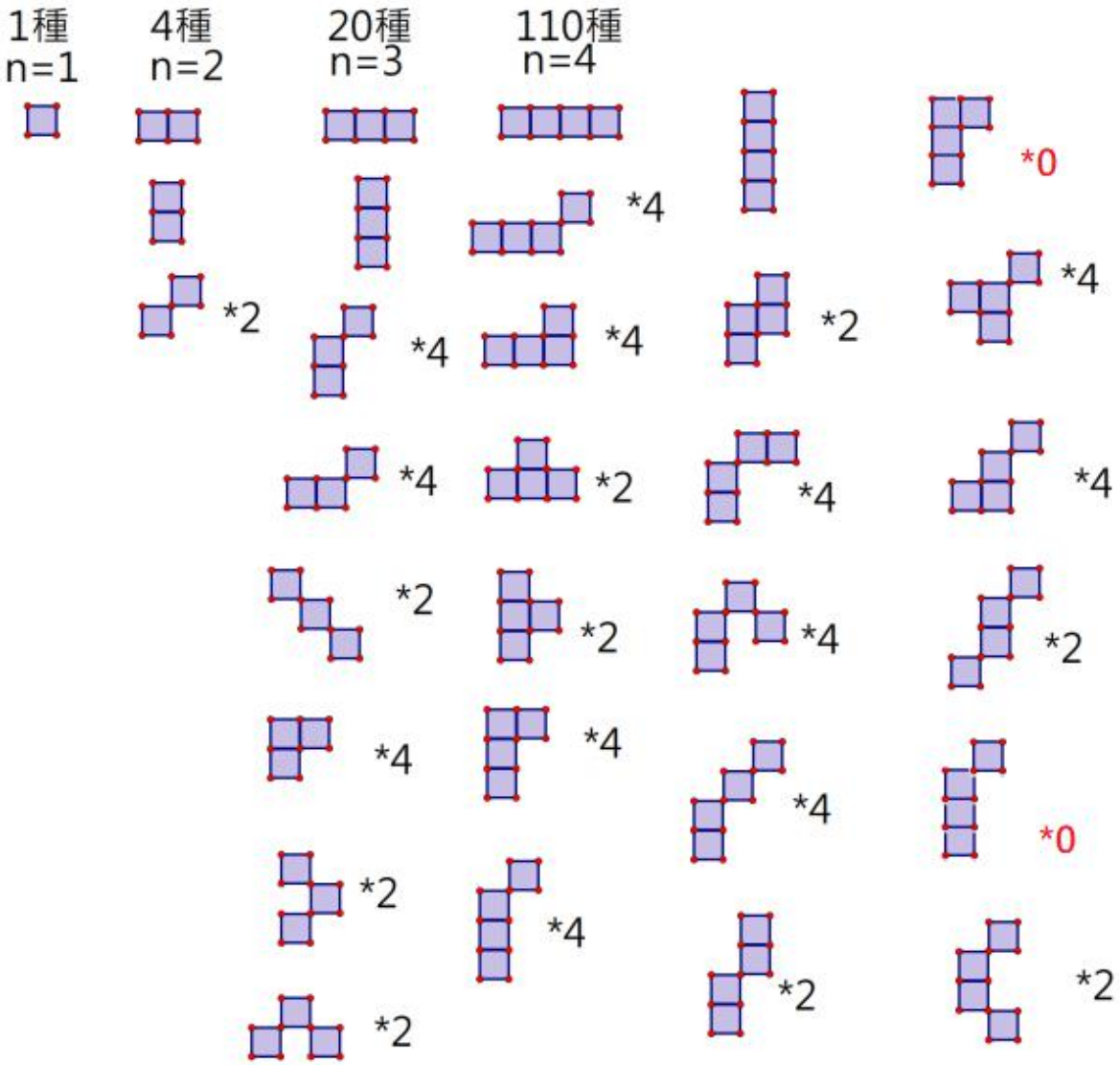
不用減掉旋轉後一樣的，因為在我們的這個狀況裡，不同的方向的同一圖案會有可能會有不同的寬和高。

所以，目前先寫出可以算出 n-方垛有幾種可能的程式吧。

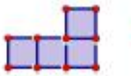
由 n=1 時，只有一種，而 n+1-方垛的可能數就是所有可能的 n-方垛的外圍在接上一個方塊的方法數剪掉其中重複的圖案再減掉一樣的。所以可以用遞迴的方式來寫，為了減低重複讀取相同結果會造成資源浪費，將會使用動態規劃的技巧。

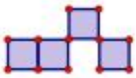
目前的程式碼，還沒有完全完成，目前可以執行但無實際功能，其中因為 Python 的基礎函式庫(library)好像不支援多維陣列，所以我使用 numpy 套件來實現多維陣列，目的是為了要儲存過程中計算出的每一個 n-方垛的所有形狀，所以有四維[n, n-方垛的數量, 圖形的 x, 圖形的 y]

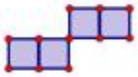
另一方面，我也試著從數字中找出規律。並試著把 n=1~5 方垛列出來，結果 5 還沒列完，我就有了想法。

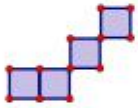


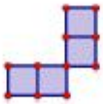

*4

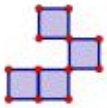

*0

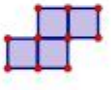

*4

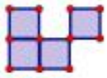

*2

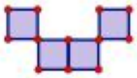

*4

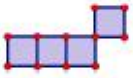

*0

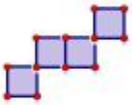

*4

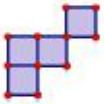

*2

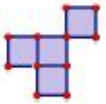

*4

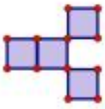

*2

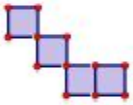

*0

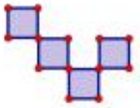

*0

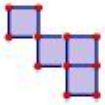

*0

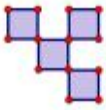

*0

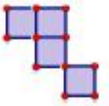

*2

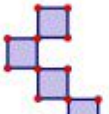

*0

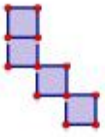

*4

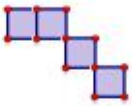

*0

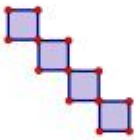

*4

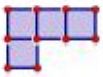

*0

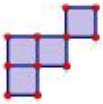

*4

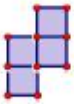

*0

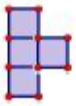

*0

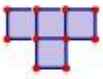

*2


*0

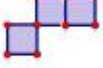

0

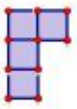

*0


*0

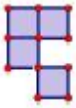

*0

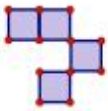

*0

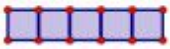

*0

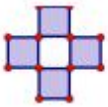

*0

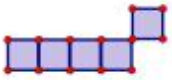
n=5

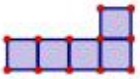

*0

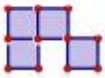

*4

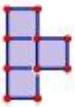


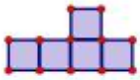


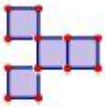

*4

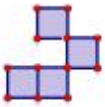

*4


*0

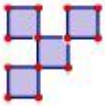

*0

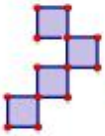

*4

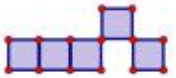

*0

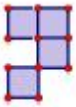

0

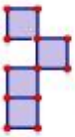

*0

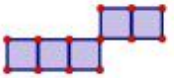

*0

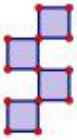

*0

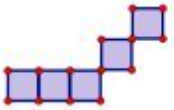

*4

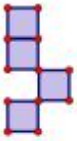

*0

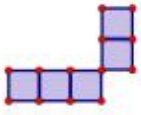

*0

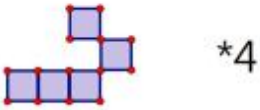

*4

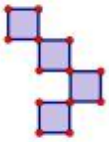

*2

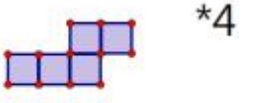

*4

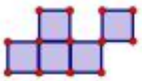

*4

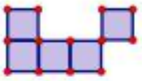

*4


*4

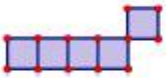

*4

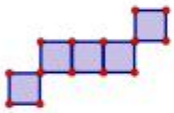

*4

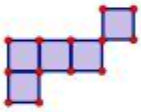

*4

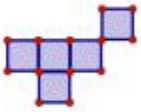

*4

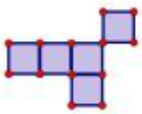

*2

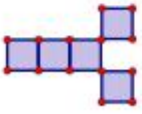

*0

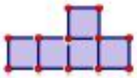

*2

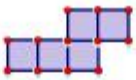

*4

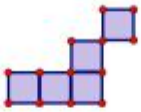

*4

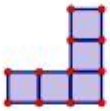

*4

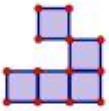

*2

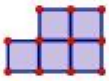

*0

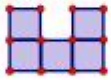

*0

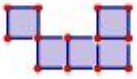

*4

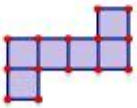

*4

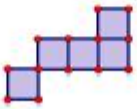

*4

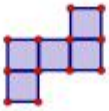

*4

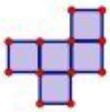

*2

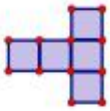

*0

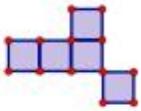

*0

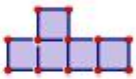

*0

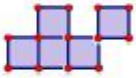

*2

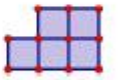

*4

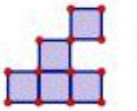

*2

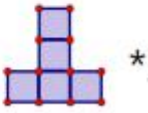

*4

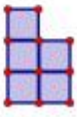

*0


*0

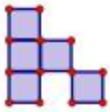

*0

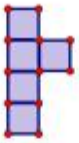

*4

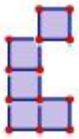

*2

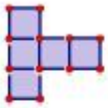

*0

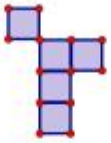
在

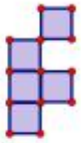

*4

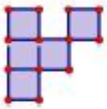

*4

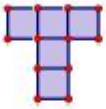

*0

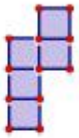

*0

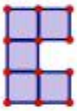

*4

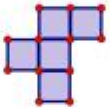

*4

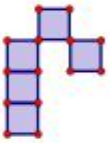

*0

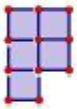

*0

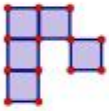

*0

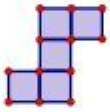

*2

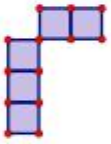

*4

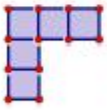

*4

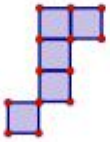

*0

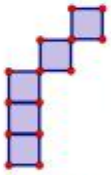

*4

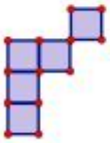

*2

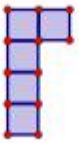

*4

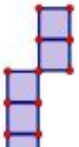

*0

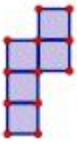

*4

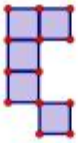

*4

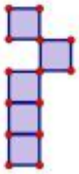

*4


*4


*4

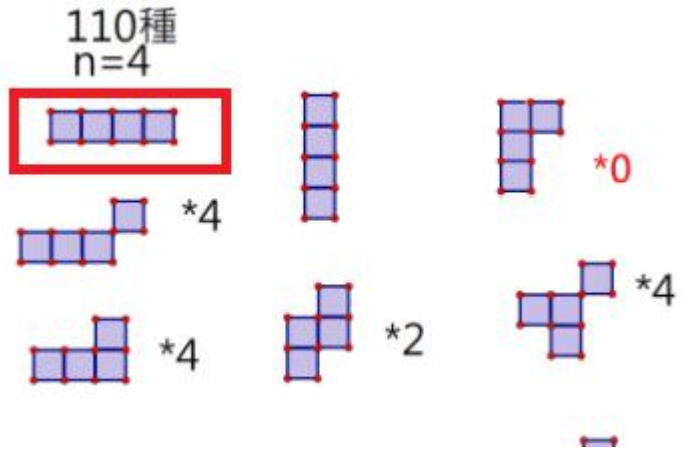

*4


*4

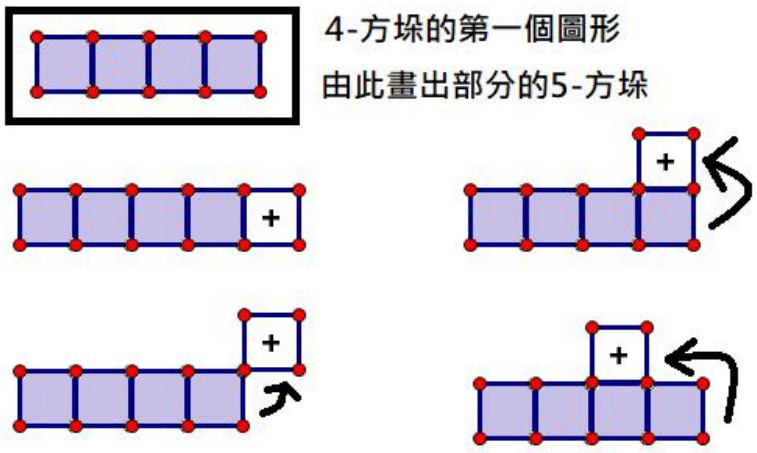

*4

經過一系列的機械性動作後，我發現我重複著以下動作。

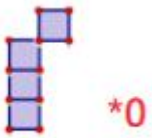
(一)、在 n-1 方垛中找還沒加工過的形狀



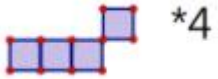
(二)、從最右下角開始加上新的方塊，然後依逆時針方向將 n-1 方垛的原圖都各加一個方塊。



(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。



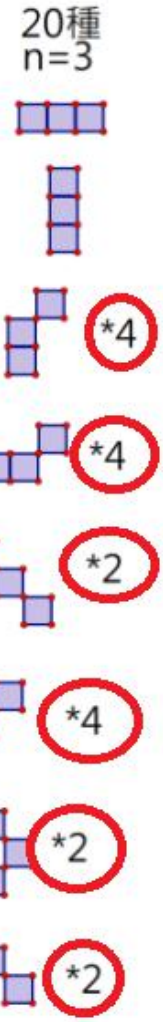
(四)、如果沒有畫過就看上下左右翻轉，不是旋轉，共有幾種不同的形狀。然後標記



(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。

五、方法二　：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。

像是這樣：

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

，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 $nm\times\frac{3}{2}$ 。長得這樣：

1	2	1	2	1
2	1	2	1	2
1	2	1	2	1
2	1	2	1	2
1	2	1	2	1

或甚至直接第 n 個位置就填 n，那至少會有 $(nm+1)nm\times\frac{3}{2}$ 的連續最大值。長得像這樣：

21	22	23	24	25
16	17	18	19	20
11	12	13	14	15
6	7	8	9	10
1	2	3	4	5

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。

所以我開始嘗試將上圖的連續最大值找出來，希望在過程中獲得靈感。

過程詳見，捌、參考資料與其他的附錄一。

在過程中，我發現一個明顯的事實，每一格只有兩種結果，撕，或者不撕。所以如果我要找這頁郵票中的連通子集合個數，或許可以將 2^(nm)種可能列出來後，再篩掉不是連通的圖形，剩下來的圖形集合就會是可行的撕法集合，這樣就不必討論 n-方垛了，就演算法的方面來

評估，原本利用 n-方垛的方法其實機複雜度約略為 $O(nm^{nm})$，而這個方法是 $2^{O(nm)}$，所以算是很大幅度的優化吧。

至於判斷一個子集合是否連通，只要確定每一格周圍都至少有一個方塊就夠了。

六、方法三：從一個基準郵票開始，一面達　連續的最大值，一面增加。
成

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。

七、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 $m(m+1)\frac{1}{2}$ 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。

八、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

一、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。

二、正在研究是否存在一種通用的填入方式。

三、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。

四、正在考慮限制 n 的數字，將複雜度降低。

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

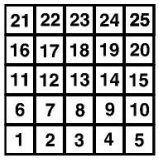
一、 參考資料

(一)、OEIS 整數數列線上大全

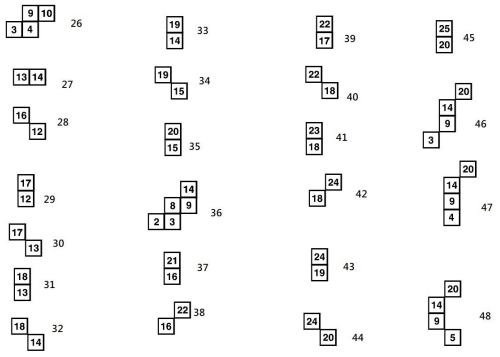
n-方垛的個數（只需要直角接觸）OEIS:A006770

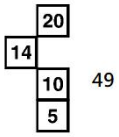
二、附錄

(一)、附錄一

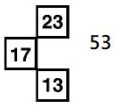


中的連通的子集合的總和的圖表，(僅列出 26~79)



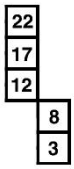


49

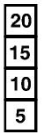


53

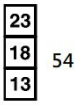
57



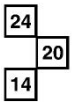
62



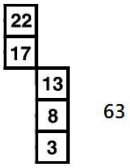
50



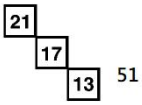
54



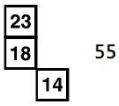
58



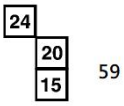
63



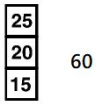
51



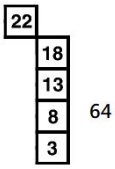
55



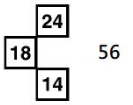
59



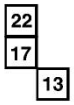
60



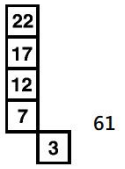
64



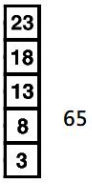
56



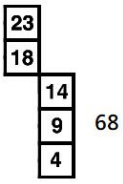
52



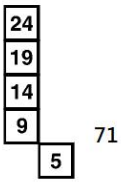
61



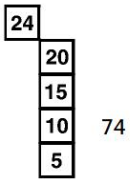
65



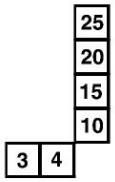
68



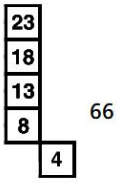
71



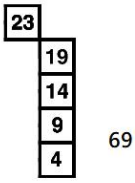
74



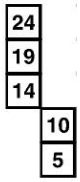
77



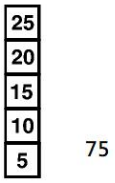
66



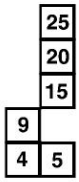
69



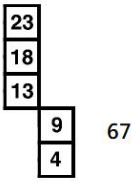
72



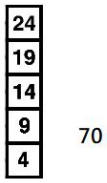
75



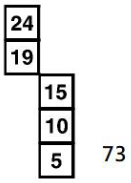
78



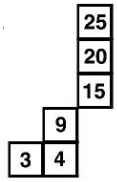
67



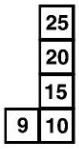
70



73



76



79