第十九屆旺宏科學獎

創意說明書

參賽編號：SA19-365

作品名稱：IC-Coloring—力爭上郵

姓名：曹禕中

關鍵字：IC-Coloring、郵票問題

# **研究題目**

研究在一張郵票裡的每一個格子填上一個數字，若從這張郵票連續的撕下郵票，其面值的總和可以湊出1到所有郵票的和。則此郵票中的面值分配方法稱為一個IC-Coloring。要求出在一張郵票裡，面值的總和可以最高的IC-Coloring的面值分配方法。

**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。並且有著令人意想不到的應用。小小的一頁郵票上竟然有著這麼多耐人尋味的細節，此點令我興奮不已。

**研究目的**

探討1×n排列的矩形郵票的各種IC-Coloring。

**研究方法**

1. 郵票問題

定義相關名詞：

1. 、郵票：可以填入面值的節點與，相鄰的節點之間有連線。
2. 、連通：即指有連線的。
3. 、連通子集合：即指一集合中所有的節點都可以透過連線互相接通。
4. 、連通子集合數：在本問題中指一郵票集合中，連通的子集合數。

二、1×n IC-Coloring

1. 、連通子集合數

我們可以很容易地得到1×n的郵票的連通子集合數為，因為，如果只撕1個，則有n種撕法，如果撕2個，則有n-1種撕法…如果撕n個，則有1種撕法。

所以1×n的郵票的子集合數為。

1. 、k的最大值的研究

這裡指的k的最大值為1×n且符合IC-Coloring排列的矩形郵票在各種面值分配下，其面值的總和。根據程式的演算，可以做出下表。程式請見附錄(五)。

|  |  |  |
| --- | --- | --- |
| n | k | 對應的排列方法 |
| 1 | 1 | [1] |
| 2 | 3 | [2, 1] |
| 3 | 6 | [2, 3, 1] |
| 4 | 9 | [3, 4, 1, 1], [2, 3, 3, 1] |
| 5 | 13 | [3, 4, 4, 1, 1], [2, 6, 1, 3, 1], [5, 2, 6, 3, 1], [2, 2, 3, 5, 1], [4, 2, 3, 7, 1] |
| 6 | 17 | [4, 5, 5, 1, 1, 1], [3, 4, 4, 4, 1, 1], [3, 2, 4, 6, 1, 1], [2, 3, 4, 6, 1, 1], [2, 3, 2, 6, 3, 1], [6, 1, 2, 8, 4, 1], [2, 2, 2, 3, 7, 1], [10, 3, 2, 4, 7, 1], [4, 2, 2, 3, 9, 1] |

三、1×n郵票的研究

1. 題目敘述

其實一開始我理解錯題意，誤以為k是越大越好，但是其實是要能湊出1到面值總和，然後面值總和要最大。

1. 連通子集合數

我們可以很容易地得到1×n的郵票的連通子集合數為，因為，如果只撕1個，則有n種撕法，如果撕2個，則有n-1種撕法…如果撕n個，則有1種撕法。

所以1×n的郵票的子集合數為。

1. 、k的最大值的研究

這裡指的k的最大值為1×n排列的矩形郵票在各種面值分配下，使其連通的郵票子集合之郵票面值總和為1到k的連續正整數。其中的k的最大值，根據程式的演算，可以做出下表。程式請見附錄。

|  |  |  |
| --- | --- | --- |
| n | k | 對應的排列方法 |
| 1 | 1 | [1] |
| 2 | 3 | [1, 2] |
| 3 | 6 | [1, 3, 2] |
| 4 | 9 | [1, 3, 3, 2], [1, 1, 4, 3], [4, 1, 2, 6] |
| 5 | 13 | [1, 3, 1, 6, 2], [1, 1, 4, 4, 3], [6, 1, 2, 2, 8] |
| 6 | 18 | [2, 5, 7, 1, 3, 6], [2, 5, 6, 3, 1, 8], [6, 4, 5, 2, 1, 13], [5, 2, 6, 3, 1, 14] |
| 7 | 24 | [8, 10, 1, 3, 2, 7, 8] |

1. 、不同演算法對計算時間的影響

目前演算法的優化的方向是，在基礎的暴力破解上，進一步改良。

1. 基礎的暴力破解(枚舉法)

每一種可能跑遍，最後排序，找出最大的k。

可以將時間花費做成下表n = 1~5

|  |  |
| --- | --- |
| n | 計算所花費時間(秒，取小數點下四位) |
| 1 | 0.0\* |
| 2 | 0.0\* |
| 3 | 0.0156 |
| 4 | 0.1670 |
| 5 | 17.2693 |

\*註：時間過小不易測量

1. 剪枝，減去沒有1的面值組合

在得知當前組合不可能實現時，跳過此組合進到下一可能。透過檢查該組合內有

無1，因為若該組合內不包含1，則必湊不出1這個面值。故可以透過先剪去這類組合達 到加快程式速度的效果。

可以將時間花費做成下表n = 1~5

|  |  |
| --- | --- |
| n | 計算所花費時間(秒，取小數點下四位) |
| 1 | 0.0\* |
| 2 | 0.0\* |
| 3 | 0.0156 |
| 4 | 0.0631 |
| 5 | 5.6868 |

\*註：時間過小不易測量

1. 剪枝，減去反轉後曾經算過的組合  
    由於郵票沒有方向，故同一個面值的組合可能被重複計算過1到2次。減去 這 類組合的面值可以省下大筆時間。且計算是否算過的時間遠小於後續的計算時間。

|  |  |
| --- | --- |
| n | 計算所花費時間(秒，取小數點下四位) |
| 1 | 0.0\* |
| 2 | 0.0010 |
| 3 | 0.0010 |
| 4 | 0.0450 |
| 5 | 3.7230 |

\*註：時間過小不易測量

可以從整體時間變少發現算法的確有效，而當n較小時，時間稍微增加應該是因為多了一個檢查的時間，且因為n夠小，故此計算時間大於不做剪枝的時間。但是可以發現當n較大時，時間就有明顯的下降了。

令目前單一面值的最大值為p，則檢測使用的方法是將整列的郵票轉為一個n位p進位的數字，若此數字倒轉過後比原本小，則代表此組合已經算過了。

詳細的流程為：

1. . 判斷第一位是否比最後一位大，若是的話，代表倒轉過後會比較大。
2. . 若(1)判斷為是，則跳過此組合，若非，就不跳過，若兩者一樣，就再比下一個 。

以下為此判斷的程式。

for j in range(check\_times):

if stamps[j] > stamps[n - j - 1]:

break # 跳過這一次

elif stamps[j] == stamps[n - j - 1]:

Continue # 再比下一個

else:

# 這裡要放後續判斷的函式，也就是不跳過的組合會執行的函式

1. 不同演算法之間的執行時間比較

|  |  |  |  |
| --- | --- | --- | --- |
| n | 枚舉法(秒，取小數點下四位) | 剪枝，減去沒有1的面值組合(秒，取小數點下四位) | 剪枝，減去反轉後曾經算過的組合(秒，取小數點下四位) |
| 1 | 0.0\* | 0.0\* | 0.0\* |
| 2 | 0.0\* | 0.0\* | 0.0010 |
| 3 | 0.0156 | 0.0156 | 0.0010 |
| 4 | 0.1670 | 0.0631 | 0.0450 |
| 5 | 17.2693 | 5.6868 | 3.7230 |

\*註：時間過小不易測量

**參考資料及附錄**

一、參考資料  
 (一)、Ebrahim Salehi 、 Sin-Min Lee 、 Mahdad Khatirinejad ,2005/8/28, Discrete

Mathematics, IC-Colorings and IC-Indices of graphs, Pages297~310  
二、程式碼  
 (一)、1×n郵票最大k值求解程式：(最佳版本，包含時間測量)

import math  
import time  
  
n = 2  
  
while True:  
 start\_time = time.time()  
 p = int(n \* (n + 2) / 2)  
  
 stamps = [0] \* n  
  
 k = n  
 k\_list = []  
 k\_dict = {}  
  
 check\_times = int((len(stamps) - (len(stamps) % 2)) / 2)  
  
 for i in range(int(math.pow(p, n))):  
  
 num\_pos = 0  
 for j in stamps: # 下一種面值分配  
 stamps[num\_pos] += 1  
 if stamps[num\_pos] == p + 1:  
 stamps[num\_pos] = 1  
 num\_pos += 1  
 else:  
 break  
  
 for j in stamps: # must be one 1 stamp  
 if j == 1:  
 break  
 else:  
 continue # no 1 in stamps  
  
 for j in range(check\_times):  
 if stamps[j] > stamps[n - j - 1]:  
 break # Skip this time  
 elif stamps[j] == stamps[n - j - 1]:  
 continue  
 else:  
 record = []  
 # print(stamps, i)  
 for m in range(n): # Methods of coloring.  
 for r in range(m, n):  
 summary = 0  
 for q in range(m, r + 1):  
 summary += stamps[q]  
 record.append(summary)  
  
 record.append(p + 1) # Make a BIG gap.  
 record.sort()  
  
 for m in range(len(record)):  
 x = record[m]  
 if x + 1 < record[m + 1]:  
 if x > k:  
 # print(k\_list)  
 k\_list = [str(i) + ':' + str(stamps)]  
 # k\_dict = {i: str(stamps)}  
 k = x  
 elif x == k:  
  
 k\_list.append(str(i) + ':' + str(stamps))  
 # k\_dict[i] = str(stamps)  
 # print(stamps)  
 # print(k\_list)  
 break  
 else:  
 print('error occur')  
 print(record)  
 input()  
 break  
  
 print(n)  
 print(k, k\_list)  
 # print(k\_dict)  
 f = open('result.md', 'a')  
 f.write(f'n: {n}\nk: {k} method: {k\_list}\n')  
 f.close()  
 n += 1  
 print(time.time() - start\_time)

1. 、找面值編號對應面值

import math

while True:

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

pos\_num = int(input('pos\_num?'))

p = int(n \* (n + 1) / 2)

stamps = [0] \* n

for i in range(pos\_num+1):

num\_pos = 0

for j in stamps: # 下一種面值分配

stamps[num\_pos] += 1

if stamps[num\_pos] == p + 1:

stamps[num\_pos] = 1

num\_pos += 1

else:

break

print(stamps)

1. 、尋找連通子集數

import math

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

m = int(input('input m'))

counter = [0] \* (n \* m)

def plus1(big\_base\_2\_num\_list, place):

if big\_base\_2\_num\_list[place] == 0:

big\_base\_2\_num\_list[place] = 1

else:

big\_base\_2\_num\_list[place] = 0

plus1(big\_base\_2\_num\_list, place - 1)

p = 0

for i in range(int(math.pow(2, n \* m))):

plus1(counter, len(counter) - 1)

print(counter)

old\_counter = counter

# checker = [0] \* (n \* m)

for j in range(m):

counter.insert((j + 1) \* n, 0)

counter = counter + [0]\*(n+1)

print(counter)

for j in range((n + 1) \* (m + 1)):

if counter[j] == 1:

if counter[j + 1] + counter[j + m + 1] == 0:

break

else:

p += 1

counter = old\_counter

print(p)

1. 、多核心加快版

import math

import time

import multiprocessing

n = 5

def find\_k(first\_stamp):

global n, p, k\_dict, k, k\_list, group\_stamps, check\_times

stamps = [0] \* (n - 1) + [first\_stamp]

for i in range(group\_stamps):

# print(i, stamps)

num\_pos = 0

for j in stamps: # 下一種面值分配

stamps[num\_pos] += 1

if stamps[num\_pos] == p + 1:

stamps[num\_pos] = 1

num\_pos += 1

else:

break

for j in stamps: # must be one 1 stamp

if j == 1:

break

else:

continue # no 1 in stamps

for j in range(check\_times):

if stamps[j] > stamps[n - j - 1]:

break # Skip this time

elif stamps[j] == stamps[n - j - 1]:

continue

else:

record = []

# print(stamps, i)

for m in range(n): # Methods of coloring.

for r in range(m, n):

summary = 0

for q in range(m, r + 1):

summary += stamps[q]

record.append(summary)

record.append(p + 1) # Make a BIG gap.

record.sort()

for m in range(len(record)):

x = record[m]

if x + 1 < record[m + 1]:

if x > k:

k\_list = [str(stamps)]

k = x

elif x == k:

k\_list.append(str(stamps))

break

else:

print('error occur')

print(record)

input()

break

if \_\_name\_\_ == '\_\_main\_\_':

start\_time = time.time()

multiprocessing.freeze\_support()

p = int(n \* (n + 1) / 2)

group\_stamps = int(math.pow(p, n - 1))

k = n

k\_list = []

k\_dict = {}

check\_times = int((n - (n % 2)) / 2)

pools = multiprocessing.Pool(4)

first\_stamp\_queue = multiprocessing.Queue()

for i in range(p):

first\_stamp\_queue.put(i)

ps\_list = []

for i in range(p):

ps\_list.append(multiprocessing.Process(find\_k(i)))

ps\_list[i].start()

for i in range(p):

ps\_list[i].join()

print(n)

print(k, k\_list)

f = open('result.md', 'a')

f.write(f'n: {n}\nk: {k} method: {k\_list}\n')

f.close()

print(time.time() - start\_time)

1. 、IC-Coloring 綜合

import math

import time

n = 4

while True:

start\_time = time.time()

p = n \* (n + 1) // 2

stamps = [1] \* n + ['ignore']

k = n

k\_list = []

# k\_dict = {}

check\_times = (len(stamps) - (len(stamps) % 2)) // 2

for i in range(int(math.pow(p, n-1))):

num\_pos = 0

for j in stamps: # 下一種面值分配

# print(stamps)

stamps[num\_pos] += 1

if stamps[num\_pos] == p + 1:

stamps[num\_pos] = 1

num\_pos += 1

else:

break

record = []

for m in range(n): # Methods of coloring.

for r in range(m, n):

summary = 0

for q in range(m, r + 1):

summary += stamps[q]

record.append(summary)

record.append(p + 1) # Make a BIG gap.

record.sort()

for m in range(len(record)):

x = record[m]

if x + 1 < record[m + 1]:

stamps\_summary = 0

for j in range(n):

stamps\_summary += stamps[j]

if x > stamps\_summary:

break

if x > k:

k\_list = [str(i) + ':' + str(stamps)]

k = x

elif x == k:

k\_list.append(str(i) + ':' + str(stamps))

break

else:

print('error occur')

print(record)

input()

print(n)

print(k, k\_list)

f = open('result.md', 'a')

f.write(f'n: {n}\nk: {k} method: {k\_list}\n')

f.close()

n += 1

print(time.time() - start\_time)