

---

# Python List 應用

臺北科技大學資訊工程系

---

# Exercise 階梯數字

- 輸入一個整數N，輸出小於N的所有階梯數字個數
  - 檢查K是否階梯數字
    - 分析一串長數字是否單調上升
    - '111111111112222334455667778999'    True
    - '111111111112222334457667778999'    False
    - 傳入一個數字字串，回傳在**第幾個字元沒有單調上升**

演算法第一版，N是一般整數

1. 輸入N，K=1，Count=0

2. While (檢查 K<N)

2.1. 檢查 K 是否階梯數字

Count+1

2.2. K 加1

#檢查 K 是否階梯數字

def check(x):

for i in range(len(x)-1,-1,-1):

if x[i]<x[i-1]:

return i-

return -1

# Exercise 階梯數字

- 輸入一個整數N，輸出小於N的所有階梯數字個數
  - N是長整數，Python整數資料型別無法表示，須以字串表示

演算法第二版，主要流程(長整數數字字串N)

1. K='1', Count=0
2. While (檢查 K<N) (cmp(K, N)函式)
  - 2.1. 檢查 K 是否階梯數字 (check(K)函式)  
Count+1
  - 2.2. K 加1 (addOne(K)函式)

cmp(K, N)函式

//若K長度較N短<，回傳True，  
//否則從最K高位，往下一個數字一個數字比 (Loop)  
//若有<則回傳 True  
//迴圈結束未返回，則回傳 False

check(K)函式

//i 從最高位往下檢查，(Loop)  
// i>i-1 則為第 i 個數字非遞增，回傳 i  
//若都沒有發生，則為遞增，回傳 -1

# 階梯數字

## addOne函式(K)

若K 個位數<'9'，加 1 回傳 return

carry = 1，K個位數9，加1要進位

K[i] 從 K 十位數開始，一直到最高位數，迴圈

K[i]若為 9，又有進位carry==1，K[i]變成 0 且下一位進1，下一個carry=1

K[i]若<9，且carry=0，中斷迴圈

最高位有進位，多一位數

```
def addOne(x):
    length = len(x)
    if x[length-1]<'9':
        return x[:length-1]+str(int(x[length-1])+1)
    carry = 1
    x = x[:length-1]+'0'
    for i in range(len(x)-2,-1,-1):
        if int(x[i])+carry==10:
            x = x[:i]+'0'+x[i+1:]
            carry = 1
        else:
            s= str(int(x[i])+carry) #x[i]=
            return x[:i]+s+x[i+1:]
    if (carry==1):
        x = '1' + x
    return x
```

```
x1 = '11111111111111111112349'
x2 = '11111111111111111112350'
x3 = '11111111111111111112355'
x4 = '1111111111111111111999999'
x5 = '111111111111111112222222'
# 從最後往前找到非9，在前一個加一，
# 後面全變成此數字
y = addOne(x2)
```

# 階梯數字

第三版，直接找下一個階梯數字，主流程(數字字串N)

1. **K='11'** , Count=0
2. While (檢查 K<N) (**cmp函式**)
  - 2.1. **找出**下一個階梯數字 (**getNext函式**)  
Count+1

(cmp 函式)

```
//若長度較短為<，回傳True，
```

```
//否則從最高位，往下一個數字一個數字比 (Loop)
```

```
//若有<則回傳 True
```

```
//迴圈結束未返回，則回傳 False
```

## getNext函式

[illegible]

```
x2 = '111111111111111111112359'
```

[illegible][illegible]

```
x5 = '1111111111111111111111999999'
```

```
x6 = '11111111111111112222222'
```

# 從最後往前找到非9，在前一個加1，後面全變成此數字

113

# 階梯數字

getNext函式(K)



//index 從最低位開始找不是 9，Loop

//若找到第 index 位不是9

//將最低位~第 index 位，都設為: 第 index位 數字 +1

//若每一位數都是 9，

//多一位數，且每一位數都設為 1

```
def getNext(x):  
    index = len(x)-1  
    for i in range(len(x)-1,-1,-1):  
        if x[i]!='9':  
            index =   
            break  
    s = str(int(x[index])+1)  
    r = x[:index]  
    for i in range(index,len(x),1):  
        r = r+  
    return r
```

# 階梯數字

長度 \ 開頭	9	8	7	6	5	4	3	2	1
1	1	1	1	1	1	1	1	1	1
2	1	2	3	4	5	6	7	8	.....
3	1	3	6	10	15	21	28	.....	
4	1	4	10	20	35	56	.....		
5	1	5	15	35	70	.....			
6	1	6	21	56	.....				
7	1	7	28	.....					
8	1	8	.....						
.....	.....	.....							

# 階梯數字

- 以9為開頭最高位且長度為1、以8為開頭且長度為1,...的階梯數字之數量。是一個經典的巴斯卡三角形。
  - 長度L的階梯數字之數量把9開頭、8開頭等長度皆為L加總。
- N為2232，累積數量
  - 長度1 + 長度2 + 長度3 = 219 個數字
  - 長度4，開頭2，所以開頭為1要計入：
    - $219 + \text{開頭為1且長度為4} = 384$
  - 長度3，(2XXX)
    - 接著數字為2，沒有遞增所以沒有其他新的階梯數字。
  - 長度2，(22XX)，後是3，遞增，以222為開頭的階梯數字加上
    - 以2開頭且長度為2的階梯數字，有8個， $384 + 8 = 392$
  - 長度1，(223X)，後是2，數字變小，接下來遞增也不會有新的
  - 因此， $\leq 2232$ 的階梯數字有392個。



# 階梯數字

```
def gen(n):  
    d0 = [1 for i in range(9)]  
    data=[d0]  
    for i in range(n):  
        d=[1]  
        for j in range(1,9):  
            c = d[j-1]+data[i][j]  
            d.append(c)  
        data.append(d)  
    return data
```

```
N = str(2232)  
count=0  
length = len(N)  
data = gen(length)
```

```
for i in range(1, length):  
    count = count + sum(data[i-1])  
    print(count)    #長度1+長度2+長度3=219 個數字  
    #length  
    for i in range(0, int(N[0])-1):  
        count = count + data[length-1][8-i]  
        print(count)    #2232長度4，開頭2，開頭為1要計入  
        #length-1  
        for i in range(int(N[0])-1,int(N[1])-1):  
            count = count + data[length-2][8-i]  
            print(count)    #2232接著的數字為2，沒有遞增  
            #length-2  
            for i in range(int(N[1])-1,int(N[2])-1):  
                count = count + data[length-3][8-i]  
                print(count)    #2232再來3，因遞增，2開頭長度2階梯數字  
                #length-3  
                for i in range(int(N[1])-1,int(N[2])-1):  
                    count = count + data[length-3][8-i]  
                    print(count)    #2232再來2，數字變小，接著不會有階梯數字
```

# 階梯數字




## □ N為2479，累積數量

- 長度1 + 長度2 + 長度3 = 219 個數字
- 長度4，開頭2，開頭1要計入， $219 + \text{開頭為1且長度為4} = 384$
- 長度3，(2XXX)，後是4，遞增，以22, 23開頭的階梯數字
  - 以2開頭且長度為3的階梯數字，有36個， $384 + 36 = 420$
  - 以3開頭且長度為3的階梯數字，有28個， $420 + 28 = 448$
- 長度2，(24XX)，後是7，遞增，以244, 245, 246開頭階梯數字
  - 以4開頭且長度為2的階梯數字，有6個， $448 + 6 = 454$
  - 以5開頭且長度為2的階梯數字，有5個， $454 + 5 = 459$
  - 以6開頭且長度為2的階梯數字，有4個， $459 + 4 = 463$
- 長度1，(227X)，後是9，遞增，以2277, 2278開頭的階梯數字
  - 以7開頭且長度為1的階梯數字，有1個， $463 + 1 = 464$
  - 以8開頭且長度為1的階梯數字，有1個， $464 + 7 = 465$
  - 長度是1，若有計入，要加1， $465 + 1 = 466$

# 階梯數字

```
def gen(n):
    d0 = [1 for i in range(9)]
    data=[d0]
    for i in range(n):
        d=[1]
        for j in range(1,9):
            c = d[j-1]+data[i][j]
            d.append(c)
        data.append(d)
    return data

def countData(data, index, start, end):
    count=0
    for i in range(start-1, end-1):
        count = count + data[index-1][8-i]
    return(count)
```

```
def compute(N):
    count=0
    length = len(N)
    data = gen(length)
    for i in range(1, length):
        count = 
        #print(count)
        count = 
        #print(count)
    for i in range(length-1,0,-1):
        if (int(N[length-i-1])>int(N[length-i])):
            break
        count = count + countData(data, i, int(N[length-i-1]), int(N[length-i]))
    if (i==1):
        count = 
        #print('=>',count)
    print('----', count,'----')
```

```
compute('54321')    # 1875
compute('23456')    # 1365
compute('88888888') # 24301
compute('525')      # 184
compute('25')       # 22
compute('1234567891234567891234') # 18239779
compute('12345678912345678912345678912345') # 332267361
compute('2479')     # 466
```

# 線段覆蓋

□ 給定一些線段，求這些線段所覆蓋的長度

□ 輸入說明

- 第一列輸入一個正整數  $n$ : 代表共有  $n$  組測試案例。

- 接著的  $n$  列每一列是一個線段的兩端點，每一個端點是一個整數介於  $0 \sim 60000$  之間，端點之間以一個空格區隔，線段個數不超過 5000。

- 每一線段小的數字先輸入。

□ 輸出說明

- 輸出一個正整數，為這些線段所覆蓋的總長度，重疊的部分只能算一次。

# 線段覆蓋

❑ 全部使用list非常耗時 60000 \* 5000

```
def getLength(m):
    sumList = []
    for x in range(m):
        inputList = [int(num) for num in input().split()]
        if 0 <= inputList[0] <= 60000 and 0 <= inputList[1] <= 60000:
            rangeList = [i for i in range(inputList[0], inputList[1])]
            for i in rangeList:
                if i not in sumList:
                    sumList.append(i)
            else:
                print("error")
    return len(sumList)
```

# 線段覆蓋

```
def getInput(m):
    inputList=[]
    for i in range(m):
        inputData = input().split()
        inputList.append([int(inputData[0]), int(inputData[1])])
    inputList.sort()          #取得所有輸入線段後排序
    return inputList

m = int(input())
inputList = getInput(m)
count = 0                    #線段覆蓋加總
last = 0                    #最後統計線段覆蓋的點
for i in range(m):
    if inputList[i][0]>last:  # 分開的線段
        count = count + (inputList[i][1]-inputList[i][0])
        last = inputList[i][1]
    elif inputList[i][1]>last: # 重複的線段
        count = count + (inputList[i][1]-last)
        last = inputList[i][1] #最後統計的點
print(inputList)
print(count)
```

# 撲克牌花色判斷

- 1. 四種花色黑桃、紅心、磚塊、梅花，定義 S, H, D, C。
- 2. 牌面符號A, 2, ..., J, Q, K，點數2~10為2~10, A為14，J為11，Q為12，K為13，共有52張牌。
- 3. 花色大小：黑桃>紅心>方塊>梅花。
- 4. 輸入編碼規則為**花色+牌面符號**，例如 **S10** 表黑桃 10、**D7** 表磚塊 7，**CQ** 表梅花 Q。
- 5. 牌型由小到大如下：
  - 散牌：單一張牌單張，沒有任何花色牌型，編號0。
  - 一對：兩張數字一樣為 Pair，編號 1。
  - 兩對：2 組 Pair 的牌為 Two pair，編號 2。
  - 三條：三張一樣數字的為 Three of a Kind，編號 3。
  - 順子：數字連續的 5 張牌為 Straight，包括[2, 3, 4, 5, 6],..., [11, 12, 13, 14, 2], [12, 13, 14, 2, 3], [13, 14, 2, 3, 4], [14, 2, 3, 4, 5]，編號 4。

# 撲克牌花色判斷

- 同花：五張同一花色的牌為 Flush，編號 5。
- 葫蘆：Three of a Kind 加一個 Pair 為 Full House，編號 6。
- 四條：四張一樣數字為 Four of a Kind，編號 7。
- 同花順：數字連續的 5 張且花色一樣為 Straight Flush，編號 8。

## ○輸入說明：

- 1. 輸入5張撲克牌，判斷哪一類型的牌形編號(0~8)。
- 2. 檢查是否錯誤輸入，若錯誤，輸出"Error input"。
- 3. 要檢查是否重複發牌，若重複，輸出"Duplicate deal"。
- 4. 第一列輸入5個編碼由空格分開，表示5張撲克牌。

## ○輸出說明：

- 1. 輸出一個0~8整數，代表牌型編號；以「最大牌型輸出」。
- 2. 數字連續定義為：K(13) 和 A(14) 有相連，A(14) 和 2 有相連，依此類推。



# 撲克牌花色判斷

```
def cardPoint(playerCardPoint):
    pork=['A','2','3','4','5','6','7','8','9','10','J','Q','K']
    points=['14','2','3','4','5','6','7','8','9','10','11','12','13']
    index=pork.index(playerCardPoint)
    return int(points[index])

def straight(card):
    #11, 12, 13, 14, 2
    card.sort()
    #排序
    if(max(card)==14 and min(card)!=10): #有14沒有10表示會跨連續
        for i in range(5):
            #跨連續小於10均加 13後可連續
            if card[i]<10:
                card[i]=card[i]+13
    card.sort()
    #排序
    for i in range(4):
        #判斷連續數字
        if card[i]!=card[i+1]-1:
            return 0
    return 1
```

# 撲克牌花色判斷

```
def getGrade(card, flow):  
    #紀錄幾張卡花色相同  
    f = [0, 0, 0, 0, 0, 0]  
    #紀錄幾張卡點數相同  
    p = [0, 0, 0, 0, 0, 0]  
    #檢查幾張卡花色相同  
    for c in ['S', 'H', 'D', 'C']:  
        f[flow.count(c)] = f[ ] + 1  
    #檢查幾張卡點數相同  
    for i in range(2, 15):  
        p[card.count(i)] = p[ ] + 1
```

```
if straight(card) == 1 and f[5] == 1: #同花順  
    return 8  
if p[4] == 1: #四條  
    return 7  
if p[3] == 1 and p[2] == 1: #葫蘆  
    return 6  
if (f[5] == 1): #同花  
    return 5  
if straight(card) == 1: #順  
    return 4  
if p[3] == 1: #三條  
    return 3  
if p[2] == 2: #兩對  
    return 2  
if p[2] == 1: #一對  
    return 1  
return 0 #散牌
```

# 撲克牌花色判斷

```
def compute(cards):
    p = [cardPoint(c[1:]) for c in cards]
    f = [c[0] for c in cards]
    print(getGrade(p,f))

def test():
    print(getGrade([13, 2, 6, 4, 14], ['S','H','S','D','C'])) #0
    print(getGrade([ 8, 8, 6, 4, 2], ['S','S','S','D','H'])) #1
    print(getGrade([13, 13, 6, 14, 14], ['S','S','S','D','C'])) #2
    print(getGrade([13, 13, 6, 14, 13], ['S','S','S','D','C'])) #3
    print(getGrade([ 7, 8, 6, 4, 5], ['S','S','S','D','C'])) #4
    print(getGrade([14, 8, 6, 4, 5], ['S','S','S','S','S'])) #5
    print(getGrade([13, 13, 14, 14, 13], ['D','D','D','D','D'])) #6
    print(getGrade([13, 13, 13, 14, 13], ['D','D','D','D','D'])) #7
    print(getGrade([12, 2, 13, 3, 14], ['S','S','S','S','S'])) #8

compute(['HQ', 'DK', 'CA', 'D2', 'S3'])
compute(['H5', 'D5', 'C5', 'D10', 'S5'])
```

# 反階梯數字

```
def cmp(x, y, xLen, yLen):
    if (xLen>yLen):
        return -1
    for i in range(xLen-1,-1,-1):
        if (x[i]>y[i]):
            return -1;
        elif ((x[i]<y[i])):
            return 1
    return 0
```

```
def add(d, xLen):
    i=0
    index=1
    if d[1]>d[0]:
        d[0] = d[0] + 1
        return d, xLen
    for index in range(2, xLen, 1):
        if (d[index]>d[index-1]):
            d[index-1]=d[index-1]+1
            for i in range(0,index-1):
                d[i]=0
            return d, xLen
    if d[xLen-1]==9:
        d.append(1)
        for i in range(0, xLen):
            d[i] = 0
        xLen = xLen + 1
        return d, xLen
    d[xLen-1]=d[xLen-1]+1;
    for i in range(0,xLen-1):
        d[i] = 0
    return d, xLen
```

# 反階梯數字

```
def test(inputX, A):
    inputX = [int(i) for i in inputX]
    A = [int(i) for i in A]
    i = count = 0
    xLen = yLen = 0
    y = [1,0]
    inputR = inputX[::-1]
    AR = A[::-1]
    target = sum(AR[1::2])
    xLen = len(inputR)
    yLen = len(y)
    while (cmp(inputR, y, xLen, yLen)<0):
        y, yLen = add(y, yLen)
        x = y[::-1]
        count = sum(x[::2])
        if (count == target):
            print(x)
        i = i + 1
```

```
def main():
    test('987654321','493584437776333')
    #test("9876543219876", "1234567891234");
    main()
```

# 數字穿插

- 給定一串數字，將數字重新排序，數字的左右兩側不能是自己。
- 輸出
  - 符合上述條件中，最小的數字。
  - 例如1 2 2 3，因左右兩側不能是自己，2的左右不能是2，符合條件:1232, 2132, 2123, 3212，
  - 其中最小數字，是1232。

# 數字穿插

```
def check(x, y):
    x = str(x)
    y = str(y)
    xLen = len(x)
    for i in range(xLen-1):
        if x[i]==x[i+1]:
            return 0
    for i in range(10):
        if x.count(str(i))!=y.count(str(i)):
            return 0
    return 1
```

```
def f(n, m):
    for i in range(n, m+1):
        if check(i, m)==1:
            print(i)
            break
```

```
#f(1122,2211)
f(11112233,33221111)
```

# 黃金分配

- N塊黃金( $N < 15$ )，每一塊黃金重 $1 \sim X$ 公克，平均分給A, B, C三個人，每人最多不得拿M塊( $M < N$ )，有幾種分法。
  - $N=5$ ，重 $= [1, 2, 3, 4, 5]$ ，每人最多拿2塊，6種分法
    - $[A, B, C] = \{([1, 4], [2, 3], [5]), ([1, 4], [5], [2, 3]), ([2, 3], [1, 4], [5]), ([2, 3], [5], [1, 4]), ([5], [2, 3], [1, 4]), ([5], [1, 4], [2, 3])\}$



# 黃金分配 - 迴圈版

```
def check(x, value, M):
    return (sum(x[0])==value and sum(x[1]) ==value and sum(x[2])==value
            and len(x[0])<=M and len(x[1])<=M and len(x[2])<=M)

def process(num, a, N, value, M):
    answer = [[],[],[ ]]
    for i in range(N):
        index = num%3
        num=num//3
        answer[index].append(a[i])
    if check(answer, value, M)==True:
        return answer
    else:
        return None

def push(result, answer):
    d=(tuple(sorted(answer[0])),tuple(sorted(answer[1])),tuple(sorted(answer[2])))
    result.add(d)
```

# 黃金分配 - 迴圈版

```
def dist(a, M):
    result = set()
    N = len(a)
    value = sum(a)//3
    count = 1
    for i in range(N):
        count = count*3
    for i in range(count):
        answer=process(i, a, N, value, M)
        if answer!=None:
            push(result, answer)
    print('-'*30, '\nn=', len(result))
    for i in result: print(i)
```

```
#dist([1, 2, 3, 4, 5], 3)
#dist([1, 2, 4, 6, 8], 2)
#dist([2, 4, 6, 8, 10, 12], 2)
#dist([2, 4, 6, 8, 10, 12, 15], 3)
dist([1, 2, 3, 4, 5, 6, 7, 8, 9],4)
dist([9, 1, 2, 3, 4, 5, 6, 7, 8, 9],5)
```

# 黃金分配 - 遞迴版

```
def dist(result, a, x, y, z, value, M):
    n = len(a)
    if n==0:
        if check(x, y, z, value, M):
            push(result, x, y, z)
        return True
    t = a[1:]
    dist(result, t, x+a[1], y, z, value, M)
    dist(result, t, x, y+a[1], z, value, M)
    dist(result, t, x, y, z+a[1], value, M)
    return True

def test(a, M):
    result = set()
    x, y, z = [], [], []
    value = sum(a)//3
    if value*3 == sum(a): dist(result, a, x, y, z, value, M)
    print('-'*30, '\nn=', len(result))
    for i in result: print(i)
```

# 賓果遊戲 V1

## □ N人賓果遊戲

- 每位玩家各輸入一個九宮格，九個數字 $n_1, n_2, \dots, n_9$ ，且 $n_i \neq n_j$ 當 $i \neq j$ ，而 $1 \leq n_i, n_j \leq 9$ ， $1 \leq i, j \leq 9$ 。
- 電腦從1~9整數中選三個數字 $C_1$ 、 $C_2$ 、 $C_3$ ，且 $C_i \neq C_j$ 當 $i \neq j$ ，而 $1 \leq C_i, C_j \leq 9$ ，且 $1 \leq i, j \leq 3$ 。
  - 這三個數字，只在一位玩家九宮格中，成一條水平線，如圖( $n_1, n_2, n_3$ )、垂直線如圖( $n_1, n_4, n_7$ )，或對角線如圖( $n_1, n_5, n_9$ )、( $n_3, n_5, n_7$ )

$n_1$	$n_2$	$n_3$
$n_4$	$n_5$	$n_6$
$n_7$	$n_8$	$n_9$

## □ 比賽結果

- 第一位玩家贏，第二位玩家贏、和平手。

Input:

A 1 2 3 4 5 6 7 8 9  
B 5 2 8 4 7 9 6 1 3  
1 5 9

1	2	3
4	5	6
7	8	9

5	2	8
4	7	9
6	1	3

# 賓果遊戲V1

- data =[1 2 3 4 5 6 7 8 9]
- selected 挑選的數字，對應到mData 【0/1檢測矩陣】
- 回傳mData 【0/1檢測矩陣】
- N=3, selected = [1, 5, 9]

0	0	1
1	1	1
1	0	1

```
def mapData(data, selected, N):  
    mData = [[0 for i in range(N)] for j in range(N)] #初始化陣列  
    for x in selected: #針對挑選數字  
        index=data.index(x) #找出九宮格相對位置  
        mData[index//N][index%N]=1 #轉出設定對應0/1檢測矩陣  
    print(mData)  
    return mData #回傳0/1檢測矩陣
```

# 賓果遊戲V1

## ❑ 判斷0/1檢測矩陣連線

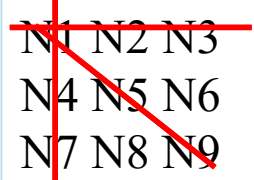
- 0/1檢測矩陣，成一條水平線，如圖n1, n2, n3、垂直線如圖n1, n4, n7，或對角線如圖n1, n5, n9，

```
def checkBingo(data, N): #0/1矩陣
    count = 0
    for i in range(N): #對角線1
        if data[i][i]==1:
            count = count + 1
    if count==N: return True
```

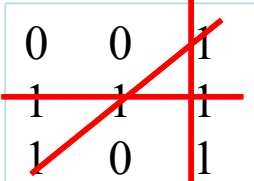
```
    for i in range(N): #對角線2
        if data[i][N-i-1]==1:
            count = count + 1
    if count==N: return True
```

```
    for i in range(N): #3條水平線
        count = 0
        for j in range(N):
            if data[i][j]==1:
                count = count + 1
        if count==N: return True
```

```
    for i in range(N): #3條垂直線
        count = 0
        for j in range(N):
            if data[j][i]==1:
                count = count + 1
        if count==N: return True
    return False
```



N1	N2	N3
N4	N5	N6
N7	N8	N9



0	0	1
1	1	1
1	0	1

# 賓果遊戲V1

## □ 測試資料

```
def test01():
    mData =[[1,0,1],
            [1,1,1],
            [0,1,0]]
    print(checkBingo(mData, 3))          #測試【判斷0/1檢測矩陣連線】
    data =[1, 3, 5, 7, 9, 2, 4, 6, 8]    #九宮格輸入
    mData = mapData(data, [1, 2, 3], 3)  #轉成【0/1檢測矩陣】
    print(checkBingo(mData, 3))          #【判斷0/1檢測矩陣連線】
```

1	0	1
1	1	1
1	1	0

```
def checkBingo(data, N):
    count=0
    for i in range(N):
        if data[i].count(1)==N: count +=1 # row
        h =[data[j][i] for j in range(N)] #column
        if h.count(1)==N: count +=1

    h =[data[j][j] for j in range(N)]
    if h.count(1)==N: count +=1
    h =[data[j][N-j-1] for j in range(N)]
    if h.count(1)==N: count +=1
    return count
```

# 賓果遊戲V2

- 檢測八條連線是否完成，**store** = [0, 0, 0, 0, 0, 0, 0, 0]
  - 編號0, 1, 2 (0~N-1) row 水平線
  - 編號3, 4, 5 (N~2\*N-1) column 垂直線
  - 編號6, 7 (2\*N, 2\*N+1) 左上右下、右上左下
  - 針對每一個選中的數字**num**，是否有在八條連線中，+1

```
def checkBingo2(data, store, num, N):  
    row, column = data.index(num)//N, data.index(num)%N #算出x, y 座標位置  
    store[row] += 1  
    store[column+N] += 1  
    if row == column:                                     #符合左上右下連線  
        store[N*2] += 1  
    if row==(N-1-column):                                 #符合右上左下連線  
        store[N*2+1] += 1
```



# 賓果遊戲V2

□ 逐一檢測八條連線是否完成，

- **store** = [0, 0, 0, 0, 0, 0, 0, 0]，紀錄八條連線狀態，任一數字=N(3)，擇有連線成功
- data 九宮格設定數字，N=3
- selected 挑選的數字

```
def compute(data, selected, N):  
    store = [0 for i in range(2*N+2)]  
    for num in selected:  
        checkBingo2(data, store, num, N)  
        print(store)  
        if N in store: return 'Bingo!'  
    return 'Sorry~'  
  
def test02():  
    data =[1, 3, 5, 7, 9, 2, 4, 6, 8]  
    print(compute(data, [1, 2, 3, 5], 3))
```

#初始化八條連線紀錄  
#針對挑選數字  
#逐一檢測八條連線是否完成  
#任一數字=N(3)，擇有連線成功

# 賓果遊戲V2

## □ AB兩人遊戲

```
def compute(dataA, dataB, selected, N, M):
    storeA = [0 for i in range(2*N+2)]
    storeB = [0 for i in range(2*N+2)]
    for num in selected:
        checkBingo(dataA, storeA, num, N)
        checkBingo(dataB, storeB, num, N)
        print(storeA, storeB)
        if (N in storeA) and (N in storeB):
            return         
        elif (N in storeA):
            return 'A Win'
        elif (N in storeB):
            return 'B Win'
    return 'Tie'
```

```
def test():
    inputData = input().split()
    N, M = int(inputData[0]), int(inputData[1])
    dataA = input().split()
    dataB = input().split()
    selected = input().split()
    print(compute(dataA,         , selected, N, M))
```

# 字母組合

- 給一字串s，求 n 個字母組合，輸入 'abcde', 2，['ab', 'ac',...]
  - 假設  $f(s, n)$  可以解問題，則問題可以分解成
    - $['a' + f('bcde', n-1)] + ['' + f('bcde', n)]$
- 遞迴  $f(s, n)$ 
  - 結束條件，回傳list(因為會有許多組合)
    - $n==0$ ，回傳空字串(list)
    - $n==len(s)$ ，回傳s(list)
  - 一般條件
    - s[0]要取，回傳： $s[0] + f(s[1:], n-1)$ ，
      - 這一層已經取一個字母， $n-1$ ，字串少第一個字母s[1:]
      - 要把取的字母s[0]加進答案
    - s[0]不要取，回傳： $f(s[1:], n)$ ，
      - 這一層不取任一個字母， $n$ ，字串少第一個字母s[1:]

# 字母組合

## □ Code

```
def findCombi(data, n):  
    if n==len(data): return[data]  
    elif n==0: return [""]  
    s0 = findCombi(data[1:], n)  
    s1 = [data[0]+s for s in findCombi(data[1:], n-1)]  
    return sorted(s0+s1)  
  
print('sol=',findCombi('abcde', 5))  
print('sol=',findCombi('abcde', 4))  
print('sol=',findCombi('abcde', 3))  
print('sol=',findCombi('abcde', 2))  
print('sol=',findCombi('abcde', 1))
```

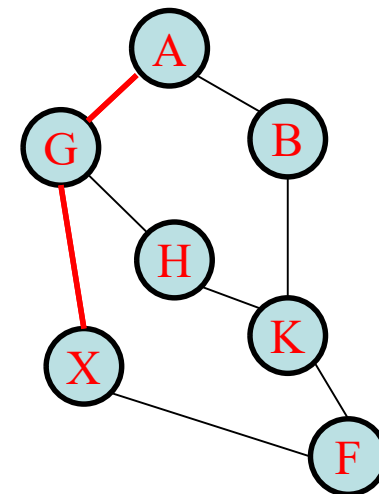
```
sol= ['abcde']  
sol= ['abcd', 'abce', 'abde', 'acde', 'bcde']  
sol= ['abc', 'abd', 'abe', 'acd', 'ace', 'ade', 'bcd', 'bce', 'bde', 'cde']  
sol= ['ab', 'ac', 'ad', 'ae', 'bc', 'bd', 'be', 'cd', 'ce', 'de']  
sol= ['a', 'b', 'c', 'd', 'e']
```

# 數字組合

- 給定 $n$ 個0~9數字，任取其中 $m$ 個數，求可以組合出幾個整數，其中，那些數，每一位數字加總可以被3整除
  - 2, 3, 5, 7, 8, 9，任取4數

# 部落旅遊

- 小明要到沙哈拉沙漠中的各部落旅遊，某些部落之間有安全通道可以走。
  - 給定N個部落間有通道資料，請問要從A部落到X部落，最少需經過幾個通道
    - 通道資料，A B, G H, A G, H K, K F, F X, G X, B K
- 建立地圖(dict, 點:相鄰點集合)
  - {'A': ['B', 'G'], 'B': ['A', 'K'], 'G': ['H', 'A', 'X'], 'H': ['G', 'K'], 'K': ['H', 'F', 'B'], 'F': ['K', 'X'], 'X': ['F', 'G']}
- 搜尋最少通道數A-X
  - A-B-K-F-X =>4
  - A-B-K-H-G-X
  - A-G-H-K-F-X=>5
  - A-G-X =>2
- 找出路徑 A-G-X



# 部落旅遊

## □ 建立地圖(dict, {點:[相鄰點集合/鄰居], ...})

```
def addNeighbour(data, x, y):
    neighbour = data[x] if x in data.keys() else [] # 若x 節點存在則取出x的鄰居，否則[]
    if y not in neighbour: neighbour.append(y)      # 加入 x 鄰居 y
    data[x]=neighbour                               # 加入 x 節點，和新的鄰居

def addPair(data, pair):
    addNeighbour(data, pair[0], pair[1])           # 加入 0 節點 1 鄰居
    addNeighbour(data, pair[1], pair[0])           # 加入 1 節點 0 鄰居

def test01():
    data=dict()
    x = [['A','B'], ['G','H'], ['A','G'], ['H','K'], ['K','F'], ['F','X'], ['G','X'], ['B','K']]
    for pair in x: addPair(data, pair)
    print(data)
```

A	B
G	H
A	G
H	K
K	F
F	X
G	X
B	K

```
{'A': ['B', 'G'], 'B': ['A', 'K'], 'G': ['H', 'A', 'X'], 'H': ['G', 'K'], 'K': ['H', 'F', 'B'], 'F': ['K', 'X'], 'X': ['F', 'G']}
```

# 部落旅遊

## □ 廣度搜尋，最少通道數A-X

```
def findNeighbour(data, source, target):  
    passedNodes = dict() # 拜訪過的節點，{節點:level}  
    stack = [[source,0]] # 待拜訪節點，[開頭點: level=0]  
    while True:  
        if len(stack)==0: return -1 # 待拜訪節點，找不到  
        currentNode = stack.pop(0) # 取出一個待拜訪節點  
        currentNodeName = currentNode[0] # 取出該節點名稱  
        level = currentNode[1] # 取出該節點 level  
        if currentNodeName==target: # 找到最終節點  
            #path = findPath(data, passedNodes, currentNodeName, level-1) #找路徑  
            return level # 回傳經過幾層  
        passedNodes[currentNodeName]=level # 不是最終節點，存入(拜訪過節點)  
        for node in data[currentNodeName]: # 針對每一個相鄰節點，存入(待拜訪節點)  
            if node not in passedNodes.keys(): # 只存(未拜訪過節點)  
                stack.append([node, level+1])  
  
def test01():  
    data=dict()  
    x = [['A','B'], ['G','H'], ['A','G'], ['H','K'], ['K','F'], ['F','X'], ['G','X'], ['B','K']]  
    for pair in x: addPair(data, pair)  
    print(findNeighbour(data, 'A', 'X')) # 2  
    print(findNeighbour(data, 'F', 'A')) # 3
```

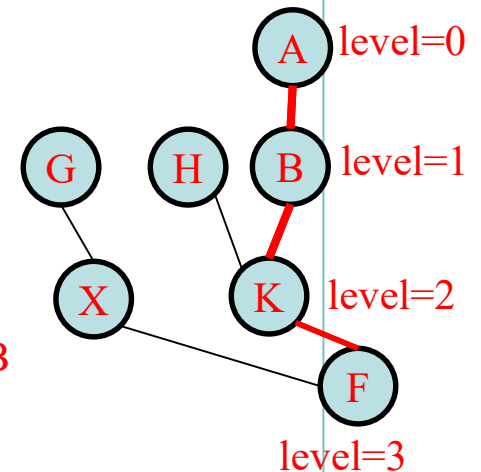
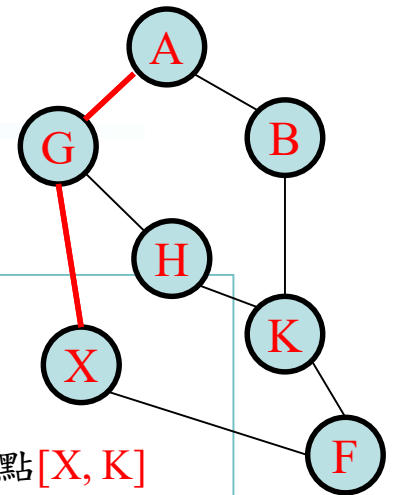


# 部落旅遊findPath(v1)

□ 從目標節點倒回去搜尋，F(0)-K(1)-B(2)-A(3)

```
def findPrevNode(data, passedNodes: dict, targetNodeName, level):#找上一層
    nodeCandidate = []
    for nodeName, nodeLevel in passedNodes.items():
        #走過的節點
        if nodeLevel==level: nodeCandidate.append(nodeName)    # F所有上一層節點[X, K]
    for nodeName in nodeCandidate:
        if targetNodeName in data[nodeName]: return nodeName    # K 相鄰節點是 F，回傳K
```

```
def findPath(data, passedNodes, currentNodeName, level):
    path = []
    while level>=0:
        #找到level=0 停止
        nodeName=findPrevNode(data, passedNodes, currentNodeName, level)
        #找 F 上一個節點 K
        path.append(nodeName)
        #level-1 往上一節點找
        level = level-1
        #找到上一節點K，繼續往上找B
        currentNodeName = nodeName
    print(path)
    return path
```



# 部落旅遊findPath(v2)

□ 從目標節點倒回去搜尋，F(0)-K(1)-B(2)-A(3)

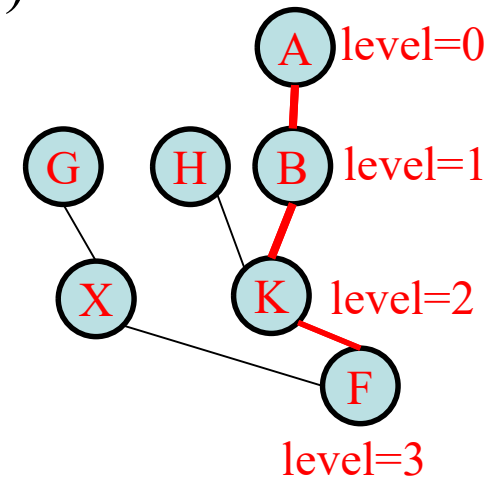
○ 04，F的相鄰點list()，data[currentNodeName]

○ 和走過的節點list()，list(passedNodes.keys())

○ 變成集合set()

○ 交集&

○ 轉成串列list()，只有一個元素[0]



```
01 def findPath(data, passedNodes, currentNodeName, source):
02     path = []
03     while currentNodeName!=source:                                # 從 F 一直找到原始點 A 停止
04         currentNodeName=list(set(data[currentNodeName])&set(list(passedNodes.keys())))[0]
05         path.append(currentNodeName)
06     print(path)
07     return path
```

□ 假設中途一定要到某部落旅遊，則最少要如何走

# 會議安排 I

## □ 問題

- 某公司有數間會議室 A, B, C, D...，24小時開放各單位登記舉辦會議，每個會議有不同的使用時間。由於會議眾多，可能發生時間衝突，總務部門需從這些活動中，選取時間上不衝突的活動讓他們如願使用會議室。公司希望所有會議室能做最有效的利用：
  - 登記排出最多的加總使用時數。
  - 登記排出最多會議的使用數量。

## □ 演算步驟

- 計算所有活動組合，1, 2, 3, 12, 13, 23, 123。
- 計算任一組合活動中，是否可以安排進2間會議室。
- 計算任一組活動中，使用會議室的總時數。
- 計算任一組活動中，總活動數。
- 輸出所有活動組合，找出最大的指標(總時數或總活動數)

# 會議安排 I

## □ 輸入說明

- 第一行輸入兩個整數M, N，M是會議室間數，N是申請舉辦會議個數。
- 其後N行，每一行有三個整數，代表每一個申請會議的編號、開始時間、結束時間。開始與結束時間以8~18代表早上八點到下午六點。

## □ 輸出說明

- 從所有的申請會議中，計算「最長使用總時數」—輸出所有可使用會議室的會議編號，讓所有會議室的總使用時數最長。
- 從所有的申請會議中，計算「最多會議使用數」。如果有多組解，依編號，由小至大依序列出所有解。

# 會議安排 I

- 計算所有活動組合，1, 2, 3, 12, 13, 23, 123。

```
def generateCode(x:int, N:int):  
    code = ""  
    for i in range(N):  
        if x>0:  
            code = code + str(x%2)  
            x = x//2  
        else:  
            code = code + str('0')  
    return code
```

```
000  
100  
010  
110  
001  
101  
011  
111
```

# 會議安排 I

- 計算任一組合活動中，是否可以安排進2間會議室。

```
def checkValidRoom(M:int, N:int, data:list, code:str):  
    section=[]          #所有使用節數時段  
    timeCount=[]        #24節數時段使用會議室量  
    for i in range(len(code)):  
        if code[i]=='1':  
            section = section + data[i]  
    timeCount=[section.count(i) for i in range(24)]  
    if max(timeCount)>M:  
        return False, 0  
    else:  
        return True, timeCount
```

# 會議安排 I

- 計算任一組合法活動中，使用會議室的總時數。找最大。

```
def computeMaxHours(M:int, N:int, data:list):  
    maxHours = 0  
    for x in range(2**N):  
        code = generateCode(x, N)  
        flag, timeCount = checkValidRoom(M, N, data, code)  
        #print(flag, count)  
        if flag==True and sum(timeCount)>maxHours:  
            maxHours=sum(timeCount)  
    return (maxHours)
```

# 會議安排 I

- 處理輸入，轉換節數時段表達，找最大總時數。

```
def process():
    x=input().split()
    M, N = int(x[0]), int(x[1])
    data = []
    for i in range(N):
        x=input().split()
        data.append([int(x[1]), int(x[2])])
    td = [[t for t in range(d[0], d[1])] for d in data] #轉換節數時段表達
    print(computeMaxHours(M, N, td))
```

```
#轉換節數時段表達
#data = [[1,3], [1, 3], [3, 4], [1, 5]]
#data = [[1,2], [1,2], [3], [1,2,3,4]]
```



# 會議安排 I

- 處理輸入，轉換節數時段表達，找最大總時數。

```
def process():
    x=input().split()
    M, N = int(x[0]), int(x[1])
    data = []
    for i in range(N):
        x=input().split()
        data.append([int(x[1]), int(x[2])])
    td = [[t for t in range(d[0], d[1])] for d in data] #轉換節數時段表達
    print(computeMaxHours(M, N, td))
```

```
#轉換節數時段表達
#data = [[1,3], [1, 3], [3, 4], [1, 5]]
#data = [[1,2], [1,2], [3], [1,2,3,4]]
```

# 會議安排 I

## □ 測試資料

輸入範例	輸出範例
1	Selective Activities:room: 3, 7, 11
1,1,4	Total Utilization Hours:12
2,3,5	Selective Activities:room: 3, 9, 11
3,0,6	Total Utilization Hours:12
4,5,7	Selective Activities:room: 5, 9, 10, 11
5,3,8	Total Utilization Hours:12
6,5,9	
7,6,10	Solutions with most utilized activities:
8,8,11	Selective Activities:room: 2, 4, 9, 10, 11
9,8,12	Total Utilization Hours:11
10,2,3	Selective Activities:room: 2, 4, 8, 10, 11
11,12,14	Total Utilization Hours:10
-1	
2	
1,1,4	
2,3,5	
3,0,6	
4,5,7	
5,3,8	
6,5,9	
7,6,10	
8,8,11	
9,8,12	
10,2,3	
11,12,14	
-1	

# 會議安排 II

- 0~23時段，每一會議可以安排在不同會議室，會議有編號、時段，會議室有編號。安排在同一個會議室的會議時間不能有衝突。

# 會議安排-遞迴

## □ 0~23時段。

```
# room 一間會議室目前有的課程
# [[101, 2, 4], [102, 5, 7]]
# rooms 包含許多間有排課程的會議室
# [[[101, 2, 4], [102, 5, 7]], [[103, 2, 4], [104, 5, 7]]]
檢查一間會議室 room，是否可以排進課程 course
def checkOneRoom(room, course):
    if len(room)==0: return True
    for item in room:
        if course[1] in range(item[1], item[2]): return False
        if (course[2]-1) in range(item[1], item[2]): return False
        if item[1] in range(course[1], course[2]): return False
        if (item[2]-1) in range(course[1], course[2]): return False
    return True
```

# 會議安排-遞迴

## □ 0~23時段。

```
# n: 會議室個數
# data: 會議室排課程的所有資訊
# room 所有會議室排課程資訊
def checkRooms(n, rooms, courses, data):
    if len(courses)==0:
        data.append(rooms)
        rooms = [[] for i in range(n)]
        return True
    for room in rooms:
        flag = checkOneRoom(room, courses[0])
        if flag==True:
            room.append(courses[0])
            if checkRooms(n, rooms, courses[1:], data)==False:
                room.remove(courses[0])
    return False
```

# 會議安排-遞迴

## □ 0~23時段。

```
def main(n, courses):
    #print(checkOneRoom([[101, 2, 4], [102, 5,7]], [103, 7,7]))
    rooms = [[] for i in range(n)]
    data=[]
    print(checkRooms(n, rooms, courses, data), rooms)
    print('--')
    for i in data:
        print(i)

#main(2, [[101, 2, 4], [102, 5,7]])
#main(2, [[101, 2, 5], [102, 5,7]])
#main(2, [[101, 2, 5], [102, 4,7]])
main(2, [[101, 0, 2], [102, 3,4], [103, 5,6], [104, 7,10], [105, 0,20]])
```

# 課程教室安排 I

- 0~23時段，**每一門課不同時段可以安排在不同教室**，教室有人數容量限制

```
#產生【某一課程組合】編碼 code
#N = 3，000 100 010 110 001 101 011 111
# 1 代表選取此課程排入教室
def genCoursesCode(x:int, N:int):
    code = ""
    for i in range(N):
        if x>0:
            code = code + str(x%2)
            x = x//2
        else:
            code = code + str('0')
    return code
```

```
#sections={節次0:[[課程人數1,課程編號1],[課程人數,課程編號],..], 節次1:[]}
#加入【某一課程組合 code】的一個課程資訊 course 到節數 (sections)
# course = [編號,人數,節數,...]
#最後排序，將人數多的放前面
def addCourse(sections, course):
    for i in range(2,len(course)):
        s = sections.get(course[i], [])
        sections[course[i]]= s + [[course[1], course[0]]]
    for k, v in sections.items():
        v.sort(reverse=True)
```

# 課程教室安排 I

```
#產生【某一課程組合】編碼 code 之課程節數資料 sections
# code = 1100
#sections={節次0:[[課程人數1,課程編號1],[課程人數2,課程編號2]], 節次1:[...], ...}
def genSections(courses:list, code:str):
    sections={}          #所有使用節數時段
    for i in range(len(code)):
        if code[i]=='1': addCourse(sections,courses[i])
    return sections

#檢查某一個節數(0, 1, 2, ...)【某一課程組合】section 是否可排入教室 rooms
# code = 1100
#sections={節次0:[[課程人數1,課程編號1],[課程人數2,課程編號2]], 節次1:[...], ...}
#若可以排入，則記錄配對資訊
#match=[節次課程編號1, 教室編號1,節次課程編號2, 教室編號2,...]
#人數多的課程先排入教室容量大的，依序往下排，人數不足則 False
def checkOneSectionValid(section, rooms):
    match=[]
    lenS, lenR = len(section), len(rooms)
    if lenS>lenR:
        return False, match
    for i in range(lenS):
        if section[i][0]>rooms[i][1]:      return False, match
        else:
            match += [section[i][1], rooms[i][0]]
    return True, match
```



# 課程教室安排 I

```
#檢查所有節數【某一課程組合】sections 是否可排入教室rooms
#若沒有 False，則記錄配對資訊
def checkAllSectionsValid(sections:list, rooms:list):
    matches = {}
    match=[]
    for k, v in sections.items():
        flag, match = checkOneSectionValid(v, rooms)
        if flag==False: return False, matches
        elif len(match)>0:matches[k]=match
    return True, matches

#產生【某一課程組合】合法的排程資訊，data=[[code1, match1, sections1], ...]
#1~2**N LOOP
#產生【某一課程組合】編碼 code
#產生【某一課程組合】編碼 code，其節次資訊
#檢查該節次資訊是否合法(能排入教室)，若合法則紀錄配對資訊
#將合法排程資訊 [code1, match1, sections1] 加入 data
def genAllValidData(N:int, courses:list, rooms: list):
    data=[]
    for x in range(2**N):
        code = genCoursesCode(x, N)
        sections = genSections(courses, code)
        flag, matches =checkAllSectionsValid(sections, rooms)
        if flag==True: data.append([code, matches, sections])
    return data
```

# 課程教室安排 I

#計算【某一課程組合】所有節次 sections 可排入的總課程時數 count

```
def computeHours(sections):
```

```
    count = 0
```

```
    for k, v in sections.items():
```

```
        if len(v)>0: count = count + len(v)
```

```
    return count
```

```
def computeNum(code):
```

```
    return code.count('1')
```

#計算【所有課程組合】所有節次可排入的總課程時數，最大值、最大值排程資訊

```
#data = [code, matches, section]
```

```
def computeMaxHours(data: list):
```

```
    hour, maxHours = 0, 0
```

```
    num, maxNum = 0, 0
```

```
    hourKey, numKey = [], []
```

```
    for d in data:
```

```
        hour = computeHours(d[2])
```

```
        if hour>maxHours:
```

```
            maxHours = hour
```

```
            hourKey = d
```

```
        num = computeNum(d[0])
```

```
        if num>maxNum:
```

```
            maxNum = num
```

```
            numKey = d
```

```
    return maxHours, hourKey, maxNum, numKey
```

# 課程教室安排 I

```
# course = [編號,人數,節數, ...]
def process():
    rooms=[]
    courses = []
    x =input().split()
    M, N = int(x[0]), int(x[1])
    for i in range(M):
        r=input().split()
        rooms+=[[int(r[0]), int(r[1])]]
    rooms.sort(key=lambda s: s[1], reverse=True)
    print(rooms)
    for i in range(N):
        x =input().split()
        courses.append([int(x[0]), int(x[1]), int(x[2]), int(x[3])])
    courses = [[d[0], d[1]]+[t for t in range(d[2], d[3])] for d in courses]
    data = genAllValidData(N, courses, rooms)
    maxHours, hourKey, maxNum, numKey = computeMaxHours(data)
    print(maxHours)
    print(hourKey[0], print(hourKey[1], print(hourKey[2]))
    print(maxNum)
    print(numKey[0], print(numKey[1], print(numKey[2]))

process()
```

# 課程教室安排 II

- 0~23時段，**同一門課所有時段必須安排在相同教室**，教室有人數容量限制

```
#產生【某一課程組合】編碼
def genCoursesCode(x:int, M:int, N:int):
    code = ""
    for i in range(N):
        if x > 0:
            code = code + str(x%(M+1))
            x = x//(M+1)
        else:
            code = code + str('0')
    return code

#roomCourses={教室0:[[課程人數1,課程編號1],[課程人數,課程編號],...], 教室1:[], ...}
# courses = [[編號1,人數1,節數, ...], ...]
# rooms = [[編號1, 人數1],...]
#產生【某一課程組合】編碼之教室課程資料
def genRoomsCourses(rooms:list, courses:list, code:str):
    roomsCourses={} #所有使用教室課程資料
    for i in range(len(rooms)):
        for j in range(len(code)):
            if code[j]==str(i+1):
                r = roomsCourses.get(rooms[i][0], [])
                roomsCourses[rooms[i][0]] = r + [courses[j]]
    return roomsCourses
```

# 課程教室安排 II

```
#檢查【某一課程組合】是否可排入某一教室
def checkOneRoom(rooms:list, roomId:str, courses:list):
    content = []
    quantity = 0
    for r in rooms:
        if r[0]==roomId: quantity = r[1]
    for i in range(len(courses)):
        if quantity<courses[i][1]: return False
    for c in courses:
        content += c[2:]
    for i in range(24):
        if content.count(i)>1: return False
    return True

#match=[節次課程編號1, 教室編號1,節次課程編號2, 教室編號2,...]
#檢查所有節數【某一課程組合】是否可排入教室
def checkAllRoomCourseValid(roomsCourses:dict, courses:list, rooms:list):
    if len(roomsCourses)==0: return False
    for rid, courses in roomsCourses.items():
        if checkOneRoom(rooms, rid, courses)==False: return False
    return True
```

# 課程教室安排 II

#產生【某一課程組合】合法的排程資訊，data=[code, match, sections]

```
def genAllValidData(M: int, N: int, courses: list, rooms: list):
```

```
    data=[]
```

```
    for i in range((M+1)**N):
```

```
        code = genCoursesCode(i, M, N)
```

```
        roomsCourses = genRoomsCourses(rooms, courses, code)
```

```
        flag = checkAllRoomCourseValid(roomsCourses, courses, rooms)
```

```
        if flag==True:
```

```
            print(code)
```

```
            print(roomsCourses)
```

```
            data.append([code, roomsCourses])
```

```
    return data
```

#計算【某一課程組合】所有節次可排入的總課程時數

```
def computeHours(roomsCourses):
```

```
    count = 0
```

```
    for rid, courses in roomsCourses.items():
```

```
        for c in courses:
```

```
            if len(c[2:])>0: count = count + len(c[2:])
```

```
    return count
```

```
def computeNumCourse(roomsCourses):
```

```
    count = 0
```

```
    for rid, courses in roomsCourses.items():
```

```
        count = count + len(courses)
```

```
    return count
```

# 課程教室安排 II

#計算【所有課程組合】所有節次可排入的總課程時數，最大值、最大值排程資訊

```
#data = [code, roomsCourses]
```

```
def computeMax(data: list):
```

```
    hour, maxHours = 0, 0
```

```
    num, maxNum = 0, 0
```

```
    hourKey, numKey = [], []
```

```
    for d in data:
```

```
        hour = computeHours(d[1])
```

```
        if hour > maxHours:
```

```
            maxHours = hour
```

```
            hourKey = d[1]
```

```
        num = computeNumCourse(d[1])
```

```
        if num > maxNum:
```

```
            maxNum = num
```

```
            numKey = d[1]
```

```
    return maxHours, hourKey, maxNum, numKey
```

# 課程教室安排 II

```
# course = [編號,人數,節數,...]
# courses = [[編號1,人數1,節數,...],...]
# rooms = [[編號1, 人數1],...]
def process():
    rooms=[]
    courses = []
    x=input().split()
    M, N = int(x[0]), int(x[1])
    for i in range(M):
        r=input().split()
        rooms+=[[int(r[0]), int(r[1])]]
    rooms.sort(key=lambda s: s[1], reverse=True)
    print(rooms)
    for i in range(N):
        x=input().split()
        courses.append([int(x[0]), int(x[1]), int(x[2]), int(x[3])])
    courses = [[d[0], d[1]]+[t for t in range(d[2], d[3])] for d in courses]
    data = genAllValidData(M, N, courses, rooms)
    maxHours, hourKey, maxNum, numKey = computeMax(data)
    print(maxHours, hourKey, maxNum, numKey)

process()
```



# 分數四則運算

- 給定兩分數A及B，以 分子/分母 或 整數(分子/分母) 表示，
- 給定一個運算符號 (+, -, \*, /) 表示兩個分數的運算行為，
- 計算兩個分數的四則運算：加法、減法、乘法、除法，並輸出兩個分數經四則運算後的結果，
- 結果需轉換為最簡分數，若為假分數，需換算成帶分數，若為整數則直接輸出整數，
- 輸出格式為 分子/分母 或 整數(分子/分母)。例如：
  - 1/2
  - 4/2
  - +

# 分數四則運算

```
01 # 計算最大公因素，約分
02 def gcd(m, n):
03     while (m > 0 and n > 0):
04         if (m > n):
05             m = m % n
06         else:
07             n = n % m
08     return (m if m > n else n)
09
10 #操作算術運算
11 def oper(x, y, op):
12     numerator, denominator = 0, 0
13     denominator = x[1]*y[1]
14     if op=='*':
15         numerator = x[0]*y[0]
16     elif op=='+':
17         numerator = x[0]*y[1]+x[1]*y[0]
18     elif op=='-':
19         numerator = x[0]*y[1]-x[1]*y[0]
20     return [numerator, denominator]
```

# 分數四則運算

```
01 #取得分子numerator、分母denominator，type轉成整數
02 def getNum(s):
03     start, end = 0, len(s)                #設定處理開始 start 和結束end 位址
04     sign, im = 1, 0                      #設定正負號 sign
05
06     if s[0]=='-':                         #若輸入第一個符號為-，負號，處理到1
07         sign=-1
08         start = 1
09     rest = s[start:end]                   #其他需要處理的分數字串部分
10
11     if '(' in s:                          #若有左括號，會有帶分數im、和右括號
12         im = int(s[start:end-1].split('(')[0]) #取出帶分數的整數 im
13         rest = s[start:end-1].split('(')[1]   #取出其他部分 - 真分數部分
14
15     denominator = int(rest.split('/')[1])    #取出分母
16     numerator = sign*(im*denominator+int(rest.split('/')[0])) #取出分子
17     return (numerator, denominator)
```

# 分數四則運算

```
01 def output(s):
02     answer=""
03     if s[1]==0: return 'Error'
04     print(s[0], s[1], gcd(abs(s[0]),s[1]))
05     numerator =abs(s[0])//gcd(abs(s[0]),s[1]) #取出分子，正負號在分子部位
06     denominator=s[1]//gcd(abs(s[0]),s[1]) #取出分母
07     im = numerator//denominator #取出帶分數整數部位
08     numerator = numerator%denominator #取出真分數的分子部位
09
10     if s[0]<0: #負數有負號
11         answer+='-'
12
13     if im>0: #有帶分數
14         answer+='{}({}/{})'.format(im, numerator, denominator)
15     else: #沒有帶分數
16         answer+='{}/{}'.format(numerator, denominator)
17     return answer
```

# 分數四則運算

```
01 print(getNum('-10(5/17)')[0])
02 print(getNum('10(5/17)'))
03 print(getNum('5/17'))
04 print(getNum('-5/17'))
05 print(oper(getNum('-5/7'), getNum('-5/8'),'*'))
06 print(oper(getNum('-5/7'), getNum('-5/8'),'+'))
07 print(oper(getNum('-5/7'), getNum('-5/8'),'-'))
08 print(output(oper(getNum('-5/7'), getNum('-5/8'),'+')))
```

# 倉庫貨架存放貨物

- 物流公司有  $N$  件貨物( $1 \leq N \leq 150$ )，編號為 $1 \sim N$ ，倉庫中有 $M$ 層貨架( $1 \leq M \leq 15$ )，貨架編號由下到上為 $1 \sim M$ 。貨架存放貨物方式：
  - 由貨物編號1開始存放於貨架，貨物由底層往上疊放於貨架。
  - 存放規則
    - 由底層貨架開始選擇。
    - 選擇有存放貨物的貨架，即將存放的貨物編號，須與貨架最頂端貨物編號相加為平方數。
    - 若不符合上述規則，選擇空的貨架存放。
  - 若貨物存放無法符合規則，表示倉庫無法存放。
- 計算所有貨物是否能存放倉庫，是輸出"True"，並輸出所有貨架上貨物編號，否則輸出"False"。

# 倉庫貨架存放貨物

## □ 輸出所有貨架上貨物編號

- 每個貨架上貨物編號，由小到大排序
- 貨架排序，依貨物數量由小到大排序，若數量相同，則依編號數字總和由小到大排序，例如(2, 3, 4) -> (1, 5, 6)。

## □ 輸入

- N=7
- M=3

## □ 輸出：

- True
- 2 7
- 4 5
- 1 3 6

# 倉庫貨架存放貨物

```
import math
#即將存放的貨物編號，須與貨架最頂端貨物編號相加為平方數
def checkin(data, key):
    for i in range(len(data)):
        if math.sqrt(data[i][len(data[i])-1]+key)%1==0:
            data[i].append(key)
            return True
    return False
# 存放貨物流程，m件貨物，n個貨架
def process(m, n):
    data=[[1]]
    num = 1
    for i in range(2,m+1):
        if num>n: return False, data
        if checkin(data, i)==False:
            data.append([i])
            num = num + 1
    return True, data
```

#檢查目前非空貨架  
#符合規則  
#目前貨物放入貨架i  
#編號1的貨物放入貨架1  
#目前使用貨架數量  
#處理貨物編號2~m  
#貨架數量超過 n 無法完成  
#第i個貨物是否可以放入非空貨架  
#無法放入，則使用空的貨架  
#使用貨架數+1



# 倉庫貨架存放貨物

```
#輸出排序
def output(data):
    # 依貨架數量、貨物編號大小排序
    data.sort(key=lambda x:(len(x), -1*x))
    for item in data:
        print(item)

def main(m,n):
    flag, data = process(m, n)
    if flag==True:                #貨物可以處理，輸出排序
        output(data)
    else:
        print(False)             #無法處理

main(23, 6)
main(150, 15)
```

---

# END

---

