In [145]:

```
#第一题
#首先生成a，b，c三个需要输入的数值
a=int(input(""))
b=int(input(""))
c=int(input(""))
#进行第一个判断，a是否大于b
if a>b:
    #如果a大于b，进行下一步判断，b是否大于c
    if b>c:
        #如果b>c，则按照此途径计算结果
        d=a+b-10*c
        print(d)
    else:
        #如果b不大于c，则进一步判断a和c的大小
        if a>c:
            #如果a>c，则按照此途径进行计算
            d=a+c-10*b
            print(d)
        else :
            #如果a<c，则按照此途径计算
            d=c+a-10*b
            print(d)
#对a小于b的情况进行补充
else :
    if b>c:
        d
    else:
        d=c+b-10*a
        print(d)
```

```
10
5
1
5
```

In [152]:

```
#第二题
import math
import numpy as np
#因为矩阵的形式比较好写输出，所以先声成一个1*n的数值均为0的矩阵
n=int(input(""))
a=np.zeros((1,n))
#因为矩阵为1*n，这里对i的定义为1-n，因为a[0,0]即F（1）=1，不参与循环计算
for i in range(1,n):
    #因为矩阵的i值要比F（x）的x小一位，比如a[0,0]=F(1),a[0,1]=F(2),所以对于ceil运算的值应相应调整为
    b=math.ceil((i+1)/3)
    #定义矩阵第一个值a[0,0]即F(1)为1
    a[0,0]=1
    #后续数值按照方程进行计算
    a[0,i]=a[0,b-1]+2*(i+1)
print(a)
```

```
100
[[  1.    5.    7.   13.   15.   17.   21.   23.   25.   33.   35.   37.   41.   43.
   45.   49.   51.   53.   59.   61.   63.   67.   69.   71.   75.   77.   79.   89.
   91.   93.   97.   99.  101.  105.  107.  109.  115.  117.  119.  123.  125.  127.
  131.  133.  135.  141.  143.  145.  149.  151.  153.  157.  159.  161.  169.  171.
  173.  177.  179.  181.  185.  187.  189.  195.  197.  199.  203.  205.  207.  211.
  213.  215.  221.  223.  225.  229.  231.  233.  237.  239.  241.  253.  255.  257.
  261.  263.  265.  269.  271.  273.  279.  281.  283.  287.  289.  291.  295.  297.
  299.  305.]]
```

In [151]:

```
#第三题
import numpy as np
#整体思路：我手中有n个筛子，我把他们依次投出去，然后记录他们的和的结果，如果这个结果满足我的期望x，我
#然后将我预期的结果（10-60）进行for循环,直到得到所有期望对应的路径数，代码中x为预期的求和；x_为当前的
#初始值为0；实际上就是在10个色子投出后所有可能的排列组合(6^10次)中找出你所期望的总和x的出现次数。
def Find_number_of_ways(x, n, x_, n_, Number_of_ways):
    #开始投色子，当前投了n_个色子
    n_ = n_ + 1
    #第n_个色子投到了结果i
    for i in range(1,7):
        #如果筛子已经投了n个，且得到的值也是我们预期的x,在Number_of_ways加1
        if n_ == n:
            if x_ + i == x:
                Number_of_ways = Number_of_ways + 1
        else:
            #如果色子没投完，继续
            Number_of_ways = Find_number_of_ways(x, n, x_ + i, n_, Number_of_ways)
    return Number_of_ways
# 输入色子的总数
n = 10
#定义一个用于储存路径总数的list
Number_of_ways = []
#对我们预期的结果（10-60）进行for循环
for x in range(10, 61):
    #记录每一个x对应的结果
    Number_of_ways.append(Find_number_of_ways(x, n, 0, 0,  0))
print(Number_of_ways)
print(np.max(Number_of_ways))  # 输出路径最多的结果对应的路径数
print(np.argmax(Number_of_ways))  # 该路径数对应的位置，结果为25，则表明第25个结果路径数最多

#最终结果：出现路径数对多的10次投掷点数的总和为x=35，对应路径数为4395456次
```

```
[1, 10, 55, 220, 715, 2002, 4995, 11340, 23760, 46420, 85228, 147940, 243925, 38347
0, 576565, 831204, 1151370, 1535040, 1972630, 2446300, 2930455, 3393610, 3801535, 41
21260, 4325310, 4395456, 4325310, 4121260, 3801535, 3393610, 2930455, 2446300, 19726
30, 1535040, 1151370, 831204, 576565, 383470, 243925, 147940, 85228, 46420, 23760, 1
1340, 4995, 2002, 715, 220, 55, 10, 1]
4395456
25
```

In [166]:

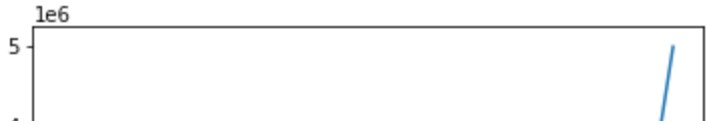
```
#第四题
import numpy as np
import matplotlib.pyplot as plt
#生成一个数组中所有子集的方法，函数来源：网上搜索
def ArraySubSet(Array):
    result = [[]]
    size = len(Array)
    for i in range(size):
        for j in range(len(result)):
            result.append(result[j]+[Array[i]])
    return result

#定义一个用于存储所有生成的子集的平均数的加和的list，用于作图和展示所有n对应的子集平均值的加和
total_sum_anerages=[]
#这里n的数值太大的话，电脑太差劲、带不动，跑不出结果，只能先用20代替一下，100是真的跑不出来
for n in range(1,21):
    #生成我们需要的数组
    Array = np.random.randint(0,11,n)
    #通过之前的代码生成构建数组的所有子集
    subset = ArraySubSet(Array)
    #查询所有子集的数量，定义i的范围
    b=len(subset)
    #定义一个用于存放子集平均值的list
    res=[]
    #此处定义1-b是因为在生成的subset中，第一项是[],没有数值
    for i in range(1,b):
        #对每一个子集求平均值
        res.append(np.mean(list(subset[i][:])))
        #对所有生成的平均值进行加和
    total_sum_anerages.append(np.sum(res))
print(total_sum_anerages)
n_number = range(1,21)
plt.plot(n_number, total_sum_anerages)
plt.xlabel("n_number")
plt.ylabel("total_sum_anerages")
plt.show
#发现最终结果随着n数值的增加大致呈现指数型增长趋势
```

```
[5.0, 13.5, 49.0, 71.25, 99.20000000000002, 357.0, 635.0000000000001, 1051.875000000
0002, 2271.1111111111113, 4501.200000000001, 10607.18181818182, 22181.25, 43475.3076
9230769, 78404.35714285713, 196601.99999999997, 335866.875, 508863.8823529412, 11505
16.5, 2400682.578947369, 4980731.250000002]
```

Out[166]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [167]:

```
#第五题
import numpy as np
#解题思路，以一个3*3的数值均为1的矩阵（所有路径走的通）举例，从a[0,0]出发，先判断a[0,1]是否走的通，
#所以此时开始尝试往下走，判断a[1,2]和a[2，2]，如果成功抵达a[2,2],则记count为1；随后开始return，首先r
#继续return至a[0，1]，发现可以向下走至a[1,1],然后a[1,1]可以往右走，直到走到终点或者遇到0后再次返回，
#走到终点或者遇到0后返回上一节点再次判断。通过count记录抵达终点的次数。
#代码中a为按照5.1要求生成的矩阵，row_cur为当前所在的行，col_cur为当前所在的列，count为可行的路径数量
def Count_path(a, row_cur, col_cur, count):
    [N,M] = np.shape(a)
    #如果成功走到终点，就在res里加一
    if row_cur == N-1 and col_cur == M-1:
        count = count + 1
    else:
        #在走到终点之前，尝试往右走，如果值为1，就继续走
        if row_cur < N-1 and a[row_cur+1,col_cur] == 1:
            count = Count_path(a, row_cur+1, col_cur, count)
        #在走到终点之前，尝试往下走，如果值为1，就继续走
        if col_cur < M-1 and a[row_cur,col_cur+1] == 1:
            count = Count_path(a, row_cur, col_cur+1, count)
    return count
#输入设定矩阵的行和列
N = int(input(""))
M = int(input(""))
#建立一个用于存储路径数的list
count = []
#进行1000次循环
for i in range(0,1000):
    a = np.random.randint(0, 2, size=(N, M))  # [row][col]
    a[0,0] = 1
    a[-1,-1] = 1
    count.append(Count_path(a, 0, 0, 0))
#对获得的路径数求平均值
mean_conut = np.mean(count)
print(mean_conut)
```

```
10
8
0.224
```

In [ ]: