In [9]:
```
#PS1-1
import random
a = random.randint(1,1000)
b = random.randint(1,1000)
c = random.randint(1,1000)

def sequence(a, b, c):
    if a > b:
        if b > c:
            x, y, z = a, b, c
            print(x, y, z)
            print(x+y-10*z)
        else:
            if a > c:
                x, y, z = a, c, b
                print(x, y, z)
                print(x+y-10*z)
            else:
                x, y, z = c, a, b
                print(x, y, z)
                print(x+y-10*z)
    else:
        if b > c:
            if a > c:
                x, y, z = b, a, c
                print(x, y, z)
                print(x+y-10*z)
            else:
                x, y, z = b, c, a
                print(x, y, z)
                print(x+y-10*z)
        else:
            x, y, z = c, b, a
            print(x, y, z)
            print(x+y-10*z)
sequence(a, b, c)
```
```
196 55 48
-229
```
In [10]:
```
#PS1-1
import random
a = 5
b = 15
c = 10

def sequence(a, b, c):
    if a > b:
        if b > c:
            x, y, z = a, b, c
            print(x, y, z)
            print(x+y-10*z)
        else:
            if a > c:
                x, y, z = a, c, b
                print(x, y, z)
                print(x+y-10*z)
            else:
                x, y, z = c, a, b
                print(x, y, z)
                print(x+y-10*z)
    else:
        if b > c:
            if a > c:
                x, y, z = b, a, c
                print(x, y, z)
                print(x+y-10*z)
            else:
                x, y, z = b, c, a
                print(x, y, z)
                print(x+y-10*z)
        else:
            x, y, z = c, b, a
            print(x, y, z)
            print(x+y-10*z)
sequence(a, b, c)
```
```
15 10 5
-25
```
In [45]:
```
# PS1-2
import math
x = int(input("请输入任意一个非1的正整数x: "))
n = x
result = 0


def F(x):
    if x == 1:
        return 1
    else:
        return F(math.ceil(x/3)) + 2*x

while n != 1:
    result += 2*n
    n = math.ceil(n/3)     #倒推计算所有的2*x
result += 1                #倒退终点，加F(1)

print("F(x)的结果是: " + str(result))
```
```
F(x)的结果是: 33
```
In [73]:
```
#PS1-3-1
x = int(input("请输入一个介于10和60之间的整数（含10和60）: "))
n = 0

for a in range(1, 7):
    for b in range(1, 7):
        for c in range(1, 7):
            for d in range(1, 7):
                for e in range(1, 7):
                    for f in range(1, 7):
                        for g in range(1, 7):
                            for h in range(1, 7):
                                for i in range(1, 7):
                                    for j in range(1, 7):
                                        if a + b + c + d + e + f + g + h + i + j == x:
                                            n += 1
print("随机掷 10 个骰子，十个点数和为" + str(x) + "的组合共有" + str(n) + "种。")
```

随机掷 10 个骰子，十个点数和为35的组合共有4395456种。

In [74]:
```python
#PS1-3-2

for x in range(10, 61):          # 外层循环：x = 10 ... 60
    n = 0
    for a in range(1, 7):        # 内层循环：得到每个x的点数组合数目
        for b in range(1, 7):
            for c in range(1, 7):
                for d in range(1, 7):
                    for e in range(1, 7):
                        for f in range(1, 7):
                            for g in range(1, 7):
                                for h in range(1, 7):
                                    for i in range(1, 7):
                                        for j in range(1, 7):
                                            if a + b + c + d + e + f + g + h + i + j == x:
                                                n += 1
    print(f"掷 10 个骰子，点数和为 {x} 的组合共有 {n} 种。")    #结果全部算出来后，显示x=35时可能性最多：4395456
# 结果是非常标准的正态分布
```

```
掷 10 个骰子，点数和为 10 的组合共有 1 种。
掷 10 个骰子，点数和为 11 的组合共有 10 种。
掷 10 个骰子，点数和为 12 的组合共有 55 种。
掷 10 个骰子，点数和为 13 的组合共有 220 种。
掷 10 个骰子，点数和为 14 的组合共有 715 种。
掷 10 个骰子，点数和为 15 的组合共有 2002 种。
掷 10 个骰子，点数和为 16 的组合共有 4995 种。
掷 10 个骰子，点数和为 17 的组合共有 11340 种。
掷 10 个骰子，点数和为 18 的组合共有 23760 种。
掷 10 个骰子，点数和为 19 的组合共有 46420 种。
掷 10 个骰子，点数和为 20 的组合共有 85228 种。
掷 10 个骰子，点数和为 21 的组合共有 147940 种。
掷 10 个骰子，点数和为 22 的组合共有 243925 种。
掷 10 个骰子，点数和为 23 的组合共有 383470 种。
掷 10 个骰子，点数和为 24 的组合共有 576565 种。
掷 10 个骰子，点数和为 25 的组合共有 831204 种。
掷 10 个骰子，点数和为 26 的组合共有 1151370 种。
掷 10 个骰子，点数和为 27 的组合共有 1535040 种。
掷 10 个骰子，点数和为 28 的组合共有 1972630 种。
掷 10 个骰子，点数和为 29 的组合共有 2446300 种。
掷 10 个骰子，点数和为 30 的组合共有 2930455 种。
掷 10 个骰子，点数和为 31 的组合共有 3393610 种。
掷 10 个骰子，点数和为 32 的组合共有 3801535 种。
掷 10 个骰子，点数和为 33 的组合共有 4121260 种。
掷 10 个骰子，点数和为 34 的组合共有 4325310 种。
掷 10 个骰子，点数和为 35 的组合共有 4395456 种。
掷 10 个骰子，点数和为 36 的组合共有 4325310 种。
掷 10 个骰子，点数和为 37 的组合共有 4121260 种。
掷 10 个骰子，点数和为 38 的组合共有 3801535 种。
掷 10 个骰子，点数和为 39 的组合共有 3393610 种。
掷 10 个骰子，点数和为 40 的组合共有 2930455 种。
掷 10 个骰子，点数和为 41 的组合共有 2446300 种。
掷 10 个骰子，点数和为 42 的组合共有 1972630 种。
掷 10 个骰子，点数和为 43 的组合共有 1535040 种。
掷 10 个骰子，点数和为 44 的组合共有 1151370 种。
掷 10 个骰子，点数和为 45 的组合共有 831204 种。
掷 10 个骰子，点数和为 46 的组合共有 576565 种。
掷 10 个骰子，点数和为 47 的组合共有 383470 种。
掷 10 个骰子，点数和为 48 的组合共有 243925 种。
掷 10 个骰子，点数和为 49 的组合共有 147940 种。
掷 10 个骰子，点数和为 50 的组合共有 85228 种。
掷 10 个骰子，点数和为 51 的组合共有 46420 种。
掷 10 个骰子，点数和为 52 的组合共有 23760 种。
掷 10 个骰子，点数和为 53 的组合共有 11340 种。
掷 10 个骰子，点数和为 54 的组合共有 4995 种。
掷 10 个骰子，点数和为 55 的组合共有 2002 种。
掷 10 个骰子，点数和为 56 的组合共有 715 种。
掷 10 个骰子，点数和为 57 的组合共有 220 种。
掷 10 个骰子，点数和为 58 的组合共有 55 种。
掷 10 个骰子，点数和为 59 的组合共有 10 种。
掷 10 个骰子，点数和为 60 的组合共有 1 种。
```

In [6]:
```python
#PS1-4-1
import random
numbers = []

def Random_integer(N):
    for i in range(N):
        numbers.append(random.randint(0, 10))
    return numbers

# test
result = Random_integer(66)
print(result)
```

```
[4, 9, 5, 8, 5, 1, 4, 7, 2, 5, 7, 9, 1, 3, 3, 1, 1, 4, 6, 3, 2, 4, 8, 9, 10, 0, 7, 6, 2, 2, 0, 7, 10, 6, 4, 1, 5, 3, 8, 5, 5, 7, 5, 3, 1, 2, 1, 5, 8, 4, 1, 1, 8, 7, 6, 10, 7, 5,
10, 2, 7, 5, 0, 3, 3, 1]
```

In [92]:
```python
#PS1-4-2

import itertools

def Sum_averages(array_n):
    n = len(array_n)
    if n == 0:
        return 0
    total = 0

    for m in range(1, n + 1):
        for sum_m in itertools.combinations(array_n, m):        # itertools.combinations(arr, m) 得到 arr 中所有大小为 m 的组合（子集）
            total += sum(sum_m) / m                              # 在上一级结果中加入含 m 个元素的子集的平均值
    return total

result02 = Sum_averages([1, 2, 3])
print(result02)
```

```
14.0
```

In [ ]:
```python
#PS1-4-3——硬循环

import itertools

# 定义计算所有非空子集平均值之和的函数
def Sum_averages(array_n):
    n = len(array_n)
    if n == 0:
        return 0
    total = 0
    for m in range(1, n + 1):
        for comb in itertools.combinations(array_n, m):
            total += sum(comb) / m
    return total

# 存放每个 N 的结果
Total_sum_averages = []

# 外层循环：N 从 1 到 100
for N in range(1, 101):
    arr = list(range(1, N + 1))   # [1, 2, 3, ..., N]
    total = Sum_averages(arr)     # 调用函数
    print("N =", N, ", Sum_averages =", round(total, 2))
    #结果呈现指数型增长，N>25时极慢，远不及下面的公式法
```

```
N = 1 , Sum_averages = 1.0
N = 2 , Sum_averages = 4.5
N = 3 , Sum_averages = 14.0
N = 4 , Sum_averages = 37.5
N = 5 , Sum_averages = 93.0
N = 6 , Sum_averages = 220.5
N = 7 , Sum_averages = 508.0
N = 8 , Sum_averages = 1147.5
N = 9 , Sum_averages = 2555.0
N = 10 , Sum_averages = 5626.5
N = 11 , Sum_averages = 12282.0
N = 12 , Sum_averages = 26617.5
N = 13 , Sum_averages = 57337.0
N = 14 , Sum_averages = 122872.5
N = 15 , Sum_averages = 262136.0
N = 16 , Sum_averages = 557047.5
N = 17 , Sum_averages = 1179639.0
N = 18 , Sum_averages = 2490358.5
N = 19 , Sum_averages = 5242870.0
N = 20 , Sum_averages = 11010037.5
N = 21 , Sum_averages = 23068661.0
N = 22 , Sum_averages = 48234484.5
N = 23 , Sum_averages = 100663284.0
N = 24 , Sum_averages = 209715187.5
```

In [101…

```
#PS1-4-3——公式计算
def Sum_averages(arr):
    n = len(arr)
    if n == 0:
        return 0
    return sum(arr) * ((2 ** n - 1) / n)

Total_sum_averages = []
N = 1

while N <= 100:
    arr = list(range(1, N + 1))
    total = Sum_averages(arr)
    print("N =", N, " Sum_averages =", round(total, 2))
    N = N + 1
```

```
N = 1  Sum_averages = 1.0
N = 2  Sum_averages = 4.5
N = 3  Sum_averages = 14.0
N = 4  Sum_averages = 37.5
N = 5  Sum_averages = 93.0
N = 6  Sum_averages = 220.5
N = 7  Sum_averages = 508.0
N = 8  Sum_averages = 1147.5
N = 9  Sum_averages = 2555.0
N = 10  Sum_averages = 5626.5
N = 11  Sum_averages = 12282.0
N = 12  Sum_averages = 26617.5
N = 13  Sum_averages = 57337.0
N = 14  Sum_averages = 122872.5
N = 15  Sum_averages = 262136.0
N = 16  Sum_averages = 557047.5
N = 17  Sum_averages = 1179639.0
N = 18  Sum_averages = 2490358.5
N = 19  Sum_averages = 5242870.0
N = 20  Sum_averages = 11010037.5
N = 21  Sum_averages = 23068661.0
N = 22  Sum_averages = 48234484.5
N = 23  Sum_averages = 100663284.0
N = 24  Sum_averages = 209715187.5
N = 25  Sum_averages = 436207603.0
N = 26  Sum_averages = 905969650.5
N = 27  Sum_averages = 1879048178.0
N = 28  Sum_averages = 3892314097.5
N = 29  Sum_averages = 8053063665.0
N = 30  Sum_averages = 16642998256.5
N = 31  Sum_averages = 34359738352.0
N = 32  Sum_averages = 70866960367.5
N = 33  Sum_averages = 146028888047.0
N = 34  Sum_averages = 300647710702.5
N = 35  Sum_averages = 618475290606.0
N = 36  Sum_averages = 1271310319597.5
N = 37  Sum_averages = 2611340115949.0
N = 38  Sum_averages = 5360119185388.5
N = 39  Sum_averages = 10995116277740.0
N = 40  Sum_averages = 22539988369387.5
N = 41  Sum_averages = 46179488366571.0
N = 42  Sum_averages = 94557999988714.5
N = 43  Sum_averages = 193514046488554.0
N = 44  Sum_averages = 395824185999337.5
N = 45  Sum_averages = 809240558043113.0
N = 46  Sum_averages = 1653665488175080.5
N = 47  Sum_averages = 3377699720527848.0
N = 48  Sum_averages = 6896136929411048.0
N = 49  Sum_averages = 1.4073748835532774e+16
N = 50  Sum_averages = 2.871044762448689e+16
N = 51  Sum_averages = 5.8546795155816424e+16
N = 52  Sum_averages = 1.1934539012531811e+17
N = 53  Sum_averages = 2.431943798780068e+17
N = 54  Sum_averages = 4.9539595901075456e+17
N = 55  Sum_averages = 1.0088063165309911e+18
N = 56  Sum_averages = 2.0536414300809462e+18
N = 57  Sum_averages = 4.1793404541998203e+18
N = 58  Sum_averages = 8.502796096475496e+18
N = 59  Sum_averages = 1.7293822569102705e+19
N = 60  Sum_averages = 3.5164105890508833e+19
N = 61  Sum_averages = 7.148113328562451e+19
N = 62  Sum_averages = 1.4526810958046272e+20
N = 63  Sum_averages = 2.9514790517935283e+20
N = 64  Sum_averages = 5.995191823955604e+20
N = 65  Sum_averages = 1.2174851088648304e+21
N = 66  Sum_averages = 2.47186370587708e+21
N = 67  Sum_averages = 5.017514388048998e+21
N = 68  Sum_averages = 1.0182602728687672e+22
N = 69  Sum_averages = 2.0660353362554698e+22
N = 70  Sum_averages = 4.19110025354681e+22
N = 71  Sum_averages = 8.500259669165361e+22
N = 72  Sum_averages = 1.7236637662474205e+23
N = 73  Sum_averages = 3.4945511973235375e+23
N = 74  Sum_averages = 7.083549724304468e+23
N = 75  Sum_averages = 1.4355994107923721e+24
N = 76  Sum_averages = 2.9089777534477015e+24
N = 77  Sum_averages = 5.893513370621317e+24
N = 78  Sum_averages = 1.1938142468694463e+25
N = 79  Sum_averages = 2.4178516392292583e+25
N = 80  Sum_averages = 4.896149569439248e+25
N = 81  Sum_averages = 9.91319172083996e+25
N = 82  Sum_averages = 2.0068168605602844e+26
N = 83  Sum_averages = 4.061990753905154e+26
N = 84  Sum_averages = 8.220695573379478e+26
N = 85  Sum_averages = 1.6634819277897297e+27
N = 86  Sum_averages = 3.3656494818071276e+27
N = 87  Sum_averages = 6.808670216069592e+27
N = 88  Sum_averages = 1.3772082937049856e+28
N = 89  Sum_averages = 2.7853650883921056e+28
N = 90  Sum_averages = 5.63262717874848e+28
N = 91  Sum_averages = 1.13890483614255e+29
N = 92  Sum_averages = 2.3025684730708073e+29
N = 93  Sum_averages = 4.6546545477130305e+29
N = 94  Sum_averages = 9.40834429856889e+29
N = 95  Sum_averages = 1.901475900342344e+30
N = 96  Sum_averages = 3.8425658819418204e+30
N = 97  Sum_averages = 7.764359926397905e+30
N = 98  Sum_averages = 1.5687176177824337e+31
N = 99  Sum_averages = 3.169126500570574e+31
N = 100  Sum_averages = 6.4016355311525585e+31
```

In [5]:

```python
# PS1-5-1

import random

# 手动输入行数和列数
N = int(input("Enter number of rows (N): "))
M = int(input("Enter number of columns (M): "))

# 创建一个N行M列的矩阵，用随机0或1填充
matrix = []
for i in range(N):
    row = []
    for j in range(M):
        row.append(random.randint(0, 1))
    matrix.append(row)

# 将两个角的数字修改为 1
matrix[0][0] = 1
matrix[N-1][M-1] = 1    # -1
```

```
# 输出矩阵
print(matrix)
```

```
[[1, 1, 0, 0, 0], [0, 1, 0, 1, 0], [0, 0, 0, 0, 1], [0, 1, 1, 0, 0], [0, 1, 0, 1, 1], [1, 0, 0, 1, 1]]
```

In [90]:

```
# PS1-5-2
import random

# 手动输入行数和列数
N = int(input("Enter number of rows (N): "))
M = int(input("Enter number of columns (M): "))

# 生成一个N行、M列的矩阵
def matrix_function(N, M):
    matrix_01 = []
    for i in range(N):
        row = []
        for j in range(M):
            row.append(random.randint(0, 1))
        matrix_01.append(row)

    # 更改两角的数字
    matrix_01[0][0] = 1
    matrix_01[N - 1][M - 1] = 1
    return matrix_01

matrix_01 = matrix_function(N, M)

for row in matrix_01:
    print(*row)

# 定义计算路径数目的函数（矩阵）
def Count_path(matrix):
    # 检查矩阵
    if len(matrix) == 0 or len(matrix[0]) == 0:
        return 0
    n = len(matrix)
    m = len(matrix[0])

    if matrix[0][0] == 0 or matrix[n - 1][m - 1] == 0:
        return 0

    dp = []        # 定义一个动态规划表，与原矩阵同样大小的矩阵，用于记录到达所有（i，j）的路径的数目

    i = 0
    while i < n:
        dp.append([0] * m)     # 初始化 dp
        j = 0
        while j < m:
            if matrix[i][j] == 1:  # 只在可走格计算
                if i == 0 and j == 0:
                    dp[i][j] = 1  # 起点
                else:
                    from_up = 0            # 先假设从上方和左方来的路径数都是 0
                    from_left = 0

                    if i > 0:                         # 如果不是第一行，可以从上面或左边来
                        from_up = dp[i - 1][j]
                    if j > 0:
                        from_left = dp[i][j - 1]

                    dp[i][j] = from_up + from_left    # 当前格子的路径数 = 上方路径 + 左方路径
            j += 1
        i += 1

    return dp[n - 1][m - 1]    # 返回到达终点（右下角）的路径数


# 计算并输出路径总数
total_paths = Count_path(matrix_01)
print("\n从左上角到右下角的路径总数：", total_paths)
```

```
1 1 1 1
1 0 1 1
1 1 0 1
0 0 1 1

从左上角到右下角的路径总数： 2
```

In [ ]: