

作业

求杨辉三角第n行第k列的值

算法1

计算到m行，打印出k项

```
# 求m行k个元素
# m行元素有m个，所以k不能大于m
# 这个需求需要保存m行的数据，那么可以使用一个嵌套机构[[[]],[[]],[[]]]
m = 5
k = 4
triangle = []
for i in range(m):
    # 所有行都需要1开头
    row = [1]
    triangle.append(row)
    if i == 0:
        continue
    for j in range(1,i):
        row.append(triangle[i-1][j-1] + triangle[i-1][j])
    row.append(1)
print(triangle)
print("-----")
print(triangle[m-1][k-1])
print("-----")
```

测试效率：120 μ s \pm 2.81 μ s per loop (mean \pm std. dev. of 7 runs, 10000 loops each)

算法2

根据杨辉三角的定理：第n行的m个数($m > 0$ 且 $n > 0$)可表示为 $C(n-1, m-1)$ ，即为从n-1个不同元素中取m-1个元素的组合数。

组合数公式：有m个不同元素，任意取 $n (n \leq m)$ 个元素，记作 $c(m, n)$ ，组合数公式为：

$$C(m, n) = m! / (n!(m - n)!)$$

```

# m行k列的值, C(m-1,k-1)组合数
m = 9
k = 5
# c(n,r) = c(m-1,k-1) = (m-1)!/((k-1)!(m-r)!)
# m 最大
n = m - 1
r = k - 1
d = n - r
targets = [] # r ,n-r, n
factorial = 1
# 可以加入k为1或者m的判断, 返回1
for i in range(1,n+1):
    factorial *= i
    if i == r:
        targets.append(factorial)
    if i == d:
        targets.append(factorial)
    if i == n:
        targets.append(factorial)
#print(targets)
print(targets[2]/(targets[0]*targets[1]))

```

109 μ s \pm 2.51 μ s per loop (mean \pm std. dev. of 7 runs, 10000 loops each)

`i==r`、`i==n`、`i==d` 这三个条件不要写在一起, 因为它们有可能两两相等。

算法说明：一趟到n的阶乘算出所有阶乘值。

转置矩阵

有一个方阵, 左边方阵, 求其转置矩阵

```

1 2 3      1 4 7
4 5 6 ==>> 2 5 8
7 8 9      3 6 9

```

规律：对角线不动, $a[i][j] \Leftrightarrow a[j][i]$, 而且到了对角线, 就停止, 去做下一行, 对角线上的元素不动。

```

# 定义一个方阵

```

```
# 1 2 3      1 4 7
# 4 5 6 ==>> 2 5 8
# 7 8 9      3 6 9
matrix = [[1,2,3], [4,5,6], [7,8,9]]
print(matrix)
count = 0
for i, row in enumerate(matrix):
    for j, col in enumerate(row):
        if i < j :
            temp = matrix[i][j]
            matrix[i][j] = matrix[j][i]
            matrix[j][i] = temp
            count += 1
print(matrix)
print(count)
```

```
# 方法2
matrix = [[1,2,3,10],[4,5,6,11],[7,8,9,12],[1,2,3,4]]
length = len(matrix)
count = 0
for i in range(length):
    for j in range(i): # j<i
        arr[i][j],arr[j][i] = arr[j][i],arr[i][j]
        count += 1
print(matrix)
print(count)
```

有一个任意矩阵，求其转置矩阵

```
1 2 3      1 4
4 5 6 <==>> 2 5
           3 6
```

这样一个矩阵，但不是方阵。

```
enumerate(iterable[, start]) -> iterator for index, value of iterable
```

返回一个可迭代对象，将原有可迭代对象的元素和从start开始的数字配对。

算法1

过程就是，扫描matrix第一行，在tm的第一列从上至下附加，然后再第二列附加
举例，扫描第一行1,2,3，加入到tm的第一列，然后扫描第二行4,5,6，追加到tm的第二列

```
# 定义一个矩阵，不考虑稀疏矩阵
# 1 2 3      1 4
# 4 5 6 ==>> 2 5
#           3 6

import datetime
matrix = [[1,2,3], [4,5,6]]
#matrix = [[1,4],[2,5],[3,6]]

tm = []
count = 0
for row in matrix:
    for i, col in enumerate(row):
        if len(tm) < i + 1: # matrix有i列就要为tm创建i行
            tm.append([])

        tm[i].append(col)
        count += 1

print(matrix)
print(tm)
print(count)
```

算法2

思考：

能否一次性开辟目标矩阵的内存空间？

如果一次性开辟好目标矩阵内存空间，那么原矩阵的元素直接移动到转置矩阵的对称坐标就行了

```
# 1 2 3      1 4
# 4 5 6 ==>> 2 5
#           3 6

# 在原有矩阵上改动，牵扯到增加元素和减少元素，麻烦，所以，定义一个新的矩阵输出
matrix = [[1,2,3], [4,5,6]]
```

```

#matrix = [[1,4],[2,5],[3,6]]

tm = [[0 for col in range(len(matrix))] for row in range(len(matrix[0]))]
count = 0

for i,row in enumerate(tm):
    for j,col in enumerate(row):
        tm[i][j] = matrix[j][i] # 将matrix的所有元素搬到tm中
        count += 1

print(matrix)
print(tm)
print(count)

```

效率测试

```

import datetime
matrix = [[1,2,3], [4,5,6], [7,8,9]]
matrix = [[1,4],[2,5],[3,6]]

print('\nMethod 1')
start = datetime.datetime.now()
for c in range(100000):
    tm = [] # 目标矩阵
    for row in matrix:
        for i, item in enumerate(row):
            if len(tm) < i + 1:
                tm.append([])

            tm[i].append(item)

delta = (datetime.datetime.now()-start).total_seconds()
print(delta)
print(matrix)
print(tm)

print('\nMethod 2')
start = datetime.datetime.now()
for c in range(100000):

```

```

tm = [0] * len(matrix[0])
for i in range(len(tm)):
    tm[i] = [0] * len(matrix)
#print(tm)

for i,row in enumerate(tm):
    for j,col in enumerate(row):
        tm[i][j] = matrix[j][i]

delta = (datetime.datetime.now()-start).total_seconds()
print(delta)
print(matrix)
print(tm)

```

说明：

上面两个方法在ipython中，使用%%timeit测试下来方法一效率高。

但是真的是方法一效率高吗？

给一个大矩阵，测试一下

```

matrix = [[1,2,3],[4,5,6],[1,2,3],[4,5,6],[1,2,3],[4,5,6],[1,2,3],[4,5,6],[1,2,3],
[4,5,6],[1,2,3],[4,5,6],[1,2,3],[4,5,6],[1,2,3],[4,5,6],[1,2,3],[4,5,6],
[1,2,3],[4,5,6],[1,2,3],[4,5,6],[1,2,3],[4,5,6],[1,2,3],[4,5,6],[1,2,3],
[4,5,6]]

```

测试发现，其实只要增加到4*4开始，方法二优势就开始了。

矩阵规模越大，先开辟空间比后append效率高。

数字统计

随机产生10个数字

要求：

每个数字取值范围[1,20]

统计重复的数字有几个？分别是什么？

统计不重复的数字有几个？分别是什么？

举例：11,7,5,11,6,7,4，其中2个数字7和11重复了，3个数字4、5、6没有重复过

思路：

对于一个排序的序列，相等的数字会挨在一起。但是如果先排序，还是要花时间，能否不排序解决？

例如11,7,5,11,6,7,4，先拿出11，依次从第二个数字开始比较，发现11就把对应索引标记，这样

一趟比较就知道11是否重复，哪些地方重复。第二趟使用7和其后数字依次比较，发现7就标记，当遇到以前比较过的11的位置的时候，其索引已经被标记为1，直接跳过。

```
import random

nums = []
for _ in range(10):
    nums.append(random.randrange(21))

#nums = [1,22,33,56,56,22,4,56,9,56,2,1]
print("Origin numbers = {}".format(nums))
print()

length = len(nums)
samenums = [] # 记录相同的数字
diffnums = [] # 记录不同的数字
states = [0] * length # 记录不同的索引异同状态

for i in range(length):
    flag = False # 假定没有重复
    if states[i] == 1:
        continue
    for j in range(i+1, length):
        if states[j] == 1:
            continue
        if nums[i] == nums[j]:
            flag = True
            states[j] = 1
    if flag: # 有重复
        samenums.append(nums[i])
        states[i] = 1
    else:
        diffnums.append(nums[i])

print("Same numbers = {1}, Counter = {0}".format(len(samenums), samenums))
print("Different numbers = {1}, Counter = {0}".format(len(diffnums), diffnums))
print(list(zip(states,nums)))
```