习题

求100内的素数

为了比较算法效率我们扩大到求100000内素数

```
import datetime
# 1 简单算法
# 一个数能被从2开始到自己的平发根的正整数整数整除，就是合数
start = datetime.datetime.now()

n = 100000
count = 0
for x in range(2, n):
    for i in range(2, int(x ** 0.5) + 1):
        if x % i == 0:
            break
    else:
        count += 1
        #print(x)

delta = (datetime.datetime.now() - start).total_seconds()
print(count) # 9592
print(delta) # 0.267015
print('-' * 30)

# 2 使用奇数
start = datetime.datetime.now()

n = 100000
count = 1
for x in range(3, n, 2):
    for i in range(3, int(x ** 0.5)+1, 2):
        if x % i == 0:
            break
    else:
        count += 1
        #print(x)

delta = (datetime.datetime.now() - start).total_seconds()
print(count) # 9592
print(delta) # 0.132007
print('-' * 30)

# 3 存储质数
# 合数一定可以分解为几个质数的乘积，2是质数
# 质数一定不能整除1和本身之内的整数
start = datetime.datetime.now()
n = 100000
```

马哥教育 IT人的高薪职业学院

```
count = 1
primenumbers = [2]

for x in range(3, n, 2):
    for i in primenumbers:
        if x % i == 0:
            break
    else:
        primenumbers.append(x)
        count += 1

delta = (datetime.datetime.now() - start).total_seconds()
print(count)  # 9592
print(delta)  # 4.02523
print('-' * 30)

# 4 缩小范围
start = datetime.datetime.now()
n = 100000
count = 1
primenumbers = [2]

for x in range(3, n, 2):
    flag = False  # 不是素数
    for i in primenumbers:
        if i > int(x ** 0.5): # 素数
            flag = True
            break

        if x % i == 0: # 合数
            flag = False
            break
    if flag:
        primenumbers.append(x)
        count += 1

delta = (datetime.datetime.now() - start).total_seconds()
print(count)  # 9592
print(delta)  # 0.315018
```

马哥教育 IT人的高薪职业学院

算法2和算法4对比，算法2的奇数应该是多于算法4的，也就是算法4应该要快一点，但是测试的记过却不是，为什么？

结果是增加了质数列表反而慢了，为什么？

修改算法如下

```
# 4 缩小范围
# x ** 0.5 在循环中只需计算一次
# 使用列表存储已有的质数，同时缩小范围

start = datetime.datetime.now()
n = 100000
```

```
count = 1
primenumbers = [2]

for x in range(3, n, 2): # 大于2质数只可能是奇数
    flag = False  # 不是素数
    edge = int(x**0.5) # 计算一次
    for i in primenumbers:
        if i > edge: # 素数
            flag = True
            break

        if x % i == 0: # 合数
            flag = False
            break
    if flag:
        primenumbers.append(x)
        count += 1

delta = (datetime.datetime.now() - start).total_seconds()
print(count)  # 9592
print(delta)  # 0.106006
```

这回测试，速度第一了。也就是增加了列表，记录了质数，控制了边界后，使用质数来取模比使用奇数计算更少。**空间换时间**，使用列表空间来换取计算时间。

素数性质

> 大于3的素数只有6N-1和6N+1两种形式，如果6N-1和6N+1都是素数称为孪生素数

```
# 大于3的素数只有6N-1和6N+1两种形式，如果6N-1和6N+1都是素数称为孪生素数
# 注意，其实测试的都是6的倍数的前后的数字，这些数字一定是奇数
n = 100
count = 5  # 2, 3, 5
x = 7
step = 4

while x < n:
    if x % 5 != 0:
        print(x) # 打印出待测试的数
    x += step
    step = 4 if step == 2 else 2
```

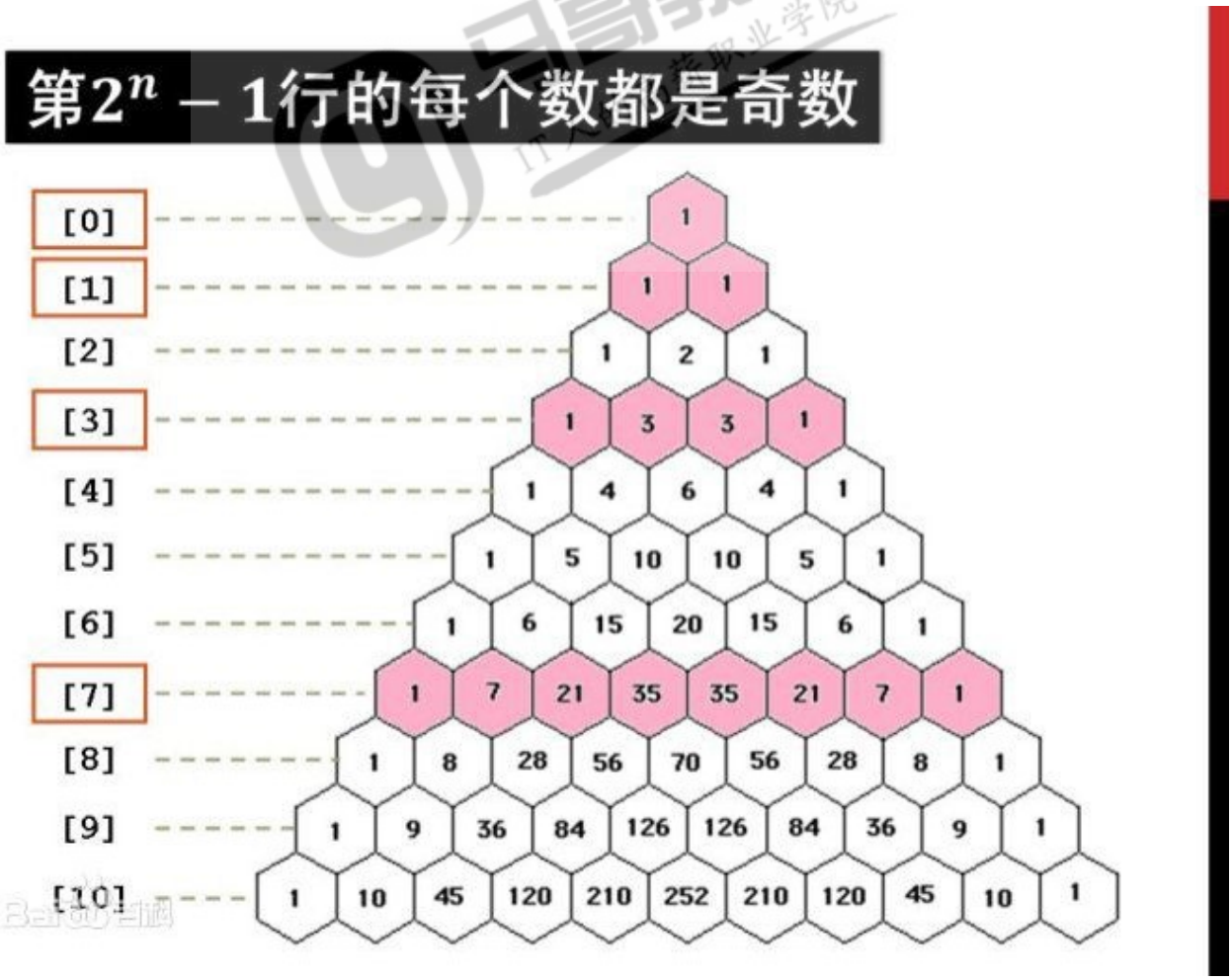
由此，得到下面算法5

```
# 5 使用素数性质
# 大于3的素数只有6N-1和6N+1两种形式，如果6N-1和6N+1都是素数称为孪生素数
start = datetime.datetime.now()
n = 100000
count = 3  # 2, 3, 5
x = 7
```

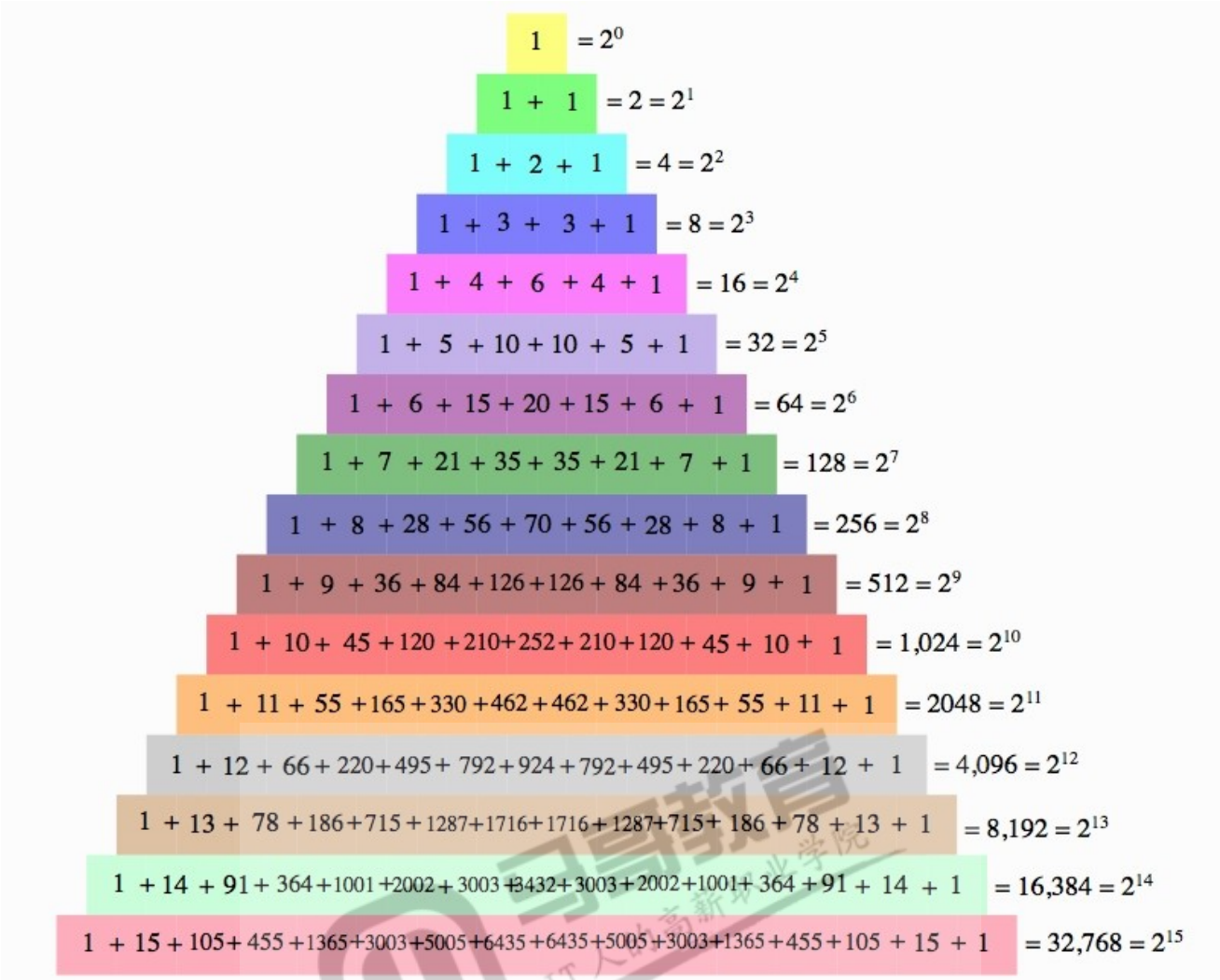
```
step = 4

while x < n:
    if x % 5 != 0:
        for i in range(3, int(x**0.5)+1, 2):
            if x % i == 0: # 合数
                break
        else:
            count += 1
    x += step
    step = 4 if step == 2 else 2

delta = (datetime.datetime.now() - start).total_seconds()
print(count)  # 9592
print(delta)  # 0.122006
```

用了这个性质并没有超过算法4，原因还是在于使用列表来存储已经计算得到的素数来减少计算。请自行使用列表完成素数的存储。

计算杨辉三角前6行



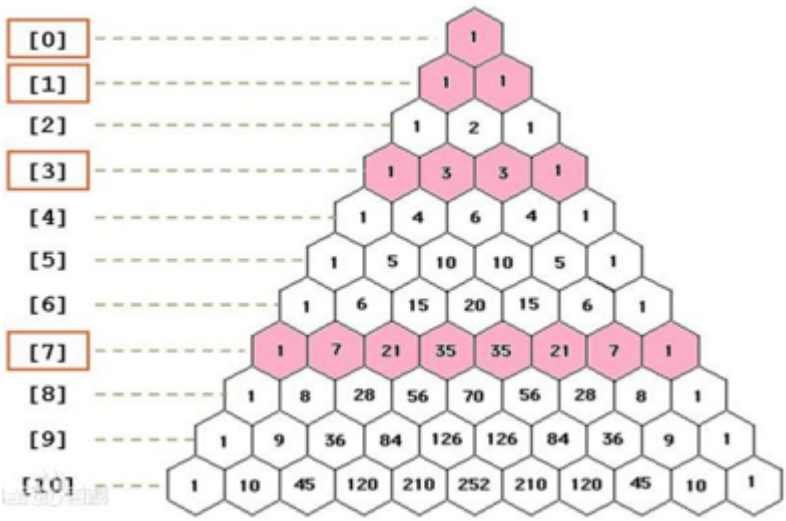
第n行有n项，n是正整数



第n行数字之和为2**(n-1)

解法1 杨辉三角的基本实现

下一行依赖上一行所有元素，是上一行所有元素的两两相加的和，再在两头各加1



预先构建前两行，从而推导出后面的所有行

```
triangle = [[1], [1, 1]]

for i in range(2, 6):
    cur = [1]
    pre = triangle[i-1]
    for j in range(len(pre) - 1):
        cur.append(pre[j] + pre[j+1]) # 前一行2项之和
    cur.append(1)
    triangle.append(cur)

print(triangle)
```

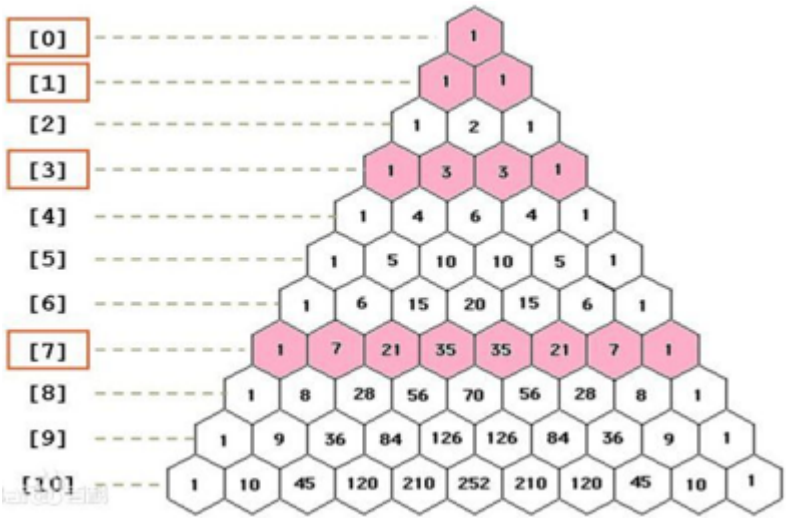
变体 从第一行开始

```
triangle = []
n = 6

for i in range(n):
    cur = [1]
    triangle.append(cur)

    if i == 0: continue

    pre = triangle[i-1]
    for j in range(len(pre) - 1):
        cur.append(pre[j] + pre[j+1]) # 前一行2项之和
    cur.append(1)

print(triangle)
```

马哥教育 IT人的高薪职业学院

解法2 补零

除了第一行以外，每一行每一个元素（包括两头的1）都是由上一行的元素相加得到。如何得到两头的1呢？ 目标是打印指定的行，所以算出一行就打印一行，不需要用一个大空间存储所有已经算出的行。



while循环实现

```
n = 6
newline = [1] # 第一行是特例，因为0+0不等于1
print(newline)


for i in range(1, 6):
    oldline = newline.copy() # 浅拷贝并补0
    oldline.append(0) # 尾部补0相当于两端补0
    newline.clear() # 使用append，所以要清除

    offset = 0
    while offset <= i:
        newline.append(oldline[offset-1] + oldline[offset])
        offset += 1

    print(newline)
```

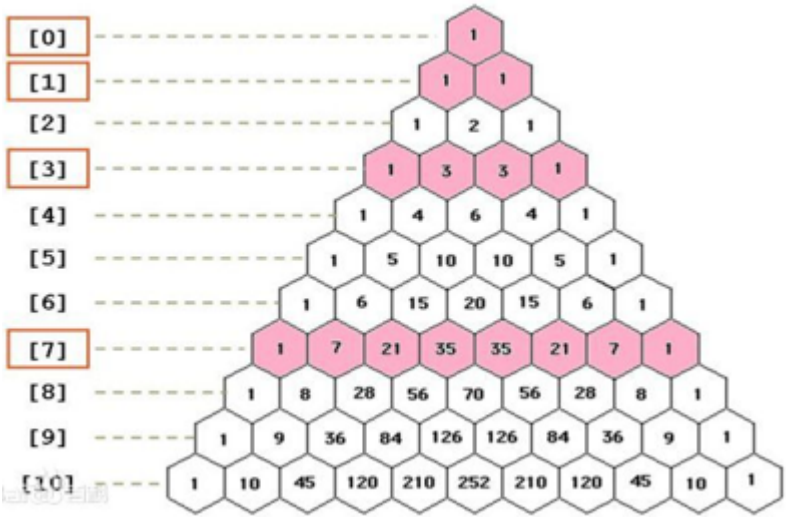
for循环实现

```
n = 6
newline = [1] # 第一行是特例，因为0+0不等于1
print(newline)


for i in range(1, 6):
    oldline = newline.copy() # 浅拷贝并补0
    oldline.append(0) # 尾部补0相当于两端补0
    newline.clear() # 使用append，所以要清除

    for j in range(i+1):
        newline.append(oldline[j - 1] + oldline[j])

    print(newline)
```

马哥教育 IT人的高薪职业学院

解法3 对称性

思路： 能不能一次性开辟空间，可以使用列表解析式或者循环迭代的方式。 能不能减少一半的数字计算。



1、 构建行

```
triangle = []
n = 6

for i in range(n):
    row = [1]
    for k in range(i):
        row.append(1) if k==i-1 else row.append(0)
    triangle.append(row)

    if i == 0: continue

print(triangle)
```

上面创建每一行的代码过于啰嗦了，一次性创建出一行

```
triangle = []
n = 6

for i in range(n):
    row = [1] * (i + 1)
    triangle.append(row)

    if i == 0: continue

print(triangle)
```

马哥教育 IT人的高薪职业学院

2、 中点的确定

```
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
把整个杨辉三角看成左对齐的二维矩阵。
i==2时，在第3行，中点的列索引j==1
i==3时，在第4行，无中点
i==4时，在第5行，中点的列索引j==2
得到以下规律，如果有i==2j，则有中点
```

```
triangle = []
n = 6

for i in range(n):
    row = [1] * (i + 1) # 一次开辟空间
    triangle.append(row)
```

```
    # i为0、1不进来
    # i为2，range(1,2)，j取1
    # i为3，range(1,2)，j取1
    # i为4，range(1,3)，j取1 2
    for j in range(1, i//2+1):
        val = triangle[i-1][j-1] + triangle[i-1][j]
        row[j] = val
        row[-j-1] = val

print(triangle)
```

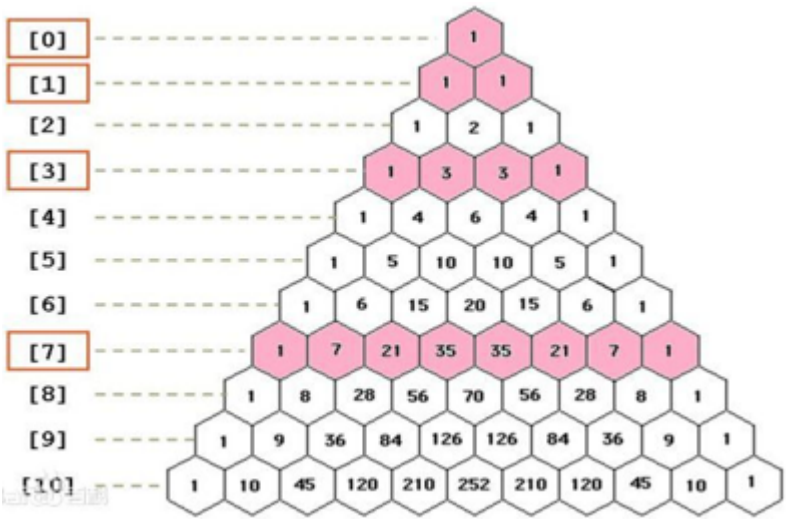
上面的代码`row[-j-1] = val`多做了一次

```
triangle = []
n = 6

for i in range(n):
    row = [1] * (i + 1) # 一次开辟空间
    triangle.append(row)

    # i为0、1不进来
    # i为2，range(1,2)，j取1
    # i为3，range(1,2)，j取1
    # i为4，range(1,3)，j取1 2
    for j in range(1, i//2+1):
        val = triangle[i-1][j-1] + triangle[i-1][j]
        row[j] = val
        if i != 2 * j:
            row[-j-1] = val

print(triangle)
```

马哥教育 IT人的高薪职业学院

解法4 单行覆盖

方法2每次都要清除列表，有点浪费时间。 能够用上方法3的对称性的同时，只开辟1个列表实现吗？



首先我们明确的知道所求最大行的元素个数，例如前6行的最大行元素个数为6个。 下一行等于首元素不变，覆盖中间元素。

```
n = 6
row = [1] * n # 一次性开辟足够的空间
print(row)
print('-' * 30)

for i in range(6):
    offset = n - i

    for j in range(1, i//2+1): # i为0,1不进入
        val = row[j-1] + row[j]
        row[j] = val
        if i != 2*j:
            row[-j-offset] = val

    print(row[:i+1])
    #print(row[:n])
```

运行结果如下

```
[1, 1, 1, 1, 1, 1]
------------------------------
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 7, 4, 1]
[1, 5, 12, 12, 5, 1]
```

马哥教育 IT人的高薪职业学院

问题出在哪里了呢？

原因在于，4覆盖了3，导致3+3变成了3+4才有了7。使用一个临时变量解决

```
n = 6
row = [1] * n # 一次性开辟足够的空间
print(row)
print('-' * 30)

for i in range(6):
    offset = n - i

    old = 1 # 相当于每行行首1，因为i从4开始就有覆盖了，引入这个变量
    for j in range(1, i//2+1): # i为0,1不进入
        val = old + row[j]
        old = row[j]
        row[j] = val
        if i != 2*j:
            row[-j-offset] = val

    print(row[:i+1])
```

也可以写成下面这样

```
n = 6
row = [1] * n # 一次性开辟足够的空间
print(row)
print('-' * 30)

for i in range(6):
    offset = n - i

    old = 1 # 相当于每行行首1，因为i从4开始就有覆盖了，引入这个变量
    for j in range(1, i//2+1): # i为0,1不进入
        # val = old + row[j]
        # old = row[j]
        # row[j] = val
        old, row[j] = row[j], old + row[j]
        if i != 2*j:
            row[-j-offset] = row[j]

    print(row[:i+1])
```

马哥教育 IT人的高薪职业学院