# 实验一

棋盘宽度为2的幂次，因此可以将棋盘分为四个部分，左上、左下、右上、右下。

特殊点一定在这个四个其中之一，将另外三个部分靠近棋盘中心的角设为伪特殊点，进行递归运算。

动态方程

T(k) = 4(k-1)+O(1) (k>0),O(1)(k=0) 时间复杂度为O(4k)

代码

import numpy as np

import random

a = np.zeros(shape=(1000, 1000), dtype=int)

cnt = [1 for i in range(1000)]

Case = 1

def solve(d, sx, sy, x, y,ca):

if d <= 0:

return

d /= 2

d = int(d)

t1 = t2 = 0

if y >= d:

t1 = 1

if x >= d:

t2 = 1

if d == 1:

# 最小棋盘

a[sx][sy + (1 - t1)] = cnt[ca]

a[sx+1][sy + (1 - t1)] = cnt[ca]

a[sx + (1 - t2)][sy] = cnt[ca]

a[sx + (1 - t2)][sy+1] = cnt[ca]

cnt[ca] += 1

else:

# 左上角

if t1 == t2 == 0:

solve(d, sx, sy, x, y,ca)

else:

solve(d, sx, sy, d-1, d-1,ca)

# 右上角

if t1 == 1 and t2 == 0:

solve(d, sx, sy + d, x, y-d,ca)

else:

solve(d, sx, sy + d, d - 1, 0,ca)

# 左下角

if t1 == 0 and t2 == 1:

solve(d, sx + d, sy, x - d, y,ca)

else:

solve(d, sx + d, sy, 0, d - 1,ca)

# 右下角

if t1 == t2 == 1:

solve(d, sx + d, sy + d, x-d, y-d,ca)

else:

solve(d, sx + d, sy + d, 0, 0,ca)

# 中心部分

solve(2, sx + d - 1, sy + d - 1, t2, t1,ca)

def go(k, x, y,ca):

sx = sy = 0

solve(k, sx, sy, x, y, ca)

d = 20

# 输出结果

for i in range(k):

for j in range(k-1,-1,-1):

print(a[i][j], end=' ')

print()

# 对颜色进行随机

color\_code = [(random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)) for i in range(cnt[ca])]

color\_code[0] = (0, 0, 0)

import turtle

try:

turtle.home()

except:

turtle.home()

turtle.speed(0.2)

turtle.colormode(255)

sx = -d \* k // 2

sy = -d \* k // 2

# 填色

for xx in range(0, d\*k, d):

for yy in range(0, d\*k, d):

turtle.penup()

turtle.goto(sx+xx, sy+yy)

turtle.pendown()

turtle.begin\_fill()

color = a[xx//d][yy//d]

turtle.fillcolor(color\_code[color])

for z in range(4):

turtle.fd(d)

turtle.left(90)

turtle.end\_fill()

turtle.hideturtle()

turtle.mainloop()

if \_\_name\_\_ == '\_\_main\_\_':

while True:

k, y, x = map(int, input().split(' '))

y -= 1

x -= 1

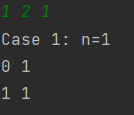
print('Case {}: n={}'.format(Case, k))

k = 2 \*\* k

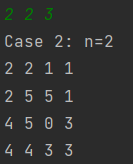
go(k, x, y, Case)

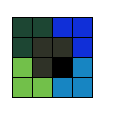
Case += 1

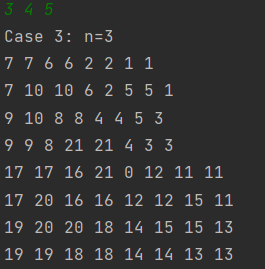
## 运行结果

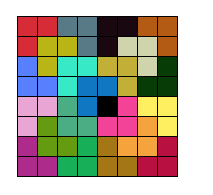












## 遇到的问题

Python在使用Turtle库时，不能连续使用画图，容易遇到turtle.Terminator错误，因此设置了异常处理，来回避异常。

## 实验体会

棋盘覆盖问题充分利用了递归与分治的思想，是一个很好的典型。得到正确的结果很容易，在可视化过程中遇到的问题反而更多，通过查阅资料等手段得以顺利解决。