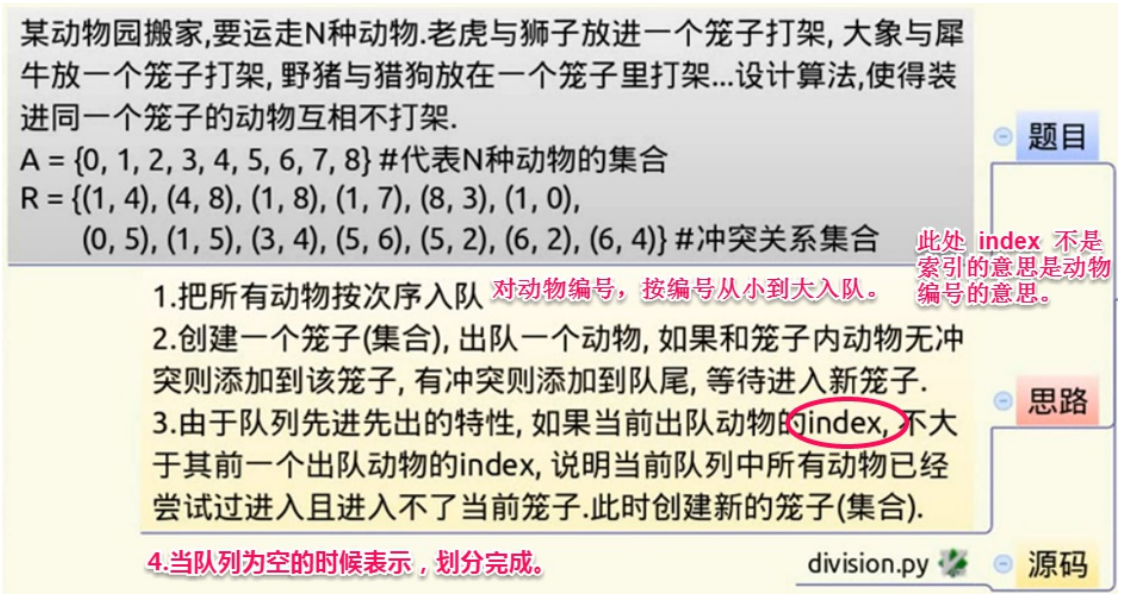
队列-2

divison.py

- **案例**
  - **案例-2：划分无冲突子集**
    - **概述**



    - **思路**
      - **如何判断需要创建新笼子？**如果当前出队（后出队）动物的 index ，小于（原文是不大于）其前一个出队（先出队）动物的 index ，说明当前队列中所有动物已经尝试过进入且进入不了当前笼子。此时创建新的笼子（集合）。因为队列是按照从小到大的编号入队的，所以应该是先出队的元素编号小于后出队的元素编号。如果先出队的元素编号大于后出队的元素编号，说明此时队首元素已经是尝试过进入且进入不了笼子重新添加到队尾的，说明当前队列中所有动物已经尝试过进入且进入不了当前笼子 。故需要创建新笼子。换句话说如果队列重新变为了从小到大排序，说明队列中所有元素都尝试进入当前笼子而没能进入（也就是5追加到队尾（**不一定要追加到队尾，进入当前笼子也行**）的瞬间，也就是先出队元素大于了后出队元素的瞬间）。

      - **如何描述笼子？**因为需要返回笼子的集合，通过栈实现。栈存储笼子的集合，栈中每一层元素是一个列表代表一个笼子。栈顶元素代表当前笼子。

[8]	栈顶
[4,5]	
[1,6]	
[0,2,3,7]	栈底

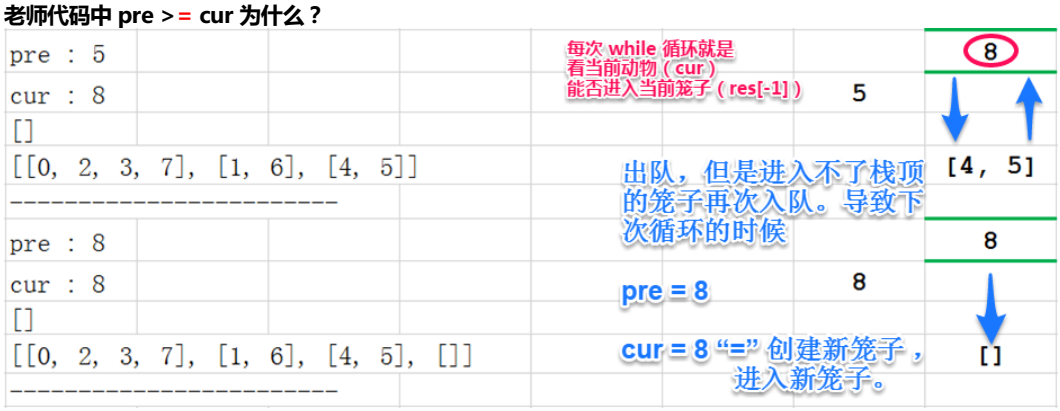
- **如何表示冲突关系-1？**通过冲突关系矩阵表示。矩阵元素为 1，代表行坐标、列坐标两元素冲突。我认为还不如使用元组的集合来表示冲突关系来的直观。

	0	1	2	3	4	5	6	7	8
0		1				1			
1	1				1	1		1	1
2							1		
3					1				1
4		1		1			1		1
5	1	1					1		
6			1		1	1			
7		1							
8		1		1	1				

- **如何表示冲突关系-2？**使用元组的集合表示冲突关系。

R = {(1, 4), (4, 8), (1, 8), (1, 7),
(4, 1), (8, 4), (8, 1), (7, 1),
(8, 3), (1, 0), (0, 5), (1, 5),
(3, 8), (0, 1), (5, 0), (5, 1),
(3, 4), (5, 6), (6, 2), (6, 2), (6, 4)
(4, 3), (6, 5), (2, 6), (2, 6), (4, 6)}

- **老师代码中 pre >= cur 为什么？**



- **代码-1：**自己思路实现。先进笼子一个动物，判断队列中剩余动物能否进笼子。

```
# coding:utf-8

from collections import deque

# R 代表冲突关系
# n 代表 0~n-1 个数字需要被划分
def divison(R, n):

    # 生成一个 0~8 的队列，代表 9 种动物按照从小到大的编号依次入队
    q = deque([i for i in range(n)])
```

```
    # 代表笼子集合
    res = []
    # 创建第一个笼子
    res.append([])
    # 队列中第一只动物入第一个笼子
    res[-1].append(q.popleft())

    # while 循环一次代表当前队首动物能否进入当前笼子，能就进入当前笼子
    # 不能追加到队列尾部，当队列重新变为从小到大时，代表队列剩余动物都
    # 没能进入当前笼子，创建新笼子
    while q:

        # 如何判断需要添加新笼子？
        # if  sorted(list(q)) == list(q):
        #     res.append([])

        # 队首动物与当前笼子中动物冲突，队首动物出队添加到队尾
        # 这里通过一个 for 循环遍历当前笼子中所有动物，与队首
        # 动物配对，看是否冲突，一旦冲突队首动物出队添加到队
        # 尾
        for i in range(len(res[-1])):
            if (q[0],  res[-1][i]) in R:
                q.append(q.popleft())
                break
        # 如果队首动物与当前笼子中动物都不冲突，也就是 if 判断
        # 体中的 break 没有触发 for 循环正常结束，走到了 for
        # 循环的 else 也就是下面部分，将队首动物添加到当前笼子
        # 中
        else:
            res[-1].append(q.popleft())

        # 如果队列是按顺序的说明队列中所有动物都尝试过进入当前笼子，
        # 但是没能进入，故都追加到了队尾
        if sorted(list(q)) == list(q):
            res.append([])
            # 这里必须进去一个动物，否则下次循环的时候，
            # 队列还是从小到大的会再创建一个笼子（我认为这就是我这个思路不好的地方）
            res[-1].append(q.popleft())

    print list(res)

if __name__ == '__main__':
    # 0~8 这九个数字需要被划分
    N = 9
    # 0~8 这九个数字的冲突关系
    R = {(1, 4), (4, 8), (1, 8), (1, 7),
         (4, 1), (8, 4), (8, 1), (7, 1),
         (8, 3), (1, 0), (0, 5), (1, 5),
         (3, 8), (0, 1), (5, 0), (5, 1),
         (3, 4), (5, 6), (6, 2), (6, 2), (6, 4),
         (4, 3), (6, 5), (2, 6), (2, 6), (4, 6)}

    print divison(R, N)
```

- **代码-2**：老师思路实现，**pre = cur** 精髓

```
# coding:utf-8

from collections import deque

def divison(R, n):
    # 笼子集合
    res = []
    # 动物编号 0~n-1 入队
    q = deque(range(n))
    # pre 用来代表之前出队的动物编号，这是初始值并不重要只要比队列第一个元素大即可，保证一开始创建一个笼子
    pre = n
```

```
    # 队列为空代表动物全部入笼，空队列为 False 终止循环
    # 每循环一次代表队首动物（cur）尝试进入当前笼子
    while q:
        # 当前出队动物编号
        cur = q.popleft()
        # 之前出队的动物编号 ≥ 当前出队的动物编号，创建新笼子
        # 必须是大于等于，pre = cur 是什么情况？见思路
        if pre >= cur:
            res.append([])

        # res[-1] 是空列表 []，for 循环一次都不走，但 for 循环正常结束进入 else
        # 这才是迭代，对比代码-1的 for 循环
        for animal in res[-1]:
            # 冲突
            if (cur, animal) in R:
                q.append(cur)
                break
        else:
            # 不冲突
            res[-1].append(cur)

        # 这是重点
        pre = cur

    return list(res)

if __name__ == '__main__':
    # 0~8 这九个数字需要被划分
    N = 9
    # 0~8 这九个数字的冲突关系。 如果删掉这两个元组，8 就可以进入栈顶的笼子，pre > cur 即可
    R = {(1, 4), (4, 8), (1, 8), (1, 7),
         (4, 1), (8, 4), (8, 1), (7, 1),
         (8, 3), (1, 0), (0, 5), (1, 5),
         (3, 8), (0, 1), (5, 0), (5, 1),
         (3, 4), (5, 6), (6, 2), (6, 2), (6, 4),
         (4, 3), (6, 5), (2, 6), (2, 6), (4, 6)}

    print divison(R, N)
```

- **And So On**