## 并查集（NBplus）：

## 01182:食物链

**描述**

动物王国中有三类动物A,B,C，这三类动物的食物链构成了有趣的环形。A吃B， B吃C，C吃A。  
现有N个动物，以1－N编号。每个动物都是A,B,C中的一种，但是我们并不知道它到底是哪一种。  
有人用两种说法对这N个动物所构成的食物链关系进行描述：  
第一种说法是"1 X Y"，表示X和Y是同类。  
第二种说法是"2 X Y"，表示X吃Y。  
此人对N个动物，用上述两种说法，一句接一句地说出K句话，这K句话有的是真的，有的是假的。当一句话满足下列三条之一时，这句话就是假话，否则就是真话。  
1） 当前的话与前面的某些真的话冲突，就是假话；  
2） 当前的话中X或Y比N大，就是假话；  
3） 当前的话表示X吃X，就是假话。  
你的任务是根据给定的N（1 <= N <= 50,000）和K句话（0 <= K <= 100,000），输出假话的总数。

**输入**

第一行是两个整数N和K，以一个空格分隔。  
以下K行每行是三个正整数 D，X，Y，两数之间用一个空格隔开，其中D表示说法的种类。  
若D=1，则表示X和Y是同类。  
若D=2，则表示X吃Y。

**输出**

只有一个整数，表示假话的数目。

**样例输入**

100 7

1 101 1

2 1 2

2 2 3

2 3 3

1 1 3

2 3 1

1 5 5

**样例输出**

3

使用 带权并查集，通过节点到根节点的距离模3表示关系：

distance % 3 == 0：与根节点同类。

distance % 3 == 1：吃根节点。

distance % 3 == 2：被根节点吃。

关键操作

find 函数

路径压缩时，同步更新节点到根节点的距离。

合并与关系判断

根据输入的 D（1或2），计算两个节点的关系是否与已有关系冲突。

def main():

import sys

sys.setrecursionlimit(1 << 25)

n, k = map(int, sys.stdin.readline().split())

parent = list(range(n + 1))

distance = [0] \* (n + 1) # 距离根节点的模3值

count = 0

def find(u):

if parent[u] != u:

orig\_parent = parent[u]

parent[u] = find(parent[u]) # 路径压缩

**distance[u] = (distance[u] + distance[orig\_parent]) % 3#这里的distance设计非常巧妙，解决了在短期操作（如：distance[root\_y] = (distance[x] - distance[y]) % 3）时候root\_y到y的路径上的将要使用点的distance没有同步更新的问题**

return parent[u]

for \_ in range(k):

d, x, y = map(int, sys.stdin.readline().split())

if x > n or y > n:

count += 1

continue

if d == 2 and x == y:

count += 1

continue

root\_x = find(x)

root\_y = find(y)

if d == 1: # X和Y同类

if root\_x == root\_y:

if distance[x] != distance[y]:

count += 1

else:

parent[root\_y] = root\_x

**distance[root\_y] = (distance[x] - distance[y]) % 3**

else: # d == 2: X吃Y

if root\_x == root\_y:

if (distance[x] - distance[y]) % 3 != 1:

count += 1

else:

parent[root\_y] = root\_x

**distance[root\_y] = (distance[x] - distance[y] - 1) % 3**

print(count)

if \_\_name\_\_ == "\_\_main\_\_":

main()

关键公式解释

1. 合并同类关系（D=1）

distance[root\_y] = (distance[x] - distance[y]) % 3

逻辑：

设 x 的根为 root\_x，y 的根为 root\_y。

合并后，root\_y 的新父节点是 root\_x，需要更新 distance[root\_y]，使得合并后 x 和 y 的关系保持一致。

根据定义：

distance[x] 是 x 到 root\_x 的关系。

distance[y] 是 y 到 root\_y 的关系。

合并后，y 到 root\_x 的路径关系应满足：

distance[y] + distance[root\_y] ≡ distance[x] (mod 3)

解得：

distance[root\_y] ≡ distance[x] - distance[y] (mod 3)

2. 合并捕食关系（D=2）

distance[root\_y] = (distance[x] - distance[y] - 1) % 3

场景：当 x 吃 y 时，合并它们的集合。

逻辑：

合并后，x 和 y 的关系应满足 x 吃 y，即：

(distance[x] - distance[y]) % 3 == 1

类似同类关系的推导，但额外减去 1 表示捕食关系：

distance[y] + distance[root\_y] ≡ distance[x] - 1 (mod 3)

解得：

distance[root\_y] ≡ distance[x] - distance[y] - 1 (mod 3)

问题但是在这种条件下如何确保rooty变化后rooty 和y 之间的节点的distance正确呢？

为了解决这个问题，带权并查集通过 路径压缩 和 递归更新 来确保所有节点的 distance 正确。

路径压缩的作用

在 find 函数中，路径压缩不仅将节点的父节点直接指向根节点，还会 递归更新 节点到根节点的 distance。具体来说：

当调用 find(u) 时，如果 u 不是根节点，递归调用 find(parent[u])。

在递归返回时，更新 u 的 distance：

distance[u] = (distance[u] + distance[parent[u]]) % 3

最后，将 u 的父节点指向根节点。

通过这种方式，路径压缩不仅优化了查找效率，还确保了所有节点的 distance 正确。