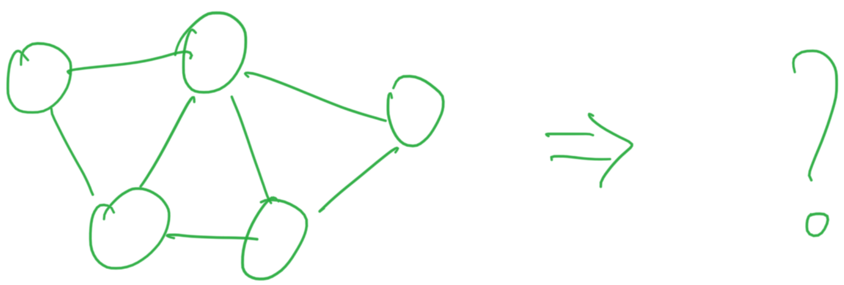
思考题：联通城市

【问题】（城市之间可联通）在给定n对城市的链接关系后，求m次(x，y)之间是否联通。n,m < 10,000

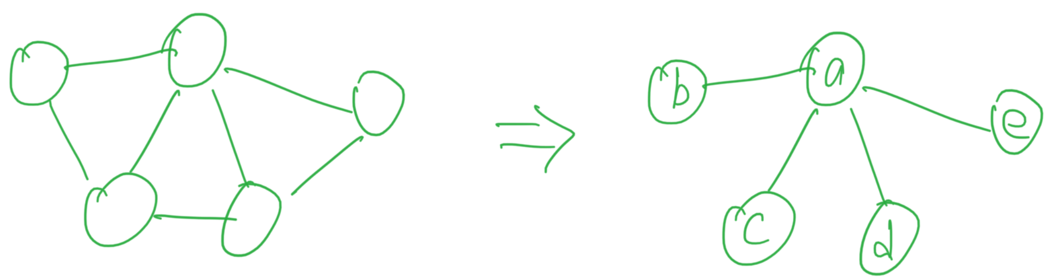
【分析】先不考虑数据量的问题。如果小数据，如何操作？

直接搜索每一组关系，查看特定关系是否可达？（一定层次上可行，但是....方法过于暴力，肯定会被卡掉。）

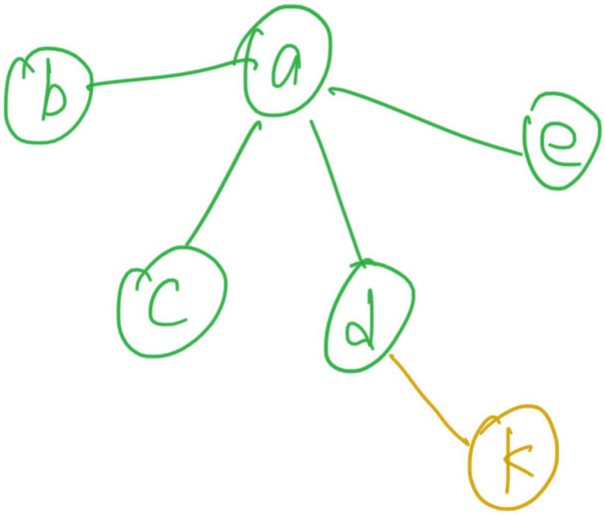
更聪明一点的方法？



其实观察了后可以发现，如果我们在这里只考虑连通性问题，不考虑具体的链接情况，则其实下图和上图并无区别，并且更快。



考虑在上述树中加入新关系 (d,k)，明显，k与a,b,c,d,e 构成新的联通图(如下图)，



那怎么知道k与c能联通呢?

一个不难想到的规律：找 $root[k]$ 和 $root[c]$，如果$root$相同，则说明k, c两者联通。

我们已经知道。在知道父亲关系的情况下，找任意一个节点的父亲方法为：

```
//递归版本
int find(int x){
    //如果自己不是自己的father，则他的爸爸另有其人，则还需继续寻找。
  if(fa[x]!=x) return find(fa[x]);
  return x; //找到了，return 自己的坐标。
}

//非递归版本（比递归版本快）
int find(int x){
  while(fa[x]!=x)x=fa[x]; //直到找到自己是自己的爸爸为止。
  return x;
}

//备注：为了满足递归版本的数组记得初始化：
for(int i=1;i<=n;i++)fa[i]=i; //自己是自己的爸爸
```

那么判断两个结点是否联通的方法：

```
if(find(a)==find(b)){
    //a b联通
}
```

合并方案：

```
//合并:
cin>>a>>b;
if(find(a)!=find(b)) //不是同一个联通图
    fa[a]=b; //任意构建一个关系即可。
```

并查集概念

这种既能支持**关系合并**又能支持**关系查询**操作的数据结构叫==并查集==

1. 合并：两个原本不相交的结合，合并成一个集合。
2. 查询：判断两个结点是否联通。

==并查集并不关心具体的链接方式，而是关心节点之间的联通性\传递性问题。==

> 传递性解释：亲戚的亲戚是亲戚

并查集是基于树的结构实现的。（父亲结点）

并查集还有许多巧妙的运用，也是很多算法的基础，e.g. :Kruscal算法(克鲁斯卡尔)

并查集路径压缩

> C法数据，专卡各种不服。
>
> 假设输入数据为结点10000个，关系9999 条
>
> 1 2
>
> 2 3
>
> ...
>
> 9999 10000
>
> 能让你的“原始”并查集卡的飞起

原因很简单，czc可以通过构造一组特殊数据，让你的查询树最终合并成一条链。

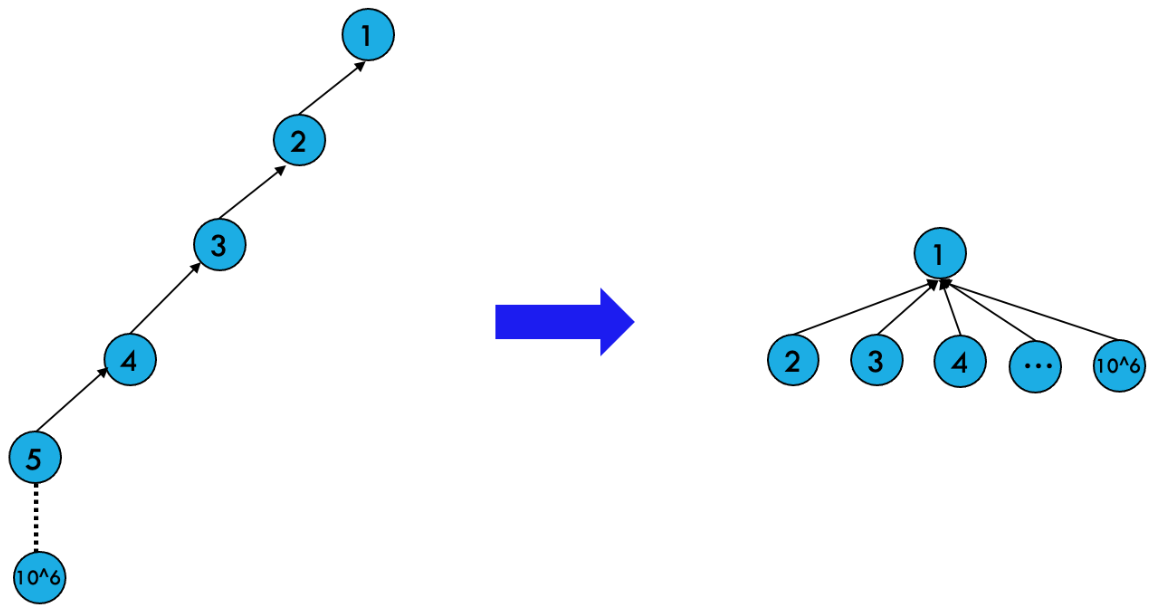
在这条链上，单点查询复杂度是O(n)，查n次，则达到了$O(n^2)$的复杂度。

为了避免这种问题，我们有了路径压缩。

回顾之前的代码：结合爸爸是所有节点的爸爸，可得

```
//递归版本
int find(int x){
  //回溯时把大父亲信息更新到所有节点。
  if(fa[x]!=x) return fa[x]=find(fa[x]);
  return x;
}
```

效果如下图所示

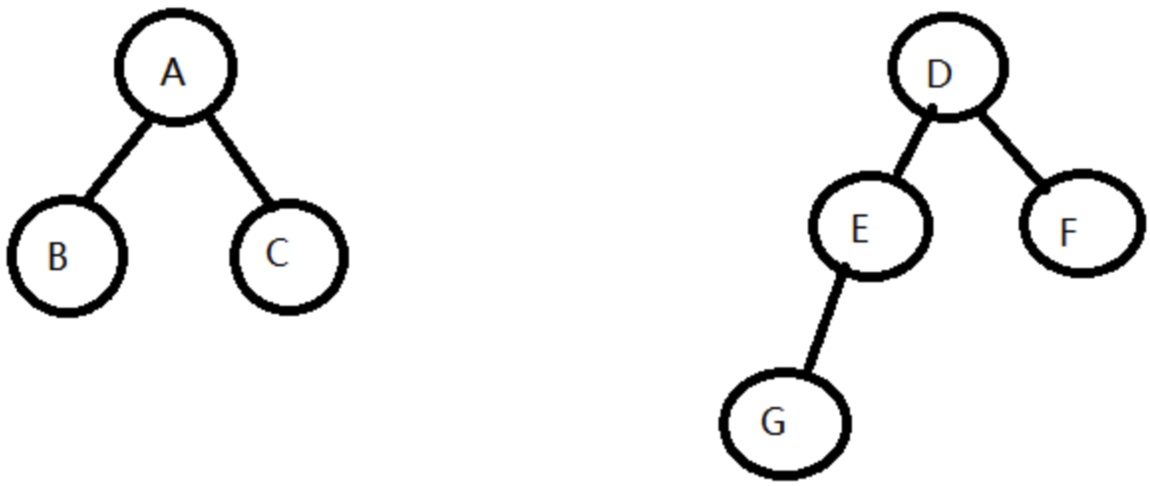


通过路径压缩，我们可以把可能的查询压缩到O(1)的时间复杂度。

那么其实可以合理猜想，合并是否也能优化？

启发式合并（按秩合并）

问题：如果A，D各自有一棵树，请问谁是谁的爸爸能让树的深度尽可能小？



读者可自行在草稿纸上，画出A是D的爸爸，和D是A的爸爸合并结果。并计算深度。

> 树的深度越小，查询耗时越低

通过绘图可以很明显的发现，应该让小的树向大的树合并。（这样可让路径压缩的更快）

> 这里的小的树，特指深度小的树（树的高度小）

操作：为了方便标记树的深度，单独维护一个rank[x]=k 表示以x号结点为根的树深度为k

```
void merge(int r1,int r2){
    int x = find(r1), y = find(r2); //先找到两个根节点
    if (rank[x] <= rank[y]) //深度不同，合并后深度不会改变。直接合并。
        fa[x] = y;
    else
        fa[y] = x;
    if (rank[x] == rank[y] && x != y)  //如果秩相同，就任意合并到一棵
        rank[y]++; //如果深度相同且根节点不同，则新的根节点的深度+1
}
```

> 风险警示：据网传oi选手的经验，在路径压缩和按秩合并联合使用的情况下，算法会在某些少数极端数据下错误(但是czc也没遇到过)

其中，rank和fa数组初始化为：

```
伪代码
for i:
    rank[i]=0; //深度为0
    fa[i]=i; //自己是自己的爸爸
```

种类并查集

> 用于解决敌人的敌人是朋友这类关系问题

【题目】食物链

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a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。

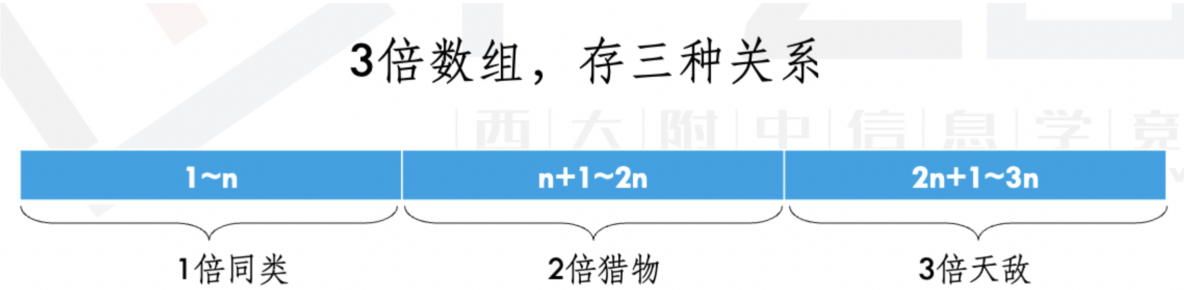
```
样例输入：
100 7
1 101 1
2 1 2
2 2 3
2 3 3
1 1 3
2 3 1
1 5 5
1
2
3
4
5
6
7
8

样例输出：
3
```

通过分析发现，在食物链中，有3种关系：同类、猎物、天敌。

传统的并查集只能维护同类关系。

要想记录多种关系，需使用种类并查集，种类并查集的思想十分简单：开大空间，具体如下：



每获得一对关系就根据d的值分类判断，如果与之前说的不矛盾则直接分门别类维护关系。如果矛盾则ans++

然后输出ans即可。

```
//样例核心代码：
int x, y, d;
    cin >> n >> k;
    for (int i = 1; i <= 3 * n; ++i)  //3倍数组
        f[i] = i;
    for (int i = 1; i <= k; ++i) {
        cin >> d >> x >> y;
        if (x > n || y > n) { ans++; continue; }
        if (d == 1) {  //是同类
            if (find(x + n) == find(y) || find(x + 2 * n) == find(y)) {
                ans++;
                continue;
```

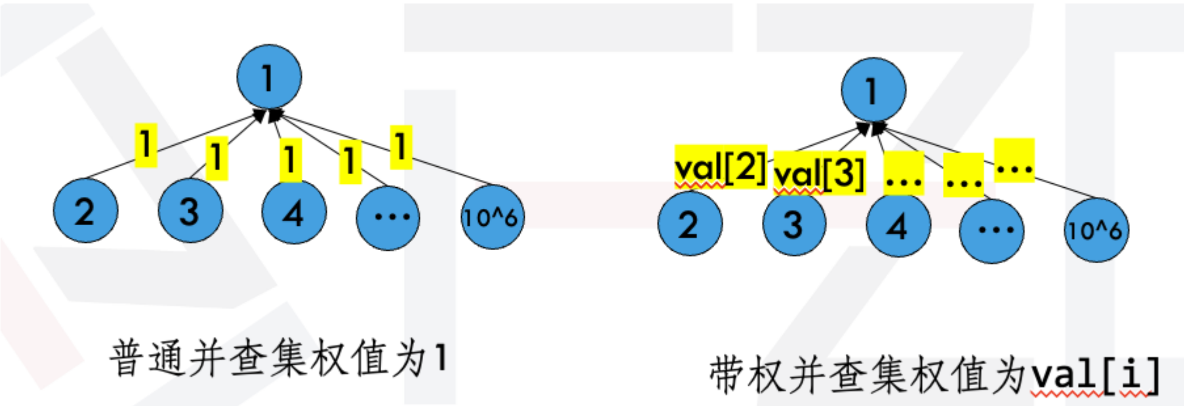
```
            }
            merge(x, y);                      //x和y是同类
            merge(x + n, y + n);              //x的猎物就是y的猎物
            merge(x + 2 * n, y + 2 * n);      //x的天敌就是y的天敌
        }
        else if (d == 2) { //是天敌
            if (find(x) == find(y) || find(x + 2 * n) == find(y)) {
                ans++;
                continue;
            }
           merge(x, y + 2 * n);      //x是y的天敌
           merge(x + n, y);          //x的猎物是y
           merge(x + 2 * n, y + n); //x的天敌是y的猎物
        }
    }
    cout << ans;
    return 0;
}
```

带权并查集

> 开大数组始终不是个事...，空间太大了，万一炸了怎么办？

运用带权并查集可以解决这个问题，种类并查集其实就是带权并查集的特殊化。

这里展示带路径压缩的带权并查集。其中val[x]=k 表示从根结点到x结点的权为k



> 带权并查集可以通过不同的权值表示不同的关系

之前的路径压缩代码：

```
//递归版本
int find(int x){
  //回溯时把大父亲信息更新到所有节点。
  if(fa[x]!=x) return fa[x]=find(fa[x]);
  return x;
}
```

现在的路径压缩代码：

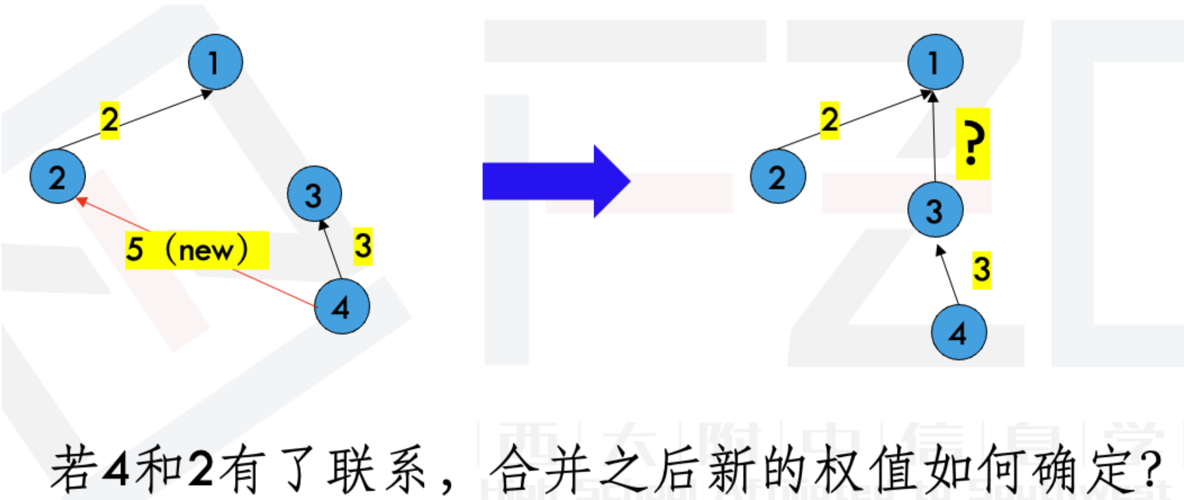
```
int find(int x){
    if (x != fa[x]){
        int t = fa[x];  //先记录下原本的父节点
        fa[x] = find(fa[x]);  //路径压缩后父节点会变为根节点
        val[x] += val[t];  //加上原本父节点的权值，即当前结点到根节点的权值
    }
    return fa[x];
}
```

表示种类关系的代码：

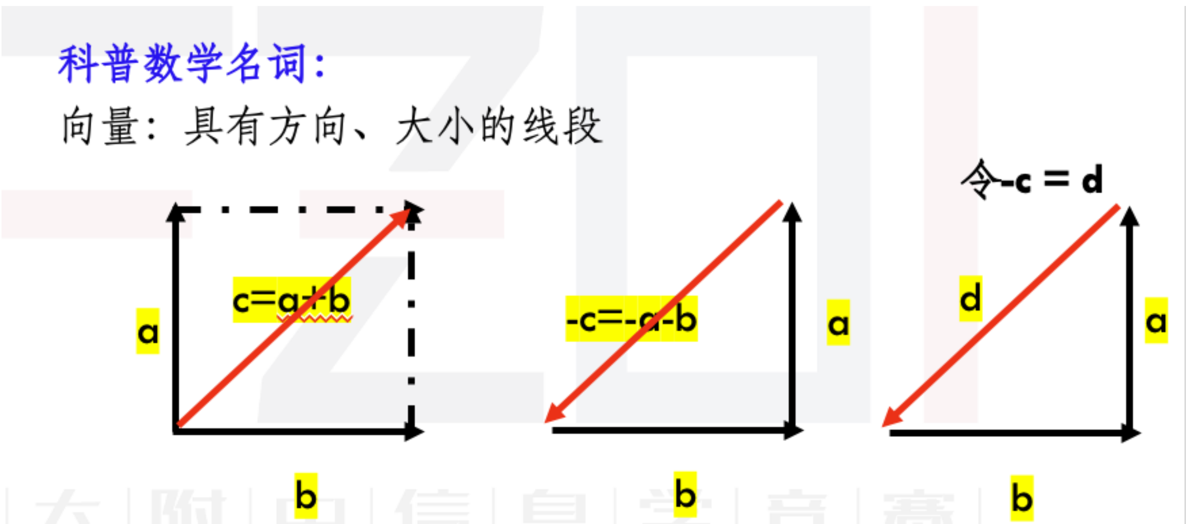
```
int find(int x){
    if (x != fa[x]){
        int t = fa[x];
        fa[x] = find(fa[x]);
        val[x] = (val[x]+val[t])%k;  //注意% ，k为关系种类数量
    }
    return fa[x];
}
```

带权并查集合并：

问题描述：



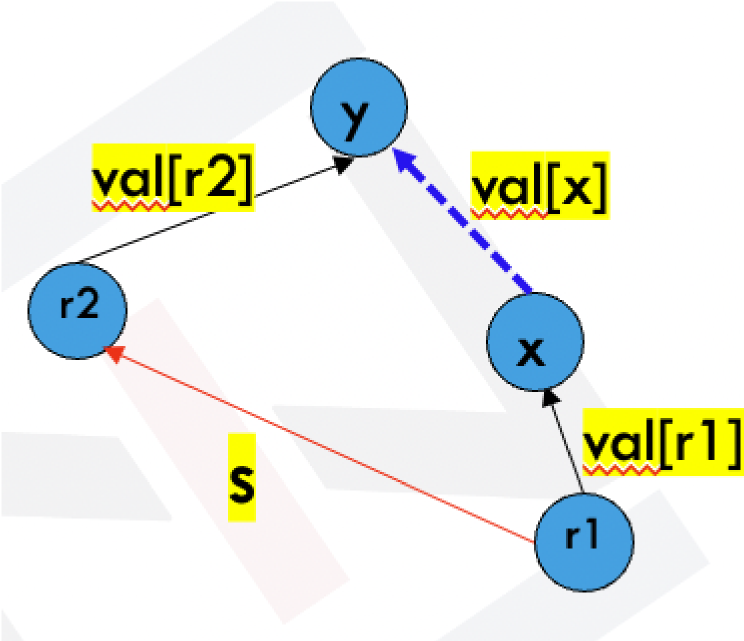
根据向量平行四边形法则（c=a+b）：封闭向量和为0



因为c=a+b,令d=-c 则 a+b+d = 0

所以，首尾相连的封闭向量，和为0

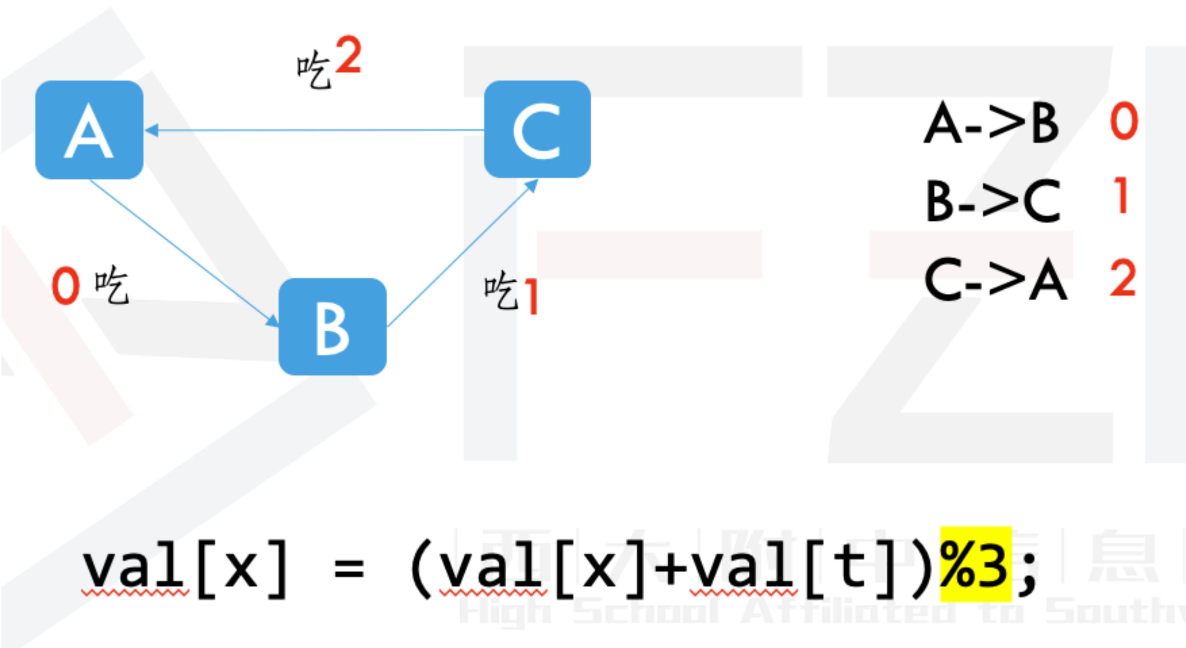
那么回到原来的问题上：



根据数学，可得到：`val[x]=-val[r1]+val[r2]+s`

```
void merge(int r1,int r2,int s){ //r1 和 r2相连，权值为s
    int x=find(r1);
    int y=find(r2);
    if(x!=y){
        fa[x]=y;
        val[x] = -val[r1] + val[r2] + s;
    }
}
```

则对于食物链来讲：



可得食物链代码

```
#include<bits/stdc++.h>
Using namespace std;
int fa[500005];
int val[500005];
int d;
int find(int x){
    if(x!=fa[x]){
        int t=fa[x];
        fa[x]=find(fa[x]);
        val[x]=(val[x]+val[t])%3;
    }
    return fa[x];
}
int merge(int d,int r1,int r2){
    int x=find(r1);
    int y=find(r2);
    if(x==y){
        if((-val[r2]+val[r1]+3)%3!=d)
            return 1;
        else
            return 0;
    }
    fa[x]=y;
    val[x]=(-val[r1]+val[r2]+d+3)%3;
    return 0;
}

int main(){
    int n,k,i,j,ans=0;
    scanf("%d%d",&n,&k);
    int d,x,y;
    for(i=0;i<=n;i++){
        fa[i]=i;
        val[i]=0;
    }
    for(i=0;i<k;i++){
      scanf("%d%d%d",&d,&x,&y);
        if(x>n || y>n){
            ans++;
            continue;
        }
        if(d==2 && x==y){
            ans++;
            continue;
        }
        if(merge(d-1,x,y))
            ans++;
    }
    printf("%d\n",ans);
    return 0;
}
```