

打击犯罪

快过年了，犯罪分子们也开始为年终奖“奋斗”了，小哼的家乡出现了多次抢劫事件。由于强盗人数过于庞大，作案频繁，警方想查清楚到底有几个犯罪团伙实在是太不容易了，不过警察叔叔还是搜集到了一些线索，需要咱们帮忙分析一下。

现在有11个强盗。

1号强盗与2号强盗是同伙。

3号强盗与4号强盗是同伙。

5号强盗与2号强盗是同伙。

4号强盗与6号强盗是同伙。

2号强盗与6号强盗是同伙。

7号强盗与11号强盗是同伙。

8号强盗与7号强盗是同伙。

9号强盗与7号强盗是同伙。

9号强盗与11号强盗是同伙。

1号强盗与6号强盗是同伙。

有一点需要注意：强盗同伙的同伙也是同伙。
你能帮助警方查出有多少个独立的犯罪团伙吗？

问题：如何表示这样的“合伙”关系？

分析

1号强盗与2号强盗是同伙。
3号强盗与4号强盗是同伙。
5号强盗与2号强盗是同伙。
4号强盗与6号强盗是同伙。
2号强盗与6号强盗是同伙。
7号强盗与11号强盗是同伙。
8号强盗与7号强盗是同伙。
9号强盗与7号强盗是同伙。
9号强盗与11号强盗是同伙。
1号强盗与6号强盗是同伙。

简单思路：染色法，如果是同伙就染成同样的颜色
缺点：可能存在大量重复染色，维护起来也非常麻烦。

一个团伙肯定是有上下级关系的， $\text{boss}[i]$

在最开始，强盗各自为政， $\text{boss}[i]=i$;

得知1,2是一伙的;

得知3,4是一伙的; $\text{boss}[3]=4$

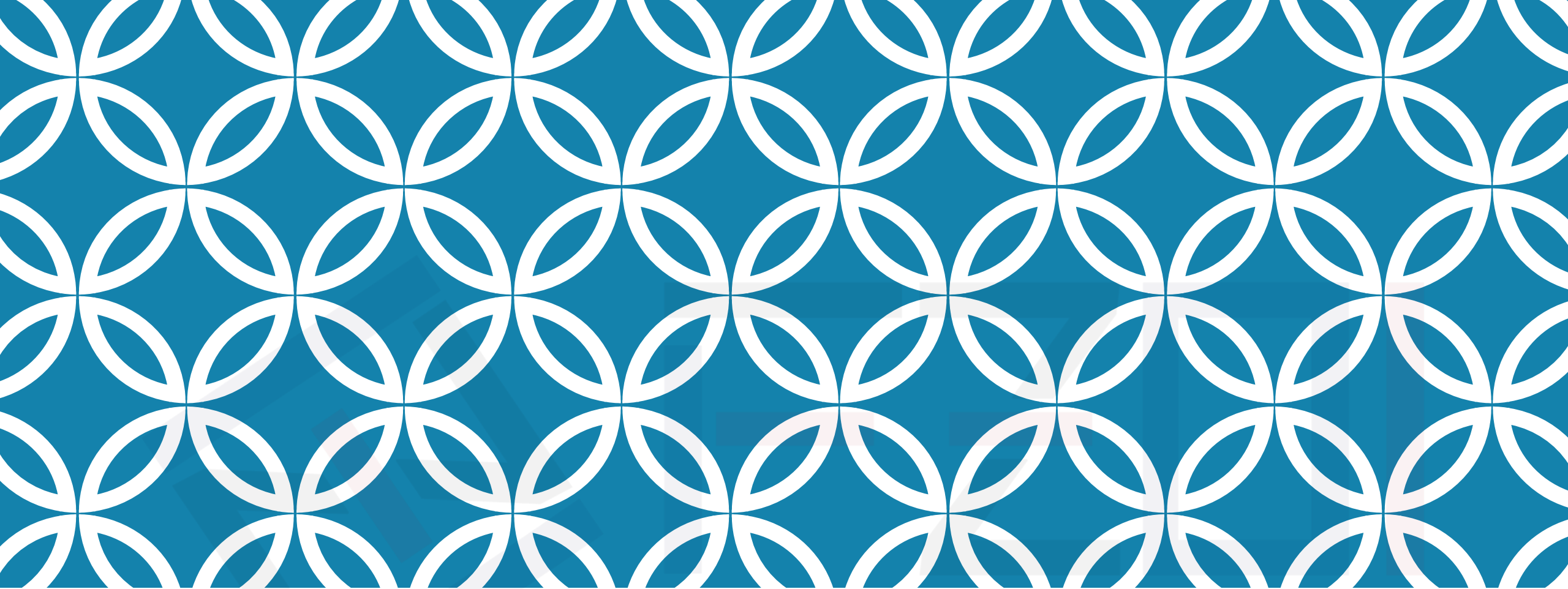
得知5,2是一伙的; $\text{boss}[5]=2$



1,2,5变成一伙了

有没有一种数据结构可以方便我们维护这样的一个关系集合?

并查集



并查集

| 西 | 大 | 附 | 中 | 信 | 息 | 学 | 竞 | 赛 |
High School Affiliated to Southwest University

QYC

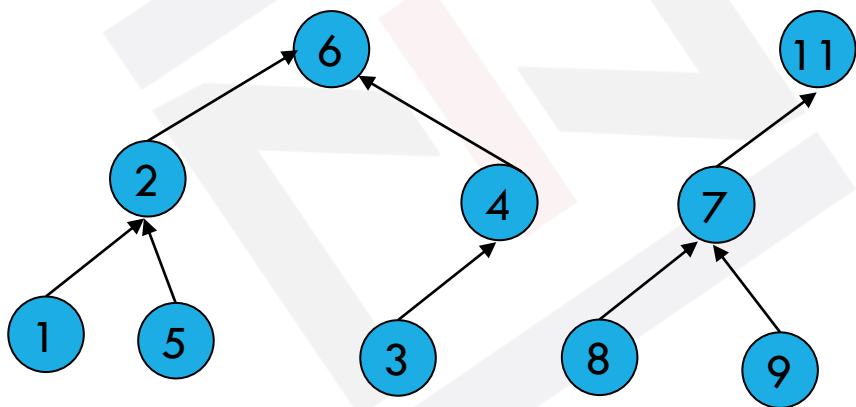
并查集

并查集支持合并、查询两种操作

并查集-合并

合并关系

把两个不相交的集合合并为一个集合



根据题目给出的关系：

boss[1]=2 boss[3]=4 boss[2]=6
boss[5]=2 boss[4]=6 boss[7]=11
boss[8]=7 boss[9]=7 boss[9]=11

最终维护出这样一个数组：

1	2	3	4	5	6	7	8	9	10	11
2	6	4	6	2	6	11	7	7	10	11

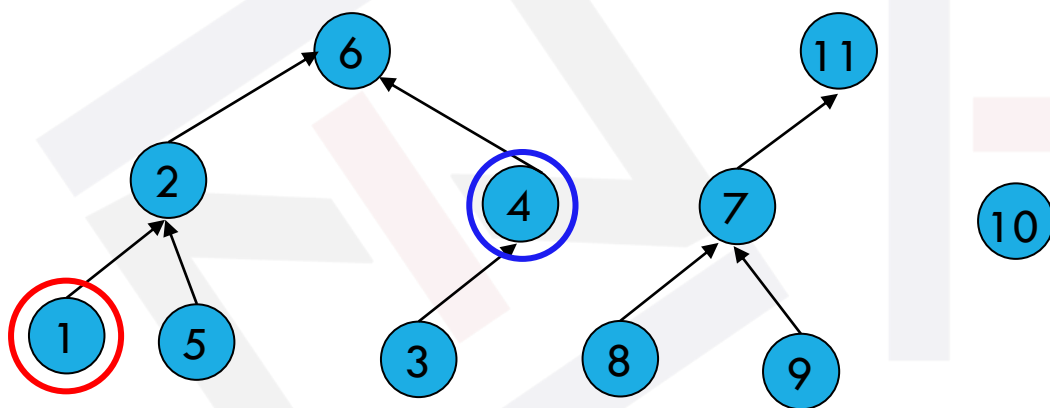
合并操作

```
void merge(int r1,int r2){  
    int x=find(r1); //找到r1的上级  
    int y=find(r2); //找到r2的上级  
    if(x!=y) boss[x]=y; //如果不是同一个上级就合并  
}
```

并查集-查询

查询关系

查询某个元素的根节点，判断两个元素是否在同一集合



如何判断两个人**同一阵营**?

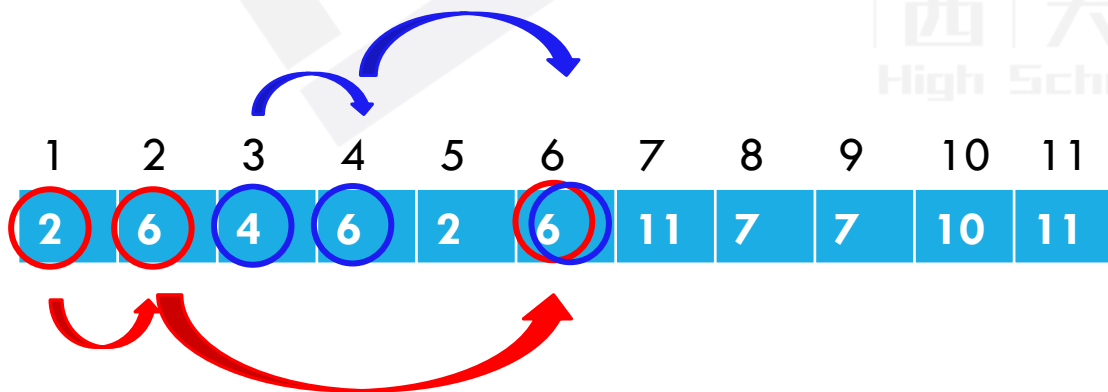
终极boss是否相等

1, 4的终极boss是谁, 如何找到?

一直寻找上级

直到boss是他本身

递归查找



查询操作

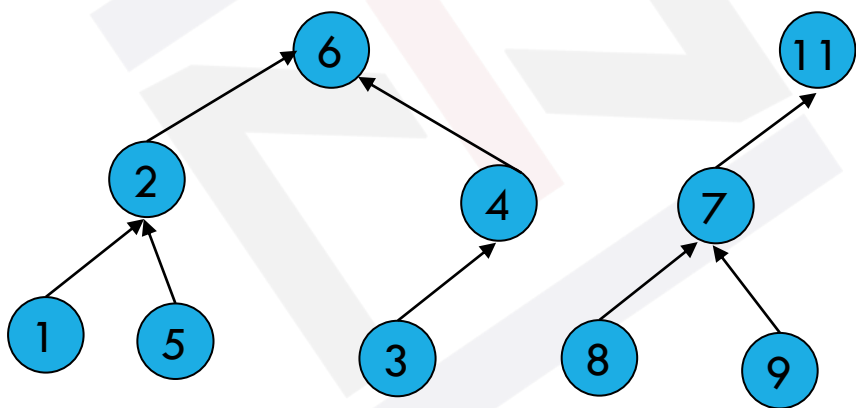
```
int find(int t){ //递归版本
    if(boss[t]!=t) return find(boss[t]);
    return t;
}
```

```
int find(int t){ //非递归版本
    while(boss[t]!=t) t=boss[t];
    return boss[t];
}
```


王的个数

众所周知，擒贼应当先擒王，应该如何去统计？

方法：遍历数组，没有上级关系的点就是“王”



1	2	3	4	5	6	7	8	9	10	11
2	6	4	6	2	6	11	7	11	10	11

统计 $\text{boss}[i]=i$ 的结点个数

路径压缩

如果关系是这样的：

1号强盗与2号强盗是同伙。

2号强盗与3号强盗是同伙。

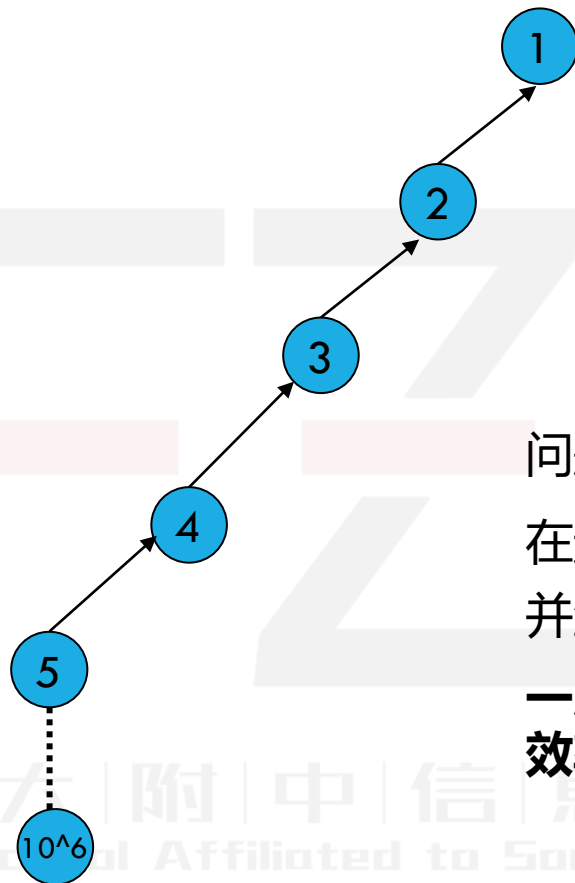
3号强盗与4号强盗是同伙。

4号强盗与5号强盗是同伙。

5号强盗与6号强盗是同伙。

...

999999号强盗与1000000号强盗是同伙。



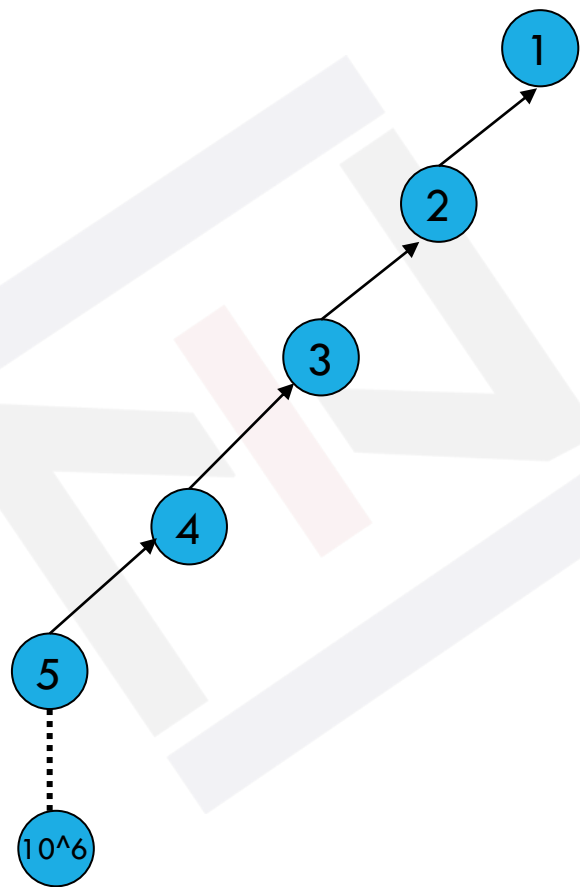
问题：树退化成为了一条链

在这种极端数据下

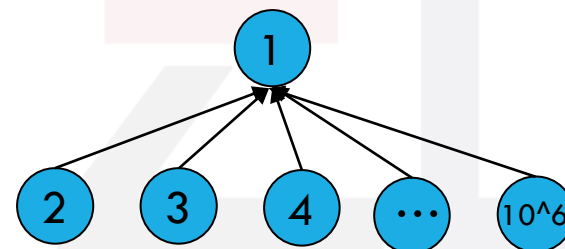
并查集的**效率降低**

一层层的上下级关系导致
效率降低，查找也不方便

路径压缩



如何简化?

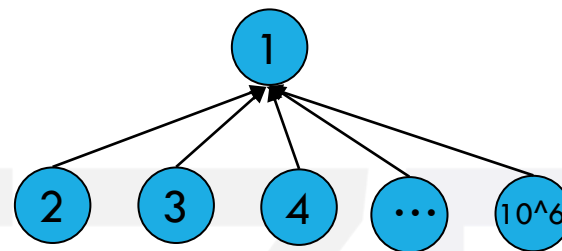


本质上，我们可以认为1是所有结点的“头儿”

路径压缩

方法：将结点直接指向根节点

在本题中，直接把各个小弟指向终极大boss



```
int find(int t){  
    if(boss[t]!=t)  
        return boss[t];  
}
```

//递归版本

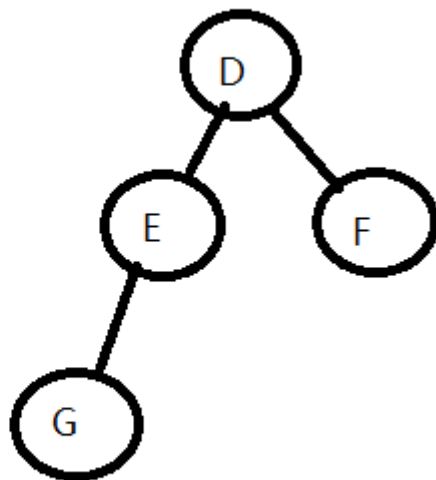
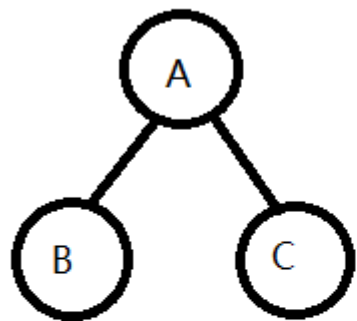
```
boss[t]=find(boss[t]);
```

思考：如何“简单地”改造这个代码？

做法：在递归回溯的时候，顺带将当前结点的上级指向根节点

合并是否也能优化？

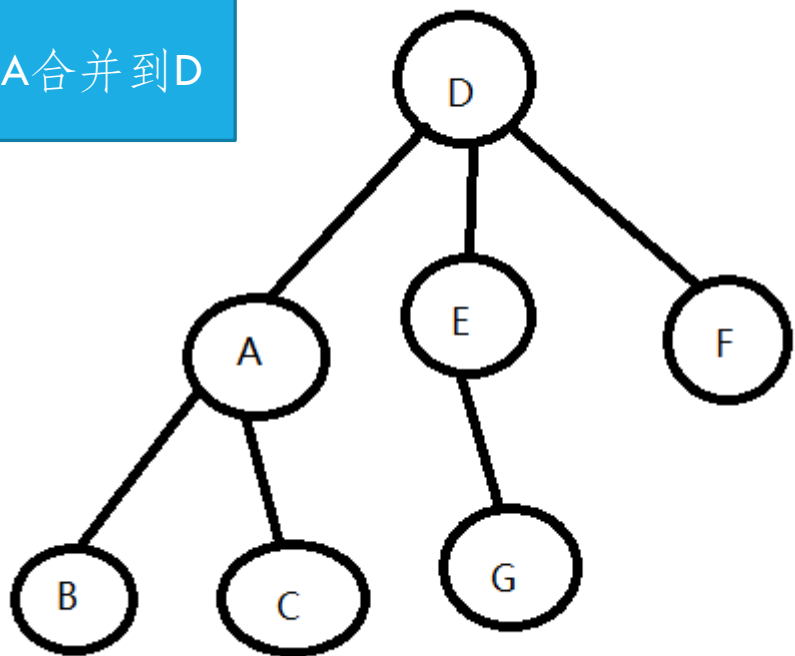
对于两棵高度不一的树，如何合并最优？



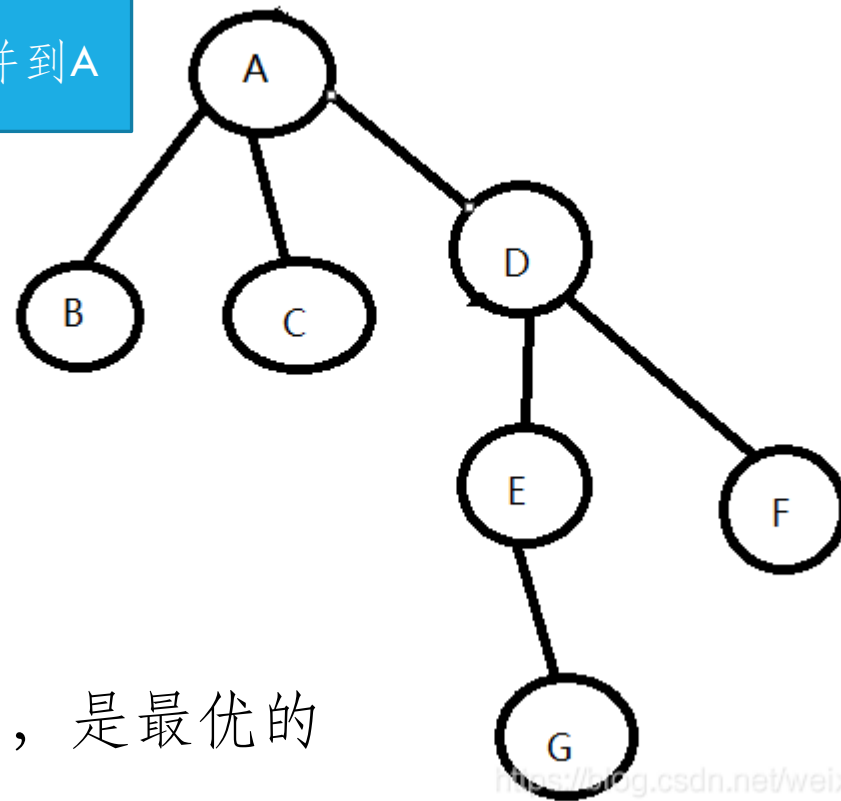
https://blog.csdn.net/weixin_43933323

合并是否也能优化？

A合并到D



D合并到A



显然：将小树合并到大树，是最优的

按秩合并

按秩合并，也称启发式合并，基本思想是使包含较少结点的树的根指向包含较多结点的树的根。
而这个树的大小可以抽象为树的高度，即高度小的树合并到高度大的树，这样资源利用更加合理。

按秩合并

具体做法：维护一个`rank[]`，合并前比较一下

初始化：每个点的`rank`都为0

```
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])
        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

小结

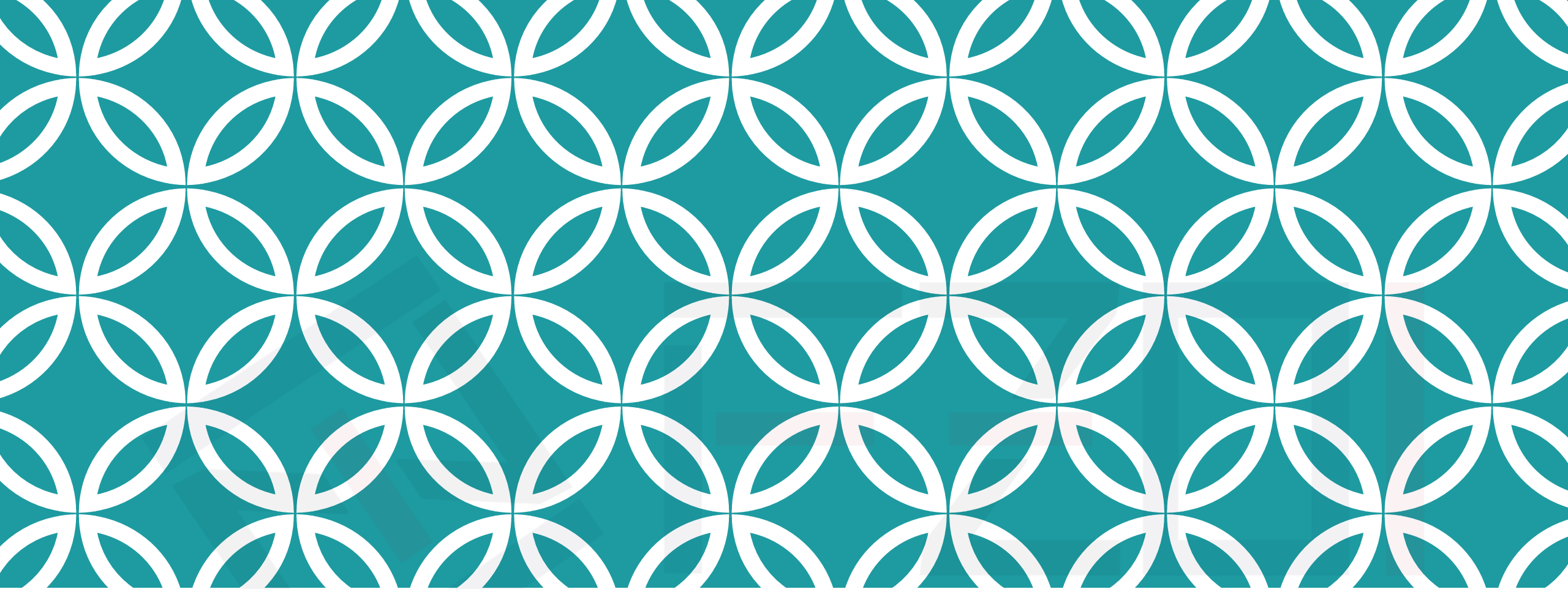
凡是涉及到元素的分组管理问题，都可以考虑使用并查集进行维护

并查集可以维护**连通性、传递性**的关系，例如亲戚的亲戚是亲戚。

并查集还有许多巧妙的运用，也是很多算法的基础，比如后续图论要学习的**Kruscal**算法

一般并查集只维护了“合伙”关系
但有时候，我们要维护另一种或几种关系，如敌人的敌人是朋友。

种类并查集就是为了解决这个问题而诞生的。



种类并查集

| 西 | 大 | 附 | 中 | 信 | 息 | 学 | 竞 | 赛 |
High School Affiliated to Southwest University



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 ($1 \leq N \leq 50,000$) 和K句话 ($0 \leq K \leq 100,000$)，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： 样例输出：

100 7 3

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

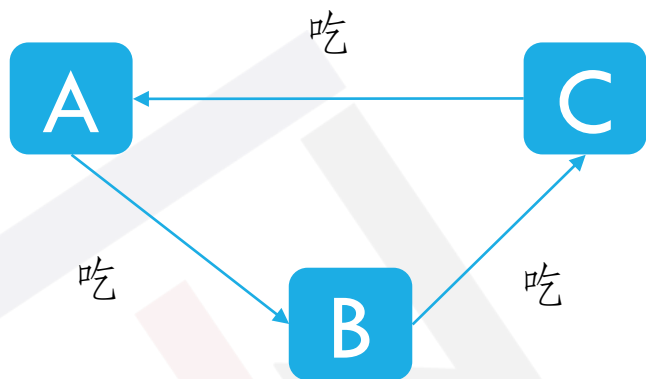
6

7

8

看懂题目，思考题目中有几种关系，如何维护？

食物链



A->B

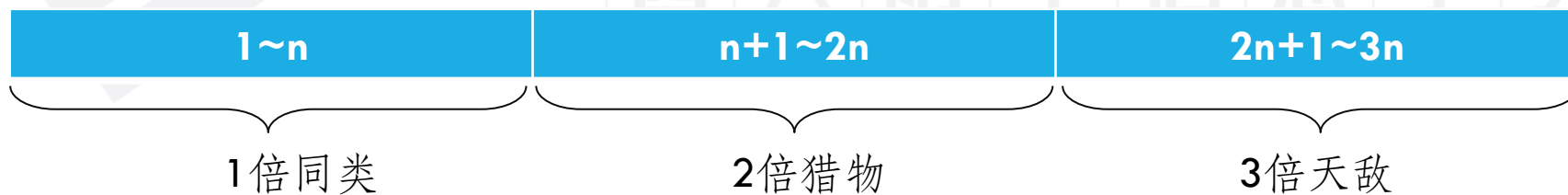
B->C

C->A

在ABC里有三种关系：
同类、猎物、天敌

一般的并查集只能存储“同伙”关系

3倍数组，存三种关系



食物链(种类并查集版)

核心代码:

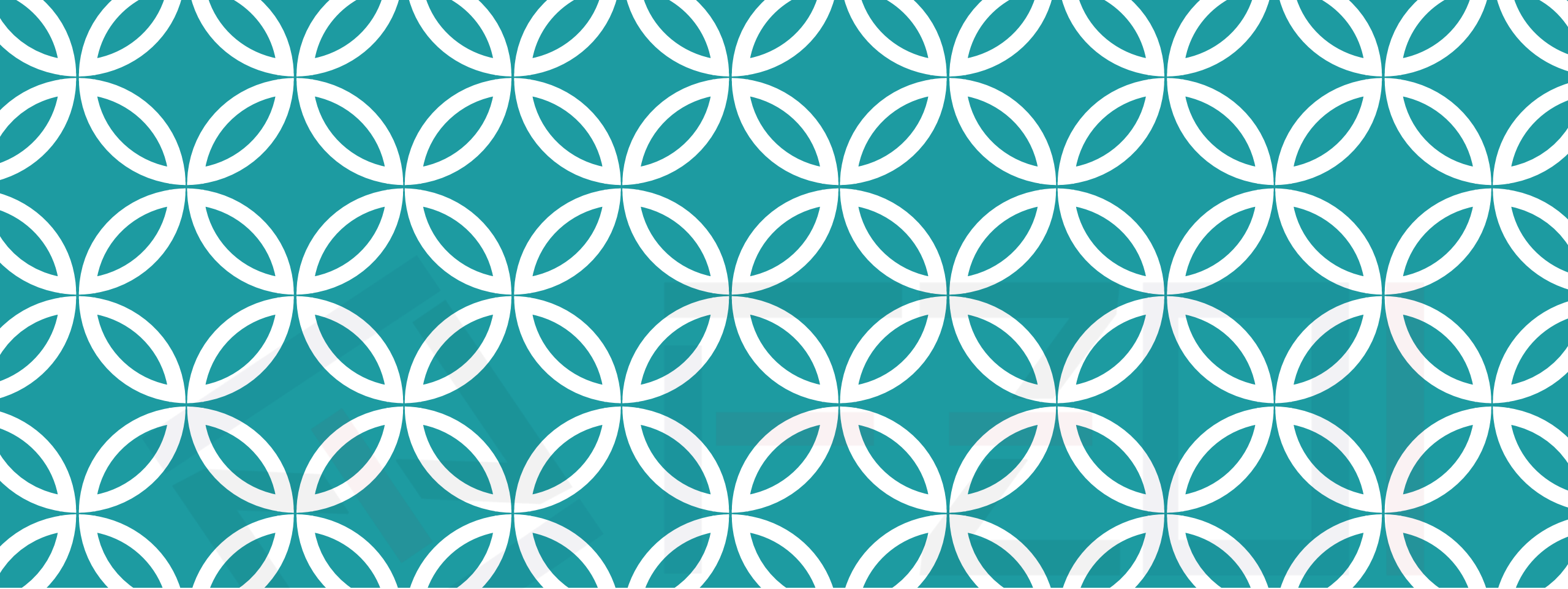
```
int x, y, d;
cin >> n >> k;
for (int i = 1; i <= 3 * n; ++i) //3倍数组
    f[i] = i;
for (int i = 1; i <= k; ++i) {
    cin >> d >> x >> y;
    if (x > n || y > n) {
        ans++;
        continue;
    }
    if (d == 1) { //是同类
        if (find(x + n) == find(y) || find(x + 2 * n) == find(y)) {
  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

种类并查集

当有更多关系的时候，开大数组显然不可行

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

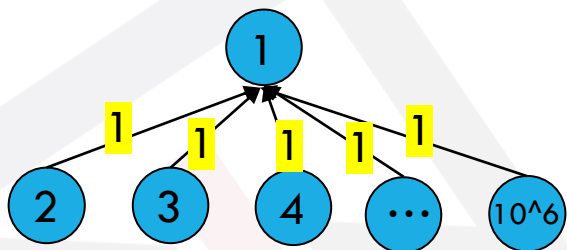


| 西 | 大 | 附 | 中 | 信 | 息 | 学 | 竞 | 赛 |
High School Affiliated to Southwest University

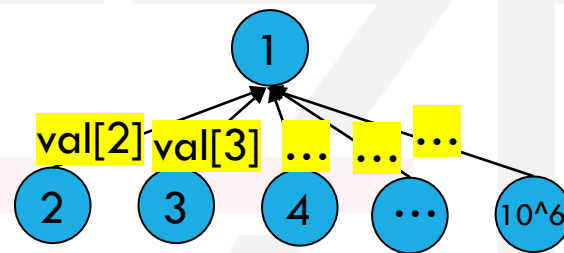
带权并查集



什么是带权并查集？



普通并查集权值为1



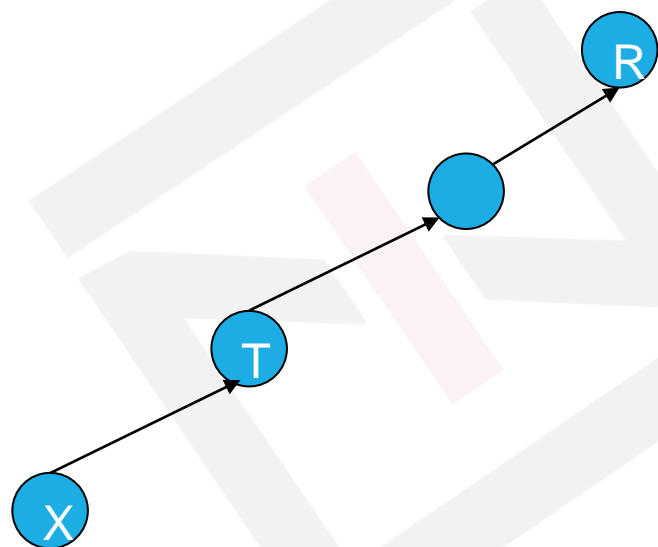
带权并查集权值为 $val[i]$

$val[i]$: 表示从 i 到根结点的权值

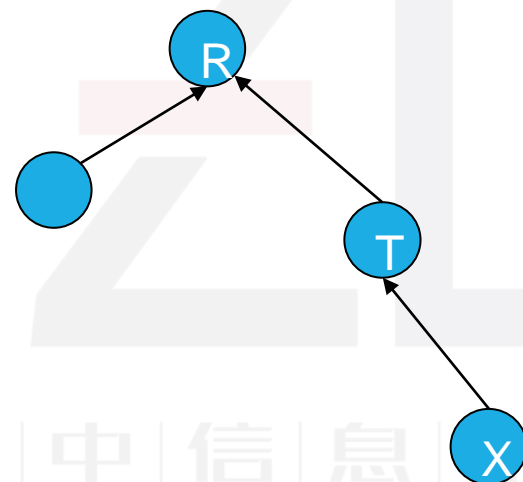
带权并查集可以通过不同的权值表示不同的关系

带权并查集-查询(带路径压缩)

问题：如何计算路径压缩后的 $val[]$?




假如：某次路径压缩后



X本来的权值再加上原父节点的权值，即为当前结点X到根节点新的权值

带权并查集-查询(带路径压缩)

```
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];  
}
```



思考：能否写 `val[x] += val[fa[x]]`?

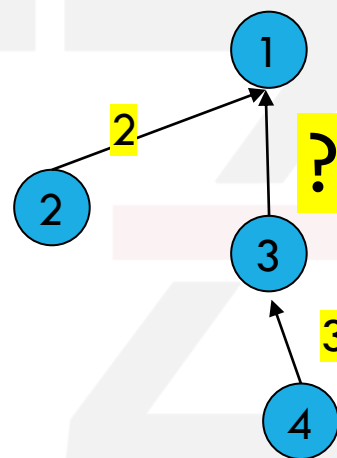
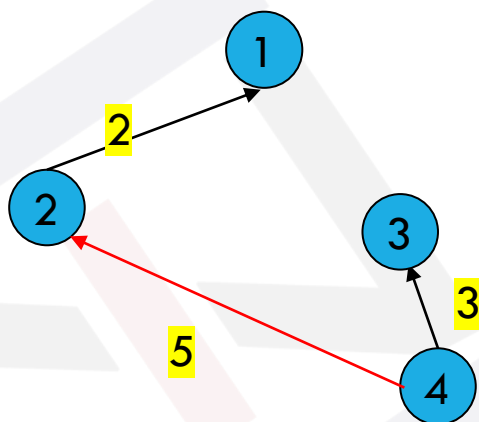
不能，此时的 `fa[x]` 已经不是原来的父节点了

带权并查集

带权并查集需要表示种类关系时，要取余， $\%k$ (k 为关系的种类数)

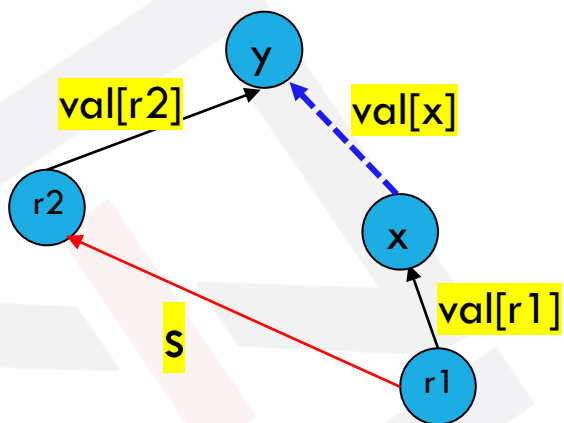
```
int find(int x){  
    if (x != fa[x]){  
        int t = fa[x]; //先记录下原本的父节点  
        fa[x] = find(fa[x]); //路径压缩后父节点会变为根节点  
        val[x] = (val[x]+val[t])%k; //加上原本父节点的权值，即当前结点到  
        根节点的权值  
    }  
    return fa[x];  
}
```

带权并查集-合并



若4和2有了联系，合并之后新的权值如何确定？

带权并查集-合并

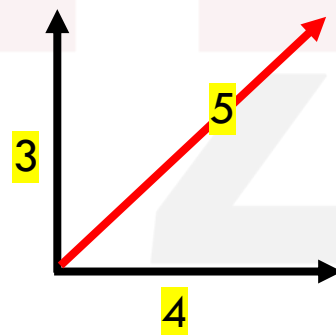


根据向量平行四边形法则：

$$val[x] = -val[r1] + val[r2] + s;$$

科普数学名词：

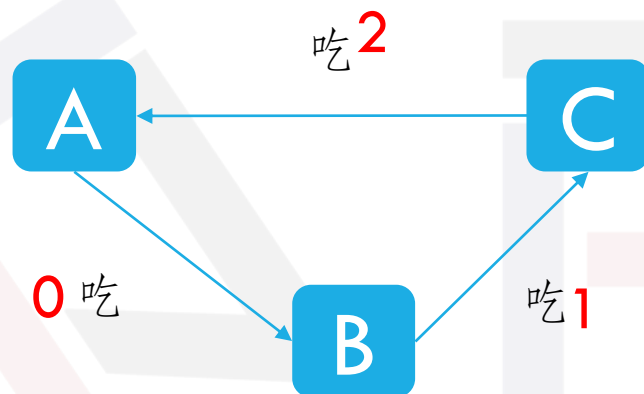
向量：具有方向、大小的线段



带权并查集-合并

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

食物链



A->B 0
B->C 1
C->A 2

$val[x] = (val[x] + val[t]) \% 3;$

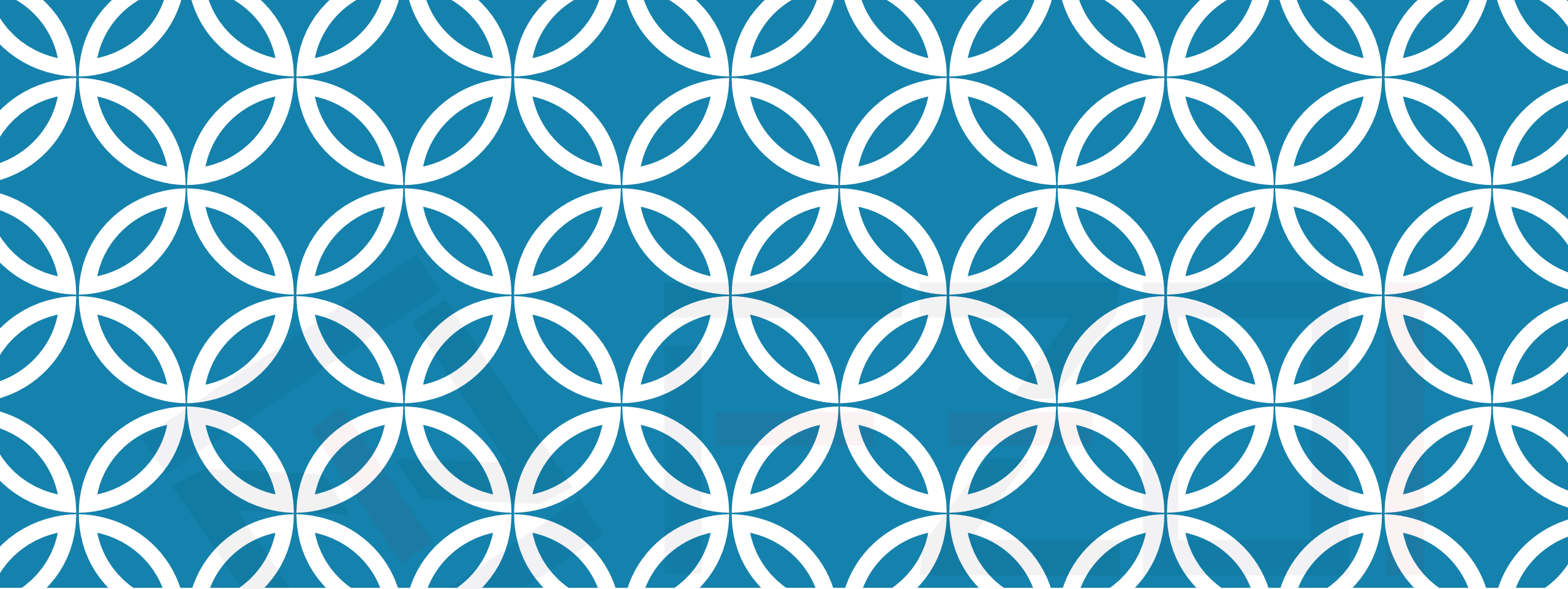
息 | 学 | 竞 | 赛 |

High School Affiliated to Southwest University

食物链(带权并查集版)

```
#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;
}
```



THANKS

| 西 | 大 | 附 | 中 | 信 | 息 | 学 | 竞 | 赛 |
High School Affiliated to Southwest University