并查集

Disjoint Sets

重庆南开信竞基础课程Helang

1.并查集入门

重庆南开信竞基础课程

引例： 亲戚（NKOJ1205）

或许你不知道，你的某个朋友是你的亲戚。他可能是你的曾祖父的外公的女婿的外甥女的表姐的孙子。如果能得到完整的家谱，就可以判断两个人是否亲戚，但如果两个人的最近公共祖先与他们相隔好几代，使得家谱十分庞大，那么检验亲戚关系实非人力所能及。在这种情况下，最好的帮手就是计算机。你将得到一些亲戚关系的信息，如同MarFy和Tom是亲戚，Tom和Ben是亲戚，等等。从这些信息中，你可以推出MarFy和Ben 是亲戚。请写一个程序，对于亲戚关系的提问，快速给出答案。输入由两部分组成：

第一部分以N，M开始。N为问题涉及的人的个数(1 ≤ N ≤ 20000)。这些人编号为1,2,3,…,N。下面有M行(1 ≤ M ≤ 100000)，每行有两个数ai，bi，表示已知ai和bi是亲戚。

第二部分以Q开始。以下Q行有Q个询问(1 ≤ Q ≤ 1 000 000)，每行为ci，di，表示询问ci和di是否为亲戚。对于每个询问ci，di，若ci和di为亲戚，则输出Yes，否则输出No。

样例输入：

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

样例输出：

```
Yes
No
Yes
```

floyd 时间复杂度为$O(n^3)$ n<=20000 不可行

并查集Disjoint Sets

并查集是一种**树型数据结构**，用于处理一些不相交集合的合并问题

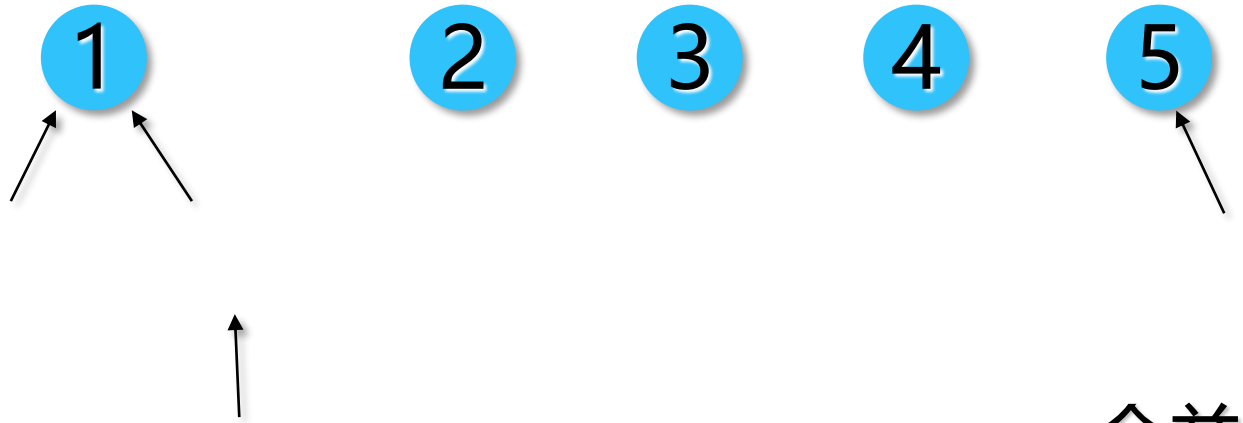
。

并查集的主要操作有：

- 1 - **合并**两个不相交集合
- 2 - **查询**两个元素是否属于同一个集合

元素的合并图示

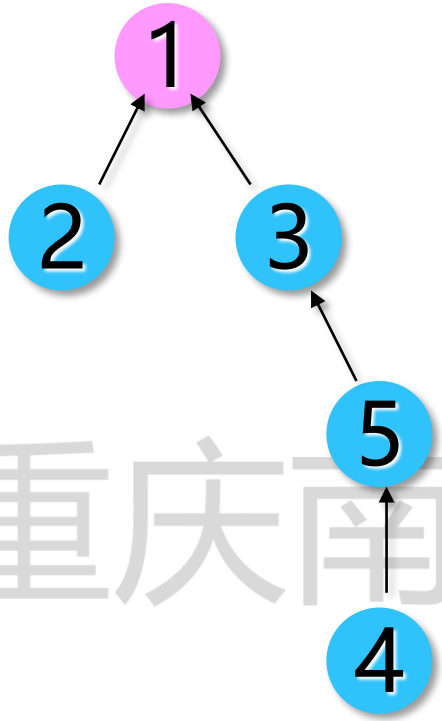
如下图：一开始有5个独立的集合



- 合并1和2
- 合并1和3
- 合并5和4
- 合并5和3

判断元素是否属于同一集合

- 用father[i]表示元素i的父亲结点，如刚才那个图所示



father[1]=1

father[2]=1

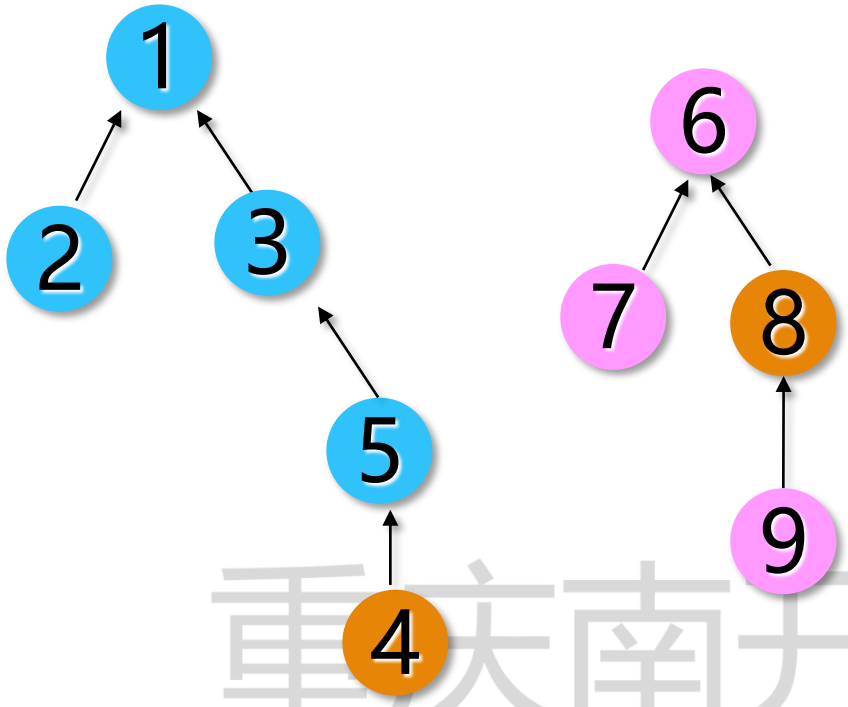
father[3]=1

father[4]=5

father[5]=3

判断元素是否属于同一集合

查询：4和8是否位于同一集合？



1.找出4所在集合的根，1号点

2.找出8所在集合的根，6号点

3.判断两个集合的根是否相同

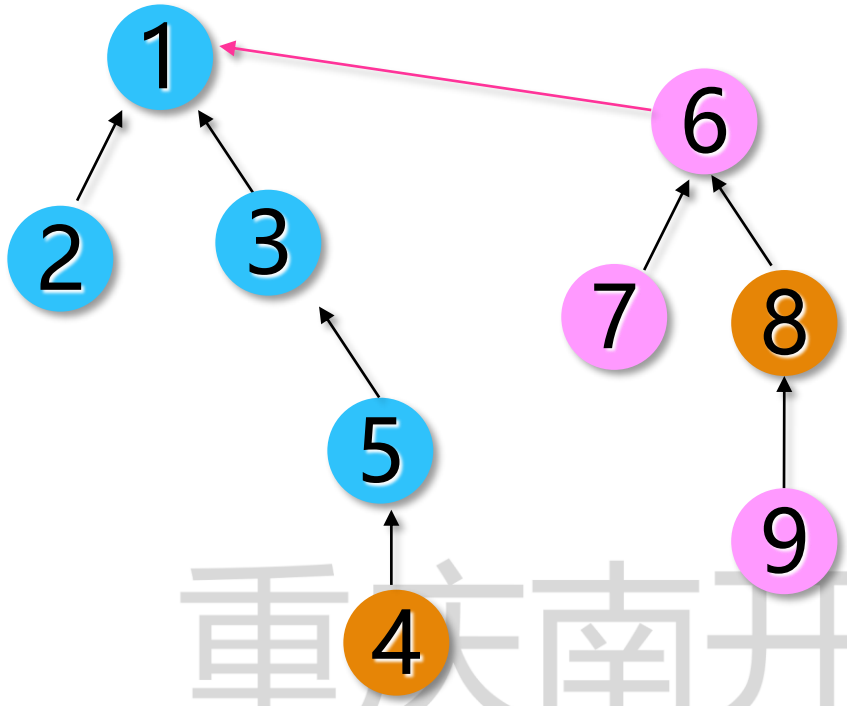
一次查询的时间复杂度？ **O(n)**

复杂度太高了！不行

- 用某个元素所在树的根结点表示该元素所在的集合
- 判断两个元素时候属于同一个集合的时候，只需要判断他们所在树的根结点是否一样即可

合并两个集合集合

合并：4和8所在集合



1.找出4所在集合的根，1号点

2.找出8所在集合的根，6号点

3.将6的父亲设为1

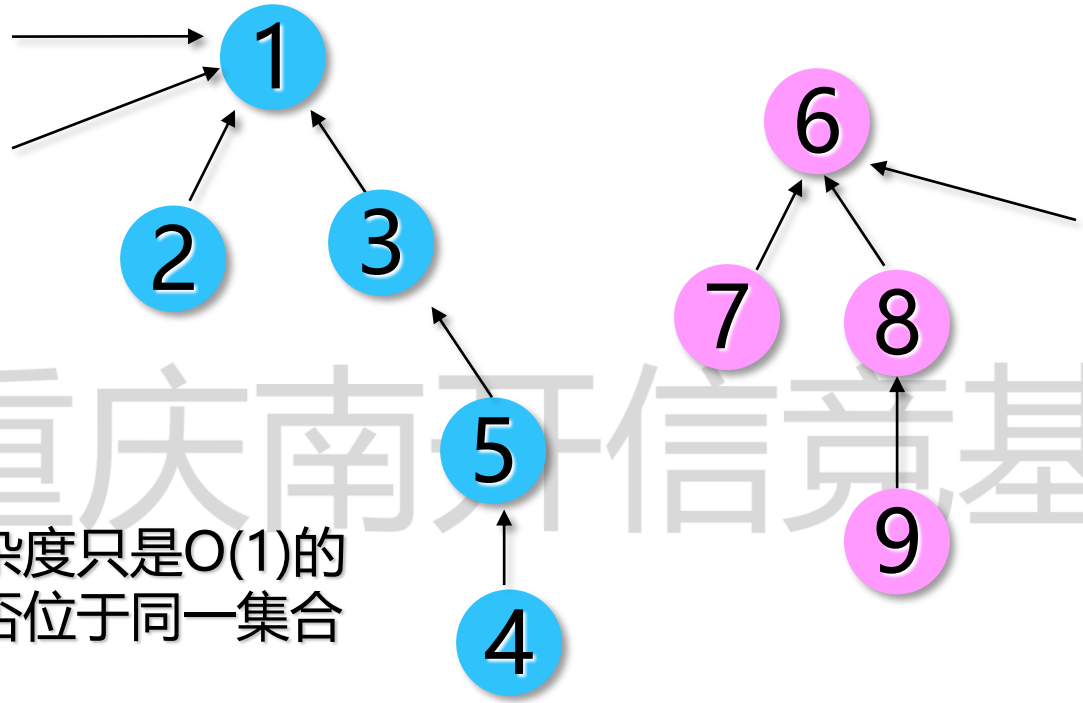
一次合并的时间复杂度？ **O(n)**

复杂度太高了！不行

当我们合并两个集合的时候，只需要在两个根结点之间连边即可

路径压缩

- 判断两个元素是否属于同一集合只需要判断它们所在的树根是否相同，需要**O(N)**的时间来完成，于是路径压缩产生了作用。
- 路径压缩实际上是**把一棵树的根节点设置为所有节点的父亲。**在找完根结点之后，在递归回来的时候顺便把路径上元素的父亲指针都指向根结点

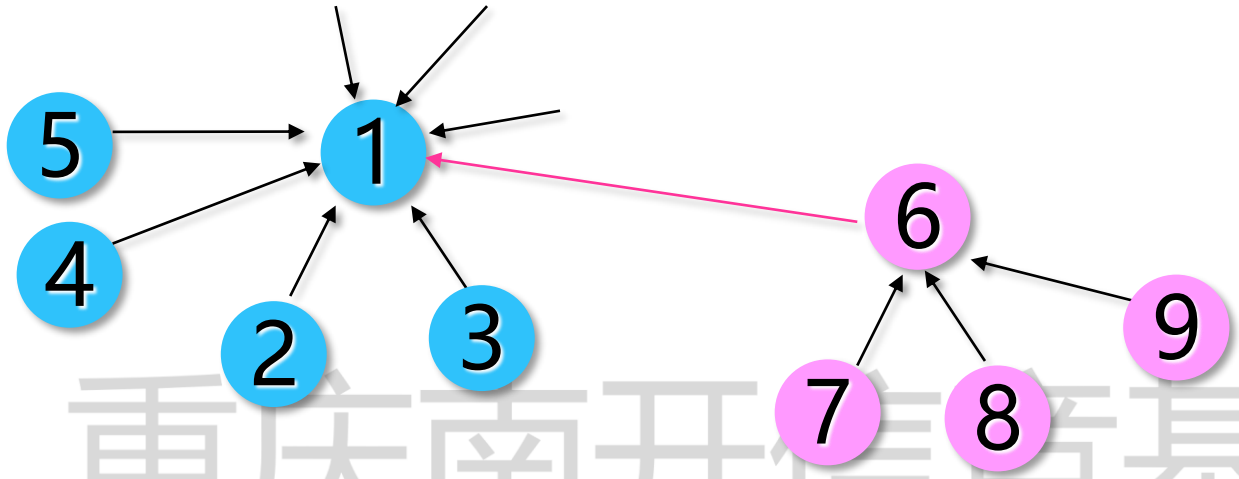


由此我们得到了一个复杂度只是O(1)的算法来判断两个元素是否位于同一集合

合并两个集合

合并4和8所在集合　　直接让1成为6的父亲

如果我们要询问5和9是否位于同一集合，该怎么操作？　　进行路径压缩

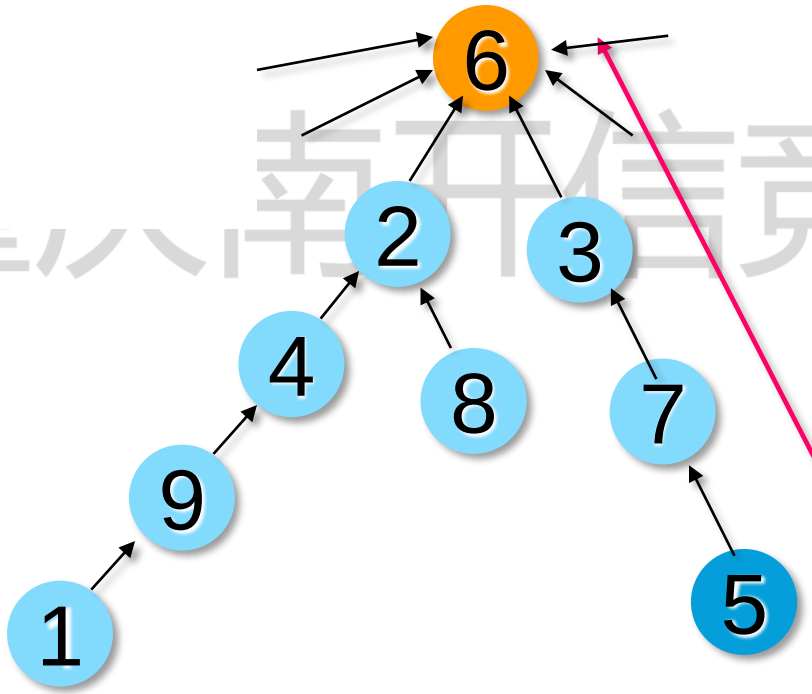


记住：一次路径压缩以后，受该操作影响的所有节点的父亲都被设为根节点。

查询两元素是否位于同一集合的时间复杂度降至O(1)

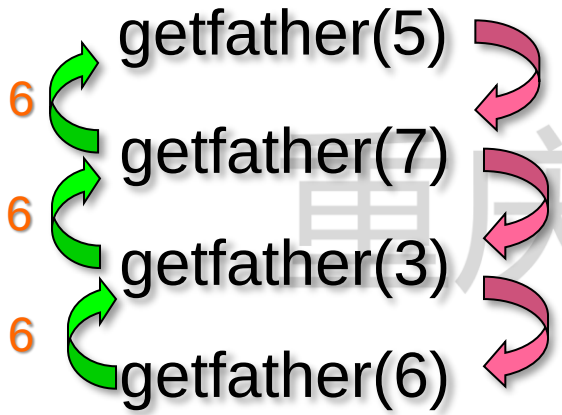
代码:查询同时进行路径压缩


```
int GetFather(int v)          //查询元素v所在集合的根节点
{
    if (father[v]==v) return v;   //v本身为根节点
    else
    {
        father[v]=GetFather(father[v]);
        return father[v];
    }
}
```



通过一次getfater,将5到根之间路径上的所有点的父亲都置为了根节点。

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



再执行getfather(9)
会得到怎样的图形?

```
int getfather(int v)
{
    if (father[v]==v) return v;
    else
    {
        father[v]=getfather(father[v]);
        return father[v];
    }
}
```

只对v到根这条路径上的节点进行路径压缩!

合并两个集合


```
void Merge(int x,int y)                    //合并元素x和元素y所在集合
{
    int fx,fy ;
    fx = getfather(x);                     //先找出x和y所在集合的根
    fy = getfather(y);
    if(fx!=fy)                             //两个根不同，说明x和y位于不同集合
       father[fx] = fy;                    //将fy设为fx的父亲，合并两个集合
}
```

解决引例：亲戚（NKOJ1205）

或许你不知道，你的家谱十分庞大，那么检验亲戚关系实非人力所能及。在这种情况下，最好的帮手就是计算机。你将得到一些亲戚关系的信息，如同MarFy和Tom是亲戚，Tom和Ben是亲戚，等等。从这些信息中，你可以推出MarFy和Ben 是亲戚。请写一个程序，对于亲戚关系的提问，快速给出答案。输入由两部分组成：

第一部分以N，M开始。N为问题涉及的人的个数(1 ≤ N ≤ 20000)。这些人编号为1,2,3,...,N。下面有M行(1 ≤ M ≤ 100000)，每行有两个数ai, bi，表示已知ai和bi是亲戚。

第二部分以Q开始。以下Q行有Q个询问(1 ≤ Q ≤ 1 000 000)，每行为ci, di，表示询问ci和di是否为亲戚。对于每个询问ci, di，若ci和di为亲戚，则输出Yes，否则输出No。

样例输入：

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

样例输出：

```
Yes
No
Yes
```

初始化：

```
for(i=1;i<=N;i++)fathter[i]=i;
```

读入关系

```
for(i=1;i<=M;i++)
{
        cin>>x>>y;
        Merge(x,y);
}
```

已经AC的同学做一做3197

回答询问

```
for(i=1;i<=Q;i++)
{
        cin>>x>>y;
        if(getfather(x)==getfater(y))
             cout<<"Yes";else cout<<"No";
}
```

时间复杂度O(m)

时间复杂度

可以证明，并查集执行m次查找的复杂度为O(m*α(m))

其中α(m)是Ackermann函数的某个反函数，你可以近似的认为它是小于5的。所以并查集的单次查找操作的时间复杂度也几乎是常数级的。

关于阿克曼函数：nkoj 5609

例1：岛屿 NKOI3197

湖面上有 n 座岛屿，编号1到n 。现在要湖上建桥使得岛屿连接起来，桥双向通行。

何老板共进行了m次询问，问题有如下两种：

问题1：他问你a,b两岛是否相互可达，是你就输出Yes；
否则你就输出No，并**在a,b之间建一座桥。**

问题2：他问你从岛c出发可以到达的岛屿的个数(不包括c 自身)，你要快速回答。

$2 \le n \le 10,000,\ 1 \le m \le 30,000$

问题分析：

问题1：查询a,b两个元素是否位于同一个集合，若不是，将两个集合合并。

问题2：询问元素c所在**集合的元素的个数。**

例1：岛屿 NKOI3197

```
int father[100001],cnt[100001];//cnt[i]记录i为根的集合的元素个数
.....
for(i=1;i<=n;i++){ father[i]=i; cnt[i]=1;}   //初始化集合
for(i=1;i<=m;i++)
{
      scanf("%d",&p);
      if(p==1)//对应题目的问题1，询问x,y是否位于同一集合，若不是，合并两个集合
      {
            scanf("%d%d",&x,&y);
            fx=getfather(x);    fy=getfather(y);
            if(fx==fy)printf("Yes\n");
            else
            {
                printf("No\n");
                father[fy]=fx;
                cnt[fx]+=cnt[fy];    //将y所在集合合并到x所在集合，更新x集合的元素个数
            }
       }
       else//对应题目的问题2，询问x所在集合的元素个数
       {
            scanf("%d",&x);
            fx=getfather(x);
            printf("%d\n",cnt[fx]-1);
       }
}
```


重庆南开信竞基础课程

例2：关押罪犯 NOIP2010 nkoi1046

有两座监狱，共关押N 名罪犯，编号分别为1~N。很多罪犯之间甚至积怨已久，随时可能爆发冲突。我们用“怨气值”来表示某两名罪犯之间的仇恨程度。如果两名怨气值为c 的罪犯被关押在同一监狱，他们俩之间会发生摩擦，并造成影响力为c 的冲突事件。

每年警察局会将本年内监狱中的所有冲突事件按影响力从大到小排成一个列表，然后上报市长那里。市长只会去看列表中的第一个事件的影响力，如果影响很坏，他就会考虑撤换警察局长。

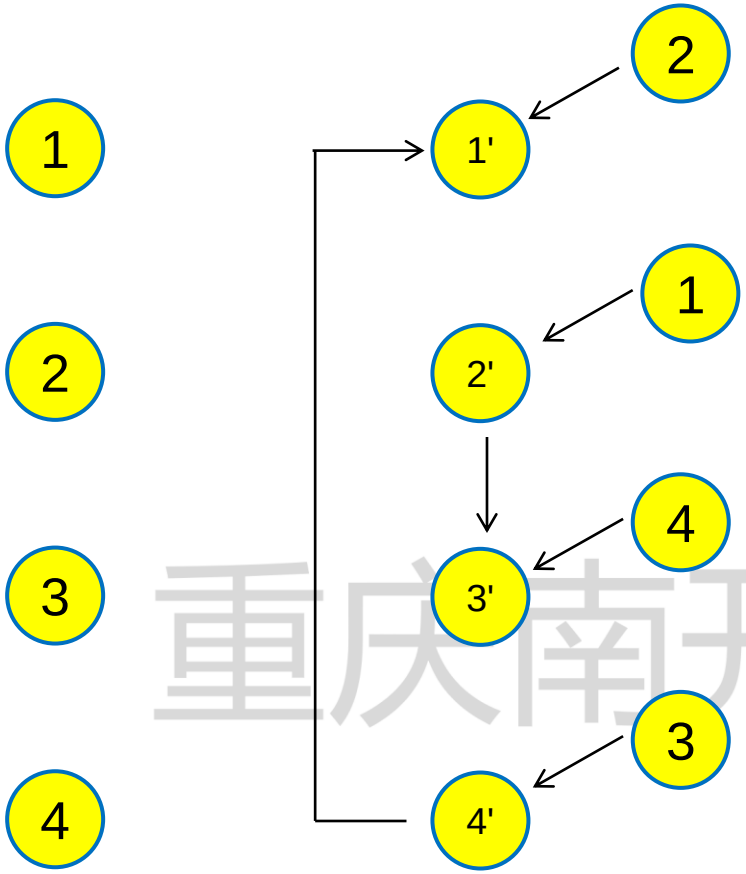
警察局长准备将罪犯们在两座监狱内重新分配，以求产生的冲突事件影响力都较小。假设只要处于同一监狱内的某两个罪犯间有仇恨，那么他们一定会发生摩擦。那么，应如何分配罪犯，才能使 市长看到的那个冲突事件的影响力最小？这个最小值是多少？

N≤ 20000，M≤ 100000。

样例输入	样例输出
4 6 1 4 2534 2 3 3512 1 2 28351 1 3 6618 2 4 1805 3 4 12884	3512

例4：关押罪犯 NOIP2010 nkoi1046

为每个囚犯建立一个对立集合，比如i的对立集合为i'
将不能与i关在一起的囚犯都关到i的对立集合中，如果某一刻对立集合中的囚犯出现了冲突，那说明该冲突无法避免



样例输入

```
4 6
1 4 2534
2 3 3512
1 2 28351
1 3 6618
2 4 1805
3 4 12884
```

先排序，优先安排怨气值大的囚犯

```
1 2 28351
3 4 12884
1 3 6618
2 3 3512
1 4 2534
2 4 1805
```