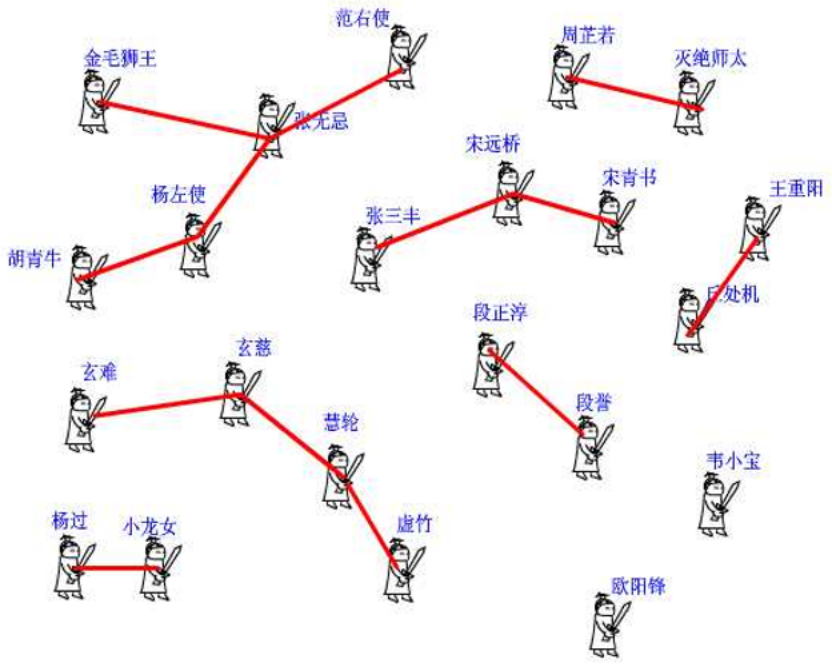
高级数据结构-图

信智学院　陈琰宏

10.3 并查集

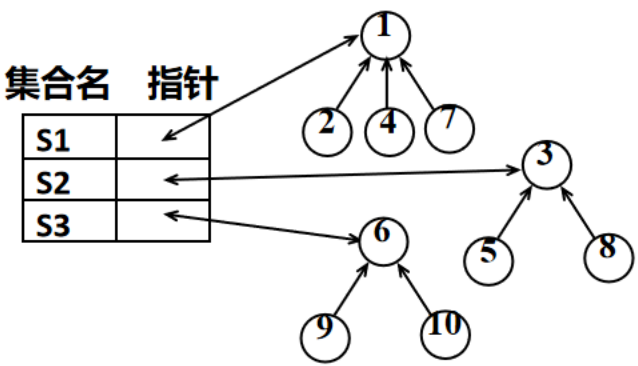
在一些有N个元素的集合应用问题中，我们通常是在开始时让每个元素构成一个单元素的集合，然后按一定顺序将属于同一组的元素所在的集合合并，其间要反复查找一个元素在哪个集合中。这一类问题看似并不复杂，但数据量极大，若用正常的数据结构来描述的话，往往超过了空间的限制，计算机无法承受；即使在空间上能勉强通过，运行的时间复杂度也极高，根本不可能在规定的运行时间内计算出试题需要的结果，只能采用一种特殊数据结构——并查集来描述。

10.3 并查集


范右使
金毛狮王
张无忌
周芷若
灭绝师太
宋远桥
宋青书
王重阳
杨左使
张三丰
胡青牛
段正淳
玄慈
玄难
慧轮
段誉
韦小宝
杨过
小龙女
虚竹
欧阳锋

10.3.1 并查集存储

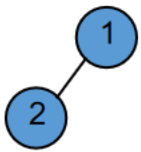
- 逻辑上，可以用树结构表示集合，树的每个结点代表一个集合元素。
- 例如，有三个互不相交的整数集合S1={1,2,4,7}、S2={3,5,8}、S3={6,9,10}，
- 这三个集合的多叉树表示形式：



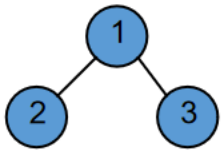
10.3.1 并查集存储

元素的合并图示：

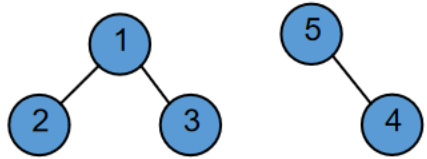
① 合并1和2



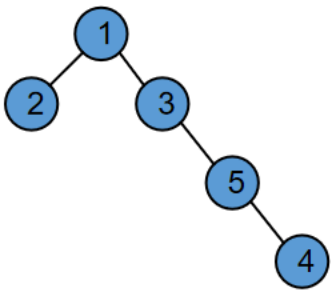
② 合并1和3



③ 合并5和4



④ 合并3和5



用father[i]表示元素i的父亲结点，
进行不断并到不同的集合中

10.3.2 核心代码

寻找父节点：

```
7  int find(int x){
8      while(x!=father[x])
9          x=father[x];//寻找根节点
10     return x;
11 }
```

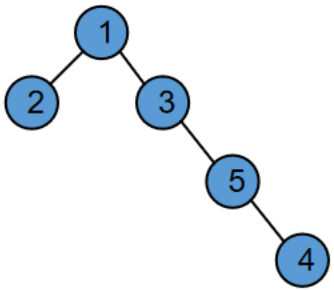
```
20 int find(int x)
21 {
22     if(x==parent[x])
23         return x;
24     int fx=find(parent[x]);
25     parent[x]=fx;
26     return fx;
27 }
```

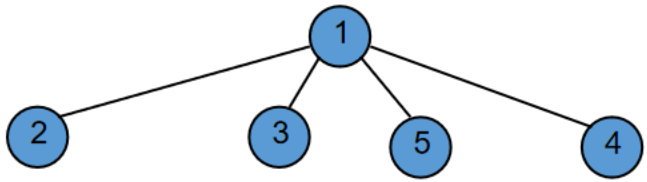
合并成一个集合：

```
12 void union1(int r1,int r2){
13     father[r2]=r1;//合并
14 }
```

优化：路径压缩

两种合并方式





路径压缩实际上是在找完根结点之后，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

不同的合并方式给查询操作带来不同的效率

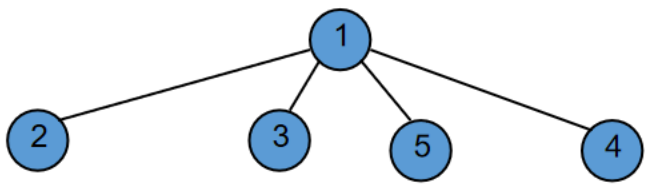
优化：路径压缩

寻找根结点编号并压缩路径：

```
int find (int x)
{
   if (x!=father[x])
        father[x] = find (father[x]);
   return father[x];
}
```

合并两个集合：

```
void unionn(int x,int y)
{
    x = find(x);y = find(y);
    if(x!=y)
    father[y] = x;
}
```



非递归路径压缩

```
 8  int find(int x)
 9  {
10      int k,g;
11      g=x;
12      while(g!=father[g])//查找根节点
13      {
14          g=father[g];
15      }    //找到根节点，用g记录下
16      while(x!=g)//非递归路径压缩操作
17      {
18          k=father[x];
19          father[x]=g; //father[x]指向根节点
20          x=k;
21      }
22      return g;  //返回根节点的值
23  }
```

```
22  int get_f(int x)
23  {//寻找父节点，并做压缩
24      while(x != f[x])
25          x = f[x] = f[f[x]];
26      return x;
27  }
```

10.3.2 路径压缩举例

```
#include <bits/stdc++.h>
using namespace std;
int father[11];
int root1,root2;
void union1(int r1, int r2){
    father[r2]=r1;//合并
}
int find(int x){//路径压缩，每次更新节点的父节点，
//将所有链上的子节点全部链接到父节点上
    if(father[x]!=x)father[x]=find(father[x]);
    return father[x];//查找
}
```

```
链1：1 2 3 4 5
链2：6 7 8 9 10
5和10普通合并后链的父节点：1 2 3 10 6 6 7 8 9 10
```

```
链1：1 2 3 4 5
链2：6 7 8 9 10
5和10压缩合并后链的父节点：1 1 1 1 1 1 6 6 6 6
```

10.3.2 路径压缩举例

```
14  int main(){
15      for(int i=1;i<=10;i++)father[i]=i;//初始化
16      for(int i=10;i>6;i--){
17          father[i]=i-1;//形成6->7->8->9->10的链
18          father[i-5]=i-6;//形成1->2->3->4->5的链
19      }
20      cout<<"链1: ";
21      for(int i=2;i<=5;i++)cout<<father[i]<<" ";
22      cout<<5<<endl;
23      cout<<"链2: ";
24      for(int i=7;i<=10;i++)cout<<father[i]<<" ";
25      cout<<10<<endl;
26      cout<<"5和10压缩合并后链的父节点: ";
27      //union1(10,5);//把5,10合并，普通合并
28      //for(int i=2;i<=10;i++)cout<<father[i]<<" ";
29      //cout<<10<<endl;
30      int r1=find(5),r2=find(10);
31      if(r1!=r2)union1(r1,r2);
32      for(int i=1;i<=10;i++)cout<<father[i]<<" ";
33      return 0;
34  }
```

10.3.3 [2903]亲戚(relation)

或许你并不知道，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，判断两个人是否亲戚应该是可行的，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。为了将问题简化，你将得到一些亲戚关系的信息，如Marry和Tom是亲戚，Tom和Ben是亲戚，等等。从这些信息中，你可以推出Marry和Ben是亲戚。请写一个程序，对于我们的关于亲戚关系的提问，以最快的速度给出答案。

输入格式】

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)**。这些人的编号为1,2,3,…，N。下面有M行(1≤M≤1 000 000)，每行有两个数**a_i，b_i，表示已知a_i和b_i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c_i，d_i，表示询问c_i和d_i是否为亲戚

10.3.3 [2903]亲戚(relation)

【输出格式】
对于每个询问ci，di，输出一行：若ci和di为亲戚，则输出“Yes”，否则输出“No”。

【输入样例】
```
10 7
2 4
5 7
1 3
8 9
1 2
5 6
2 3
3
3 4
7 10
8 9
```

输出样例】
```
Yes
No
Yes
```

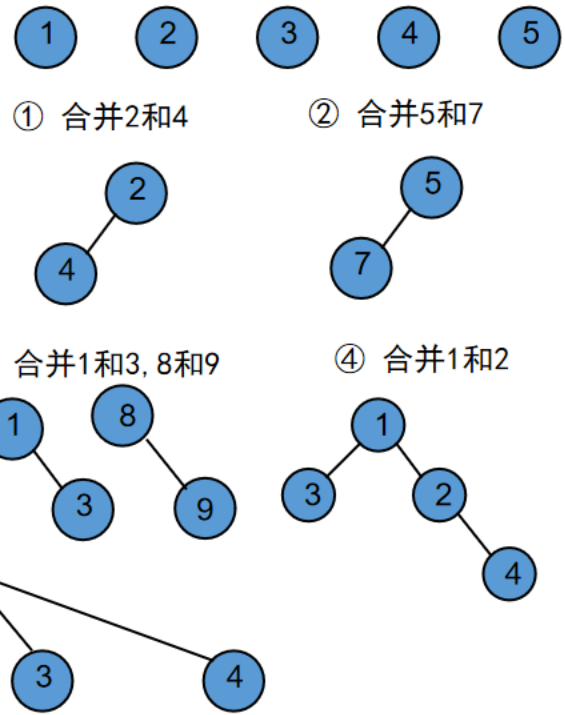
10.3.3 [2903]算法分析

将每个人抽象成为一个点，数据给出M个边的关系，两个人是亲戚的时候两点间有一条边。很自然的就得到了一个N个顶点M条边的图论模型。

用集合的思路，对于每个人建立一个集合，开始的时候集合元素是这个人本身，表示开始时不知道任何人是他的亲戚。以后每次给出一个亲戚关系时，就将两个集合合并。这样实时地得到了在当前状态下的集合关系。

10.3.3 [2903]算法分析

输入关系	分离集合
初始状态	{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}
(2,4)	{1}{2,4}{3}{5}{6}{7}{8}{9}{10}
(5,7)	{1}{2,4}{3}{5,7}{6}{8}{9}{10}
(1,3)	{1,3}{2,4}{5,7}{6}{8}{9}{10}
(8,9)	{1,3}{2,4}{5,7}{6}{8,9}{10}
(1,2)	{1,2,3,4}{5,7}{6}{8,9}{10}
(5,6)	{1,2,3,4}{5,6,7}{8,9}{10}
(2,3)	{1,2,3,4}{5,6,7}{8,9}{10}



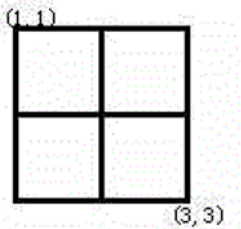
10.3.3 [2903]代码

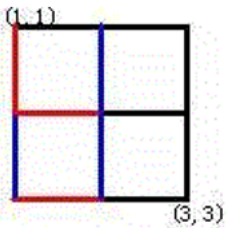
```
3   int father[20000+5];
4   void union1(int r1, int r2){
5       father[r2]=r1;//合并
6   }
7   int find(int x){
8       while(father[x]!=x)x=father[x];
9       return x;//查找
10  }
```

```
12  int main()
13  {
14      int n,m,q,x,y;
15      cin>>n>>m;
16      for(int i=1;i<=n;i++)
17          father[i]=i;//初始化每个结点
18      for(int i=1;i<=m;i++){
19          scanf("%d %d",&x,&y);
20          int r1=find(x),r2=find(y);
21          if(r1!=r2)union1(r1,r2);
22      }//完成合并操作
23      cin>>q;
24      for(int i=1;i<=q;i++){
25          scanf("%d %d",&x,&y);
26          int r1=find(x),r2=find(y);
27          if(r1==r2)printf("Yes\n");
28          else printf("No\n");
29      }
30      return 0;
31  }
```

12.2 [2908] 格子游戏

Alice和Bob玩了一个古老的游戏：首先画一个n × n的点阵（下图n = 3）接着，他们两个轮流在相邻的点之间画上红边和蓝边：





直到围成一个封闭的圈（面积不必为1）为止，**“封圈”的那个人就是赢家**。因为棋盘实在是太大了(n ≤300)，他们的游戏实在是太长了！他们甚至在游戏中都不知道谁赢得了游戏。于是请你写一个程序，帮助他们计算他们是否结束了游戏?

12.2.1 [2908] 格子游戏

输入
输入数据第一行为两个整数n和m。m表示一共画了m条线。以后m行，每行首先有两个数字(x, y)，代表了画线的起点坐标，接着用空格隔开一个字符，假如字符是"D "，则是向下连一条边，如果是"R "就是向右连一条边。输入数据不会有重复的边且保证正确。
输出
输出一行：在第几步的时候结束。假如m步之后也没有结束，则输出一行“draw”。
样例输入
3 5
1 1 D
1 1 R
1 2 D
2 1 R
2 2 D
样例输出
4

分析

题目讲到“封圈”的那个人就是赢家，可以理解为如果即将连笔的两个位置都在一个集合内，那么只要在添加一笔就形成封圈。这样题目就转换为判断端点是非在同一个集合内的问题，即并查集问题。

每一步形成环等价于划线的两个端点已经在同一个集合里面。把二位坐标转化为一维坐标

(x, y)= n*x+y

这道题用结构体实现二维的并查集，在每一次比较的时候只需要把x和y都比较一下就可以；
每一次划线，先判断一下，如果起点在集合里，终点也在集合里，那么符合条件，输出当时的步数就可以了；如果不符合上面的条件，把点入集合；如果到最后还没有符合条件的情况，输出draw就可以了。

【参考程序】

```
#include <iostream>
using namespace std;
struct node
{ int x,y;}f[301][301],k1,k2;
int i,j,m,n,x,y;
char c;
node root(node k)
{
  if((f[k.x][k.y].x==k.x)&&(f[k.x][k.y].y==k.y))return k;
  f[k.x][k.y]=root(f[k.x][k.y]);
  return f[k.x][k.y];
}
```

并查集的基本思想

```
int main()
{
  cin>>n>>m;
  for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
       {f[i][j].x=i;f[i][j].y=j;}
for(i=1;i<=m;i++)
   {
     cin>>x>>y>>c;
     if(c=='D')
        {k1=root(f[x][y]);k2=root(f[x+1][y]);}
     if(c=='R')
        {k1=root(f[x][y]);k2=root(f[x][y+1]);}
     if((k1.x==k2.x)&&(k1.y==k2.y))
         {cout<<i<<endl;return 0;}
       else f[k1.x][k1.y]=k2;
    }
  cout<<"draw"<<endl;
  return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;

//做题时的记录
//算法：并查集
//思路：如果成"圈"，那么必定有一个环
//                -> 可以用树来存联通的点
//                -> 合并过程像merge -> 并查集
//复杂度：约O((n ^ 2) log (n ^ 2))
//实现细节：二维转一维 -> (i, j) <=> i * n + j

typedef long long ll;

const int N = 40009;
int n, m, p[N];

int find(int x)
{
    if(x == p[x]) return x;
    else return p[x] = find(p[x]);
}

void merge(int x, int y)
{
    p[find(x)] = find(y);
}

int main() {
    cin >> n >> m;
    for(int i = 1;i <= n * n;++ i) p[i] = i;//初始化放cin后~~

    char c;
    for(int i = 1, a, b;i <= m;++ i)
    {
        cin >> a >> b >> c;//注意这不是输入的点的坐标

        int x = (a - 1) * n + b, y;//点的坐标
        if(c == 'D') y = a * n + b;//向下连边
        else y = (a - 1) * n + b + 1;//向右连边

        if(find(x) == find(y))//如果在同一格子内
        {
            cout << i << endl;
            return 0;
        }
        merge(x, y);//合并两个集合
    }
    cout << "draw" << endl;
    return 0;
}
```

12.2.2 [2930] 搭配购买(buy)

Joe觉得云朵很美，决定去山上的商店买一些云朵。商店里有n朵云，云朵被编号为1，2，……，n，并且每朵云都有一个价值。但是商店老板跟他说，一些云朵要搭配来买才好，所以买一朵云则与这朵云有搭配的云都要买。

但是Joe的钱有限，所以他希望买的价值越多越好。

输入
第1行n，m，w，表示n朵云，m个搭配，Joe有w的钱。
第2~n+1行，每行ci，di表示i朵云的价钱和价值。
第n+2~n+1+m行，每行ui，vi，表示买ui就必须买vi，同理，如果买vi就必须买ui。

输出
一行，表示可以获得的最大价值。

样例输入	样例输出
5 3 10 3 10 3 10 3 10 5 100 10 1 1 3 3 2 4 2	1

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 10010;
int price[N],value[N];
int n,m,money;
int p[N];
int f[N];
int find(int x)
{
    if (p[x]!=x) p[x]=find(p[x]);
    return p[x];
}
int main()
{
    cin>>n>>m>>money;
    for (int i=1;i<=n;i++) p[i]=i; //每件物品初始化时都是独立的
    //此时输入每件物品的价格和价值，实际完成了并查集维护的价格数组和价值数组的初始化
    for (int i=1;i<=n;i++) cin>>price[i]>>value[i];
    while (m--)
    {
        int a,b;
        cin>>a>>b;
        int  pa=find(a),pb=find(b);
        if (pa!=pb)
        {
            price[pb]+=price[pa];
            value[pb]+=value[pa];
            p[pa]=pb;
        }
    }
    //此时，完成了需要搭配的物品之间的连通。
    /*接下来，就转化成了：最多money的钱，共有已知数量的物品
    每个物品最多选一次，问能最多选取的价值为多少？
    */
    //这不就是典型的0-1背包
    for (int i=1;i<=n;i++) //正向枚举每件物品
    {
        if (find(i)==i) //只枚举祖宗节点的情况，祖宗节点代表连通后的一件物品
        {
            for (int j=money;j>=price[i];j--) //反向枚举钱，一维0-1背包的注意事项
                f[j]=max(f[j],f[j-price[i]]+value[i]); //取max
        }
    }
    cout<<f[money]<<endl;
    return 0;
}
```

12.2.3 [7292] 银河英雄传说

有一个划分为N列的星际战场，各列依次编号为1,2,…,N。

有N艘战舰，也依次编号为1,2,…,N,其中第i号战舰处于第i列。

有T条指令，每条指令格式为以下两种之一：

1、Mij，表示让第i号战舰所在列的全部战舰保持原有顺序，接在第j号战舰所在列的尾部。

2、Cij，表示询问第i号战舰与第j号战舰当前是否处于同一列中，如果在同一列中，它们之间间隔了多少艘战舰。

现在需要你编写一个程序，处理一系列的指令。

输入格式

第一行包含整数T，表示共有T条指令。
接下来T行，每行一个指令，指令有两种形式：M i j或C i j。
其中M和C为大写字母表示指令类型，i和j为整数，表示指令涉及的战舰编号。

输出格式

你的程序应当依次对输入的每一条指令进行分析和处理：
如果是M i j形式，则表示舰队排列发生了变化，你的程序要注意到这一点，但是不要输出任何信息；
如果是C i j形式，你的程序要输出一行，仅包含一个整数，表示在同一列上，第i号战舰与第j号战舰之间布置的战舰数目，如果第i号战舰与第j号战舰当前不在同一列上，则输出-1。
数据范围

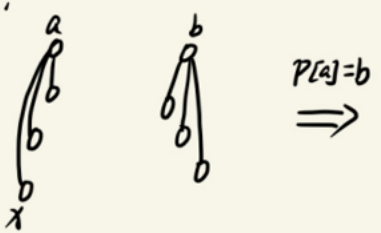
N≤30000,T≤500000

样例输入
4
M 2 3
C 1 2
M 2 4
C 4 2
样例输出
-1
1

什么题目要用到并查集?那就是具有非常明显的传递关系的题目,或者说并查集擅长动态维护许多具有传递性的关系,能在无向图中维护节点之间的连通性.

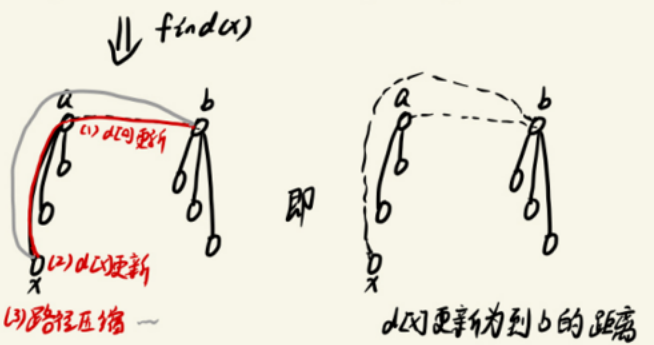
这道题目要注意的就是,我们要边带权,也就是我们不能只处理集合的关系,而是要多一个附带的数组,这个数组就来记录这道题目中最特殊的间隔了多少战舰.听上去很高大上的边带权,实际上就是格外多了一个数组跟随着merge和find一起走而已.也就多了两三行代码,精简容易得很,直接上代码.

图解：

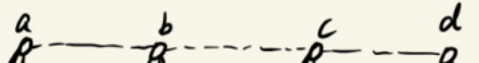


d[x]表示x到当前祖宗的距离

每find一次就会把x到最新祖宗的路径上的点的d[i]全部更新一遍，并路径压缩



p[a]=b　　p[b]=c　　p[c]=d



此时find(x)

12.2.4 [2926] 团伙

在某城市里住着n个人，任何两个认识的人不是朋友就是敌人，而且满足：

1、我朋友的朋友是我的朋友；

2、我敌人的敌人是我的朋友；

所有是朋友的人组成一个团伙。告诉你关于这n个人的m条信息，即某两个人是朋友，或者某两个人是敌人，请你编写一个程序，计算出这个城市最多可能有多少个团伙？

【输入格式】

第1行为n和m，1<n<1000,1<=m<=100 000；

以下m行，每行为p x y，p的值为0或1，p为0时，表示x和y是朋友，p为1时，表示x和y是敌人。

【输出格式】

一个整数，表示这n个人最多可能有几个团伙。

输入	输出
6 4 1 1 4 0 3 5 0 4 6 1 1 2	3

12.2.4 [2926]打击犯罪

某个地区有n(n<=1000)个犯罪团伙，当地警方按照他们的危险程度由高到低给他们编号为1-n，他们有些团伙之间有直接联系，但是任意两个团伙都可以通过直接或间接的方式联系，这样这里就形成了一个庞大的犯罪集团，犯罪集团的危险程度唯一由集团内的犯罪团伙数量确定，而与单个犯罪团伙的危险程度无关（该犯罪集团的危险程度为n）。现在当地警方希望花尽量少的时间（即打击掉尽量少的团伙），使得庞大的犯罪集团分离成若干个较小的集团，并且他们中最大的一个的危险程度不超过n/2。为达到最好的效果，他们将按顺序打击掉编号1到k的犯罪团伙，请编程求出k的最小值。

【输入格式】

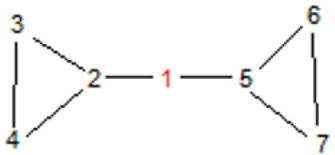
第一行一个正整数n。接下来的n行每行有若干个正整数，第一个整数表示该行除第一个外还有多少个整数，若第i行存在正整数k，表示i，k两个团伙可以直接联系。

【输出格式】

一个正整数，为k的最小值

[2926]打击犯罪

【样例输入】
7
2 2 5
3 1 3 4
2 2 4
2 2 3
3 1 6 7
2 5 7
2 5 6
【样例输出】
1
【提示】



输出1（打击掉红色团伙）

12.2.5 [2932]家谱

现代的人对于本家族血统越来越感兴趣，现在给出充足的父子关系，请你编写程序找到某个人的最早的祖先。

输入文件由多行组成，首先是一系列有关父子关系的描述，其中每一组父子关系由二行组成，用#name的形式描写一组父子关系中的父亲的名字，用+name的形式描写一组父子关系中的儿子的名字；接下来用?name的形式表示要求该人的最早的祖先；最后用单独的一个$表示文件结束。规定每个人的名字都有且只有6个字符，而且首字母大写，且没有任意两个人的名字相同。最多可能有1000组父子关系，总人数最多可能达到50000人，家谱中的记载不超过30代。

【输出格式】

按照输入文件的要求顺序，求出每一个要找祖先的人的祖先，格式：本人的名字+一个空格+祖先的名字+回车。

12.2.5 [2932]家谱

【输入样例】

```
#George
+Rodney
#Arthur
+Gareth
+Walter
#Gareth
+Edward
?Edward
?Walter
?Rodney
?Arthur
$
```

【输出样例】

```
Edward Arthur
Walter Arthur
Rodney George
Arthur Arthur
```

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
const int maxn = 1e2 + 10;
const int inf = 0x3f3f3f3f;
int n, fa[111];
struct node {
    int u, v, dis;
} e[10000 + 10];

bool cmp (node a, node b) {
    return a.dis < b.dis;
}

int find_fa(int x) {//寻找父节点
    if(fa[x] == x) return x;
    return fa[x] = find_fa(fa[x]);
}
///合并两个集合、即合并两颗树
int Union(int x, int y) {
    int fx = find_fa(x), fy = find_fa(y);
    if(fx == fy) return 0;
    fa[fx] = fy;
    return 1;
}
```

并查集：

1. 将两个集合合并
2. 询问两个元素是否在一个集合当中

基本原理：每个集合用一棵树来表示。树根的编号就是整个集合的编号。每个节点存储它的父节点，p[x]表示x的父节点

问题1：如何判断树根：if (p[x] == x)

问题2：如何求x的集合编号：while (p[x] != x) x = p[x];

问题3：如何合并两个集合：px 是x的集合编号，py是y的集合编号。p[x] = y

今天的课程结束啦……

下课了…
同学们再见！