马上开始

35tang-C++竞赛系列三阶课程

三五堂

《35tang-C++竞赛系列三阶课程》

本节目标

- 并查集复习
- 特殊的图
- 入度和出度
- 并查集和图的连通
- 实际上就是用DFS，并查集，入度出度等各种手段来处理一些图的问题。

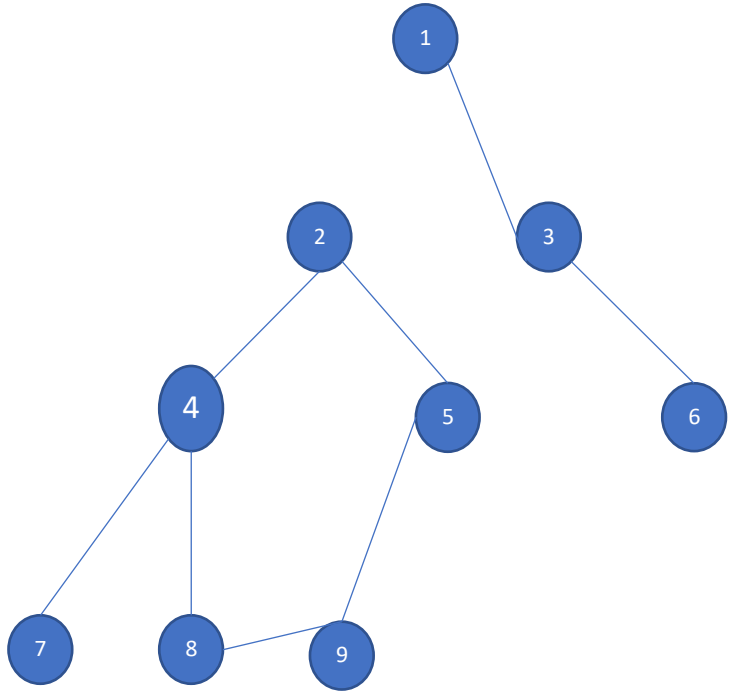
并查集找环

- 什么是环？就是两个点之间的道路不止一条，有图2，4，5节点就形成了一个环。

如何判断一个图有没有环？

如果用DFS，可以从所有的没有访问过的点出发，找所有连接到的点（访问过设置visited标记），什么时候有相邻节点被访问过，也就是第二次访问，就说明出现了环。还要考虑传递当前节点的上一个节点作为参数，避免找到上一个节点被访问过而当作环的情况。

并查集呢？其实就是不断的union集合，什么时候union两个点的时候发现他们所在的集合相同，说明这两个点曾经连接过，有环。



给定若干个城市，编号1到n，（n<=10^3），他们之间m条双向道路(m <=10^5)。

输入：第1行2个整数n,m空格隔开，后面第2行倒m+1行为m条边，每行两个整数a b表示节点a和节点b有双向道路。道路不会重复。

输出：一个整数，判断有没有两个城市之间出现了环（就是两个城市之间至少有两条不同的道路可以连接）（连接指的是：不一定有直接的道路相连，只要互相间接通过道路可达即可），有环输出1，否则输出0

示例输入1：

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

示例输出1:

```
1
```

示例输入2：

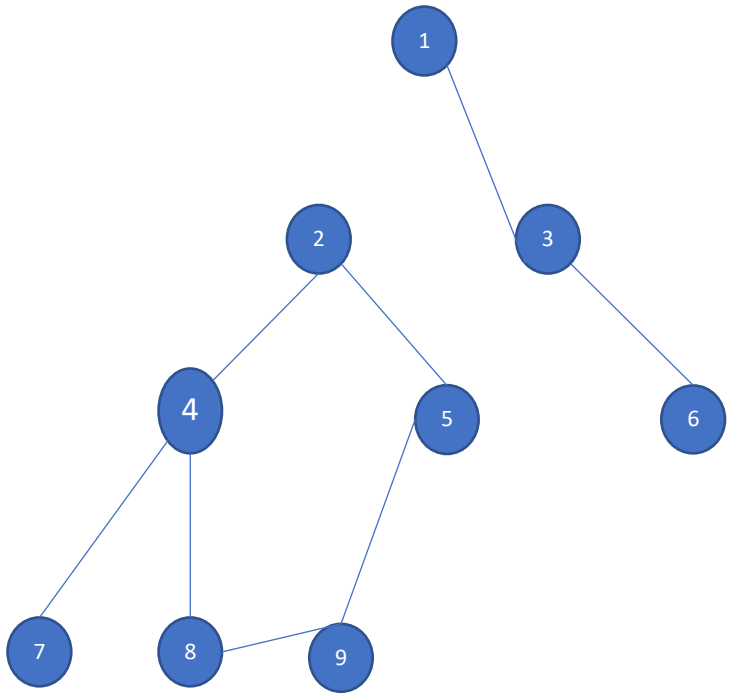
```
9 8
1 2
3 1
2 4
5 2
3 6
4 7
8 9
5 9
```

示例输出2:

```
0
```

画图理解示例数据

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

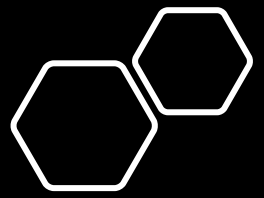


并查集模板程序就是并和查：

读入每一条边的两个节点：查找两个节点当前的集合，

合并这两个集合。

```
int n,m;
int parent[1001];
int findroot(int x)//查,寻找x在哪一个集合
{
  if (parent[x]==x) return x;
  return (findroot(parent[x]));
  //递归查找集合的代表元素也就是根元素
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)  //初始化个集合，数组值等于小标的点为根节点。
                parent[i]=i;
    for (int i=1;i<=m;i++)
   {
                int a,b;
                cin>>a>>b;  //读入两个节点，比如1，3
                int x=findroot(a),y=findroot(b);//找到ab所在集合的根元素
                if(x==y) {cout<<1<<endl; return 0;}//有环，也是模板程序唯一的修改
                parent[x]=y;  //合并两个根所在集合
 }
  cout<<0<<endl;
  return 0;
}
```



例：
亲戚关系

给出n个人，编号1到n，他们之间有些人有亲戚关系，给出m对存在亲戚关系的人，求任意给出的两个人是否具有亲戚关系。这里：x和y是亲戚，y和z是亲戚，那么x和z也是亲戚。

输入格式（文件名relation.in）：

第一行：三个整数n,m,k（n<=10^5,m<=10^5,k<=10^5），分别表示有n个人，m个亲戚关系，求k对人是否有亲戚关系。

以下m行：每行两个数a，b (1=<a,b<=n）)表示a和b有亲戚关系

再下面k行，每行两个数p，q (1=<p,q<=n）)表示求p和q是否有亲戚关系

输出格式（文件名relation.out）：

k行，每行一个整数1或者0，对于输入中的最后k行，如果每一行的两个人存在亲戚关系则输出1,否则输出0.

样例数据
DFS搜索？

输入样例：

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

输出样例

```
1
0
1
1
```

样例说明：

1和9是亲戚。9和8是亲戚，所以1和8是亲戚，所以第一行输出1;

1和6不是亲戚，输出0

2和3是亲戚，输出1

2和9是亲戚，输出1

很多问题都可以转换为图的连通问题

有亲戚关系（有边）的人（点）互相之间通过亲戚关系（边）连接，组成了图。人就是点，关系就是边。

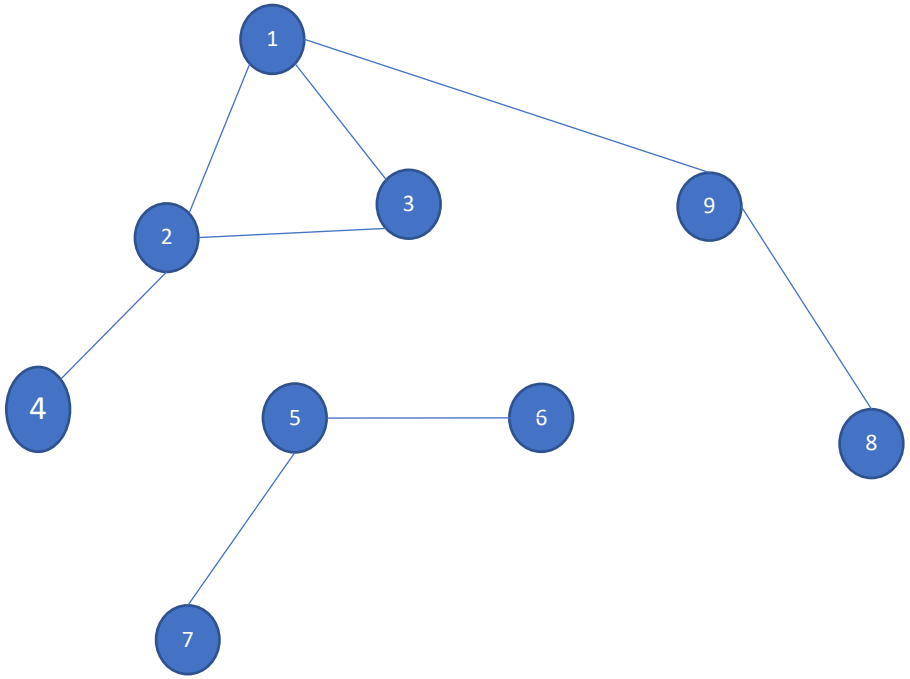
如果a和b是亲戚，b和c亲戚，那么a和c是亲戚===》如果a和b两个点有边，b和c有边，那么a和c连接。

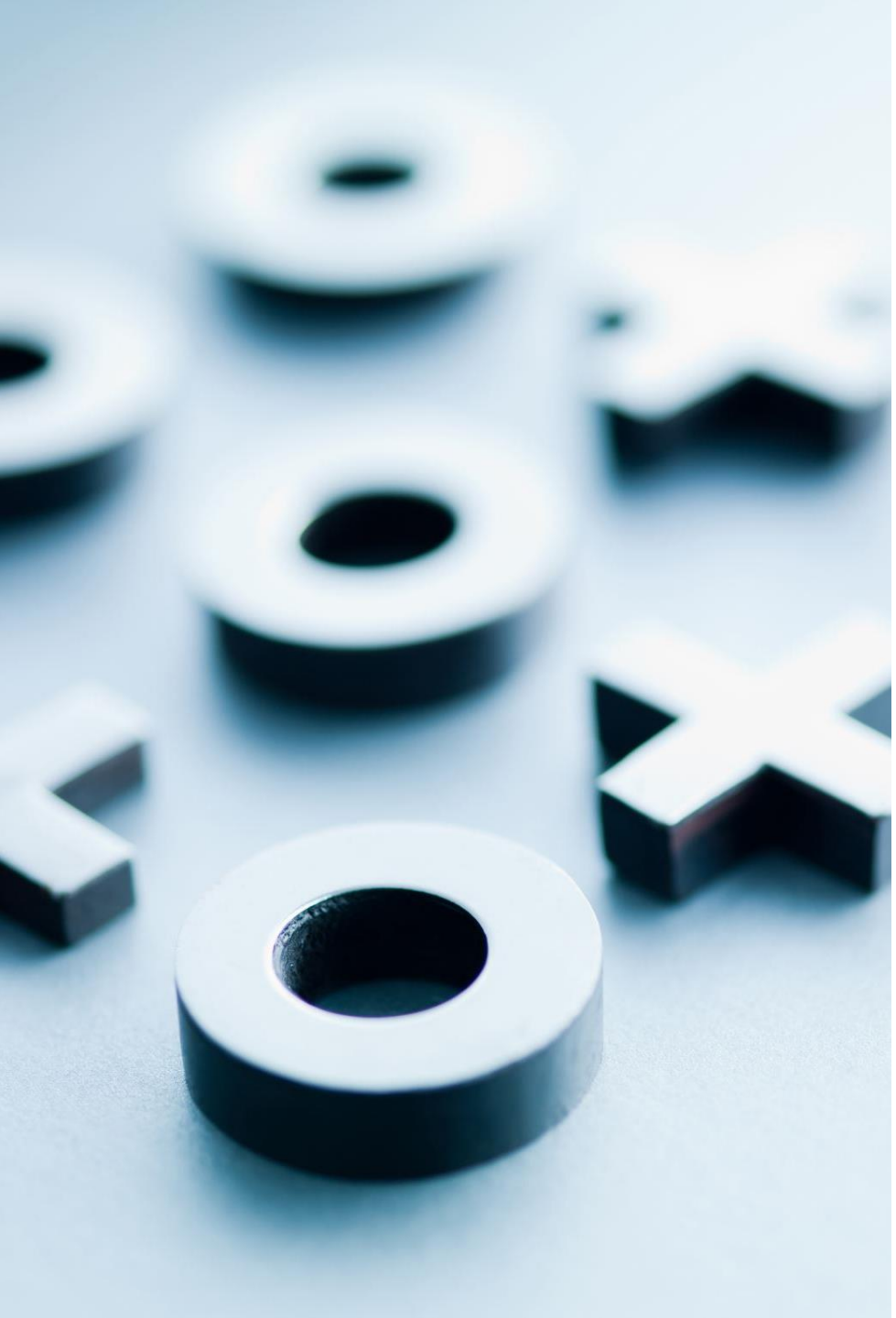
DFS搜索，找a和b是否有亲戚关系就是看a能不能通过访问相邻点到达b，或者说a和b是否连接（上节课作业1的类似方法）。

转为图

输入样例：

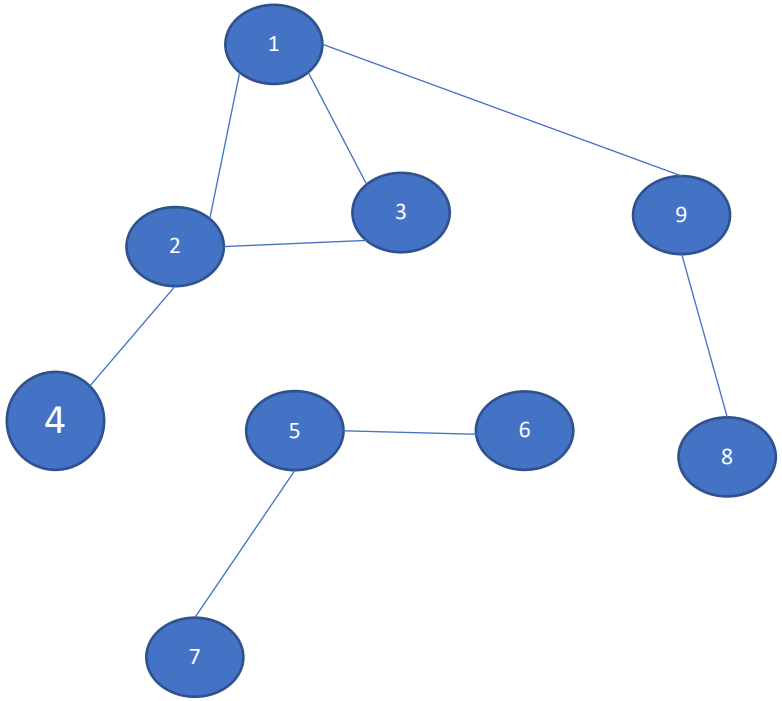
9 8 4
2 4
5 7
1 3
8 9
1 2
5 6
2 3
1 9
1 8
1 6
2 3
2 9





30分？

- **由于n，m，k可能都是10^5，如果我们对于输入的每一个k都用DFS取搜索两人（两个节点）是否连接，那么最后时间复杂度就会超过10的10次方，爆掉。**
- 但是，如果想不到别的方法，就搜索！30分很重要！

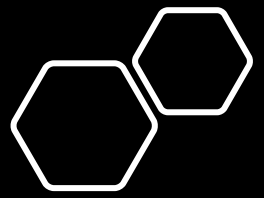


分析

- 是一个图：每个人是一个节点，如果两个人存在亲戚关系，他们之间就有一个连接，最后找两人是否存在亲戚关系实际上是在问这两人的节点是否连通（直接连接或者通过其他节点连接）
- 也可以理解为集合（单元），所有互相有亲戚关系的人算在一个集合（单元），最后找两人是否存在亲戚关系实际上是在问这两人是否在一个集合（单元）内。
- **所以最好的办法就是连通，依次搜索找到所有人的连通子图或者是集合（并查集），后面判断任意两人就是查找他们是否在一个集合或者一个连通图内。**
- 求连通：用floodfill（dfs或者bfs）或者并查集

很多问题都可以转换为图的连通问题

- 这个题目：有亲戚关系（有边）的人（点）互相之间通过亲戚关系（边）连接，就构成了一个连通子图
- 所以，题目其实是在求某些人是否在同一个连通子图里面（或者说同一个单元）
- 尤其是找多个点是否连接，如果用dfs，那么求多少个点的关系就要多少遍DFS，但是用连通子图的方式，其实一遍就把所有的节点都归到单元了，剩下找点的关系其实就是一个数组的访问，O(1)的时间函数。



例：
亲戚关系

给出n个人，编号1到n，他们之间有些人有亲戚关系，给出m对存在亲戚关系的人，求任意给出的两个人是否具有亲戚关系。这里：x和y是亲戚，y和z是亲戚，那么x和z也是亲戚。

输入格式（文件名relation.in）：

第一行：三个整数n,m,k（n<=10^5,m<=10^5,k<=10^5），分别表示有n个人，m个亲戚关系，求k对人是否有亲戚关系。

以下m行：每行两个数a，b (1=<a,b<=n）)表示a和b有亲戚关系

再下面k行，每行两个数p，q (1=<p,q<=n）)表示求p和q是否有亲戚关系

输出格式（文件名relation.out）：

k行，每行一个整数1或者0，对于输入中的最后k行，如果每一行的两个人存在亲戚关系则输出1,否则输出0.

DFS

```
vector<vector<int> > aplist(100001);
int component[100001]={0};//标记每一个节点的单元编号，同时用来避免重复访问
int componentnumber=0;
void dfsfill(int cur {
        component[cur]=componentnumber;
        for (int i = 0;i < aplist[cur].size(); i++) //寻找没有访问过的sp的相邻节点
                if ( component[aplist[cur][i]]==0)   dfsfill(aplist[cur][i]);
}
int main (){
        cin>>n>>m>>k;
        for (int i = 1; i <= m;i++) //读取并插入aplist
        {
                cin>>a>>b;  //读入两个节点，比如1，3
                aplist[a].push_back(b);
                aplist[b].push_back(a);
        }
        for (int i=1;i<=n;i++) //循环处理: componentnumber++；找到任意一个component[i]为0的i点
                if (component[i]==0) {componentnumber++;      dfsfill(i);  }
        for (int i=1;i<=k;i++) {
                cin>>a>>b;
                if (component[a]==component[b]) cout<<1<<endl; //如果a和b的单元编号一样，说明在一个联通，也就是说有亲戚关系
                else cout<<0<<endl;
        }
        return 0;
}
```

并查集

```
int parent[5001];
int n,m,k;
int findroot(int x)
//标准并查集的“查”
{
        if (parent[x]==x)
                return x;
        return findroot(parent[x]);
}
```

并查集模板程序

就是并和查：

读入每一条边的两个节点：查找两个节点当前的集合，

合并这两个集合。

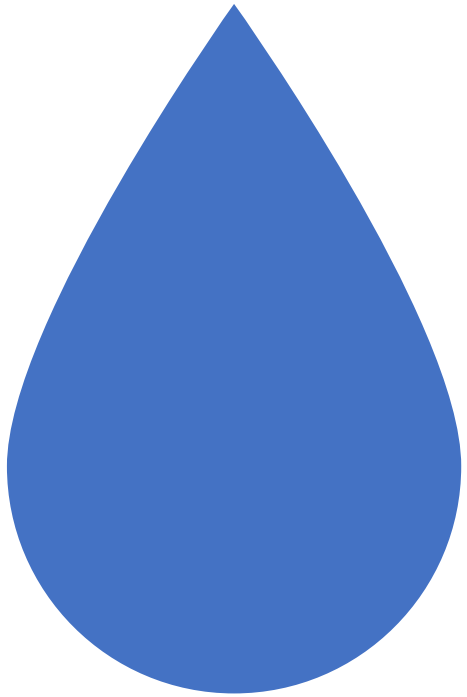
```
int main(){
    cin >> n>>m>>k;
   for(int i=1;i<=n;i++) //初始化集合，数组值等于小标的点为根节点。
        parent[i]=i;
    int a,b,x,y;
    for (int i=1;i<=m;i++)
    {
        cin>>a>>b;
        x=findroot(a),y=findroot(b);//找到ab所在集合的根元素
        if (x!=y) parent[x]=y;  //合并两个根所在集合
    }
    for(int i=1;i<=n;i++) parent[i]=findroot(i);//压缩
    for (int i=1;i<=k;i++)
    {
        cin>>a>>b;
        //如果a和b在一个集合，也就是他们的root一样，就是在
        //一个联通内，就是有亲戚关系
        if (findroot(a)==findroot(b)) cout<<1<<endl;
        else cout<<0<<endl;
    }
    return 0;
}
```

一些方法

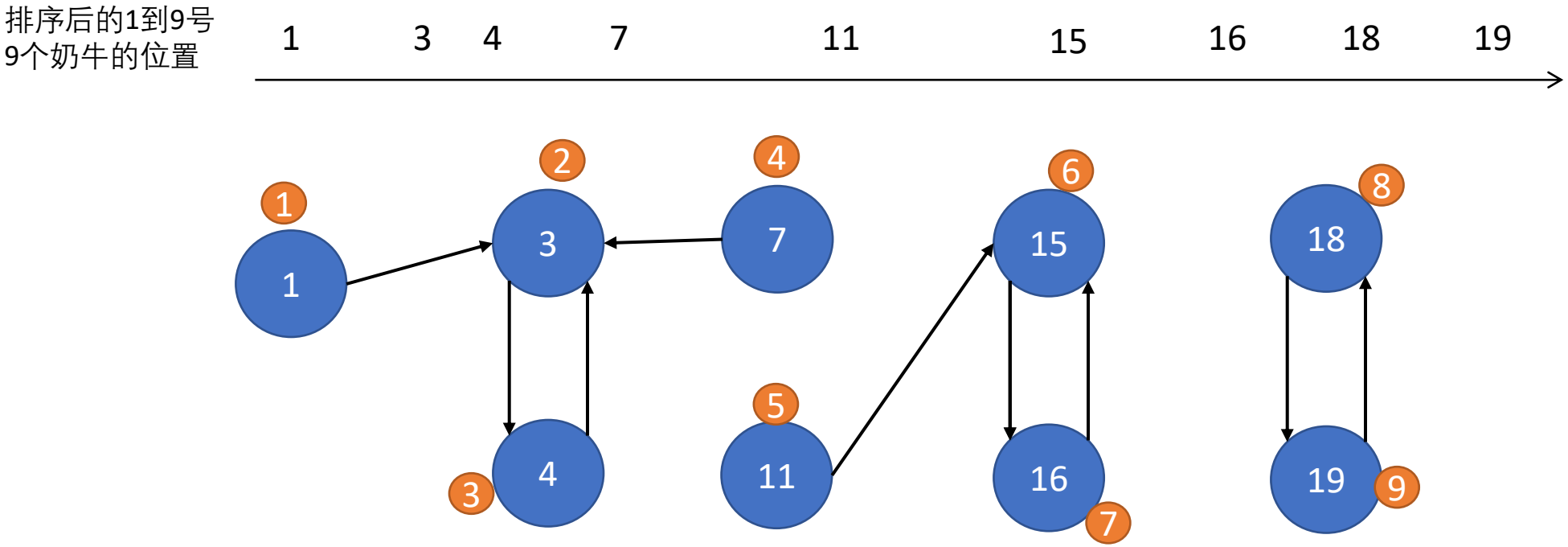
- 无论是无环还是有环，往往转换为连通子图的处理问题。为什么？因为处理连通子图可以一次DFS，或者一次并查集操作就把所有的连通子图标注出来，就可以知道几个连通子图，就可以知道任意两个节点是否在同一个连通子图（单元内）
- 有向图有个非常特殊的性质，就是入度为0的点和出度为0的点，很多问题都可以从这些点出发找规律。
- 有向图也可以找连通子图，但是往往从入度为0的点出发开始找（为什么？方向！）。

USACO 2018 February Contest, Bronze　Hoofball

- 为了准备即将到来的蹄球锦标赛，Farmer John正在训练他的N头奶牛（方便起见，编号为1…N其中$1 \le N \le 100$）进行传球。这些奶牛在牛棚一侧沿直线排列，第i号奶牛位于距离牛棚X_i的地方($1 \le X_i \le 1000$)。每头奶牛都在不同的位置上。
- 在训练开始的时候，Farmer John会将若干个球传给不同的奶牛。当第i号奶牛接到球时，无论是从Farmer John或是从另一头奶牛传来的，她会将球传给最近的奶牛（如果有多头奶牛与她距离相同，她会传给其中距左边最远的那头奶牛也就是右面那个if multiple cows are the same distance from her, she will pass the ball to the cow farthest to the left among these）。为了使所有奶牛都有机会练习到传球，Farmer John想要确保每头奶牛都持球至少一次。帮助他求出为了达到这一目的他开始时至少要传出的球的数量。假设他在开始的时候能将球传给最适当的一组奶牛。



有向图：方法一：找规律



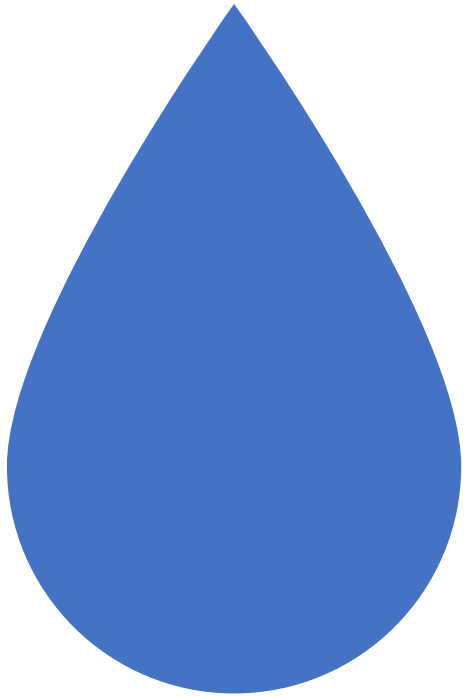
其实我们把奶牛按照位置排序以后发现，由于每一个奶牛位置不同，一个奶牛i传给的一定是左面（i-1）或者右面（i+1）最近的一个，如果这两个相同，就传给i-1。上图箭头表示传球。圈内标注的是位置，但是实际上程序中根据位置计算后，操作的是奶牛编号1到9（由于最终和奶牛开始的编号无关，所以这个1到9是排序后按照顺序的新的编号）。

第一种做法：如果找到每一个节点的passto（传给谁），比如passto[1]为2表示1号传球给2号。位置遍历所有的passto，就可以得到所有点的入度（几个人传球给他）。然后对于入度为0的点，计数++；对于入度是1的点，如果他的passto入度也是1，而且互相pass（passto的那个点入度也是1），也计数++。

```
fin >> n;
for (int i=1; i<=n; i++)    fin >> a[i];
sort(a+1,a+n+1);
a[0]=-INTMAX;  a[n+1]=INTMAX;//方便边界处理
for (int i=1; i<=n; i++) { //计算passto 和入度
                if (a[i]-a[i-1]<=a[i+1]-a[i]) passto[i]=i-1;
                else  passto[i]=i+1;
                incoming[passto[i]]++;
}
for (int i=1;i<=n;i++)
{
                if (incoming[i]==0) {
                                r++;//对于入度为0的点，计数++;
                }
                //对于入度是1的点，如果他的passto入度也是1，而且互相pass，也计数++
                if ( incoming[i]==1 && incoming[passto[i]]==1 && i==passto[passto[i]] )
                {
                                r++;
                                incoming[i]=INTMAX;//设置不为1，防止重复计算
                                incoming[passto[i]]=INTMAX;
                }
}
fout << r << "\n";
```

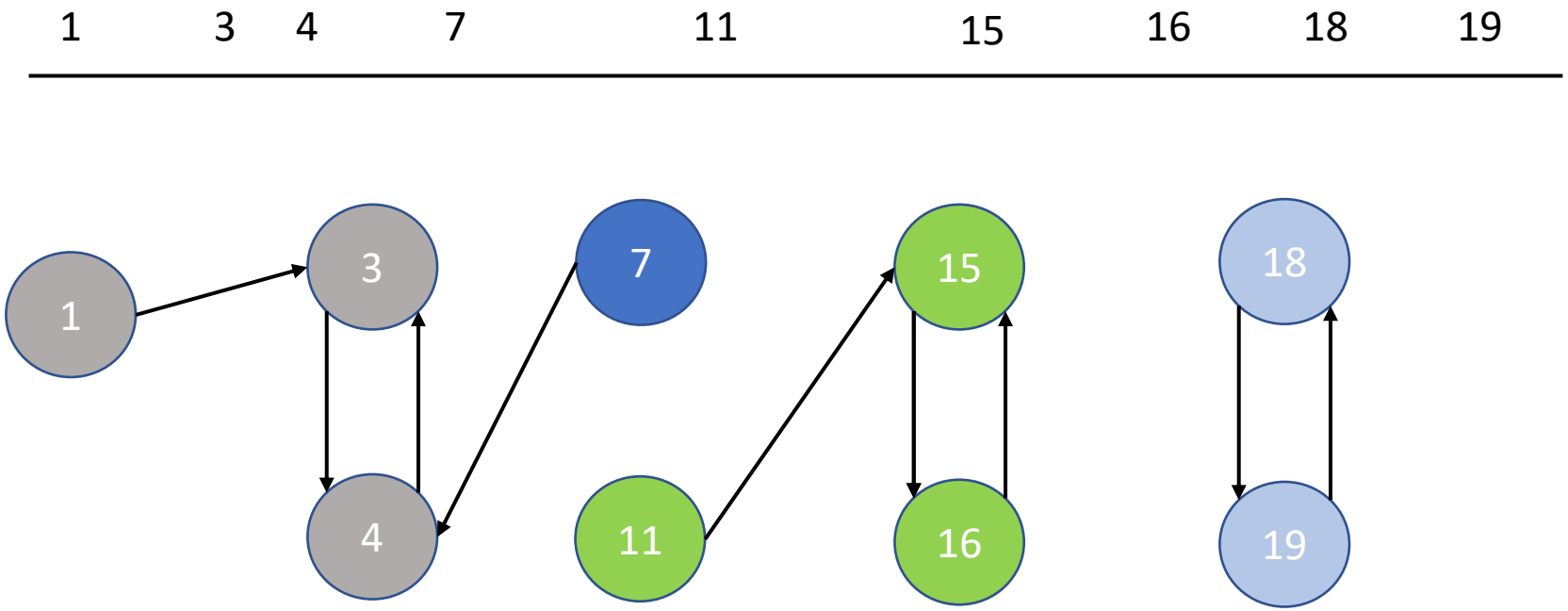
规律没有找到怎么办?

前面讲的方法比较特殊，因为环只会存在2个点之间。更通用的方法是Flood fill或者并查集找联通。有几个联通就是+几，但是不同的是，需要先统计入度，然后选择入度为0的点开始寻找联通图。



有向图：方法二：找连通

排序后的1到9号
9个奶牛的位置



入度为0的点一定需要第一次拿到球，因为没有人给他传球。所以对于最终入度为0的点，按照他的联通节点（就是能传球传到的）进行标注；注意1号传给2号不代表2号可以传给1号，有向的。

然后对于所有其他没有联通的节点（有环的）再分别寻找联通，最后有多少联通子图，就是答案了。图中是4个连通子图。

如何寻找入度为0的点？可以用刚才的办法，也可以直接统计：遍历所有节点，对于每一个节点判断他的左右节点哪个近，然后给近的节点入度+1.

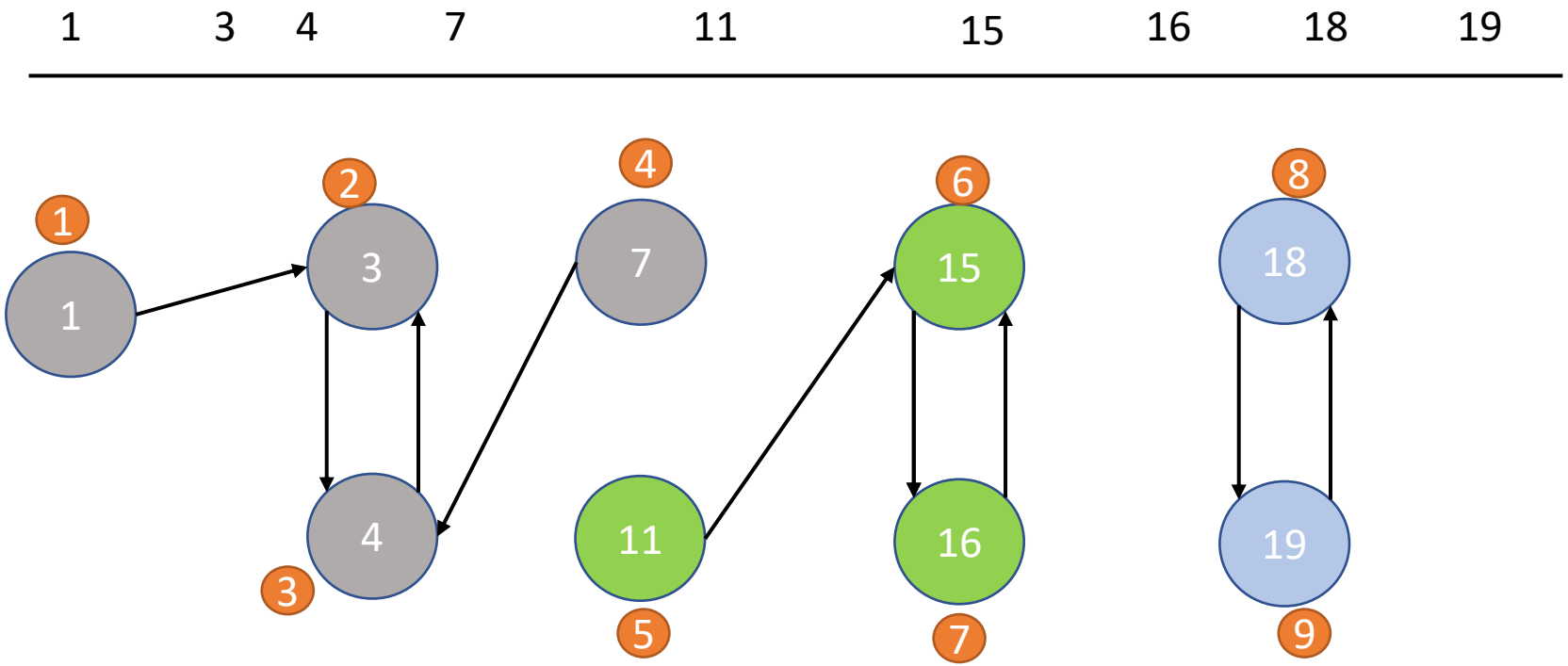
计算passto[]和入度incoming[]与前面一个算法的程序一样

```
//对于入度为0的点，寻找component
 for (int i=1;i<=n;i++)
 {
                if (incoming[i]==0&&component[i]==0) {
                                componentnum++;
                                findcomp(i,componentnum);
                }
  }
 //对于其他点寻找component
 for (int i=1;i<=n;i++)
 {
                if (component[i]==0) {
                                componentnum++;
                                findcomp(i,componentnum);
                }
  }
  fout << componentnum << "\n";
```

```
void findcomp(int i,int num)
{
                component[i]=num;
//注意，这里i的相邻点就是passto[i],也就是i传球给谁
                if (component[passto[i]]==0)
                                findcomp(passto[i],num);
                return;
}
```

特殊的并查集?

排序后的1到9号
9个奶牛的位置



如果按照并查集做法，其实节点1，2，3，4都是一个集合的，应为他们连接，但是这是有向图，需要的答案不光是连通图的个数，还包括了parent[i]!=i 但是incoming[i]==0的节点，比如节点1，2，3，4他们最后是一个根，1个集合，但是4号点虽然不是根，也需要一个球。图中3个集合，4号的节点特殊处理。 parent[i]!=i&& incoming[i]==0的点计数也要++

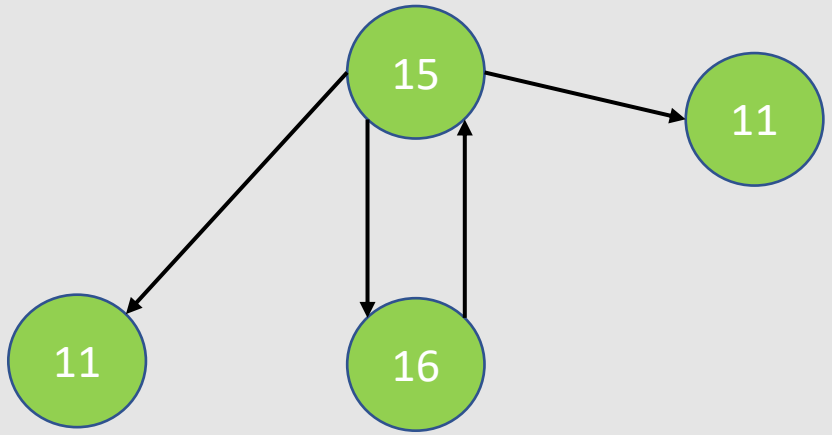
并查集

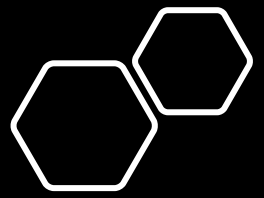
```
for(int i=1;i<=n;i++)      parent[i]=i;

for (int i=1;i<=n;i++) {

       int x=findroot(i),y=findroot(passto[i]);

      //找到ab所在集合的根元素

       if (x!=y) parent[y]=x;  //合并两个根所在集合

}

int cnt=0;

for(int i=1;i<=n;++i)      //统计集合个数，即为连通分量个数

{

       if(parent[i]==i|| parent[i]!=i&& incoming[i]==0)

                     ++cnt;

}

fout<<cnt<<endl;
```

```
int n, incoming[102];
int passto[102] ;
int a[102];
int parent[102];

int findroot(int x)
{
                     if (parent[x]==x)
                                          return x;
                     return findroot(parent[x]);
}
```

注意

- 这里能够有向图连通是因为这个有向图的特殊性。如果是这样的图就不能找连通了。找不到入度为0的出发点。





总结

- 类似传递问题，其实根据题目的不同，解决方法各不相同，需要具体分析
- 无论如何DFS是一个非常常用的方法，至少可以解决一般问题（效率问题）。
- 并查集，图的连通，DP都是常用手段
- 有向图也可以用连通问题解决，但是需要特殊考虑或者处理入度为0的点。

下节课预习

- 记得队列结构么？queue？
- 2019 NOIP 公交换乘
- 不需要做，最好可以理解题目要求

作业1：最大连接牛棚maxstall.cpp

给出n个牛棚，编号1到n。其中有的牛棚之间有双向路，有些没有。如果牛棚A和牛棚B之间有路，牛棚B和牛棚C之间也有路，显然牛棚A可以到达牛棚C。

给定一个 n * n 的矩阵，表示n个牛棚之间的道路信息。矩阵的i行j列的值表示牛棚i和j之间有没有直接的道路，如果是1表示有路，0表示没有路。

现在农场主希望知道他从任意一个牛棚出发，最多可以到达多少不同的牛棚。

输入格式：

第一行为一个整数n（n<=1000）,表示有n个牛棚。

后面n行，每一行n个0或者1的数字，i行j列的值表示牛棚i和j之间有没有直接的道路，如果是1表示有路，0表示没有路。

输出格式：

一个整数，表示从任意一个牛棚出发，最多可以到达多少不同的牛棚。

作业1说明和要求

输入示例:

4

0101

1001

0000

1100

输出示例:

3

示例说明，4个牛棚，第一行的0101表示1号到2号，4号有路。最大的可以经过的牛棚数目应该是1，2，4，无论怎么走，到不了3号。

提示：其实就是在求最大连通子图节点的数目。这里的图，没有按照前面的方式给出每一条边，而是直接给了你邻结矩阵。

要求：用flood fill寻找连通图的方法或者用并查集的方法都可以，但是请思考你喜欢哪一个方法，为什么?

作业2：用DFS,BFS,或者并查集(选做)的方法完成road.cpp

- 给定若干个城市，编号1到n，（n<=10^3），他们之间m条双向道路(m <=10^5)。
- 输入(road.in)：第1行2个整数n,m空格隔开，后面第2行倒m+1行为m条边，每行两个整数a b表示节点a和节点b有双向道路。
- 输出(road.out)：一个整数，表示从任意一个城市出发，可以通过给定道路连接到达的最多的城市数目。
- 示例输入：

9 8

1 2

3 1

2 4

5 2

3 6

4 7

8 9

5 4

- 示例输出:

7

提示：其实就是在问最大联通子图内的节点个数。我们在找联通子图的时侯可以把当前联通找到的节点数目加起来，打擂就好了。

挑战作业3

- USACO 2019 US Open Contest, BronzeMilk Factory

- 提示：先找规律看看？其实这个题目不用搜索啥的，非常简单。关键是你能不能找到规律：什么样的点是满足条件的点？

由易到难，思维体系训练
实战结合，创新协作培养
兴趣导向，未来职业引领

https://www.35tang.com

https://www.三五堂.com

扫码关注公众号

添加辅导老师