*详细教程：https://zhuanlan.zhihu.com/p/93647900

一、核心思路：

并查集被很多OIer认为是最简洁而优雅的数据结构之一，主要用于解决一些元素分组的问题。它管理一系列不相交的集合，并支持两种操作：

- **合并（Union）：把两个不相交的集合合并为一个集合。**
- **查询（Find）：查询两个元素是否在同一个集合中。**

二、应用场景

当然，这样的定义未免太过学术化，看完后恐怕不太能理解它具体有什么用。所以我们先来看看并查集最直接的一个应用场景：**亲戚问题。**

（洛谷P1551）亲戚

题目背景：

若某个家族人员过于庞大，要判断两个是否是亲戚，确实还很不容易，现在给出某个亲戚关系图，求任意给出的两个人是否具有亲戚关系。

题目描述：

规定：x和y是亲戚，y和z是亲戚，那么x和z也是亲戚。如果x,y是亲戚，那么x的亲戚都是y的亲戚，y的亲戚也都是x的亲戚。

输入格式：

第一行：三个整数n,m,p，（n<=5000,m<=5000,p<=5000），分别表示有n个人，m个亲戚关系，询问p对亲戚关系。

以下m行：每行两个数Mi，Mj，1<=Mi，Mj<=N，表示Mi和Mj具有亲戚关系。

接下来p行：每行两个数Pi，Pj，询问Pi和Pj是否具有亲戚关系。

输出格式

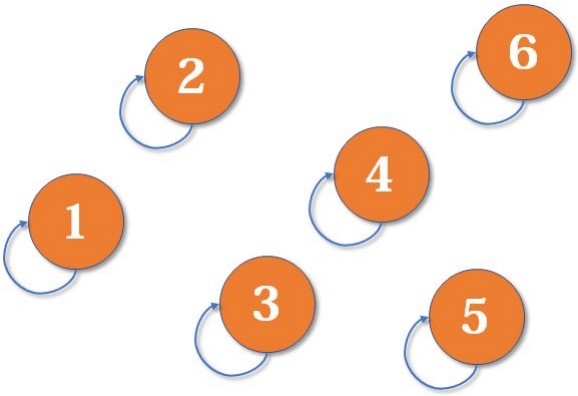
P行，每行一个‘Yes’或’No’。表示第i个询问的答案为“具有”或“不具有”亲戚关系。

这其实是一个很有现实意义的问题。我们可以建立模型，把所有人划分到若干个不相交的集合中，每个集合里的人彼此是亲戚。为了判断两个人是否为亲戚，只需看它们是否属于同一个集合即可。因此，这里就可以考虑用并查集进行维护了。

三、并查集的引入

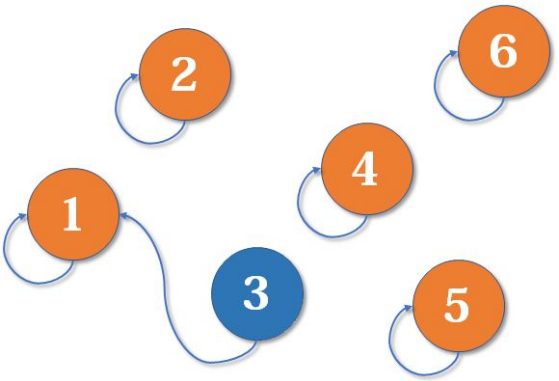
并查集的重要思想在于，用集合中的一个元素代表集合。

我曾看过一个有趣的比喻，把集合比喻成帮派，而代表元素则是帮主。接下来我们利用这个比喻，看看并查集是如何运作的。

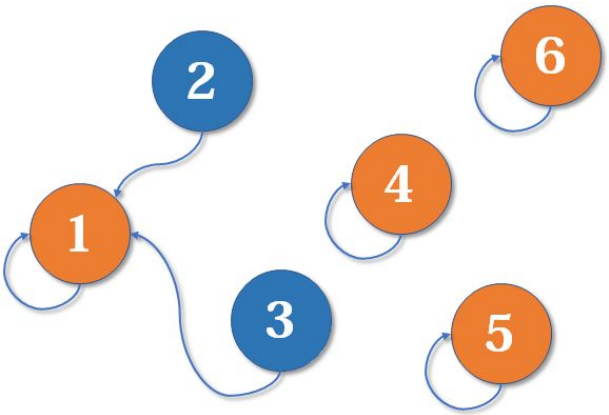


最开始，所有大侠各自为战。他们各自的帮主自然就是自己。*（对于只有一个元素的集合，代表元素自然是唯一的那个元素）*

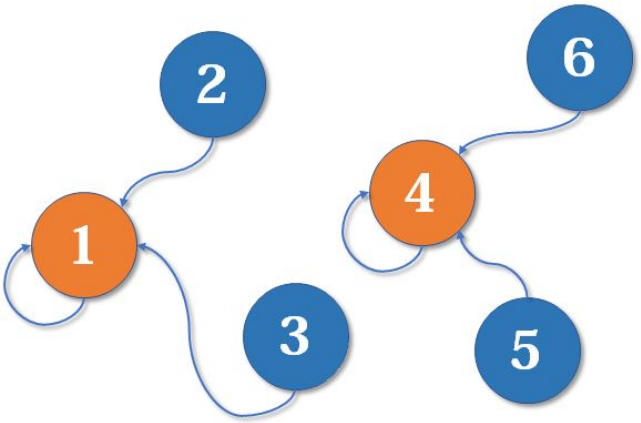
现在1号和3号比武，假设1号赢了（这里具体谁赢暂时不重要），那么3号就认1号作帮主*（合并1号和3号所在的集合，1号为代表元素）*。



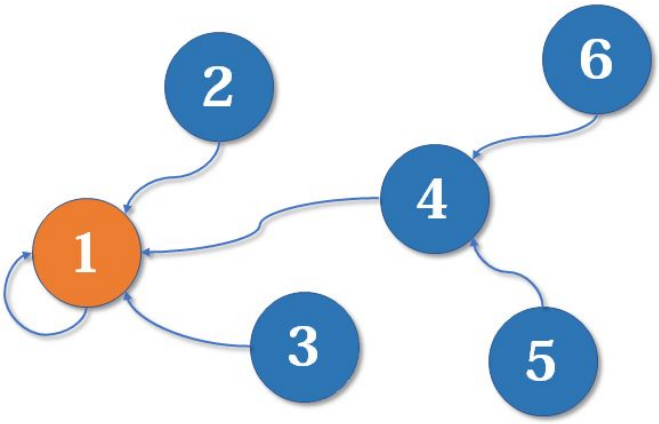
现在2号想和3号比武*（合并3号和2号所在的集合）*，但3号表示，别跟我打，让我帮主来收拾你*（合并代表元素）*。不妨设这次又是1号赢了，那么2号也认1号做帮主。



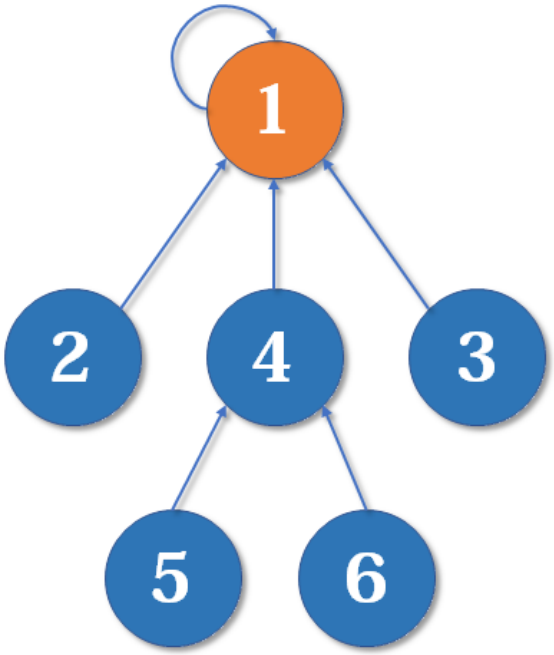
现在我们假设4、5、6号也进行了一番帮派合并，江湖局势变成下面这样：



现在假设2号想与6号比，跟刚刚说的一样，喊帮主1号和4号出来打一架（帮主真辛苦啊）。1号胜利后，4号认1号为帮主，当然他的手下也都是跟着投降了。



好了，比喻结束了。如果你有一点图论基础，相信你已经觉察到，这是一个树状的结构，要寻找集合的代表元素，只需要一层一层往上访问父节点（图中箭头所指的圆），直达树的根节点（图中橙色的圆）即可。根节点的父节点是它自己。我们可以直接把它画成一棵树：



（好像有点像个火柴人？）

四、代码

用这种方法，我们可以写出最简单版本的并查集代码。

初始化

```
int fa[MAXN];
inline void init(int n)
{
    for (int i = 1; i <= n; ++i)
        fa[i] = i;
}
```

假如有编号为1，2，3，...，n的n个元素，我们用一个数组fa[]来存储每个元素的父节点（因为每个元素有且只有一个父节点，所以这是可行的）。一开始，我们先将它们的父节点设为自己。

查询

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

我们用递归的写法实现对代表元素的查询：一层一层访问父节点，直至根节点（根节点的标志就是父节点是本身）。要判断两个元素是否属于同一个集合，只需要看它们的根节点是否相同即可。

合并

```
inline void merge(int i, int j)
{
    fa[find(i)] = find(j);
}
```

合并操作也是很简单的，先找到两个集合的代表元素，然后将前者的父节点设为后者即可。当然也可以将后者的父节点设为前者，这里暂时不重要。本文末尾会给出一个更合理的比较方法。