基础数据结构

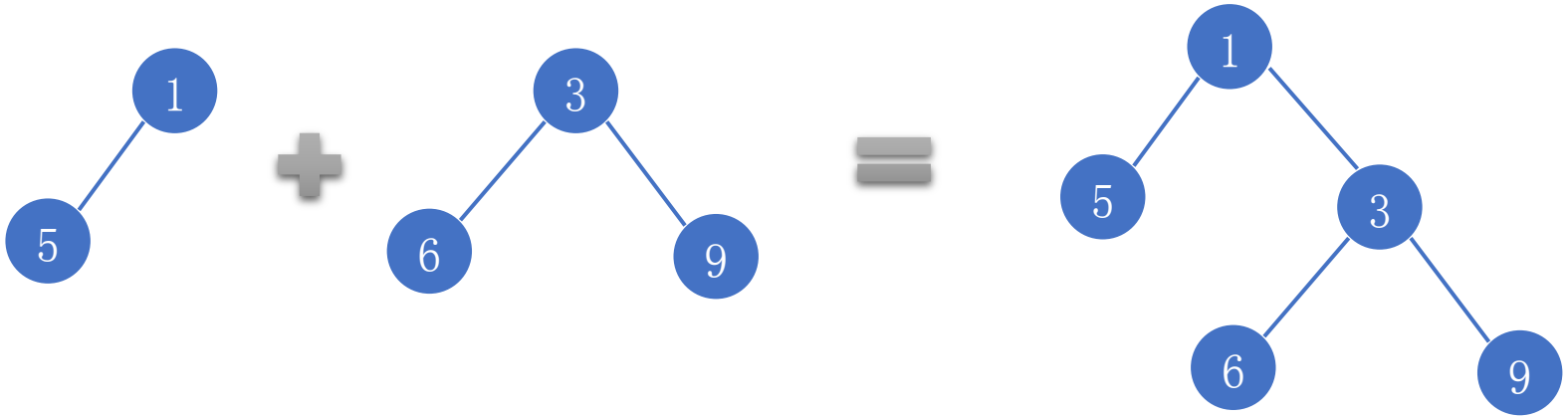
湖南师大附中 许力

森林（forest）

- 森林就是n棵互不相交的树

- 森林与二叉树的转化

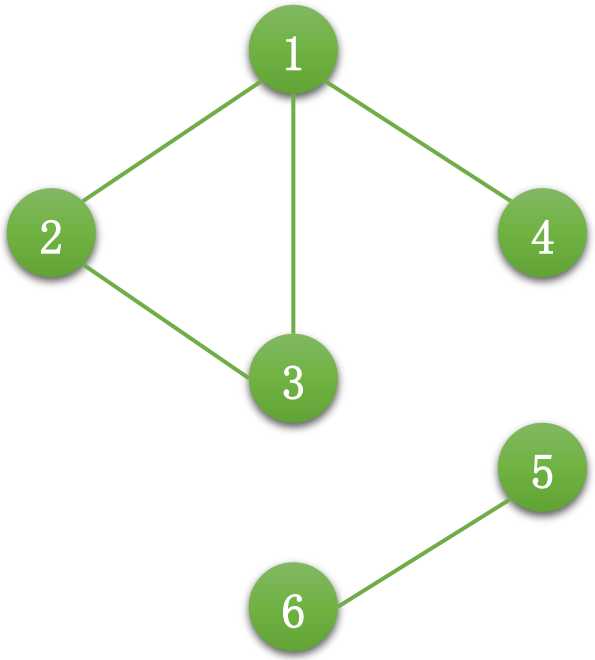
设$F=\{T_1, T_2, T_3, \cdots\cdots T_n\}$



图的连通性问题

- 给定一张有n个顶点，m条边的图

1. 询问图中指定的一对顶点间是否有路径可以到达
2. 图中如果任意多对顶点都互相有路径可以到达，这多对顶点就构成连通块，询问图中的连通块有多少个
3. 至少需要添加几条边才可以使整张图连通

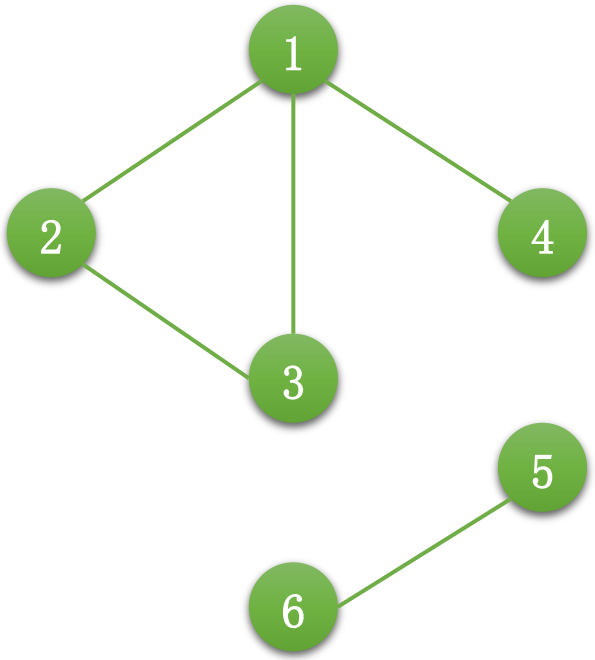


并查集（disjoint set）

- 并查集是描述不相交集合的数据结构，并支持对这些不相交集合进行合并、查找操作。

- 因此并查集主要就是两个操作：

1. 合并操作
2. 查找操作



并查集（disjoint set）

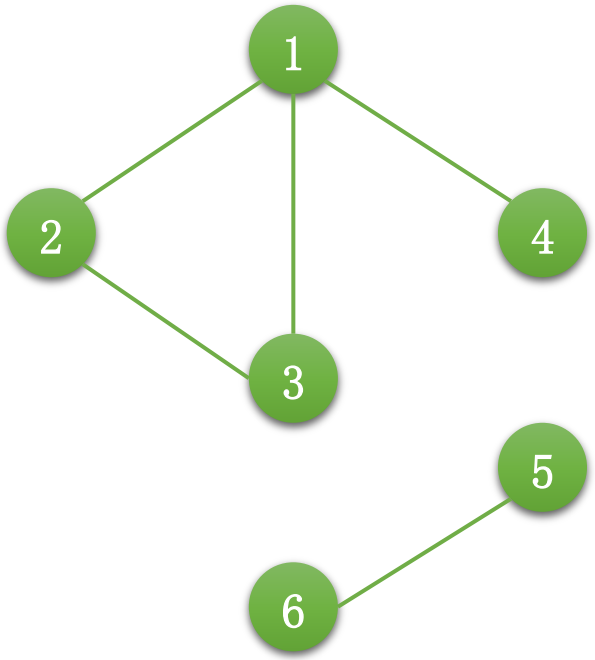
- 我们首先把顶点用一维数组存下来

数组下标记录节点编号

数组元素记录该节点的祖先节点

刚开始，所有节点的祖先节点就是它自己

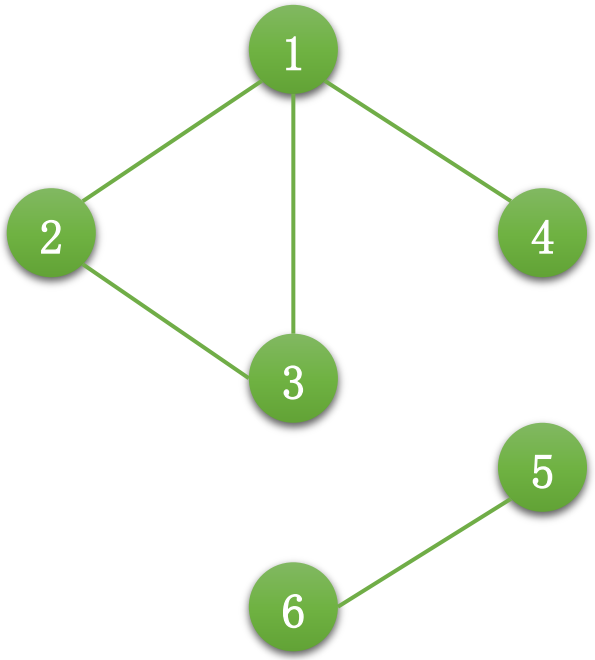
id	1	2	3	4	5	6
data	1	2	3	4	5	6



并查集（disjoint set）

- 我们首先把顶点用一维数组存下来
- 读入图的信息后，发现1-2有边相连，于是修改2号的祖父节点为1号的祖父节点，也就是1

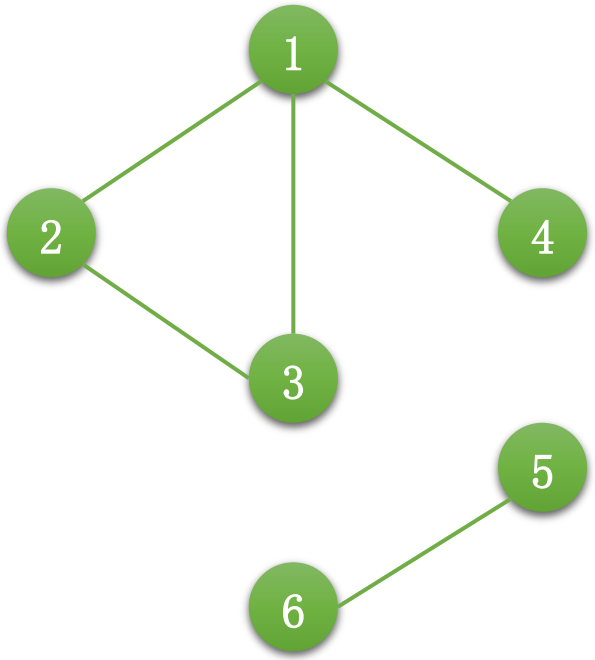
id	1	2	3	4	5	6
data	1	1	3	4	5	6



并查集（disjoint set）

- 我们首先把顶点用一维数组存下来
- 读入图的信息后，发现1-2有边相连，于是修改2号的祖父节点为1号的祖父节点，也就是1
- 此后3、4、6号依次类推
- 注意这里当发现2-3有边相连时的处理

id	1	2	3	4	5	6
data	1	1	1	1	5	5



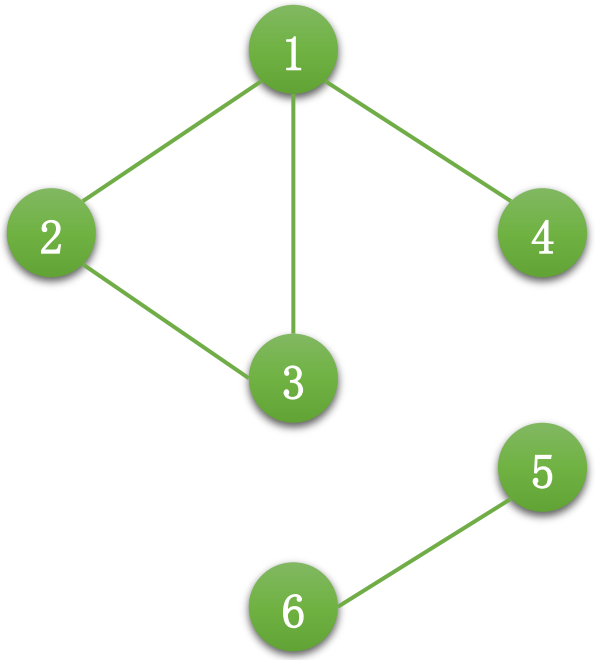
并查集（disjoint set）

- 经过这样的处理后，我们发现：

```
if (fa[i] == i) i就是该连通块的祖先节点
祖先节点的数量,就是连通块的数量

if (fa[i] == fa[j]) i和j同处于一个连通块
要使得整张图连通,只需要把连通块的祖先节点连接起来
```

id	1	2	3	4	5	6
data	1	1	1	1	5	5



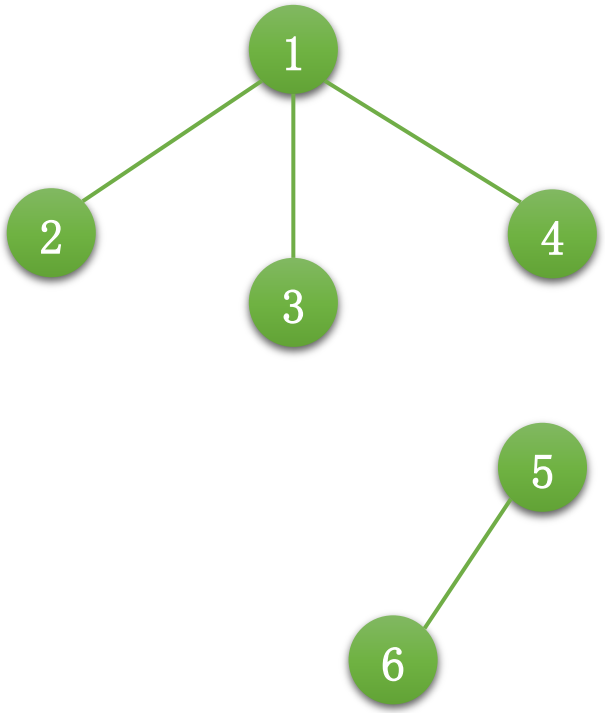
并查集（disjoint set）

- 我们可以把树改成这个样子：

```
if (fa[i] == i) i是祖先节点

if (fa[i] == j && j != i) j是i的父节点
```

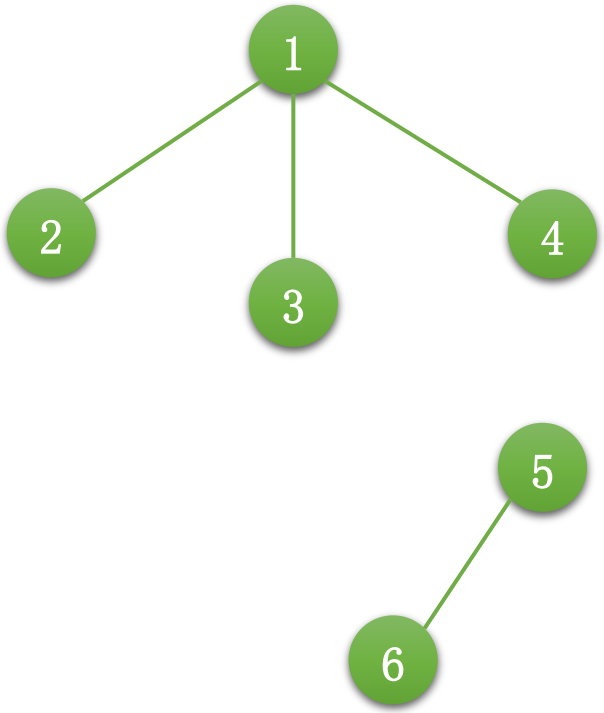
id	1	2	3	4	5	6
data	1	1	1	1	5	5



并查集（disjoint set）

- 并查集的本质，是通过一维数组模拟，来维护一个森林
- 刚开始的时候，森林中只有孤立的点，之后通过一些条件逐步合成为若干棵树

id	1	2	3	4	5	6
data	1	1	1	1	5	5

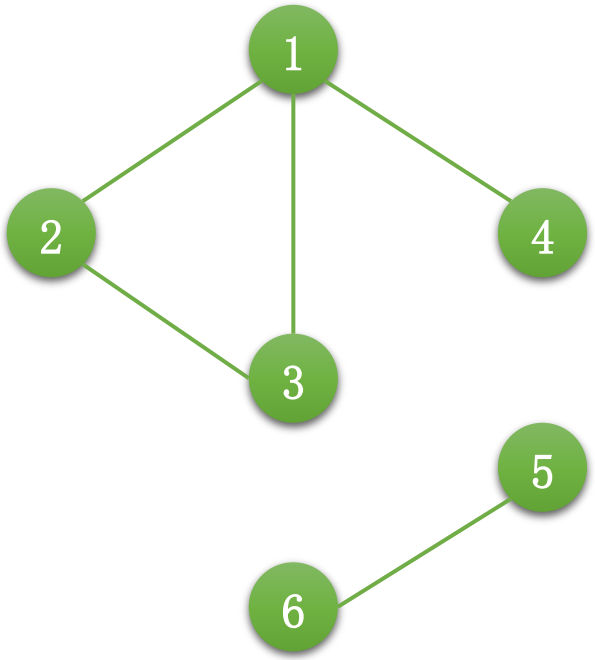


并查集的代码实现

1. 并查集的查找操作（find）

① 查询某元素属于哪个集合

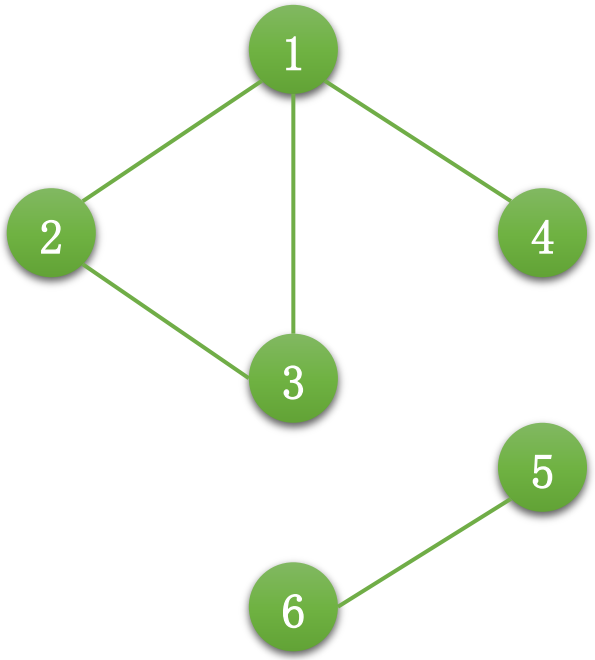
② 查询某两个元素是否同属一个集合



并查集的代码实现

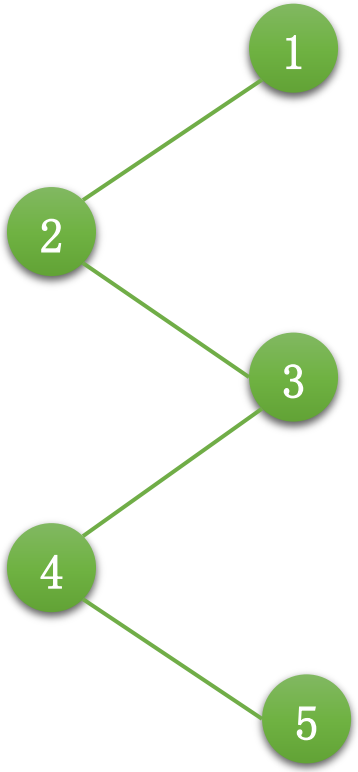
1. 并查集的查找操作（find）

```
int find(int x)
{
    while (fa[x] != NULL)
        x = fa[x];
    //往上追溯父节点直至根节点
    return x;
 }
```



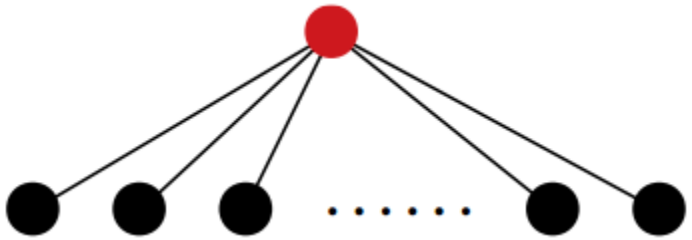
路径压缩

- 但实际上我们一般不会写成上面的样子
- 原因很简单：当树的层次比较深的时候，回溯到根节点比较费时
- 极端情况下，退化成这样：



路径压缩

- 注意到并查集是没有删除节点操作的，因此同一集合中的节点数只增不减
- 如果一棵（并查集中的）树长下面这样：

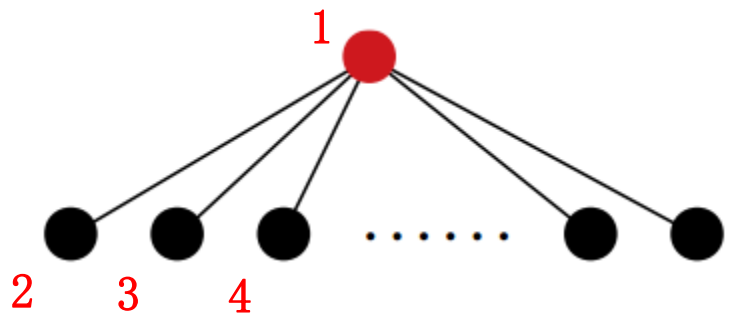


路径压缩

- 如果一棵（并查集中的）树长下面这样：
- 现实中，你可能会觉得这棵树一定臃肿无比，但实际上：

没有任何负担！

id	1	2	3	4	5	6	……
data	1	1	1	1	1	1	1

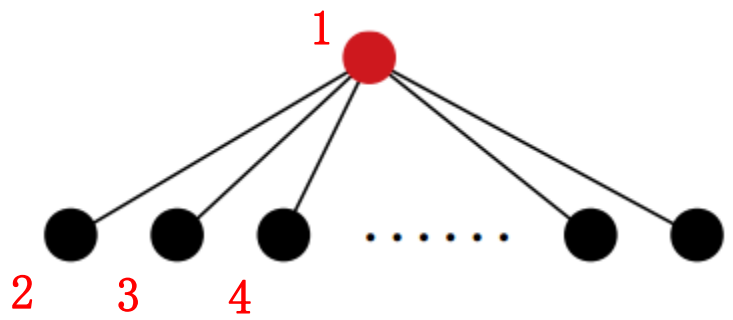


路径压缩

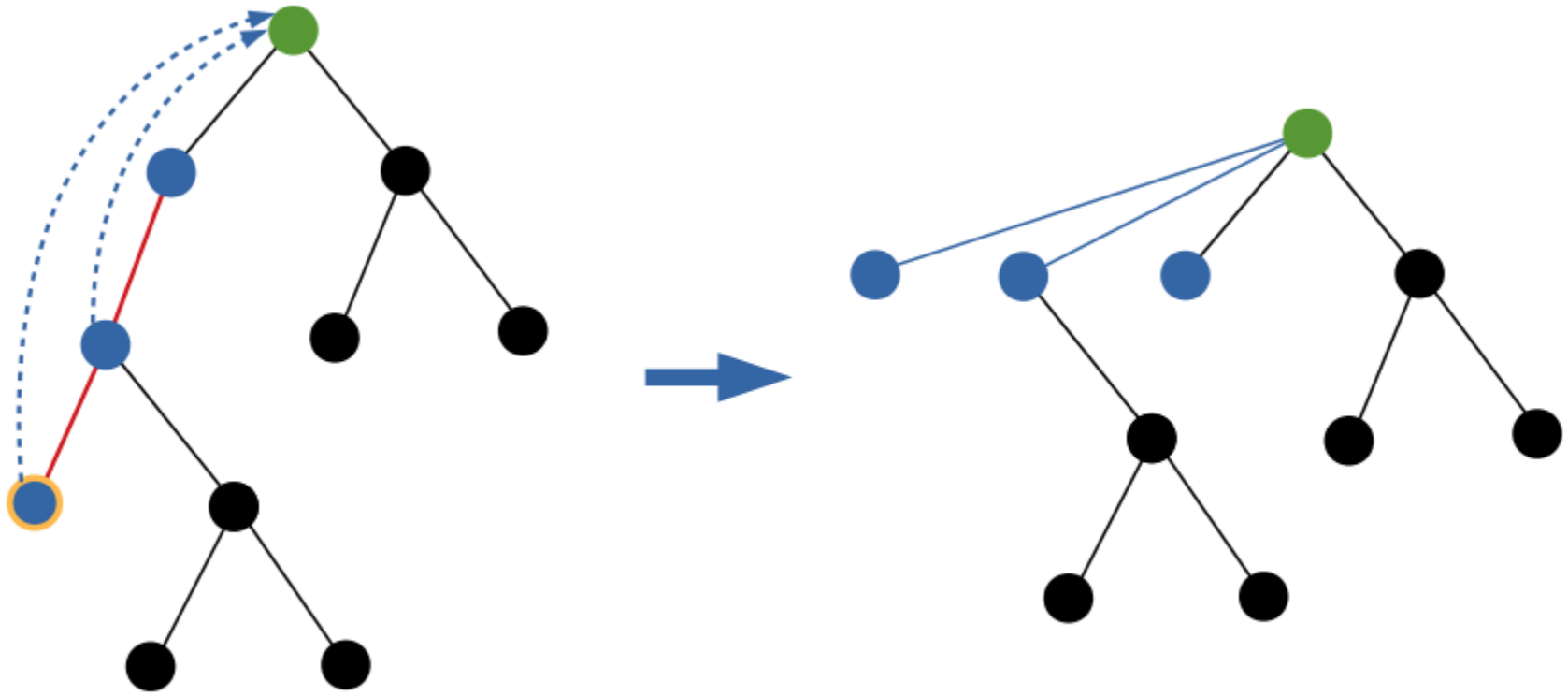
- 换句话说：在每次find的时候，把访问到的节点的父节点全部换为根节点，以便下次查询该节点时可以一步到位

- 就是扁平化管理没有中间商赚差价，所有节点都直接由大boss领导！

id	1	2	3	4	5	6	……
data	1	1	1	1	1	1	1



路径压缩示意图



带路径压缩的find操作

```
int find(int x)
{
    if (x == fa[x]) return x;
        else
        {
            fa[x] = find(fa[x]);
        //路径压缩，每次函数返回时顺便把其指向的父节点直接改为根节点
            return fa[x];
        }
}
```

带路径压缩的find操作简化版

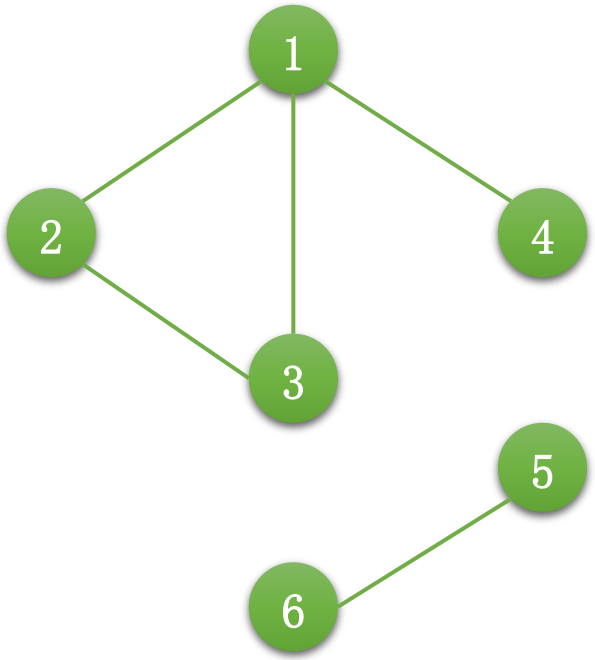
- 这个是终极版本了

```
int find(int x)
{
    return x == fa[x] ? x : fa[x] = find(fa[x]);
}
```

并查集的代码实现

2. 并查集的合并操作（join）

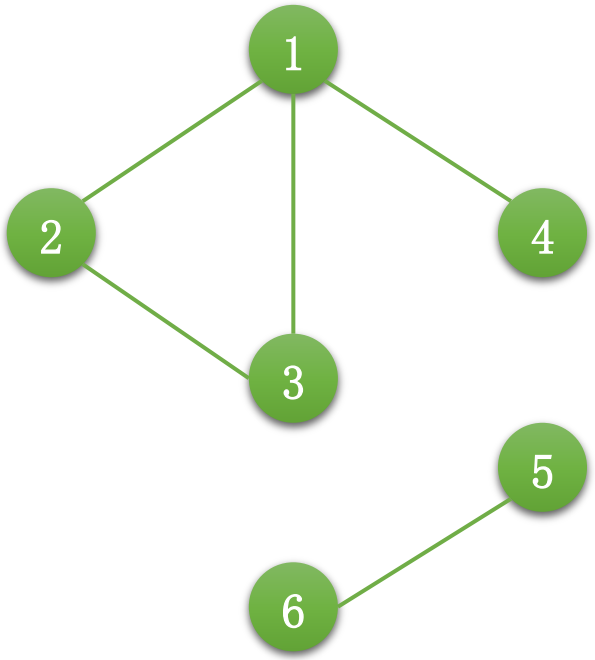
① 合并两个集合



并查集的代码实现

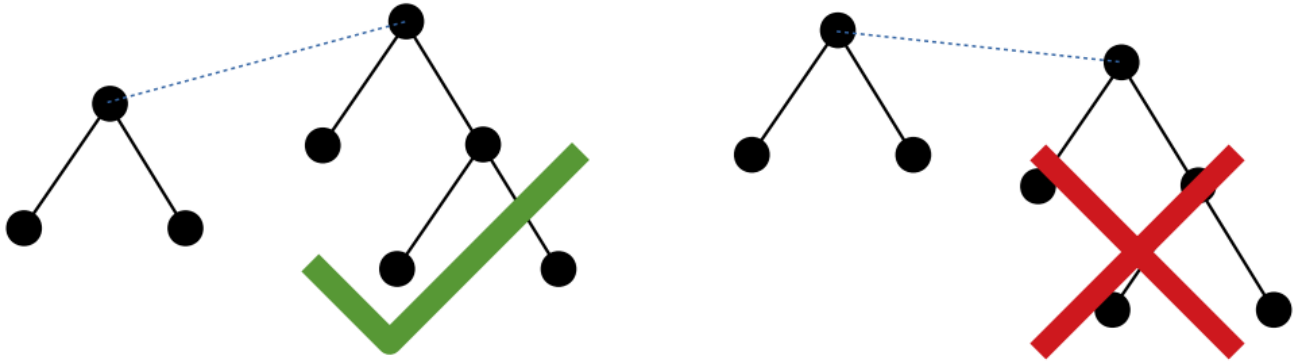
2. 并查集的合并操作（join）

```
void join(int x, int y)
{
    if (find(x) != find(y)) fa[find(y)] = find(x);
    //判断两个节点是否有同一个根节点
    //如果不是就直接把右边节点的根节点设为左边节点
}
```



按秩合并

- 按秩合并是针对合并操作的优化
- 记录每棵树的高度，将高度小的树合并到高度大的树，以便降低整棵树的高度



按秩合并

- 假设两棵树的高度分别是h1和h2，则合并后树的高度h是：

```
if (h1 == h2) h = h1 + 1;

if (h1 != h2) h = max(h1, h2);
```

- 这里的秩即树的高度

按秩合并的代码

- 不过因为并查集的合并操作，复杂度已经接近O(1)，所以实际写的代码中，一般不再写按秩合并

```
void join(int x, int y)
{
    if (find(x) == find(y)) return;
    if (h[find(x)] < h[find(y)]) swap(x, y);
    //使高度大的树的根节点为合并后的根节点
    if (h[find(x)] == h[find(y)]) h[find(x)] ++;
    fa[find(y)] = find(x);
    //合并
}
```

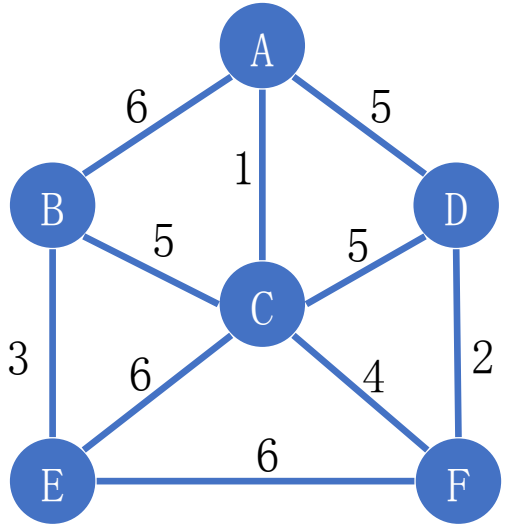
参考代码

```
#include<bits/stdc++.h>
using namespace std;
int n, m, fa[10010];
int find(int x)
{
    return x == fa[x] ? x : fa[x] = find(fa[x]);
    //带路径压缩的查找操作
}
```

```
int main()
{
    scanf("%d%d%d", &n, &m);
    for (int i = 1; i <= n; i ++)
        fa[i] = i;  //并查集初始化
    while (m --)
    {
        scanf("%d%d", &x, &y);  //读入一条边
        fa[find(x)] = find(y);  //合并操作
    }
    scanf("%d%d", &a, &b);
    if (find(a) == find(b)) printf("Yes\n");
    else printf("No\n");
    return 0;
}
```

Kruskal算法借助于并查集的例子

顶点编号	A	B	C	D	E	F
所属集合	1	2	3	4	5	6
所属集合	1	2	1	4	5	6
所属集合	1	2	1	4	5	4
所属集合	1	2	1	4	2	4
所属集合	1	2	1	1	2	1
所属集合	1	1	1	1	1	1



集合

- 给定a、b和p，初始时为所有[a,b]范围内的整数n建立一个集合。
- 如果存在两个处在不同集合的数字x和y，它们之间有不小于p的公共质因子，则将x和y所在的集合合并
- 问最后会剩下几个集合

  a,b≤100,000，2≤p≤b

分析

- 枚举[p,b]中的所有质数x，将x在范围[a,b]内的所有倍数用并查集连起来
- 最后统计不同的集合个数

团伙

- 有n个强盗和m条信息，每条信息会表明某两个强盗之间是朋友关系还是敌人关系。并且他们坚信：朋友的朋友是朋友；敌人的敌人也是朋友
- 朋友之间会构成团伙，现在根据m条信息推出n个强盗之间会构成多少个团伙

  $n, m \leqslant 100,000$

分析

- 如果关系是朋友，就直接连起来
- 注意到一个人的所有敌人之间都是朋友关系，所以如果关系是敌人，先存起来，最后每个人的所有敌人之间都连起来即可

食物链

- 动物王国中有三类动物A、B、C，构成了循环的食物链：A吃B，B吃C，C吃A
- 现在有n只动物（但我们不知道这些动物具体是哪一类的），和m条信息。这m条信息依次说出来，并且是以下两种说法之一：

1. 动物x和动物y是同类
2. 动物x吃动物y

- 如果一句话自相矛盾或者与之前说过的真话有冲突，则为假话，否则为真话
- 现在要求m句话中假话的数量

分析

- 由于不清楚动物的具体信息，所以尝试用假设的方法来处理：如果动物x捕食动物y，那么当x为A时y就为B，x为B时y就为C，.....
- 相当于两个不同状态之间的等价关系
- 因此为每个动物建3个点，分别表示其为A、B和C时的状态。
- 使用并查集来处理这些等价关系，同时可以查询“x为A，y为B”是否可以从之前的真话推出来

分析

- 上面是针对真话的处理方法，在这之前还需要判断这一句话是不是真话。
- 我们可以查询与这句话相矛盾的情况，在并查集中查询即可
- 如果并查集中没有查出，则为真话

程序自动分析

- 有10^9个变量，分别为x_1至x_{10^9}，此外还有n个已知条件。每个已知条件会指定两个数字i和j，要么是$x_i=x_j$，要么是$x_i \neq x_j$
- 需要判断这n个条件是否存在冲突

  $n \leqslant 100,000$

分析

- 等式具有传递性，将相等的变量放入一个集合内，用并查集进行维护
- 处理完所有等式之后就可以知道哪些变量是相等的了
- 然后考虑所有的不等式是否能满足要求
- 如果不等式两边的变量在同一个集合内，就说明发生了冲突

课后加练

- luogu 3367 并查集模板
- luogu 1551 亲戚/家族
- luogu 1892 团伙
- luogu 2078 朋友
- luogu 1621 集合
- luogu 1111 修复公路
- luogu 2024 食物链
- luogu 1197 星球大战
- luogu 2256 校运会百米跑
- luogu 2307 迷宫
- luogu 3144 关闭农场
- luogu 3958 奶酪
- luogu 1955 程序自动分析