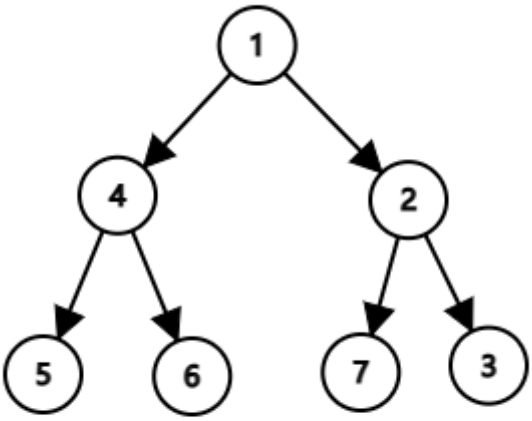
并查集

部分图源自$Pecco$的https://zhuanlan.zhihu.com/p/93647900，有兴趣的同学可以去看一下

问题引入1

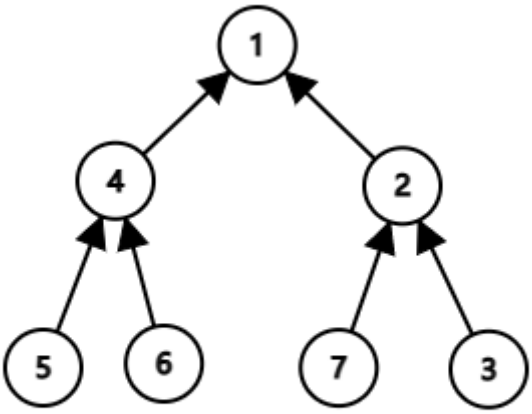
假设给你一张族谱图，族谱图的形式以若干条形似$a\ b$的形式给你，表示a是b的父亲



现在给定多张这样子的族谱，任意给定一个成员，问他的祖先是谁

解决方法

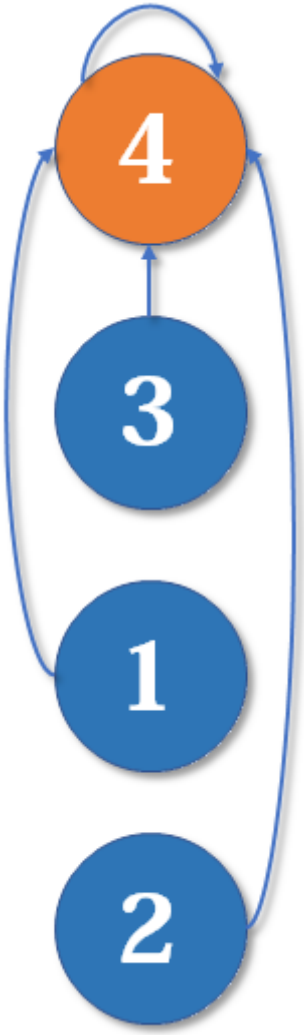
分析之后我们可以得到这样的一个做法，因为a是b的父亲，我们可以从b向a去连接一条边，然后再从我们所需要的节点开始递归即可。我们刚开始先假设所有人的父亲节点都是他本身；



那么我们可以写出以下的代码

```
int fa[maxn];
void init()
{
    for(int i = 1;i <= n;++i) fa[i] = i;
}
int find(int s)
{
    if(fa[s] == s) return s;
    return find(fa[s]);
}
```

但是这样子我们发现对于多次询问来说，如果我们最终形成的图很大的话，在多次找下面的结点会导致时间复杂度很大，还有就是在找深度较深的节点的时候，我们实际上也可以顺便把这条路径上的节点都更新了。因为我们最后实际上只会关心**一个节点** 那么为了解决这个问题我们可以采用 **路径压缩** 的方法，也就是每个节点到根节点的路径尽可能的短




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

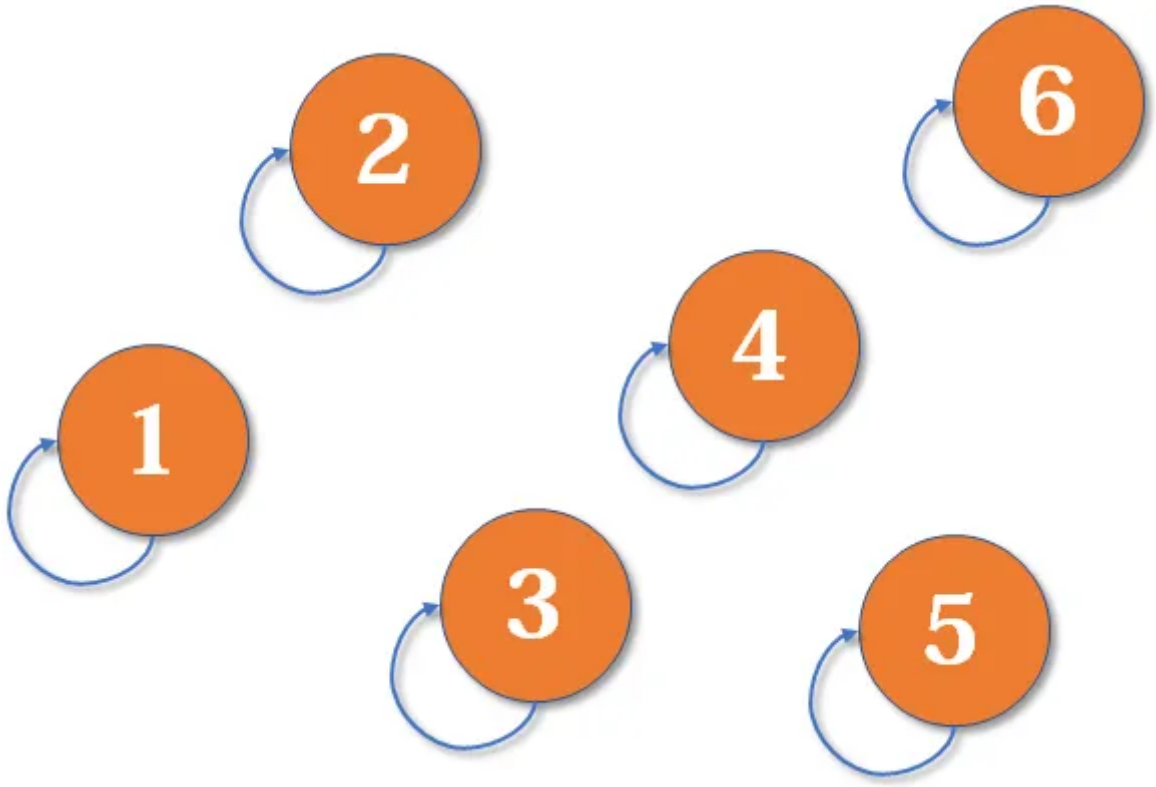
通常为了方便起见写成一行来用

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

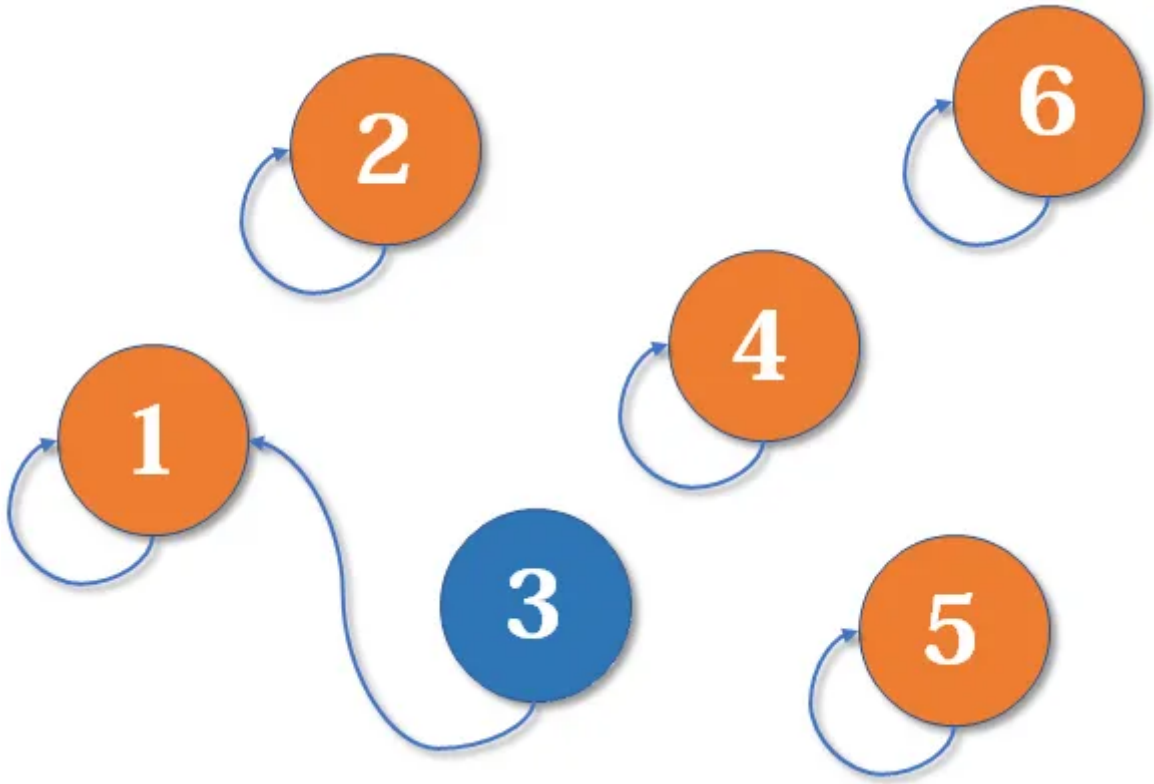
以上的操作我们可以认为是并查集中的 **查询** 操作

问题引入2

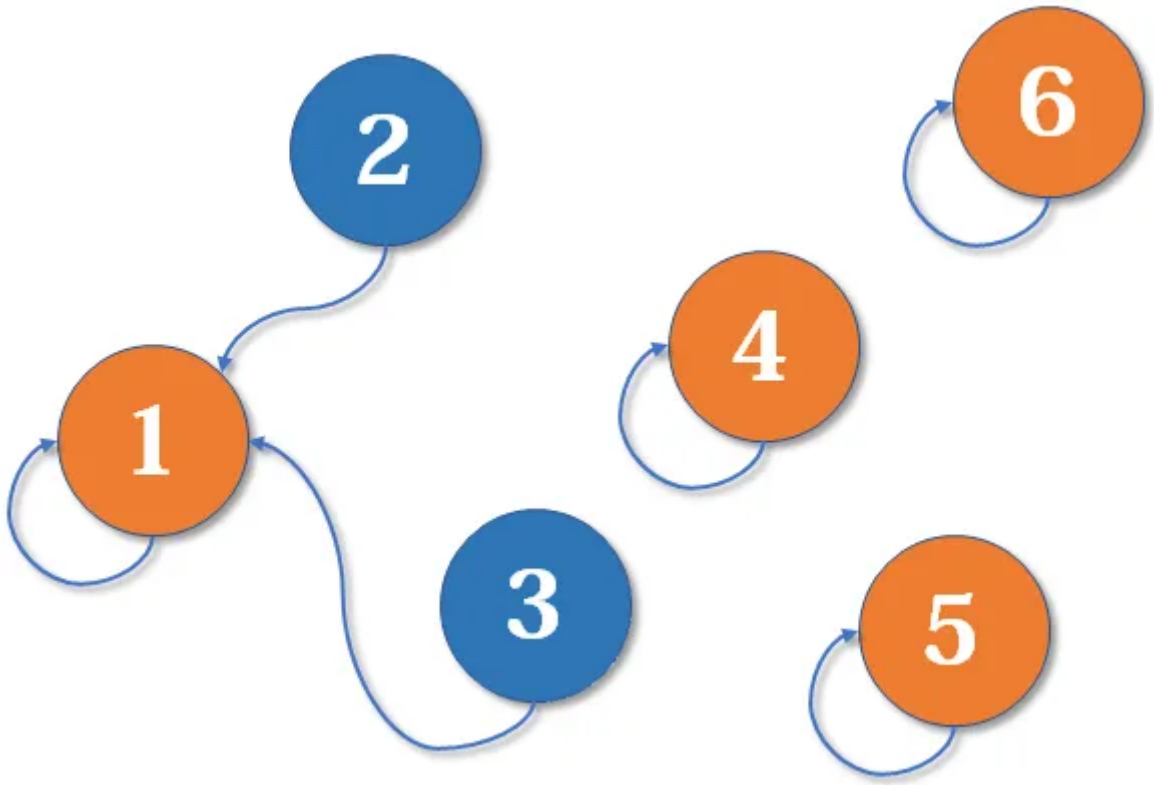
假设这是一个武林世界，这个武林世界中存在着若干个帮派，刚开始每个帮派只有帮主一个人。



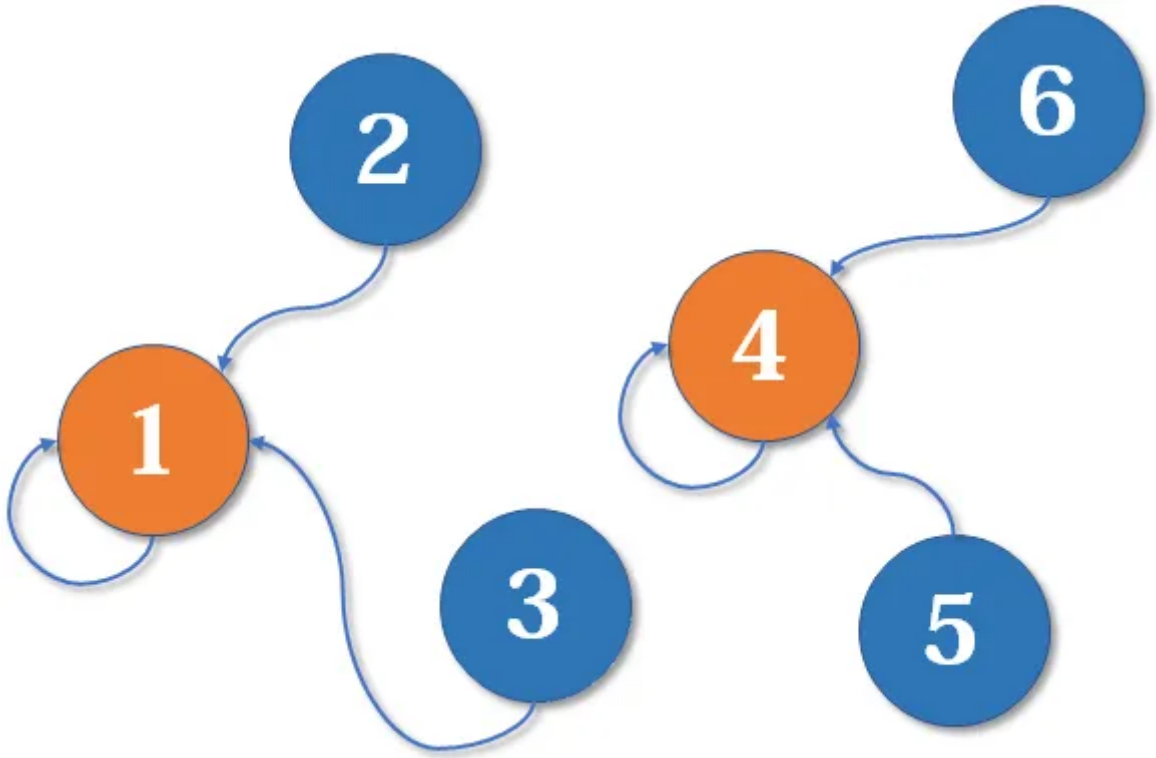
如果刚开始我们先让1号帮派和3号帮派比武，假设1赢了，那么1号帮派就把3号帮派也收入手下



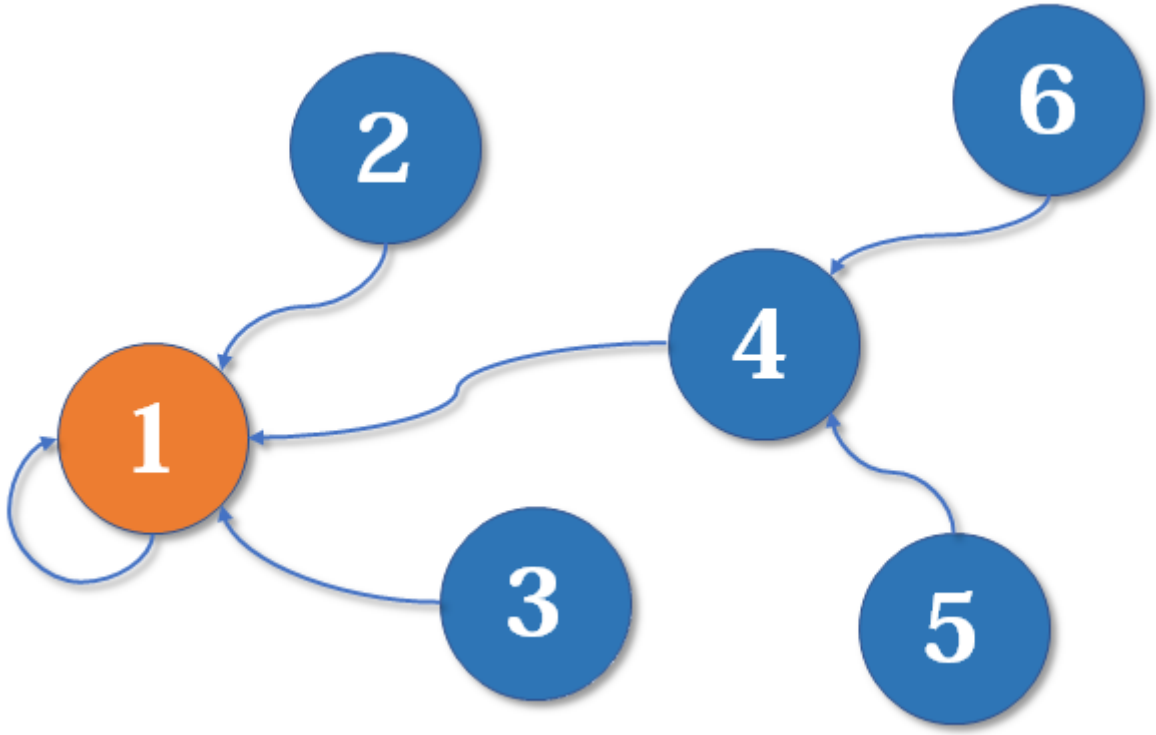
然后我们假设2号想要和3号比武，但3号表示让我帮主来收拾你(合并代表元素)。不妨设这次又是1号赢了，那么2号也认1号做帮主。



假设我们经历了一些江湖斗争之后局势变成了以下的情况:



然后在这个局势下，如果2号想要和6号比武，和刚刚说的一样，喊帮助1号和4号出来大家，1号胜利之后，4号认1号为帮主，当然4下面的所有手下也变成了1的小弟了



那么对于合并我们可以把他抽象成下面一个过程，对于两个集合来说，先给定任意两个不同集合的元素，然后通过这两个元素找他他们所在集合的代表元素，然后将某一个代表元素给合并到另外一个代表元素中

那么对于合并来说我们可以写下面的代码

```
void merge(int x, int y)
{
    int fx = find(x);
    int fy = find(y);
    fa[fx] = fy;
}
```

例1

给定n个点m条边，问在这片森林中每个连通块的点数和边数

做法：维护点数和边数的信息，在合并的时候把信息添加上去

例2

Problem - C - Codeforces

有三个排列a, b, c，其中给a序列和b序列，对于第i个位置来说，$c_i = a_i \ or \ b_i$

问c序列最多有多少种组成方法

做法: 首先我们先考虑序列是以 **排列** 的形式给出，排列的特征使得如果$a_i \neq b_i$，那么对于一个数字来说我最多可能出现在c的两个不同的位置

假设我们对于一个数字x，他在第i位和第j位都有出现，假设我第i位选了x，那么第j位我就选另外一个数字，反之同理，并且我们可以发现位置和位置之间都有相互牵制制约的关系，假设我们把这个位置看成边，两个数字看成点，那么这种关系形成了若干个环

那么如果是只有一个点的连通块，只有一种选择，我们只计算一次答案，如果是多个点的是存在两个选择那么我们就乘以2即可

小结

以上两个问题分别引入了并查集的两个重要操作，一个是 **查询(find)**另外一个是**合并(union)**

并查集的实际操作是对一些**集合**来进行的，来解决一些相交集合中或者不相交集合中元素之间的关系。一个集合中可能有多个属性，并查集在使用的时候可以将多个属性也同时合并起来

另外合并中还有按秩合并和启发式合并，有兴趣的同学可以去了解一下。

最小生成树

首先我们先来介绍一下图论的基础相关概念

相关概念

图是一个二元组$G = (V(G), E(G))$，其中$V(G)$为点的集合，$E(G)$为边的集合

对于V中的每个元素，我们可以称它为顶点或者节点

一个没有固定根结点的树称为 **无根树**（unrooted tree）。无根树有几种等价的形式化定义：

- 有n个结点，$n-1$条边的连通无向图
- 无向无环的连通图
- 任意两个结点之间有且仅有一条简单路径的无向图
- 任何边均为桥的连通图
- 没有圈，且在任意不同两点间添加一条边之后所得图含唯一的一个圈的图

在无根树的基础上，指定一个结点称为 **根**，则形成一棵 **有根树**（rooted tree）。

- **生成树（spanning tree）**：一个连通无向图的生成子图，同时要求是树。也即在图的边集中选择条，将所有顶点连通。

那么 **最小生成树** 就是边权和最小的生成树

注意：只有连通图才有生成树，而对于非连通图，只存在生成森林。

Kruskal算法

该算法的基本思想是从小到大加边，是个**贪心算法**

那么显而易见的我们首先对这些若干条边按照边权从小到大排序；然后进行加边操作

那加边怎么加边呢？

首先我们对于当前**需要加边**的两个点来说，他们一定不属于同一棵树内，他们一定属于两个森林

两个森林相当于两个集合，维护集合我们只需要使用并查集即可

如果两个点属于同一个集合，那么这两个就在同一棵树当中，我们就不需要往里面进行加边

```
int getf(int x)
{
    return x == f[x] ? x : f[x] = getf(f[x]);
}
void merge(int x,int y)
{
    int fx = getf(x);
    int fy = getf(y);
    f[fx] = fy;
}
array<int,3> edge[maxn * 100];
int ans,num;
void kru()
{
    sort(edge + 1,edge + 1 + m);
    for(int i = 1;i <= n;++i) f[i] = i;
    ans = num = 0;
    for(int i = 1;i <= m;++i)
    {
        auto [w,u,v] = edge[i];
        int fu = getf(u);
        int fv = getf(v);
        if(fu != fv)
        {
```

```
            num++;
            ans += w;
            f[fu] = fv;
        }
    }
}
```

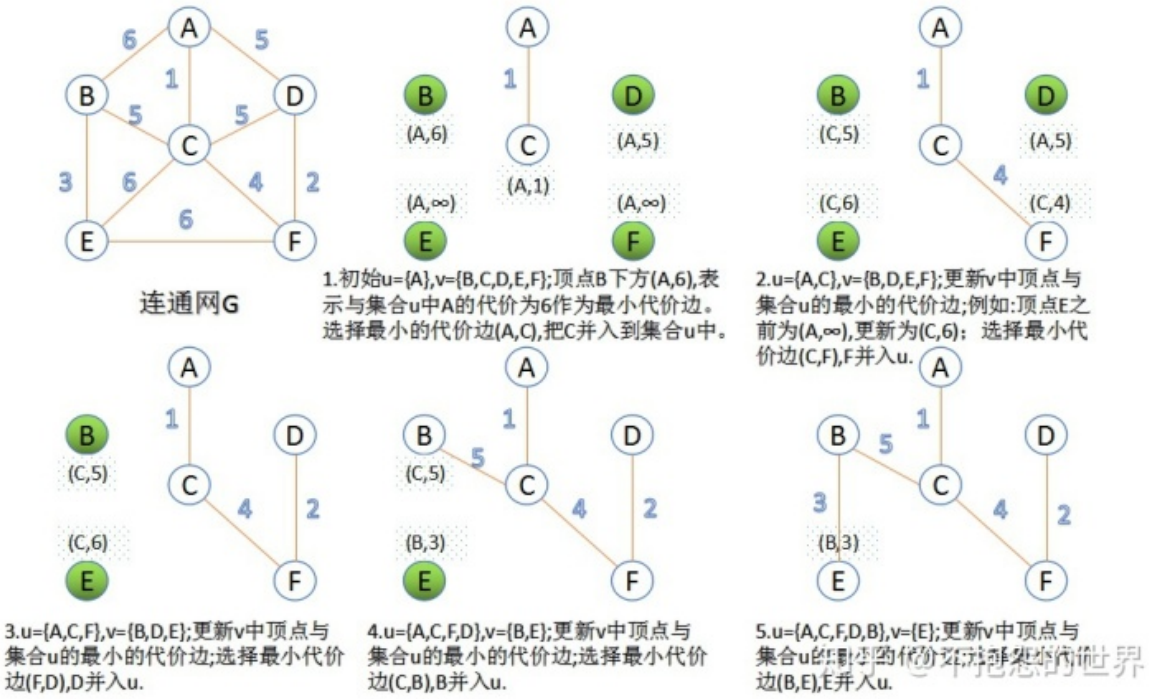
Prim算法

$Prim$算法的基本思想是从一个节点开始，不断加点，具体来说就是每次选择一个还没有被使用过的距离最小的节点然后去用这个节点新的边去更新其他节点的距离

那么这个算法实际上就是把一个点集划分成两个集合，一个集合是已经形成最小生成树的点的集合，另外一个集合是还没有形成最小生成树的点的集合

那么我们在具体写的时候可以先假定1点已经在这个集合中了，然后我们在这个基础上进行如下的操作：

1. 选取距离生成树集合最近的点，加入集合，并且将这个最近的距离加入权值
2. 若找到了$n-1$条边，则停止，否则继续进行上述操作




```
A:0   in
B:inf 6    5   5   5   in
C:inf 1    in
D:inf 5    5   2   in
E:inf inf  6   6   6   3   in
F:inf inf  4   in
```

```
template<typename T> inline bool chkmin(T &a, const T &b) { return a > b ? a =
b, 1 : 0; }
#define int long long
```

```
const int maxn = 1e2 + 10;
const int inf = 1e18;
int mp[maxn][maxn];
int dis[maxn],vis[maxn],n,m;
void prim()
{
    for(int i = 1;i <= n;++i)
    {
        dis[i] = inf,vis[i] = 0;
    }
    dis[1] = 0;
    for(int i = 1;i <= n - 1;++i)
    {
        int x = 0;
        for(int j = 1;j <= n;++j)
        {
            if(!vis[j] && (x == 0 || dis[j] < dis[x])) x = j;
        }
        vis[x] = 1;
        for(int j = 1;j <= n;++j)
        {
            if(!vis[j])
            {
                chkmin(dis[j],mp[x][j]);
            }
        }
    }
}
```

小结

$Kruskal$ 一般用于稀疏图上面，如果是稠密图的话可以考虑$Prim$