



浙江财经大学

Zhejiang University Of Finance & Economics



# 高级数据结构-图

信智学院 陈琰宏

---

# 主要内容



01

prim算法

02

并查集

03

04

05

# 什么是最小生成树



一个带权连通无向图 $G$ （假定每条边上的权值均大于零）中可能有多棵生成树，每棵生成树中所有边上的权值之和可能不同；**图的所有生成树中具有边上的权值之和最小的树称为图的最小生成树。**

按照生成树的定义， $n$ 个顶点的连通图的生成树有 $n$ 个顶点、 $n-1$ 条边。因此构造最小生成树的准则有几条：

- (1) 必须只使用该图中的边来构造最小生成树；
- (2) 必须使用且仅使用 $n-1$ 条边来连接图中的 $n$ 个顶点，生成树一定是连通的；
- (3) 不能使用产生回路的边。
- (4) 最小生成树的权值之和是最小的，但一个图的最小生成树不一定是唯一的。

构造最小生成树的方法：**克鲁斯卡尔(Kruskal)**算法和**普里姆(Prim)**算法。都得遵守以上准则

# 10.1 普里姆算法



普里姆（Prim）算法是一种构造性算法。假设 $G=(V,E)$ 是一个具有 $n$ 个顶点的带权无向连通图， $T=(U,TE)$ 是 $G$ 的最小生成树，其中 $U$ 是 $T$ 的顶点集， $TE$ 是 $T$ 的边集，则由 $G$ 构造从起始顶点 $v$ 出发的最小生成树 $T$ 的步骤如下：

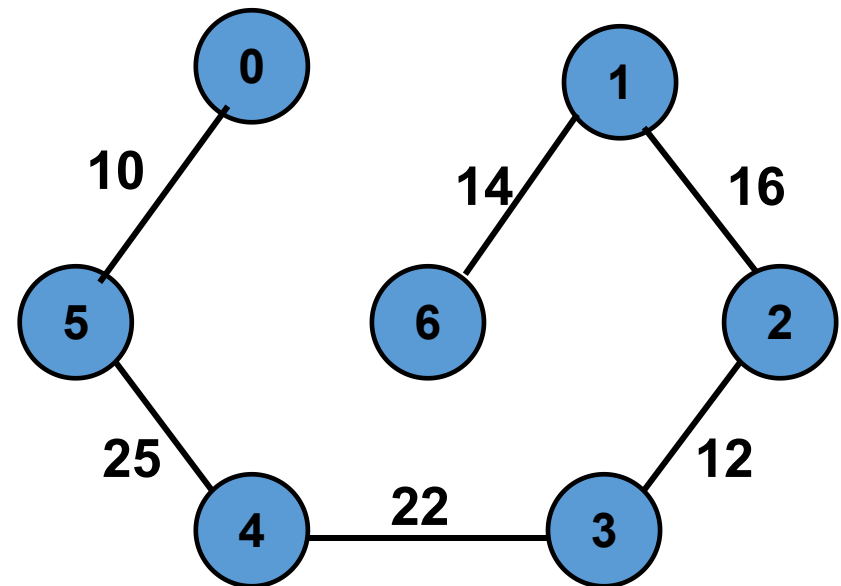
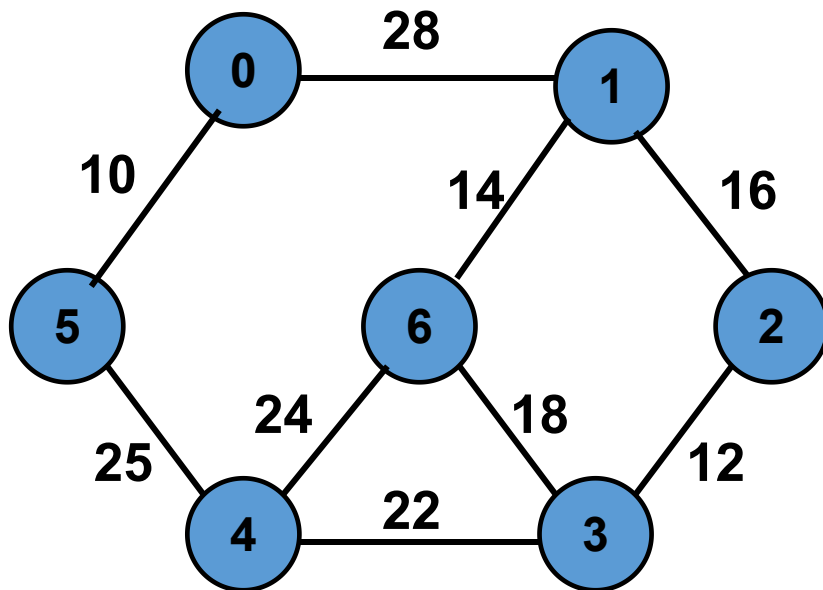
- (1) 初始化 $U=\{v\}$ 。以 $v$ 到其他顶点的所有边为候选边；
- (2) 重复以下步骤 $n-1$ 次，使得其他 $n-1$ 个顶点被加入到 $U$

中：

- ① 从候选边中挑选权值最小的边加入 $TE$ ，设该边在 $V-U$ 中的顶点是 $k$ ，将 $k$ 加入 $U$ 中；
- ② 考察当前 $V-U$ 中的所有顶点 $j$ ，修改候选边：若 $(k,j)$ 的权值小于原来和顶点 $j$ 关联的候选边，则用 $(k,j)$ 取代后者作为候选边。

# 10.1.1 prim构造最小生成树的过程

U: 当前生成树顶点集合,  
V: 不属于当前生成树的顶点集合。



# 10.1.1 prim构造最小生成树的过程



采用邻接矩阵来存储图。为了实现上述思想，必须定义2个辅助数组：

**cost[]**: 存放顶点集合V(生成树外)内的各顶点到顶点集合U(生成树顶点集合)内的权值最小的边的权值。

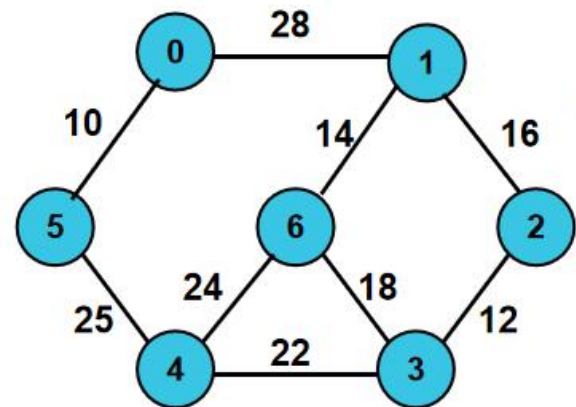
**nearvex[]**: 记录顶点的状态(是否已被选择)

从顶点0出发，2个辅助数组的初始状态为：

	0	1	2	3	4	5	6
<b>cost</b>	0	28	$\infty$	$\infty$	$\infty$	<u>10</u>	$\infty$

	0	1	2	3	4	5	6
<b>nearvex</b>	-1	0	0	0	0	0	0

V=5  
选边(0,5)



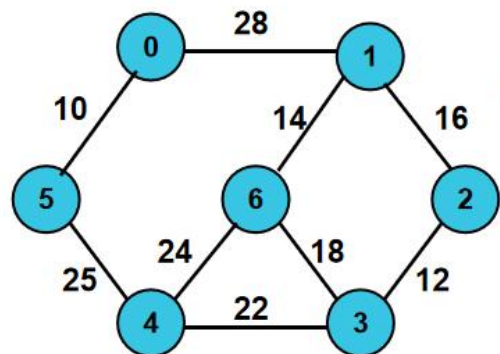
## 10.1.1 prim构造最小生成树的过程

	0	1	2	3	4	5	6		0	1	2	3	4	5	6	
<i>cost</i>	0	28	$\infty$	$\infty$	$\infty$	<u>10</u>	$\infty$		-1	0	0	0	0	0	0	
<i>nearvex</i>																

V=5  
选边(0,5)

因为在生成树顶点集合内最初只有一个顶点0，因此在*nearvex*数组中，只有表示顶点0的数组元素*nearvex*[0]=-1，其他都是0，表示其他点都未被处理。

数组*cost*表示生成树顶点集合（目前只有顶点0）顶点到生成树外各顶点的边的最小权值。初始值为邻接矩阵中顶点0所在的行。



## 10.1.1 prim构造最小生成树的过程



在prim算法里要重复做以下工作(简化):

1. 在`cost[ ]`中选择`nearvex[i] != -1`且`lowcost[i]`最小的边`i`, 用`v`标记它。
2. 将`nearvex[v]`改为`-1`, 表示它已经加入到生成树顶点的集合。
3. 修改`cost[ ]`。
4. 修改`nearvex[ ]`。



## 10.1.1 prim构造最小生成树的过程



在prim算法里要重复做以下工作(具体):

1. 在 $cost[ ]$ 中选择 $nearvex[i] \neq -1$ 且 $cost[i]$ 最小的边 $i$ , 用 $v$ 标记它。  
如第一次选中的 $v=5$ , 则边 $(0,5)$ 是选中的权值最小的边, 相应的权值为 $cost[5]=10$ 。
2. 将 $nearvex[v]$ 改为 $-1$ , 表示它已经加入到生成树顶点的集合, 如需要可输出选中的顶点 $v$ 。
3. 更新 $cost[ ]$ : 注意, 原来顶点 $v$ 不属于生成树顶点集合, 现在 $v$ 加入进来了, 则顶点集合 $V$ (生成树外)内的各顶点到顶点集合 $U$ (生成树顶点集合)内的权值需要更新,

$$cost[i] = \min\{cost[i], AdjMatrix[v][i]\},$$

$cost[i]$ 表示原权值,  $AdjMatrix[v][i]$ 表示新加入点 $v$ 到其他点的权重

# 10.1.1 prim构造最小生成树的过程

<i>cost</i>	0				10		
-------------	---	--	--	--	----	--	--

<i>nearvex</i>	-1				0		
----------------	----	--	--	--	---	--	--

V=5, 选边(0,5)

<i>cost</i>	0	28	∞	∞	25	10	∞
-------------	---	----	---	---	----	----	---

<i>nearvex</i>	-1	0	0	0	5	-1	0
----------------	----	---	---	---	---	----	---

V=4, 选边(5,4)

<i>cost</i>	0	28	∞	22	25	10	24
-------------	---	----	---	----	----	----	----

<i>nearvex</i>	-1	0	0	4	-1	-1	4
----------------	----	---	---	---	----	----	---

V=3, 选边(4,3)

<i>cost</i>	0	28	12	22	25	10	18
-------------	---	----	----	----	----	----	----

<i>nearvex</i>	-1	0	3	-1	-1	-1	3
----------------	----	---	---	----	----	----	---

V=2, 选边(3,2)

<i>cost</i>	0	16	12	22	25	10	18
-------------	---	----	----	----	----	----	----

<i>nearvex</i>	-1	2	-1	-1	-1	-1	3
----------------	----	---	----	----	----	----	---

V=1, 选边(2,1)

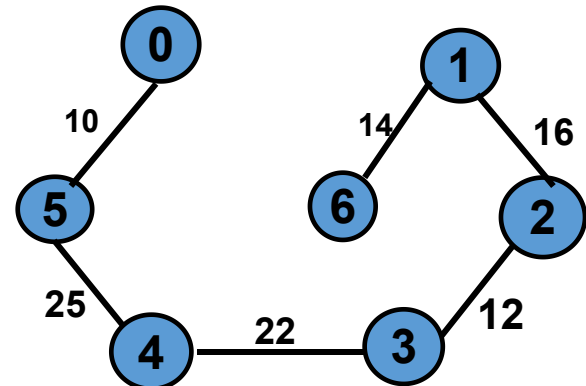
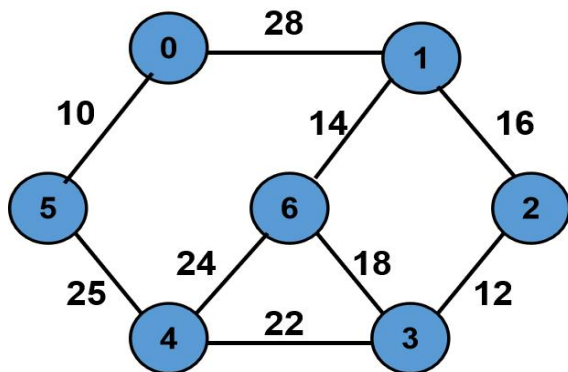
<i>cost</i>	0	16	12	22	25	10	14
-------------	---	----	----	----	----	----	----

<i>nearvex</i>	-1	-1	-1	-1	-1	-1	1
----------------	----	----	----	----	----	----	---

V=6, 选边(1,6)

<i>cost</i>	0	16	12	22	25	10	14
-------------	---	----	----	----	----	----	----

<i>nearvex</i>	-1	-1	-1	-1	-1	-1	-1
----------------	----	----	----	----	----	----	----



## 10.1.2 prim代码实现

```
10 void prim(){
11     int i,j,minn;
12     int cnt=1,sum=0;
13     for(i=1;i<=n;i++){
14         vis[i]=0; //初始全部的点在集合U中
15         cost[i]=e[1][i];
16     }
17     vis[1]=1; //把初始第一点添加到V集合
18     while(cnt<n){
19         minn=inf;
20         for(i=1;i<=n;i++){
21             if(vis[i]==0&&cost[i]<minn){ //寻找未被处理的最小cost[i]
22                 minn=cost[i];
23                 j=i; //找到新的添加到集合u中的点
24             }
25             vis[j]=1; //把新的点添加到集合u中
26             for(i=1;i<=n;i++){ //更新cost[i]
27                 //因为j的加入, 比较j到其他点的距离e[j][i]与原cost[i]
28                 if(vis[i]==0&&e[j][i]<cost[i])
29                     cost[i]=e[j][i]; //
30             }
31             cnt++;
32         }
33     }
```

```
3 #define maxn 100
4 const int inf=0xffff;
5 int e[maxn][maxn];
6 cost[maxn];
7 int vis[maxn];
8 //U: 当前生成树顶点集合,
9 //V: 不属于当前生成树的顶点集合。
10 //cost[i]表示 i节点当前加入集合中的最小花费
11 //相当于vetnearvex数组
12 // 标记 i 节点是否已加入集合u中
```

## 10.1.2 prim代码实现



上述算法的初始化时间是 $O(1)$ ， $k$ 循环中有两个循环语句，其时间大致为：

$$\sum_{k=0}^{n-2} \left( \sum_{j=k}^{n-2} O(1) + \sum_{j=k+1}^{n-2} O(1) \right) \approx 2 \sum_{k=0}^{n-2} \sum_{j=k}^{n-2} O(1)$$

令 $O(1)$ 为某一正常数 $C$ ，展开上述求和公式可知其数量级仍是 $n$ 的平方。所以，整个算法的时间复杂性是 $O(n^2)$

## 10.1.3 [3316] 还是畅通工程



浙江大学研究生复试题：某省调查乡村交通状况，得到的统计表中列出了任意两村庄间的距离。省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可），并要求铺设的公路总长度为最小。请计算最小的公路总长度。

输入

测试输入包含若干测试用例。每个测试用例的第1行给出村庄数目 $N$  ( $< 100$ )；随后的 $N(N-1)/2$ 行对应村庄间的距离，每行给出一对正整数，分别是两个村庄的编号，以及此两村庄间的距离。为简单起见，村庄从1到 $N$ 编号。

当 $N$ 为0时，输入结束，该用例不被处理。

输出

对每个测试用例，在1行里输出最小的公路总长度。

# [3316] 还是畅通工程



样例输入

```
3
1 2 1
1 3 2
2 3 4
4
1 2 1
1 3 4
1 4 1
2 3 3
2 4 2
3 4 5
0
```

样例输出

```
3
5
```

# [3316] 还是畅通工程

```
35 int main(){
36     while(cin>>n){
37         if(n==0)
38             break;
39         m=n*(n-1)/2;
40         for(int i=1;i<=n;i++)
41             for(int j=1;j<=n;j++)
42                 if(i==j)
43                     e[i][j]=0;
44                 else
45                     e[i][j]=inf;
46         int u,v,w;
47         for(int i=1;i<=m;i++){
48             cin>>u>>v>>w;
49             e[u][v]=e[v][u]=w;
50         }
51         prim();
52     }
53     return 0;
54 }
```

```
1 #include<iostream>
2 using namespace std;
3 #define maxn 100
4 const int inf=0xffff;
5 int e[maxn][maxn],cost[maxn];
6 int vis[maxn];
7 int m,n,j;
8 // cost[i] 表示 i 节点当前加入集合中的最小花费
9 // vis [i] 标记 i 节点是否已加入集合u中
```

# [3316] 还是畅通工程

```
10 void prim(){
11     int i,j,minn;
12     int cnt=1,sum=0;
13     for(i=1;i<=n;i++){
14         vis[i]=0; // 初始全部的点在集合A中
15         cost[i]=e[1][i];
16     }
17     vis[1]=1; // 把初始第一点添加到B集合
18     while(cnt<n){
19         minn=inf;
20         for(i=1;i<=n;i++){
21             if(vis[i]==0&&cost[i]<minn){ // 在集合v中找d[i]最小的权值
22                 minn=cost[i];
23                 j=i; // 找到新的添加到集合u中的点
24             }
25             vis[j]=1; // 把新的点添加到集合u中
26             sum=sum+cost[j]; // 求和, 计算最小距离
27             cnt++;
28             for(i=1;i<=n;i++){ // 更新cost[i]
29                 if(vis[i]==0&&e[j][i]<cost[i])
30                     cost[i]=e[j][i];
31             }
32         }
33     }
34     cout<<sum<<endl;
}
```



# [2910] 城市公交网建设问题



有一张城市地图，图中的顶点为城市，无向边代表两个城市间的连通关系，边上的权为在这两个城市之间修建高速公路的造价，研究后发现，这个地图有一个特点，即任一对城市都是连通的。现在的问题是，要修建若干高速公路把所有城市联系起来，问如何设计可使得工程的总造价最少？

输入

$n$ （城市数， $1 \leq n \leq 100$ ）

$e$ （边数）

以下 $e$ 行，每行3个数 $i, j, w_i$ ，表示在城市 $i, j$ 之间修建高速公路的造价。

输出

$n-1$ 行，每行为两个城市的序号，表明这两个城市间建一条高速公路。注意看测试数据的输出顺序，有排序要求。

# [2910] 城市公交网建设问题



样例输入

```
5 8
1 2 2
2 5 9
5 4 7
4 1 10
1 3 12
4 3 6
5 3 3
2 3 8
```

样例输出

```
1 2
2 3
3 4
3 5
```

## [2910] 城市公交网建设问题

典型最小生成树，同时输出选中的边。定义边结构。

```
1 #include <iostream>
2 #include <cstring>
3 #include <algorithm>
4 using namespace std;
5 #define inf 0x3f3f3f3f
6 int n,m,e[110][110]; //e数组保存输入原始数据
7 struct Node
8 {
9     int pre; //边的起始点
10    int to; //边的终点
11    int cost; //权值
12 } edge[110], ans[110]; //ans保存被选中的边
13 int k=0;
14 bool cmp(Node a, Node b)
15 { // 先按起始点排序, 再按终点排序
16     if (a.pre != b.pre) return a.pre < b.pre;
17     else return a.to < b.to;
18 }
```

```
57 int main ()
58 {
59     cin >> n >> m;
60     int u, v, w;
61     memset(e, inf, sizeof(e)); // 初始化为极大值
62     for (int i=1; i<=n; i++)
63         e[i][i]=0;
64     for (int i=0; i<m; i++)
65     { // 输入数据
66         cin >> u >> v >> w;
67         e[u][v]=w;
68         e[v][u]=w;
69     }
70     prim();
71     sort(ans, ans+k, cmp);
72     for (int i=0; i<k; i++)
73         cout << ans[i].pre << " " << ans[i].to << endl;
74     return 0;
75 }
```

```

21 void prim()
22 {
23     int book[110]={0};
24     //记录当前点是否已经被处理
25     for (int i=1;i<=n;i++)
26     { //输入数据
27         edge[i].cost=e[1][i]; //权值
28         edge[i].pre=1;
29         edge[i].to=i;
30     }
31     book[1]=1;
32     // 赋值为1, 表示起
33     int minn,u;
34     for (int i=0;i<n-1
35     {
36         for (int i=0;i<n-1;i++)
37         {
38             minn=inf;
39             for (int j=1;j<=n;j++)
40             { //如果当前点为处理, 且权值最小
41                 if (!book[j]&&edge[j].cost<minn)
42                 {
43                     minn=edge[j].cost;
44                     u=j;
45                 }
46             }
47             if (edge[u].pre<edge[u].to)
48                 ans[k].pre=edge[u].pre,ans[k].to=edge[u].to;
49             else ans[k].pre=edge[u].to,ans[k].to=edge[u].pre;
50             k++; //
51             book[u]=1; //把权值最小的边的节点加入 处理集合
52             for (int j=1;j<=n;j++)
53             if (!book[j]&&edge[j].cost>e[u][j])
54             { //更新dis[j]数组, 记录集合u中的点到 集合v中的的距离
55                 edge[j].cost=e[u][j]; //u->j
56                 edge[j].pre=u;
57                 edge[j].to=j;
58             }
59         }
60     }
61 }

```

# 2[2937] 局域网



某个局域网内有 $n(n \leq 100)$ 台计算机，由于搭建局域网时工作人员的疏忽，现在局域网内的连接形成了回路，我们知道如果局域网形成回路那么数据将不停的在回路内传输，造成网络卡的现象。因为连接计算机的网线本身不同，所以有一些连线不是很畅通，我们用 $f(i,j)$ 表示 $i,j$ 之间连接的畅通程度( $f(i,j) \leq 1000$ )， $f(i,j)$ 值越小表示 $i,j$ 之间连接越通畅， $f(i,j)$ 为0表示 $i,j$ 之间无网线连接。现在我们需要解决回路问题，我们将除去一些连线，使得网络中没有回路，并且被除去网线的 $\sum f(i,j)$ 最大，请求出这个最大值。

第一行两个正整数 $n$   $k$

接下来的 $k$ 行每行三个正整数 $i$   $j$   $m$ 表示 $i,j$ 两台计算机之间有网线联通，通畅程度为 $m$ 。

输出

一个正整数， $\sum f(i,j)$ 的最大值。

样例输入

```
5 5
1 2 8
1 3 1
1 5 3
2 4 5
3 4 2
```

样例输出

```
8
```

# 2[2937] 代码

非常基础的最小生成树的题，直接用prim算法。先求出总的权值的和，然后用prim累加最小权值和，前者减去后者就是最大值。

```
22 int main()
23 {
24     cin>>n>>m;
25     for(i=1;i<=n;i++)
26         for(j=1;j<=n;j++)
27             if(i==j)
28                 e[i][j]=0;
29         else
30             e[i][j]=inf;//构造邻接矩阵
31     for(i=1;i<=m;i++)
32     {
33         cin>>a>>b>>c;
34         e[a][b]=e[b][a]=c;
35         max1+=c;//max储存所有的畅通程度
36     }//读入数据并存入矩阵
37     prim();
38
39     for(i=1;i<=n;i++)
40         sum+=minn[i];//累加权值
41     cout<<max1-sum;
42     return 0;
43 }
```

# 2[2937] 代码

```
1  #include<iostream>
2  using namespace std;
3  int a,b,c,i,j,k,l,m,n,inf=9999999,sum,max1;
4  int e[101][101],minn[101];
5  bool u[101];
6  void prim(){
7      memset(minn,0x7f,sizeof(minn));
8      memset(u,1,sizeof(u));//初始化为True,表示所有点未被访问
9      minn[1]=0;
10     for(i=1;i<=n;i++)
11     {
12         k=0;
13         for(j=1;j<=n;j++)//寻找一个与已访问的点相连的权值最小的未被访问的点k
14             if(u[j] && minn[j]<minn[k])
15                 k=j;
16         u[k]=false;//将k加入最小生成树,标记已访问
17         for(j=1;j<=n;j++)//修改与k相连的所有未被访问的点
18             if(u[j] && e[k][j]<minn[j])
19                 minn[j]=e[k][j];
20     }
21 }
```

## 10.2 kruskal算法



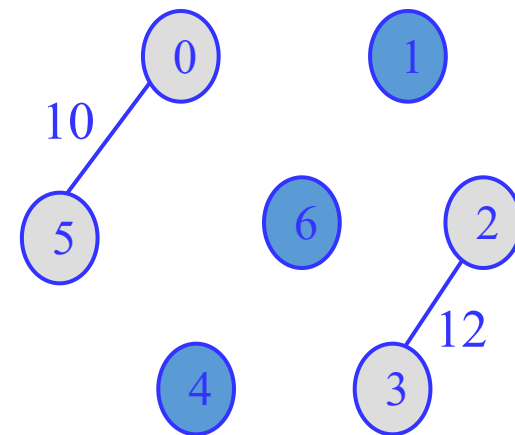
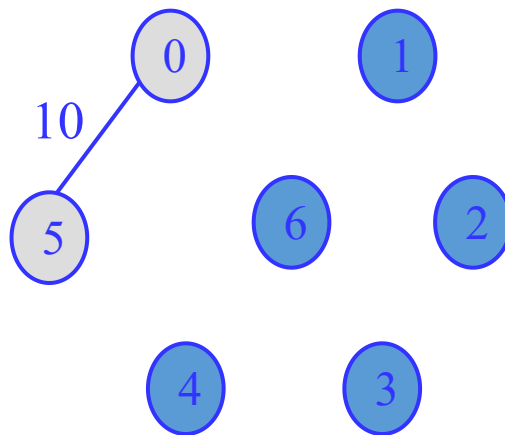
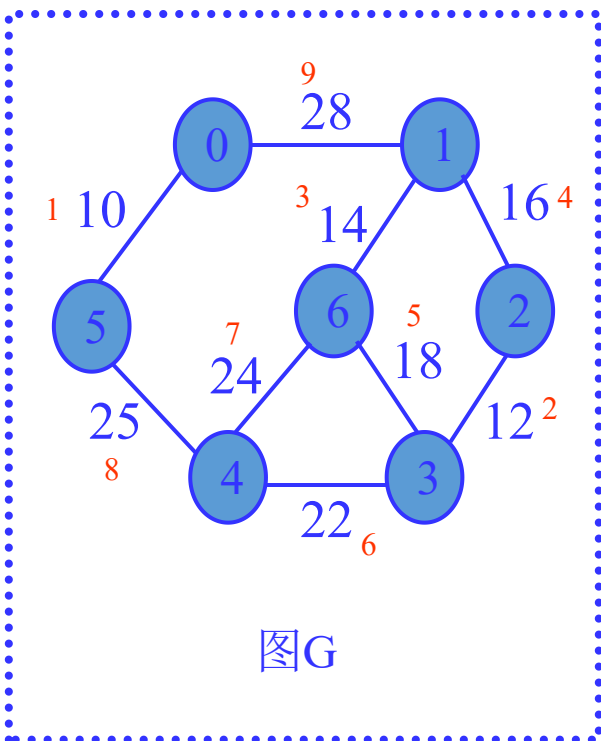
克鲁斯卡尔（Kruskal）算法是一种按权值的递增次序选择合适的边来构造最小生成树的方法。假设 $G=(V,E)$ 是一个具有 $n$ 个顶点的带权连通无向图， $T=(U,TE)$ 是 $G$ 的最小生成树，则构造最小生成树的步骤如下：

（1）置 $U$ 的初值等于 $V$ （即包含有 $G$ 中的全部顶点）， $TE$ 的初值为空集（即图 $T$ 中每一个顶点都构成一个分量）。

（2）将图 $G$ 中的边按权值从小到大的顺序依次选取：若选取的边未使生成树 $T$ 形成回路，则加入 $TE$ ；否则舍弃，直到 $TE$ 中包含 $n-1$ 条边为止。



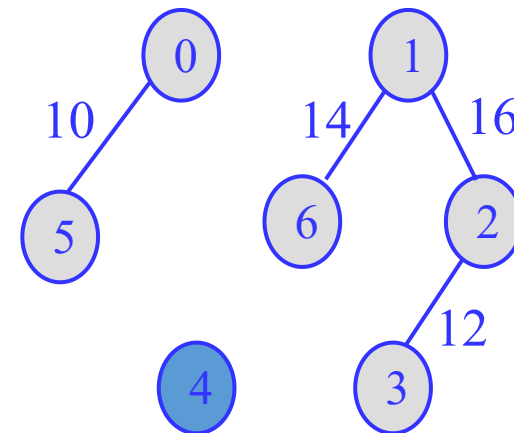
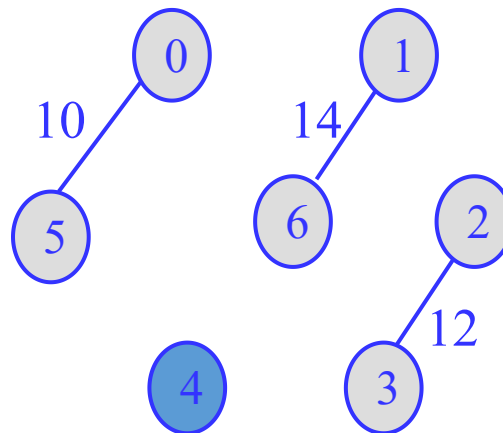
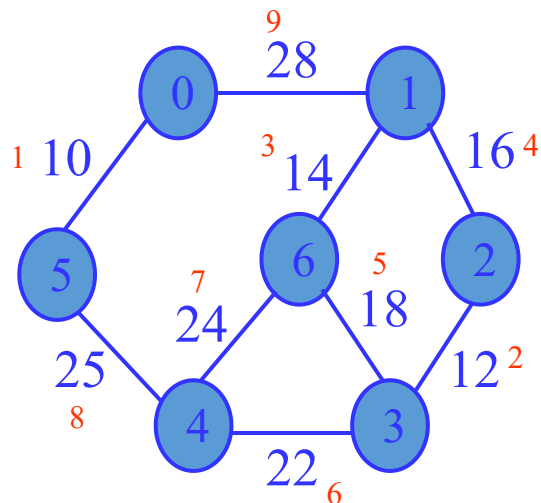
# 10.2.1 kruskal算法求解过程



按边大小递减排序

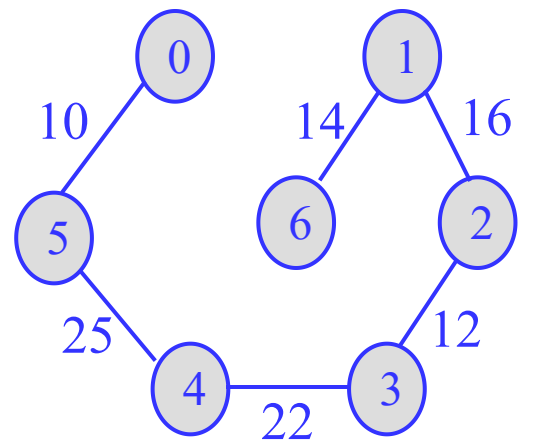
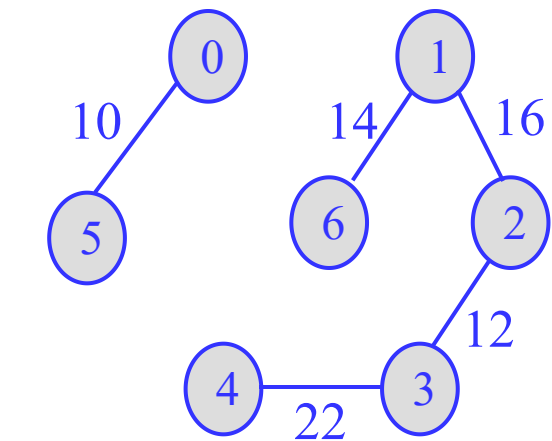
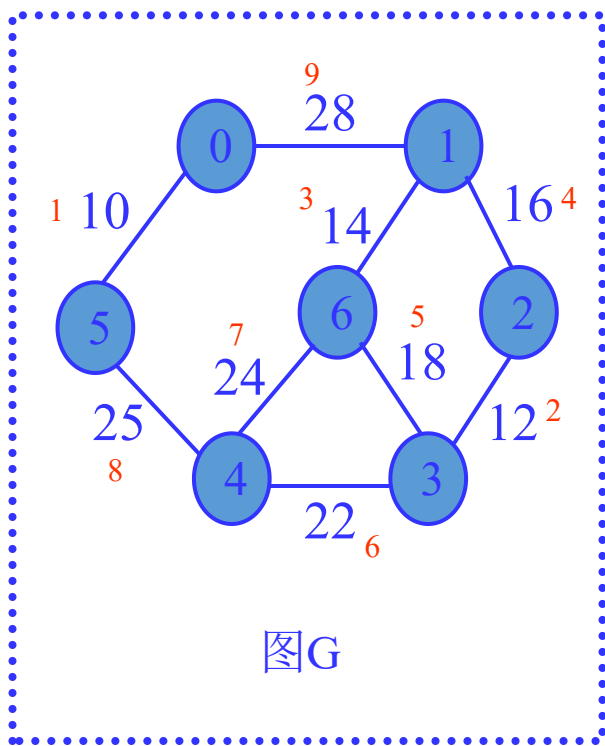
克鲁斯卡尔算法求解最小生成树的过程 (1)

# 10.2.1 kruskal算法求解过程



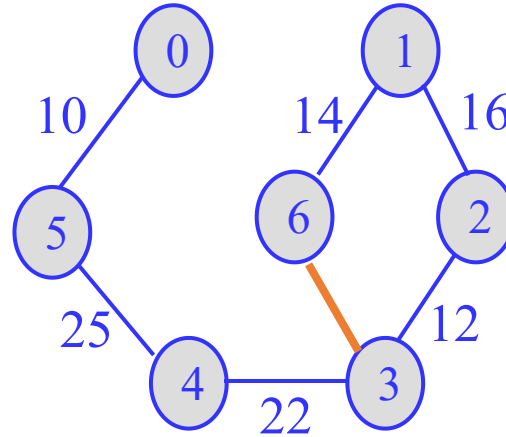
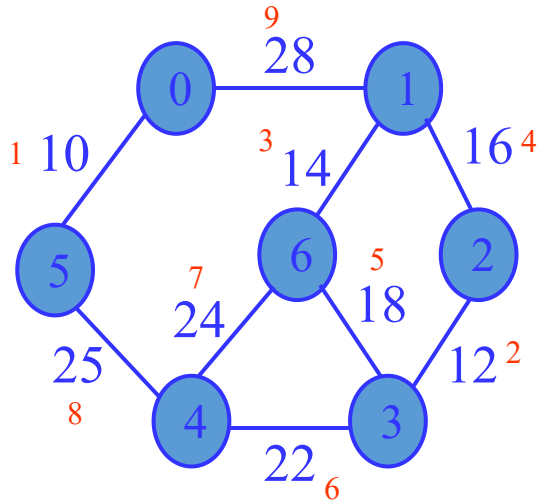
克鲁斯卡尔算法求解最小生成树的过程 (2)

# 10.2.1 kruskal算法求解过程



克鲁斯卡尔算法求解最小生成树的过程（3）

# 如何解决出现回路的问题?



# Kruskal (克鲁斯卡尔) 算法的伪代码:



```
void Kruskal ( Graph G )
{ T = {空集};
  while ( T 中收集的边不到n-1条 && 原图的边集E非空 ) {
    从E中选择最小代价边(v, w);
    从E中删除此边(v, w);
    if ( (v, w)的选取在T中构成回路 )
      将(v, w)加入T; /* 此判断由并查集的Find完成 */
    else
      彻底丢弃(v, w);
  } /* 结束while */
  if ( T 中收集的边不到n-1条 )
    printf( “图不连通” );
}
```

时间复杂性主要体现在  
while循环中, 循环 $|E|$ 次;  
带路径压缩的Find函数和  
Union函数完成需要 $O$   
 $(\log|V|)$ ; 总体时间复杂  
度为 $O(|E|\cdot\log|V|)$ 。

# ❖ Kruskal 核心代码



在实现克鲁斯卡尔算法Kruskal()时，用一个数组E存放图G中的所有边，要求它们是按权值递增排列的，为此先从图G的邻接矩阵中获取所有边集E，再采用直接插入排序法对边集E按权值递增排序。克鲁斯卡尔算法如下：

```
int kruskal() {
    int ans = 0;
    sort(e, e+n*(n-1)/2, cmp);
    for(int i = 1; i <= n; i++) fa[i] = i;// 并查集初始化

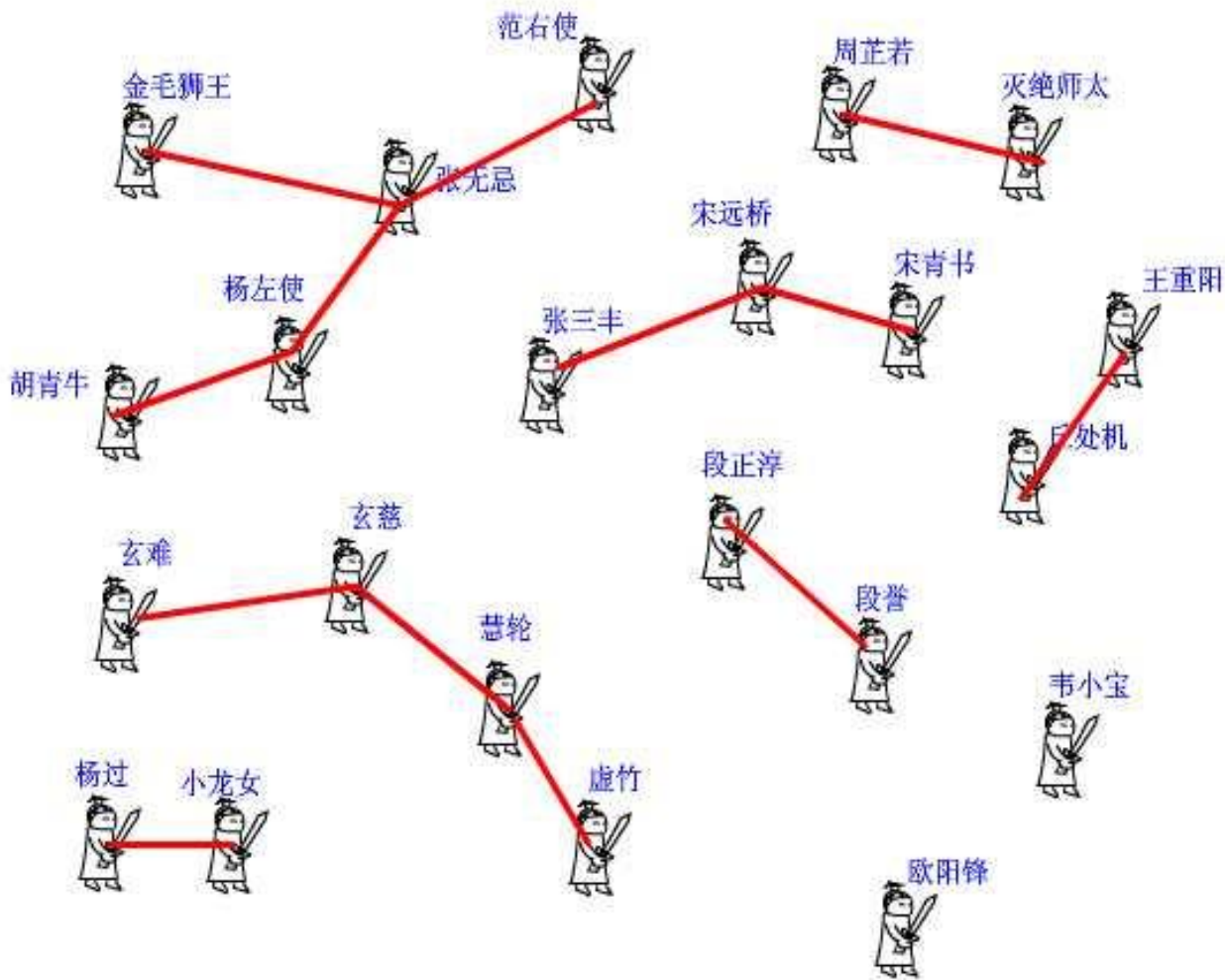
    for(int i = 0; i <= n*(n-1)/2; i++) {
        if(Union(e[i].u, e[i].v))
            ans += e[i].dis;
    }
    return ans;
}
```

## 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。

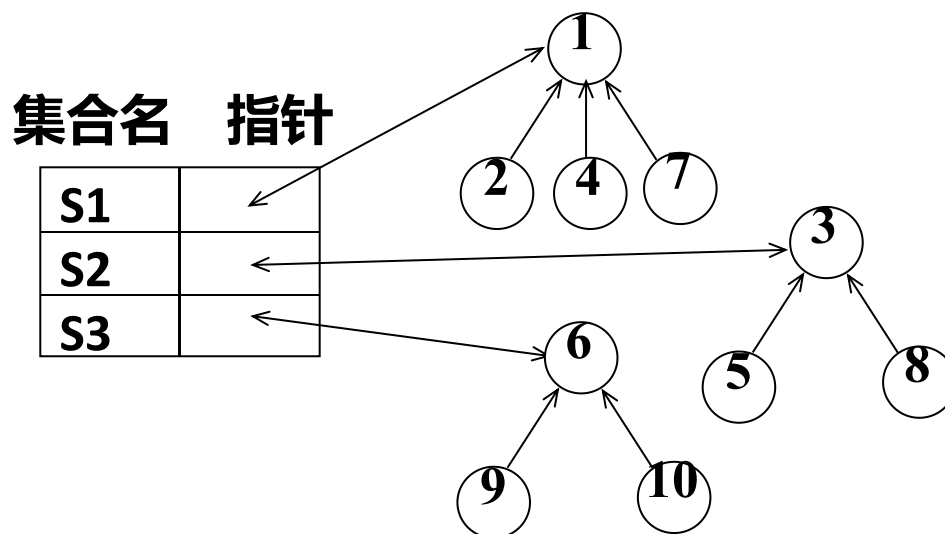
# 10.3 并查集





# 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：

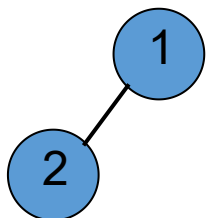


# 10.3.1 并查集存储

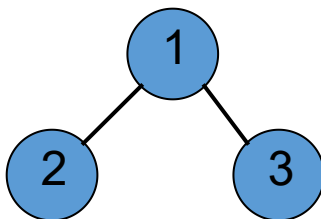
元素的合并图示：



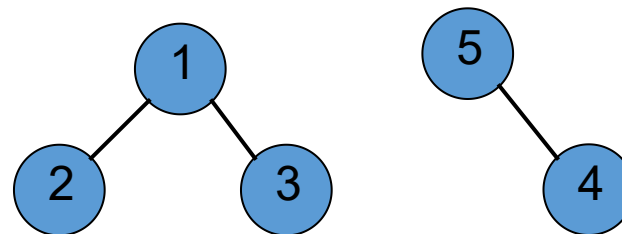
① 合并1和2



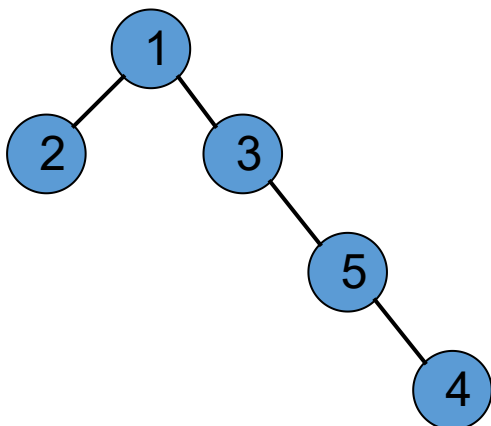
② 合并1和3



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 }
```

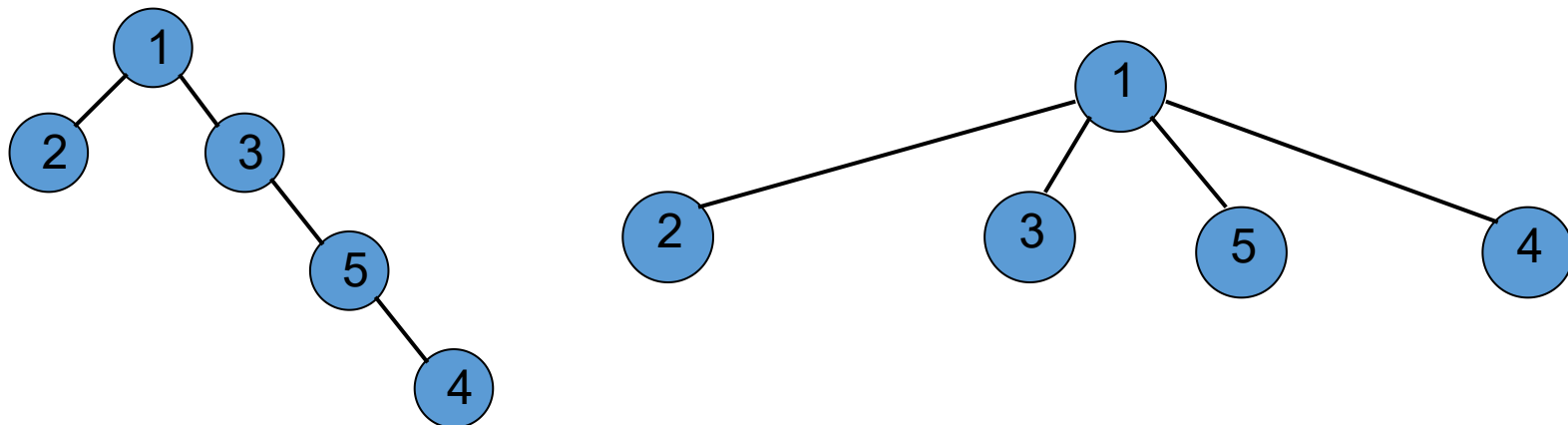
```
20 int find(int x)  
21 {  
22     if(x==parent[x])  
23         return x;  
24     int fx=find(parent[x]);  
25     parent[x]=fx;  
26     return fx;  
27 }
```

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：

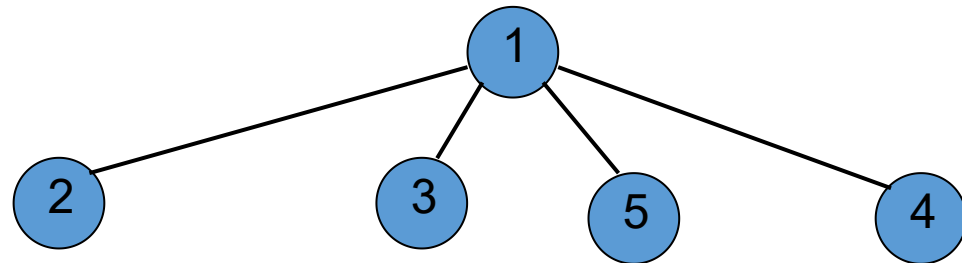
```
void unionn(int x,int y)
{
```

```
    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

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  int father[11];
4  int root1,root2;
5  void union1(int r1, int r2){
6      father[r2]=r1;//合并
7  }
8  int find(int x){//路径压缩, 每次更新节点的父节点,
9      //将所有链上的子节点全部链接到父节点上
10     if(father[x]!=x)father[x]=find(father[x]);
11     return father[x];//查找
12 }
```

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** ( $1 \leq N \leq 20000$ )。这些人的编号为1, 2, 3, ..., N。下面有M行 ( $1 \leq M \leq 1\ 000\ 000$ )，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** ( $1 \leq Q \leq 1\ 000\ 000$ )，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 $c_i, d_i$ ,  
输出一行：若 $c_i$ 和 $d_i$ 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

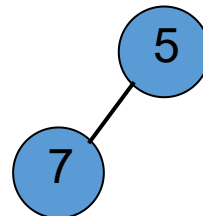
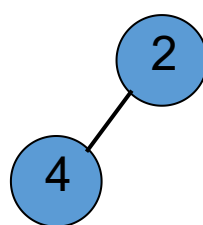
分离集合

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}



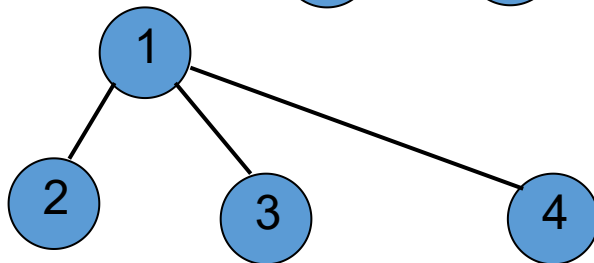
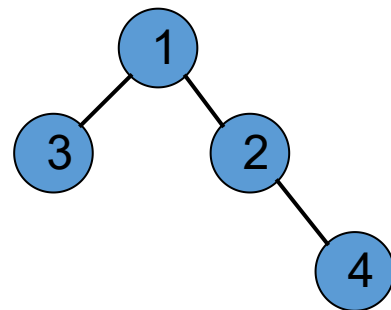
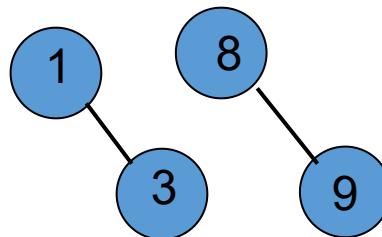
① 合并2和4

② 合并5和7



③ 合并1和3, 8和9

④ 合并1和2



## 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 }
```

## 10.4 kruskal算法实现

```
void Kruskal ( Graph G )
{ T = {空集};
  while ( T 中收集的边不到n-1条 && 原图的边集E非空 ) {
    从E中选择最小代价边(v, w);
    从E中删除此边(v, w);
    if ( (v, w)的选取在T中构成回路 )
      /* 判断由并查集的Find完成 */
      将(v, w)加入T; /* 此步在本题中可省略 */
    else
      彻底丢弃(v, w);
  } /* 结束while */
  if ( T 中收集的边不到n-1条 )
    printf( “图不连通” );
}
```

时间复杂性主要体现在  
while循环中，循环 $|E|$ 次；  
带路径压缩的Find函数和  
Union函数完成需要 $O$   
 $(\log|V|)$ ；总体时间复杂  
度为 $O(|E| \cdot \log|V|)$ 。

## 10.4.1 [3316] 还是畅通工程



浙江大学研究生复试题：某省调查乡村交通状况，得到的统计表中列出了任意两村庄间的距离。省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可），并要求铺设的公路总长度为最小。请计算最小的公路总长度。

输入

测试输入包含若干测试用例。每个测试用例的第1行给出村庄数目 $N$  ( $< 100$ )；随后的 $N(N-1)/2$ 行对应村庄间的距离，每行给出一对正整数，分别是两个村庄的编号，以及此两村庄间的距离。为简单起见，村庄从1到 $N$ 编号。

当 $N$ 为0时，输入结束，该用例不被处理。

输出

对每个测试用例，在1行里输出最小的公路总长度。

# [3316] 还是畅通工程



样例输入

```
3
1 2 1
1 3 2
2 3 4
4
1 2 1
1 3 4
1 4 1
2 3 3
2 4 2
3 4 5
0
```

样例输出

```
3
5
```



```
1 #include <iostream>
2 #include <algorithm>
3 using namespace std;
4 const int maxn = 1e2 + 10;
5 const int inf = 0x3f3f3f3f;
6 int n, m, x, y, z, fa[maxn];
7 struct node {
8     int u, v, dis;
9 } e[maxn*maxn];
10 //定义边数组, maxn个顶点, 最多 maxn*maxn条边
11 bool cmp (node a, node b) {
14 int find_fa(int x) { //查找父节点
18 int Union (int x, int y) {
25 void kruskal() {
36 int main()
37 {
38     while(scanf("%d", &n) && n) {
39         m = n * (n-1) / 2;
40         for(int i = 1; i <= m; i ++){
41             scanf("%d%d%d", &e[i].u, &e[i].v, &e[i].dis);
42             kruskal();
43         }
44         return 0;
45     }
```

```
11 bool cmp (node a,node b) {
12     return a.dis < b.dis;
13 } // 对边的权值做从小到大排序
14 int find_fa(int x) { //查找父节点
15     if(x == fa[x]) return x;
16     return fa[x] = find_fa(fa[x]);
17 }
18 int Union (int x, int y) {
19
20     int fx = find_fa(x), fy = find_fa(y);
21     if(fx == fy) return 0;
22     fa[fx] = fy;
23     return 1;
24 }
25 void kruskal() {
26     int ans = 0;
27     for(int i = 1; i <= n; i++) fa[i] = i;
28     sort(e+1, e+1+m, cmp);
29     for(int i = 1; i <= m; i++) {
30         if(Union(e[i].u, e[i].v)) {
31             ans += e[i].dis;
32         }
33     }
34     printf("%d\n", ans);
35 }
```

## 10.4 [2910]城市公交网建设问题



有一张城市地图，图中的顶点为城市，无向边代表两个城市间的连通关系，边上的权为在这两个城市之间修建高速公路的造价，研究后发现，这个地图有一个特点，即任一对城市都是连通的。现在的问题是，要修建若干高速公路把所有城市联系起来，问如何设计可使得工程的总造价最少？

输入

$n$ （城市数， $1 \leq n \leq 100$ ）

$e$ （边数）

以下 $e$ 行，每行3个数 $i, j, w_{ij}$ ，表示在城市 $i, j$ 之间修建高速公路的造价。

输出

$n-1$ 行，每行为两个城市的序号，表明这两个城市间建一条高速公路。注意看测试数据的输出顺序，有排序要求。

## 10.4 [2910]城市公交网建设问题



样例输入

```
5 8
1 2 2
2 5 9
5 4 7
4 1 10
1 3 12
4 3 6
5 3 3
2 3 8
```

样例输出

```
1 2
2 3
3 4
3 5
```

输入

$n$  (城市数,  $1 \leq n \leq 100$ )

$e$  (边数)

以下 $e$ 行, 每行3个数 $i, j, w_{ij}$ , 表示在城市 $i, j$ 之间修建高速公路的造价。

输出

$n-1$ 行, 每行为两个城市的序号, 表明这两个城市间建一条高速公路。

注意看测试数据的输出顺序, 有排序要求。

## 10.4 [2910]城市公交网建设问题

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define maxn 10010
4 struct edge
5 {
6     int from,to,flag,value;
7     // flag标记改变是否被选择 value权值
8 }num[maxn];
9 int parent[110], n,m;
10 bool cmp1(edge x,edge y)
11 { //按照权值排序
12     return x.value<y.value;
13 }
14 bool cmp2(edge x,edge y)
15 { //按照边排序
16     if(x.from==y.from)
17         return x.to<y.to;
18     return x.from<y.from;
19 }
```

## 10.4 [2910]城市公交网建设问题

```
20 int find(int x)
21 { //找到父节点
22     if(x==parent[x])
23         return x;
24     int fx=find(parent[x]);
25     parent[x]=fx;
26     return fx;
27 }
28 void kruskal()
29 {
30     for(int i=1;i<=m;i++)
31     {
32         int fx=find(num[i].from);
33         int fy=find(num[i].to);
34         if(fx!=fy)
35         {
36             parent[fx]=fy;
37             num[i].flag=1; //选取的边设置为1
38         }
39     }
40 }
```

## 10.4 [2910]城市公交网建设问题

```
43 int main()
44 {
45     scanf("%d%d",&n,&m);
46     for(int i=1;i<=n;i++)
47         parent[i]=i; // 初始自己为父节点
48     for(int i=1;i<=m;i++)
49     {
50         int x,y;
51         scanf("%d%d%d",&x,&y,&num[i].value);
52         num[i].from=min(x,y);
53         num[i].to=max(x,y);
54         num[i].flag=0;
55     }
56     sort(num+1,num+1+m,cmp1); // 按照权值排序
57     kruskal();
58     sort(num+1,num+1+m,cmp2); // 按照起始点排序
59     for(int i=1;i<=m;i++)
60     {
61         if(num[i].flag)
62             printf("%d %d\n",num[i].from,num[i].to);
63     }
64     return 0;
65 }
```

通过改进可以降低该算法的时间复杂度，通常认为克鲁斯卡尔算法的时间复杂度为 $O(e \log_2 e)$ 。由于与 $n$ 无关，所以克鲁斯卡尔算法特别适合于稀疏图求最小生成树。



# [3659] 北极通讯网络



北极的某区域共有 $n$ 座村庄 ( $1 \leq n \leq 500$ )，每座村庄的坐标用一对整数  $(x, y)$  表示，其中  $0 \leq x, y \leq 10000$ 。为了加强联系，决定在村庄之间建立通讯网络。通讯工具可以是无线电收发机，也可以是卫星设备。所有的村庄都可以拥有一部无线电收发机，且所有的无线电收发机型号相同。但卫星设备数量有限，只能给一部分村庄配备卫星设备。

不同型号的无线电收发机有一个不同的参数 $d$ ，两座村庄之间的距离如果不超过 $d$ 就可以用该型号的无线电收发机直接通讯， $d$ 值越大的型号价格越贵。拥有卫星设备的两座村庄无论相距多远都可以直接通讯。

现在有 $k$ 台 ( $1 \leq k \leq 100$ ) 卫星设备，请你编写一个程序，计算出应该如何分配这 $k$ 台卫星设备，才能使所有的无线电收发机的 $d$ 值最小，并保证每两座村庄之间都可以直接或间接地通讯。

# [3659] 北极通讯网络

---

输入

第一行包括两个整数 $n$ 、 $k$ ，表示村庄的数量和卫星设备的数量。

之后的 $n$ 行，输入 $x_i$ ， $y_i$ ，表示第 $i$ 个村庄的坐标。

输出

输出一个数，代表 $d$ 的最小值。输出保留两位小数。

样例输入

```
3 2  
10 10  
10 0  
30 0
```

样例输出

```
10.00
```

克鲁斯卡尔算法的扩展应用。

题目讲的有点抽象。我们抽象一下题目，发现题目是在求这样一个东西：我们需要找到一个最小的 $d$ 值，使得在删去权值大于 $d$ 的所有边后，剩下的联通块个数不超过 $k$ 。抽象完题目后，我们会发现一个事实：随着 $d$ 的值不断增大，所能形成的联通块数量会逐渐减小。这是一个显然的结论一看到联通块，我们可以快速的想到克鲁斯卡尔算法。

回忆克鲁斯卡尔算法的过程：

1. 将边权从小到大排序.
2. 依次从小到大扫描每一条边, 合并没有合并的点集.

我们发现第二步本质上就是在维护联通快的个数. 由此我们就得到了这道题的解决方法：

**正常执行克鲁斯卡尔算法，记录当前已经加入了几条边。当加入某条边后，恰好加入了 $n-k$ 条边，那么这条边的权值就是最终答案. 时间复杂度 $O(n \log n)$ 。**

```
1  #include<bits/stdc++.h>
2  #define N 510
3  #define INF 0x7fffffff
4  using namespace std;
5  struct Node{
6      int x,y;
7  }p[N];
8  struct node{
9      int from,to;
10     double dis;
11 }e[N*N];
12 int n,k;
13 int fa[N];
14 int cnt,tot;
15 double path(int x1,int y1,int x2,int y2)//计算距离
16 {
19 void add(int u,int v)//加边
20 {
26 int find(int x)//并查集基本操作
27 {
32 void merge(int x,int y)//并查集基本操作
33 {
36 bool cmp(const node &x,const node &y)//按边权从小到大排序
37 {
```

```
40 int main()
41 {
42     scanf("%d %d",&n,&k);
43     for(int i=1;i<=n;i++){
44         scanf("%d %d",&p[i].x,&p[i].y);
45         fa[i]=i;//并查集初始化
46     }
47     for(int i=1;i<n;i++)
48         for(int j=i+1;j<=n;j++)
49             add(i,j);
50     sort(e+1,e+1+tot,cmp);
51     for(int i=1;i<=tot;i++){
52         int u=e[i].from,v=e[i].to;
53         if(find(u)!=find(v)){
54             merge(u,v);//合并点集
55             cnt++;
56         }
57     }
58     if(cnt==n-k){//已经加入了n-k条边
59         printf("%.21f\n",e[i].dis);//输出结果
60         return 0;
61     }
62 }
```

```
15 double path(int x1,int y1,int x2,int y2)//计算距离
16 {
17     return sqrt(pow((x1-x2),2)+pow((y1-y2),2));
18 }
19 void add(int u,int v)//加边
20 {
21     tot++;
22     e[tot].from=u;
23     e[tot].to=v;
24     e[tot].dis=path(p[u].x,p[u].y,p[v].x,p[v].y);
25 }
26 int find(int x)//并查集基本操作
27 {
28     if(fa[x]==x)
29         return x;
30     return fa[x]=find(fa[x]);
31 }
32 void merge(int x,int y)//并查集基本操作
33 {
34     fa[find(y)]=find(x);
35 }
36 bool cmp(const node &x,const node &y)//按边权从小到大排序
37 {
38     return x.dis<y.dis;
39 }
```

## [3658] 黑暗城堡



你知道黑暗城堡有 $N$ 个房间 ( $1 \leq N \leq 1000$ )， $M$ 条可以制造的双向通道，以及每一条通道的长度。

城堡是树形的并且满足以下条件：如果所有的通道都被修建，设  $D[i]$  为第 $i$ 号房间与第一号房间的最短路径长度；而 $S[i]$ 为实际修建的树形城堡中第 $i$ 号房间与第1号房间的路径长度；要求对于所有整数 $i$  ( $1 \leq i \leq N$ )，有 $S[i]=D[i]$ 成立。

你想知道有多少种；不同的城堡修建放啊。当然，你只需要输出答案对  $2^{31} - 1$  取模之后的结果就行了。

输入

第一行有两个整数  $N$  和  $M$ 。

之后  $M$  行，每行三个整数  $X$ ， $Y$  和  $L$ ，表示可以修建  $X$  和  $Y$  之间的一条长度为  $L$  的通道。

输出

输出一个整数，表示答案对  $2^{31} - 1$  取模之后的结果。

# [3658] 黑暗城堡



样例输入

4 6 //第一行有两个整数  $N$  和  $M$ 。

1 2 1 // $X, Y, L$ 表示可以修建  $X$  和  $Y$  之间的一条长度为  $L$  的通道

1 3 2

1 4 3

2 3 1

2 4 2

3 4 1

样例输出

6

提示





这是一个最短路径树的典例之一。题意很明确——问你有多少棵最短路径树。虽然看起来 $n$ 比较大，但是是单源的所以不用担心复杂度会爆掉。求一遍最短路，然后枚举每个点以及连出去的边，看看有多少条到达这个点的路径长度是等于最短路径的。最后累乘起来就行了

# 今天的课程结束啦.....

---



下课了...  
同学们**再见!**