并查集

1、并查集基础

并查集基础问题。

```
#include<bits/stdc++.h>//1-1921-1        javacn
using namespace std;
int n,m,p,f[5100],c,d,a,b;

// 找到元素 x 的根（不含路径压缩）
int find(int x)
{
        // 如果 x 的父节点就是自己，那么 x 就是根，否则找 f[x] 的根
        return f[x]==x?x:find(f[x]);
        // 路径压缩版本的写法
        // 如果 x 的父节点就是自己，那么 x 就是根，否则将 x 的父节点直接指向 x 所在集合的根
        //return f[x]==x?x:f[x]=find(f[x]);
}

// 合并：将 x 和 y 合并到同一个集合
void merge(int x,int y)
{
        // 将 y 的根的父节点，设置为 x 的根
        f[find(y)] = find(x);
}
```

```
int main()
{
	//n 个人，m 个亲戚关系，p 次查询
	cin>>n>>m>>p;
	// 初始化，每个人的根节点都指向自己
	for(int i = 1;i <= n;i++)
	{
		f[i] = i;
	}

	for(int i = 1;i <= m;i++)
	{
		cin>>c>>d;
		//cd 是亲戚关系，将其合并到同一个集合
		merge(c,d);
	}

	//p 次查询
	for(int i = 1;i <= p;i++)
	{
		cin>>a>>b;
		// 问：ab 是否有亲戚关系，如果 ab 的根节点相同，那么是亲戚
		if(find(a)==find(b))
		{
			cout<<" Yes" <<endl;
		}
		else
		{
			cout<<" No" <<endl;
		}
	}
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//1-1921-2    是不是亲戚    w2016010182
using namespace std;
int parent[10100];
void Build(int n)
{
	for(int i=1;i<=n;i++)
	{
		parent[i]=i;
	}
}

int Find(int x)
{
	if(parent[x]!=x)
	{
		parent[x]=Find(parent[x]);
	}
	return parent[x];
}

void Un(int x,int y)
{
	parent[Find(y)]=Find(x);
}
```

```
int main()
{
	int n,m,p;
	cin>>n>>m>>p;
	Build(n);
	int x,y;
	for(int i=1;i<=m;i++)
	{
		cin>>x>>y;
		Un(x,y);
	}
	for(int i=1;i<=p;i++)
	{
		cin>>x>>y;
		if(Find(x)==Find(y))
		{
			cout<<"Yes"<<endl;
		}
		else
		{
			cout<<"No"<<endl;
		}
	}
	return 0;
}
```

求集合数量 -1，也就是要修的路的条数。

```
#include<bits/stdc++.h>//2-1922-1  修路     javacn
using namespace std;

int f[1005];// 存储每个结点的根

// 初始化
void init()
{
	for(int i = 1;i <= 1005;i++)
	{
		f[i] = i;
	}
}

// 查找，找出 x 的根
int find(int x)
{
	// 路径压缩 x 的父元素，直接指向 x 的根
	return x==f[x]?x:f[x]=find(f[x]);
}

// 合并
void merge(int x,int y)
{
	// 先找出根
	int p = find(x);
	int q = find(y);
	// 如果不在一个集合
	if(p!=q)
	{
		f[p] = q;// 将 q 的父元素指向 p
	}
}
```

```
int main()
{
	int n,m,p;
	while(true)
	{
		cin>>n;
		if(n == 0) return 0;
		cin>>m;

		init();// 初始化

		int x,y;
		for(int i = 1;i <= m;i++)
		{
			cin>>x>>y;
			merge(x,y);// 合并 xy
		}

		int cnt = 0;// 存储最少要修多少路（联通分量数量 -1）
		// 找出联通分量的个数
		for(int i = 1;i <= n;i++)
		{
			// 数一数有几个根
			if(i == f[i])
			{
				cnt++;
			}
		}

		cout<<cnt-1<<endl;
	}

	return 0;
}
```

```cpp
#include<bits/stdc++.h>//2-1922-2  修路    w2016010182
using namespace std;
int dl[1005];
int Find(int k)
{
	if(dl[k]!=k)
	{
		dl[k]=Find(dl[k]);
	}
	return dl[k];
}
void And(int x,int y)
{
	int fx=Find(x);
	int fy=Find(y);
	if(fx!=fy)
	{
		dl[fy]=fx;
	}
}
```

```
int main()
{
	while(1==1)
	{
		int n,m;
		scanf("%d",&n);
		if(n==0)	{	break;	}
		scanf("%d",&m);
		for(int i=1;i<=n;++i)
		{
			dl[i]=i;
		}
		for(int i=1;i<=m;++i)
		{
			int a,b;
			scanf("%d%d",&a,&b);
			And(a,b);
		}
		// 没有查询，fa 数组并不是都指向根节点，可有指向上一层父亲节点
		for(int i=1;i<=n;++i)
		{
			Find(i); // 查询一遍，fa 数组都更新指向根节点
		}
		map<int,int>p;
		for(int i=1;i<=n;++i)
		{
			pair<int,int> ap(dl[i],i);
			p.insert(ap);
		}
		printf("%d\n",p.size()-1);
		/*for(int i=1;i<=n;++i) {	printf("%d ",dl[i]);}
		printf("\n");*/
	}
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//2-1922-3  修路    w2016010182
using namespace std;
int fa[1015];
int sum=0;
int Find(int x)
{
	if(fa[x]!=x)
	{
		fa[x]=Find(fa[x]);
	}
	return fa[x];
}
void Un(int x,int y)
{
	fa[Find(y)]=Find(fa[x]);
}
int Find2(int x)
{
	if(fa[x]==x)
	{
		sum++;
	}
	return sum;
}
```

```
int main()
{
	int n,m,x,y;
	while(scanf("%d",&n))
	{
		sum=0;
		if(n==0)
		{
			break;
		}
		scanf("%d",&m);
		for(int i=1;i<=n;i++)
		{
			fa[i]=i;
		}
		for(int i=1;i<=m;i++)
		{
			scanf("%d%d",&x,&y);
			Un(x,y);
		}
		 //没有查询，fa 数组并不是都指向根节点，可有指向上一层父亲节点
		for(int i=1;i<=n;i++)
		{
			Find2(i);
		}
		cout<<sum-1<<endl;
	}
	return 0;
}
```

看到求最大值的最小判断要用二分解决。我们可以二分这个拥挤度，在判断这个拥挤度是否可行时，把所有拥挤度大于 mid 的边都去掉，最后并查集判断 s 点与 t 点是否联通即可。

```
#include<bits/stdc++.h>//3-1923-1      躲避拥堵的最佳路线      javacn
using namespace std;
int l=INT_MAX,r=INT_MIN,n,m,s,t,ans;
//cost: 记录拥挤度
int f[20010],x[20010],y[20010],cost[20010];
// 查找
int find(int x)
{
	return x == f[x]?x:f[x]=find(f[x]);
}

// 二分函数，看拥挤度在 mid 的情况下从 s 到 t 是否有通路
bool check(int mid)
{
	for(int i = 1;i <= n;i++) f[i] = i;// 初始化
	//m 条路
	for(int i = 1;i <= m;i++)
	{
		// 代价大于 mid 的通路无效
		if(cost[i] > mid) continue;
		int fx = find(x[i]),fy = find(y[i]);
		// 将 x[i] 和 y[i] 合并到同一个集合
		if(fx != fy)
		{
			f[fx] = fy;
		}
	}

	// 如果 s 到 t 有通路
	if(find(s) == find(t)) return true;
	else return false;
}
```

```cpp
int main()
{
	cin>>n>>m>>s>>t;
	// 读入 m 条路的拥挤度
	for(int i = 1;i <= m;i++)
	{
		cin>>x[i]>>y[i]>>cost[i];
		l = min(l,cost[i]);
		r = max(r,cost[i]);
	}

	// 二分拥挤度，找左边界
	int mid;
	while(l <= r)
	{
		mid = (l + r) >> 1;
		// 如果在拥挤度为 mid 的情况下有通路，尝试降低 mid
		if(check(mid))
		{
			r = mid - 1;
		}
		else
		{
			l = mid + 1;
		}
	}

	// 输出左边界
	cout<<l;
	return 0;
}
```

```
#1 可以对道路的 w 值先排序，再来二分，提高二分的效率
#include<bits/stdc++.h>//3-1923-2    躲避拥堵的最佳路线      zhangjin
using namespace std;
int a[10001];
int n,m,s,t;
struct Road
{
	int u,v,w;
}x[20001];
bool cmp(Road i,Road j)
{
	return i.w<j.w;
}
void init()
{
	for(int i=1;i<=n;i++)
		a[i] = i;
}
int find(int k)
{
	if(a[k]==k)
		return k;
	else
		return a[k]=find(a[k]);
}
bool check(int mid)
{
	init();
	for(int i=1;i<=n&&x[i].w<=mid;i++)
	{
		if(find(x[i].u!=x[i].v))
			a[find(x[i].u)] = find(x[i].v);
	}
	return find(s)==find(t);
}
```

```
int main()
{
	scanf("%d%d%d%d",&n,&m,&s,&t);



	int i;
	for(i=1;i<=m;i++)
		scanf("%d%d%d",&x[i].u,&x[i].v,&x[i].w);

	sort(x+1,x+1+m,cmp);

	int low,high,mid;
	low=x[1].w,high=x[m].w;

	while(low+1<high)
	{
		mid = (low+high)/2;
		if(check(mid))
			high = mid;
		else
			low = mid;
	}

	if(check(low))
		printf("%d",low);
	else
		printf("%d",high);

	return 0;
}
```

```
#include<bits/stdc++.h>//3-1923-3     躲避拥堵的最佳路线      w2016010182
using namespace std;
int m,n,s,t,l=INT&#95;MAX,r=INT&#95;MIN,mid,u[20015],v[20015],w[20015],ans;
int fa[20015];
int find(int x)
{
      if(fa[x]==x) return x;
      else return fa[x]=find(fa[x]);
}
void un(int x,int y)
{   int  fx=find(x);
      int  fy=find(y);
      if(fx!=fy)
      {
            fa[fx]=fy;
      }
}
bool check(int x)
{
      for(int i=1;i<=n;i++) fa[i]=i;
      for(int i=1;i<=m;i++)
      {
            if(w[i]<=mid)
            {
                  un(u[i],v[i]);
            }
            else continue;
      }
      if(find(fa[s])==find(fa[t])) return true;
      else return false;
}
```

```
int main()
{
	scanf("%d%d%d%d",&n,&m,&s,&t);
	for(int i=1;i<=m;i++)
	{
		scanf("%d%d%d",&u[i],&v[i],&w[i]);
		l=min(l,w[i]);
		r=max(r,w[i]);
	}
	while(l<=r)
	{
		mid=(l+r)>>1;
		if(check(mid))
		{
			r=mid-1;
			ans=mid;
		}
		else l=mid+1;
	}
	cout<<ans;
}
```

题目的两种关系已经说得很明确了，我们将朋友关系的两个人合并。
对于是敌人关系的两个人，由于敌人的敌人是我的朋友，
所以我们可以建立一个自己虚拟的敌人
（比如：认为 x 和 x+n 是敌人，那么如果 x 和 y 是敌人的话，y 和 x+n 就是朋友）
再与对方形成朋友关系。

```
#include<bits/stdc++.h>//4-1925-1    团队数量    javacn
using namespace std;
#define N 1010
//f：存储每个节点的父节点
int n,m,ans,f[N],e[N];
char c;

// 压缩路径求解 x 的根
int find(int x)
{
	//x 的父直接指向 x 的根
	return x == f[x]?x:f[x]=find(f[x]);
}

// 将 x 和 y 合并到同一个集合
void merge(int x,int y)
{
	int fx = find(x);
	int fy = find(y);
	// 如果 x 和 y 不在一个集合
	if(fx != fy)
	{
		f[fx] = fy;
	}
}
```

```
int main()
{
	cin>>n>>m;
	for(int i = 1;i <= 2 * n;i++) f[i] = i;
	int x,y;
	//m 个关系
	for(int i = 1;i <= m;i++)
	{
		cin>>c>>x>>y;
		// 朋友关系直接合并
		if(c == 'F') merge(x,y);
		else if(c ==  ‘E’ )
		{
			// 敌人
			merge(x+n,y);
			merge(y+n,x);
		}
	}

	for(int i = 1;i <= n;i++)
	{
		if(f[i] == i) ans++;
	}

	cout<<ans;
	return 0;
}
```

```
#include<bits/stdc++.h>//4-1925-2    团队数量   w2016010182
using namespace std;
int m,n,s,t,ans;
char c;
int fa[50015];
int find(int x)
{
      if(fa[x]==x) return x;
      else return fa[x]=find(fa[x]);
}
void un(int x,int y)
{
      int fx=find(x);
      int fy=find(y);
      if(fx!=fy)
      {
            fa[fx]=fy;// 把 y 的根节点的值     赋值给     x 这个集合的根节点     1-3
3- 3    5-5
      }
}
```

```
int main()
{
	scanf("%d%d",&n,&m);
	for(int  i=1;i<=2*n;i++)// 建立自己虚拟的敌人，x 与 x+n 为敌人    y 与 y+n 为敌人
	{
		fa[i]=i;
	}
	for(int i=1;i<=m;i++)
	{
		cin>>c>>s>>t;
		if(c=='F')
		{
			un(s,t);
		}
		else if(c=='E')
		{
			un(s+n,t);     //x 与 y+n 为朋友   y 与 x+n 为朋友
			un(t+n,s);
		}
	}
	for(int i=1;i<=n;i++)
	{
		if(fa[i]==i) ans++;
	}
	cout<<ans;
}
```

要尽可能大的将危害大的罪犯放到两个集合，那么将怨气值降序排序，逐个讨论每对罪犯，如果发现有两对罪犯在一个集合，那么计算结束，输出这个怨气值。如果不在一个集合，那么该罪犯要和对方的敌人在一个集合，对方的敌人要和该罪犯在一个集合。

```
#include<bits/stdc++.h>//5-1930-1    关押罪犯    javacn
using namespace std;
/*
1. 将怨气值大的罪犯，尽可能拆到不同的监狱
2. 维护种类并查集
将每个人和其敌人的敌人合并到一个集合
x,y+n 合并，y,x+n 合并
3. 直到出现第一组两个人已经在一个监狱的情况
此时的怨气值就是最大的怨气值
*/
int f[40010];
int n,m;
//x 和 y 是敌人，怨气值为 v
struct node{
	int x,y,v;
};
node a[100010];
// 按怨气值降序
bool cmp(node n1,node n2)
{
	return n1.v > n2.v;
}
// 查询
int find(int x)
{
	return x==f[x]?x:f[x]=find(f[x]);
}
// 合并
```

```
//合并
void merge(int x,int y)
{
	int fx = find(x);
	int fy = find(y);
	if(fx != fy)	{	f[fx] = fy; }
}
int main()
{
	cin>>n>>m;
	//读入关系
	for(int i = 1;i <= m;i++)	{	cin>>a[i].x>>a[i].y>>a[i].v;	}
	//将怨气值降序排序，从最大的开始拆分
	sort(a+1,a+m+1,cmp);
	//初始化
	for(int i = 1;i <= n * 2;i++)	{	f[i] = i;	}
	//处理关系
	for(int i = 1;i <= m;i++)
	{
		//如果发现 2 个人已经在一座监狱，必定有冲突
		if(find(a[i].x) == find(a[i].y))
		{
			cout<<a[i].v;
			return 0;
		}
		else
		{
			//将每个人和其假想的好友合并
			merge(a[i].x+n,a[i].y);
			merge(a[i].y+n,a[i].x);
		}
	}
	cout<<0;
	return 0;
}
```

/*并查集能维护连通性、传递性，通俗地说，亲戚的亲戚是亲戚。
然而当我们需要维护一些对立关系，比如 敌人的敌人是朋友 时，正常的并查集就很难满足我们的需求。这时，种类并查集就诞生了。
常见的做法是将原并查集扩大一倍规模，并划分为两个种类。
在同个种类的并查集中合并，和原始的并查集没什么区别，仍然表达他们是朋友这个含义。
考虑在不同种类的并查集中合并的意义，其实就表达 他们是敌人 这个含义了。
按照并查集美妙的 传递性，我们就能具体知道某两个元素到底是 敌人 还是 朋友 了。
至于某个元素到底属于两个种类中的哪一个，由于我们不清楚，因此两个种类我们都试试。
本题只需要最大的影响力，所以可以根据影响力大小将每一对冲突关系降序排序，就可以保证不满足条件时的影响力最小
首先考虑如何用并查集解决问题：我们可以设置数组[1--2n]，其中a[0-n]表示处于一个监狱，a[n-2n]表示处于另一个监狱
从头扫描每一对关系，将a和b+n，a+n和b合并（即两人关在不同监狱）
合并之前检查c和d，c+n和d+n是否在同一集合，如果是，则最大影响力即为此对关系的影响力

```
#include<bits/stdc++.h>//5-1930-2    关押罪犯    w2016010182
using namespace std;
struct pep
{
	int num1;
	int num2;
	int t;
}dl[100005];
int a[40005];
int Find(int x)
{
	if(a[x]!=x)	{	a[x]=Find(a[x]);	}
	return a[x];
}
void Un(int x,int y)
{
	int fx=Find(x);
	int fy=Find(y);
	if(fx!=fy)	{	a[fx]=fy;	}
}
```

```
bool cmp(pep a,pep b)
{
	return a.t>b.t;
}
int main()
{
	int n,m;
	scanf("%d%d",&n,&m);
	for(int i=1;i<=n*2;++i)
	{
		a[i]=i;
	}
	for(int i=1;i<=m;++i)
	{
		scanf("%d%d%d",&dl[i].num1,&dl[i].num2,&dl[i].t);
	}
	sort(dl+1,dl+1+m,cmp);
	for(int i=1;i<=m;++i)
	{
		if(Find(dl[i].num1)!=Find(dl[i].num2))
		{
			Un(dl[i].num1 +n,dl[i].num2);
			Un(dl[i].num2 +n,dl[i].num1);
		}
		else
		{
			printf("%d",dl[i].t);
			return 0;
		}
	}
	printf("0");
	return 0;
}
```

求出 a 公司有多少人和 1 号在一个集合，再求出 b 公司有多少人和 -1 号在一个集合，最后求两者的较小值。

由于 b 公司的人编号是负数，我们可以将 f 数组放长，用下标 n+1~n+m 来存储这 m 个人的关系。

```
#include <bits/stdc++.h>//6-1932    舞伴      javacn
using namespace std;

int f[20010];// 描述集合关系

// 查：查询元素的根
int find(int x)
{
      return f[x]==x?x:f[x]=find(f[x]);
}

// 并：合并元素 xy
void merge(int x,int y)
{
      int fx = find(x);
      int fy = find(y);
      if(fx != fy)
      {
            f[fx] = fy;
      }
}
```

```
int main()
{
	int n,m,p,q;
	cin>>n>>m>>p>>q;
	int ra = 0,rb = 0;//ab 公司每个集合中和小明、小明认识的人的数量
	// 初始化
	for(int i = 1;i <= n + m;i++) f[i] = i;

	// 读入 p 个关系
	int x,y;
	for(int i = 1;i <= p;i++)
	{
		cin>>x>>y;
		merge(x,y);
	}

	// 读入 q 个关系
	for(int i = 1;i <= q;i++)
	{
		cin>>x>>y;
		x = x * -1;
		y = y * -1;
		merge(x+n,y+n);
	}
	// 求和 1 号以及 n+1 号是朋友的人的数量
	for(int i = 1;i <= n;i++)
	{
		if(find(i) == find(1)) ra++;
	}
	for(int i = n+1;i <= n + m;i++)
	{
		if(find(i) == find(n+1)) rb++;
	}
	cout<<min(ra,rb);
}
```

首先把实力相同的全都用并查集并起来，把在每个集合的人数记录下来。

然后就是背包问题了，注意，要使原来的 m 尽可能接近的选出学霸的数目，可能选出多于 m 个人，所以把背包容量扩大为 2m。

```
#include <bits/stdc++.h>//7-1933     比赛组队     javacn
using namespace std;

/*
思路：
1. 用数组 p 存储每个集合的人数，默认初始值都为 1
2. 使用并查集将有关联的人合并到一个集合，同时合并 p 数组
3. 使用 01 背包求背包容量在 2*m 的范围内，每个单位的背包容量，能得到的最大值
4. 在 0~2*m 的范围内，求一个离 m 最近的结果
*/
int n,m,k;
int fa[20010],dp[40010],p[20010];

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

```
int main()
{
	cin>>n>>m>>k;
	// 初始化 fa 数组
	for(int i = 1;i <= n;i++)
	{
		fa[i] = i;
		p[i] = 1;// 每个集合默认有 1 个人
	}
	int x,y;
	// 读入 k 对关系
	for(int i = 1; i <= k; i++)
	{
		cin>>x>>y;
		int fx = find(x);
		int fy = find(y);
		if(fx != fy)
		{
			fa[fx] = fy;
			p[fy] += p[fx];// 人数合并
			p[fx] = 0; //fx 集合对应的人数清零
		}
	}

	//01 背包求解
	for(int i = 1;i <= n;i++)
	{
		// 过滤被合并的数据，否则时间会超限
		if(p[i] == 0) continue;
		for(int j = m * 2;j >= p[i];j--)
		{
			dp[j] = max(dp[j],dp[j-p[i]] + p[i]);
		}
	}
```

```
        // 求离 m 最近的结果
        int mi = INT_MAX;// 代表离 m 最近的差值
        int ans;// 代表题目要求的离 m 最近的人数
        for(int i = 0;i <= 2 * m;i++)
        {
                if(abs(dp[i] - m) < mi)
                {
                        mi = abs(dp[i] - m);
                        ans = dp[i];
                }
        }

        cout<<ans;
        return 0;
}
```

思路：把所有质因数 >=p 的数求出来合并，再并查集求解。

求质因数 >=p 的数的方法是：

先筛选出 p~b 之间所有的素数；

逐个看每个素数 *2、*3、*4……的结果是否在 a~b 的范围内，如果在，那么这些数就是符合条件的，应当在一个集合范围内的数。

```
#include<bits/stdc++.h>//8-1924    集合    javacn
#define N 100010
using namespace std;
int a,b,x,ans,cnt;
int fa[N],notp[N],p[N];
// 路径压缩的查找算法，找出 x 的根
int find(int x)
{
	//x 的父直接指向 x 的根
	return x == fa[x]?x:fa[x] = find(fa[x]);
}
// 集合合并
void merge(int x,int y)
{
	int fx = find(x);
	int fy = find(y);
	if(fx != fy) fa[fx] = fy;
}
// 素数筛选
```

```
// 素数筛选
void getprime()
{
	notp[1] = 1;//1 不是素数
	// 筛选 2~b 范围内的非素数
	for(int i = 2;i <= b;i++)
	{
		if(notp[i]) continue; // 如果 i 已经是非素数了
		for(int j = 2*i;j <= b;j=j+i)
		{
			notp[j] = 1;
		}
	}

	// 将大于 x~b 之间的素数找出来
	for(int i = x;i <= b;i++)
	{
		if(!notp[i])
		{
			cnt++;
			p[cnt] = i;
		}
	}
}
```

```
int main()
{
	cin>>a>>b>>x;
	// 并查集初始化
	for(int i = a;i <= b;i++) fa[i] = i;
	getprime();// 筛选素数

	// 遍历 x~b 之间所有的素数
	for(int i = 1;i <= cnt;i++)
	{
		for(int j = 2;j * p[i] <= b;j++)
		{
			// 两个数有 >=x 的质因子
			if(j * p[i] >= a && (j-1) * p[i] >= a)
			{
				merge(j*p[i],(j-1)*p[i]);
			}
		}
	}

	// 看有几个集合
	for(int i = a;i <= b;i++)
	{
		if(fa[i]==i) ans++;
	}
	cout<<ans;
	return 0;
}
```

典型的 01 背包，但要求同一组的物品都要购买。我们可以采用并查集将同一组有关系的礼品的价值、价格汇总到该集合的根节点上，这样就保证了一个集合中的礼品都购买的情况。

```
#include<bits/stdc++.h>//9-1928    采购礼品    javacn
using namespace std;

int f[10100];// 存储物品之间的关系
int q[10100],v[10100];// 价钱、价值
int dp[10100];// 以拥有的钱来定义背包容量

// 查：查询元素的根
int find(int x)
{
	return f[x]==x?x:f[x]=find(f[x]);
}

// 并：合并元素 xy
void merge(int x,int y)
{
	int fx = find(x);
	int fy = find(y);
	if(fx != fy)
	{
		f[fx] = fy;
	}
}
```

```
int main()
{
	int n,m,w;
	cin>>n>>m>>w;
	//n 个物品的价钱和价值
	for(int i = 1;i <= n;i++)
	{
		cin>>q[i]>>v[i];
		// 并查集初始化
		f[i] = i;
	}

	//m 个物品的关系
	int x,y;
	for(int i = 1;i <= m;i++)
	{
		cin>>x>>y;
		merge(x,y);
	}

	// 将有关系的物品合并到这组物品的根上
	for(int i = 1;i <= n;i++)
	{
		// 该物品不是根，则将价钱和价值都合并到根上
		if(f[i] != i)
		{
			q[find(i)]+=q[i];
			v[find(i)]+=v[i];
			// 将该组物品的价钱和价值清零
			q[i]=0;
			v[i]=0;
		}
	}

	//01 背包计算结果
```

```
	//01 背包计算结果
	for(int i = 1;i <= n;i++)
	{
		// 从背包容量（有多少钱）~ 该物品的价钱降序
		for(int j = w;j >= q[i];j--)
		{
			dp[j] = max(dp[j],dp[j-q[i]]+v[i]);
		}
	}

	cout<<dp[w];

	return 0;
}
```

```cpp
#include <bits/stdc++.h>//10-2030      信息传递 [NOIP2015 提高组]     wsjszvip
using namespace std;
const int N = 200100;
int fa[N], d[N], n, mi;

// 带权并查集更新数据
int find(int x)
{
	if (x == fa[x]) return x;
	else
	{
		int t = fa[x];// 记录父元素
		fa[x] = find(fa[x]);// 查找根，路径压缩
		d[x] = d[x] + d[t];// 更新路径长度
		return fa[x];
	}
}

void merge(int x, int y)
{
	int fx = find(x);
	int fy = find(y);
	// 如果不相连，则连接两个点并更新父节点和路径长度
	if (fx != fy)
	{
		fa[fx] = fy;
		d[x] = d[y] + 1;
	}
	else
	{
		// 已经连接，更新环的最小长度
		mi = min(mi, d[x] + d[y] + 1);
	}
}
```

```
int main()
{
	cin >> n;
	// 初始化，同时 d[i] 默认为 0
	for (int i = 1; i <= n; i++) fa[i] = i;

	mi = 0x7777777;
	int t;
	for (int i = 1; i <= n; i++)
	{
		cin >> t;
		merge(i, t);
	}

	cout << mi;
	return 0;
}
```

2、最小生成树问题

最小生成树：克鲁斯卡尔（Kruskal）算法，基本思想：按照权值从小到大的顺序选择 n-1 条边，并保证这 n-1 条边不构成回路。

具体做法：首先构造一个只含 n 个顶点的森林，然后依权值从小到大从连通网中选择边加入到森林中，并使森林中不产生回路，直至森林变成一棵树为止。

```
#include<bits/stdc++.h>//1-1929-1    最短网络 Agri-Net（USACO3.1）    javacn
using namespace std;
struct node
{
      int x,y,w;
} a[10010];// 农场之间的距离

// 农场之间的集合关系
int f[110];

bool cmp(node n1,node n2)
{ // 结构体排序
      return n1.w<n2.w;
}

int find(int x)
{
      // 并查集说白了就是找父结点的过程，同一个父节点即同一个区间
      if(x==f[x]) return x;
      else return f[x]=find(f[x]);
}
```

```
int main()
{
	int n,k,m=0;
	cin>>n;
	for(int i=1; i<=n; i++)
	{
		f[i]=i;// 初始化
		// 每行 n 个数
		for(int j=1; j<=n; j++)
		{
			cin>>k;
			// 看一半就行，看右上角的值
			if(j>i)
			{
				m++;
				a[m].x=i;
				a[m].y=j;
				a[m].w=k;
			}
		}
	}

	// 按长度升序
```

```
    //按长度升序
    sort(a+1,a+m+1,cmp);//排序

    int ans=0,c=0;
    for(int i=1; i<=m; i++)
    {
        //如果不在一个集合
        if(find(a[i].x)!=find(a[i].y))
        {
            ans+=a[i].w;
            f[find(a[i].x)]=a[i].y;
            //合并两个节点
            c++;
            if(c == n - 1) break;
        }
    }

    cout<<ans;
    return 0;
}
```

// 针对 1：// 首先这道题和其他题一样，你所读入的矩阵中的每一个数字都代表着相邻两点之间的边的边权，而这个点所处的位置即可以用 (i,j) 来表示，然后再进行操作即可
// 针对 2：// 我们会发现这个 N*N 的矩阵是关于对角线对称的，所以我们只需要读上面一部分或者下面一部分即可（要读上面一部分读入时则特判 j>i，相反则特判 j<i...
// 针对 3：// 这 Kruskal 算法是通过并查集，按照边的权重顺序（从小到大）将边加入生成树中，但是若加入该边会与生成树形成环则不加入该边，选其次。直到树中含有 n - 1 条边为止。时间复杂度：O（E log E）（E 为边数）：n 个点，n-1 条边，所以在合并的时候我们定义一个 k，存储合并后边的条数，每合并一次则加一，最后当 k == n-1 时跳出即可

```
#include <bits/stdc++.h>//1-1929-2    最短网络 Agri-Net （USACO3.1） w2016010182
using namespace std;
int n, m, v, k, ans, fa[10000001];
struct node// 定义结构体存图
{
	int x, y, z;//z 表示 x 连 y 的权值
}stu[100001];
int find(int x)// 并查集
{
	if(x != fa[x])
	{
		fa[x] = find(fa[x]);
	}
	return fa[x];
}// 查找
void unity(int x, int y)
{
	int r1 = find(x);
	int r2 = find(y);
	fa[r1] = r2;
}// 合并

bool cmp(node a, node b)// 从小到大结构体排序
{
	return a.z < b.z;
}
```

```
int main()
{
	scanf("%d", &n);
	for(int i = 1; i <= n; i++)		{	fa[i] = i;// 并查集初始化	}
	for(int i = 1; i <= n; i++)
	{
		for(int j = 1; j <= n; j++)
		{
			scanf("%d", &v);
			if(j > i)// 邻接矩阵上下对称，存一半就行了
			{
				m++;
				stu[m].x = i;
				stu[m].y = j;
				stu[m].z = v;
			}
		}
	}
	sort(stu + 1, stu + m + 1, cmp);// 排序
	for(int i = 1; i <= m; i++)
	{
		if(find(stu[i].x) != find(stu[i].y))
		{
			ans += stu[i].z;// 加上最小生成树中边的权值
			unity(stu[i].x, stu[i].y);// 连接起来
			k++;// 记录边数
			if(k == n - 1)//n - 1 条边就行了
			{
				printf("%d", ans);
				return 0;
			}
		}
	}
	return 0;
}
```

一个不存在回路的无向联通图叫做树

生成树：生成树是连通图的极小联通子图。（加边会出现回路，减边变为非连通图）

最小生成树： 生成树的各边权值总和最小的树（最小代价树）

最小生成树一般有两种算法：prim 和 kruskal。prim 一般用于稠密图，kruskal 用于稀疏图。

这里介绍一下 kruskal。

kruskal 算法的基本思想是：始终选择当前可用的最小权值边（贪心

因此很容易想到题目所求的最大边就是最后一条边（每次加边权取 max 也可以）。

那么怎么判断当前选择的边是否可用呢？

并查集：意思就是说，开始的每个点都是独立的集合，加边的时候就合并起来，如果已经是联通的就跳过。

```
#include<bits/stdc++.h>//2-1931   Out of Hay S[USACO05MAR]   javacn
using namespace std;
// 节点
struct node
{
    int x,y,len;
}a[10010];
// 按边长度降序排序
bool cmp(node n1,node n2)
{
      return n1.len < n2.len;
}
//f: 存储农场之间的集合关系
int f[2100],n,m;
int find(int x)// 查
{
      return x==f[x]?x:f[x]=find(f[x]);
}
void merge(int x,int y)// 并
{
      int fx = find(x);
      int fy = find(y);
      if(fx != fy) f[fx] = fy;
}
```

```
int main()
{
	cin>>n>>m;
	for(int i = 1;i <= m;i++)
	{
	  cin>>a[i].x>>a[i].y>>a[i].len;
	}

    // 按边长升序排序
	sort(a+1,a+1+m,cmp);

    // 初始化
	for(int i = 1;i <= n;i++) f[i] = i;

    // 讨论合并：克努斯卡尔算法
	int c = 0;// 已经使用的边的数量
	for(int i = 1;i <= m;i++)
	{
	  // 如果 x 和 y 不在一个集合
		if(find(a[i].x)!=find(a[i].y))
		{
			// 合并
			merge(a[i].x,a[i].y);
			c++;
			// 建立最小生成树
			if(c == n - 1)
			{
				cout<<a[i].len;
				 break;
			}
		}
	}

	return 0;
}
```

n 个村庄，至少需要 n-1 条路，但要注意，如果两个村庄之间本来就有通路，再修路，那么就不算在这 n-1 条路中。比如：5 个村庄，1 2、1 3 之间有路，再在 2 3 之间修路，不能算 3 条，因为 1 3 本来已经有通路了。

```
#include<bits/stdc++.h>//3-1927     最短的通路时间     javacn
using namespace std;

// 存储一条路连接的两端的编号 以及 修好的时间
struct node
{
	int x,y,t;
}a[100100];

int f[1100];// 存储村庄之间是否有路的关系描述

// 查：查询元素的根
int find(int x)
{
	return f[x]==x?x:f[x]=find(f[x]);
}

// 排序辅助函数：按时间升序
bool cmp(node n1,node n2)
{
	return n1.t < n2.t;
}
```

```cpp
int main()
{
	int n,m,i;
	cin>>n>>m;
	for(int i = 1;i <= m;i++)
	{
		cin>>a[i].x>>a[i].y>>a[i].t;
	}
	// 所有数据，按时间升序
	sort(a+1,a+1+m,cmp);
	for(int i = 1;i <= n;i++)	// 初始化
	{
		f[i] = i;
	}
	// 合并（修路）
	int fx,fy,c = 0;
	for(int i = 1;i <= m;i++)
	{
		fx = find(a[i].x);
		fy = find(a[i].y);
		// 如果两者之间没有路
		if(fx != fy)
		{
			f[fx] = fy;
			c++;
		}
		if(c == n - 1)
		{
			cout<<a[i].t;
			return 0;
		}
	}
	cout<<-1;
	return 0;
}
```

对边长升序排序，先将已修的边中不形成回路的 merge，再看还需要修多少能修满 n-1 条边。

```
#include <bits/stdc++.h>//4-2070  道路规划   javacn
using namespace std;
struct node
{
	int x,y,len;
};
int f[110];// 父元素
node a[6010];// 边
int t,k = 0;
int n,q;
// 查询
int find(int x)
{
	return x == f[x]?x:f[x] = find(f[x]);
}
// 合并
void merge(int x,int y)
{
	int fx = find(x);
	int fy = find(y);
	if(fx != fy)
	{
		f[fx] = fy;
	}
}

// 排序
bool cmp(node n1,node n2)
{
	return n1.len < n2.len;
}
```

```
int main()
{
	cin>>n;
	// 存储不重复的边（邻接矩阵的一半）
	for(int i = 1; i <= n; i++)
	{
		for(int j = 1; j <= n; j++)
		{
			cin>>t;
			if(i > j)
			{
				k++;
				a[k].x = i;
				a[k].y = j;
				a[k].len = t;
			}
		}
	}

	// 并查集初始化
	for(int i = 1; i <= n; i++)
	{
		f[i] = i;
	}

	sort(a+1,a+k+1,cmp);
	cin>>q;
	int x,y;
	int c = 0;
	// 已有的边修起来，注意形成回路的不要修
	// 修了会导致 c 的统计错误
```

```
        for(int i = 1; i <= q; i++)
        {
                cin>>x>>y;
                if(find(x) != find(y))
                {
                        c++;
                        merge(x,y);
                }
        }

        // 如果已经够了 n-1 条边，结束
        if(c == n - 1)
        {
                cout<<0;
                return 0;
        }

        // 继续修边，直到修满 n-1 条边
        int s = 0;
        for(int i = 1; i <= k; i++)
        {
                if(find(a[i].x) != find(a[i].y))
                {
                        merge(a[i].x,a[i].y);
                        c++;
                        s = s + a[i].len;
                }

                if(c == n - 1)
                {
                        cout<<s;
                        break;
                }
        }
}
```

n 个村庄在不构成回路的情况下，需要 n-k 条路，才能划分为 k 个区。

比如：5 个村庄，修建 1 条路，可以划分出 4 个区，修建 2 条路可以划分出 3 个区，修建 3 条路可以划分 2 个区，修建 4 条路可以划分 1 个区。（注意：不能构成回路）

因此：本题使用并查集求解最小生成树可求解。

```
#include <bits/stdc++.h>//6-2090    片区划分    javacn
using namespace std;

/*
n 个点，m 个已知关系，要组成 k 个集合，最小代价是多少
*/
struct node
{
	int x,y,len;
};

node a[10010];
int f[1010];
int n,m,k;

bool cmp(node n1,node n2)
{
	return n1.len < n2.len;
}

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

```cpp
int main()
{
	cin>>n>>m>>k;
	for(int i = 1;i <= m;i++)
	{
		cin>>a[i].x>>a[i].y>>a[i].len;
	}
	sort(a+1,a+1+m,cmp);
	for(int i = 1;i <= n;i++)
	{
		f[i] = i;
	}

	//修路
	int s = 0,c = 0;
	for(int i = 1;i <= m;i++)
	{
		int fx = find(a[i].x);
		int fy = find(a[i].y);
		if(fx != fy)
		{
		f[fx] = fy;
		c++;
		s = s + a[i].len;
		}

		if(c == n - k)
		{
			cout<<s;
			return 0;
		}
	}
	cout<<"No Answer";
	return 0;
}
```