实验舱
青少年编程
走近科学 走进名校

蛟龙五班

并查集、最小生成树、拓扑排序

Mas

并查集

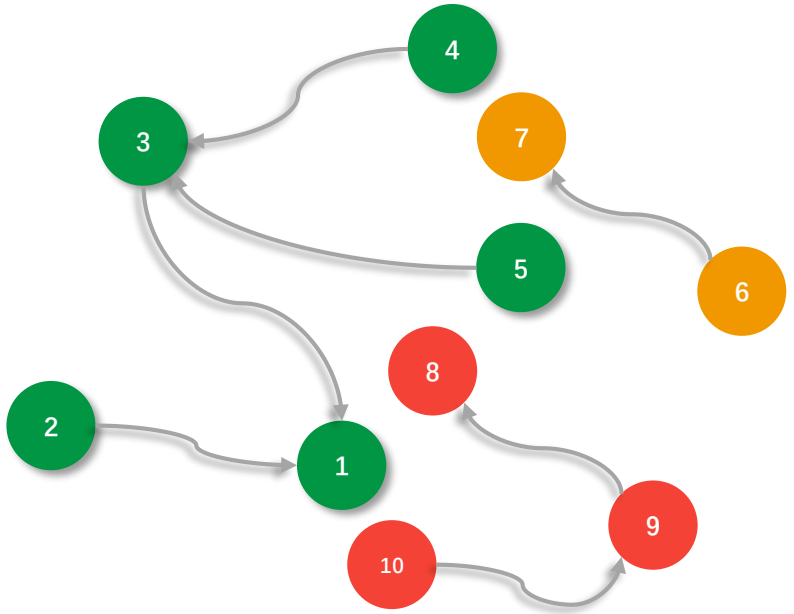
并查集(Disjoint-Set)是一种可以动态维护若干个不重叠的集合，并支持合并与查询两种操作的一种数据结构。

合并（Union）：把两个不相交的集合合并为一个集合。

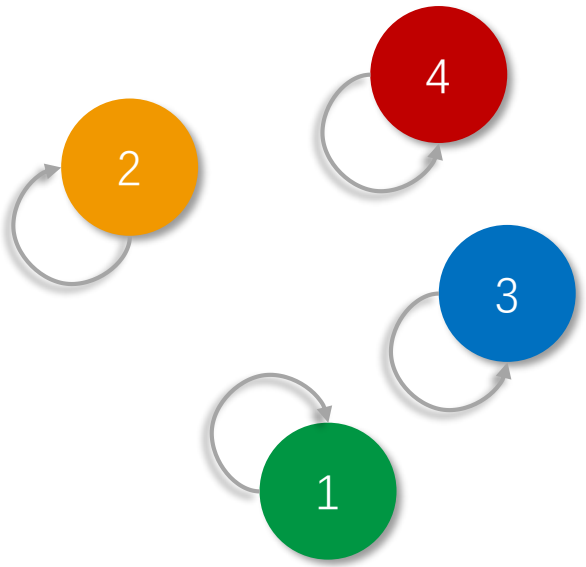
查询（Find）：查询两个元素是否在同一个集合中。

并查集的重要思想在于，**用集合中的一个元素代表集合**。

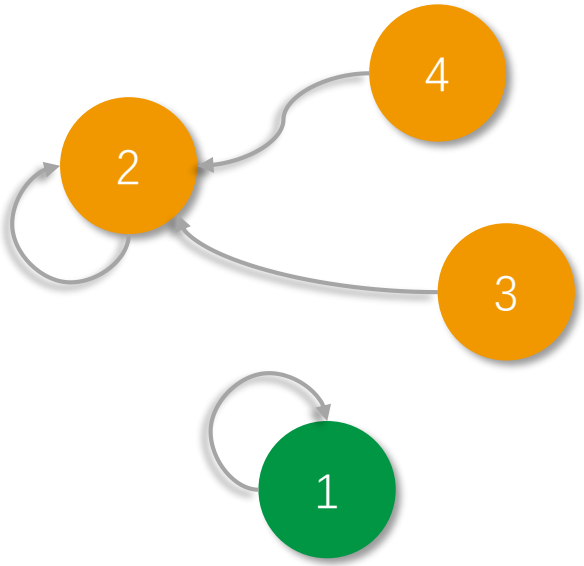
并查集可以用于判断两个元素是否存在直接或间接的联系、连通块计数等问题



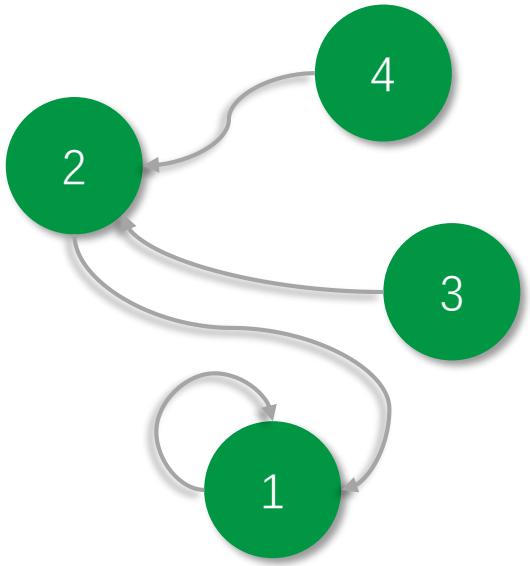
并查集



初始时，每个元素都是一个集合(自己是自己的祖先)father[i] = i



如果a,b是一个集合,将a的祖先设为b的祖先即可



当需要查询a,b是否在同一个集合时

只需要向上找到它们的祖先编号比较

并查集

```
int f[100001], n, m, a, b, q;

void init(int n)
{
    for (int i = 1; i <= n; i++)
        f[i] = i;
}
int find(int x)
{
    if (f[x] == x)
        return x;
    return find(f[x]);
}
int merge(int a, int b)
{
    f[find(a)] = find(b);
}

bool judge(int a, int b)
{
    return find(a) == find(b);
}
```

每次合并都需要查找该集合的代表

每次查找都需要递归找到最顶层的祖先

是否可以更高效?

#2160 并查集

题目描述

一共有 n 个数，编号是 $1 \sim n$，最开始每个数各自在一个集合中。

现在要进行 m 个操作，操作共有两种：

`M a b`，将编号为 a 和 b 的两个数所在的集合合并，如果两个数已经在同一个集合中，则忽略这个操作；
`Q a b`，询问编号为 a 和 b 的两个数是否在同一个集合中；

输入格式

第一行输入整数 n 和 m。

接下来 m 行，每行包含一个操作指令，指令为 `M a b` 或 `Q a b` 中的一种。

输出格式

对于每个询问指令 `Q a b`，都要输出一个结果，如果 a 和 b 在同一集合内，则输出 `Yes`，否则输出 `No`。

每个结果占一行。

输入样例：

```
4 5
M 1 2
M 3 4
Q 1 2
Q 1 3
Q 3 4
```

输出样例：

```
Yes
No
Yes
```

#2161 又是并查集

题目描述

给定一个包含 n 个点（编号为 $1 \sim n$）的无向图，初始时图中没有边。

现在要进行 m 个操作，操作共有三种：

`M a b`，在点 a 和点 b 之间连一条边，a 和 b 可能相等；
`Q a b`，询问点 a 和点 b 是否在同一个连通块中，a 和 b 可能相等；
`C a`，询问点 a 所在连通块中点的数量；

输入格式

第一行输入整数 n 和 m。

接下来 m 行，每行包含一个操作指令，指令为 `C a b`，`Q1 a b` 或 `Q2 a` 中的一种。

输出格式

对于每个询问指令 `Q1 a b`，如果 a 和 b 在同一个连通块中，则输出 `Yes`，否则输出 `No`。

对于每个询问指令 `Q2 a`，输出一个整数表示点a所在连通块中点的数量

每个结果占一行。

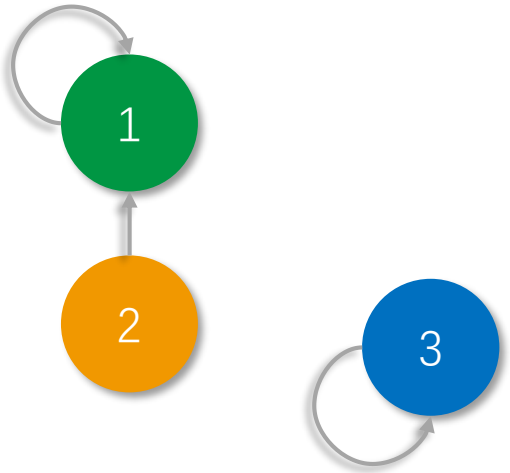
输入样例：

```
5 5
C 1 2
Q1 1 2
Q2 1
C 2 5
Q2 5
```

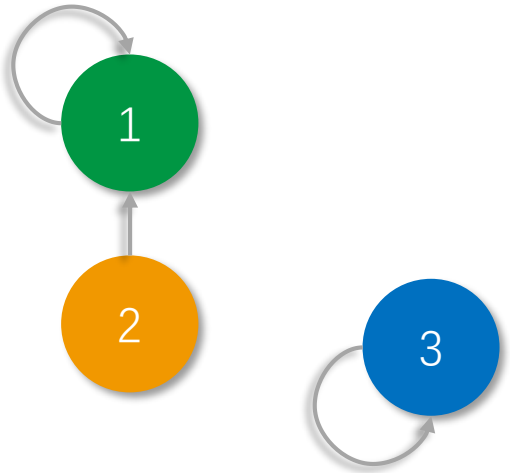
输出样例：

```
Yes
2
3
```

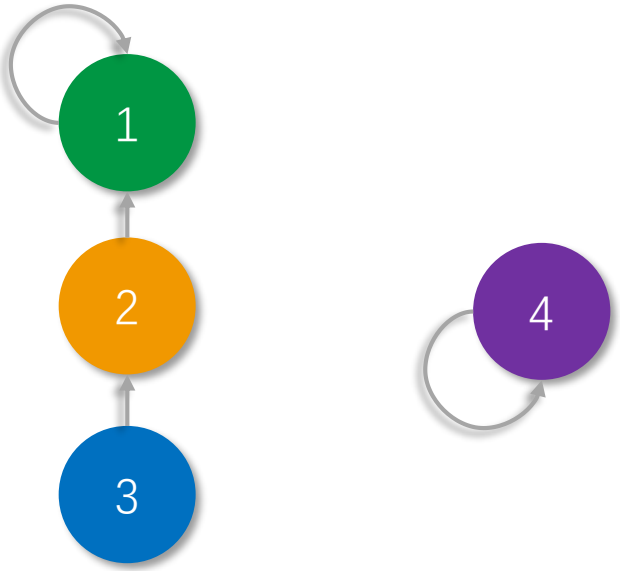
并查集-路径压缩



合并1,2时需要找到它们的祖先

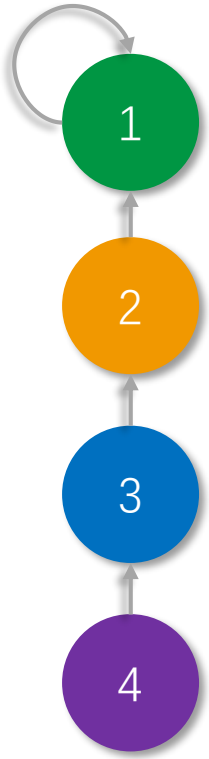


合并2,3时需要向上找1层找到2的祖先，找到3的祖先3，进行合并



合并3,4时需要向上找2层找到3的祖先，找到4的祖先4，进行合并

并查集-路径压缩

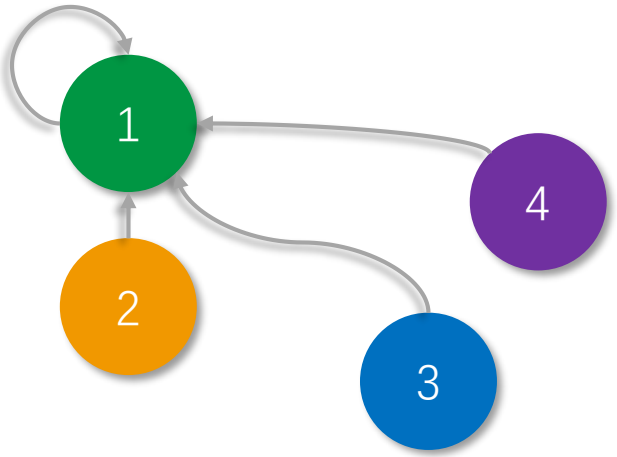


随着合并的元素越来越多，集合会形成一条很长的链，每次查询层数会越来越多

对于每个元素我们往往需要的是他的祖先编号，并不关心它的上层祖先编号

只要我们在查询的过程中，**把沿途的每个节点的父节点都设为根节点**。

下一次再查询时效率就会大大提高。



路径压缩后

#1274 亲戚1

【题目描述】

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，判断两个人是否是亲戚应该是可行的，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】

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 $1 \sim N$ 。下面有 M 行($1 \leq M \leq 1000000$)，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($1 \leq Q \leq 1000000$)，每行为 c_i, d_i ，表示询问 c_i 和 d_i 是否为亲戚。

【输出】

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 ”。

```
int f[100001], n, m, a, b, q;

void init(int n)
{
    for (int i = 1; i <= n; i++)
        f[i] = i;
}
int find(int x)
{
    return f[x] == x ? x : f[x] = find(f[x]);
}
int merge(int a, int b)
{
    f[find(a)] = find(b);
}

bool judge(int a, int b)
{
    return find(a) == find(b);
}
```

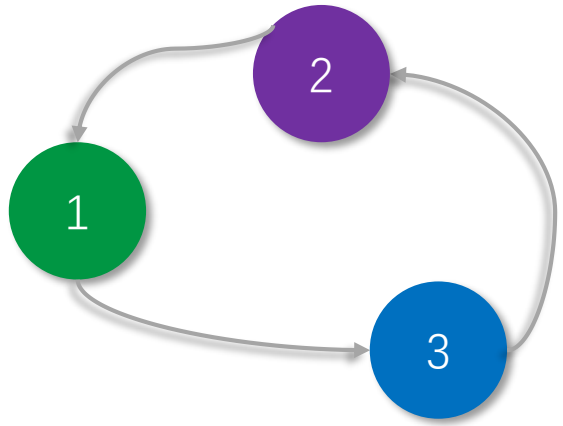
拓扑排序

在日常生活中，一项大的工程可以看作是由若干个子工程（这些子工程称为“活动”）组成的集合，这些子工程（活动）之间必定存在一些先后关系，即某些子工程（活动）必须在其它一些子工程（活动）完成之后才能开始，我们可以用有向图来形象地表示这些子工程（活动）之间的先后关系，子工程（活动）为顶点，子工程（活动）之间的先后关系为有向边，这种有向图称为“顶点活动网络”，又称“AOV网”。

在AOV网中，有向边代表子工程（活动）的先后关系，我们把一条有向边起点的活动成为终点活动的前驱活动，同理终点的活动称为起点活动的后继活动。

而只有当一个活动全部的前驱全部都完成之后，这个活动才能进行。

一个AOV网必定是一个有向无环图。



拓扑排序

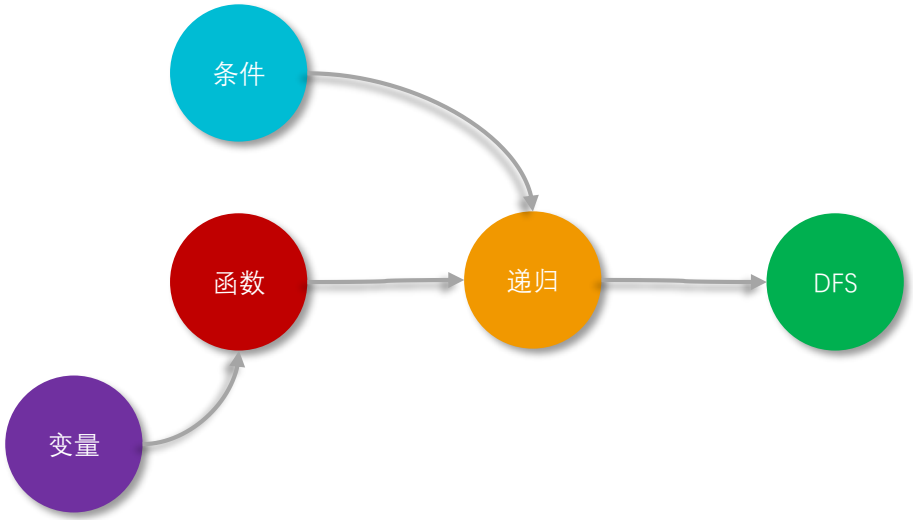
把AOV网中的所有活动排成一个序列， 使得每个活动的所有前驱活动都排在该活动的前面，这个过程称为“拓扑排序”，所得到的活动序列称为“拓扑序列”。

构造拓扑序列可以帮助我们合理安排一个工程的进度，由AOV网构造拓扑序列具有很高的实际应用价值。

构造拓扑序列的拓扑排序算法思想很简单：

- 选择一个入度为0的顶点并输出
- 然后从AOV网中删除此顶点及以此顶点为起点的所有关联边；

重复上述两步，直到不存在入度为0的顶点为止。



#1878 家谱树

【问题描述】

有个人的家族很大，辈分关系很混乱，请你帮整理一下这种关系。

给出每个人的孩子的信息。

输出一个序列，使得每个人的后辈都比那个人后列出。

【输入格式】

第 1 行一个整数 N（$1 \leq N \leq 100$），表示家族的人数。

接下来 N 行，第 I 行描述第 I 个人的儿子。

每行最后是0表示描述完毕。

【输出格式】

输出一个序列，使得每个人的后辈都比那个人后列出。

如果有多解输出任意一解。

最小生成树(MST)

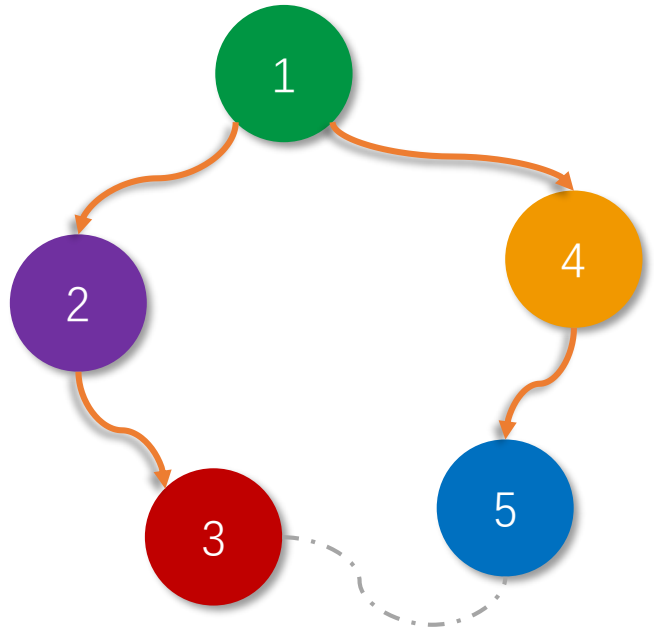
一个有N个点的图，边一定是大于等于N-1条的。

图的最小生成树，就是在这些边中选择N-1条出来，连接所有的N个点。

这N-1条边的边权之和是所有方案中最小的。

最小生成树用来解决什么问题?

就是用来解决如何用最小的“代价”用N-1条边连接N个点的问题。



Kruskal算法

Kruskal（克鲁斯卡尔）算法是一种巧妙利用并查集来求最小生成树的算法。

Kruskal算法将一个连通块当做一个集合。

首先将所有的边按从小到大顺序排序，并认为每一个点都是孤立的，分属于n个独立的集合。

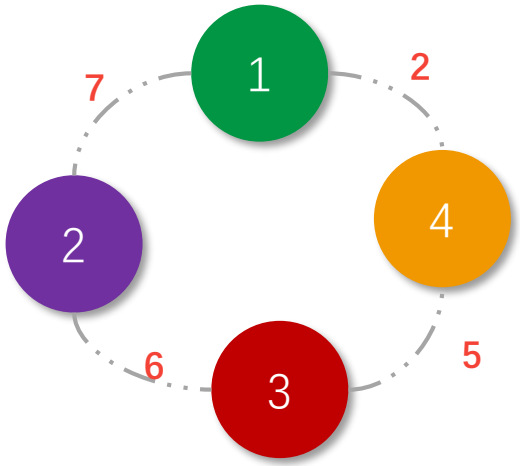
然后按顺序枚举每一条边

如果这条边连接着两个不同的集合，那么就把这条边加入最小生成树，这两个不同的集合就合并成了一个集合；

如果这条边连接的两个点属于同一集合，就跳过。

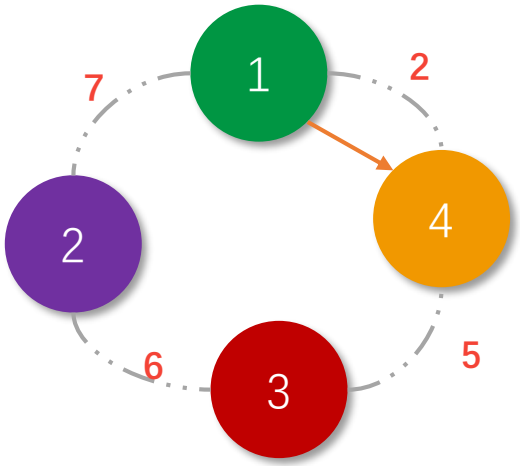
直到选取了n-1条边为止。

Kruskal算法



S={ {1},{2},{3},{4} }

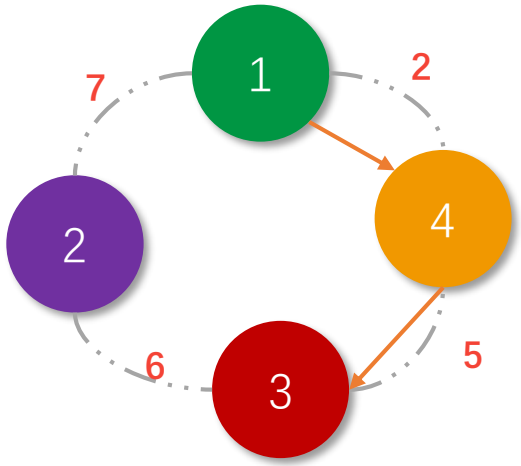
V={}



选取当前未被合并最小权值边 <1 , 4>

S={ {1 , 4},{2},{3}}

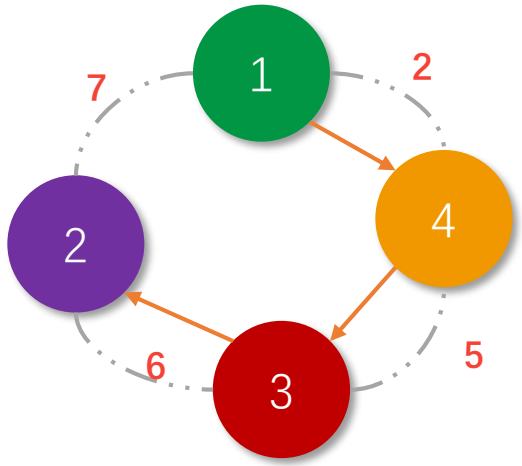
V={ <1 , 4> }



选取当前未被合并最小权值边 <4 , 3>

S={ {1 , 4 , 3},{2} }

V={ <1 , 4> ,<4 , 3>}



选取当前未被合并最小权值边 <3 , 2>

S={ {1 , 4 , 3 , 2} }

V={ <1 , 4> ,<4 , 3> , <3 , 2>}

#1886 最短网络

【题目描述】

农民约翰被选为他们镇的镇长！

他其中一个竞选承诺就是在镇上建立起互联网，并连接到所有的农场。当然，他需要你的帮助。

约翰已经给他的农场安排了一条高速的网络线路，他想把这条线路共享给其他农场。

为了用最小的消费，他想铺设最短的光纤去连接所有的农场。

你将得到一份各农场之间连接费用的列表，你必须找出能连接所有农场并所用光纤最短的方案。

每两个农场间的距离不会超过 100000 。

【输入】

第一行：农场的个数，$N(3 \leq N \leq 100)$ 。

第二行到结尾:后面的行包含了一个 $N \times N$ 的矩阵,表示每个农场之间的距离。

理论上，他们是 N 行，每行由 N 个用空格分隔的数组成，实际上，他们限制在 80 个字符，因此，某些行会紧接着另一些行。

当然，对角线将会是 0 ，因为不会有线路从第 i 个农场到它本身。

【输出】

只有一个输出，其中包含连接到每个农场的光纤的最小长度。

```
sort(edg.begin(), edg.end());
for (int i = 0; i < edg.size() && cnt < n - 1; i++)
{
    cur = edg[i];
    if (find(cur.u) != find(cur.v))
    {
        merge(cur.u, cur.v);
        ans -= cur.w;
        cnt++;
    }
}
```

谢谢观看