程序设计实训

南京大学智能科学与技术学院 史桀绮

联系方式

jayceesjq@gmail.com

南雍楼

大作业：下节课开始按顺序报告

- 五人一组完成，结课提交源代码、实验报告和一份6分钟以内的介绍视频，主要解释实现的重要功能，并附上相关的代码片段，展示完整的编译-运行、试玩流程。

- 完成的题目选题(可作为参考):
  - 扫雷小游戏
  - 贪吃蛇小游戏
  - 2048
  - 自选

- 电脑端程序，使用C++完成，需要运用面向对象的编程方法

- 最后三节课会在课上播放大家的视频，每组附3分钟提问-答疑时间，并在教学平台上互相提交评分，作为大作业得分的20%

4 蒋雨泽 蔡志诚 胡溢洋 平安
13 黄逸冰 李嘉睿 巨蛟文 潘志宇 赵歆烨
8 江哲浩 曾宇航 杨胤鸿 方煜东
7 王敬 蓝馨如 傅钰程 郑嘉怡
21 宋文豪 于啸洋 郭毓斌 潘昊
25 蒋涵 唐炜程 陈凯煜 殷悦涵 林之栋
33 黄乐彤 干佳惠 梁星茹 钱雨熠 肖桐滢
3 刘霄 崔泽宇 孙刘瑄 孙俊晖 独宇涵
2 陈柏均 刘晓睿 彭晋 叶晋
36
1 王子曰 陈浩扬 黄荣昊 周训澜
28 李尚谕 李乐佳 柯雨虹 陆姝一
9 李宛芩 张凌钰 孔泽慧 李晞文 张欣越
19 张子晗 谷羽 庄骐嘉 汤子恒 胡云迪
34 杨饶江源 郑昊 王皓然 蒋屹竹 张奕珩
15 毛绎然 陈旺 梁大锴 滕燎原 苏敬茗
30 吴凯航 李霖昊 黄栎浩 牛永辉 颜子为
20 陈智恒 唐国川 王睿清 陈嘉伟
11 景子杰 邓卓 廖海华 刘浩铭 王祺皓
12 王董浩 林锦程 关宸昊 林晨浩
31 杜昊宸 徐志杰 黄宏星 许浩博
17 吴伟铭 邱昱宁 黄子越 顾锦睿 米德瑞
10 刘旭 张梓萌 陆慕尧 周思澄
32 崔云路、邹林峰、郑泽旭、高贵桐
26 周丁杰 常博 杨艳浩 郑星宇
23 李建霖 王彦博 陈晓东 赵思棋
29 黄晴 甘志东 廖晨晰
18 拾乙淅 张竣超 袁浩程 吕禹泽
6 张智皓 周江浩 曹笑怀 李知涵
14 许政阳 卞兆洲 王丹义 黄健鑫
16 陈益佳 高悦雯 李嘉平 杨若柳 张丁玲
22 郭笑文 张子杰 马嘉骏
35 马楷恒 王晟宇 朱博文 朱善哲 李畅锦
27 柳星如 赵雨昕 彭玥琳 任启华
5 蒋依帆 戴敏欣 陈安琪 钱丹越 曹佳诺
24 孙洪涛 欧湘隽 汤志远 王辰崧

算法分类复习—数论图论

两只鼹鼠

一个圆形花圃被分为k个扇形区域（0,1，…，k-1）。有两只小鼹鼠A，B分别位于其中的两个区域。鼹鼠从一个扇形区域移动到相邻的区域需要1分钟。已知A鼹鼠总是沿顺时针方向移动(序号增大)，B鼹鼠总是沿逆时针方向移动（序号减小），两只鼹鼠都是每隔一段时间钻出地面一次。请你求出两只鼹鼠第一次同时在同一个扇形区域钻出地面的时间。如果永远遇不到，则输出“no answer”。

两只鼹鼠

输入

第一行为一个整数k，表示扇形区域的个数。1 <= k< = 10000
第二行为两个整数la，lb分别表示两只鼹鼠初始时所处的位置。0 <= la, lb < k。
第三行为两个整数ia，ib分别为两只鼹鼠钻出地面的间隔时间。0 < ia, ib < 100。
第四行为两个整数fa，fb分别为两只鼹鼠第一次钻出地面的时间。0 <= fa < ia, 0 <= fb < ib。

输出

用一个整数表示两只鼹鼠第一次同时在同一个扇形区域钻出地面的时间。
若两只鼹鼠永远无法相遇，输出“no answer”。

两只鼹鼠

样例输入
7
1 5
3 5
1 2

样例输出
37

两只鼹鼠

环形相遇问题。可以先计算相遇的时间，再计算相遇时是否在地面上。

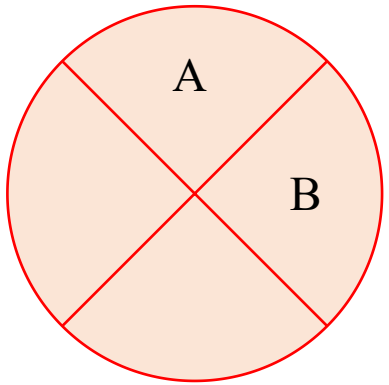
A、B鼹鼠在第x秒的位置是：la + x，lb - x。如果希望相遇，则需要:

la + x = lb - x + y * k，那么相遇的时间一定是x = (lb - la + y * k) / 2。

两只鼹鼠

先考虑不可能相邻的特殊情况：

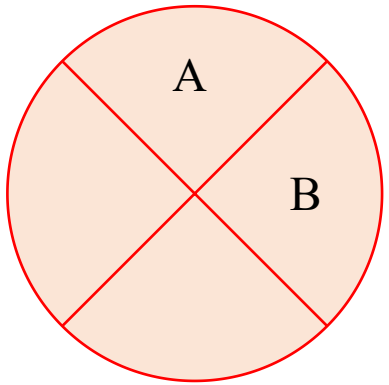
假设两者一开始相邻，那么当区域数量是偶数时，两只鼹鼠永远不可能相邻。

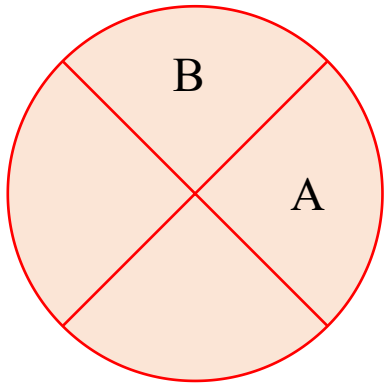


两只鼹鼠

先考虑不可能相邻的特殊情况：

假设两者一开始相邻，那么当区域数量是偶数时，两只鼹鼠永远不可能相邻。

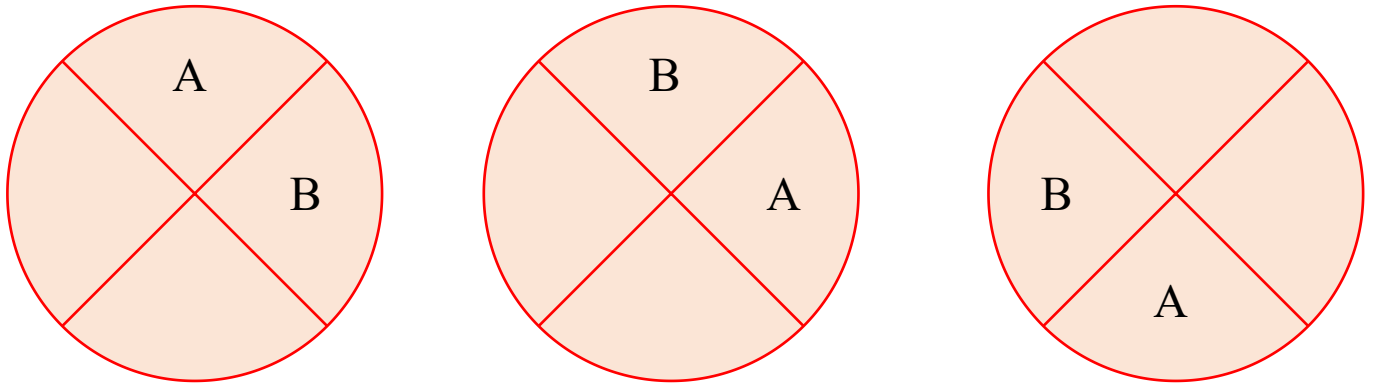




两只鼹鼠

先考虑不可能相邻的特殊情况：

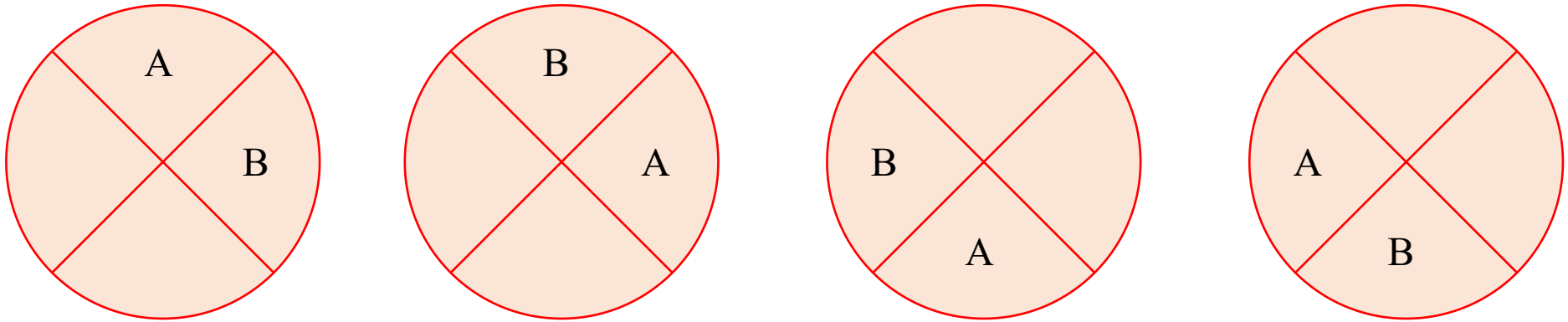
假设两者一开始相邻，那么当区域数量是偶数时，两只鼹鼠永远不可能相邻。



两只鼹鼠

先考虑不可能相邻的特殊情况：

假设两者一开始相邻，那么当区域数量是偶数时，两只鼹鼠永远不可能相邻。



两只鼹鼠

通过枚举相遇时间x = (lb - la + y * k) / 2，判断是否满足A, B在这个时间点出来透气，来获得最终的结果。

1. 在k是偶数时，dis = [(lb - la + k) % k]。dis是奇数时直接返回不可能相遇。dis是偶数时，dis = dis / 2，step = k / 2。
2. 在k是奇数时，dis = [(lb - la + k) % k]。dis是奇数则dis = (dis + k) / 2，否则dis = dis / 2, step = k。

两只鼹鼠

通过枚举相遇时间x = (lb - la + y * k) / 2，判断是否满足A, B在这个时间点出来透气，来获得最终的结果。

1. 在k是偶数时，dis = [(lb - la + k) % k]。dis是奇数时直接返回不可能相遇。dis是偶数时，dis = dis / 2，step = k / 2。
2. 在k是奇数时，dis = [(lb - la + k) % k]。dis是奇数则dis = (dis + k) / 2，否则dis = dis / 2, step = k。

判断内容:
1. (dis - fa) % ia == 0，并且(dis - fb) % ib == 0，即两只鼹鼠都探头，返回dis。
2. 很多轮依然没有碰头，可以近似假设不可能碰头。
3. A, B都走到了重复的状态，仍然没有碰面，进入循环。

两只鼹鼠

```
if(k%2==0){
    dis=(lb-la+k)%k;
    if(dis%2!=0)return -1;
    dis=dis/2;
    step=k/2;
 }
 else{
    dis=(lb-la+k)%k;
    if(dis%2==0)dis=dis/2;
    else dis=(dis+k)/2;
    step=k;
 }
```

两只鼹鼠

```
int iter=0;
while(True){
    int t1=(dis-fa)%ia;
    int t2=(dis-fb)%ib;
    if(t1==t2&&t1==0)
        return dis;
    dis+=step;

    if(visited[t1]==t2)return -1;
    visited[t1]=t2;
    iter++;
    if(sum>1e6)return -1;
}
```

宗教信仰

世界上有许多宗教，你感兴趣的是你学校里的同学信仰多少种宗教。
你的学校有n名学生（0 < n <= 50000），你不太可能询问每个人的宗教信仰，因为他们不太愿意透露。但是当你同时找到2名学生，他们却愿意告诉你他们是否信仰同一宗教，你可以通过很多这样的询问估算学校里的宗教数目的上限。你可以认为每名学生只会信仰最多一种宗教。

输入

输入包括多组数据。
每组数据的第一行包括n和m，0 <= m <= n(n-1)/2，其后m行每行包括两个数字i和j，表示学生i和学生j信仰同一宗教，学生被标号为1至n。输入以一行 n=m=0 作为结束。

输出

对于每组数据，先输出它的编号（从1开始），接着输出学生信仰的不同宗教的数目上限。

宗教信仰

样例输入

```
10 9
1 2
1 3
1 4
1 5
1 6
1 7
1 8
1 9
1 10
10 4
2 3
4 5
4 8
5 8
0 0
```

样例输出

Case 1: 1
Case 2: 7

宗教信仰

典型的并查集数问题。

并查集是一种树型的数据结构，用于处理一些不相交集合的合并及查询问题（即所谓的并、查）。比如说，我们可以用并查集来判断一个森林中有几棵树、某个节点是否属于某棵树等。

并查集的主要作用是求连通分支数（如果一个图中所有点都存在可达关系（直接或间接相连），则此图的连通分支数为1；如果此图有两大子图各自全部可达，则此图的连通分支数为2……）

并查集一般由合并操作和查找操作组成。

宗教信仰

合并操作：

```
int father[100000];
memset(father,-1,sizeof(father));
for(int i=1;i<=m;i++){
    cin>>a>>b;
    int fa=find(a),fb=find(b);
    if(fa!=fb)father[fa]=fb,n--;//并
 }
 cout<<"Case "<<++tmp<<": "<<n<<endl;
```

注意合并操作不需要顺着father到son的路径修改所有元素，只需要找到根节点并修改。

宗教信仰

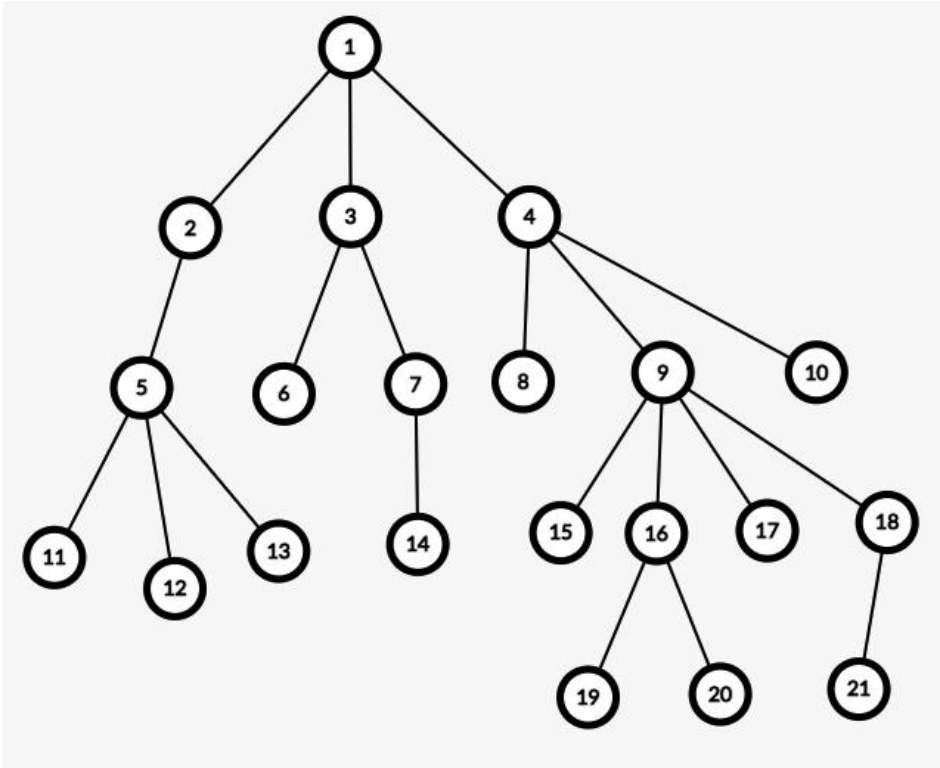
查找操作:

```
int find(int x){
    if(father[x]==-1)return x;
    else return find(father[x]);
}
```

最近公共祖先(LCA)问题

给出一棵包含 N 个节点的有根树，节点编号 1 到 N，根节点为 R。请回答 Q 个查询，计算节点 u 和节点 v 的最近公共祖先。

例如，下图是以 1 号节点为根的有根树，其中 11 和 13 的最近公共祖先为 5，2 和 8 的最近公共祖先为 1，19 和 21 的最近公共祖先为 9，16 和 1 的最近公共祖先为 1，15 和 15 的最近公共祖先为 15。



最近公共祖先(LCA)问题

输入

第一行包含两个整数 N, R (1<=R<=N<=1000) 分别表示节点总数和根结点编号。
接下来 N-1 行，每行表示一条边，包含两个整数 u, v (1<=u,v<=N, u≠v)。输入保证全部节点构成一棵树。
接下来一行，包含一个整数 Q (1<=Q<=2000) 表示询问次数。
接下来 Q 行，每行两个整数 x, y (1<=x,y<=N) 表示询问节点 x 和 节点 y 的最近公共祖先。

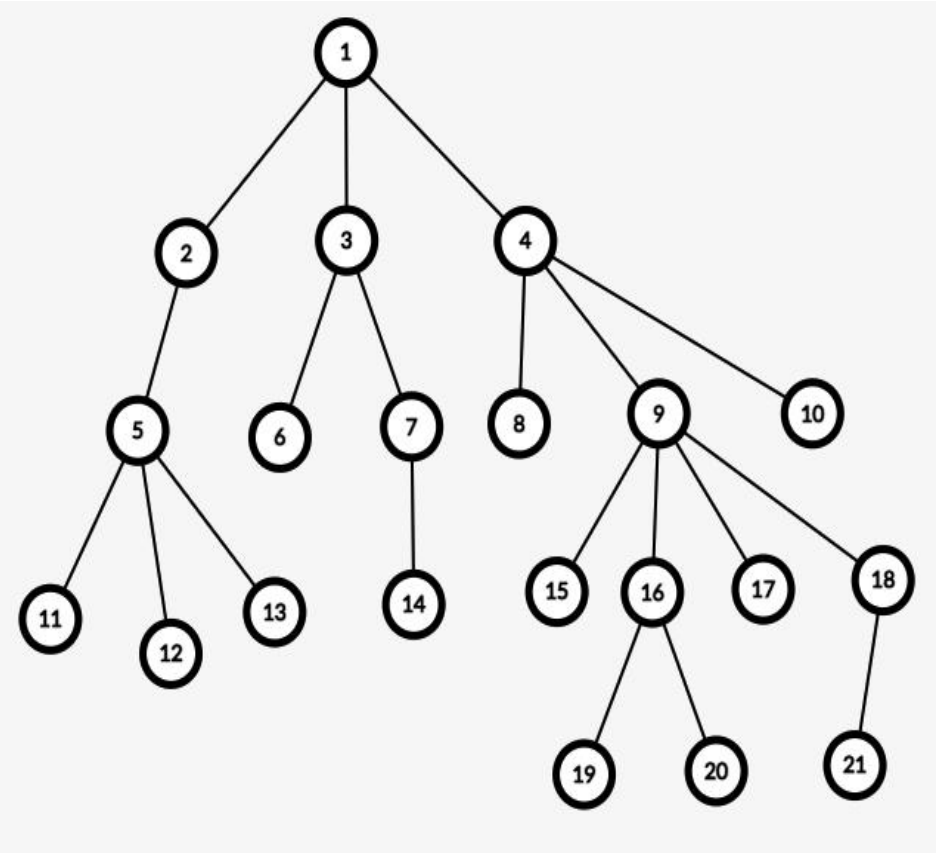
输出

一共 Q 行，每行一个整数表示答案。

最近公共祖先(LCA)问题

样例输入

```
21 1
1 2
1 3
1 4
2 5
3 6
3 7
8 4
9 4
10 4
11 5
12 5
13 5
14 7
9 15
16 9
17 9
9 18
16 19
16 20
18 21
10
19 20
19 18
16 10
12 21
11 6
2 5
3 9
14 6
17 1
15 15
```



样例输出

```
16
9
4
1
1
2
1
3
1
15
```

最近公共祖先(LCA)问题

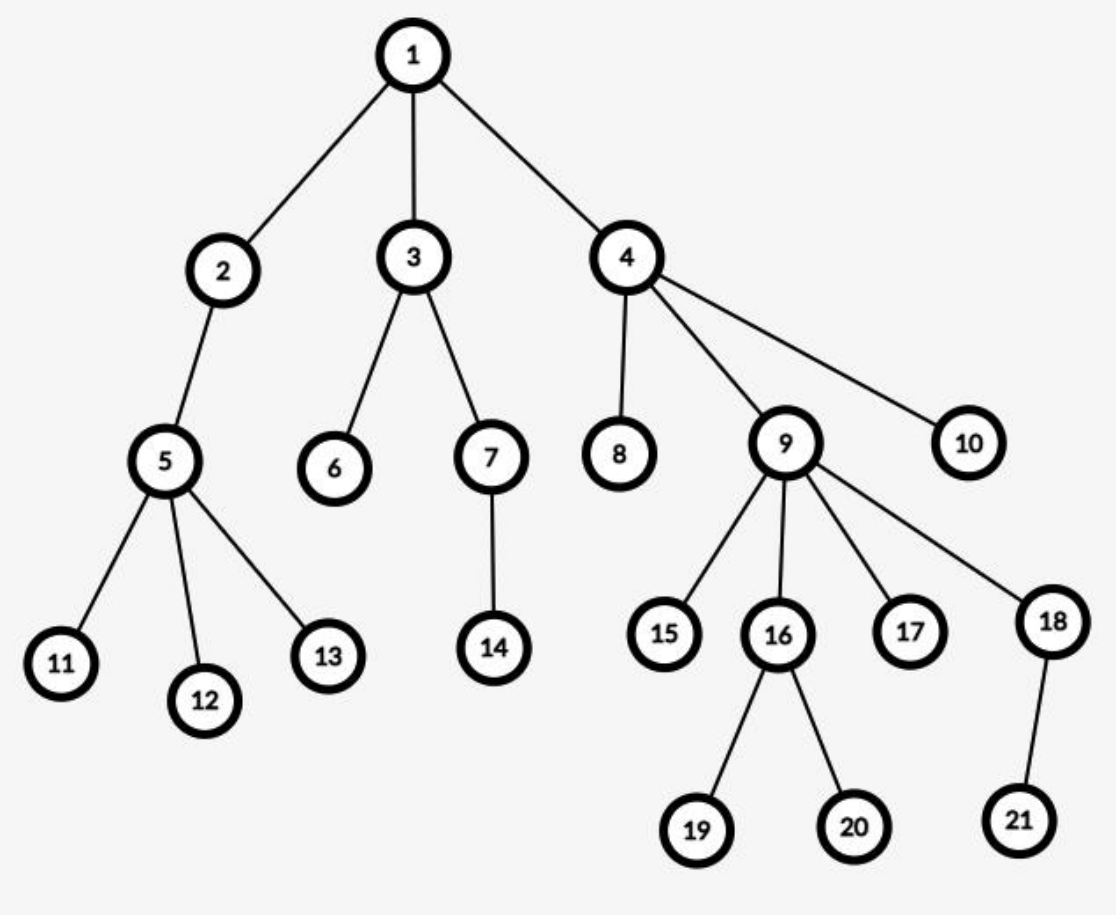
注意与之前我们提到的二叉树公共祖先不同，LCA问题不保证树是一颗搜索树，因此无法使用大小来判断在树的哪个子节点。

可以得知的规律:

1. 若两个结点u、v分别分布于某节点t 的不同子树，那么此节点 t即为u和v的最近公共祖先。

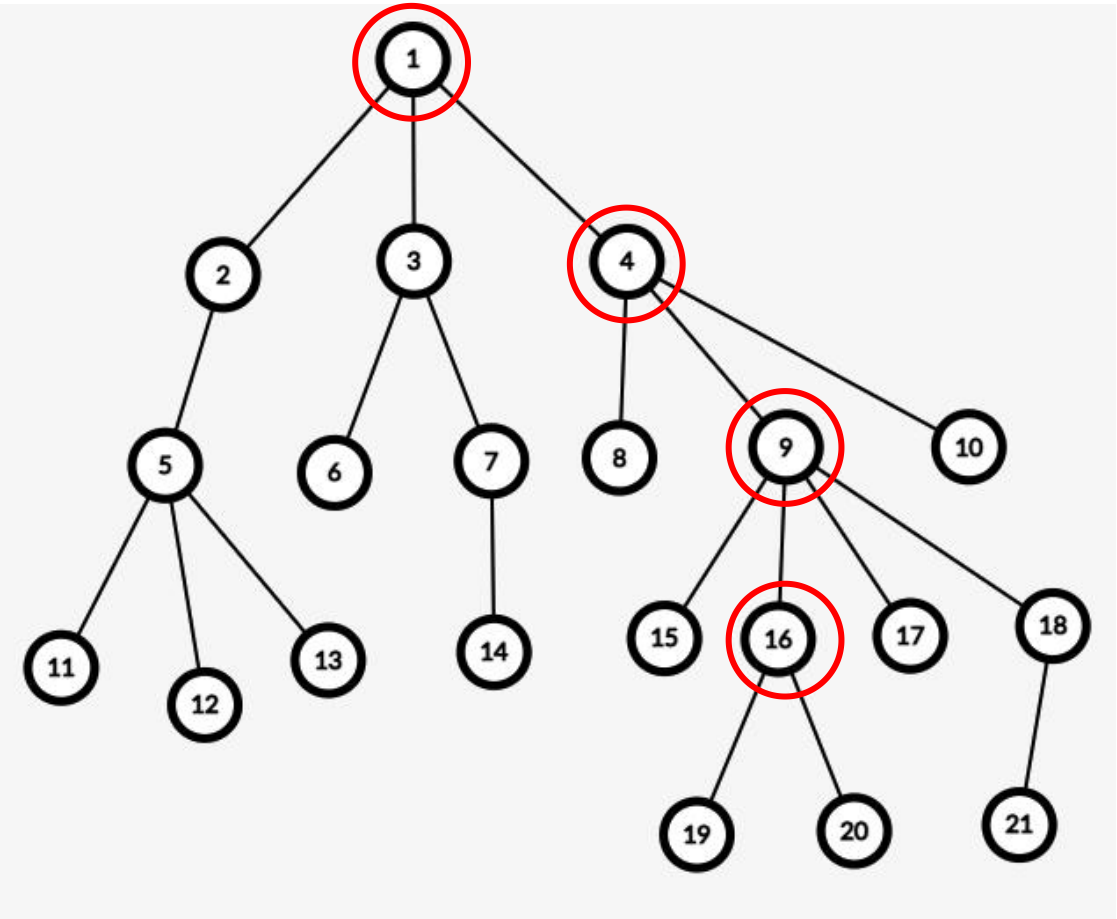
2. 若某结点t 是两结点u、v的祖先之一，且这两结点并不分布于该结点t 的一棵子树中，而是分别在结点t 的左子树、右子树中，那么该结点t 即为两结点u、v的最近公共祖先。

最近公共祖先(LCA)问题



假设找
19 20
19 18
16 10
12 21
11 6

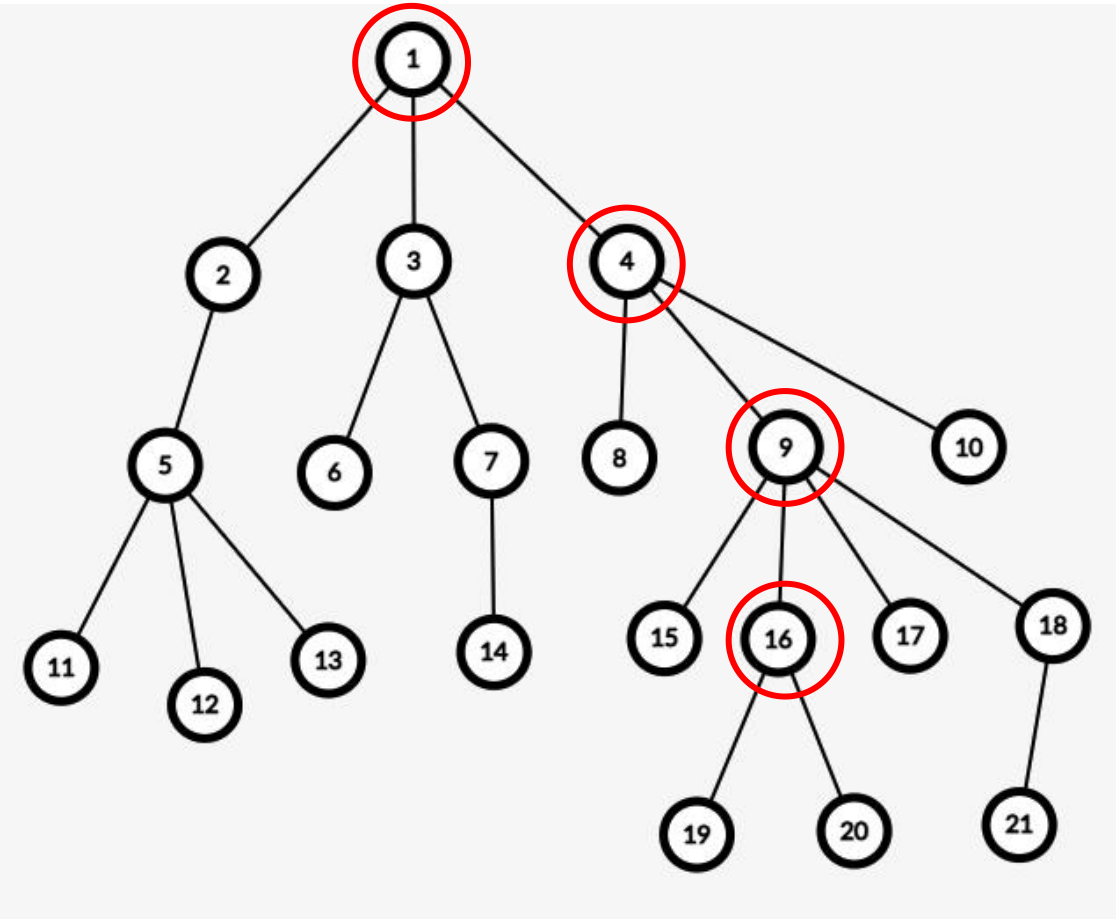
最近公共祖先(LCA)问题



我们偷懒，假设一开始就从1-4-9-16-19这条路开始寻找。

在16，发现其有两个子节点，先搜索19。

最近公共祖先(LCA)问题

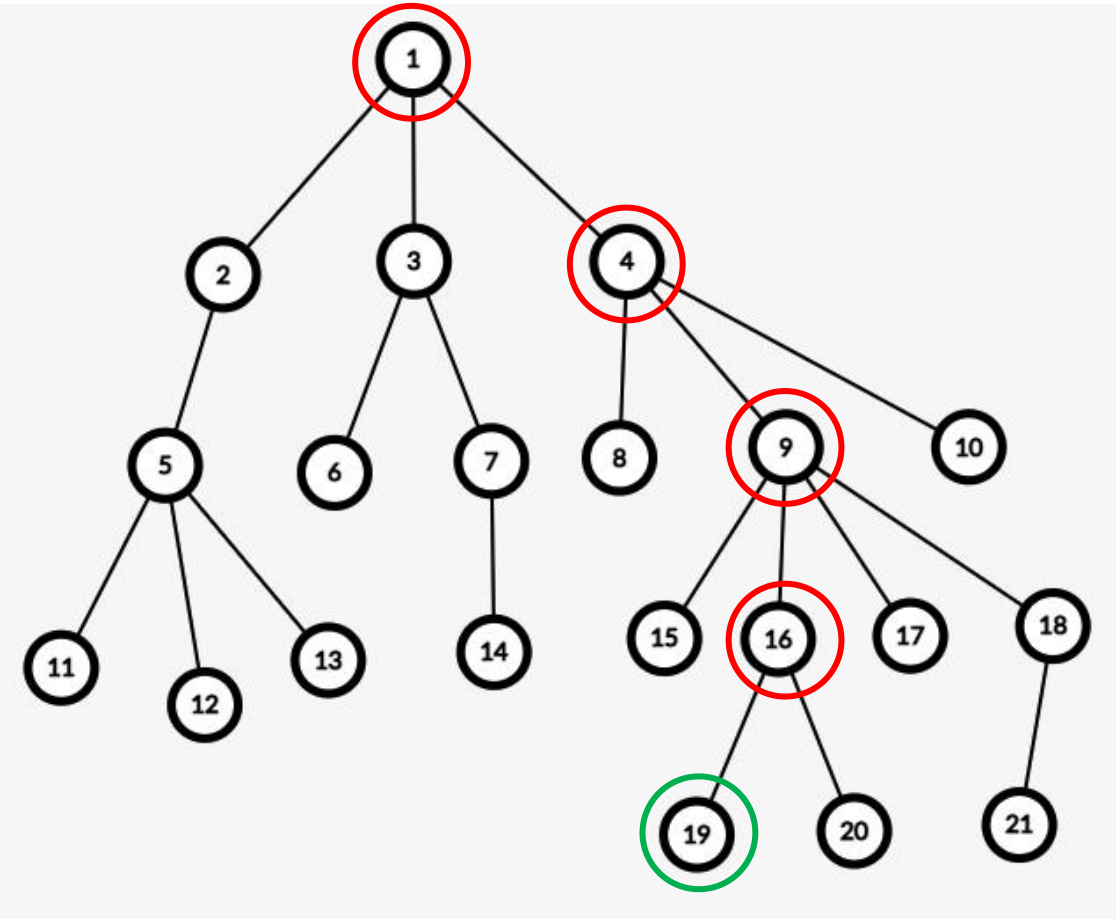


我们偷懒，假设一开始就从1-4-9-16-19这条路开始寻找。

在16，发现其有两个子节点，先搜索19。

19没有子节点，因此开始寻找询问中和19有关的节点：20和18，都还没有访问。

最近公共祖先(LCA)问题



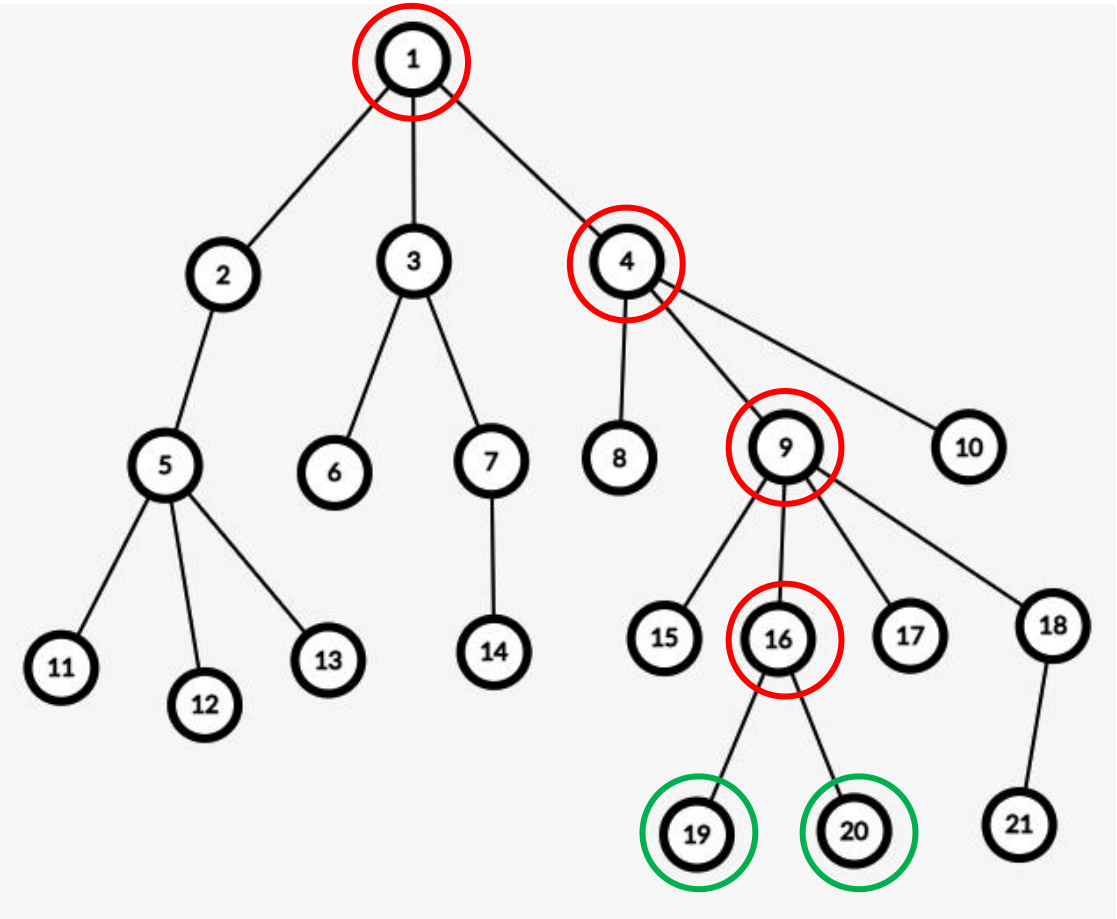
我们偷懒，假设一开始就从1-4-9-16-19这条路开始寻找。

在16，发现其有两个子节点，先搜索19。

19没有子节点，因此开始寻找询问中和19有关的节点：20和18，都还没有访问。

19设为visited，f[19]=16。

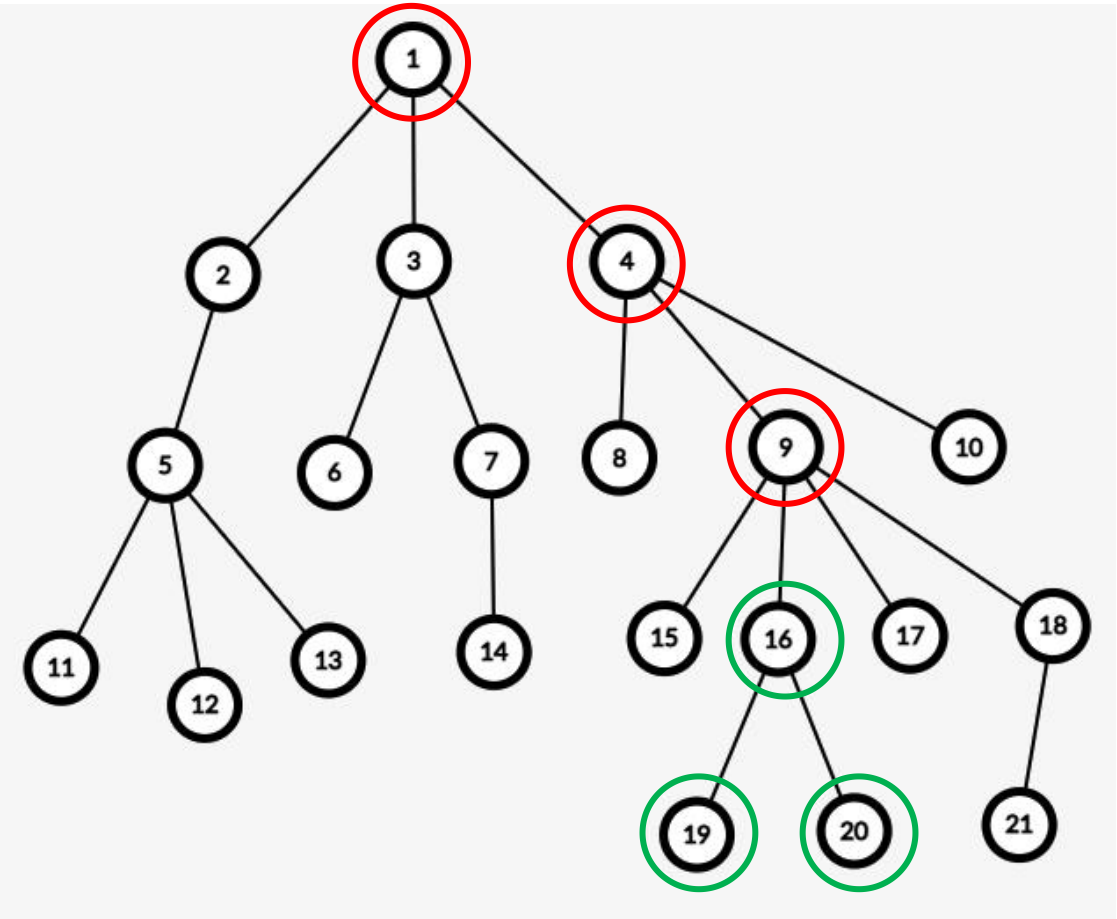
最近公共祖先(LCA)问题



返回16，16还有节点20，进入节点20。和20有关的节点是19和16。16的visit还是false，19是true，因此采用find(19)，返回16。19,20的LCA是16。

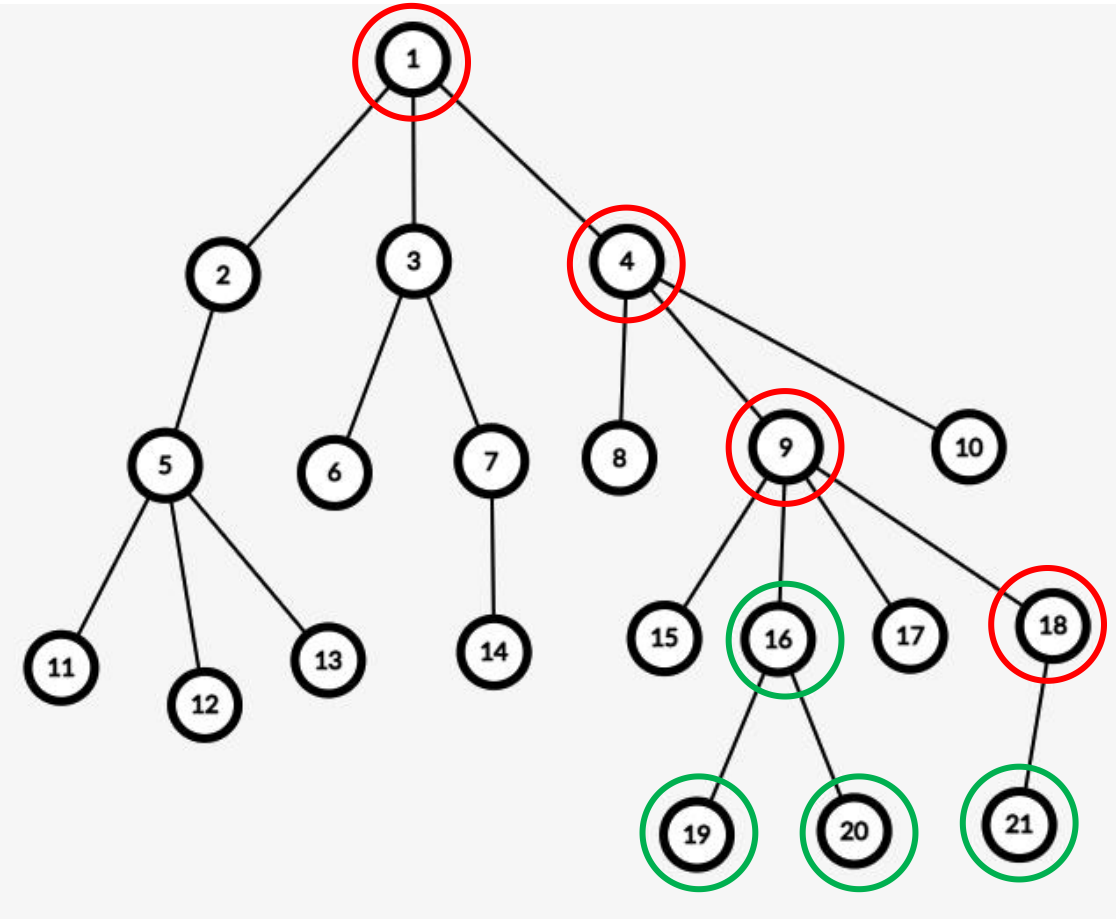
20处理完毕，设置为visited，f[20]=16。

最近公共祖先(LCA)问题



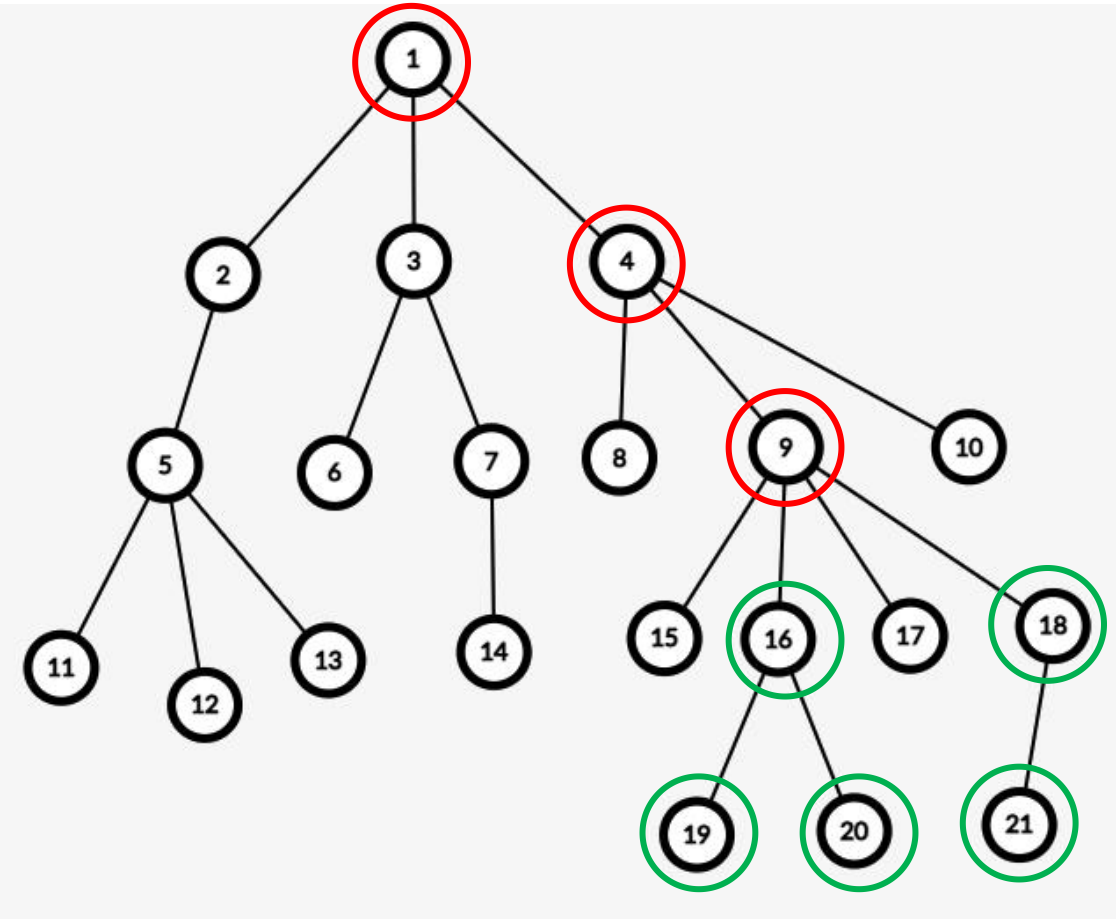
返回16，16没有未处理子节点了，因此开始查找和16有关的问询，包括10。10未访问，因此16处理完，visited[16]=True,f[16]=9，继续返回。

最近公共祖先(LCA)问题



9访问子节点18，18有子节点21。和21有关的问询是12，还未访问，因此21处理结束。返回18。

最近公共祖先(LCA)问题



9访问子节点18，18有子节点21。和21有关的问询是12，还未访问，因此21处理结束。返回18，f[21]=18。

18没有别的子节点，但是有相关问询19 18。19已经处理过，因此返回find(19)=9。18返回，f[18]=9。

最近公共祖先(LCA)问题

Tarjan算法伪代码：

```
Tarjan(u){
        for each(v){     //v是u的子节点
             Tarjan(v);        //继续往下遍历
             marge(u,v);    //合并v到u上
        }
        for each(问询<u,e>){
            如果e被访问过;
            u,e的最近公共祖先为find(e);
        }
}深度优先搜索+并查集
```

最近公共祖先(LCA)问题

另一种解法：ST算法(Sparse Table系数表)。

1. 运用DP思想求解区间最值，并将结果保存到一个二维数组中。

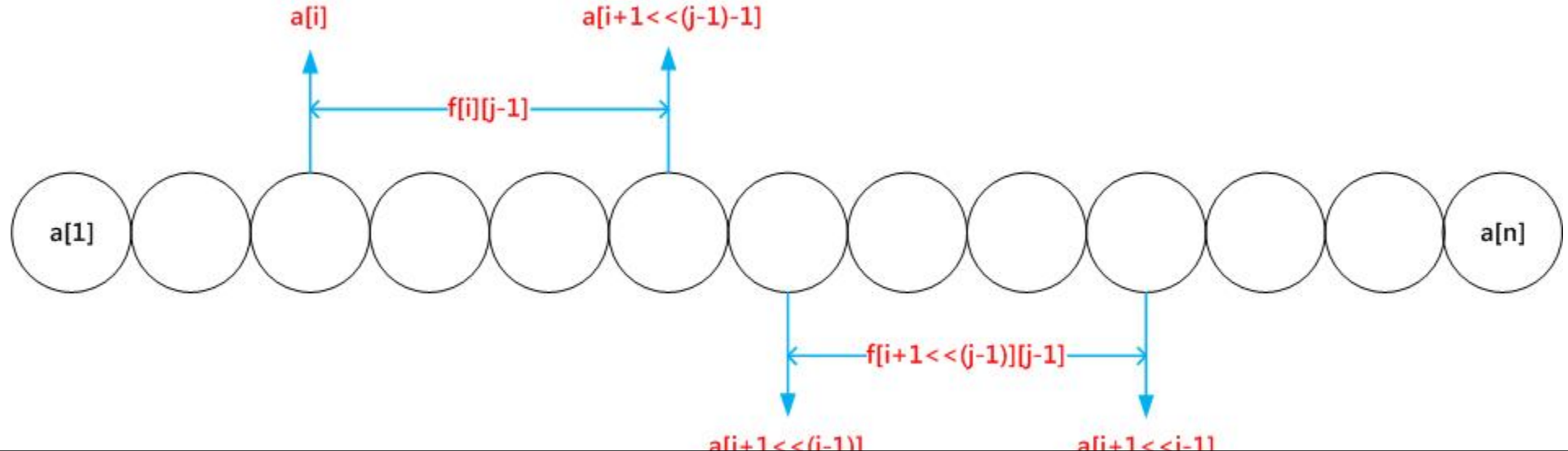
2. 对给定区间进行分割，并借助上步中的二维数组求最值。

最近公共祖先(LCA)问题

另一种解法：ST算法(Sparse Table系数表)。

1. 运用DP思想求解区间最值，并将结果保存到一个二维数组中。

对于从a[i]到a[i+2^j-1]的区间求取最大值，可以二分查找后进行归并。



最近公共祖先(LCA)问题

另一种解法：ST算法(Sparse Table系数表)。

1. 运用DP思想求解区间最值，并将结果保存到一个二维数组中。

对于从a[i]到a[i+2^j-1]的区间求取最大值，可以二分查找后进行归并。

用f[i][j]表示从 a[i] 到 a[i+2^j-1] 范围内的最大值。ST算法用到了倍增思想，将 2^j 个数从中间平均分成两等分，每一部分有 2^(j-1) 个数。

f[i][j]=max(f[i][j-1],f[i+2^(j-1)][j-1])

最近公共祖先(LCA)问题

2. 对给定区间进行分割，并借助上步中的二维数组求最值。

给定查询区间 [x,y]。首先求出最大的 k，使之满足 2^k ≤ y-x+1。

在此基础上，区间 [x,y]=[x,x+2^k-1]∪[y-2^k+1,y]，则区间 [x,y] 内的最大值为 max(f[x][k],f[y-(1<<k)+1][k]) 。

注意，求区间最大值时有交集不会影响最后结果。

总结上述规律，可以得到：
k=log2(y-x+1)，max(f[x][k],f[y-(1<<k)+1][k])

最近公共祖先(LCA)问题

在本题中，使用深度优先遍历，可以为树构造一个<序号，节点，*>的表格。

深度优先遍历的连续区间其实代表一颗子树，而在找两个节点的lca时，其实就是在找包含这两个点的子树的根节点。因此，只需要将根节点与某个特性进行关联，就可以用求最大最小值的方法直接查表。

最近公共祖先(LCA)问题

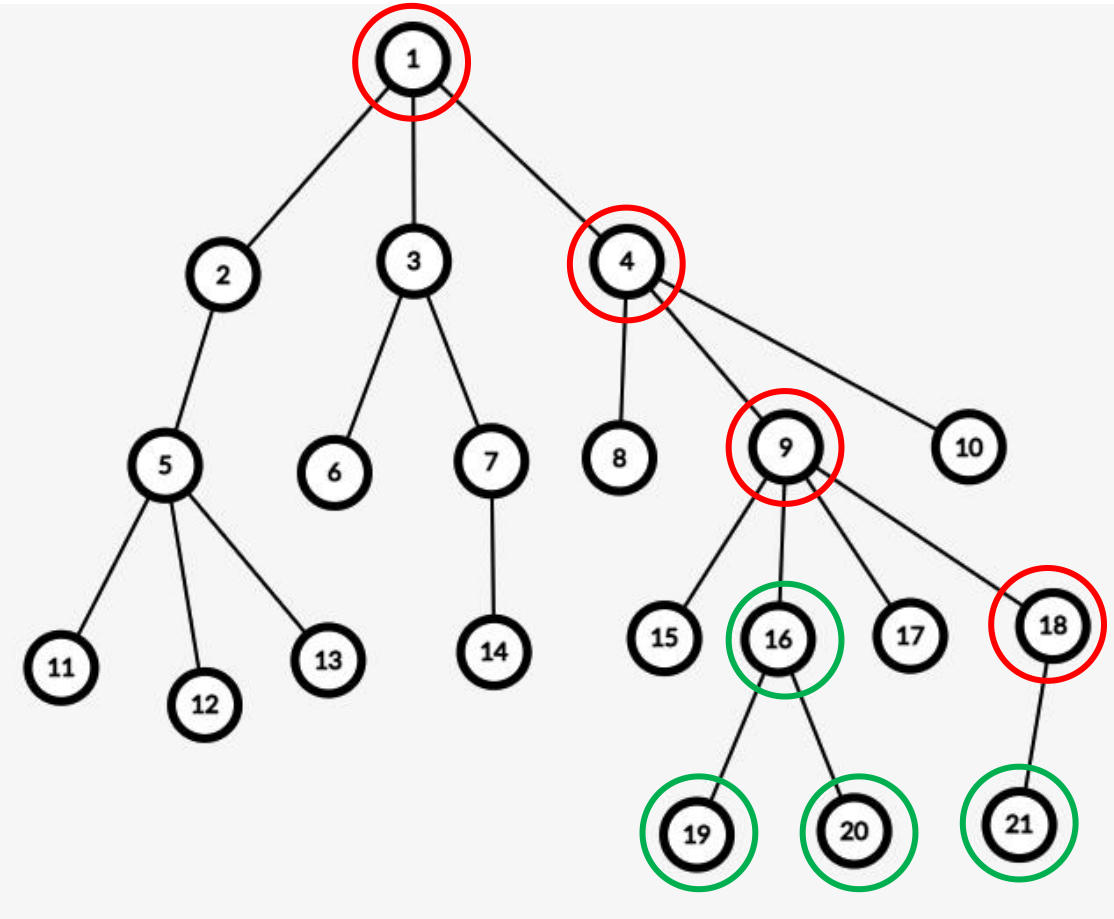
在本题中，使用深度优先遍历，可以为树构造一个<序号，节点，*>的表格。

深度优先遍历的连续区间其实代表一颗子树，而在找两个节点的lca时，其实就是在找包含这两个点的子树的根节点。因此，只需要将根节点与某个特性进行关联，就可以用求最大最小值的方法直接查表。

显然，可以选用深度作为根节点的标志，构造一个<序号，节点，深度>的表格。

对于输入的任意一堆问询，查找ST表格中对应序列的深度最小值，对应的就是lca。

最近公共祖先(LCA)问题

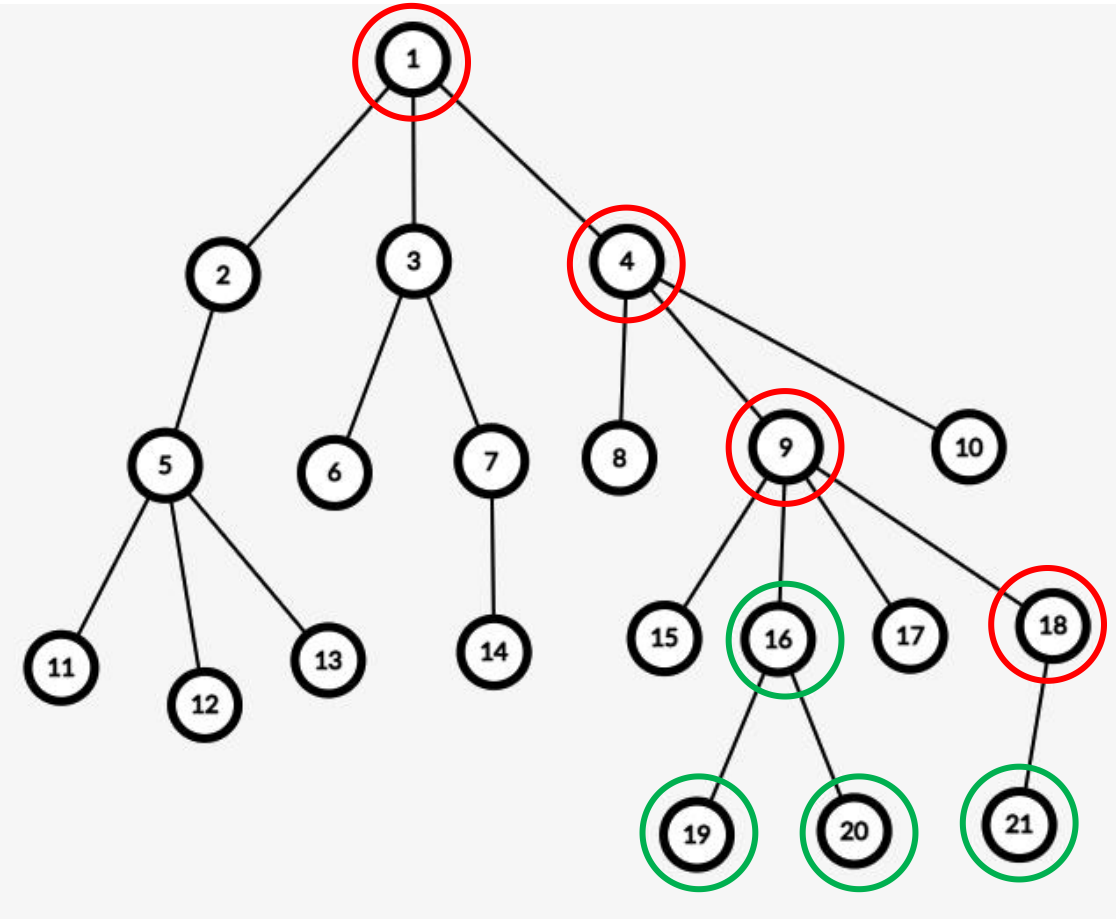


我们为9-16-19-29-18-21建立一个ST表。问询<19,20>, <19, 18>。

1	2	3	4	5	6
9	16	19	16	20	16
0	1	2	1	2	1

7	8	9	10	11	12
9	18	21	18	9	X
0	1	2	1	0	X

最近公共祖先(LCA)问题



我们为9-16-19-29-18-21建立一个ST表。问询<19,20>, <19, 18>。

1	2	3	4	5	6
9	16	19	16	20	16
0	1	2	1	2	1

7	8	9	10	11	12
9	18	21	18	9	X
0	1	2	1	0	X

地鼠一家

有n个地鼠洞和m个地鼠洞，每个洞都位于不同的（x，y）坐标处。一只鹰来了，如果地鼠在s秒内没有到达一个洞，它很容易被吃掉。一个洞最多只能救一只地鼠。所有的地鼠都以相同的速度奔跑。地鼠家族需要一种逃生策略，以尽量减少易受攻击的地鼠数量。

输入

输入包含几个案例。每种情况的第一行包含四个小于100的正整数：n、m、s和v。接下来的n行给出地鼠的坐标；以下m线给出了地鼠洞的坐标。所有距离均以米为单位；所有时间均以秒为单位；所有速度均以米每秒为单位。

输出

输出由每种情况的单行组成，给出了易受攻击的地鼠数量。

地鼠一家

样例输入

2 2 5 10
1.0 1.0
2.0 2.0
100.0 100.0
20.0 20.0

样例输出

1

地鼠一家

把地鼠看做二分图的左半部，把地洞看做二分图的右半部，如果老鼠能跑进地洞就在老鼠和地洞之间连一条边。

问题转化为求二分图的最大匹配。使用匈牙利算法。

注意，本题的边其实并不带权重，因为只有跑进/跑不进两种选择。

地鼠一家

把地鼠看做二分图的左半部，把地洞看做二分图的右半部，如果老鼠能跑进地洞就在老鼠和地洞之间连一条边。

问题转化为求二分图的最大匹配。使用匈牙利算法。

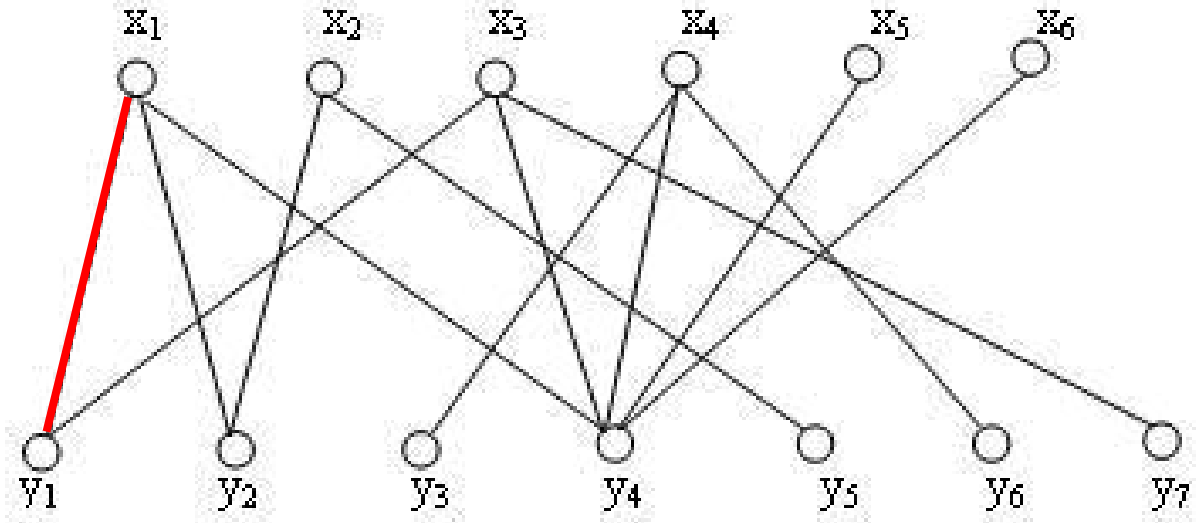
定义：

交错路径：给定图G的一个匹配M，如果一条路径的边交替出现在M中和不出现在M中，我们称之为一条M-交错路径。

而如果一条M-交错路径，它的两个端点都不与M中的边关联，我们称这条路径叫做M-增广路径。

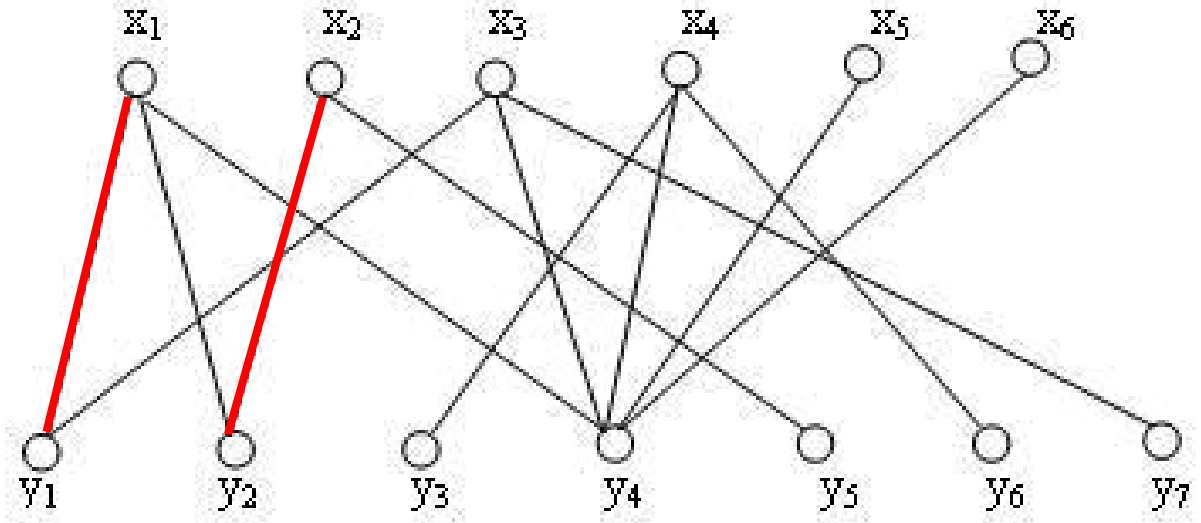
地鼠一家

首先考虑不讨论成本的二部图匹配，每个地鼠都能在单位时间1内到达任意地鼠洞，只考虑匹配上。



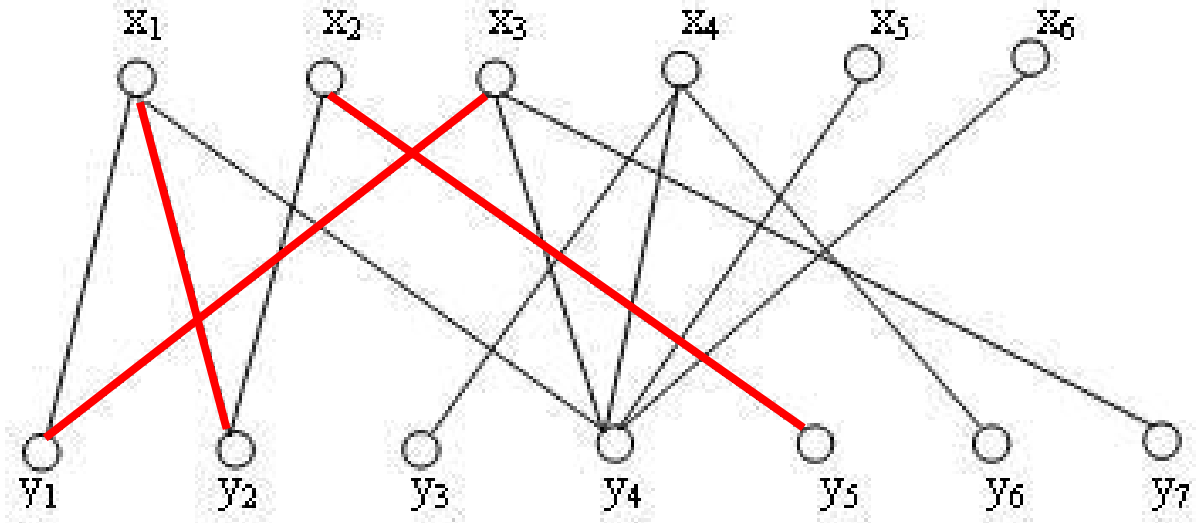
地鼠一家

首先考虑不讨论成本的二部图匹配，每个地鼠都能在单位时间1内到达任意地鼠洞，只考虑匹配上。



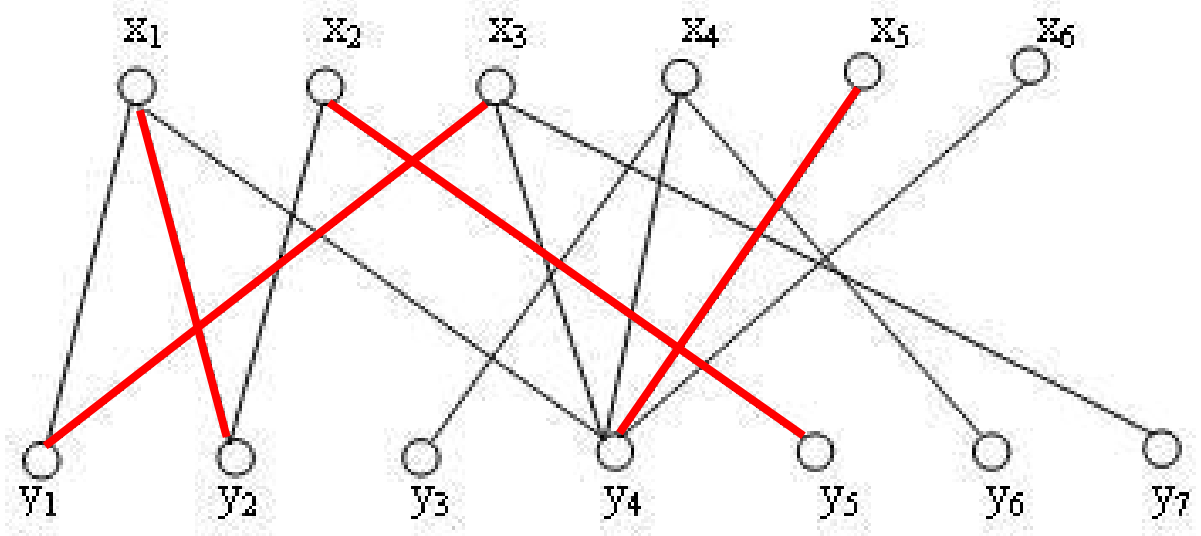
地鼠一家

首先考虑不讨论成本的二部图匹配，每个地鼠都能在单位时间1内到达任意地鼠洞，只考虑匹配上。



地鼠一家

首先考虑不讨论成本的二部图匹配，每个地鼠都能在单位时间1内到达任意地鼠洞，只考虑匹配上。



对二部图来说，最大匹配不唯一，但是最大匹配的大小唯一。

地鼠一家

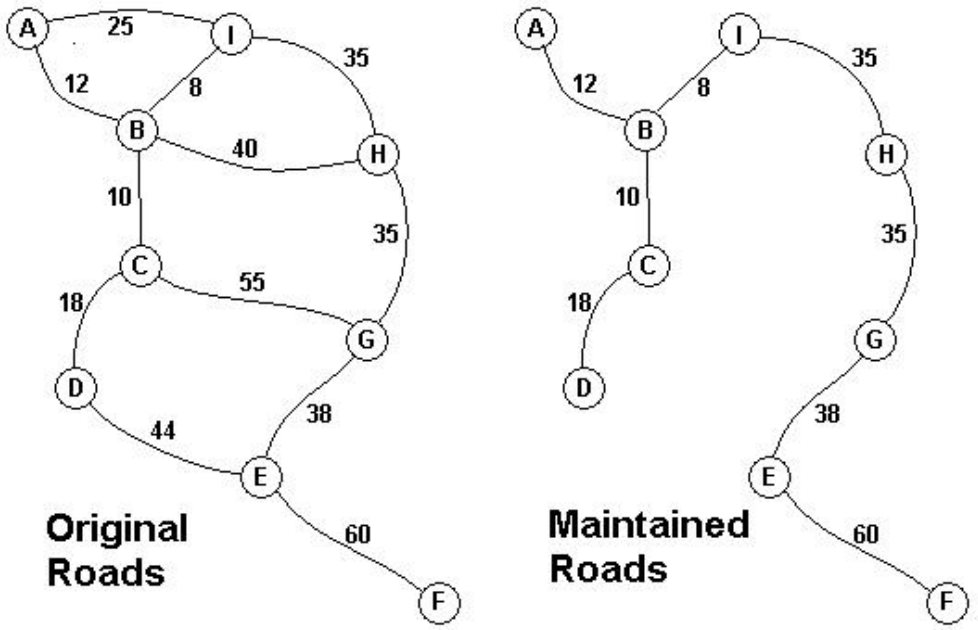
```
bool dfs(int i){
    int j;
    for (j=1;j<=m;j++)
        if (neighbour[i][j]&&visited[j]==false){
        //在一次查找中，判断j有没有被前序节点用过
            visited[j]=true;
            if (connect[j]==-1||dfs(connect[j])){ //j还没有匹配或者有增广路径
                connect[j]=s;                     //找到了，返回
                return true;
            }
     }
    return false;
}
```

地鼠一家

```
bool dfs(int i){
    int j;
    for (j=1;j<=m;j++)
        if (neighbour[i][j]&&visited[j]==false){
        //在一次查找中，判断j有没有被前序节点用过
            visited[j]=true;
            if (connect[j]==-1||dfs(connect[j])){ //j还没有匹配或者有增广路径
                connect[j]=s;                     //找到了，返回
                return true;
            }
      }
    return false;
} //注意visited数组要每次都重置
```

丛林中的路

热带岛屿Lagrishan的首领现在面临一个问题：几年前，一批外援资金被用于维护村落之间的道路，但日益繁茂的丛林无情的侵蚀着村民的道路，导致道路维修开销巨大，长老会不得不放弃部分道路的维护。上图左侧图显示的是正在使用道路的简图以及每条路每个月的维修费用（单位为aacms）。现在长老会需要提出一种方案，即需要保证村落之间都可以互相到达，又要将每个月的道路维修费用控制在最小。村子编号为从A到I。上图右侧显示的方案最小维修开销为216 aacms每月。



丛林中的路

输入

输入包含1~100组数据，最后一行为0。每组数据第一行为村落数目n, 1<n< 27, 依次用字母表的前n个字母标记。接下来有n-1行，每行的第一个数据是按字母顺序排列的村子编号（不包括最后一个村庄）。每个村庄后面的数据k代表该村庄通往编号在其之后的村庄的道路数目，如A 2 B 12 I 25，代表A村庄有2个编号在A之后的村庄和其相连。若k大于0，k后面会依次给出这k个村庄的编号以及各自到起始村庄的道路维修费用，如A 2 B 12 I 25，代表A和B之间道路维修费用为12，A和I之间道路维修费用为25（维修费用为不超过100的正整数）。路的总数目不超过75条，每个村庄到其他村庄不会有超过15条路（包括编号在其之前和之后的）。

输出

每一组数据有一个输出：针对解决方案每个月维修道路的小费用。

提示：暴力算法虽能找出解决方案，但将会超出时间限制。

丛林中的路

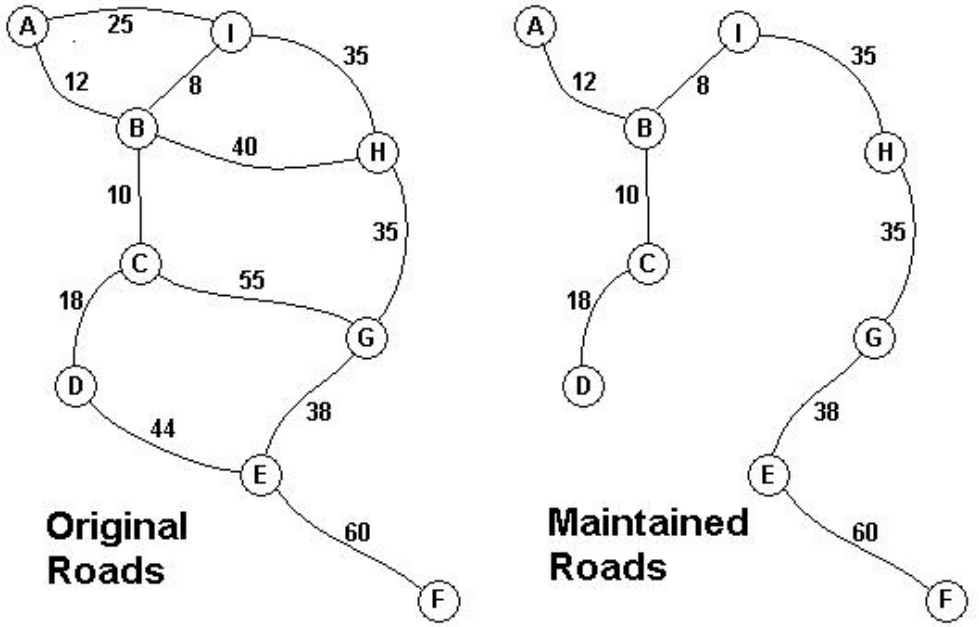
样例输入

```
9
A 2 B 12 I 25
B 3 C 10 H 40 I 8
C 2 D 18 G 55
D 1 E 44
E 2 F 60 G 38
F 0
G 1 H 35
H 1 I 35
3
A 2 B 10 C 40
B 1 C 20
0
```

样例输出

216

30

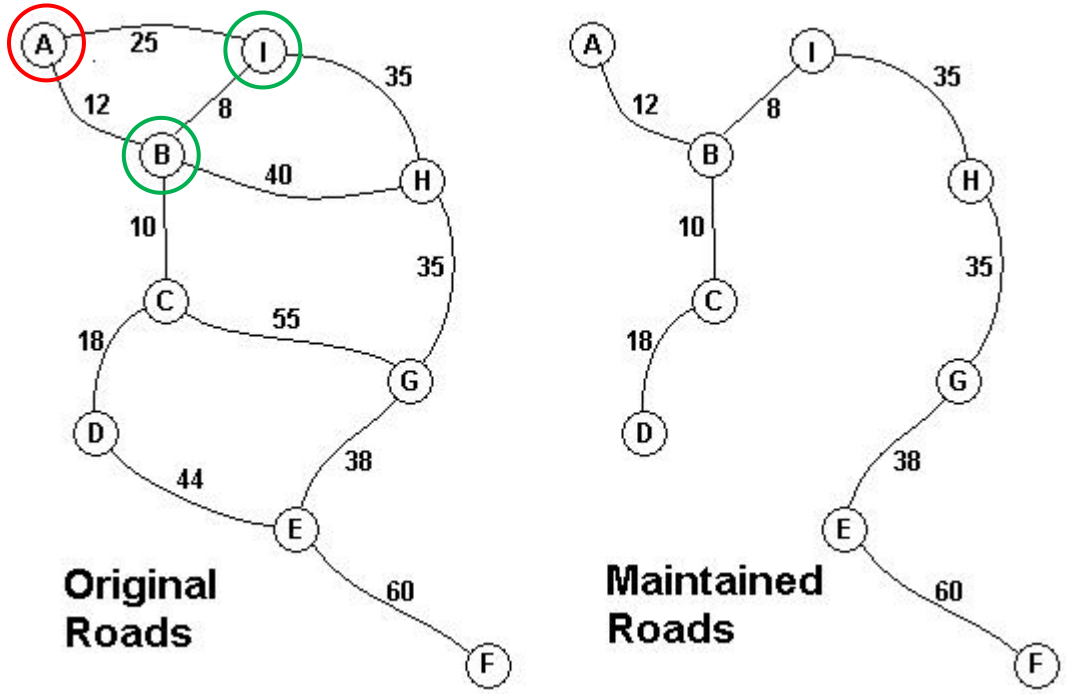


丛林中的路

最小生成树模板题目。常用Prim和Kruskal算法。

丛林中的路

prim算法：加点法。每次总是选出一个离生成树距离最小的点去加入生成树，最后实现最小生成树。

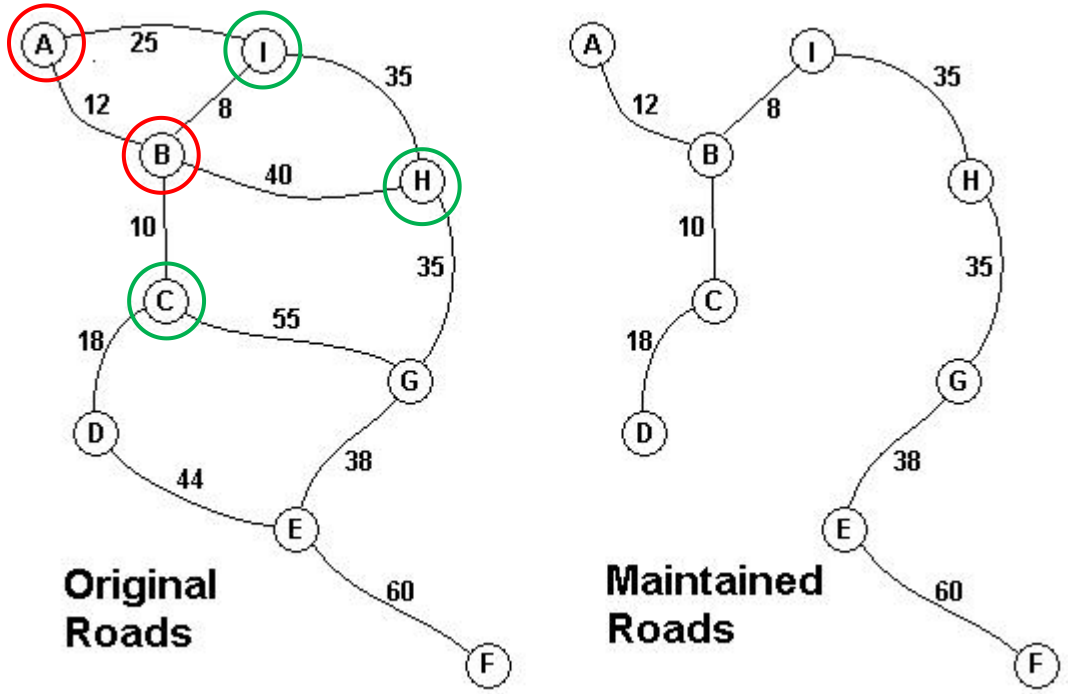


A	B	C	D	E	F	G	H	I
0	12	∞	∞	∞	∞	∞	∞	25

设置一个一维矩阵，保存生成树到其他点的距离。假设最开始加入A。

丛林中的路

prim算法：加点法。每次总是选出一个离生成树距离最小的点去加入生成树，最后实现最小生成树。

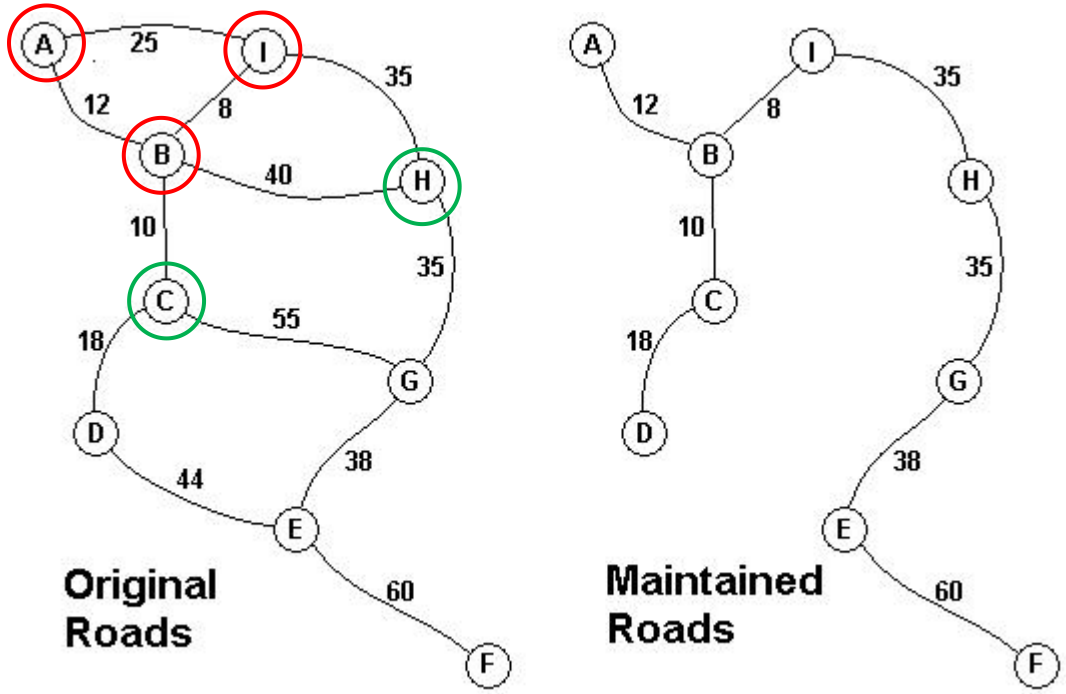


A	B	C	D	E	F	G	H	I
0	0	10	∞	∞	∞	∞	40	8

距离集合最近的点变成B，加入A-B。

丛林中的路

prim算法：加点法。每次总是选出一个离生成树距离最小的点去加入生成树，最后实现最小生成树。

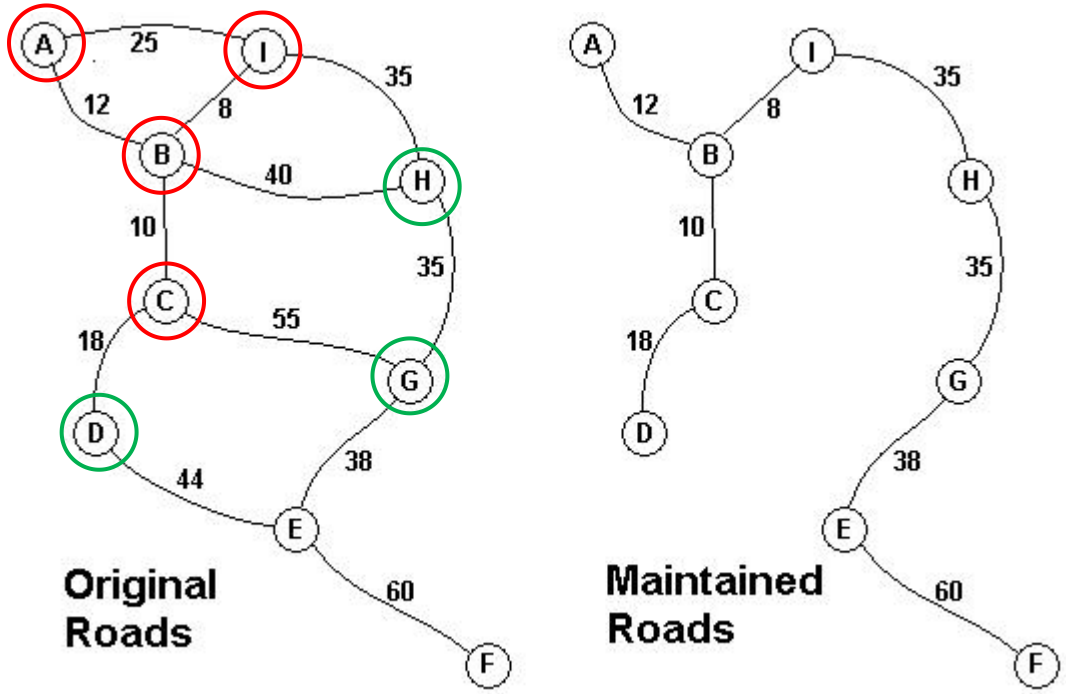


A	B	C	D	E	F	G	H	I
0	0	10	∞	∞	∞	∞	35	0

距离集合最近的点变成I，加入B-I。

丛林中的路

prim算法：加点法。每次总是选出一个离生成树距离最小的点去加入生成树，最后实现最小生成树。

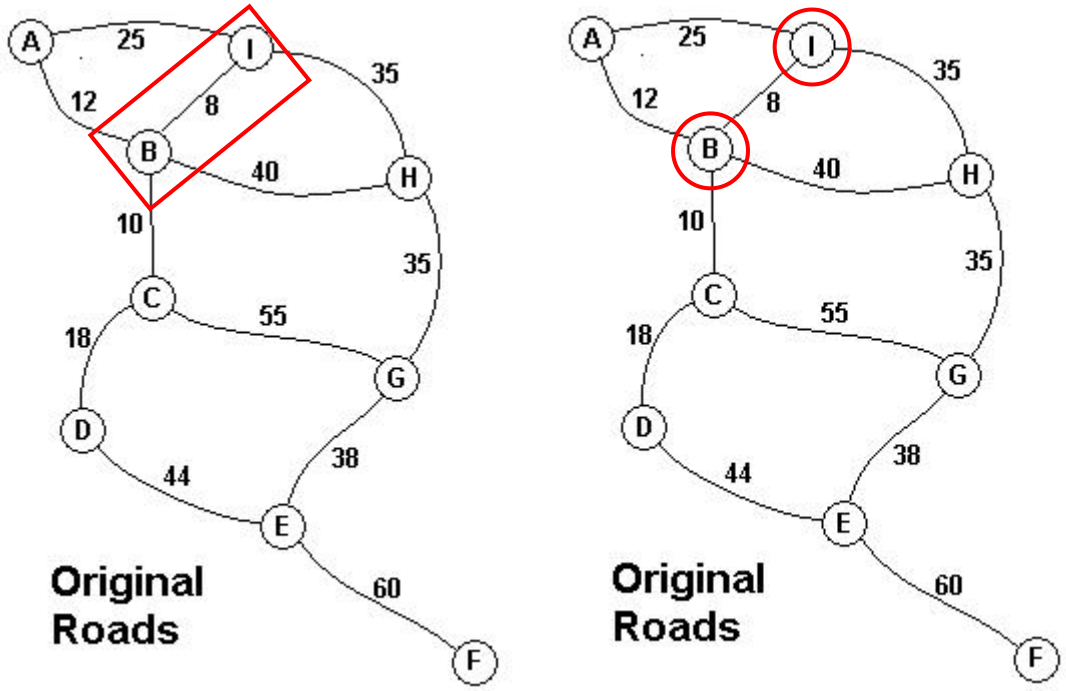


A	B	C	D	E	F	G	H	I
0	0	0	18	∞	∞	55	35	0

距离集合最近的点变成C，加入B-C。

丛林中的路

Kruskal算法：加边法。初始最小生成树边数为0，每迭代一次就选择一条满足条件的最小代价边，加入到最小生成树的边集合里。



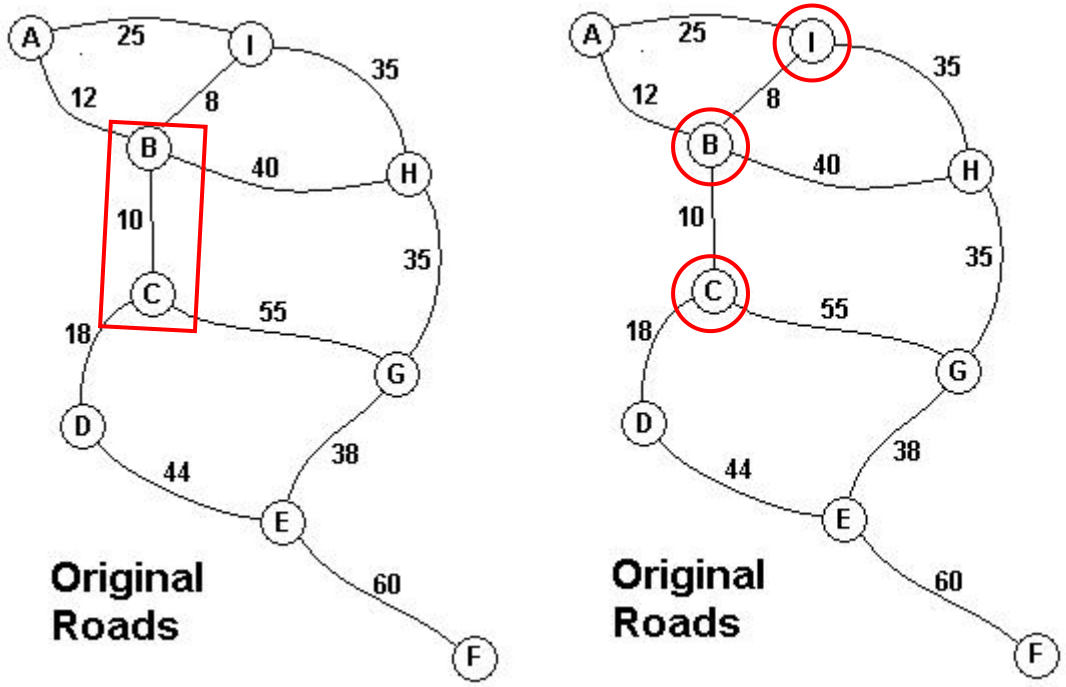
1. 首先对所有边进行排序，将n个点看做n个独立的森林。

2. 按照权从小到大选边，所选边的节点需要属于不同的森林。

3. 生成最小生成树的一条边，并将这两颗树合并作为一颗树。

丛林中的路

Kruskal算法：加边法。初始最小生成树边数为0，每迭代一次就选择一条满足条件的最小代价边，加入到最小生成树的边集合里。



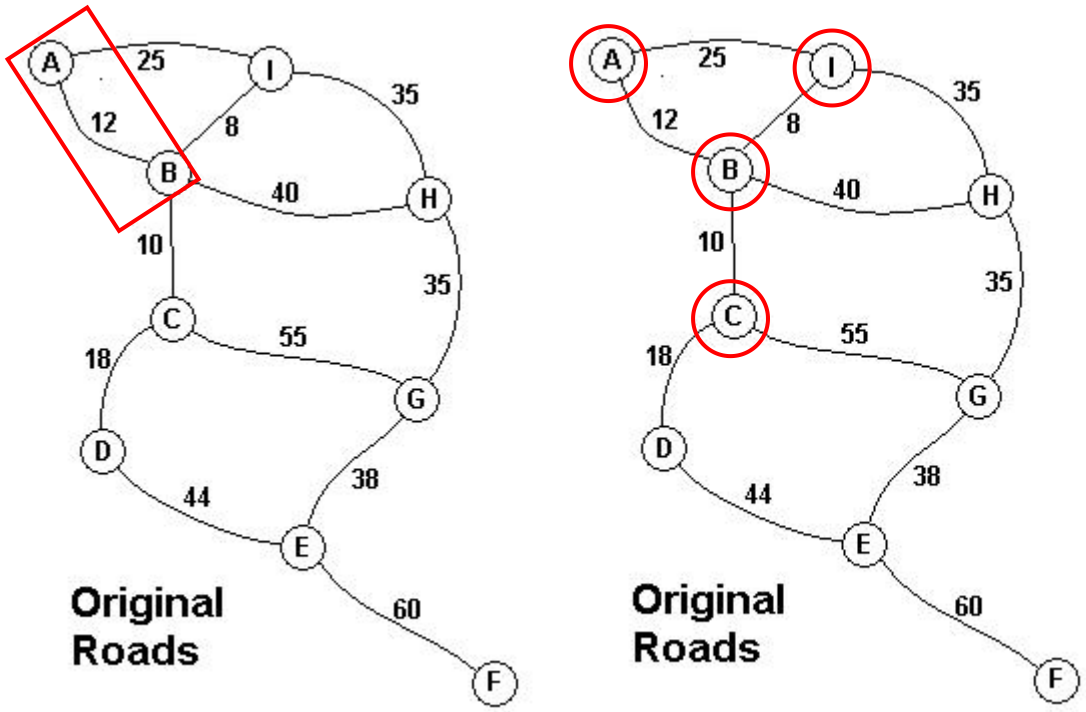
1. 首先对所有边进行排序，将n个点看做n个独立的森林。

2. 按照权从小到大选边，所选边的节点需要属于不同的树。

3. 生成最小生成树的一条边，并将这两颗树合并作为一颗树。

丛林中的路

Kruskal算法：加边法。初始最小生成树边数为0，每迭代一次就选择一条满足条件的最小代价边，加入到最小生成树的边集合里。



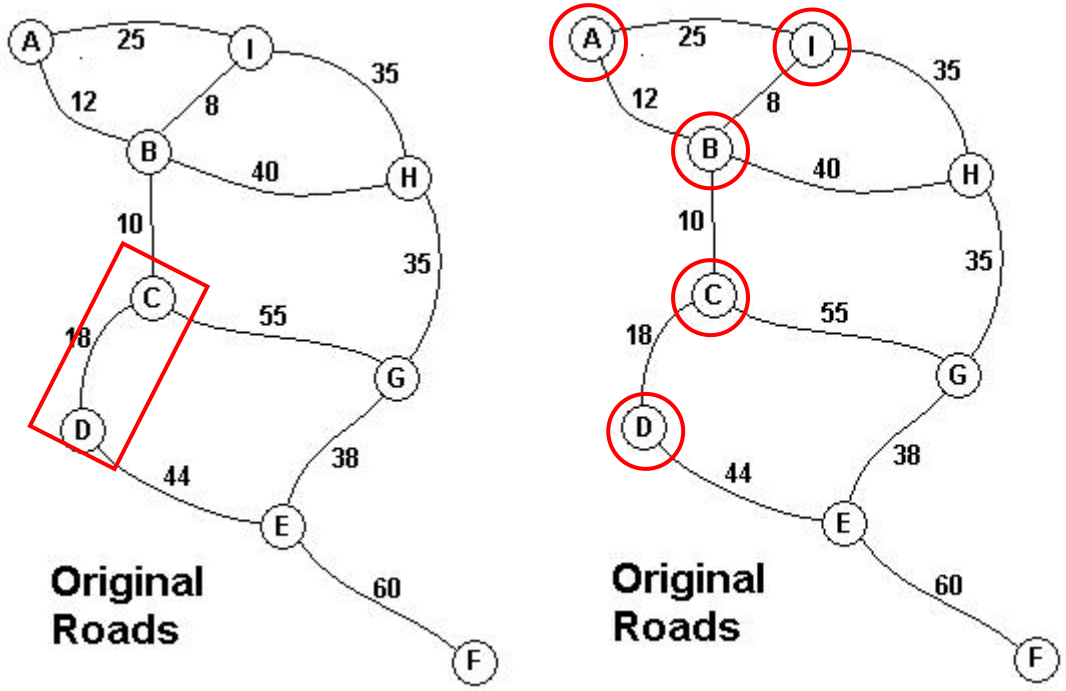
1. 首先对所有边进行排序，将n个点看做n个独立的森林。

2. 按照权从小到大选边，所选边的节点需要属于不同的树。

3. 生成最小生成树的一条边，并将这两颗树合并作为一颗树。

丛林中的路

Kruskal算法：加边法。初始最小生成树边数为0，每迭代一次就选择一条满足条件的最小代价边，加入到最小生成树的边集合里。



1. 首先对所有边进行排序，将n个点看做n个独立的森林。

2. 按照权从小到大选边，所选边的节点需要属于不同的树。

3. 生成最小生成树的一条边，并将这两颗树合并作为一颗树。

丛林中的路

```
char find(char x){                    //简单的并查集模板
    if(fa[x]==x)
        return x;
    else
        return fa[x]=find(fa[x]);
}
void _union(char x,char y){
    char father_x= find(x);
    char father_y = find(y);
    if(father_x!=father_y){
        fa[father_y] = father_x;
    }
}
```

丛林中的路

```
sort(paths,paths+m,cmp);
for(int i=0;i<m;i++){
    char x=paths[i].x;
    char y=paths[i].y;
    if(find(x)!=find(y)){
        _union(x,y);
        count++;
        ans+=paths[i].w;
    }
}
```

运输重物

在联系上起重机销售商后，小明决定成为运输业巨头，但被重量问题难倒了。小明需要考虑不同道路、桥梁的承重能力，确定是否有一个办法，把重物从运输到目的地。

幸运的是，小明已经获得了城市所有街道、桥梁和所有允许的重量的说明。不幸的是，他不知道如何找到最大重量，以便告诉他的客户最多能装多少东西。但你肯定知道。

您会收到城市的平面图，由十字路口之间的街道（有重量限制）描述，这些街道从1到n编号。您的任务是找到从十字路口1（小明所在地）到十字路口n（客户所在地）可以运输的最大重量。你可以假设至少有一条路径。所有街道都可以双向行驶。

运输重物

输入

第一行包含城市的数量。对于每个城市，第一行给出了街道交叉口的数量n（1<=n<=1000）和街道的数量m。
之后的m行包含三个整数，指定街道的起点和终点以及最大允许权重，该权重为正且不大于1000000。每对十字路口之间最多只有一条街道。

输出

每组输出占两行。第一行为"Scenario #i:"，其中i是从1开始的城市编号。
第二行为可以运送给客户的最大允许重量。
用空行终止输出。

运输重物

样例输入

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

样例输出

```
4
```

运输重物

将题目抽象：

给一个无向图，共有m条边，找出在所有从结点1到结点n的路径中，最小边的权值最大的那一条。输出这个权值。

运输重物

将题目抽象：

给一个无向图，共有m条边，找出在所有从结点1到结点n的路径中，最小边的权值最大的那一条。输出这个权值。

两种方法：最短路径，最大生成树。

运输重物

最短路径:

Dijkstra算法。每次遍历到始点距离最近且未访问过的顶点的邻接节点，直到扩展到终点为止。

“最短路径的子路径也是最短路”。维护一个数组表示当前循环中各个点到起始点的距离，则该数组中的最小值就是最小值对应的节点到起始点的最短路。

运输重物

最短路径:

Dijkstra算法。每次遍历到始点距离最近且未访问过的顶点的邻接节点，直到扩展到终点为止。

“最短路径的子路径也是最短路”。维护一个数组表示当前循环中各个点到起始点的距离，则该数组中的最小值就是最小值对应的节点到起始点的最短路。

维护每个点到起始点的最大容量。如果一个节点到起始点的容量比其余所有点到起始点的容量都大，则该点到起始点的最大容量就可以确定下来，因为增加仍和一条边都不会增加容量。只需要修改dijkstra模板，求最大值。

运输重物

```
void Dijkstra() {
    memset(vis,0,sizeof(vis));
    for(int i = 1; i<=n; i++){
        dis[i] = mapp[1][i];                    //dis初始化为到1的连边的距离
    }
    vis[1] = 1;
    for (int i = 0; i < n; i++) {
      int max_ = -inf, k = start;
      for (int j = 1; j <= n; j++)
        if (!vis[j] && dis[j] > max_){          //未确定点的最大值
          max_ = dis[j]; k = j; }
```

运输重物

```
        vis[k] = 1;
        //求每条路径的某个路段的最小承受重量
        for(int j = 1; j<=n; j++){
            dis[j] = max(dis[j],min(dis[k],mapp[k][j]));
        }
    }
}
```

运输重物

最大生成树：

生成“最小边的权值最大的那一条路径”。

生成过程中边的权值顺序从大到小的，所以只要找到通路就可以跳出，不需要生成整棵树。

运输重物

```
sort(Edges + 1, Edges + 1 + m);
for(int i=1;i<=m;i++) {
        Edge& e = Edges[i];
        int x = find(e.from);
        int y = find(e.to);
        if (find(1) != find(n)) {
                ans = e.dis;
                p[x] = y;
        }
        else break;
}
```

把他们全排了

升序排序序列是指使用某种形式的小于运算符，将元素从最小到最大排序的序列。例如，排序序列A、B、C、D意味着A<B、B<C和C<D。在这个问题中，我们将给你一组A<B形式的关系，并要求你确定是否指定了排序顺序。

输入

输入由多组数据组成。每组数据都以两个正整数n和m开始。第一个值表示要排序的对象数量，其中2<=n<=26。要排序的对象将是大写字母表的前n个字符。第二个值m表示在这组数据中给出的A<B形式的关系的数量。接下来是m行，每行包含一个由三个字符组成的关系：一个大写字母、字符“<”和第二个大写字母。任何字母都不会超出字母表前n个字母的范围。n=m=0的值表示输入结束。

把他们全排了

升序排序序列是指使用某种形式的小于运算符，将元素从最小到最大排序的序列。例如，排序序列A、B、C、D意味着A<B、B<C和C<D。在这个问题中，我们将给你一组A<B形式的关系，并要求你确定是否指定了排序顺序。

输出

对于每组数据，输出由一行组成。此行应为以下三行之一：

Sorted sequence determined after xxx relations: yyy...y.

Sorted sequence cannot be determined.

Inconsistency found after xxx relations.

其中xxx是确定排序序列或发现不一致时处理的关系数，以先到者为准，yyy。。。y是排序的升序。

把他们全排了

样例输入

```
4 6
A<B
A<C
B<C
C<D
B<D
A<B
3 2
A<B
B<A
26 1
A<Z
```

样例输出

```
Sorted sequence determined after 4 relations: ABCD.
Inconsistency found after 2 relations.
Sorted sequence cannot be determined.
```

把他们全排了

题目理解：给定一系列的关系，从给出的关系中从小到大排一个序。如果给出的关系中发生冲突，输出发生冲突的那一步；如果可以得到一个序，输出题目给出到第几个关系才能确定顺序。

把他们全排了

题目理解：给定一系列的关系，从给出的关系中从小到大排一个序。如果给出的关系中发生冲突，输出发生冲突的那一步；如果可以得到一个序，输出题目给出到第几个关系才能确定顺序。

1. 给出一个关系。

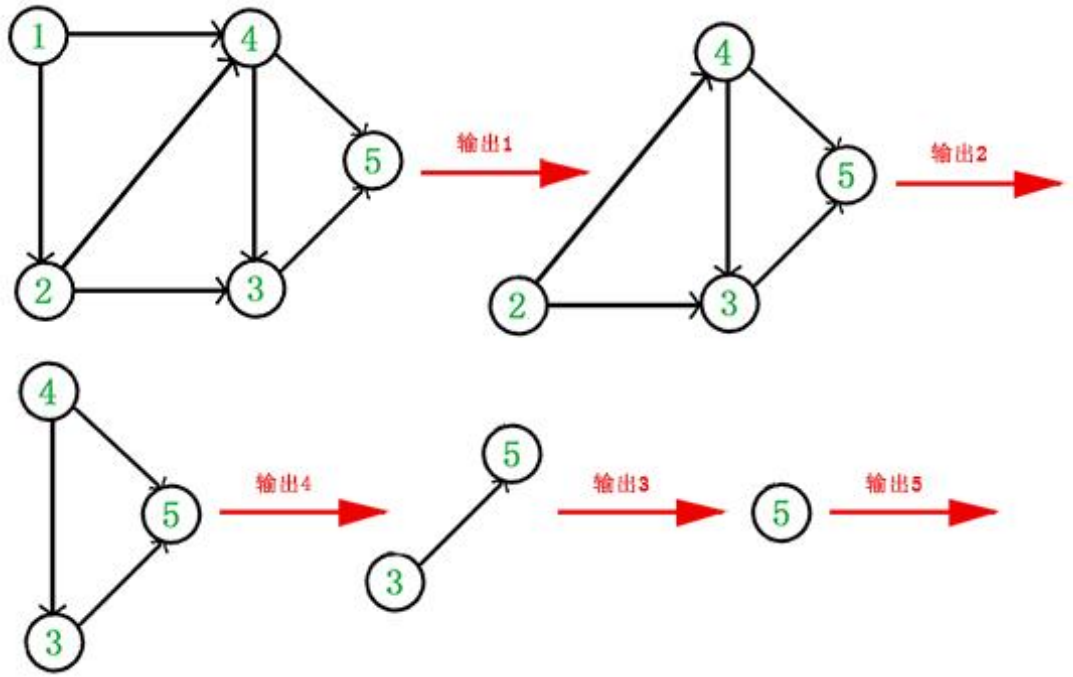
2. 拓扑排序判断是否冲突。如果题目给出的关系发生冲突，那么必然可以形成环。

3. 如果能够确定n个点的关系，那么每次只能出现一个出度/入度为0的点。

比如，递增序列，一定只有一个点没有比别的小，或者没有比任何一个别的大。

把他们全排了

题目理解：给定一系列的关系，从给出的关系中从小到大排一个序。如果给出的关系中发生冲突，输出发生冲突的那一步；如果可以得到一个序，输出题目给出到第几个关系才能确定顺序。



把他们全排了

```
int TopoSort(int n){
    int t=0,temp[27],p,m,flag=1;                    //flag=1:有序 flag=-1:不确定
    for(int i=1;i<=n;i++)temp[i]=indegree[i];//记录入度
    for(int i=1;i<=n;i++){
       m=0;
       for(int j=1;j<=n;j++)                        //查找入度为零的顶点个数
          if(temp[j]==0) {m++,p=j;}
       if(m==0)return 0;                            //有环
       if(m>1) flag=-1;                             //无序
       q[++t]=p,temp[p]=-1;                         //入度为零的点入队，减去相邻点继
       for(int j=1;j<=n;j++)if(map[p][j]==1)temp[j]--;
   }
   return flag;
}
```