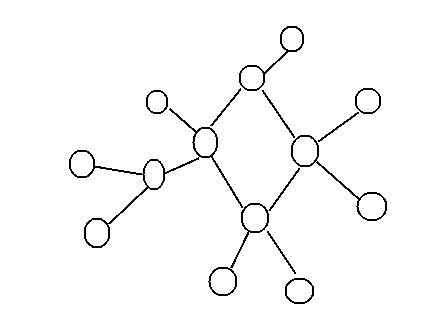
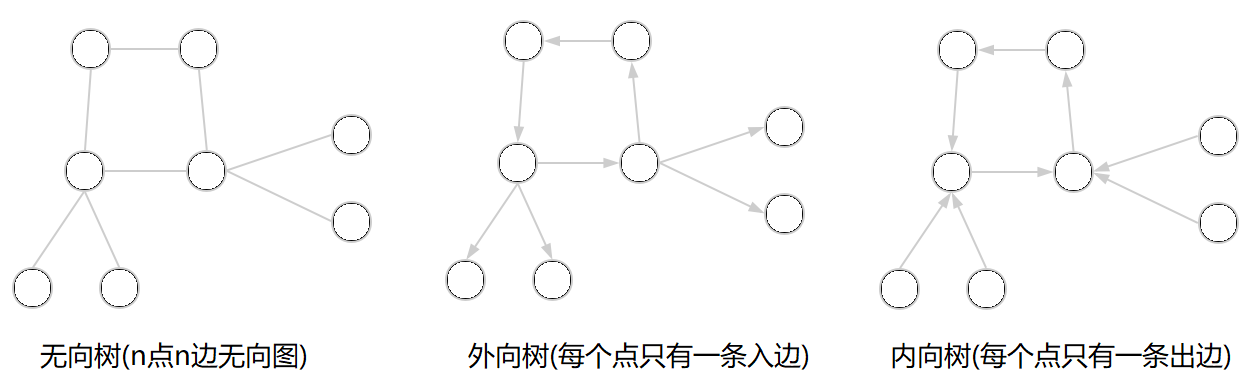
基环树

基环树（环套树），就是有n个点n条边的图，也就是有一个环的树，环上的每个点都可以看作是树根。



图

如图所示，有向基环树又分内向和外向基环树，当然也有无向的。



图

发现环的算法可以使用并查集，在输入边的同时，利用并查集判断当前两点是否已经连通，如果已经连通，那么这两点一定在环上，并且这条边也是环上的。那么以这两点分别作为起点和终点，用DFS找到起点到终点的路径，这条路径上的所有点就是环上的所有点！以下题为例：

【题目描述】发现环（loop）第八届蓝桥杯决赛

小光的实验室有N台电脑，编号1~N。原本这N台电脑之间有N－1条数据链接相连，恰好构成一个树形网络。在树形网络上，任意两台电脑之间有唯一的路径相连。

不过在最近一次维护网络时，管理员误操作使得某两台电脑之间增加了一条数据链接，于是网络中出现了环路。环路上的电脑由于两两之间不再是只有一条路径，使得这些电脑上的数据传输出现了BUG。

为了恢复正常传输。小光需要找到所有在环路上的电脑，你能帮助他吗？

【输入格式】

输入第一行包含一个整数N。

随后N行，每行两个整数a和b，表示a和b之间有一条数据链接相连。

对于30%的数据，1≤N≤1 000。

对于100%的数据，1≤N≤100 000， 1≤a, b≤N。

输入保证合法。

【输出格式】

按从小到大的顺序输出在环路上的电脑的编号，中间由一个空格分隔。

【输入样例】

5

1 2

3 1

2 4

2 5

5 3

【输出样例】

1 2 3 5

参考程序如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59 | //发现环 — 并查集算法  #include <bits/stdc++.h>  **using** **namespace** std**;**  const int MAXN **=** 100005**;**  int n**,**s**,**f**,**u**,**v**;**  int father**[**MAXN**],** vis**[**MAXN**],** ans**[**MAXN**];**  vector**<**int**>** edge**[**MAXN**];**  int Find**(**int x**)**  **{**  **return** father**[**x**]==**x **?** x **:** father**[**x**]=**Find**(**father**[**x**]);**  **}**  void DFS**(**int u**,** int index**)**  **{**  ans**[**index**]** **=** u**;**  **if(**u**==**f**)** //找到环了  **{**  sort**(**ans**,** ans**+**index**+**1**);** //排序后输出  **for(**int i**=**0**;** i**<=**index**;** i**++)**  printf**(**"%d%c"**,** ans**[**i**],** i**==**index**?**'\n'**:**' '**);**  **return;**  **}**  vis**[**u**]=**1**;**  **for(**int i**=**0**;** i**<**edge**[**u**].**size**();** i**++)**  **{**  int v**=**edge**[**u**][**i**];**  **if(!**vis**[**v**])**  DFS**(**v**,**index**+**1**);**  **}**  vis**[**u**]=**0**;**  **}**  int main**()**  **{**  **while(**scanf**(**"%d"**,** **&**n**)==**1**)**  **{**  **for(**int i**=**1**;** i**<=**n**;** i**++)**  father**[**i**]** **=** i**;**  **for(**int i**=**0**;** i**<**n**;** i**++)**  **{**  scanf**(**"%d%d"**,** **&**u**,** **&**v**);**  int ru**=**Find**(**u**);**  int rv**=**Find**(**v**);**  **if(**ru**==**rv**)**  s**=**u**,** f**=**v**;** //找到起始点和结束点,不连边  **else**  **{**  father**[**ru**]=**rv**;** //合并  edge**[**u**].**push\_back**(**v**);**  edge**[**v**].**push\_back**(**u**);**  **}**  **}**  memset**(**vis**,** 0**,** **sizeof(**vis**));**  DFS**(**s**,** 0**);**  **}**  **return** 0**;**  **}** |

还可以使用拓扑排序算法判断环，因为拓扑排序算法的核心思想是每次找入度为0的点进入输出队列，然后将与此点相连的节点入度减1……当做n－1次后还有点没进输出队列，那么这些点就是环上的。道理很简单：环上的各点入度都为1，没有入度为0的，所以就不能更新。

原始的拓扑排序算法解决的是有向无环图，拓扑出入度为0的点，但是本题的边是无向的，可以用双向边表示，显然在环上的点的入度至少为2，于是可以将拓扑出入度为0的点，变形为拓扑出入度为1的点，最后剩下的是一个各点入度为2的环。

参考程序如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | //发现环 - 拓扑排序算法  #include <bits/stdc++.h>  **using** **namespace** std**;**  vector**<**int**>**vec**[**100005**];**  queue**<**int**>**que**;**  int deg**[**100005**];** //保存各点的入度值  bool vis**[**100005**];**  int n**;**  void TopSort**()**  **{**  memset**(**vis**,true,sizeof(**vis**));**  **while(!**que**.**empty**())**  que**.**pop**();**  **for(**int i**=**1**;** i**<=**n**;** i**++)** //寻找入度为1的点  **if(**deg**[**i**]==**1**)**  que**.**push**(**i**);**  **while(!**que**.**empty**())**  **{**  int now**=**que**.**front**();**  que**.**pop**();**  vis**[**now**]=false;**  **for(**int i**=**0**;** i**<**vec**[**now**].**size**();** i**++)**  **if(--**deg**[**vec**[**now**][**i**]]** **==** 1**)**  que**.**push**(**vec**[**now**][**i**]);**  **}**  **}**  int main**()**  **{**  scanf**(**"%d"**,&**n**);**  **for(**int i**=**0**,**a**,**b**;** i**<**n**;** i**++)**  **{**  scanf**(**"%d%d"**,&**a**,&**b**);**  vec**[**a**].**push\_back**(**b**);**  vec**[**b**].**push\_back**(**a**);**  deg**[**a**]++;** //入度++  deg**[**b**]++;**  **}**  TopSort**();**  **for(**int i**=**1**;** i**<=**n**;** i**++)**  **if(**vis**[**i**])**  printf**(**"%d%c"**,**i**,**i**==**n**?**'\n'**:**' '**);**  **return** 0**;**  **}** |

【题目描述】信息传递（msg.cpp）NOIP 2015

有 n个同学（编号为 1到 n）正在玩一个信息传递的游戏。在游戏里每人都有一个固定的信息传递对象，其中，编号为 i的同学的信息传递对象是编号为Ti的同学。

游戏开始时，每人都只知道自己的生日。之后每一轮中，所有人会同时将自己当前所知的生日信息告诉各自的信息传递对象（注意：可能有人可以从若干人那里获取信息，但是每人只会把信息告诉一个人，即自己的信息传递对象）。当有人从别人口中得知自己的生日时，游戏结束。请问该游戏一共可以进行几轮？

【输入格式】

输入第一行包含1个正整数 n，表示n个人。

第二行包含 n个用空格隔开的正整数 T1，T2，⋯⋯，Tn，其中第 i个整数 Ti表示编号为i的同学的信息传递对象是编号为Ti的同学，Ti≤n且Ti≠i。

【输出格式】

输出一个整数，表示游戏一共可以进行多少轮。

【输入样例】

5

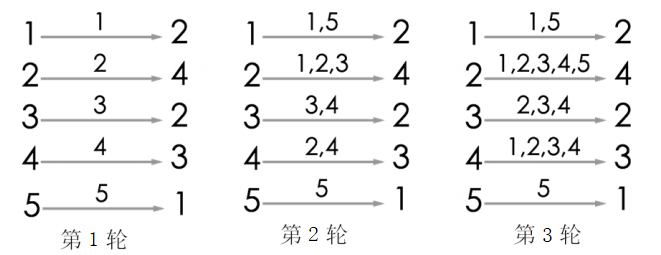
2 4 2 3 1

【输出样例】

3

【样例说明】

游戏的流程如图所示。当进行完第3轮游戏后，4号玩家会听到2号玩家告诉他自己的生日，所以答案为3。当然，第3轮游戏后，2号玩家、3号玩家都能从自己的消息来源得知自己的生日，同样符合游戏结束的条件。



图

【数据规模】

对于30%的数据，n≤200；

对于60%的数据，n≤2 500；

对于100%的数据，n≤200 000。

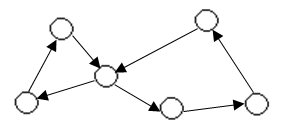
【算法分析】

将每个同学当作一个结点，将每次传递当作一条有向边，不难发现，如果想要有一个同学说出的生日重新传回自己的耳中，图中必定有一个环。

使用tarjan找强连通分量的方法也可以找到环，即在Tarjan算法里加一个堆栈，每次DFS到新结点u时将它入栈，当DFS结束时，如果该结点的dfn[u]＝low[u]，说明结点u是强连通分量的根，则对堆栈中结点u之上的结点依次出栈，因为这些结点均属于同一强连通分量。参考代码如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | //信息传递  #include <bits/stdc++.h>  **using** **namespace** std**;**  const int MAXN**=**2e5**+**10**;**  stack**<**int**>**st**;**  vector**<**int**>**vec**[**MAXN**];**  int n**,**cnt**,**idx**,**ans**=**MAXN**;**  int dfn**[**MAXN**],**low**[**MAXN**],**vis**[**MAXN**];**  void Tarjan**(**int x**)**  **{**  low**[**x**]=**dfn**[**x**]=++**idx**;**  st**.**push**(**x**);**  vis**[**x**]=**1**;**  **for(**int i**=**0**;** i**<**vec**[**x**].**size**();** i**++)**  **{**  int v**=**vec**[**x**][**i**];**  **if(!**dfn**[**v**])**  **{**  Tarjan**(**v**);**  low**[**x**]=**min**(**low**[**x**],**low**[**v**]);**  **}**  **else** **if(**vis**[**v**])**  low**[**x**]=**min**(**low**[**x**],**dfn**[**v**]);**  **}**  **if(**low**[**x**]==**dfn**[**x**])** //找到环  **{**  vis**[**x**]=**0**;**  **for(**cnt**=**1**;** st**.**top**()!=**x**;** cnt**++)**//cnt统计环中结点个数,初始为1  st**.**pop**();** //因为栈顶=x时循环结束,cnt少算一个  **if(**cnt**>**2**)** //一个结点不是环  ans**=**min**(**ans**,**cnt**);**  **}**  **}**  int main**()**  **{**  scanf**(**"%d"**,&**n**);**  **for(**int i**=**1**,**x**;** i**<=**n**;** i**++)**  **{**  scanf**(**"%d"**,&**x**);**  vec**[**i**].**push\_back**(**x**);**  **}**  **for(**int i**=**1**;** i**<=**n**;** i**++)**  **if(!**dfn**[**i**])**  Tarjan**(**i**);**  printf**(**"%d\n"**,**ans**);**  **return** 0**;**  **}** |

因为题目中每个人只会把信息告诉一个人，所以可以使用tarjan算法求最小环，但如图所示，如果每个人可以把信息告诉不止一个人的话，tarjan算法并不适用。



【题目描述】旅行（NOIP 2018）

小Y是一个爱好旅行的 OIer。她来到 X 国，打算将各个城市都玩一遍。

小Y了解到，X国的n个城市之间有m条双向道路。每条双向道路连接两个城市。不存在两条连接同一对城市的道路，也不存在一条连接一个城市和它本身的道路。并且，从任意一个城市出发，通过这些道路都可以到达任意一个其他城市。小Y只能通过这些道路从一个城市前往另一个城市。

小Y的旅行方案是这样的：任意选定一个城市作为起点，然后从起点开始，每次可以选择一条与当前城市相连的道路，走向一个没有去过的城市，或者沿着第一次访问该城市时经过的道路后退到上一个城市。当小 Y 回到起点时，她可以选择结束这次旅行或继续旅行。需要注意的是，小Y要求在旅行方案中，每个城市都被访问到。

为了让自己的旅行更有意义，小Y决定在每到达一个新的城市（包括起点）时，将它的编号记录下来。她知道这样会形成一个长度为n的序列。她希望这个序列的字典序最小，你能帮帮她吗？对于两个长度均为 n的序列A和B，当且仅当存在一个正整数 x，满足以下条件时，我们说序列 A的字典序小于B。

对于任意正整数1≤i < x1≤i<x，序列A的第 i个元素 A\_i和序列 B的第 i个元素 B\_i相同。

序列 A的第 x个元素的值小于序列 B的第 x个元素的值。

【输入格式】

输入文件共 m + 1行。第一行包含两个整数 n，m(m≤n)，中间用一个空格分隔。

接下来 m 行，每行包含两个整数 u,v (1 ≤ u，v ≤ n)，表示编号为 u和 v的城市之间有一条道路，两个整数之间用一个空格分隔。

【输出格式】

输出文件包含一行，n个整数，表示字典序最小的序列。相邻两个整数之间用一个空格分隔。

【输入样例1】

6 5

1 3

2 3

2 5

3 4

4 6

【输出样例1】

1 3 2 5 4 6

【输入样例2】

6 6

1 3

2 3

2 5

3 4

4 5

4 6

【输出样例2】

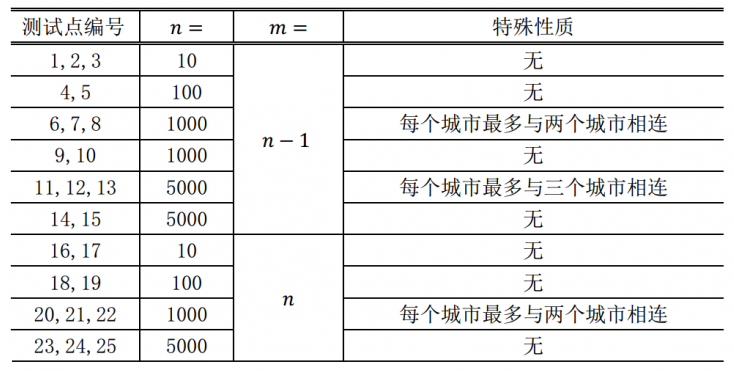
1 3 2 4 5 6 （行末可以有一个空格）

【数据规模】

对于100%的数据和所有样例，1≤n≤5000 且 m = n − 1或 m = n。

【算法分析】

对于不同的测试点， 我们约定数据的规模如下：



【算法分析】

首先要保证60分做法正确，即m＝n－1。因为是一棵树并要求字典序最小，那么从1出发，按照字典序的大小依次DFS访问节点并记录输出即可。

参考程序如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | //旅行  #include <bits/stdc++.h>  using namespace std;  int n,m,ans[5005],sum=1,x,y;  vector <int> v[5005];  void Dfs(int u,int father)  {  if(sum>n) //从1出发，所以sum事先已初始为1了  return;  ans[sum]=u;  for(int i=0; i<v[u].size(); i++)  if(v[u][i]!=father)  {  sum++;  Dfs(v[u][i],u);  }  }  int main()  {  scanf("%d %d",&n,&m);  for(int i=1; i<=m; i++)  {  scanf("%d %d",&x,&y);  v[x].push\_back(y);  v[y].push\_back(x);  }  for(int i=1; i<=n; i++)  sort(v[i].begin(),v[i].end());//字典序最小，所以先排序  Dfs(1,0);  for(int i=1; i<=n; i++)  printf("%d ",ans[i]);  return 0;  } |

再考虑m =n的情况，显然这是一个基环树（环套树），这只要枚举删除一条边，再按跑一遍DFS就可以了，因为如果一个图里有环的话（在没有重边自环的情况下肯定有环），这一个环里肯定有一条边是用不到的，显然，找到树上的环，枚举环上的边即可（枚举所有边也可以，因为n≤5000，但要对代码进行优化，例如每DFS一次，如果大于之前找到的最小值，则剪枝）。

枚举所有边的参考代码如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105 | //旅行-剪枝优化  #include <bits/stdc++.h>  using namespace std;  const int MAXN = 100010;  const int INF = 2140000000;  int DelA, DelB;//删除边的两个节点  bool looptree = false;//判断是否基环树  struct map  {  int c;  int nxt;  bool used;  } e[2\*5010];  int head[5010];  bool visited[5010];  int minans[5010];//即最小字典序  int ans[5010];//即当前搜索的字典序  int cntans = 0, maxcntans = 0;//cntans:搜索到的结点数  int cnt = 0;  int n, m;  bool better() //比较ans是否大于minans  {  for (int i = 1; i <= cntans; i++)  {  if (ans[i] > minans[i])  return false;  if (ans[i] < minans[i])  return true;  }  return true;  }  void DFS(int sur)  {  visited[sur] = true; //设置该节点已访问过  ans[++cntans] = sur; //添加节点到答案中  if (looptree && cntans <= maxcntans && !better())//剪枝  return;  priority\_queue <int, vector<int>, greater<int> >q;//优先队列存从小到大DFS  for (int i = head[sur]; i; i = e[i].nxt)  {  if (visited[e[i].c]||(e[i].c==DelB && sur==DelA)|| (e[i].c==DelA && sur==DelB))  continue; //边已被访问过或者已被删除则忽略  q.push(e[i].c);  }  while (!q.empty())  {  DFS(q.top());  q.pop();  }  }  void AddE(int x, int y) //前向星存边  {  cnt++;  e[cnt].c = y;  e[cnt].nxt = head[x];  head[x] = cnt;  }  int main()  {  freopen("travel.in","r",stdin);  freopen("travel.out","w",stdout);  int x, y;  scanf("%d%d", &n, &m);  for (int i = 1; i <= m; i++)  {  scanf("%d%d", &x, &y);  AddE(x, y);//前向星存储,vector会超时  AddE(y, x);  }  int sur = 1;  if (m == n - 1)//是树的情况  {  cntans = 0;  DelA = -1;  DelB = -1;  DFS(1);  for (int i = 1; i <= cntans; i++)  i==cntans?printf("%d\n",ans[i]):printf("%d ",ans[i]);  }  else //基环树的情况  {  looptree = true; //设为基环树  for (int i = 1; i <= 2\*m; i+=2)  {  memset(visited, 0, sizeof(visited));  cntans = 0;  DelA = e[i+1].c;//选出删除的边的结点1  DelB = e[i].c; //选出删除的边的结点2  DFS(sur);  if (cntans>maxcntans || (cntans>=maxcntans && better()))//更新最优解  {  memcpy(minans, ans, sizeof(ans));//复制ans到minans  maxcntans = cntans;  }  }  for (int i = 1; i <= maxcntans; i++)  i==maxcntans?printf("%d\n",minans[i]):printf("%d ",minans[i]);  }  return 0;  } |

所以旅行一题使用tarjan算法的参考代码如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109 | //旅行-tarjan算法找环  #include <bits/stdc++.h>  using namespace std;  #define M 5010  vector<int> v[M];  int n,m,num,sum,cnt,top;  int low[M],dfn[M],circle[M],ans[M],an[M],Stack[M];  bool vis[M],ok;  int DFS(int w,int fa)  {  an[++num]=w;  for(int i=0; i<v[w].size(); i++)  if(!(v[w][i]==fa ||v[w][i]==0 ))  DFS(v[w][i],w);  }  int tarjan(int u,int father)  {  if(ok)return 0;  low[u]=dfn[u]=++cnt;  Stack[++top]=u;  vis[u]=true;  for(int i=0; i<v[u].size(); i++)  {  if(v[u][i]==father || v[u][i]==0)continue;  if(!dfn[v[u][i]])  {  tarjan(v[u][i],u);  low[u]=min(low[u],low[v[u][i]]);  }  else low[u]=min(low[u],dfn[v[u][i]]);  if(ok)return 0;  }  if(dfn[u]==low[u])  {  int j;  sum=0;  do  {  j=Stack[top--];  vis[j]=0;  circle[++sum]=j;  }  while(j!=u);  if(sum>1)ok=true;  }  }  int main()  {  scanf("%d%d",&n,&m);  for(int i=1; i<=m; i++)  {  int s1,s2;  scanf("%d%d",&s1,&s2);  v[s1].push\_back(s2);  v[s2].push\_back(s1);  }  for(int i=1; i<=n; i++)  sort(v[i].begin(),v[i].end());  if(m!=n)  {  DFS(1,0);  for(int i=1; i<=n; i++)  printf("%d%c",an[i],i==n?'\n':' ');  }  else  {  memset(ans,127,sizeof(ans));  ok=false;  tarjan(1,0);//寻找环  for(int i=1; i<=sum; i++)  {  num=0;  int w1,w2,a=circle[i],b=circle[i%sum+1];  for(int j=0; j<v[a].size(); j++)  if(v[a][j]==b)  {  w1=j;  v[a][j]=0;//删边  break;  }  for(int j=0; j<v[b].size(); j++)  if(v[b][j]==a)  {  w2=j;  v[b][j]=0;//删边  break;  }  DFS(1,0);//遍历  v[a][w1]=b;//恢复删除的边  v[b][w2]=a;//恢复删除的边  for(int j=1; j<=n; j++)  {  if(ans[j]<an[j])break;  if(ans[j]>an[j])  {  for(int k=j; k<=n; k++)  ans[k]=an[k];  break;  }  }  }  for(int i=1; i<=n; i++)  printf("%d%c",ans[i],i==n?'\n':' ');  }  return 0;  } |