图论中的圈与块

绍兴县柯桥中学 黄劲松

基本概念

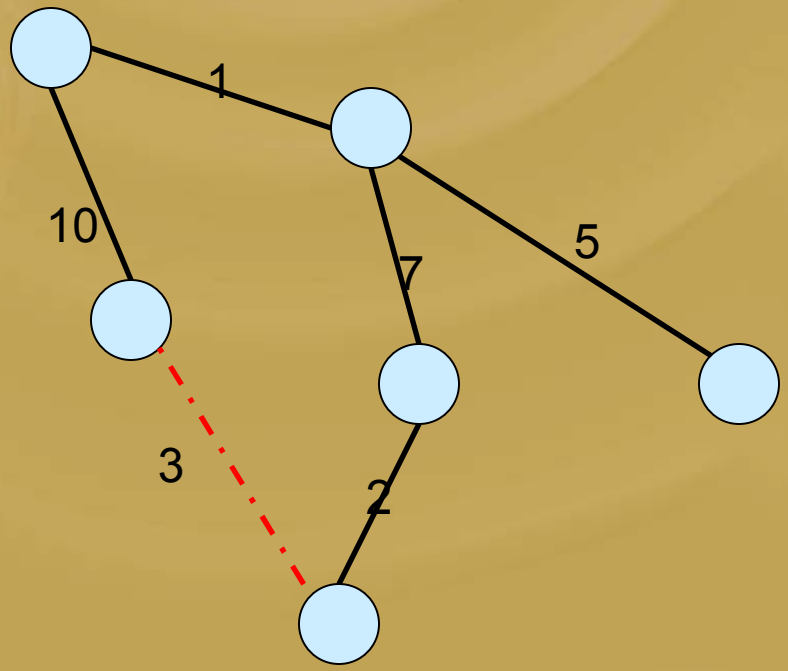
- 圈(环)
- 割点
- 割边(桥)
- 块
- 强连通子图(强连通分量(支,块))

圈及其相关知识

- MST(最小生成树) 另类算法
- 最小环问题

MST 另类算法

- 任意构造一棵原图的生成树，然后不断的添边，并删除新生成的环上的最大边。



算法证明？

水管局长 (1)

- 给定一张带权无向连通图，定义 max(p) 为路径 p 上的最大边， min(u,v) 为连接 u 和 v 的所有路径中， max(p) 的最小值。动态的做如下两个操作：
  - 1：询问某两个点之间的 min(u,v)
  - 2：删除一条边
- 你的任务是对于每个询问，输出 min(u,v) 的值。 (WC2006)

水管局长 (2)

- 数据范围约定
  - 结点个数 N≤1000
  - 图中的边数 M≤100000
  - 询问次数 Q≤100000
  - 删边次数 D≤5000

水管局长 (3)

- 根据 kruskal 算法可以知道，最小生成树上的连接两点之间的唯一路径一定是最大边最小的
- 那么，只要维护一棵图的最小生成树，那么就可以在 O(N) 的时间内回答每一个 min(u,v) 的询问
- 不断的删边然后维护最小生成树？

水管局长 (4)

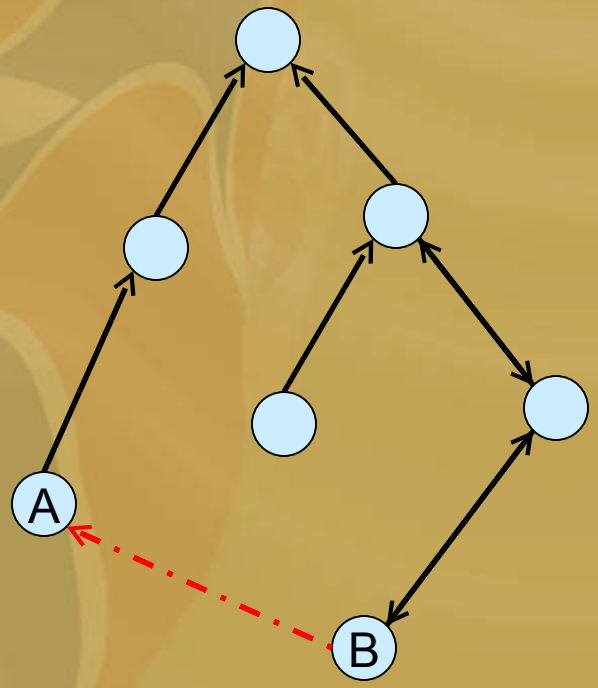
- 通过删边的形式我们似乎很难维护一张图的最小生成树
- 根据刚才提到的 MST 的另类做法，我们反向处理它的每个操作，也就是先删除所有要删的边，然后再逆向添边并回答 min(u,v)
- 于是该问题就可以用另类 MST 算法解决了

水管局长 (5)

- 这里涉及到一些图与树的存储操作，如何在 O(N) 的时间内找到环上最大边，并维护一棵最小生成树呢？
- 如果采取邻接表的存储方式来记录一棵最小生成树，从添加的边的某个点开始遍历整棵树，寻找出环上的最大边，虽然理论复杂度是 O(N) 的，但是有很多的冗余

水管局长 (6)

- 这里我们采取父亲表示法来存储一棵最小生成树，如图所示：



如果被删边在 A(V,AB) 和 B(V,AB) 的最近公共祖先 } 的那条路径上，则定义 AB 的方向为 B->A，即 A 是 B 的父亲，并将被删边到 B 的这条路径上的所有边反向 (同理可得被删边在 A 到 LCA(A,B) 的那条路径上的情况)

小 H 的聚会 (1)

- 给定每个节点的度限制，求在满足所有度限制的条件下的最大生成树。 (NOI2005)
- 这是一道提交答案式的题目，对于后面的几个较大的数据，用另类 MST 算法对你的解进行调整也能取得不错的效果！

最小环问题

- 虽然涉及到要求最小环的题目并不多(Ural1004 Sightseeing trip)，但是下面介绍的一些求最小环的算法也会对你有一定的启示意义
  - 有向带权图的最小环问题 (直接用 floyd 算法可解)
  - 无向带权图的最小环问题

朴素算法

- 令 e(u,v) 表示 u 和 v 之间的连边，再令 min(u,v) 表示，删除 u 和 v 之间的连边之后， u 和 v 之间的最短路
- 最小环则是 min(u,v) + e(u,v)
- 时间复杂度是 EV^2

一个错误的算法

- 预处理出任意两点之间的最短路径，记作 min(u,v)
- 枚举三个点 w,u,v ，最小环则是 min(u,w) + min(w,v) + e(u,v) 的最小值
- 如果考虑 min(u,w) 包含边 u-v 的情况？
- 讨论：是否有解决的方法？

改进算法

- 在 floyd 的同时，顺便算出最小环

```
g[i][j]=i,j 之间的边长
dist:=g;
for k:=1 to n do
begin
    for i:=1 to k-1 do
        for j:=i+1 to k-1 do
            answer:=min(answer,dist[i][j]+g[i][k]+g[k][j]);
    for i:=1 to n do
        for j:=1 to n do
            dist[i][j]:=min(dist[i][j],dist[i][k]+dist[k][j]);      算法证明？
end;
```

块及其相关知识

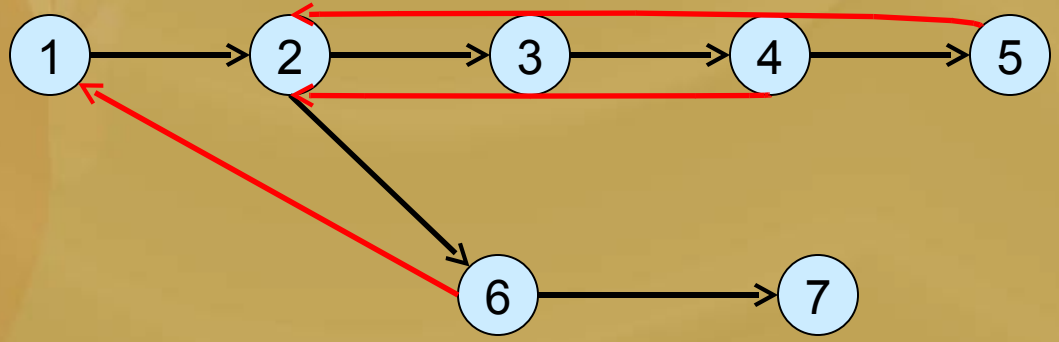
- DFS 算法
- 割点 （一般对于无向图而言）
- 割边 （一般对于无向图而言）
- 块 (一般对于无向图而言)
- 强连通子图 (一般对于有向图而言)

DFS 算法

- 1973 年， **Hopcroft** 和 **Tarjan** 设计了一个有效的 DFS 算法
- PROCEDURE DFS(v);
- begin
- inc(sign);
- dfn[v] := sign; // 给 v 按照访问顺序的先后标号为 sign
- for 寻找一个 v 的相邻节点 u
- if 边 uv 没有被标记过 then
- begin
- 标记边 uv;
- 给边定向 v→u;
- 如果 u 被标记过，记 uv 为父子边，否则记 uv 为返祖边
- if u 未被标记 then DFS(u);
- end;
- end;

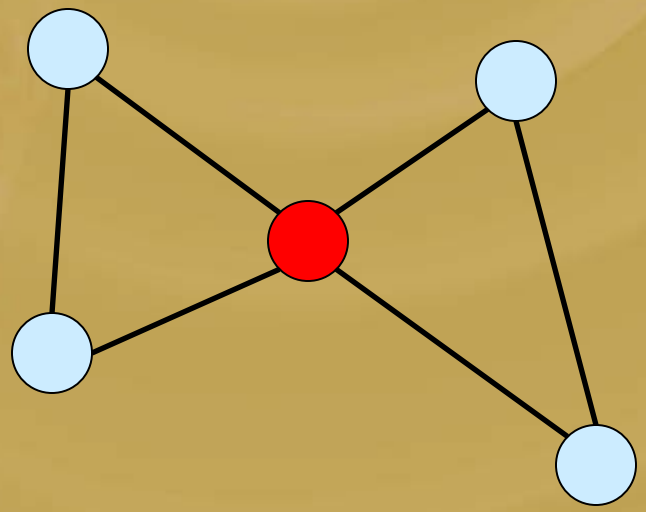
DFS 算法

- 父子边用黑色标记，返祖边用红色标记
- 如下图，除掉返祖边之后，我们可以把它看作一棵 DFS 树



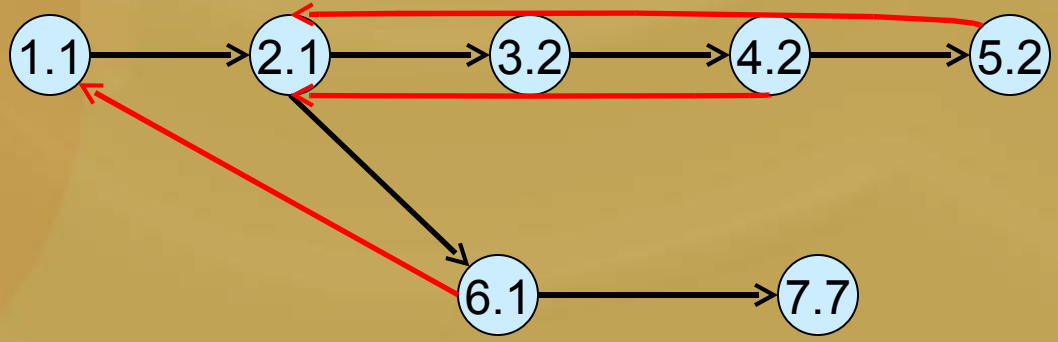
割点

- G 是连通图， $v \in V(G)$， $G - v$ 不再连通，则称 v 是 G 的割顶。



求割点的算法

- 我们通过 DFS 把无向图定向成有向图，定义每个顶的一个 lowlink 参数， lowlink[v] 表示沿 v 出发的有向轨能够到达的点 u 中， dfn[u] 的值的最小值。 (经过返祖边后则停止)



三个定理

- 定理 1： DFS 中， e=ab 是返祖边，那么要么 a 是 b 的祖先，要么 a 是 b 的后代子孙。

证明？

- 定理 2： DFS 中， e=uv 是父子边，且 dfn[u]>1 ， lowlink[v]≥dfn[u] ，则 u 是割点。

证明？

- 定理 3： DFS 的根 r 是割点的充要条件是：至少有 2 条以 r 为尾 (从 r 出发) 的父子边

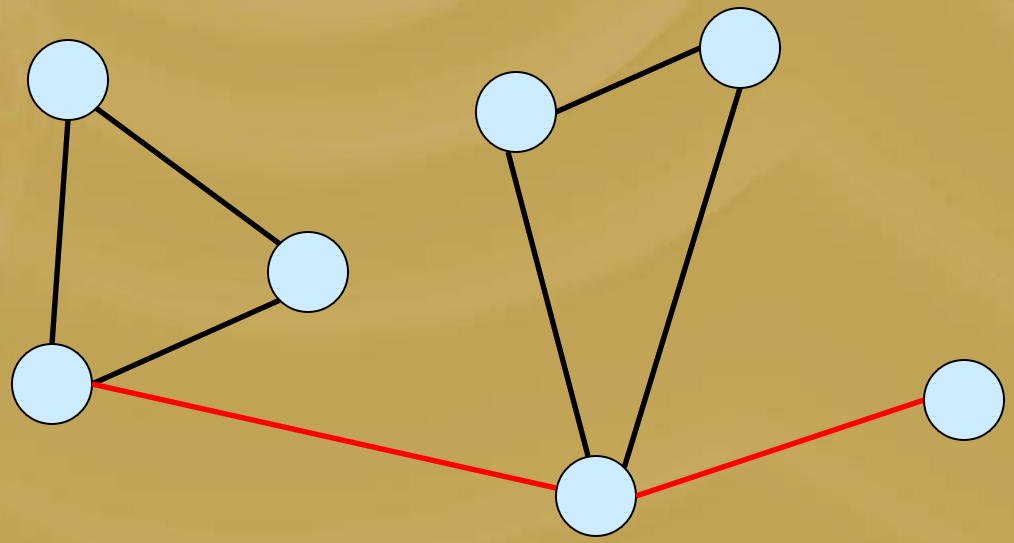
证明？

程序代码

```
PROCEDURE DFS(v);
begin
        inc(sign); dfn[v] := sign; // 给 v 按照访问顺序的先后标号为 sign
        lowlink[v] := sign; // 给 lowlink[v] 赋初始值
        for  寻找一个 v 的相邻节点 u
                if  边 uv 没有被标记过   then
                begin
                        标记边 uv;
                        给边定向 v→u;
                        if u 未被标记过   then
                        begin
                                DFS(u); //uv 是父子边，递归访问
                                lowlink[v] := min(lowlink[v],lowlink[u]);
                                if lowlink[u] >= dfn[v] then v 是割点
                        end
                        else    lowlink[v] := min(lowlink[v],dfn[u]); //uv 是返祖边
        end;
end;
```

割边

- G 是连通图， $e \in E(G)$， $G - e$ 不再连通，则称 e 是 G 的割边，亦称做桥。



求割边的算法

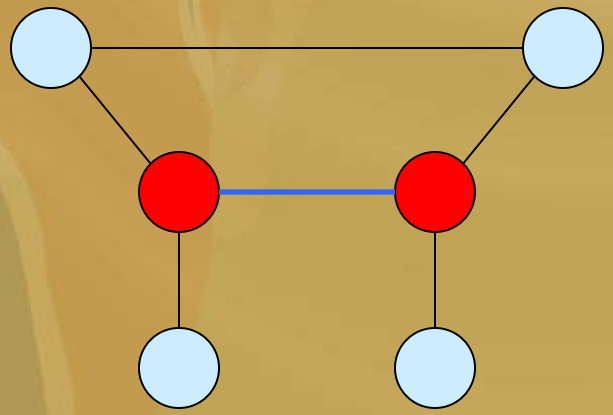
- 与割点类似的，我们定义 lowlink 和 dfn 。父子边 e=u→v ，当且仅当 lowlink[v] > dfn[u] 的时候， e 是割边。
- 我们可以根据割点算法的证明类似的证明割边算法的正确性。

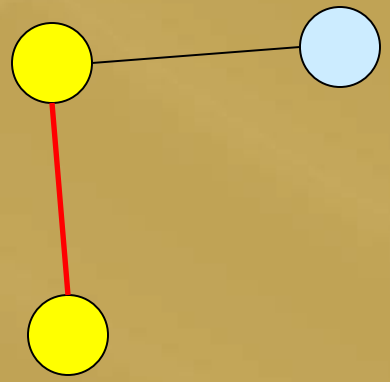
程序代码

```
PROCEDURE DFS(v);
begin
        inc(sign); dfn[v] := sign; // 给 v 按照访问顺序的先后标号为 sign
        lowlink[v] := sign; // 给 lowlink[v] 赋初始值
        for  寻找一个 v 的相邻节点 u
                if  边 uv 没有被标记过   then
                begin
                        标记边 uv;
                        给边定向 v→u;
                        if u 未被标记过   then
                        begin
                                DFS(u); //uv 是父子边，递归访问
                                lowlink[v] := min(lowlink[v],lowlink[u]);
                                if lowlink[u] > dfn[v] then vu 是割边
                        end
                        else    lowlink[v] := min(lowlink[v],dfn[u]); //uv 是返祖边
                end;
end;
```

割点与割边

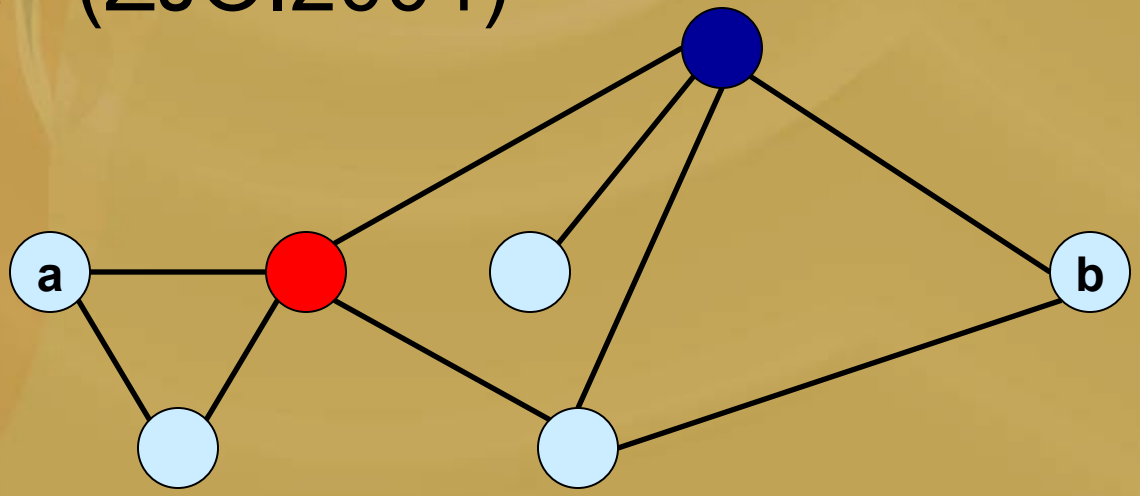
- 猜想：两个割点之间的边是否是割边？割边的两个端点是否是割点？
- 都错！





嗅探器 (1)

- 在无向图中寻找出所有的满足下面条件的点：割掉这个点之后，能够使得一开始给定的两个点 a 和 b 不连通，割掉的点不能是 a 或者 b 。 (ZJOI2004)



嗅探器 (2)

- 数据范围约定
  - 结点个数 N≤100
  - 边数 M≤N*(N-1)/2

嗅探器 (3)

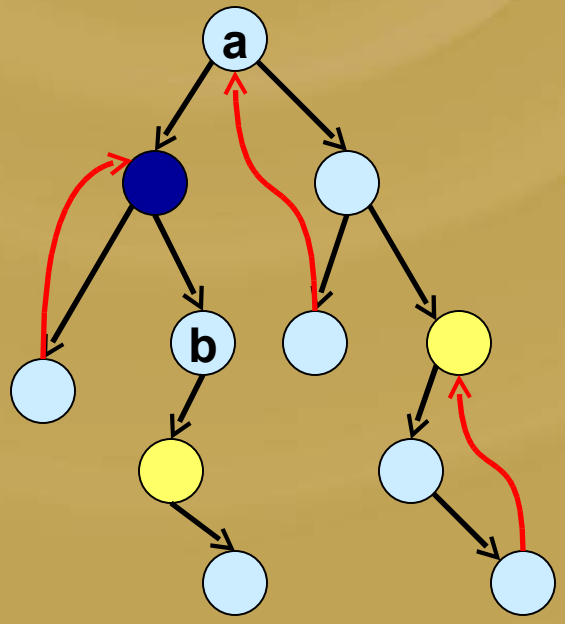
- 朴素算法：
- 枚举每个点，删除它，然后判断 a 和 b 是否连通，时间复杂度 O(NM)
- 如果数据范围扩大，该算法就失败了！

嗅探器 (4)

- 题目要求的点一定是图中的割点，但是图中的割点不一定题目要求的点。如上图中的蓝色点，它虽然是图中的割点，但是割掉它之后却不能使 a 和 b 不连通
- 由于 a 点肯定不是我们所求的点，所以可以以 a 为根开始 DFS 遍历整张图。
- 对于生成的 DFS 树，如果点 v 是割点，如果以他为根的子树中存在点 b，那么该点是问题所求的点。

嗅探器 (5)

- 时间复杂度是 O(M) 的
- 如图，蓝色的点表示问题的答案，黄色的点虽然是图的割点，但却不是问题要求的答案



关键网线 (1)

- 无向连通图中，某些点具有 A 属性，某些点具有 B 属性。请问哪些边割掉之后能够使得某个连通区域内没有 A 属性的点或者没有 B 属性的点。 (CEOI2005)
- 数据范围约定
  - 结点个数 N≤100000
  - 边数 M≤1000000

关键网线 (2)

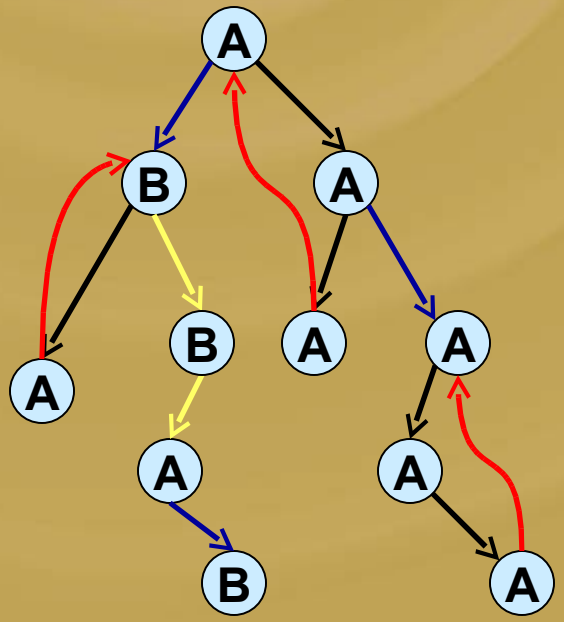
- 朴素算法：
- 枚举每条边，删除它，然后判断是否有独立出来的连通区域内没有 A 属性或者没有 B 属性。复杂度 $O(M^2)$
- 当然，这个复杂度太大了！

关键网线 (3)

- 正如嗅探器一样，题目要求的边一定是原图中的割边，但是原图中的割边却不一定是题目中要求的边。
- 设 A 种属性总共有 SUMA 个， B 中属性总共有 SUMB 个。和嗅探器类似的，如果边 e=u→v 是割边，且以 v 为根的子树中， A 种属性的数目为 0 或者为 SUMA ，或者 B 种属性的数目为 0 或者为 SUMB ，那么 e 就是题目要求的边。

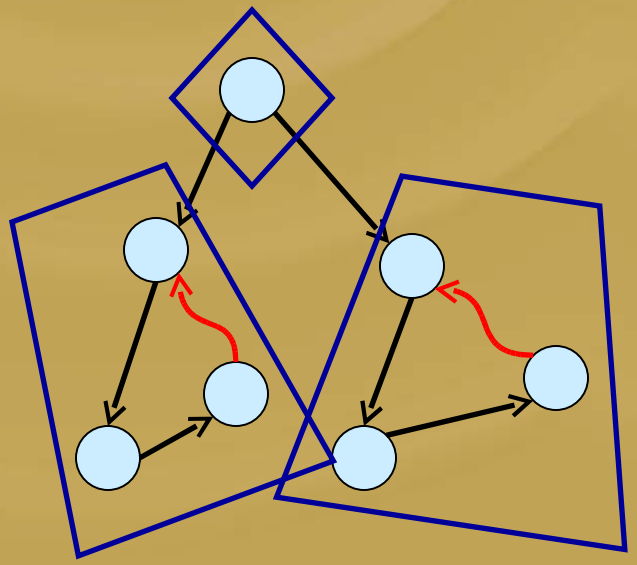
关键网线 (4)

- 下图中，蓝色的边表示题目要求的边，黄色的边表示虽然是图中的割边，但不是题目要求的边。



块

- 没有割点的图叫 2- 连通图，亦称做块， G 中成块的极大子图叫做 G 的块。把每个块收缩成一个点，就得到一棵树，它的边就是桥。



求块的算法

- 在求割点的算法中，当结点 u 的所有邻边都被访问过之后，如果 lowlink[u]=dfn[u]，我们把 u 下方的整块和 u 导出作为图中的一个块。
- 这里需要用一个栈来表示哪些元素是 u 代表的块。

程序代码

```
PROCEDURE DFS(v);
begin
        inc(sign); dfn[v] := sign; // 给 v 按照访问顺序的先后标号为 sign
        lowlink[v] := sign; // 给 lowlink[v] 赋初始值
        inc(tot); stack[tot] := v; //v 点进栈
        for  寻找一个 v 的相邻节点 u
                if  边 uv 没有被标记过   then
                begin
                        标记边 uv;
                        给边定向 v→u;
                        if u 未被标记过   then
                        begin
                                DFS(u); //uv 是父子边，递归访问
```

程序代码

```
                    lowlink[v] := min(lowlink[v],lowlink[u]);
                end
                else    lowlink[v] := min(lowlink[v],dfn[u]); //uv 是返祖边
            end;
        if lowlink[v] = dfn[v] then
        begin
            块数目 number+1;
            repeat
                标记 stack[tot] 这个点为 number;
                dec(tot); //  点出栈
            until stack[tot+1] = v;
        end;
    end;
```

新修公路 (1)

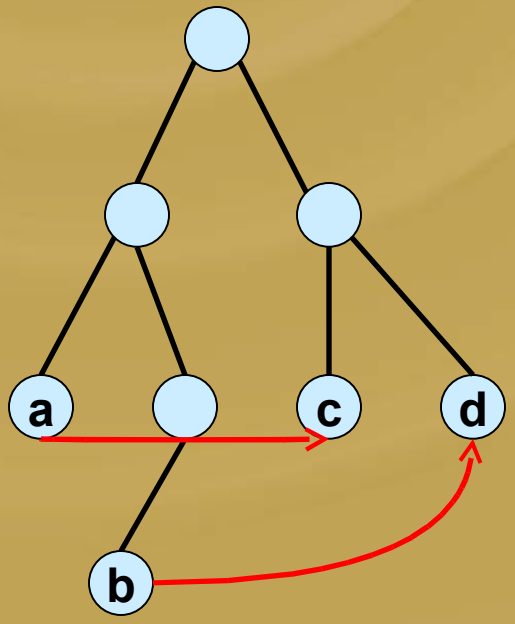
- 给出一张简单无向图，问最少添加几条边能够使得原图中没有割边。 (CEOI2000)
- 数据范围约定
  - 结点个数 N≤2500
  - 边数 M≤20000

新修公路 (2)

- 为了简化数据关系，我们先将原图收缩，变成一棵树，容易知道的是，剩下的任务就是添最少的边，使得树成为一个块。(树中的两个结点之间连边相当于原图中两个块中分别任意取点连在一起)
- **猜想**：每添一条边，就选择树中的两个叶子结点，将它们连起来，于是最少的添边数目就是 (叶子结点个数 +1)/2

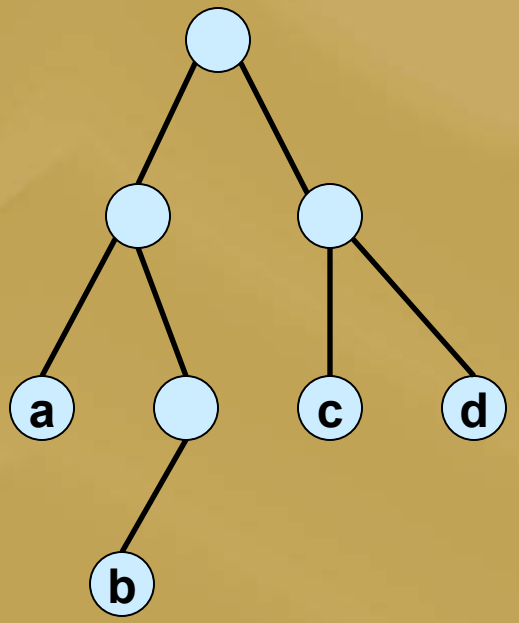
新修公路 (3)

- 如图所示，点代表了原图中的一个块，它们之间的连边是割边。连接 a 与 c， b 与 d 之后，图中就没有割边了。



新修公路 (4)

- 但并不是任意连接两个叶子结点就可以达到目标。假如连接了 a 与 b，c 与 d，原图并没有变成一个块。

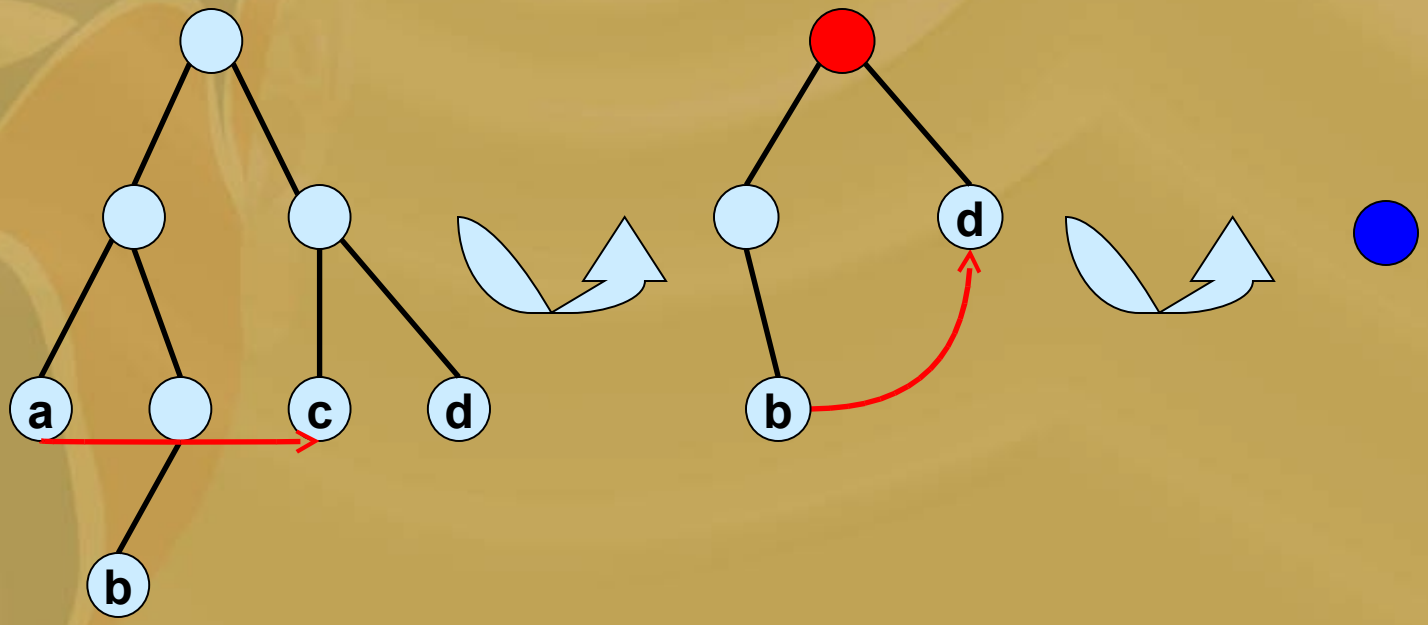


新修公路 (5)

- 进一步分析刚才的算法，每次连接两个叶子结点之后，把新生成的圈压缩成为一个点，以前和圈上的点关联的点，都和新生成的这个“压缩点”相关联。于是原来的树在添加一条边之后，又变回了一棵树。

新修公路 (6)

- 在连接 a 与 c 之后，新生成的树只剩下 2 个叶子结点；连接 b 与 d 之后，树就被压缩成了一个点。



新修公路 (7)

- 而如果先连接 a 与 b，那么新生成的树会剩下 3 个叶子结点，连接 c 与 d 之后，树中还剩 2 个叶子结点，所以这种连接方法还需要多连一条边。
- 现在的问题是，是否一定能找出这样子的两个叶子结点，使得压缩成的点不会成为新的叶子节点呢？

新修公路 (8)

- 连接的两个点的那条树中的唯一路径上，如果除了它们的最近公共祖先到自己的父亲有连边以外，其他的结点没有别的分叉，那么连接这两个点之后缩圈得到的点将会是一个叶子结点。
- 假设图中的任意两个叶子连接之后，都会多产生一个叶子结点。
- 当图中的叶子结点是 2 个或者 3 个的时候，怎么连都没有区别。

新修公路 (9)

- 当图中的叶子结点有 4 个的时候， a 和 b 到它们的最近公共祖先都没有别的分叉，且 c 和 d 到它们的最近公共祖先没有别的分叉，可以知道， a 和 c 到它们的最近公共祖先上一定有分叉。
- 这个与假设矛盾。所以我们总能找到两个叶子结点，使得它们连边之后缩成的树不会新产生叶子结点。

新修公路 (10)

- 具体实现：
  - 首先一个问题是会碰到图的压缩，一个简单易行的方法是，新建一棵树来表示压缩过之后的图。
  - 接着还会碰到一个缩圈的问题，怎么实现这一个环节？是否需要重新建树？
  - 可以采取标号法，当缩一个圈的时候，在圈上取一个代表点，并把其他的点都标记为该代表点。一个潜在的问题是，压缩成的点可能还会被再次压缩，那么标记的时候就比较麻烦了。所以这里可以用并查集来实现标号这一步。

新修公路 (11)

- 算法流程：
  - (1) 求出图中的所有块，建立一棵代表树
  - (2) 挑出 2 个叶子结点，使得连接他们之间的唯一路径上的分叉数目最多
  - (3) 连接这两个叶子结点，并压缩新生成的圈，得到一棵新的树
  - (4) 如果树中剩下一个叶子结点和一个根结点，直接连接它们，算法结束；如果树已经成为一个点，算法结束，否则转 (2)

有向图的 DFS

- 有向图的 DFS 与无向图的 DFS 的区别在于搜索只能顺边的方向进行，所以有向图的 DFS 不止一个根，因为从某个结点开始不一定就能走完所有的点。
- 另外，有向图的 DFS 除了产生父子边和返祖边以外，还会有横叉边。我们这样定义它：
- u 和 v 在已形成的 DFS 森林中没有直系上下关系，并且有 dfn[v]>dfn[u]，则称 e=uv 是横叉边。**注意，没有 dfn[v]<dfn[u] 这种横叉边。**

连通与强连通图

- 定义：
  - 将所有有向边改为无向边，如果该无向图是连通的，那么原有向图也称之为连通图。
  - 对于图中的任意两个点 A 和 B，同时存在一条从 A 到 B 的路径和一条从 B 到 A 的路径，则称该图为强连通图。
- 对于一个连通的无向图，他是一个强连通图，这里着重介绍一下有向图的强连通子图，也称做强连通分量，强连通分支和强连通分块。

求强连通子图的另类算法

- 可以知道，圈上的点都是满足强连通性质的，所以我们可以不断的找圈，然后压缩它，直到找不到圈为止。
- 该算法因为时间复杂度过大，本身没有什么实质的作用，但是会给我们的解题思路和算法证明带来一定的帮助。

求强连通子图的算法 1

- 一种求有向图强连通子图的算法和求无向图块的方法几乎一样，不同的是，我们需要特殊考虑一下横叉边的处理。如果 e=u→v 是横叉边，那么 lowlink[u] := min(lowlink[u],dfn[v]) 这一步就无需再做。

程序代码

```
PROCEDURE DFS(v);
begin
        inc(sign); dfn[v] := sign; // 给 v 按照访问顺序的先后标号为 sign
        lowlink[v] := sign; // 给 lowlink[v] 赋初始值
        inc(tot); stack[tot] := v; //v 点进栈
        instack[v] := true; // 这个用来判断横叉边
        for  寻找一个 v 的相邻节点 u
                if  边 uv 没有被标记过   then
                begin
                        标记边 uv;
                        给边定向 v→u;
                        if u 未被标记过   then
                        begin
                                DFS(u); //uv 是父子边，递归访问
                                lowlink[v] := min(lowlink[v],lowlink[u]);
                        end
```

程序代码

```
                else
                if instack[u] then
                        lowlink[v] := min(lowlink[v],dfn[u]); //uv 是返祖
    边
            end;
        if lowlink[v] = dfn[v] then
        begin
            块数目 number+1;
            repeat
                标记 stack[tot] 这个点为 number;
                instack[stack[tot]] := false;
                dec(tot); //  点出栈
            until stack[tot+1] = v;
        end;
    end;
```

求强连通子图的算法 2

- 基于两次 DFS 的有向图强连通子图算法
  - (1) 对图进行 DFS 遍历，遍历中记下所有的 dfn[v] 的值。遍历的结果是构造了一座森林 W1 ；
  - (2) 改变图 G 中的每一条边的方向，构造出新的有向图 Gr ；
  - (3) 按照 dfn[v] 由大到小的顺序对 Gr 进行 DFS 遍历。遍历的结果是构造了新的森林 W2 ， W2 中的每棵树上的顶点构成了有向图的极大强连通子图。

算法证明？

有向图的压缩

- 将有向图中的强连通子图都压缩成为一个点之后，是否和无向图压缩之后的结果一样呢？
- 有向图压缩之后，连接不同结点之间的边有两种：父子边，横叉边。压缩后的图，不是一个标准意义上的树(将边看作无向)。它是一个无有向圈的有向图，即不可再压缩的图。
- 有向图压缩的意义，在后面的例题《受欢迎的奶牛》中我们会看到。

探索第二部 (1)

- A 和 B 两位侦探要合力解决一起谋杀案。现在有 N 条线索，单独的解决一些线索 A 和 B 花费的时间是有差别的。而在解决掉某些线索之后，可以毫不费力的解决掉另外一些线索。现在你的任务是求出 A 和 B 一起配合解决掉所有线索所需要花费的总时间。
- 数据范围约定：
  - 线索数目 N≤1000
  - 解决每条线索 A 和 B 花费的时间 ai 和 bi 都不超过 15

探索第二部 (2)

- 如果解决了线索 x 顺边就能解决线索 y，那么在 x 和 y 之间连一条有向边。可知，如果解决了 x 之后能解决 y，解决 y 之后能解决 z，那么说明，我们只需要解决掉 x，就能解决 y 和 z 。
- 一个显而易见的性质：如果 x 能通过有向边到达 y， y 不能通过有向边到达 x，那么无论如何， y 都不必解决。

探索第二部 (3)

- 而如果存在 x 和 y 能互达，那么从中任意挑出一个来解决就可以。也就是说，在一个强连通子图内，我们只需要任意挑出一个线索将它解决，就能解决掉该子图内所有的线索。
- 现在的任务便成了，挑出所有的必须被解决线索。然后分配 A 和 B 去解决他们。这个问题，我们可以用动态规划来解决。

探索第二部 (4)

- 那么如何处理一个强连通子图的情况呢？如果让 A 来解决掉一个线索，那么肯定挑出 A 花费时间最少的那条线索；同理如果 B 来解决掉一个线索，那么肯定挑出 B 花费时间最少的那条线索。
- 于是可以将整个子图压缩成为一个点， A 解决它所需要的时间是所有点中 ai 的最小值，B 解决它所需要的时间是所有点中 bi 的最小值。

探索第二部 (5)

- 算法流程：
  - (1) 根据输入建图
  - (2) 求出途中的所有强连通子图，并压缩成一个点
  - (3) 挑出森林中所有的根结点，这些是必须被解决的线索
  - (4) 用动态规划算法解决最小总花费的问题

受欢迎的奶牛 (1)

- N 头奶牛，给出若干个欢迎关系 A B，表示 A 欢迎 B，欢迎关系是单向的，但是是可以传递的。另外每个奶牛都是欢迎他自己的。求出被所有的奶牛欢迎的奶牛的数目。(USACO FALL03)
- 数据范围约定：
  - 奶牛数目 N≤10000
  - 直接的欢迎关系数目 M≤50000

受欢迎的奶牛 (2)

- 可以想到的是，如果图中包含有强连通子图，那么就可以把这个强连通缩成一个点，因为强连通子图中的任意两个点可以到达，强连通子图中所有的点具有相同的性质，即它们分别能到达的点集都是相同的，能够到达它们的点集也是相同的。
- 通过***大胆猜想***，我们得到一个结论：
- 问题的解集是压缩后的图中，唯一的那个出度为 0 的点。

受欢迎的奶牛 (3)

- 首先，如果该图不是一张连通图，那么问题肯定是无解的。在假定图是一张连通图的情况下，我们需要证明如下一些东西：
  - (1) 解集为什么一定构成一个强连通子图？
  - (2) 同时存在 2 个出度为 0 的独立的强连通子图的时侯，为什么就一定无解？
  - (3) 只有一个出度为 0 的强连通子图的时候，为什么该强连通子图一定是问题的解集？
  - (4) 如果一个强连通子图的出度不为 0，为什么就一定不是问题的解集？

受欢迎的奶牛 (4)

- (1) 解集为什么一定构成一个强连通子图？
- 证明：
- 假设 A 和 B 都是最受欢迎的 cow，那么，A 欢迎 B，而且 B 欢迎 A，于是，A 和 B 是属于同一个强连通子图内的点，所以，问题的解集构成一个强连通子图。

受欢迎的奶牛 (5)

- (2) 同时存在 2 个出度为 0 的独立的强连通子图的时侯，为什么就一定无解？
- 证明：
- 如果存在两个独立的强连通分量 a 和 b，那么 a 内的点和 b 内的点一定不能互相到达，那么，无论是 a 还是 b 都不是解集的那个连通分量，问题保证无解。

受欢迎的奶牛 (6)

- (3) 只有一个出度为 0 的强连通子图的时候，为什么该强连通子图一定是问题的解集？
- 证明：
- 假设在压缩过的图中，存在结点 A，它到出度为 0 的结点 (设为 Root) 没有通路，因为 A 的出度一定不为 0，那么设他可以到 B，于是 B 到 Root 没有通路，因为 B 的出度也一定不为 0，那么设他可以到 C……，如此继续下去，因为该图已经不可再压缩，所以这样下去不会出现已经考虑过的点 (否则就存在有向环)，那么这样下去之后，所有的点都到 Root 没有通路，而 Root 到其他所有的点也是没有通路的，因为它的出度为 0，所以 Root 与其他所有的点是独立的，这与大前提“该图是连通图”矛盾。所以假设不成立。

受欢迎的奶牛 (7)

- (4) 如果一个强连通子图的出度不为 0，为什么就一定不是问题的解集？
- 证明：
- 如果某个强连通子图内的点 A 到强连通分量外的点 B 有通路，因为 B 和 A 不是同一个强连通子图内的点，所以 B 到 A 一定没有通路，那么 A 不被 B 欢迎，于是 A 所在的强连通子图一定不是解集的那个强连通子图。

受欢迎的奶牛 (8)

- 算法流程：
  - (1) 压缩有向图
  - (2) 判断连通性，并找到图中出度为 0 的点的个数。
  - (3) 如果图不连通或者出度为 0 的点的个数超过 1，输出无解，否则转 (4)
  - (4) 输出出度为 0 的点代表的强连通子图上的点

科学是在不断的大胆猜想与小心求证中进步的！


Thank you for listening ！

参考文献

- 王树禾《离散数学引论》
- 刘汝佳 / 黄亮《算法艺术与信息学竞赛》
- 吴文虎 / 王建德《图论的算法与程序设计》

MST 另类算法证明

- 我们通过 kruskal 算法的正确性来证明该算法的正确性
- 设该算法得到的 MST' 为 T'，它不是原图的最小生成树 T，则存在一条边 e，有 e∈T' 且 e∈T。由于 T' 不可再调整，所以在 T' 中添加 e 之后，e 是所成环上的最大边。因而在做 kruskal 算法时候，该环上的所有边在 e 之前都会被事先考虑是否加入 MST 中，而在考虑是否加入 e 这条边的时候，该环上的所有点都已经连通了，所以 e 一定不会被加入 MST 中。推出矛盾。

最小环改进算法的证明

- 一个环中的最大结点为 k(编号最大)，与他相连的两个点为 i,j，这个环的最短长度为 g[i][k]+g[k][j]+i 到 j 的路径中，所有结点编号都小于 k 的最短路径长度
- 根据 floyd 的原理，在最外层循环做了 k-1 次之后， dist[i][j] 则代表了 i 到 j 的路径中，所有结点编号都小于 k 的最短路径
- 综上所述，该算法一定能找到图中最小环

定理 1 的证明

- 设 dfn[a]<dfn[b], 在 DFS 的活动中心，即算法中的 v，只沿父子边移动。若 a 不是 b 的祖先，但由 dfn[a]<dfn[b] 可知， a 比 b 先“生”，即活动中心移至 b 之前，已从 a 移到 a 的某个前辈。然而，由算法可知，仅当与 a 关联的边皆被用过后才倒行至其父，这说明 e 已被用过， b 在 a 之前已被发现，应有 dfn[b]<dfn[a] 。矛盾。顾 a 是 b 的祖先。
- 同理可得当 dfn[b]<dfn[a] 的时候， b 是 a 的祖先

定理 2 的证明

- 令 S 是从根 r 到 u 的轨上含 r 不含 u 的一切点组成的集合， T 是以 v 为根的子树上的点集。由定理 1，不存在连接 T 与 V-(S∪{u}∪T) 的边。若存在连接 t∈T 与 s∈S 的边 ts，则它是返祖边，且 dfn[s]<dfn[u] 。这时 lowlink[v] ≤ dfn[s] ≤ dfn[u]，与已知 lowlink[v] ≥ dfn[u] 矛盾，故 ts 这种边不存在，故 u 是割顶，证毕。

定理 3 的证明

- 充分性：
- 若 r 是 G 的 DFS 的生成树的根，且 r 是 G 的割顶，$V_1,V_2,\ldots,V_m$ 是 V-{r} 的一个划分，$G[V_i]$ 是 G-r 的连通片，$i=1,2,\ldots,m$。$i\neq j$ 时，V_i 与 V_j 之间的轨都含 r，这时没有起于边 $rv(v\in V_i)$ 而止于 V_j 中顶的树上的有向轨，故至少有两条以 r 为尾的父子边。
- 必要性：
- 设 rv_1，rv_2 是两条父子边，T 是根在 v_1 的子树，由定理 1，无连接 V(T) 与 $V(G)-(V(T)\cup\{r\})$ 之顶的边，又 $V(G)-V(T)\cup\{r\}\neq\odot$，故 r 是 G 的割顶，证毕。

求强连通子图算法 2 的证明

- 考虑 G 中属于同一个强连通分支的两个顶点 x 和 y 。因为 G 中 x 和 y 可以互达，所以 Gr 中 x 和 y 也可以互达。 x 和 y 一定在 W2 中的同一棵树上。
- 考虑 W2 种顶点 Root 为根的同一棵树上的两个顶点 x 和 y 。图 Gr 中从 Root 到 x 有一条路，因而图 G 中从 x 到 Root 有一条路，所以 Root 要么是 x 的祖先，要么是先于 x 遍历完毕的旁系亲戚。又由于 dfn[Root]>dfn[x] ，所以 Root 不可能是先于 x 遍历完毕的旁系亲戚，而只可能是 x 的祖先，所以图 G 中存在从 Root 到 x 的一条路，所以 x 、 Root 是可以互达的顶点。 x 和 Root 可互达，同理 y 和 Root 也互达，所以 x 和 y 亦互达。
- 综上所述，当且仅当 x 和 y 是 W2 中同一棵树上的顶点时， x 和 y 互达。 W2 中的每棵树构成了一个有向图的极大强连通子图