强连通分量与割点桥

Tarjan dfn/low 时间戳：有向图缩点成 DAG 与无向图寻找关键桥梁

HGNU-ACM 实验室算法组

配套：SCC 缩点成 DAG + dfn/low/in_stk 原理 + 割点与桥 + 4 大真题精讲





本章目录（上）：SCC 缩点与 Tarjan 原理

强连通分量（SCC）与缩点变 DAG

Tarjan 算法：`dfn`、`low` 与 `in_stk`

第 1～2 节：从“单行道互达圈子”认识强连通分量与缩点，再彻底吃透 Tarjan 的 `dfn[u]`、`low[u]` 和为什么必须判断 `in_stk[v]`。

本章目录（下）：割点与桥、例题与收尾

无向图割点与桥：>= vs > 与边编号

四大经典例题精讲

常见错误、检查清单与练习安排

第 3～5 节：分清无向图求桥（>）与求割点（>=）及重边防坑，精讲《受欢迎的牛》《消息的传递》《旅游航道》《电力》。

承前启后：从“无向连通”到“有向互达圈子”

生活场景：全是单行道的城市快递网

第 4 章我们处理的都是双向通车的无向图；但现实里很多路是单行道（有向图）。假设快递员从 A 区能开车到 B 区，却未必能从 B 区开回 A 区！

- 如果有一群城区（比如 $\{1,2,3\}$），它们内部任意两点都能顺着单行道互相到达，我们就说它们抱成了一个“强连通圈子”。
- 站在全局看：在同一个圈子里，消息或货物只要送到其中任意一人手里，整个圈子立刻全都知道了！
- 把每个这样的极大圈子捏成一个“超级大点”，整张复杂有向图就会变得无比清爽！

本页要点

无向图讲“连通块”，有向图讲“强连通分量（**SCC**）”——要求必须双向都能走到！

新术语大白话：强连通、SCC、缩点与 DAG

术语	一句大白话解释
强连通	在有向图里，点 u 能走到 v，同时 v 也能走回 u
强连通分量（**SCC**）	互相都能到达的极大点集（再加任何外面的点就做不到互达了）
缩点	把同一个 SCC 里的所有点合并成 1 个超级大点，只保留跨块之间的边
DAG（有向无环图）	缩点之后得到的新图——绝对不可能再有环了！

- 为什么缩点后一定没有环？因为如果新图里还有环，说明环上的几个大块其实也能互达，早在第一步就被合并成同一个更大的 SCC 了！

本页要点

记住核心转化：任意有向图 $\xrightarrow{\text{Tarjan 缩点}}$ 有向无环图（**DAG**）！

为什么需要缩点？暴力判互达为什么会 TLE？

先算一算暴力做法的操作次数

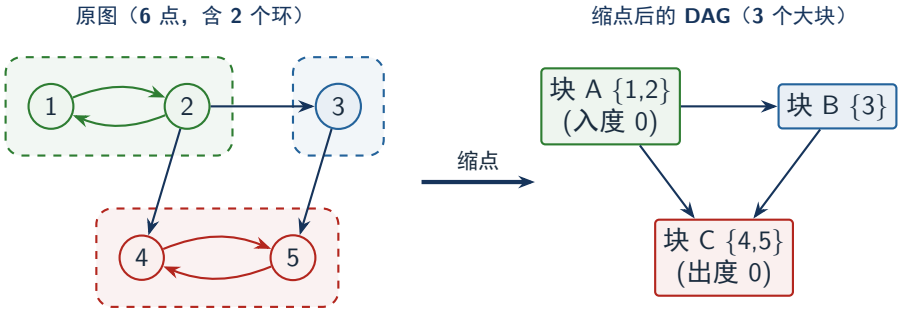
如果不用 Tarjan，想判断哪几个点属于同一个 SCC：需要对每个点 $i = 1 \sim n$ 都跑一次 BFS/DFS，看看 i 能到谁、谁能到 i。一次遍历 $O(n+m)$，n 次就是 $O(n(n+m))$！当 $n = 10^4, m = 5 \times 10^4$ 时，操作次数高达 6×10^8 以上，必 **TLE**！

- 更麻烦的是：有环的图既不能直接做第 1 章学的拓扑排序，也不能直接做 **DAG** 上的动态规划（因为会兜圈子算不完）。
- 而 Tarjan 算法只需要一遍 **DFS**，就能在 $O(n+m)$ 线性时间内把所有 SCC 全部找出来！

本页要点

Tarjan 把找 SCC 从 $O(n(n+m))$ 降到 $O(n+m)$，并把有环图洗白成可以直接拓扑/DP 的 DAG！

缩点变 DAG 独占图解（6 个点缩成 3 个超级块）



本页要点

缩点后，块 A 入度为 0（万源之祖），块 C 出度为 0（百川归海）；原图的所有跨块有向边构成了完美的 DAG！

缩点后最常见的两种题型直觉

- 问法一：“至少要告诉几个人，才能让全图人都知道?”（如《消息的传递》）
  - 在 DAG 里，只要给所有入度为 0 的块各派 1 个人，他们就能顺着有向边把消息传遍全图！
- 问法二：“有多少个人被所有其他人都仰慕?”（如《受欢迎的牛》）
  - 在 DAG 里，被所有人都能走到的点只能是唯一一个出度为 0 的块！若有 ≥ 2 个出度为 0 的块，彼此走不到，答案就是 0。
- 问法三：“最少加几条边，能让整张有向图变成一个大强连通分量?”
  - 设入度为 0 的块有 p 个，出度为 0 的块有 q 个（且总块数 > 1），答案是 $\max(p, q)$！

本页要点

缩点成 DAG 后，90% 的题目只需数一数新图上入度为 0 或出度为 0 的大块有几个！

手算缩点小例子：5 个点 6 条有向边

分析步骤	纸上演算结果
1. 列出有向边	$1 \to 2,\ 2 \to 3,\ 3 \to 1,\ 3 \to 4,\ 4 \to 5,\ 5 \to 4$
2. 找互相可达的圈子	$\{1,2,3\}$ 形成 $1 \to 2 \to 3 \to 1$ 大环；$\{4,5\}$ 形成 $4 \to 5 \to 4$ 小环
3. 画出缩点后的 DAG	块 $1 = \{1,2,3\}$，块 $2 = \{4,5\}$，唯一跨块边：块 $1 \to$ 块 2
4. 统计出入度	块 1：入度 0 出度 1；块 2：入度 1 出度 0

本页要点

注意：$3 \to 4$ 是单向边，所以块 1 能走到块 2，但块 2 回不去块 1，它们绝对不能合并！

常见认知误区：无向图连通块 $\neq$ 有向图强连通分量

- 误区 **1**：看到 $1 \to 2 \leftarrow 3$，以为 $\{1,2,3\}$ 在同一块。
  - 错！1 能到 2，但 2 回不去 1；$1,2,3$ 各自是 3 个独立的 SCC！
- 误区 **2**：把有向图当无向图直接用并查集。
  - 错！并查集只能维护双向等价关系，完全处理不了单向边！
- 误区 **3**：以为孤立的单点不算强连通分量。
  - 错！每个单点自己到自己总是可达的，一个没有入环的点自己就是一个大小为 1 的 SCC。

本页要点

单点自己是一个合法的 SCC；两块之间只有单向边没有回边，绝不能合并！

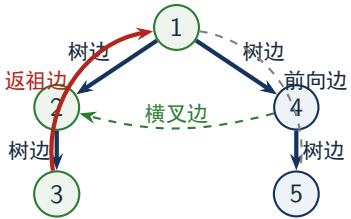
本节小结：3 个概念自测小问

1. 如果一张有向图本身就是一棵树（或者 **DAG**），它有多少个强连通分量？
   - 没有任何环，每个点自己自成一块，共有 n 个大小为 1 的 SCC。
2. 图里有重边（两条一样的 $u \to v$）或自环（$u \to u$）会影响 **SCC** 缩点结果吗？
   - 不会影响 SCC 的划分；但在统计缩点后新图的出入度时，如果要去重（有些题算度不去重也不影响判 0），记得判断 `if (scc[u] != scc[v])` 过滤掉块内边！
3. 整张有向图不连通，是不是也能缩点？
   - 是的！只要外层对 $1 \sim n$ 每个未访问的点都启动一次 Tarjan，照样全部缩成 DAG。

本页要点

接下来第 2 节，我们就来看看 Tarjan 是如何只靠 `dfn`、`low` 和一个栈就一遍 DFS 找出所有 SCC 的！

Tarjan 核心直觉：先看懂 DFS 搜索树上的 4 种边



- 树边：DFS 走新点踩出的边；返祖边：指回“当前正在访问的祖先”（能成环！）；
- 横叉边：指向“已经访问完退出的另一支分支”（不能回到当前祖先，成不了环！）。

本页要点

只有返祖边（以及连向还在栈里的点的边）才能形成环，把节点拉进同一个 SCC！

大白话剖析两个核心数组：dfn[u] 与 low[u]

dfn[u]（出生时间戳）

- 记录 DFS 第一次走到点 u 时的全局步数编号（$1, 2, 3, \ldots$）。
- dfn 越小，说明这个点越“老”（越早被访问，在搜索树上越靠上方）。
- 同时还能兼职充当 vis 数组：dfn[u] == 0 就代表还没访问过！

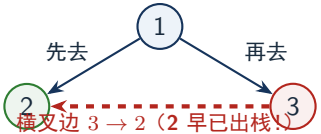
low[u]（能抱到的最老大腿）

- 从点 u 及其子树出发，最多走 1 条还在栈里的非树边，能绕回的最小 dfn 值（最早祖先）。
- 刚进 u 时，先设 low[u] = dfn[u]（至少能到达自己）。
- 每当儿子 v 回来，或者遇到老熟人 v，就尝试把 low[u] 更新得更小！

本页要点

记住人话：dfn[u] 是“出生序号”，low[u] 是“不脱离当前栈能绕回的最老祖宗序号”！

为什么 in_stk[v]（是否在栈中）判断绝对不能漏？



- 假设 DFS 先走完左支 $1 \to 2$：点 2 的 low[2]=dfn[2]=2，独立成块并弹出栈（in_stk[2]=0）！
- 接着 DFS 走右支 $1 \to 3$：点 3 沿着横叉边看到 2（dfn[2]=2 < dfn[3]=3）。
- 如果不判断 if（in_stk[2]），点 3 会误把 low[3] 改成 2，导致 low[3] != dfn[3]，把本不互达的 3 和 1 错误捏在一起！

本页要点

只有还在栈里的点（in_stk[v] == 1），才说明 v 是能和 u 互达的“同代或祖先”，早已出栈的废块绝不能碰！

什么时候能捏成一个 SCC？——low[u] == dfn[u] 的含义

- 每访问一个新点 u，就把它压入栈 `st.push_back(u)`，并标记 `in_stk[u] = 1`。
- 当 u 的所有出边都递归搜完、准备回溯离开 u 时：
  - 如果 `low[u] < dfn[u]`：说明 u 或者它的子孙能绕回 u 上方的更老祖先，u 还不能独立成块，继续留在栈里等上面的老大带走。
  - 如果 `low[u] == dfn[u]`：说明 u 子树里谁也绕不出 u 的上方！于是栈里从 u 到栈顶的所有点正好构成一个完整的强连通分量！
- 此时开启 `while` 循环连续弹栈，直到把 u 自己也弹出来为止，同时清除 `in_stk[x] = 0`！

本页要点

`low[u] == dfn[u]` 就是这个 SCC 的“最高代表（根）”——在它回溯时，把栈顶到它为止的所有点一锅端！

Tarjan SCC 手算跟踪表（含返祖边与横叉边，5 点 6 边）

步	动作（$1\to2, 2\to3, 3\to1, 1\to4, 4\to2, 4\to5$）	dfn	low	栈 st	弹出
1	进 $1\to$ 进 $2\to$ 进 3	$1,2,3$	$1,2,3$	$[1,2,3]$	—
2	$3\to1$（1 在栈中），low[3]=1	—	$low[3]=1$	$[1,2,3]$	—
3	回溯到 $2,1$：low[2]=1，low[1]=1	—	$1,1$	$[1,2,3]$	—
4	$1\to4$，看 $4\to2$（2 在栈中），low[4]=2	$dfn[4]=4$	$low[4]=2$	$[1,2,3,4]$	—
5	$4\to5$，5 无出边，low[5]==dfn[5]=5	$dfn[5]=5$	$low[5]=5$	$[1,2,3,4]$	**{5}**
6	回溯 $4\to1$：low[4]=2，low[1]=1	—	$low[1]=1$	$[1,2,3,4]$	—
终	1 搜完且 low[1]==dfn[1]	—	—	[]	**{4,3,2,1}**

本页要点

第 4 步 4 靠指向栈内 2 的边把 low[4] 降成 $2<4$，所以 4 和 $\{1,2,3\}$ 强连通，最后一起弹出！

为什么 Tarjan SCC 一定对？——栈不变量论证

阶段	循环不变量：栈里所有点在原图中互相之间或向下都能连通到当前的搜索路径
初始化	开始前栈为空，所有 `dfn[i]=0`，`in_stk[i]=0`。
保持	每个刚访问的 u 进栈；凡是能连回栈内更老点 v 的边，都会拉低 `low[u]`，保证同一环上的点在回溯到环最高点前绝不会被提前弹出。
终止	当某点 u 回溯结束仍满足 `low[u]==dfn[u]`，栈中 u 上方的所有点既能从 u 到达，又因留在栈中而必能绕回 u，故恰好是一个极大 SCC！

本页要点

栈就像一个“临时收容所”，只有确认下面这拨人谁也逃不出当前根 u 时，才把他们打包成一个 SCC 放出！

Tarjan SCC 完整模板代码（1/2）：DFS 与更新 low

```
vector<int> dfn(n + 1, 0), low(n + 1, 0), scc(n + 1, 0), sz(n + 1, 0), st;
vector<char> inst(n + 1, 0);
int timer = 0, cnt = 0;
auto tarjan = [&](auto &&self, int u) -> void {
    dfn[u] = low[u] = ++timer; st.push_back(u); inst[u] = 1;
    for (int v : g[u]) {
        if (!dfn[v]) {
            self(self, v); low[u] = min(low[u], low[v]); // 树边用子树的 low 更新
        } else if (inst[v]) {
            low[u] = min(low[u], dfn[v]);           // 必须在栈内！用 dfn[v] 更新
        }
    }
```

本页要点

树边回来用 low[v] 更新，栈内老点用 dfn[v] 更新；两者不要写混！

Tarjan SCC 完整模板代码（2/2）：弹栈缩点与外层循环

```
    if (low[u] == dfn[u]) {                          // u 是当前强连通分量的根
        ++cnt;
        while (true) {
            int x = st.back(); st.pop_back(); inst[x] = 0; // 出栈必须清 inst[x]
            scc[x] = cnt; ++sz[cnt];
            if (x == u) break;
        }
    }
};
for (int i = 1; i <= n; ++i)
    if (!dfn[i]) tarjan(tarjan, i);                  // 外层循环防图不连通！
```

本页要点

完整源码可对照 `code/tarjan_cows.cpp`；外层 `for (1..n)` 千万不能只写一次 `tarjan(1)`！

为什么这样写？（两个经典灵魂追问）

- 追问 1：为什么 `else if (inst[v])` 里写 `low[u] = min(low[u], dfn[v])`，写成 `low[v]` 行不行？
  - 在求有向图 SCC 时，写成 `low[v]` 碰巧也能对（因为同在栈里）；但在下一节无向图求割点时，写成 `low[v]` 会直接 **WA**（把祖宗的祖宗穿过去）！所以统一死记：老点永远用 `dfn[v]` 更新，绝对不踩坑！
- 追问 2：缩点后，新图的拓扑序和 `cnt` 编号有什么关系？
  - 因为越晚出栈（越靠近起点）的块编号越大，所以 `cnt`，`cnt-1`，...，1 天然就是缩点后 DAG 的一个正拓扑序（完全不用再写队列跑拓扑排序了）！

本页要点

记住两条铁律： 非树边更新永远用 `dfn[v]`； SCC 编号从大到小天然就是拓扑序！

上机验证：3 组测试与反事实追问

测试场景	输入有向图	预期 SCC 划分
1. 纯单环	$1 \to 2, 2 \to 3, 3 \to 1$	1 个 SCC：$\{1,2,3\}$
2. 含横叉边双支	$1 \to 2, 1 \to 3, 3 \to 2$	3 个独立 SCC：$\{2\},\{3\},\{1\}$
3. 不连通孤立点	3 个点，1 条边 $1 \to 2$	3 个 SCC：$\{2\},\{1\},\{3\}$

反事实追问

在第 2 组输入（$1 \to 2, 1 \to 3, 3 \to 2$）中，如果把 `else if (inst[v])` 删掉变成 `else`：

$\to$ 搜完 2 出栈后，搜 3 时走横叉边 $3 \to 2$ 会把 `low[3]` 错改成 $2 < dfn[3] = 3$，导致 3 无法弹栈！最后会把 $\{1,3\}$ 捏成一个假环！

本页要点

横叉边测试用例 $1 \to 2, 1 \to 3, 3 \to 2$ 是检验你 Tarjan SCC 写得对不对的试金石！

Tarjan SCC 易错对照（WA / RE 定位）

错误表现	常见错误写法	正确做法与原因
WA（假环串块）	漏写 `inst[v]` 判断	必须写 `else if (inst[v])` 拦住已出栈的横叉边
WA（死锁或乱块）	弹栈时忘记写 `inst[x] = 0`	已出栈的点仍被当成“在栈内”参与后续更新
WA（漏掉连通块）	主函数只调用 `tarjan(1)`	必须写 `for(i=1..n) if(!dfn[i]) tarjan(i)`
WA（重算自环度）	统计新图度数没判 `scc[u]!=scc[v]`	块内边会变成新图自环，导致入度/出度虚高

本页要点

查 Tarjan SCC 的 BUG 就盯这 4 处：`inst[v]`、`inst[x]=0`、外层循环、`scc[u]!=scc[v]`！

生活引入：交通网里的“独木桥”和“咽喉路口”

场景对比：双向通车的城市公路网（无向图）

- 桥（割边，**Bridge**）：整座城市只有这一座跨江大桥；一旦这座桥被洪水冲断，江南和江北就再也无法互相通车（连通块数量增加）。
- 割点（**Articulation Point**）：某个核心立交桥路口；一旦把这个路口封锁（连同接在它上面的所有路一起拆掉），原本连通的城市就裂成两半或更多块。

- 如果只关心“删一条边会怎样” $\Longrightarrow$ 找桥；如果关心“删一个点会怎样” $\Longrightarrow$ 找割点。
- 无向图 **DFS** 树的重要性质：因为边是双向的，无向图 DFS 树里绝对没有横叉边，只有树边和返祖边！所以不用开栈也不用 `in_stk`！

本页要点

无向图找割点和桥比有向图更简单：不用栈、不用 `in_stk`，只需分清 `>` 和 `>=`！

为什么不能暴力删边/删点？

算一算暴力做法的操作次数

如果每条边都试着删掉一次，再跑一遍 DFS 看图还连不连通：删 m 条边就需要跑 m 次 DFS，总复杂度 $O(m(n+m))$。
当 $n = 3 \times 10^4, m = 3 \times 10^4$ 时，操作次数高达 1.8×10^9，必定 **TLE**！

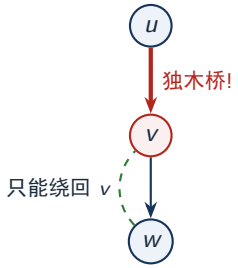
- 而用 Tarjan 只需要一遍 **DFS**，在 $O(n+m)$ 线性时间内就能把全图所有的桥和割点一次性全部揪出来！
- 还是用老朋友：`dfn[u]`（出生序号）和 `low[u]`（儿子子树不走来边能绕回的最早祖先）。

本页要点

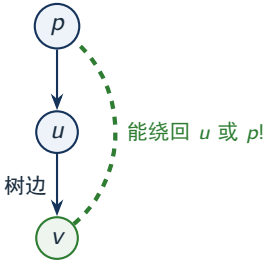
一遍 $O(n+m)$ DFS 替代 m 遍暴力遍历，关键全在判断儿子 v 能不能绕过父亲 u 回到更早的祖先！

找桥的判定条件：low[v] > dfn[u] 独占图解

情况 **A**：(u, v) 是桥（low[v] > dfn[u]）



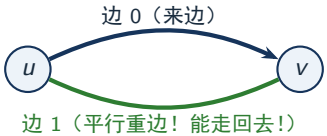
情况 **B**：(u, v) 不是桥（low[v] <= dfn[u]）



本页要点

当儿子 v 搜完回来，如果 low[v] > dfn[u]（严格大于！连 u 自己都绕不回来），说明 (u, v) 是唯一的独木桥！

重边陷阱！为什么不能写 `if (v == fa) continue;`？

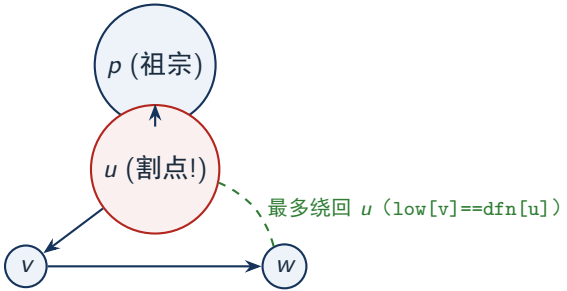


- 假设 u 和 v 之间有两条平行重边（边 0 和边 1），显然这两条边都不是桥（断了一条还有另一条）。
- 如果按“父节点 `fa`”过滤（`if (v == fa) continue;`），站在 v 时会把两条通向 u 的边全部跳过，误以为 v 回不到 u，错判成桥！
- 正确做法：给每条无向边一个编号 $e = 0, 1, \ldots, m-1$，递归时传入走过来的那条边编号 `pe`，只跳过 `if (e == pe) continue;`！

本页要点

有重边时，按“来边编号 `pe`”拦截原路返回，另一条平行重边就能正常把 `low[v]` 拉低成 `dfn[u]`！

找割点的判定条件：low[v] >= dfn[u] 独占图解



本页要点

找桥要求严格 low[v] > dfn[u]；找割点只需 low[v] >= dfn[u]！

为什么找桥是 >，而找割点是 >=?

- 虽然 v 能绕回 u（所以边 (u, v) 不是桥），但如果把点 u 整个删掉，这条绕回 u 的路也会一起断掉！
- 只要 v 绕不到 u 上方更老的祖宗 p（即 `low[v] >= dfn[u]`），删掉 u 之后 v 子树就被彻底孤立了！

本页要点

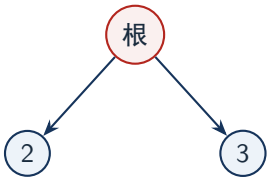
删边只断那一条边，删点会连带断掉所有连向 u 的返祖边，所以 `low[v] == dfn[u]` 时 u 也是割点！

为什么 DFS 根节点必须特判 child >= 2？

根只有 1 个儿子：不是割点



根有 ≥ 2 棵独立子树：是割点！



- 对于根 root，没有更老的点，所以儿子的 low[v] 必 $\geq$ dfn[root] $= 1$。
- 若根只有 1 个儿子，删掉根下面仍完整；只有根在 DFS 树有 ≥ 2 棵独立子树（child >= 2）时，删掉根才会裂开！

本页要点

非根节点看 low[v] >= dfn[u]；根节点只看 DFS 树上有几个亲儿子 child >= 2！

无向图求桥手算跟踪表（5 点 5 边，含 1 环 2 桥）

步	动作（$0:(1,2),1:(2,3),2:(3,1),3:(2,4),4:(4,5)$）	dfn	low	桥判定
1	从 1 出发：$1 \xrightarrow{e=0} 2 \xrightarrow{e=1} 3$	$1,2,3$	$1,2,3$	—
2	3 走 $e=2$ 见 1（$e \neq pe=1$），low[3]=1	—	$low[3]=1$	—
3	回溯到 2：low[2]=min(2, low[3])=1	—	$low[2]=1$	$1 \not> 2$（非桥）
4	2 走 $e=3$ 到 4，再走 $e=4$ 到 5	$4,5$	$4,5$	—
5	5 回溯到 4：low[5]=5 > dfn[4]=4	—	$low[4]=4$	$(4,5)$ 是桥!
6	4 回溯到 2：low[4]=4 > dfn[2]=2	—	$low[2]=1$	$(2,4)$ 是桥!
终	2 回溯到 1：low[2]=1 == dfn[1]=1，结束	—	—	共 2 条桥

本页要点

环 $\{1,2,3\}$ 里的三条边 low 都能绕回 1，一条桥都没有；伸出去的链 $(2,4),(4,5)$ 全是桥！

无向图求桥完整模板代码（边编号 pe 版）

```cpp
vector<int> dfn(n + 1, 0), low(n + 1, 0);
int timer = 0, bridges = 0;
auto dfs = [&](auto &&self, int u, int pe) -> void {
    dfn[u] = low[u] = ++timer;
    for (auto [v, e] : g[u]) {
        if (e == pe) continue;                  // 拦截来边，兼容重边
        if (!dfn[v]) {
            self(self, v, e); low[u] = min(low[u], low[v]);
            if (low[v] > dfn[u]) ++bridges; // 严格 > 才是桥
        } else low[u] = min(low[u], dfn[v]);
    }
};
```

本页要点

源码见 code/tarjan_bridges.cpp；外层写 for(i=1..n) if(!dfn[i]) dfs(dfs,i,-1);！

无向图求割点完整模板代码（带 cut[u] 去重）

```
vector<int> dfn(n + 1, 0), low(n + 1, 0);
vector<char> cut(n + 1, 0); int timer = 0;
auto dfs = [&](auto &&self, int u, int root) -> void {
    dfn[u] = low[u] = ++timer; int child = 0;
    for (int v : g[u]) {
        if (!dfn[v]) {
            ++child; self(self, v, root);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u] && (u != root || child >= 2)) cut[u] = 1;
        } else low[u] = min(low[u], dfn[v]);        // 割点不怕重边，也不用跳过父
    点！
    }
};
```

本页要点

求割点一定要用 cut[u] = 1 标记去重，否则一个割点有多个儿子会被重复计数！

为什么这样写？（割点与桥的 3 个关键对比）

对比项	无向图求桥	无向图求割点
判定不等号	`low[v] > dfn[u]`（严格大于）	`low[v] >= dfn[u]`（大于等于）
根节点特判	不需要特判根节点	必须特判 `u != root || child >= 2`
重边处理	必须传“来边编号 pe”跳过原边	不怕重边，甚至不用跳过 `fa`

- 为什么求割点时“连 `fa` 都不用跳过”？因为求割点时本来就可以通过来边看回 `dfn[fa]`，哪怕把 `low[u]` 更新成 `dfn[fa]`，只要不大于等于 `dfn[fa]` 的判定就只发生在 `fa` 看 `u` 时，完全不影响 `low[v] >= dfn[u]`！

本页要点

一张表记死求桥与求割点的三大差异，考试默写绝不写混！

上机验证：3 组测试与反事实追问

测试场景	输入无向图	预期输出（桥数 / 割点数）
1. 两点两平行重边	$n=2, m=2$：$(1,2),(1,2)$	桥数 0 / 割点数 0
2. 链式 **3** 点	$n=3, m=2$：$(1,2),(2,3)$	桥数 2 / 割点数 1（点 2）
3. 三角形带尾巴	$(1,2),(2,3),(3,1),(2,4)$	桥数 1（$(2,4)$）/ 割点数 1（点 2）

反事实追问

在三角形带尾巴（第 3 组）中，如果求桥时不小心把 `low[v] > dfn[u]` 写成了 `>=`，会多算哪条边？

$\rightarrow$ 从 1 出发走 $1 \to 2 \to 3$，回溯到 1 时 `low[2]=1 == dfn[1]=1`，会把 $(1,2)$ 错判成桥！

本页要点

交题前必测“两点两重边”，桥数必须输出 0 才算过关！

割点与桥易错对照（WA 定位）

错误表现	常见错误写法	正确做法与原因
WA（重边算成桥）	求桥写 `if (v == fa) continue;`	必须传入边编号 `pe`，写 `if (e == pe) continue;`
WA（非桥算成桥）	求桥写成 `low[v] >= dfn[u]`	找桥必须严格 `low[v] > dfn[u]`
WA（割点算重复）	满足条件直接 `++ans`	同一割点若有 3 个子树会被加 3 次！必须开 `cut[u]=1`
WA（根错算割点）	根只有 1 个儿子也算成割点	必须判 `u != root || child >= 2`
WA（多组污染）	多组输入没清空 `dfn`，`low`，`timer`	每组数据重新构造或清空数组与 `timer=0`

本页要点

割点与桥的题目多为多组测试数据，每组结束务必清空 `dfn`，`low`，`timer`！

例 1 | [基础] 受欢迎的牛（一本通 1513）： 先读懂要求

大白话题意

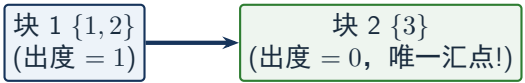
N（$1 \le N \le 10^4$）头牛，M（$1 \le M \le 5 \times 10^4$）对单向仰慕关系 (A, B)（表示 A 认为 B 受欢迎，且关系可传递）。求有多少头牛被除自己之外的所有牛认为受欢迎。

- 输入：第 1 行 N, M；接下来 M 行每行两个整数 A, B（可能有重边）。
- 输出：明星牛的数量。
- 样例：$N = 3, M = 3$，边 $(1, 2), (2, 1), (2, 3) \implies$ 只有 3 号牛被所有人仰慕，输出 1。

本页要点

难度标签：**[基础]**。有向图传递闭包问题，用 Tarjan 缩点成 DAG 后找唯一汇点！

例 1 | 思路与建模：缩点成 DAG 后数出度为 0 的块



- 在同一个强连通分量里，牛互相仰慕，地位完全等价（同荣共辱）。
- 缩点变成 DAG 后，如果一头牛被全图所有牛仰慕，它所在的块不能再仰慕别的块（否则它仰慕的块里那些牛就不仰慕它了），所以这个块出度必须为 0！
- 若全图恰好只有 1 个出度为 0 的块，答案就是该块的大小 `sz[c]`；若出现 ≥ 2 个出度为 0 的块，它们彼此互不仰慕，答案必为 0！

本页要点

图不连通时也至少会产生 2 个出度为 0 的块，代码自动输出 0，无需额外特判！

例 1 | 样例手推跟踪表

阶段	处理对象	dfn / low	栈	结果
DFS 1	$1 \to 2 \to 3$（3 无出边）	$dfn = [1,2,3], low = [1,1,3]$	$[1,2,3]$	弹 **{3}**, sz[1]=1
DFS 2	$2 \to 1$ 更新 low[2]=1	$low[1] == dfn[1] == 1$	$[1,2]$	弹 **{2,1}**, sz[2]=2
建 DAG	扫 $(1,2),(2,1),(2,3)$	仅 $(2,3)$ 跨块	—	outdeg[2]=1, out[1]=0
统计	数 outdeg[c]==0	只有块 1 出度为 0	—	zero_out==1 $\Longrightarrow$ 1

本页要点

手推确认：块 $2 = \{1,2\}$ 仰慕块 $1 = \{3\}$，块 1 出度为 0 且唯一，答案为 1！

例 1 | 核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. 对全图跑 Tarjan 缩点	每个点归属 `scc[i]` 及块大小 `sz[c]`	外层写 `for(1..n)` 了吗？
2. 统计跨块边出度	`if(scc[u]!=scc[v]) ++outdeg[scc[u]]`	过滤同块内边了吗？
3. 统计 `outdeg[c]==0`	块数 `zero_out` 与块大小 `ans`	只有 `zero_out==1` 才输出 `ans`！

本页要点

相对 **Tarjan SCC** 模板改了哪里？只需在弹栈后多写 5 行统计跨块 `outdeg` 即可。

例 1 | 核心代码（1/2）：Tarjan 缩点（code/tarjan_cows.cpp）

```
auto tarjan = [&](auto &&self, int u) -> void {
    dfn[u] = low[u] = ++timer; st.push_back(u); inst[u] = 1;
    for (int v : g[u]) {
        if (!dfn[v]) { self(self, v); low[u] = min(low[u], low[v]); }
        else if (inst[v]) low[u] = min(low[u], dfn[v]);
    }
    if (low[u] == dfn[u]) { ++cnt;
        while (true) {
            int x = st.back(); st.pop_back(); inst[x] = 0;
            scc[x] = cnt; ++sz[cnt]; if (x == u) break;
        }
    }
};
```

本页要点

弹栈时顺便用 `++sz[cnt]` 统计出每个强连通分量里包含几头牛。

例 1 | 核心代码（2/2）：统计 DAG 出度

```
for (int i = 1; i <= n; ++i) if (!dfn[i]) tarjan(tarjan, i);
vector<int> outdeg(cnt + 1, 0);
for (int u = 1; u <= n; ++u)
    for (int v : g[u]) if (scc[u] != scc[v]) ++outdeg[scc[u]];
int zero_out = 0, ans = 0;
for (int c = 1; c <= cnt; ++c) if (outdeg[c] == 0) { ++zero_out; ans = sz[c]; }
cout << (zero_out == 1 ? ans : 0) << '\n';
```

本页要点

全部牛互仰慕成 1 个大环时，只有 1 个块且出度为 0，输出 N，完全正确！

例 1 | 易错与 3 组自测边界

- 易错 **1**（忘过滤同块边）：漏写 `scc[u] != scc[v]` 会把块内边算成出度，导致唯一汇点出度 > 0 误输出 0（WA）。
- 易错 **2**（不连通输出错）：图不连通时会有 ≥ 2 个出度为 0 的块，必须严格判 `zero_out == 1` 才输出 ans。
- **3** 组自测边界：
  1. 样例 $N = 3, M = 3 \implies$ 输出 1；
  2. 两头孤立牛 $N = 2, M = 0 \implies 2$ 个汇点块，输出 0；
  3. 全图大环 $1 \to 2 \to 3 \to 1 \implies 1$ 个汇点块，输出 3。

本页要点

记住：图不连通时也至少有 2 个出度为 0 的块，所以 `zero_out == 1` 天生帮我们判了不连通！

例 2 | [基础] 消息的传递（一本通 1516）： 先读懂要求

大白话题意

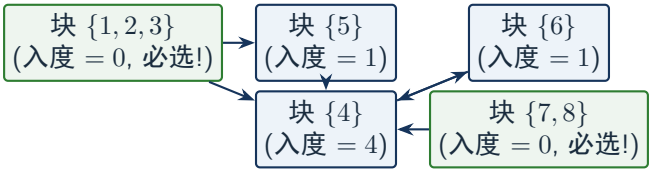
有 N（$N \le 1000$）个奸细，给一张 $N \times N$ 的 $0/1$ 矩阵：$C_{i,j}=1$ 表示 i 能直接把消息传给 j（单向）。问最少先告诉几个奸细，就能让所有奸细都收到消息？

- 输入：第 1 行 N；接下来 $N \times N$ 矩阵。
- 输出：最少先告诉的奸细个数。
- 样例：$N=8$（含环 $\{1,2,3\}$ 和 $\{7,8\}$，其余点沿单向边往下传） $\Longrightarrow$ 输出 2。

本页要点

难度标签：**[基础]**。跟《受欢迎的牛》正好相反：那题看出度为 0，这题看入度为 0！

例 2 | 思路与建模：DAG 里所有入度为 0 的块必须亲自告知



- 同一个 SCC 内部只要告诉其中任何 1 人，整个块就全收到了。
- 缩点变成无环 DAG 后：任何入度 >0 的块都可以靠上游传过来；而入度为 0 的块没有任何上游，只能亲自通知 1 人！

本页要点

答案就是缩点后 DAG 中入度为 0 的 **SCC** 块个数！

例 2 | 样例手推跟踪表

阶段	识别的 SCC 块	跨块连边	入度统计
1. 找环缩点	$\{1,2,3\},\{4,6\},\{7,8\},\{5\}$	$\{1,2,3\}\to\{5\},\{4,6\}$	$\{1,2,3\}$ 入度 0
2. 继续连边	$\{5\}\to\{4,6\}$；$\{7,8\}\to\{4,6\}$	汇入 $\{4,6\}$	$\{5\},\{4,6\}$ 入度 >0
3. 统计答案	入度 0 块仅 $\{1,2,3\}$ 与 $\{7,8\}$	共 2 个源头块	输出 2

本页要点

哪怕 $\{4,6\}$ 内部也有环（$4\to6\to4$），但它被上游指向所以入度 >0，不用额外派人通知！

例 2 | 核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. 读入 $N \times N$ 矩阵建邻接表	若 $x == 1$ 则 `g[i].push_back(j)`	是有向边，别连成双向！
2. 跑 Tarjan SCC 缩点	每个点对应的块号 `scc[u]`	这题不用统计块大小 `sz`
3. 统计 `indeg[scc[v]]`	注意加的是终点 `v` 的入度	有没有写成 `indeg[scc[u]]`？
4. 数 `indeg[c] == 0` 个数	直接输出源头块数 `ans`	记得循环 `c = 1..cnt`

本页要点

相对例 1 改了哪里？ 矩阵读入转邻接表； 统计的是终点 `scc[v]` 的入度 `indeg`，数 0 入度块！

例 2 | 核心代码（源码 code/tarjan_message.cpp）

```
for (int i = 1; i <= n; ++i)
    for (int j = 1, x; j <= n; ++j) { cin >> x; if (x == 1) g[i].push_back(j);
}
// ... 中间执行标准 Tarjan SCC 缩点（求得 scc[1..n] 与总块数 cnt） ...
for (int i = 1; i <= n; ++i) if (!dfn[i]) tarjan(tarjan, i);

vector<int> indeg(cnt + 1, 0);
for (int u = 1; u <= n; ++u)
    for (int v : g[u]) if (scc[u] != scc[v]) ++indeg[scc[v]];
int ans = 0;
for (int c = 1; c <= cnt; ++c) if (indeg[c] == 0) ++ans;
cout << ans << '\n';
```

本页要点

全部是 0 的矩阵谁也传不到谁，有 N 个入度为 0 的单点块，输出 N!

例 2 | 易错与 3 组自测边界

- 易错 **1**（入度出度写反）：把 `++indeg[scc[v]]` 误写成 `++indeg[scc[u]]`，会算成出度为 0 的块数（WA）。
- 易错 **2**（建双向边）：矩阵 $C_{i,j}=1$ 只表示 $i \to j$ 单向传递，绝不能连成双向边。
- **3** 组自测边界：
  1. 样例 $N=8 \implies$ 输出 2；
  2. 全 0 矩阵 $N=3 \implies$ 互不传递，输出 3；
  3. 全 1 矩阵 $N=3 \implies$ 全成 1 个大环，输出 1。

本页要点

记住对照口诀：《受欢迎的牛》数出度为 0（汇点），《消息的传递》数入度为 0（源点）！

例 3 | [基础] 旅游航道（一本通 1524）：先读懂要求

大白话题意

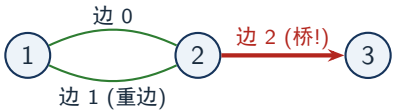
多组数据。每组给 n（原题记作 $m \le 3 \times 10^4$）个星球与 m 条双向航道。如果删除某条航道会使某些星球不再可达，它就叫“主要航道”（即桥）。求主要航道的总数。输入以 `0`（或 `0 0`）结束。

- 输入：多组 n, m 及 m 行 a, b，最后遇到 0 终止。
- 输出：每组一行整数，表示桥的数量。
- 样例：`2 1` / `1 2` / `0 0` $\Longrightarrow$ 输出 `1`。

本页要点

难度标签：**[基础]**。无向图求桥标准模板题，务必注意多组数据清空与重边兼容！

例 3 | 思路与建模：用边编号 pe 防重边求桥



- 本题没有保证无重边，建图时存 pair<int，int>{邻接点，边编号 e}。
- DFS 传入走过来的边编号 pe，只有 e == pe 才跳过；这样平行的重边 $(1,2)$ 不会误判成桥，只有真正的独木桥 $(2,3)$ 满足 low[3] > dfn[2]！

本页要点

用 while (cin >> n && n != 0) 读多组数据，遇到单行 0 或 0 0 都能安全退出！

例 3 | 样例与重边案例手推跟踪表

测试图	DFS 过程	`dfn / low`	桥判定	输出
样例 2 1: $(1,2)$	$1 \xrightarrow{e=0} 2$，遇 $e=0$ == *pe* 跳过	$[1,2]/[1,2]$	$low[2] > dfn[1]$	1
重边 2 2: $(1,2)\times 2$	$1 \xrightarrow{e=0} 2$，跳 $e=0$ 走 $e=1$ 见 1	$low[2]=1$	$1 \not> 1$（非桥）	0

本页要点

对比两行手推可见：边编号 pe 让重边安全通过，只有真正无绕路可走的边才会被记为桥！

例 3 | 核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. `while(cin>>n && n!=0)`	读入 m 并清空图 g	遇到结尾单行 0 会不会多读？不会！
2. 建双向边存 {v, e}	第 e 条边连 $a \leftrightarrow b$	边编号 e 从 $0 \sim m-1$ 唯一吗？
3. 局部重置 `dfn,low,timer`	跑遍所有 $1 \sim n$ 未访问点	每组数据都重置 `bridges=0` 了吗？

本页要点

相对求桥模板改了哪里？把全部数组放到 `while` 循环内部定义，天然实现每组数据自动清空！

例 3 | 核心代码（1/2）：多组读入与边编号建图

```
int n, m;
while (cin >> n && n != 0) {
    cin >> m;
    vector<vector<pair<int, int>>> g(n + 1);
    for (int e = 0, a, b; e < m; ++e) {
        cin >> a >> b;
        g[a].push_back({b, e}); g[b].push_back({a, e}); // 同一条无向边共用编号
e
    }
    vector<int> dfn(n + 1, 0), low(n + 1, 0);
    int timer = 0, bridges = 0;
```

本页要点

把 g，dfn，low，timer，bridges 全放进 while 内部，彻底杜绝多组数据残留污染！

例 3 | 核心代码（2/2）（code/tarjan_bridges.cpp）

```cpp
auto dfs = [&](auto &&self, int u, int pe) -> void {
    dfn[u] = low[u] = ++timer;
    for (auto [v, e] : g[u]) {
        if (e == pe) continue; // 传 pe 兼容重边
        if (!dfn[v]) {
            self(self, v, e); low[u] = min(low[u], low[v]);
            if (low[v] > dfn[u]) ++bridges;
        } else low[u] = min(low[u], dfn[v]);
    }
};
for (int i = 1; i <= n; ++i) if (!dfn[i]) dfs(dfs, i, -1);
cout << bridges << '\n';
```

本页要点

匹配 code/tarjan_bridges.cpp；外层 for(1..n) 兼容不连通图！

例 3 | 易错与 3 组自测边界

- 易错 **1**（用父节点判重边）：写 `if (v == fa) continue;` 遇两条平行重边会错把非桥算成桥（WA）。
- 易错 **2**（不等号写成 `>=`）：求桥必须严格 `low[v] > dfn[u]`，等于只算割点不算桥（WA）。
- **3** 组自测边界：
  1. 样例 `2 1` / `1 2` / `0 0` $\Longrightarrow$ 输出 `1`；
  2. 两点两重边 `2 2` / `1 2`, `1 2` $\Longrightarrow$ 输出 `0`；
  3. 单点无边 `1 0` $\Longrightarrow$ 输出 `0`。

本页要点

记住：多组数据每组都在 `while` 内部重新分配 `g`，`dfn`，`low`，绝不串数据！

例 4 | [提高] 电力（CTU Open 2004）：先读懂要求

大白话题意

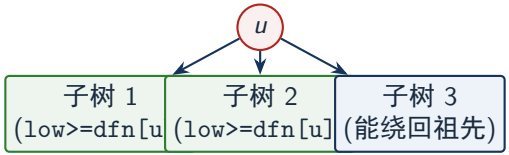
多组数据。每组给 P（$1 \le P \le 10^4$）个点（编号 $0 \sim P-1$）和 C 条无向边（保证无重边）。问恰好删去一个点之后，剩下的图最多能分成多少个连通块？以 `0 0` 结束。

- 输入：多组 P, C 及 C 行 p_1, p_2（0-indexed）。
- 输出：删去一个最优的点后，最多剩下的连通块数量。
- 样例：`3 3`（三角形）$\to 1$；`4 2`（两条不相连边）$\to 2$；`3 1`（一链一孤立点）$\to 2$；`1 0` $\to 0$。

本页要点

难度标签：**[提高]**。不仅要找割点，还要算“删掉这个点能裂出几个新块”，并注意孤立点与多连通块！

例 4 | 思路与建模：原连通块数 base + 删点额外增量 split



- 设原图本来就有 base 个连通块。
- 对于当前连通块里的点 u：每有一个儿子 v 满足 low[v] >= dfn[u]，删掉 u 就会多割出一块，记这种儿子数为 split：
  - 若 u 不是根：删掉 u 后原来那 1 块变成 $1 + \texttt{split}$ 块（额外增加 split 块）。
  - 若 u 是根且有 c 个儿子：删掉根后变成 c 块（若 $c = 0$ 孤立点，删掉后块数 -1；若 $c \geq 1$，也是增加 $c - 1$ 块!）。

本页要点

如果对根节点只在 child >= 2 时把 split 从 0 开始加，那么非孤立根额外增量恰好也是 split!

例 4 | 样例手推跟踪表

样例输入	原块数	各点删后额外增量 split	best	输出
3 3（三角形）	1	根 0 只有 1 儿子 $\to 0$；其余 0	0	$1+0=1$
4 2（$0-1, 2-3$）	2	每点均 1 儿子 $\to 0$	0	$2+0=2$
3 1（$0-1$ 与 2）	2	点 $0,1 \to 0$；孤立点 $2 \to -1$	0	$2+0=2$
1 0（单孤立点）	1	唯一孤立点 0, child==0 $\to -1$	-1	$1+(-1)=0$

本页要点

看最后一行 1 0：全图只有一个孤立点时，删掉它就剩 0 个块！所以 best_extra 初值必须设为 -1！

例 4 | 核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. 0-indexed 建无向图	`g(n), dfn(n,0), low(n,0)`	注意点编号从 0 到 $n-1$！
2. 外层统计 base	每进一次未访问根 `++base`	初值 `best_extra = -1` 对吗？
3. `low[v]>=dfn[u]` 时累加	`if(u!=root || child>=2) ++split`	根有 2 儿子时 `split=1`，恰好少 1！
4. 孤立根特判	`u==root && child==0` 贡献 `-1`	最终输出 `base + best_extra`

本页要点

相对割点模板改了哪里？不只是标记 `cut[u]=1`，而是用局部变量 `split` 精确数出删掉 u 额外增加的连通块数！

例 4 | 核心代码（1/2）：多组读入与割点增量计算

```
int n, m;
while (cin >> n >> m && (n != 0 || m != 0)) {
    vector<vector<int>> g(n);
    for (int i = 0, u, v; i < m; ++i) {
        cin >> u >> v; g[u].push_back(v); g[v].push_back(u);
    }
    vector<int> dfn(n, 0), low(n, 0);
    int timer = 0, base = 0, best_extra = -1;
```

本页要点

注意退出条件是 `(n != 0 || m != 0)`，因为存在 `1 0`（1 个点 0 条边）的合法输入！

例 4 | 核心代码（2/3）：Tarjan 算每个点的分裂增量

```
auto dfs = [&](auto &&self, int u, int root) -> void {
    dfn[u] = low[u] = ++timer;
    int child = 0, split = 0; // split: 删 u 多出的块数
    for (int v : g[u]) {
        if (!dfn[v]) {
            ++child; self(self, v, root);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u] && (u != root || child >= 2)) ++split;
        } else low[u] = min(low[u], dfn[v]);
    }
    if (u == root && child == 0) best_extra = max(best_extra, -1);
    else best_extra = max(best_extra, split);
};
```

本页要点

根有 2 棵子树时第 1 棵不计入，第 2 棵起计数，故 split = child - 1！

例 4 | 核心代码（3/3）：外层连通块统计（code/tarjan_power.cpp）

```
        for (int i = 0; i < n; ++i)
            if (!dfn[i]) { ++base; dfs(dfs, i, i); }
        cout << base + best_extra << '\n';
    }
```

- 易错与自测：
  - 易错：忘记给孤立点算 -1（1 0 错输出 1），或者循环写成 1..n（RE/WA）。
  - **3** 组自测：1 0 $\to 0$；星形图 $n = 5$（中心连 4 点）$\to 4$；环图 $n = 4 \to 1$。

本页要点

完全匹配 code/tarjan_power.cpp！

强连通分量与割点桥高频错误对照表

题目 / 场景	常见错误写法	正确做法与原因
有向缩点忘判 inst[v]	已访问分支直接写 else low[u]=min(...,dfn[v])	横叉边连向已出栈旧块会把两个无环块捏到一起 **WA**
找桥用父点判重边	写 if (v == fa) continue;	两点间有 2 条平行重边时误跳过，不是桥也算成桥！
割点 vs 桥不等号混	找桥写成 low[v] >= dfn[u]	等于只说明刚好绕回父亲 u；找桥严格 >，割点才是 >=
图不连通只搜 1 号	只调用一次 dfs(1)	未覆盖全部连通块漏掉孤立分量，必写 for (i=1..n)
多组数据忘清空	多组测试沿用全局 dfn, low, timer	每组开始全部重置并重设 timer = 0

本页要点

“有向缩点记 inst[v]，无向求桥记边号 pe，找桥严格 > 割点 >=，外层必写 for(1..n)”！

评测结果快速定位指南（WA / TLE / RE）

评测结果	优先排查方向
WA	1. 有向图是否漏写 `else if (inst[v])` 或出栈忘了清 `inst[x]=0`； 2. 统计缩点新图入度/出度时是否漏了 `if (scc[u] != scc[v])`； 3. 无向图求桥有重边时是否误用了 `v == fa` 而非边编号 `e == pe`； 4. 割点问题是否漏了根节点 `child >= 2` 或没用 `cut[u]` 去重。
TLE	1. 是否在多组数据里用 `memset` 清空了过大的全局数组； 2. 图的读入（如 $N = 1000$ 矩阵 10^6 个数）是否忘了关闭同步流 `ios::sync_with_stdio(false)`。
RE	1. 下标是 0-indexed（如《电力》）还是 1-indexed； 2. 无向图邻接表开成单向或递归深度过深导致栈溢出（注意全局静态数组 vs 局部）。

本页要点

遇到 WA 先测“横叉边反例”和“两点两重边”；遇到 RE 先核对题目点编号是从 0 还是从 1 开始！

提交前检查清单（六必查）

1. 启动 Tarjan 前的外层循环是不是遍历了全部点（`if (!dfn[i]) tarjan(i)`）预防图不连通？
2. 有向图强连通分量中，遇到 `dfn[v] != 0` 的老节点时，是不是严格限制了 `else if (inst[v])`？
3. 出栈捏成新块时，`inst[x] = 0` 有没有同步随着 `pop_back()` 清除掉？
4. 无向图求桥时，是否传入“来边编号 `pe`”来兼容平行重边，且判定为严格 `low[v] > dfn[u]`？
5. 无向图求割点时，判定是否为 `low[v] >= dfn[u]`，且单独特判根节点 `child >= 2` 并去重？
6. 缩点建新 DAG 图后，统计出度/入度时是否跳过了同块自环 `if (scc[u] != scc[v])`？

本页要点

交题前自测两组：一组含 2 条平行重边的两岛图（桥数应为 0），一组含横叉边有向图 $1 \to 2, 1 \to 3, 3 \to 2$！

本章全部练习与完成标准

1. 有向图 **Tarjan** 缩点：受欢迎的牛（一本通 1513，`code/tarjan_cows.cpp`）、消息的传递（1516，`code/tarjan_message.cpp`）、网络协议（1515）。
2. 无向图割点与桥：旅游航道（1524，`code/tarjan_bridges.cpp`）、电力（CTU Open 2004，`code/tarjan_power.cpp`）、分离的路径（1520）。

完成标准

能准确解释 `dfn` 与 `low` 的区别；能说明为何横叉边需判 `inst[v]`、重边需传边号 `pe`、割点根需判 `child >= 2`；独立 AC 上述精选习题。

课后安排与下一篇预告

- 复习：合上课件再画一遍有向图“左支连到已出栈右支”横叉边反例，看加不加 `inst[v]` 区别。
- 练习：独立复写 `code/` 下四份源码，并亲自测重边、孤立单点和多组数据清零用例。
- 下一篇预告：恭喜完成全部《06_ 图论算法》篇！下一篇正式开启《07_ 字符串算法》第 1 章《字符串哈希与 KMP 扩展》——看如何用多项式取模把整串变数字 $O(1)$ 比相等，以及让模式匹配失配指针永不后退的 KMP！

本章一句话

记牢每个点的出生号与能绕回的最老祖宗，一遍 DFS 既能捏环缩点成 DAG，又能揪出全图独木桥与咽喉割点。

谢谢！