

割点和桥

相关阅读：[双连通分量](#)

割点和桥更严谨的定义参见 [图论相关概念](#)。

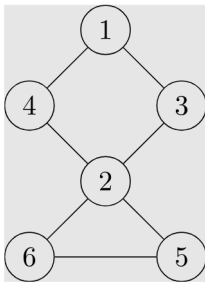
割点

对于一个无向图，如果把一个点删除后这个图的极大连通分量数增加了，那么这个点就是这个图的割点（又称割顶）。

过程

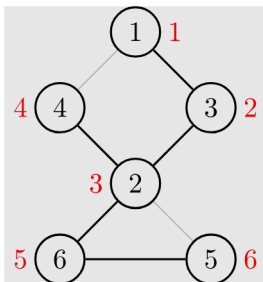
如果我们尝试删除每个点，并且判断这个图的连通性，那么复杂度会特别的高。所以要介绍一个常用的算法：Tarjan。

首先，我们上一个图：



很容易的看出割点是 2，而且这个图仅有这一个割点。

首先，我们按照 DFS 序给他打上时间戳（访问的顺序）。



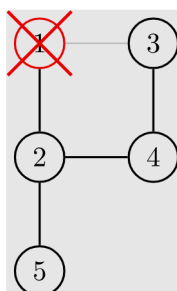
这些信息被我们保存在一个叫做 `dfn` 的数组中。

还需要另外一个数组 `low`，用它来存储不经过其父亲能到达的最小的时间戳。

例如 `low[2]` 是 1，`low[5]` 和 `low[6]` 是 3。

然后我们开始 DFS，我们判断某个点是否是割点的根据是：对于某个顶点 u ，如果存在至少一个顶点 v (u 的儿子)，使得 $low_v \geq dfn_u$ ，即不能回到祖先，那么 u 点为割点。

此根据惟独不适用于搜索的起始点，其需要特殊考虑：若该点不是割点，则其他路径亦能到达全部结点，因此从起始点只「向下搜了一次」，即在搜索树内仅有一个子结点。如果在搜索树内有两个及以上的儿子，那么他一定是割点了（设想上图从 2 开始搜索，搜索树内应有两个子结点 3 或 4 及 5 或 6）。如果只有一个儿子，那么把它删掉，不会有任何的影响。比如下面这个图，此处形成了一个环。



我们在访问 1 的儿子时候，假设先 DFS 到了 2，然后标记用过，然后递归往下，来到了 4，4 又来到了 3，当递归回溯的时候，会发现 3 已经被访问过了，所以不是割点。

更新 low 的伪代码如下：

```
1  if  $v$  is a son of  $u$ 
2       $low_u = \min(low_u, low_v)$ 
3  else
4       $low_u = \min(low_u, dfn_v)$ 
```

例题

[洛谷 P3388 【模板】割点（割顶）](#)

例题代码

```

1  /*
2  洛谷 P3388 【模板】割点（割顶）
3  */
4  #include <iostream>
5  #include <vector>
6  using namespace std;
7  int n, m; // n: 点数 m: 边数
8  int dfn[100001], low[100001], idx, res;
9  // dfn: 记录每个点的时间戳
10 // low: 能不经父亲到达最小的编号, idx: 时间戳, res: 答案数量
11 bool vis[100001], flag[100001]; // flag: 答案 vis: 标记是否重复
12 vector<int> edge[100001]; // 存图用的
13
14 void Tarjan(int u, int fa) { // u 当前点的编号, fa 自己爸爸的编号
15     vis[u] = true; // 标记
16     low[u] = dfn[u] = ++idx; // 打上时间戳
17     int child = 0; // 每一个点儿子数量
18     for (const auto &v : edge[u]) { // 访问这个点的所有邻居
19         (C++11)
20         if (!vis[v]) {
21             child++; // 多了一个儿子
22             Tarjan(v, u); // 继续
23             low[u] = min(low[u], low[v]); // 更新能到的最小节点编号
24             if (fa != u && low[v] >= dfn[u] && !flag[u]) { // 主要代码
25                 // 如果不是自己, 且不通过父亲返回的最小点符合割点的要求, 并且
26                 没有被标记过
27                 // 要求即为: 删了父亲连不上去了, 即为最多连到父亲
28                 flag[u] = true;
29                 res++; // 记录答案
30             }
31             } else if (v != fa) {
32                 // 如果这个点不是自己的父亲, 更新能到的最小节点编号
33                 low[u] = min(low[u], dfn[v]);
34             }
35         }
36         // 主要代码, 自己的话需要 2 个儿子才可以
37         if (fa == u && child >= 2 && !flag[u]) {
38             flag[u] = true;
39             res++; // 记录答案
40         }
41     }
42
43 int main() {
44     cin >> n >> m; // 读入数据
45     for (int i = 1; i <= m; i++) { // 注意点是从 1 开始的
46         int x, y;
47         cin >> x >> y;
48         edge[x].push_back(y);
49         edge[y].push_back(x);

```

```

50     } // 使用 vector 存图
51     for (int i = 1; i <= n; i++) // 因为 Tarjan 图不一定连通
52         if (!vis[i]) {
53             idx = 0; // 时间戳初始为 0
54             Tarjan(i, i); // 从第 i 个点开始，父亲为自己
55         }
56     cout << res << endl;
57     for (int i = 1; i <= n; i++)
58         if (flag[i]) cout << i << " "; // 输出结果
    return 0;
}

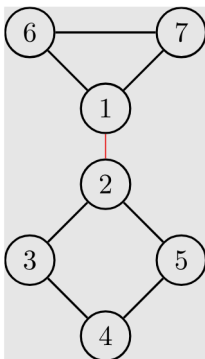
```

割边（无重边时）

和割点差不多，叫做桥。

对于一个无向图，如果删掉一条边后图中的连通分量数增加了，则称这条边为桥或者割边。严谨来说，就是：假设有连通图 $G = \{V, E\}$ ， e 是其中一条边（即 $e \in E$ ），如果 $G - e$ 是不连通的，则边 e 是图 G 的一条割边（桥）。

比如说，下图中，



红色的边就是割边。

过程

和割点差不多，只要改一处： $low_v > dfn_u$ 就可以了，而且不需要考虑根节点的问题。

割边是和是不是根节点没关系的，原来我们求割点的时候是指点 v 是不可能不经过父节点 u 为回到祖先节点（包括父节点），所以顶点 u 是割点。如果 $low_v = dfn_u$ 表示还可以回到父节点，如果顶点 v 不能回到祖先也没有另外一条回到父亲的路，那么 $u - v$ 这条边就是割边。

实现

下面代码实现了对 **无重边** 的无向图求割边，其中，当 `isbridge[x]` 为真时，`(father[x], x)` 为一条割边。

C++

```
1  int low[MAXN], dfn[MAXN], idx;
2  bool isbridge[MAXN];
3  vector<int> G[MAXN];
4  int cnt_bridge;
5  int father[MAXN];
6
7  void tarjan(int u, int fa) {
8      father[u] = fa;
9      low[u] = dfn[u] = ++idx;
10     for (const auto &v : G[u]) {
11         if (!dfn[v]) {
12             tarjan(v, u);
13             low[u] = min(low[u], low[v]);
14             if (low[v] > dfn[u]) {
15                 isbridge[v] = true;
16                 ++cnt_bridge;
17             }
18         } else if (v != fa) {
19             low[u] = min(low[u], dfn[v]);
20         }
21     }
22 }
```

Python

```
1  low = [0] * MAXN
2  dfn = [0] * MAXN
3  idx = 0
4  isbridge = [False] * MAXN
5  G = [[0 for i in range(MAXN)] for j in range(MAXN)]
6  cnt_bridge = 0
7  father = [0] * MAXN
8
9
10 def tarjan(u, fa):
11     father[u] = fa
12     idx = idx + 1
13     low[u] = dfn[u] = idx
14     for i in range(0, len(G[u])):
15         v = G[u][i]
16         if dfn[v] == False:
17             tarjan(v, u)
18             low[u] = min(low[u], low[v])
19             if low[v] > dfn[u]:
20                 isbridge[v] = True
21                 cnt_bridge = cnt_bridge + 1
22         elif v != fa:
23             low[u] = min(low[u], dfn[v])
```

割边（有重边时）

然而，上述无重边时的做法在有重边的无向图上是有问题的。

因为两节点间可能不止有一条边，此时它们都不会是桥。

过程

一种思路是将参数 `fa` 改为刚刚走过的边的编号（每条边的编号一致）即可，即将「不用父节点更新」改为「不用来时的边更新」。

另一种更简单的思路是设立一个标记判断是否已有一条边抵达父节点，标记后再访问到父节点时正常更新。

下面代码实现了对可能 **有重边** 的无向图求割边。

C++

```
1  int low[MAXN], dfn[MAXN], idx;
2  bool isbridge[MAXN];
3  vector<int> G[MAXN];
4  int cnt_bridge;
5  int father[MAXN];
6
7  void tarjan(int u, int fa) {
8      bool flag = false;
9      father[u] = fa;
10     low[u] = dfn[u] = ++idx;
11     for (const auto &v : G[u]) {
12         if (!dfn[v]) {
13             tarjan(v, u);
14             low[u] = min(low[u], low[v]);
15             if (low[v] > dfn[u]) {
16                 isbridge[v] = true;
17                 ++cnt_bridge;
18             }
19         } else {
20             if (v != fa || flag)
21                 low[u] = min(low[u], dfn[v]);
22             else
23                 flag = true;
24         }
25     }
26 }
```


练习


- [P3388【模板】割点（割顶）](#)
- [POJ2117 Electricity](#)
- [HDU4738 Caocao's Bridges](#)

- [HDU2460 Network](#)
- [POJ1523 SPF](#)

Tarjan 算法还有许多用途，常用的例如求强连通分量，缩点，还有求 2-SAT 的用途等。

 本页面最近更新：2025/6/2 12:25:06，[更新历史](#)

 发现错误？想一起完善？ [在 GitHub 上编辑此页！](#)

 本页面贡献者：[Ir1d](#), [sshwy](#), [StudyingFather](#), [H-J-Granger](#), [countercurrent-time](#), [Entertainer](#), [GavinZhengOI](#), [NachtgeistW](#), [ouuan](#), [Planet6174](#), [Oxis-cn](#), [AngelKitty](#), [CCXXXI](#), [cjsoft](#), [diauweb](#), [Early0v0](#), [ezoixx130](#), [GekkaSaori](#), [Henry-ZHR](#), [iamtwz](#), [Konano](#), [LovelyBuggies](#), [Makkiy](#), [Marcythm](#), [mgt](#), [minghu6](#), [P-Y-Y](#), [PotassiumWings](#), [SamZhangQingChuan](#), [Suyun514](#), [tder6](#), [Tiphereth-A](#), [weiyong1024](#), [ylxmf2005](#), [ChungZH](#), [CoelacanthusHex](#), [Error-Eric](#), [Gesrua](#), [HeRaNO](#), [ImpleLee](#), [kenlig](#), [ksyx](#), [kxccc](#), [lychees](#), [mcendu](#), [Menci](#), [Peanut-Tang](#), [Qiu-Quanzhi](#), [shawlleyw](#), [SukkaW](#), [t123yh](#), [Xeonacid](#), [yiyangit](#), [yusancky](#)

© 本页面的全部内容在 [CC BY-SA 4.0](#) 和 [SATA](#) 协议之条款下提供，附加条款亦可能应用