割点和割边

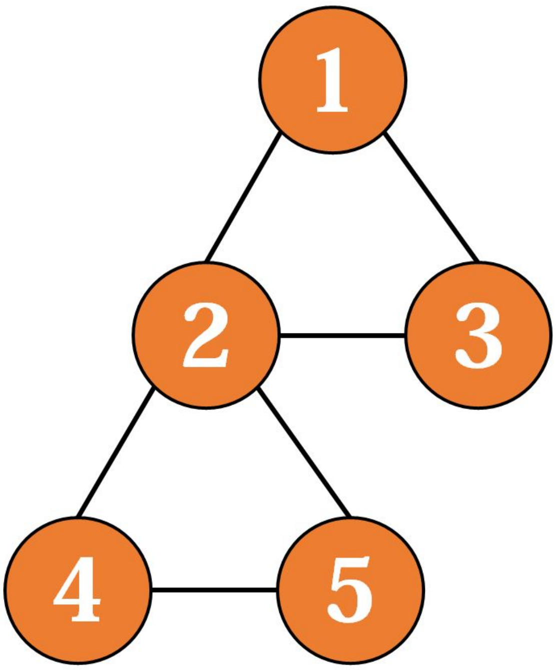
如果删除**无向图**中的某个**点**会使无向图的**连通分量数**增多，则把这个点称为**割点**。类似地，如果删除无向图中的某条**边**会使无向图的连通分量数增多，则把这个点称为**割边**或**桥**。割点与桥可以用**Tarjan算法**求出。

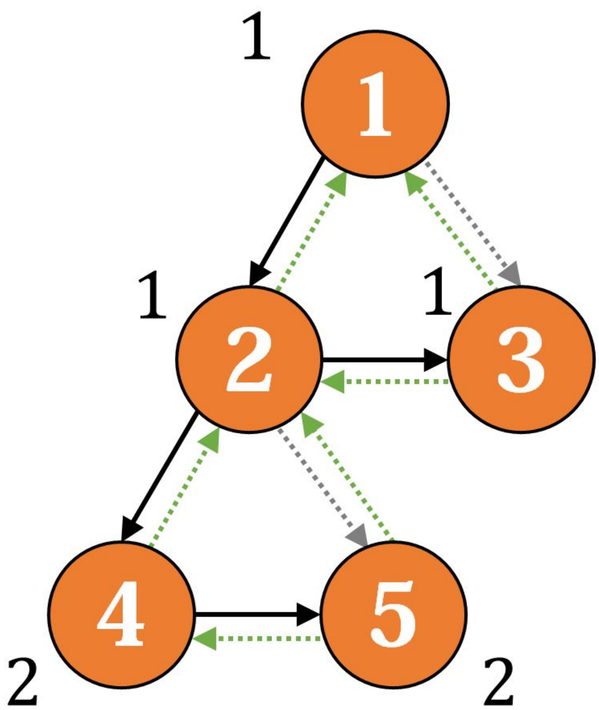
割点

设 $\mathrm{low}(u)$ 表示 u 所在子树中的节点经过**至多一条非树边**能到达的节点中最小的dfs序[1]。实际上，这里只需要考虑**反向边**，很容易发现无向图是不存在横叉边的，前向边则对 low 没有影响。

如果 p 存在一个子结点 q 满足 $\mathrm{low}(q) \geq \mathrm{dfsn}(p)$ ，说明 q 无法通过它的子树“逃”到比 p 的dfs序更小的节点。那么，既然走子树走不通， q 如果想到达这样的点，只能选择经过它的父节点 p 。因此，如果删去 p ， q 和dfs序小于 p 点的点就分开了。

这时我们一般可以说 p 是割点了，只有一种特殊情况，就是 p 是dfs生成树的**根节点**的情形。这时，整个连通分量都不存在比 p 的dfs序更小的点。





例如，上图中，2号点是割点，但1号点不是，因为它是dfs生成树的根节点。这种特殊情况也很好处理：对于根节点，它如果有两个以上子节点，那么它就是割点（显然删除根节点后这两个分支将会互不相连）。

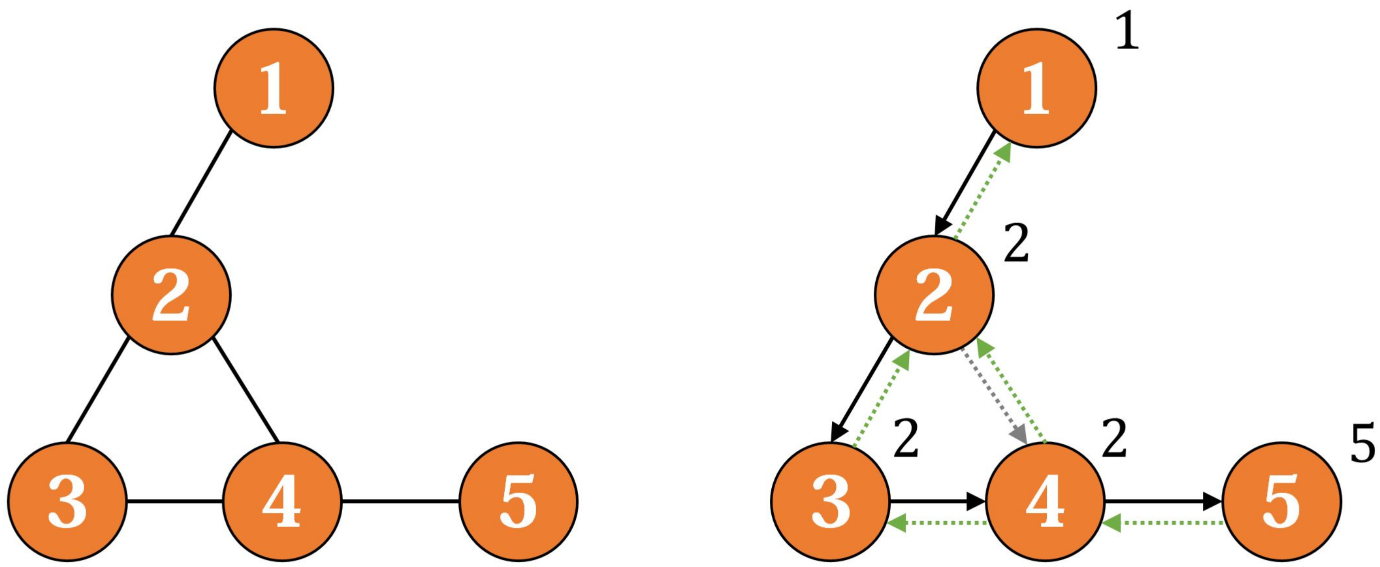
在实现时，不再需要像强连通分量一样维护一个栈：

```
int dfsn[MAXN], low[MAXN], cnt;
vector<int> cut; // 存储所有割点
void tarjan(int p, bool root = true)
{
    int tot = 0;
    low[p] = dfsn[p] = ++cnt;
    for (auto q : edges[p])
    {
        if (!dfsn[q])
        {
            tarjan(q, false);
            low[p] = min(low[p], low[q]);
            tot += (low[q] >= dfsn[p]); // 统计满足low[q] >= dfsn[p]的子节点数目
        }
        else
            low[p] = min(low[p], dfsn[q]);
    }
    if (tot > root) // 如果是根，tot需要大于1；否则只需大于0
        cut.push_back(p);
}
```

桥

为了找到桥，我们要稍微修改一下 low 的定义：我们限定经过的那条**非树边**不能是从**子节点**直接到**父节点**的反向边。对于修改后的 low ，我们可以断言：如果 p 是 q 的父节点，并且 $\mathrm{low}(q) > \mathrm{dfsn}(p)$ ，那么 $p \leftrightarrow q$ 是桥。

因为如果 $p \leftrightarrow q$ 不是桥，那么删掉这条边[2]后 q 一定有其他路径可以到达 p 。注意无向图没有横叉边，想要到达 p 只能通过子树走反向边实现，那么 $\mathrm{low}(q) \leq \mathrm{dfsn}(p)$ 应该成立，然而这与条件矛盾。因此 $p \leftrightarrow q$ 正是桥。



上图中 $1 \leftrightarrow 2$ 和 $4 \leftrightarrow 5$ 都是桥，如果不修改定义，将有 $\text{low}(2)=1$ 和 $\text{low}(5)=4$ ，一座桥都找不到。

```
vector<pair<int, int>> bridges;
int dfsn[MAXN], low[MAXN], fa[MAXN], cnt;
void tarjan(int p)
{
    low[p] = dfsn[p] = ++cnt;
    for (auto to : edges[p])
    {
        if (!dfsn[to])
        {
            fa[to] = p; // 记录父节点
            tarjan(to);
            low[p] = min(low[p], low[to]);
            if (low[to] > dfsn[p])
                bridges.emplace_back(p, to);
        }
        else if (fa[p] != to) // 排除父节点
            low[p] = min(low[p], dfsn[to]);
    }
}
```

参考

^ 这里因为是无向边，不用限定非树边u→v中v可达u（它总是成立）。
^ 这就是为什么要修改定义，修改后的定义限定了不可走p↔q。