Undirected Graph

Undirected Graph DFS

由于计算机的限制，不能直接表示无向图，只能通过有向对称图的方式表现无向图，即通过有向边 $u \to v, v \to u$ 表示无向边 uv

这样的表示方式会带来“二次遍历”的问题，即两条有向边在 DFS 的过程中均会被遍历，而第二次遍历需要被直接忽略。

在 DFS 中可以根据遍历推进的方向为无向图定向

Edge in DFS

无向图也可以根据遍历的推进过程将边分为 TE, BE, DE, CE，但由于边的无向特性，导致对边的标记与有向图有一些差异

TE

遍历过程中发现白色节点并递归地进行遍历时，将连接该白色节点的边标记为 TE，遍历过程中 TE 组成遍历树

BE

遍历节点 u 时，发现一条边指向灰色节点 v

此时若边 vu 是 TE，表示遍历 v 时发现白色节点 u 并对其遍历，显然节点 u 必然遍历到其灰色父节点 v ，此时是二次遍历，应当剔除。

若 v 是 u 的某个非父节点的祖先节点，标记为 BE

DE

遍历节点 u 时，发现一条边指向节点 v ，且 v 是 u 在遍历树中的后继节点，这次遍历一定是二次遍历，应当被剔除

考虑节点 v 的颜色

- v 不可能是白色，否则 uv 为 TE
- v 不可能是灰色，否则 uv 为 BE
- v 只可能是黑色，即 v 已完成遍历，且一定完成了边 vu 的遍历，根据节点 u, v 之间的祖先后继关系，vu 首次遍历时被标记为 BE

CE

遍历无向图时不可能出现 CE，考虑遍历 u 时发现邻居 v ，且 u, v 之间没有祖先后继关系，由之前的分析，v 只能为黑色，则遍历 v 时 u 显然未被遍历，u 为白色，边 vu 为 TE，与 u, v 之间没有祖先后继关系矛盾

DFS skeleton

与有向图 DFS 的不同之处在于，无向图的 DFS 需要正确处理二次遍历，需要为节点维护其直接祖先的信息

DFS(v, parent)

```
v.color := GRAY
<Preorder processing of node v>
foreach neighbor w of v do
    if w.color = WHITE then
        <Exploratory processing of TE vw>
        DFS(w, v)
        <Backtrack processing for TE vw>
    else
        if w.color = GRAY and w != parent
            <check BE vw>
<Postorder processing of node v>
v.color := BLACK
```

Biconnected Graph

引入 k 连通的概念

> **定义 4.7** k 点连通， k 边连通
>
> 对于连通的无向图 G ，如果其中任意去掉 $k-1$ 条点/边，图仍连通，则称 G 是 k 点/边连通

我们更关心 $k=2$ 的特殊情况。

当一个图不是 2-点连通时，图中存在某个点，删去该点图便不再连通，同理可得当一个图不是 2-边连通时，图中必然存在某个边，删去该边图便不再连通。

由此引入割点（articulation point）和桥（bridge）的概念

> **定义 4.8** 割点和桥
>
> 对于一个连通的无向图 G ，称节点 v 为割点，如果去掉 v 后 G 不再连通；称边 uv 为桥，如果去掉 uv 后 G 不再连通

Articulation point

Definition

割点的定义显然是个全局的性质（去掉后图不连通），根据这个性质很难高效地判断割点，因此需要将割点的全局性质等价变换为一个局部的性质

> **引理 4.7** 割点基于路径的定义
>
> 节点 v 为割点 $\iff$ 存在节点 w, x ，满足 v 出现在所有从 w 到 x 的路径上

若想在图遍历的过程中寻找到割点，需要更进一步得到割点基于 DFS 的定义

> **引理 4.8** 割点基于 DFS 的定义
>
> 假设在一次 DFS 中，节点 v 不是遍历树的根节点，则 v 为割点 $\iff$ 在遍历树中存在 v 的某个子树，没有任何 BE 指向 v 的祖先节点。
>
> $(\Leftarrow)$：易验证若存在 v 的某个子树，没有任何 BE 指向 v 的祖先节点，则删去 v 后该子树与其他部分断连
>
> $(\Rightarrow)$：若 v 是割点，则存在不同于 v 的两个顶点 x, y ，满足 v 在从 x 到 y 的所有路径上。显然 x, y 中至少有一个是 v 的后继节点，否则有不经过 v 的路径
>
> $$x \rightsquigarrow LCA(x, y) \rightsquigarrow y \quad (\text{LCA means Lowest Common Ancestor})$$
>
> 故 x, y 中至少有一个是 v 的后继，v 不是叶节点。此时假设任意 v 的子树均存在 BE 指向 v 的祖先，此时显然可构造出一条从 x 到 y 且不经过 v 的路径（通过 BE），矛盾。
>
> 对于根节点，若其遍历树子树个数大于 1 则其为割点，证明显然

Strategy

根据引理 4.8 即可将寻找割点的方法转换为计算机可实现的操作。具体来说，为每个节点维护一个变量 back 来判断其是否为割点

- v 首次被发现时，$v.back = v.discoverTime$
- 当发现一条 BE vw 时，$v.back = \min\{v.back, w.discoverTime\}$
- 当遍历完 v 的邻居 w 回退到 v 时，$v.back = \min\{v.back, w.back\}$

显然 back 值只会减小，有两种情况会导致一个节点的 back 值减小

- 遍历处理 BE vw 时由于 w 作为祖先节点有更小的 discoverTime，导致 v 的 back 减小
- 遍历完 TE vw 回退至 v 时，back 的减小会从 w 传递至 v

在 DFS 中由于每次只发现一个顶点，故每个顶点的 discoverTime 都是不相同的，可以看出 back 变量代表了**不通过父节点可到达的最早的祖先节点**的 discoverTime，若子树中没有 BE 指向祖先节点，则节点的 back 值即为自身的 discoverTIme

显然，若遍历完 TE vw 回退后，发现 $w.back \geqslant v.discoverTime$ ，则说明 w 不通过 v 不能到达更早的祖先节点，根据引理 4.8 可确定 v 是割点

Algorithm

将上述策略嵌入 DFS 框架即可得到寻找割点的算法

ARTICULATION-POINT-DFS(v)

```
v.color := GRAY
time := time + 1
v.discoverTime := time
v.back := v.discoverTime
foreach neighbor w of v do
    if w.color = WHITE then
        ARTICULATION-POINT-DFS(w)
        if w.back >= v.discoverTime:
            Output v as an articulation point
        v.back := min(w.back, v.back)
    else
        if vw is BE then
            v.back := min(v.back, w.discoverTime)
```

算法代价与图遍历相同，为 $O(m+n)$

Correctness

> **定理 4.5** ARTICULATION-POINT-DFS 是正确的

> 当遍历完 TE vw 后若 $w.back \geqslant v.discoverTime$ ，根据 back 的更新方法，若以 w 为根的子树中存在 BE 指向 v 的祖先，则其祖先的 discoverTime 会被赋给该节点的 back，且随着遍历的回退会被传递到 w 的 back，由于 v 祖先的 discoverTime 一定小于 v 的 discoverTime，故 $w.back < v.discoverTime$ ，存在一条 BE 指向 v 的祖先。
>
> 反之，若 $w.back \geqslant v.discoverTime$ 则说明以 w 为根的子树不存在 BE 指向 v 的祖先，根据引理 4.8，v 为割点

Bridge

Definition

与割点的定义类似，需要将桥的全局性质转换成局部性质以判断桥

> **引理 4.9** 桥基于 DFS 的定义
>
> 给定遍历树中的 TE uv ，u 是 v 的父节点，则 uv 是桥 $\iff$ 以 v 为根的所有遍历树的子树中，没有 BE 指向 v 的祖先
>
> 证明过程类似割点的基于 DFS 的定义

Strategy

此部分与割点的策略相同，为每个节点维护一个 back 变量

- v 首次被发现时，$v.back = v.discoverTime$
- 当发现一条 BE vw 时，$v.back = \min\{v.back, w.discoverTime\}$
- 当遍历完 v 的邻居 w 回退到 v 时，$v.back = \min\{v.back, w.back\}$

对于一条 TE uv ，当遍历结束回退时，若 $v.back > u.discoverTime$ ，则 uv 为桥。显然若 uv 不是桥，则以 v 为根的子树可不通过边 uv 与图的其他部分连通，即子树中存在一条 BE 指向 u 或 u 的祖先。其 back 值传递到 v 的 back 值，显然 u 和 u 祖先的 discoverTime 不超过 u 的 discoverTime，$v.back \leqslant u.discoverTime$ ，故当 $v.back > u.discoverTime$ 时，uv 是桥。

Algorithm

BRIDGE-DFS(v)

```
v.color := GRAY
time := time + 1
v.discoverTime := time
v.back := v.discoverTime
foreach neighbor w of v do
    if w.color = WHITE then
        BRIDGE-DFS(w)
        if w.back > v.discoverTime:
            Output vw as a bridge
        v.back := min(w.back, v.back)
    else
        if vw is BE then
            v.back := min(v.back, w.discoverTime)
```

Other undirected graph problem

Orientation of an undirected graph

为每条边定向，且满足每个顶点的入度至少为 1

hint：BE

Get MST in $O(m+n)$ time

权值只有 1 和 2 的图寻找 MST

hint：两次 DFS，分别对权为 1 和权为 2 的边