A7 毛九弢 221900175

# 1 CLRS 20-2

割点

Let $G$ be a connected, undirected graph. An **articulation point** of $G$ is a vertex whose removal disconnects $G$. A **bridge** of $G$ is an edge whose removal disconnects $G$. A **biconnected component** of $G$ is a maximal set of edges such that any two edges in the set lie on a common simple cycle. Figure 20.10 illustrates these definitions. You can determine articulation points, bridges, and biconnected components using depth-first search. Let $G_\pi = (V, E_\pi)$ be a depth-first tree of $G$.

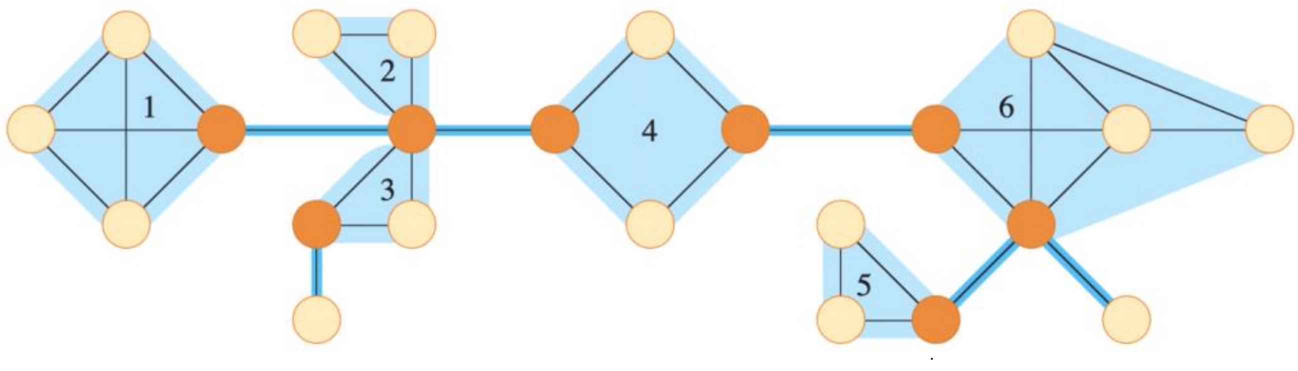



**Figure 1** The articulation points, bridges, and biconnected components of a connected, undirected graph for use in Problem 20-2. The articulation points are the orange vertices, the bridges are the dark blue edges, and the biconnected components are the edges in the light blue regions, with a *bcc* numbering shown.

1. Prove that the root of $G_\pi$ is an articulation point of $G$ if and only if it has at least two children in $G_\pi$.

prove ⇒)

若 $G_\pi$ 的根 $r$ 为一个割点，那么设在 $G$ 中，$r$ 与点 $x_1, x_2, \cdots x_k$ ($k \geq 2$) 相邻

(若 $k=1$，显然 $r$ 不为割点)

使用反证法：

1. 假设 $r$ 没有 children ⇒ $V=\{r\}$，$r$ 不是割点（矛盾）

2. 假设 $r$ 只有1个 children 假设为 $x_a$，∵ $r$ 为割点，$k \geq 2$ ∴ $\exists x_b \neq x_a$，删去 $r$ 后 $x_a$ 与 $x_b$ 不连通。 后补证.

又在 dfs 中 $r.expand = 1$，$x_a.expand = 2$，$x_b.expand > x_a.finish > x_a.expand$ (结束时间) (发现时间)

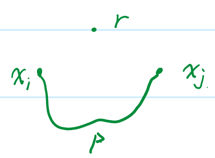
当 $t = x_a.finish + 1$ 时，回溯到 $r$ 后，$r$ 仍有一个未拓展的邻居 $x_b$

跟据 dfs 规则，我们应该去拓展这个邻居 ⇒ $r \to x_b$ 为 tree edge.（与1个children 矛盾）

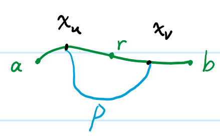
∴ $r$ 的 children $\geq 2$.

补证：

假设：若 $\forall x_i \neq x_j$ 都 $\exists$ 不经过 $r$ 路径 $p$ 连通 $x_i$，$x_j$.




在 $G$ 中取 $\forall$ 包含 $r$ 的路径 $q$ ($r$ 不为端点) 设两端点为 $a$, $b$




⇒ $r$ 不是割点（矛盾）

⇒ $\exists x_j \neq x_T$，删去 $r$ 后不连通.

prove ⇐

若 $G_\pi$ 的根 $r$ 至少2个孩子 (取2个，设为 $x_a$ 与 $x_b$.（不妨 $x_a.expand < x_b.expand$）

设与 $r$ 相邻的节点有 $x_1 \cdots x_k$ ($k \geq 2$)，易知 $x_a.expand < x_a.finish < x_b.expand < x_b.finish$.

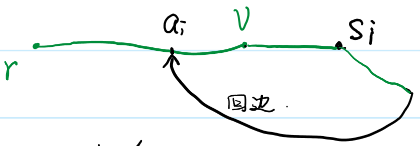
∴ 不存在 path $p$ 不经过 $r$，连通 $x_a$、$x_b$.（否则 $x_a.finish > x_b.finish$，$x_a$ 为 $x_b$ 的祖先）

∴ $r$ 在 $G$ 中为割点.（删去 $r$，$x_a$、$x_b$ 不连通）

2. Let $v$ be a nonroot vertex of $G_\pi$. Prove that $v$ is an articulation point of $G$ if and only if $v$ has a child $s$ such that there is no back edge from $s$ or any descendant of $s$ to a proper ancestor of $v$.

prove ⇒ （反证）假设不存在 $S$，即 $V$ 的所有 child $S_1 \sim S_k$ ($k \geq 1$) 及其子孙都有指向 $V$ 祖先的回边

设 $G_\pi$ 根为 $r$，易知 $r \to V$ 有一条仅由 tree edge 组成的路径，设为 $p$.，$P$ 上的点即为 $V$ 的祖先

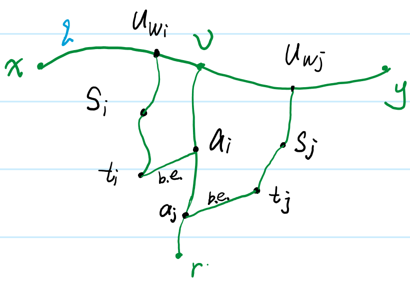



设 $V$ 的所有邻居 $u_1 \sim u_n$ ($n \geq k$)

解释，$t_i$ 为 $S_i$ 及其子孙中某个有指向 $V$ 祖先 $a_i$ 的回边的点（$t_i$ 可以为 $S_i$）

$a_i$可以是$r$但不可以是$v$

则：在 $G$ 中 ∀ 包含 $V$ 的路径 $q$（$V$ 不为端点，设端点为 $x$、$y$）

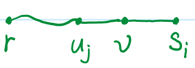



解释：(1) $S_i$ 为随机一个与 $u_{wi}$ 连通的点，$S_j$ 同理

(2) 路径 $x \to u_{wi} \to S_i \to t_i \to a_i$
$\to a_j \to t_j \to S_j \to u_{wj} \to y$，可能走"回头路"

删去 $v$，$x$、$y$ 仍连通.（由于任意性，与割点矛盾）假设不成立

prove ⇐




假设为 $S_i$，假设 $V$ 的父亲为 $u_j$，则删去 $V$ 后
有 $S_i$ 与 $u_j$ 不连通，否则一定有回边，或 $S_i$ 不为 $V$ 的 child
（早已被拓展）
⇒ $V$ 为割点

3. Let

$$v.low = \min \begin{cases} v.d, & \text{discover} \\ w.d : (u,w) \text{ is a back edge for some descendant } u \text{ of } v \end{cases}$$

Show how to compute $v.low$ for all vertices $v \in V$ in $O(E)$ time.

感觉这里的定义有点问题，
这里的表述认为u不可以取v，
但是实际上v.low的定义包含v本身的回边的才比较有用；
如果包含v本身的回边的话，我定义的res其实就是v.low.

```
def ComputeAllLow(G):
    time := 0
    for v in V:
        v.fa := None
    v = random(V)
    Visit(v, v)
```

因为是一个连通的无向无环图，所以只用搜一遍就好，且dfs-tree只有tree edge和back edge

```
def Visit(v, fa):
    v.fa := fa
    time := time + 1
    v.d := time
    v.low := v.d
    res = inf  #自身及子孙的w.d
    for w in edge[v]:
        if w.fa is not None: # we have searched before
            if w != fa: # (w,v) is not tree edge
                res := min(res, w.d)
```

因为是一个连通的无向无环图，所以只用搜一遍就好，且dfs-tree只有tree edge和back edge

整体思路是dfs返回一个值res，这个值记录了自身及其子孙节点的回边指向的最早的祖先的发现时间。如果自身及其子孙节点没有回边，则该值为inf（无穷大）。

故,v.low为所有children的dfs的返回值和v.d的最小值

而在搜索前，我们初始化所有节点的fa(父亲)为空（None），若dfs(v)中搜索w时 fa 为空就证明w未搜索到（v,w）为tree-edge，如果不为空就证明w已经搜索过，若(w,v)不为tree-edge，则(v,w)为back-edge。根据res的定义，很简单写出右侧伪代码

```
if w.fa is not None: # we have searched before
        if w != fa: # (w,v) is not tree edge
            res := min(res, w.d)
    else :
        des_res := Visit(w)
        res :=  min(res, des_res)
        v.low := min(v.low,des_res)
time := time + 1
v.f  := time
return res
```

4. Show how to compute all articulation points in $O(E)$ time.

利用3.的代码 和 1.2的结论

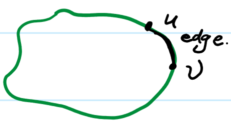
```
ap_set := Empty Set

def ComputeAllArticulation(G)->割点集:
    time := 0
    for v in V:
        v.fa := None
        v.sonNum := 0
    v = random(V)
    Visit(v, v)
```

```
def Visit(v, fa):
    v.fa := fa
    time := time + 1
    v.d := time
    res = inf  #自身及子孙的w.d
    can_be_ap := False
    for w in edge[v]:
        if w.fa is not None: # we have searched before
            if w == fa: continue
            res := min(res, w.d)
        else :
            v.sonNum := v.sonNum + 1
            des_res := Visit(w)
            if des_res == inf : # a child with no backedges
                can_be_ap := True
            res :=  min(res, des_res)
    time := time + 1
    v.f := time
    if v == fa: # v is root
        if v.sonNum > 1: ap_set.append(v)
    else: # v is not root
        if can_be_ap: ap_set.append(v)
    return res
```

5. Prove that an edge of $G$ is a bridge if and only if it does not lie on any simple cycle of $G$.

Prove ⇒.

若 (u,v) 为割边.




反证法：假设 (u,v) 在 G 中的一个简单环上。称 u 到 v 不经过 (u,v) 的路径为 p.

称删去边 (u,v) 后的图为 G'，则对于 G 中任意连通的两点 x, y.

若 x 到 y 的路径包含 (u,v) 则一定存在另一条路径，包含 p 但不包含 (u,v)

故，删去在 G' 中 x,y 仍连通. 由于 x,y 的任意性. 推得 (u,v) 不是割边.（矛盾），

故而 (u,v) 一定不在环上 #

故而 $(u,v)$ 一定在环上 #

prove ⇐

若 $(u,v)$ 不在环上，则删去 $(u,v)$. $u.v$ 不连通（否则 $(u,v)$ 在环上）.
故 $(u,v)$ 为割边

6. Show how to compute all the bridges of $G$ in $O(E)$ time.

利用5.的结论和3.4.的算法生成所有割边。

如果在访问某个节点v时，该节点v有一条指向祖先w的回边，

就证明w,v有环，该环上任意一条边都不为割边。

那么我们在dfs时，返回一个res值，

记录该节点v及其子孙节点的回边指向的最早的祖先的发现时间。

如果自身及其子孙节点没有回边，则该值为inf（无穷大）

如果该节点v的father.d不小于res，则证明(father,v)不为割边

```
bridge_set := Empty Set

def ComputeAllBridge(G)->割边集:
    time := 0
    for v in V:
        v.fa := None
    v = random(V)
    Visit(v, v)
```

```
def Visit(v, fa):
    v.fa := fa
    time := time + 1
    v.d := time
    res = inf  #自身及子孙的w.d
    for w in edge[v]:
        if w.fa is not None: # we have searched before
            if w != fa: res := min(res, w.d)
        else :
            v.sonNum := v.sonNum + 1
            des_res := Visit(w)
            if des_res > v.d: # no circle
                bridge_set.append((v,w))
            res :=  min(res, des_res)
    time := time + 1
    v.f := time
    return res
```

7. Prove that the biconnected components of $G$ partition the nonbridge edges of $G$.

这里关于双连通分量英文表述有点歧义，但是应该指的是边双连通分量的边集：

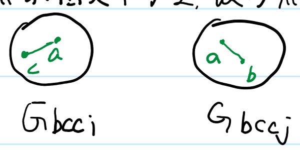
即：若一个无向图中的任意两条边都在一个环上，则称作边双连通图。

一个无向图中的每一个极大边双连通子图称作此无向图的边双连通分量。

假设双连通分量为集合$E_{bcc_i}(1 \le i \le k)$,对应的子图$G_{bcc_i}$为并假设$k \ge 1$且每个集合不为空

1. 首先证明$\forall i, E_{bcc_i}$不含割边
   根据双连通分量的定义我们知道，该集合中的任意两条边都在一个简单环上，
   故，根据5.的结论，$E_{bcc_i}$中不会有割边。
2. 其次证明不在$E_{bcc_1} \dots E_{bcc_k}$的边全为割边
   首先证明任意两个极大边双连通子图的点集交集为空，边集交集为空。

如果点集相交不为空，设为点 $a$，则 $\forall e_i \in E_{bcc_i}$，$e_i$ 与 $ac$ 共环




$\forall e_j \in E_{bcc_j}$. $e_j$ 与 $ab$ 共环

$\Rightarrow$ $e_i$ 与 $e_j$ 共环 $\Rightarrow$ $E_{bcc_i}$ 与 $E_{bcc_j}$ 与极大矛盾.

而既然点集相交为空，
边集自然为空

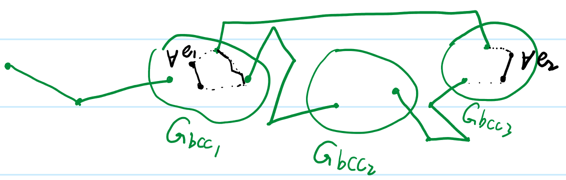
其次定义两个极大边双连通子图相邻指的是

∃路径$u \to v$,满足$u \in G_{bcc_u}, v \in G_{bcc_v}$,且路径上的所有边不在$E_{bcc_1} \dots E_{bcc_k}$中。

断言1：该路径唯一，如果有超过一条路径, 则同上文的证明相似，与“极大”矛盾。

断言2：a与b相邻，b与c相邻，a与c不相邻。

否则如右图，也将与“极大”的定义矛盾。




断言3：G中所有的边分为三类：

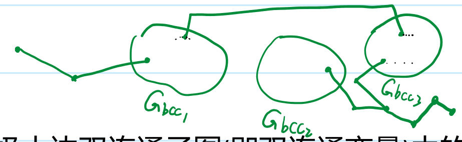
类1：在极大边双连通子图中，

类2：在两个极大边双连通子图的相邻的路径上

类2：在两个极大边双连通子图的相邻的路径上，

类3：不为类2，但是在某个邻居个数为1的点到"最近"的某个极大边双连通子图的点路径上

如下图：




第二者是不在极大边双连通子图(即双连通变量)中的边，

显然第二者一定不在某个环上，即第二者为割边，否则违背断言2；

而证明第三者为割边的证明如下：

原图为G, 我们不断删去所有邻居个数为1的点和连接该点的边，直至没有邻居为1的点，

得到处理后的图G'，显然删去的边为类3。

即要证删去的边为割边。

又因为我们删边(u,v)的时候，该边的某一端点一定只有一条边，所以(u,v)一定是割边

而在连续删边的时候，之前删的边也无法经不包含所删边的路径到达另一端点，

故而删去的边均为割边。

断言3得证。

于是 不在$E_{bcc_1}$ … $E_{bcc_k}$的边全为割边 得证

综上 $\forall i, E_{bcc_i}$不含割边，不在$E_{bcc_1}$ … $E_{bcc_k}$的边全为割边

故而 $E_{bcc_1}$ … $E_{bcc_k}$ 划分了割边。

8. Give an $O(E)$-time algorithm to label each edge $e$ of $G$ with a positive integer $e.bcc$ such that $e.bcc = e'.bcc$ if and only if $e$ and $e'$ belong to the same biconnected component.

利用7.的结论容易推得：

连接两个边双连通分量的边即是割边

在基于6.的算法生成所有双连通分量的划分。

使用一个计数器cnt,初始值为1，

如果是割边，则cnt+1,如果不是,则e.bcc = cnt;

```
    cnt_idx := 0

def MarkBCC(G)->割边集:
    time := 0
    for v in V:
        v.fa := None
    v = random(V)
    Visit(v, v)
```

```
def Visit(v, fa):
    v.fa := fa
    time := time + 1
    v.d := time
    res = inf  #自身及子孙的w.d
    for w in edge[v]:
        if w.fa is not None: # we have searched before
            if w != fa: res := min(res, w.d)
        else :
            v.sonNum := v.sonNum + 1
            des_res := Visit(w)
            if des_res > v.d: # no circle
                cnt_idx := cnt_idx + 1
            else :
                edge(v,w).bcc = cnt_idx
            res :=  min(res, des_res)
    time := time + 1
    v.f := time
    return res
```

## 2 Analyze topological sort

Am alternative algorithm for topological sort was given at our course without correctness proof. Can you prove the correctness of this algorithm using the properties of source and sink? And try to analyze the time complexity of this algorithm.

### Alternative Algorithm for Topological Sort

(1) Find a source node $s$ in the (remaining) graph, output it.

(2) Delete $s$ and all its outgoing edges from the graph.

(3) Repeat until the graph is empty.

Formal proof of correctness?
How efficient can you implement it?

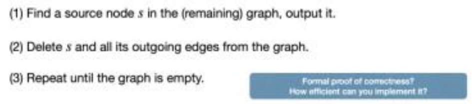



**Figure 2** The alternative algorithm for topological sort in slide 14 *Application of DFS*

证明:

1. 首先 一个能进行拓扑排序的图一定没有环，所以一定存在一个source node 和 sink node
2. 其次，拓扑序的定义在于，任意一条有向边(u,v)，拓扑序中u一定在v之前出现
   而当一个点v为source点时，就不存在指向v的有向边，
   也就是本身不存在指向v的有向边, 或者所有的u已经被删除了(也就**满足拓扑序的定义**)
3. 每次删掉一个source点和由该点出发的所有边，并不会影响"没有环"这个循环不变性，
   故而下次一定能找到一个source node并将其删除，而由于点的有限性，循环必定会**终止**。

故而该算法能够终止且得到的序列不违背拓扑序的性质，那么算法正确。

时间复杂度：O(n+m)
设 点数为n，边数为m。
（1）遍历边，使用hash计算并统计每个点的入度 时间O(m)
（2）遍历点，找出入度为0的点集加入队列q中 时间O(n)
（3）q中弹出点x，将x通向的所有点y入度减一，如果为0加入队列q，直到q为空；
　　实际上相当于是每条边遍历一遍每个点遍历了一遍，时间O(n+m)
综上 时间复杂度为 O(n+m);

## 3 *Implement Prim's Algorithm with Fibonacci Heap

*Note: This question is optional.*

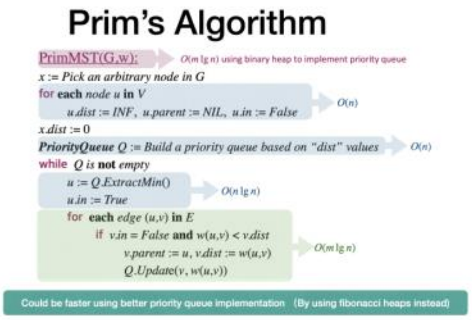



**Figure 3** Prim's algorithm in slide 15 *Minimum Spanning Tree*

We've mentioned that Prim's Algorithm can be implemented with a priority queue. Try to implement Prim's Algorithm with Fibonacci Heaps [1].

感觉只是在优先队列的是实现上，从使用二叉堆转为使用斐波那契堆即可，具体接口名并不会变化。
然而在插入上，平均时间复杂度 从二叉堆的O(lg n)变为O(1).
这里附上 所看博客的链接，实现上未自己实现，但是将博客中的Java改写成cpp了。
[数据结构——斐波那契堆 - luanxm - 博客园 (cnblogs.com)]