# Biconnectivity

# BICONNECTIVITY

We now consider an application of depth-first search to determining the biconnected components of an undirected graph.

Let $G = (V, E)$ be a connected, undirected graph.

A vertex $a$ is said to be an articulation point (точка сочленения) of $G$ if there exist vertices $v$ and $w$ such that $v$, $w$, and $a$ are distinct, and every path between $v$ and $w$ contains the vertex $a$.

Stated another way, $a$ is an articulation point of $G$ if removing $a$ splits $G$ into $\geq 2$ parts.

The graph $G$ is biconnected (двусвязен) if for every distinct triple of vertices $v, w, a$ there exists a path between $v$ and $w$ not containing $a$.

Thus an undirected connected graph is biconnected $\Leftrightarrow$ it has no articulation points.

We can define a natural relation on the set of edges of $G$ by saying that two edges $e_l$ and $e_2$ are related if

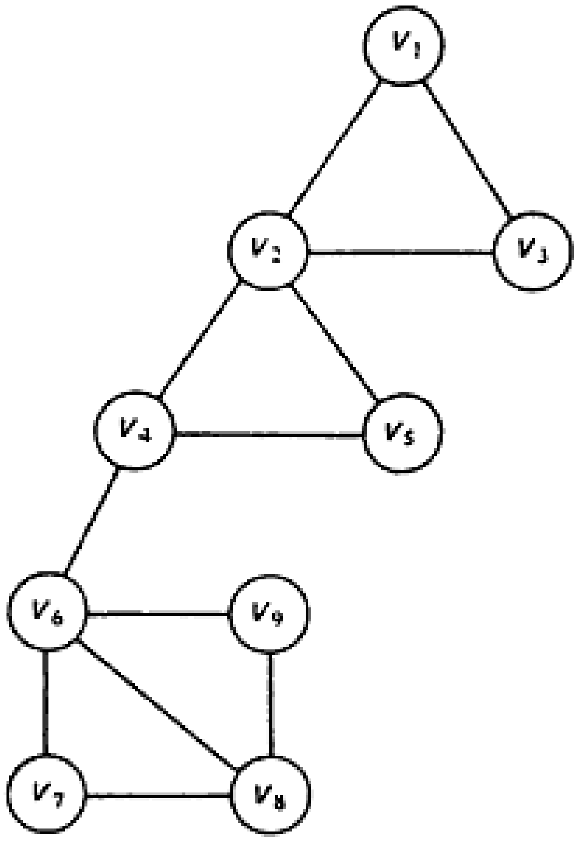
$e_1 = e_2$ or there is a cycle containing both $e_1$ and $e_2$.

It is easy to show that this relation is an equivalence relation that partitions the edges of $G$ into equivalence classes $E_1$ , $E_2$ , … , $E_k$ such that two distinct edges are in the same class ⇔ if they lie on a common cycle.

For $1 \leq i \leq$ k, let $V_i$ be the set of vertices of the edges in $E_i$.

Each graph $G_i = (V_i, E_i)$ is called a biconnected component of $G$.

Example. Consider the undirected graph bellow. How many biconnected components are shown?



Solution There are 4 biconnected components.

Vertex $v_4$ , for example, is an articulation point, since every path between $v_1$ and $v_7$ passes through $v_4$

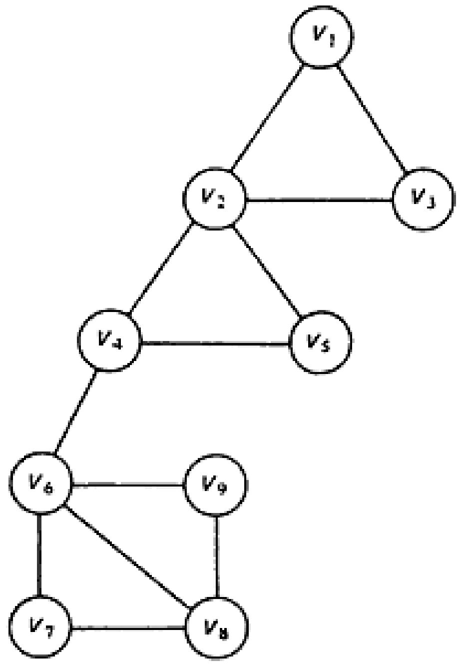
The equivalence classes of edges lying on common cycles are

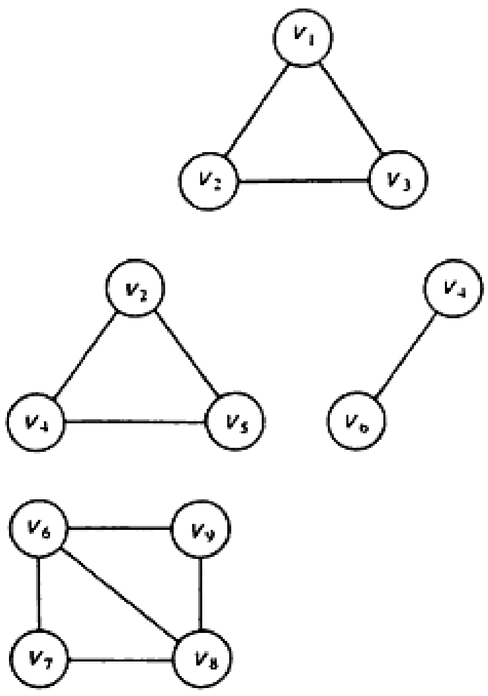
$\{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}\}$,

$\{\{v_2, v_4\}, \{v_2, v_5\}, \{v_4, v_5\}\}$,

$\{\{v_4, v_6\}\}$,

$\{\{v_6, v_7\}, \{v_6, v_8\}, \{v_6, v_9\}, \{v_7, v_8\}, \{v_8, v_9\}\}$.



# Properties of biconnected components

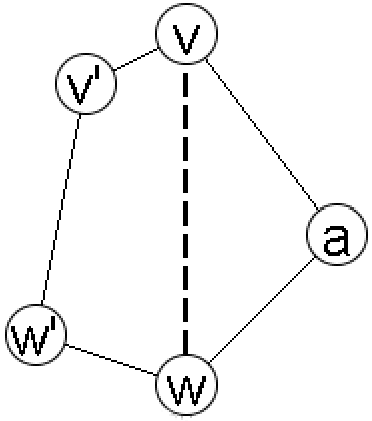
Lemma 1. For $1 \leq I \leq k$, let $G_i = (V_i, E_i)$ be the biconnected components of a connected undirected graph $G = (V, E)$.

Then

1. $G_i$ is biconnected for each $i$, $1 \leq i \leq k$.
2. For all $i \neq j$, $V_i \cap V_j$ contains at most 1 vertex.
3. $a$ is an articulation point of $G$ $\Leftrightarrow$ $a \in V_i \cap V_j$ for some $i \neq j$.

1) $G_i$ is biconnected for each $i$, $1 \le i \le k$.

Proof




1. Suppose there are 3 distinct vertices $v$, $w$, and $a$ in $V$ such that all paths in $G_i$ between $v$ and $w$ pass through $a$.

Then surely $(v, w)$ is not an edge in $E_i$. Thus there are distinct edges $(v, v')$ and $(w, w')$ in $E_i$ and there is a cycle in $G_i$ including these edges.

By the definition of a biconnected component, all edges and vertices on this cycle are in $E_i$ and $V_i$, respectively.

Thus there are 2 paths in $G_i$ between $v$ and $w$, only 1 of which could contains $a$,

a contradiction.

2. For all $i \neq j$, $V_i \cap V_j$ contains at most 1 vertex.

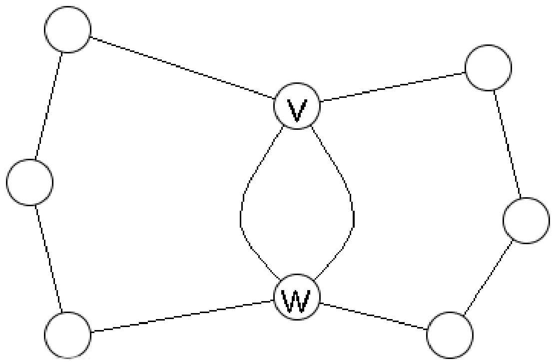
Suppose 2 distinct vertices $v$ and $w$ are in $V_i \cap V_j$.

Then there exists a cycle $C_1$ in $G_i$ that contains $v$ and $w$, and a cycle $C_2$ in $G_j$ that also contains $v$ and $w$.

Since $E_i$ and $E_j$ are disjoint, the sets of edges in $C_1$ and $C_2$ are disjoint.

However, we may construct a cycle containing $v$ and $w$ that uses edges from both $C_1$ and $C_2$, implying that at least 1 edge in $E_i$ is equivalent to an edge in $E_j$.

Thus $E_i$ and $E_i$ are not equivalence classes, as supposed.



3. $a$ is an articulation point of $G$ $\Leftrightarrow$ $a \in V_i \cap V_j$ for some $i \neq j$.

=>Suppose vertex $a$ is an articulation point of $G$.

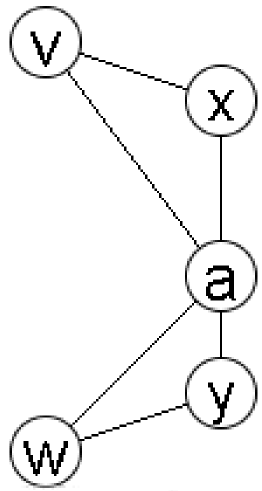
Then there exist 2 vertices $v$ and $w$ such that $v$, $w$, and $a$ are distinct and every path between $v$ and $w$ contains $a$.

Since $G$ is connected, there is at least 1 such path.

Let $\{x, a\}$ and $\{y, a\}$ be the 2 edges on a path between $v$ and $w$ incident with $a$.

If there is a cycle containing these 2 edges then there is a path between $v$ and $w$ not containing $a$.

Thus $\{x, a\}$ and $\{y, a\}$ are in different biconnected components, and $a$ is in the intersection of their vertex sets.

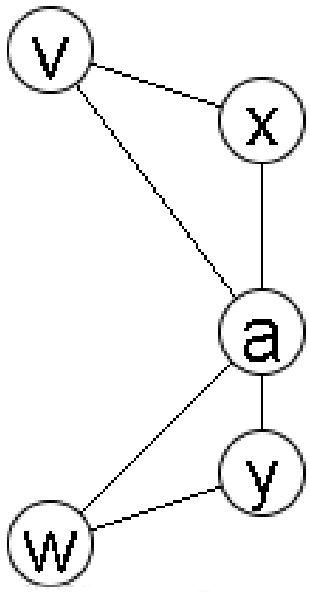


<= If $a \in V_i \cap V_j$ then there are edges $\{x, a\}$ and $\{y, a\}$ in

$E_i$ and $E_j$ respectively.

Since both these edges do not occur on any one cycle, it follows that every path from $x$ to $y$ contains $a$.

Thus $a$ is an articulation point.



Depth-first search is particularly useful in finding the biconnected components of an undirected graph.

One reason for this is that there are no "cross edges."

That is if vertex $v$ is neither an ancestor nor a descendant of vertex $w$ in the spanning forest then there can be no edge from $v$ to $w$.

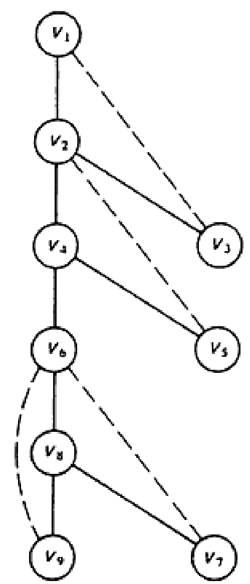
If vertex $a$ is an articulation point, then the removal of $a$ and all edges incident with $a$ splits the graph into 2 or more parts.

One consists of a son $s$ of a and all of its descendants in the depth-first spanning tree.

Thus in the depth-first spanning tree $a$ must have a son $s$ such that there is no back edge between a descendant of $s$ and a proper ancestor of $a$.

Conversely, with the exception of the root of the spanning tree, the absence of cross edges implies that vertex $a$ is an articulation point if there is no back edge from any descendant of some son of $a$ to a *proper ancestor of a*.

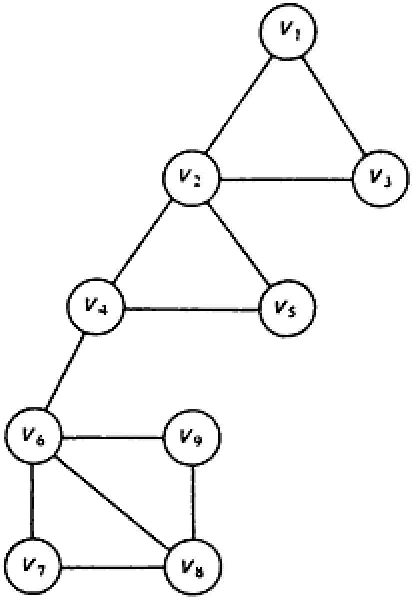
The root of the depth-first spanning tree is an articulation point ⇔ it has 2 or more sons.



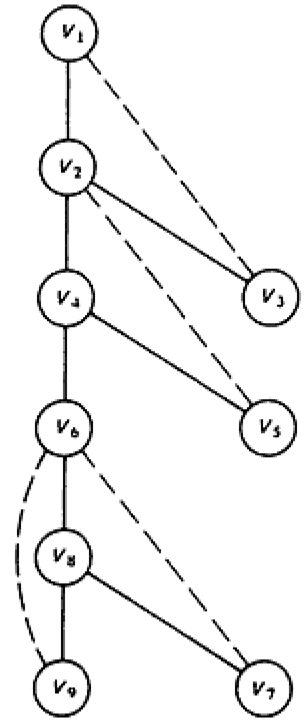
Example. A depth-first spanning tree for the graph (a) is shown in (b). The articulation points are $v_2$, $v_4$ and $v_6$ .

Vertex $v_2$ has son $v_4$ , and no descendant of $v_4$ has a back edge to a proper ancestor of $v_2$ .

Likewise, $v_4$ has son $v_6$ , and $v_6$ has son $v_8$ with the analogous property.



(a)



(b)

The preceding ideas are embodied in the following lemma.

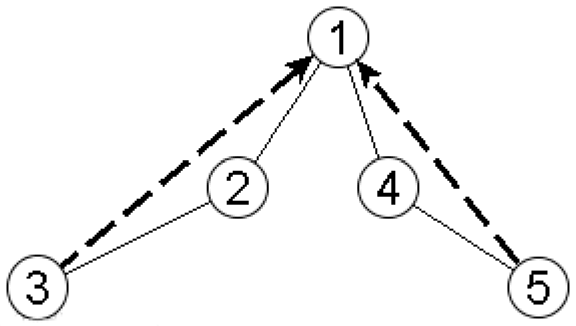
Lemma 2. Let $G = (V, E)$ be a connected, undirected graph, and let $S = (V, T)$ be a depth-first spanning tree for $G$.

Vertex $a$ is an articulation point of $G$ $\Leftrightarrow$ either

(1) $a$ is the root and $a$ has $> 1$ son,

or

(2) $a$ is not the root and for some son $s$ of $a$ there is no back edge between any descendant of $s$ (including $s$ itself) and a proper ancestor of $a$.




It is easy to show that the root is an articulation point ⇔ it has > 1 son. (An exercise).

2=> Suppose condition 2 is true.

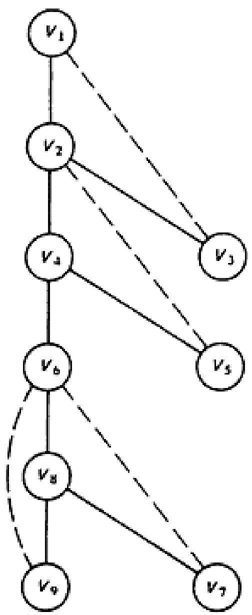
Let $p$ be the parent of $a$. Each back edge goes from a vertex to an ancestor of the vertex.

Thus any back edge from a descendant $v$ of $s$ goes to an ancestor of $v$.

By the hypothesis of the lemma the back edge cannot go to a proper ancestor of $a$.

Hence it goes either to $a$ or to a descendant of $s$.

Thus every path from $s$ to $p$ contains $a$, implying that $a$ is an articulation point.



<= To prove the converse, suppose that $a$ is an articulation point but not the root.

Let $x$ and $y$ be distinct vertices other than $a$ such that every path in $G$ between $x$ and $y$ contains $a$.

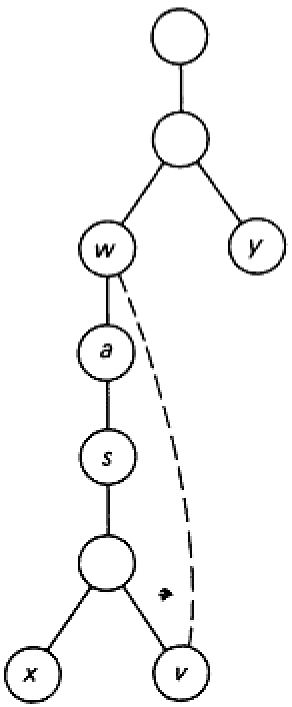
At *least one of* $x$ *and* $y$, *say* $x$, *is a proper descendant of* $a$ in $S$, else there is a path in $G$ between $x$ and $y$ using edges in $T$ and avoiding $a$.

Let $s$ be the son of $a$ such that $x$ is a descendant of $s$ (perhaps $x = s$).

Either there is no back edge between a descendant $v$ of $s$ and a proper ancestor $w$ of $a$, in which case condition 2 is immediately true,

or there is such an edge $\{v, w\}$.

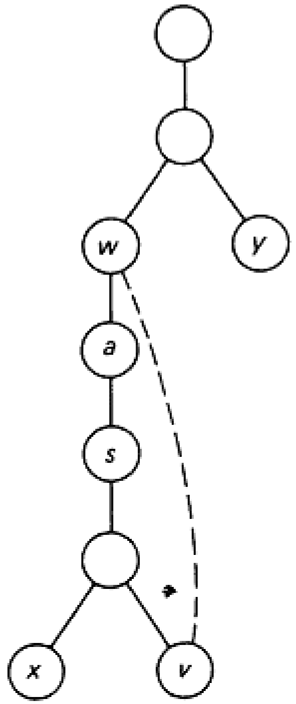
In the latter situation we must consider 2 cases.




(a)

CASE 1. Suppose $y$ is not a descendant of $a$.

Then there is a path from $x$ to $v$ to $y$ that avoids $a$, a contradiction.




(a)

CASE 2. Suppose $y$ is a descendant of $a$.

Surely $y$ is not a descendant of $s$, else there is a path from $x$ to $y$ that avoids $a$.

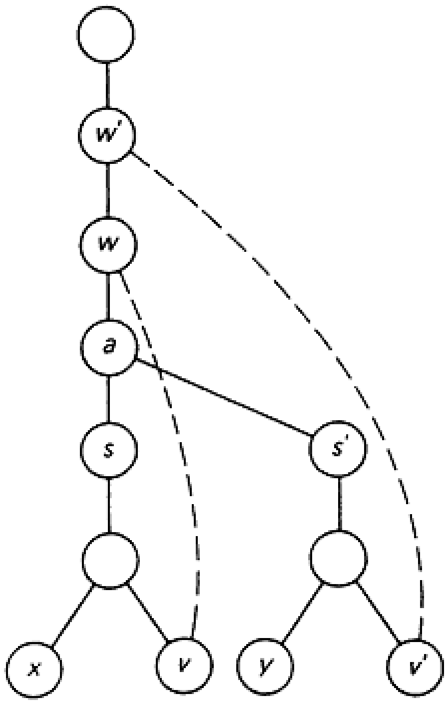
Let $s'$ be the son of a such that $y$ is a descendant of $s'$.

Either there is ho back edge between a descendant $v'$ of $s'$ and a proper ancestor $w'$ of $a$, in which case condition 2 is immediately true,

or there is such an edge ( $v'$, $w'$).

In the latter case there is a path from $x$ to $v$ to $w$ to $w'$ to $v'$ to $y$ that avoids $a$, a contradiction.

We conclude that condition 2 is true.



Let $T$ and $B$ be the sets of tree and back edges produced by a depth-first search of a connected, undirected graph $G = (V, E)$.

We assume the vertices in $V$ are named by their depth-first numbers ( $v.d$ ).

For each $v$ in $V$, we define

$v.Low = \text{MIN}(\{v\} \cup \{w \mid$ there exists a back edge $\{x, w\} \in B$

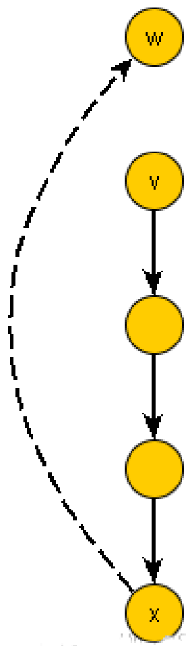
- such that $x$ is a descendant of $v$,
- and $w$ an ancestor of $v$ in the depth
- first spanning forest $(V, T)\})$

(1)

The preorder numbering implies that if $x$ is a descendant of $v$ and $\{x, w\}$ is a back edge such that $w < v$, then $w$ is a *proper ancestor of v.*

Thus by Lemma 2, if vertex $v$ is not the root, then $v$ is an articulation point $\Leftrightarrow$ $v$ has a son $s$ such that

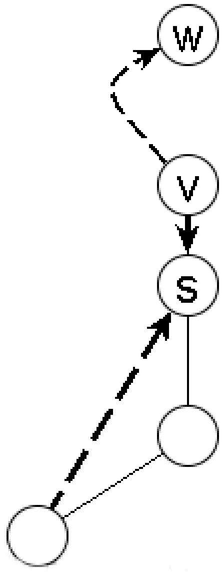
$s.Low \geq v$.



We can embed into the procedure DFS_visit a calculation to determine the *Low* value of each vertex if we rewrite (1) to express *v.Low* in terms of the vertices adjacent to *v*, via back edges and the values of *Low* at the sons of *v*.

Specifically, *v.Low* can be computed by determining the minimum value of those vertices *w* such that either

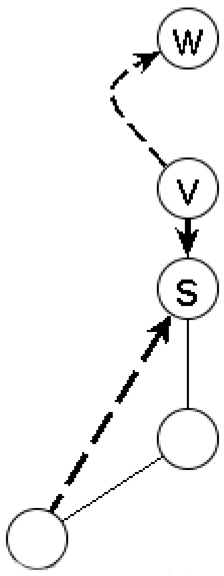
1. $w = v$, or
2. $w =$ s.*Low* and *s* is a son of *v*, or
3. $(v, w)$ is a back edge in *B*.



The minimum value of $w$ can be determined once Adj[$v$], the list of vertices adjacent to $v$, is exhausted.

Thus (1) is equivalent to

$v.Low = \mathrm{MIN}(\{v\} \cup \{s.Low \mid s \text{ is a son of } v\} \cup \{w \mid \{v, w\} \in B\})$.
(2)



# DFS_visit with LOW computation

Procedure *DFS_Visit_Biconn(v);*

*v.color* = GRAY;

*v.d = time*;

*time = time*+1;

*v.Low = v.d;*

# DFS with LOW computation (continued)

```
for each vertex w ∈ Adj[v] do {
   if w.color == "WHITE" then{ /* (v, w) is a tree edge*/
        w.π = v /* add (v, w) to T and place on stack*/
        DFS_Visit_Biconn(w);
        if w.Low ≥ v.d  then /* a biconnected component has been found,  pop
          from stack all edges up to and including (v,w)
        v.Low = MIN(v.Low, w.Low);
      } /* end of the tree edge processing*/
  else if v.π != w then /*(v, w) is a back edge*/
      v.Low =MIN(v.Low, w.d);
 }
}
```

We have incorporated both the renaming of the vertices by first visit and the computation of *LOW* into the revised version of DFS_visit (algorithm 2).

First, we initialize *v.LOW* to its maximum possible value.

If vertex *v* has a son *w* in the depth-first spanning forest, then we adjust *v.LOW* if *w.LOW* is less than the current value of *v.LOW*.

If vertex *v* is connected by a back edge to vertex *w*, then we make *v.LOW* be *w.d* if the depth-first number of vertex *w* is less than the current value of *v.LOW*.

The test checks for the case that (*v, w*) is not really a back edge because *w* is the parent of *v* on the depth-first spanning tree.

Having found *v.LOW* for each vertex *v*, we can easily identify the articulation points.

Algorithm 3. Finding biconnected components.

Input. A connected, undirected graph $G = (V, E)$.

Output. A list of the edges of each biconnected component of $G$.

1. Initially set $T$ to $\varnothing$ and time to 0.

Also, mark each vertex in $V$ as being "WHITE."

Then select an arbitrary vertex $v_0$ in $V$ and call DFS_Visit_Biconn($v_0$) to build a depth-first spanning tree $S = (V, T)$ and to compute v.$LOW$ for each $v$ in $V$.

When vertex $w$ is encountered by DFS_Visit_Biconn put edge $(v, w)$ on STACK, if it is not already there.

After discovering a pair $(v, w)$ such that $w$ is a son of $v$ and $w.LOW \geq v$, pop from STACK all edges up to and including $(v, w)$.

These edges form a biconnected component of $G$.

(Note that if $(v, w)$ is an edge, $v$ is on $Adj[w]$ and $w$ is on $Adj[v]$. Thus $(v, w)$ is encountered twice, once when vertex $v$ is visited und once when vertex $w$ is visited.

We can test whether $(v, w)$ is already on STACK by checking if $v < w$ and $w$ is "old" or if $v > w$ and $w = v.\pi$ .

Example. The depth-first spanning tree of *G* is shown with the vertices renamed by *v*.*d* and the values of *Low* indicated.

For example, *DFS_Visit_Biconn*(6) determines that 6.*Low* = 4, since back edge (6, 4) exists.

Then *DFS_Visit_Biconn*(5), which called *DFS_Visit_Biconn*(6), sets 5.*Low* = 4, since 4 is less than the initial value of 5.*Low*, which is 5.

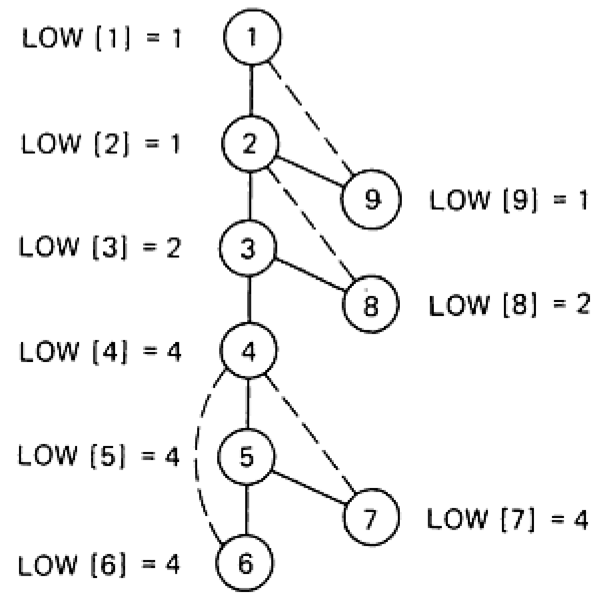
On completion of *DFS_Visit_Biconn*(5) we discover that 5.*Low* = 4.

Thus 4 is an articulation point.

At this point the stack contains the edges (from bottom to top)

(1, 2) (2, 3), (3, 4), (4, 5), (5, 6), (6, 4). (5, 7), (7, 4).

Thus we pop the edges down to and including (4, 5).

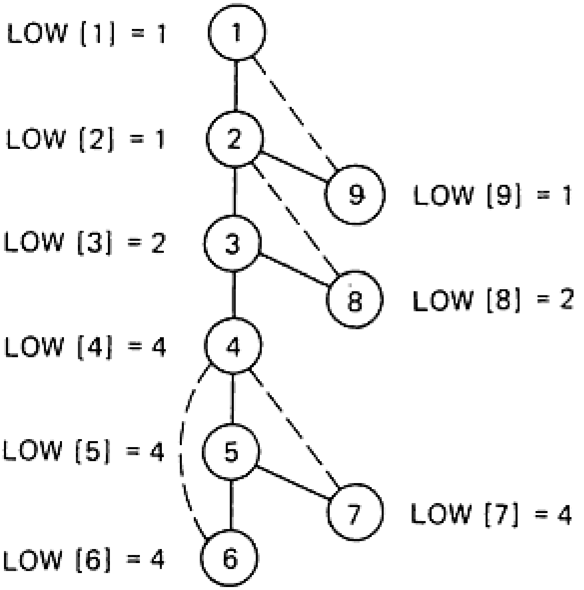


That is, we output the edges (7, 4), (5, 7), (6, 4), (5, 6), and (4, 5) which are the edges of the first biconnected component found.

Observe that on completion of *DFS_visit_Biconn*(2) we discover that

*2.Low* = 1 and empty the stack of edges even though 1 is not an articulation point.

This insures that the biconnected component containing the root is emitted.



Theorem 3. Algorithm Finding Biconnected components correctly finds the biconnected components of $G$ and requires $O(e)$ time if $G$ has $e$ edges.

Proof: The proof that step 1 requires $O(e)$ time is a simple extension of that observation for *DFS_visit* .

Step 2 examines each edge once, places it on a stack, and subsequently pops it.

Thus step 2 is $O(e)$.

For the correctness of the algorithm, Lemma 2 assures us that the articulation points are correctly identified.

Even if the root is not an ·articulation point it is treated as one in order to emit the biconnected component containing the root.

We must prove that if $w.Low \geq v$, then when *DFS_Visit_Biconn*($w$) is completed the edges above $\{v, w\}$ on STACK will be exactly those edges in the biconnected component containing $(v, w)$.

This is done by induction on the number $b$ of biconnected components of $G$.

The basis, $b = 1$ is trivial since in this case $v$ is the root of the tree, $\{v, w\}$ is the only tree edge out of $v$, and on completion of *DFS_Visit_Biconn* ($w$) all edges of $G$ are on STACK.

Now, assume the induction hypothesis is true for all graphs with $b$ biconnected components, and let $G$ be a graph with $b + 1$ biconnected components.

Let *DFS_Visit_Biconn* ($w$) be the first call *of DFS_Visit_Biconn()* to end with $w.Low \geq v.d$, for $\{v, w\}$ a tree edge.

Since no edges have been removed from STACK the set of edges above $\{v, w\}$ on STACK is the set of all edges incident with descendants of $w$.

It is easily shown that these edges are exactly the edges of the biconnected component containing $\{v, w\}$.

On removal of these edges from STACK, the algorithm behaves exactly as it would on the graph $G'$ that is obtained from $G$ by deleting the biconnected component with edge $\{v, w\}$.

The induction step now follows since $G'$ has $b$ biconnected components.