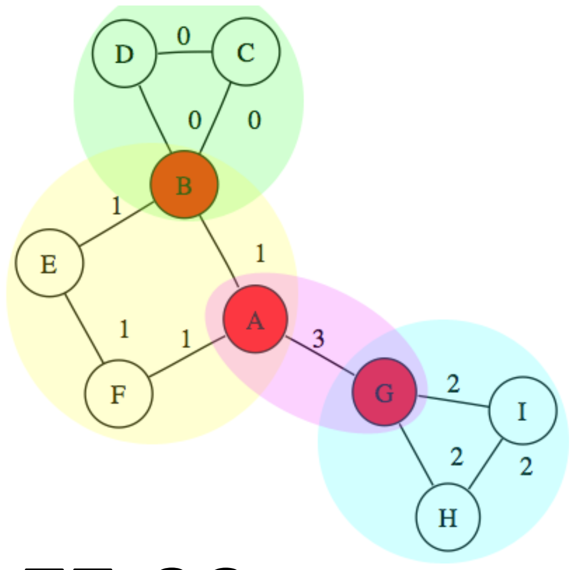
Articulation Points and Biconnected Components

Textbook Reading

Algorithms: Special Topics

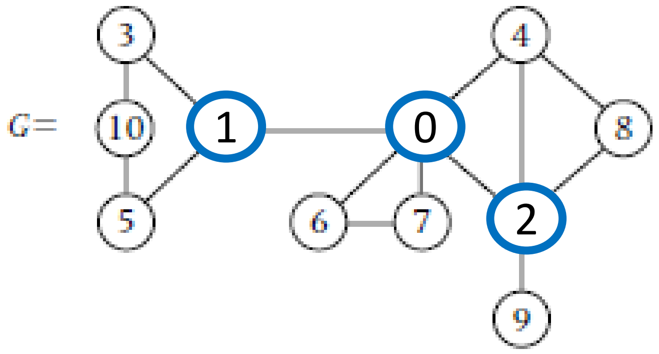
Chapter 3, Section 3.2, pp. 77-86



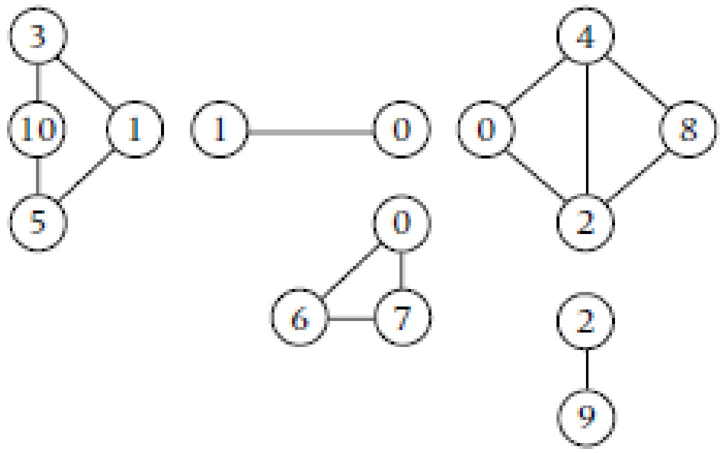
Articulation Points and Biconnected components

- Let $G = (V, E)$ be a connected graph.
- A vertex $v \in V$ is an **articulation point** (also called **cut vertex**) if the graph $G - v$ obtained from G by deleting vertex v and all incident edges is not connected.
- A **biconnected component** of G (also called a **block** of G) is a maximal subgraph B such that B has no articulation points.

Articulation Points and Biconnected Components for an Example Connected Graph



Graph *G* with articulation points {0,1,2}



Biconnected components of *G*

Network Vulnerability

- Graph model networks
- An articulation point represents potential vulnerability in the network.
- If an adversary should attack an articulation point and cause it to fail or if it fails for some other reason, the network becomes disconnected.

PSN. Test for Articulation Point

Design and analyze an algorithm for determining whether a vertex *v* is an articulation point, i.e., cut vertex.

Finding all Articulation Points

- One way to find all the articulation points is to scan all the vertices and apply the articulation point test at each vertex.
- The drawback is that this straightforward algorithm is not efficient. It has worst-case complexity

$$W(n) \in \Theta(mn)$$

Equivalent Conditions to Biconnected

Theorem 3.2.1. Let G be a connected graph with at least three vertices. Then, the following statements are equivalent:

1. The graph G is biconnected.
2. Every two vertices u and v belong to a common cycle. Equivalently, there exist two internally disjoint u–v paths.
3. Every vertex and edge belong to some common cycle.
4. Every two edges belong to some common cycle.
5. Given two vertices u, v and an edge e, there is a u–v path containing e.
6. Given three distinct vertices u, v, w of G, there exists a u–v path containing w.
7. Given three distinct vertices u, v, w of G, there exists a u–v path not containing w.

Faster Algorithm for Finding All Articulation Points

- We now design a faster algorithm ***ArticulationPoints*** for finding all the articulation points of a connected graph.
- *ArticulationPoints* is based on computing two numberings of the vertices based on a single depth-first search.
- The first numbering, ***DFSNum***, is simply the order in which the vertices are visited by a depth-first search rooted at vertex *r*.
- The second numbering, ***Lowest***, can be computed recursively at the same time and is defined in terms of the notion of a bypass edge.

Observation 1

A **cross edge** is an edge which joins two vertices that do not have any ancestor or descendant relationship between them, i.e., joins two vertices of different branches of the tree.

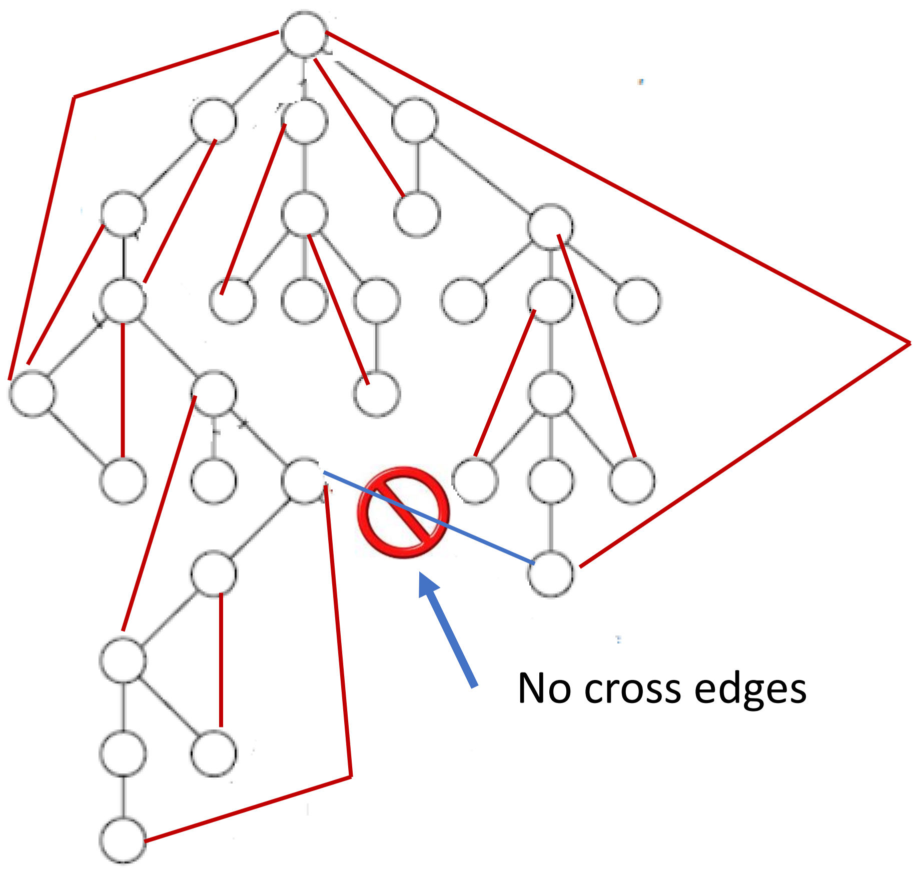
The Depth First Search tree T_r has **no** cross edges.

Observation 2

The root r is an articulation point in G iff r has two or more children in T_r.

Black edges belong to DFS Tree T_r and red edges belong to G but not T_r

G =

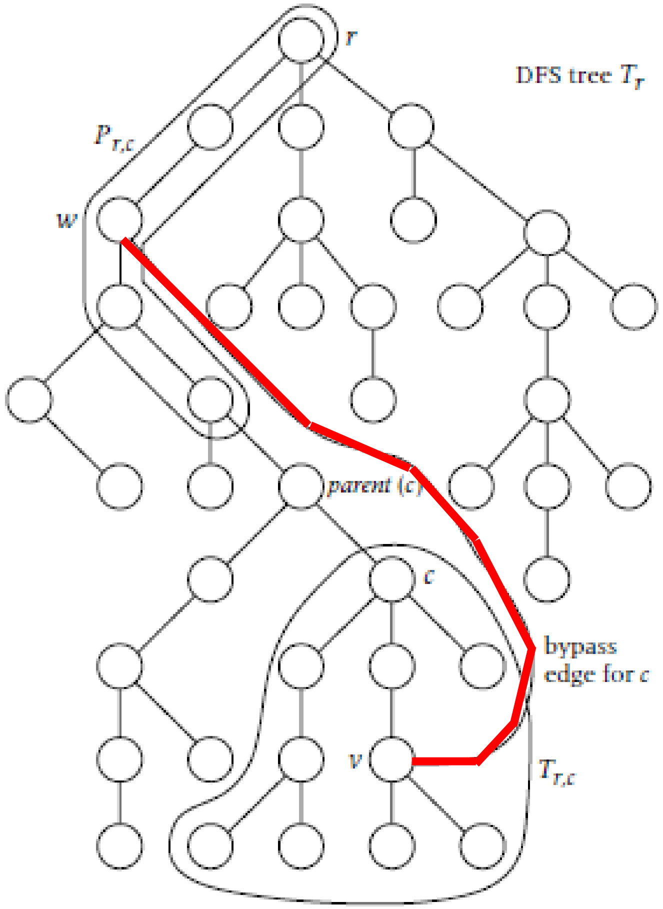


- Note that there are no cross edges.
- Since root r has 3 children r is an articulation point.

Bypass Edge

- Let $T_{r,c}$ denote the subtree of T_r having root vertex c, and let $P_{r,c}$ denote the path in T_r joining r to $parent(parent(c))$.
- A **bypass edge** for c is an edge $vw \in E$ such $v \in T_{r,c}$ and $w \in P_{r,c}$.

Illustration of Bypass Edge



DFS-tree T_r, vertex c, subtree $T_{r,c}$, path $P_{r,c}$, and bypass edge vw for c

Observation

Given a vertex u in T_r different from r, u is an articulation point iff u has a child c in T_r that has no bypass edge.

Lowest Weighting

- Define $Lowest[c]$ to be the minimum of the *DFS* numbers of c and all vertices $w \in P_{r,c}$ for which there is a non-tree edge *vw*, where *v* is either c or a descendent of c in T_r and *w* is an ancestor of c in T_r .

- It is immediate that *c* has a bypass edge iff

$$Lowest[c] < DFSNum(parent(c)).$$

Observations

Let *p* be any vertex at depth at least two in the tree T_r and let *c* be a child of *p.*

Observation 1. c has a bypass edge iff

$$Lowest[c] \; < \; DFSNum(parent(c)).$$

Observation 2. p is an articulation point iff for some child *c*

$$Lowest[c] \; \geq \; DFSNum(parent(c)).$$

Recurrence Relation for Lowest

$$Lowest[v] = min\begin{cases} \min\{Lowest[c]: c\ is\ a\ child\ of\ \ v\ in\ T_r\ , \\ \min\{DFSNum[w]: vw \in E, w \neq partent(v)], \\ DFSNum[v]. \end{cases}$$

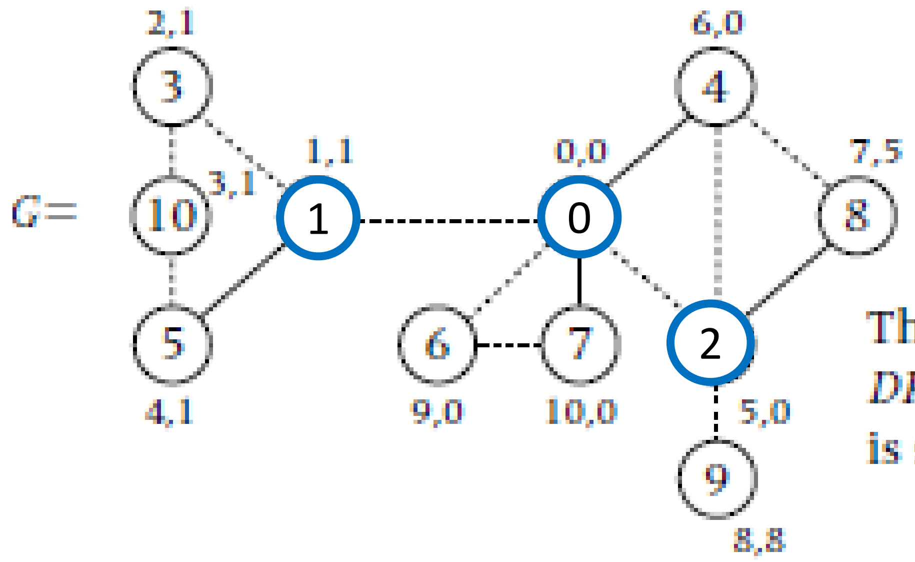
Pseudocode fore computing *DFSNum* and *Lowest*

```
procedure DFSNumberings(G, v) recursive
Input: G (a connected graph with n vertices and m edges)
       v (a vertex of G) //v is root vertex r of DFS on initial call
       Mark[0:n – 1] (global array initialized to zeros)
       TreeDFS[0:n – 1] (global array initialized to zeros)
       DFSNum[0:n – 1] (global array)
       Lowest[0:n – 1] (global array)
       i (global integer initialized to -1)
Output: the DFS and Lowest numberings of the vertices of G, and the parent
implementation of the depth-first search tree TreeDFS($T_r$) rooted at vertex r
       $i \leftarrow i + 1$
       $Mark[v] \leftarrow 1$
       $DFSNum[v] \leftarrow i$
       for each u adjacent to v do
           if $Mark[u] = 0$ then
               $TreeDFS[u] \leftarrow v$
             DFSNumberings(G,u)
           endif
       endfor
       $Min1 \leftarrow \{Lowest[c]$ for each child c of $v\}$
       $Min2 \leftarrow$ min $\{DFSNum[w]$ : v adjacent to w but w is not the parent of $v\}$
       $Lowest[v] \leftarrow$ min $\{DFSNum[v], Min1, Min2\}$
end DFSNumberings
```

Pseudocode for *ArticulationPoints*

```
procedure ArticulationPoints(G,Art)
Input: G (a connected graph with n vertices and m edges)
Output:     Art (the set of articulation points)
dcl TreeDFS[0:n – 1], DFSNum[0:n – 1], Lowest[0:n – 1], Mark[0:n – 1]
    for $v \leftarrow 0$ to $n - 1$ do  //initialize Mark and TreeDFS
        $Mark[v] \leftarrow 0$
        $TreeDFS[v] \leftarrow 0$
    endfor
    $i \leftarrow 0$
    $Art \leftarrow \varnothing$
    DFSNumberings(G,0)  // compute TreeDFS[u], DFSNum[u], //Lowest[u]
                        // for each vertex $u \in V$ use Corollary 3.2.3 to test
                        // whether root $r = 0$ is an articulation point,
                        // that is, check to see if vertex 0 has more than one
                        //child in $T_1$
    $Number \leftarrow 0$
    $c \leftarrow 1$
    while $Number < 2$ .and. $c \leq n$ do
        if $TreeDFS[c] = 1$ then  //c is a child of root vertex $r = 1$
            $Number \leftarrow Number + 1$
        endif
        $c \leftarrow c + 1$
    endwhile
    if $Number = 2$ then        //root vertex is an articulation point
        $Art \leftarrow Art \cup \{0\}$
    endif
    for $c \leftarrow 1$ to $n - 1$ do
        $v \leftarrow TreeDFS[c]$
        if $Lowest[c] \geq DFSNum[v]$ .and. $v \neq 0$ then
            $Art \leftarrow Art \cup \{v\}$
        endif
    endfor
end ArticulationPoints
```

Values of *DFSNum* and *Lowest* for *G* and the array *TreeDFS*[0:9] for our example graph



The pair *DFSNum*[*v*], *Lowest*[*v*] is shown outside each node *v*.

i	0	1	2	3	4	5	6	7	8	9	10
TreeDFS[*i*]	−1	0	0	1	2	10	0	6	4	2	3

Complexity Analysis

- *ArticulationPoints* involves performing a depth-first search to compute the depth-first search numbers and lowest numbers, which has worst-case complexity $\Theta(m + n)$. Then, it scans the vertices comparing the lowest number of a vertex to the depth-first search number of its parent, which takes time $\Theta(n)$.
- Thus, *ArticulationPoints* has worst-case complexity

$$W(n) \in \Theta(m + n).$$

Biconnected Knock-Knock Joke