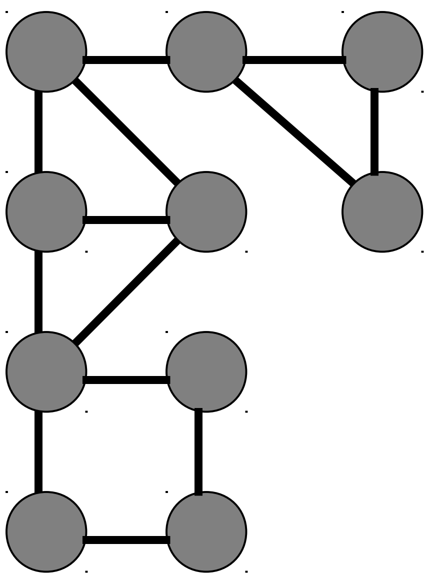
Applications of DFS: Articulation Points and Biconnected Components

Articulation Point

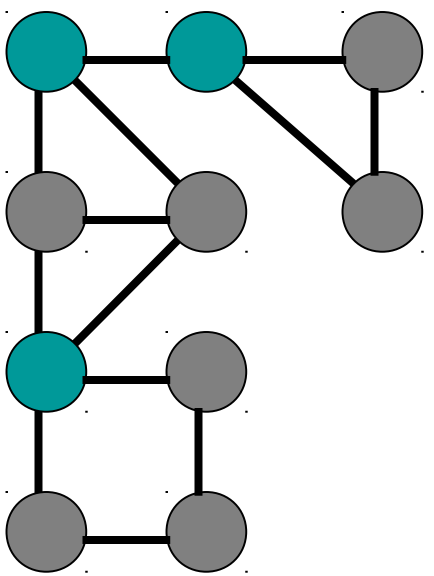
- Let G = (V,E) be a connected undirected graph.

Articulation Point: is any vertex of G whose removal results in a disconnected graph.



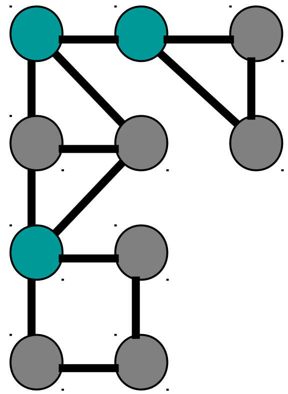
Articulation Point

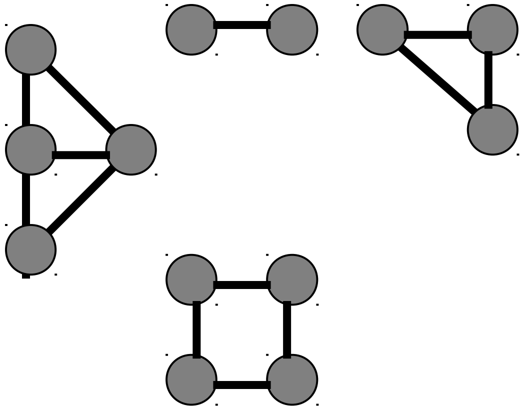
Articulation Point: is any vertex of G whose removal results in a disconnected graph.



Biconnected components

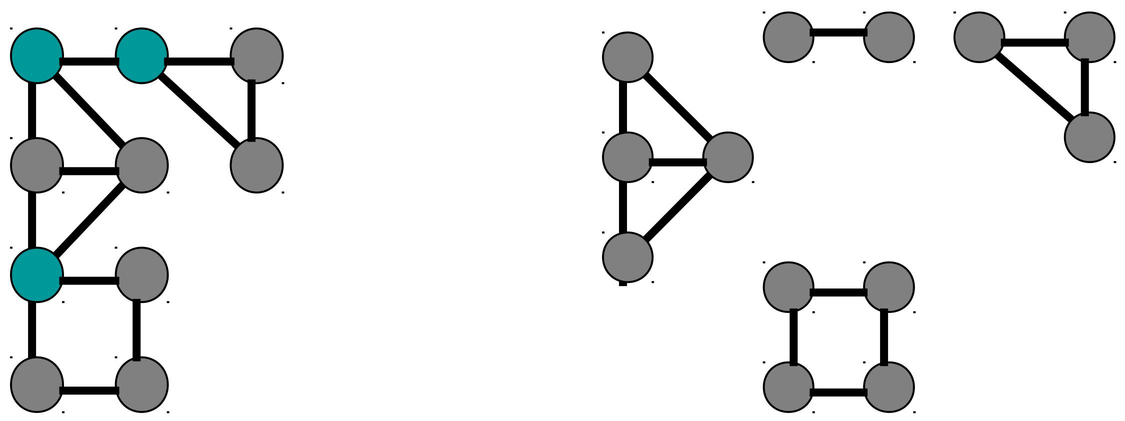
- A graph is biconnected if it contains no articulation points.
- Two edges are cocyclic if they are equal or if there is a simple cycle that contains both edges. (Two different ways of getting from one edge to the other)
  - This defines an equivalence relation on the edges of the graph
- Biconnected components of a graph are the equivalence classes of cocyclicity relation





Biconnected components

- A graph is biconnected if and only if it consists of a single biconnected component
  - No articulation points



Articulation points and DFS

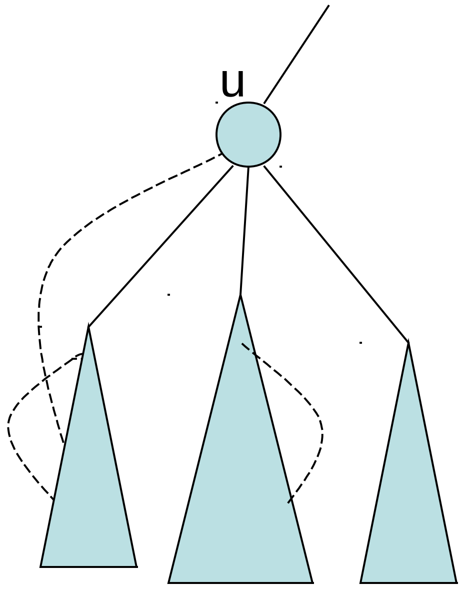
- How to find articulation points?
  - Use the tree structure provided by DFS
  - G is undirected: tree edges and back edges (no difference between forward and back edges, no cross edges)

- Assume G is connected

internal vertex u

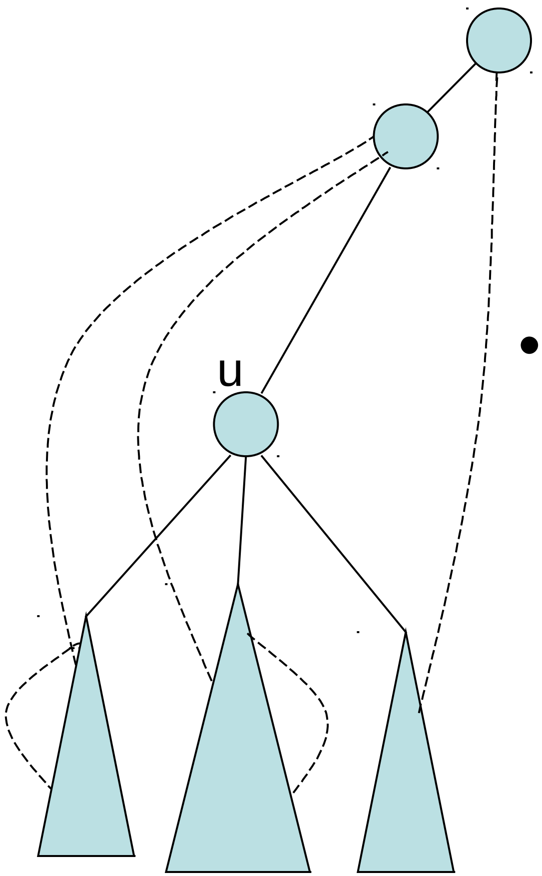
- Consider an internal vertex u
  - Not a leaf,
  - Assume it is not the root

- Let v1, v2,…, vk denote the children of u
  - Each is the root of a subtree of DFS
  - If for some child, there is no back edge from any node in this subtree going to a proper ancestor of u, then u is an articulation point



Here u is an articulation point

internal vertex u



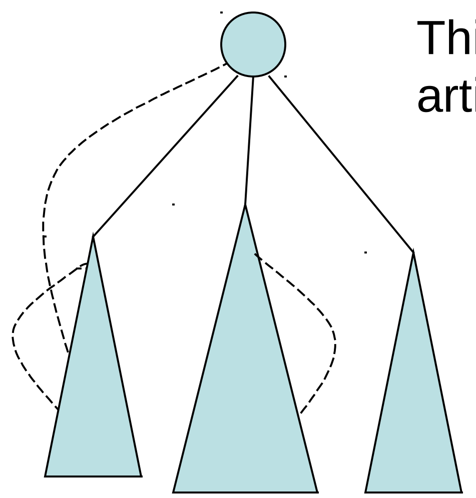
- Here u is not an articulation point
  - A back edge from every subtree of u to proper ancestors of u exists

What if u is a leaf

- A leaf is never an articulation point
- A leaf has no subtrees..

What about the root?

- the root is an articulation point if an only if it has two or more children.
  - Root has no proper ancestor
  - There are no cross edges between its subtrees



This root is an articulation point

How to find articulation points?

- Keep track of all back edges from each subtree?
  - Too expensive

- Keep track of the back edge that goes highest in the tree (closest to the root)
  - If any back edge goes to an ancestor of u, this one will.

- What is closest to root?
  - Smallest discovery time

Define Low[u]

- Low[u]: minimum of d[u]

  and

  {d[w] | where (v,w) is a back edge and v is a (nonproper) descendent of u.}

Computing Low[u]

- Initialization:

  Low[u] = d[u]

- When a new back edge (u, v) is detected:

  Low[u] = min(Low[u], d[v])

- Tree edge (u, v):

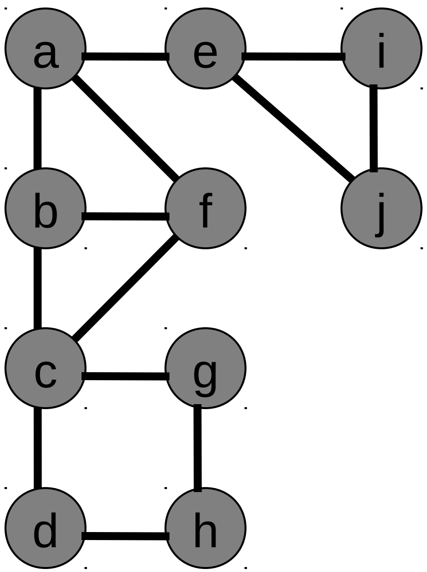
  Low[u] = min(Low[u], Low[v])

- Once Low[u] is computed for all vertices u, we can test whether a nonroot vertex u is an articulation point

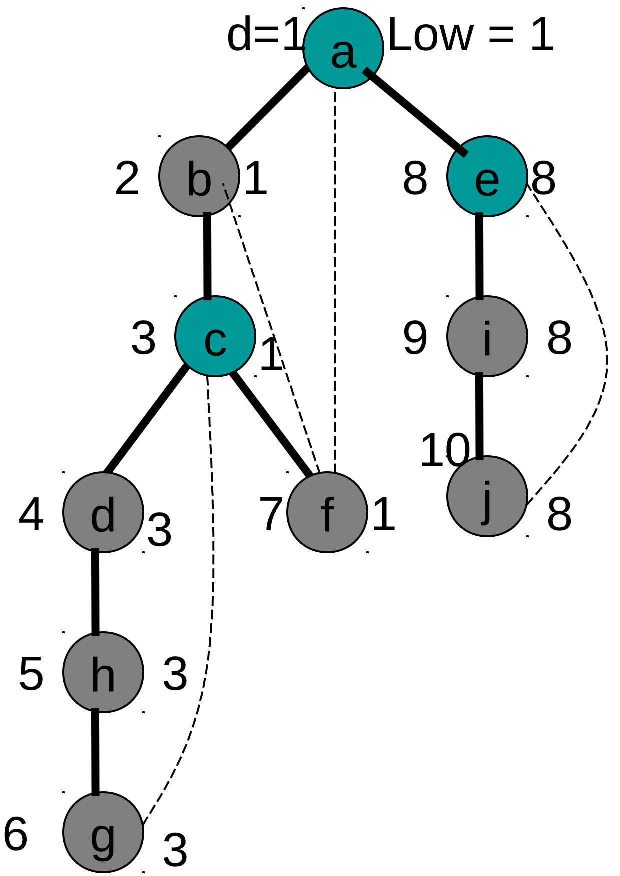
- u is an articulation point iff it has a child v for which Low[v] >= d[u]

```
DFS(G)
    for each (u in V) do
            color[u] = white
            pred[u] = NIL
    time = 0
    for each (u in V) do
        if (color[u] == white)
            findArticPts(u)
```

```
findArticPts(u)   //vertex u is just discovered
    color[u] = gray
    Low[u] = d[u] = ++time
    for each (v in Adj[u]) do {
        if (color[v] == white) then   { //(u, v) is a tree edge
            pred[v] = u
                findArticPts(v)
                Low[u] = min(Low[u], Low[v])  //update Low[u]
            if (pred[u] == NIL)  {//u is root
                    (if v is u's second child)
                            add u to set of articulation points
            }
              else if (Low[v] >= d[u]) // if there is no back edge to an ancestor of u
                    add u to set of articulation points
        }
        else if (v != pred[u]) //(u, v) is a back edge
                Low[u] = min(Low[u], d[v])  //this back edge goes closer to the root
    }
```



a
e
i
b
f
j
c
g
d
h



d=1
a
Low = 1
2
b
1
8
e
8
3
c
1
9
i
8
10
j
8
4
d
3
7
f
1
5
h
3
6
g
3