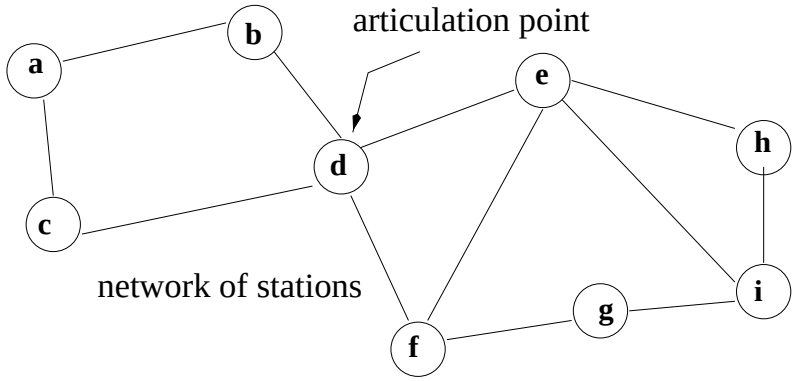
Articulation Points and Biconnected Components

Suppose you are a terrorist, seeeking to distroy a telephone network !!!

Which station would you blow up ?



- **Articulation point**

  - Any vertex whose removal results in a disconnected graph

  - If *v* is an articulation point, then there exist distinct vertices *w* and *x* such that *v* is in every path from *w* to *x*

- **Biconnected graph**

  - Any graph which contains no articulation points

  - Biconnected graphs have at least **two vertex-disjoint** paths between any pair of vertices

  - To disconnect a biconnected graph, **we must remove at least two vertices**

  - We can generalize the above concepts to *k*-connected graphs (*k* vertices must be removed to disconnect the graph)

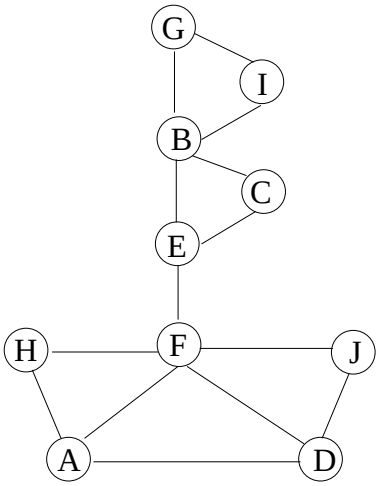
- **Biconnected components of a graph**

  - A maximal biconnected subgraph (i.e., not contained in any larger biconnected subgraph)

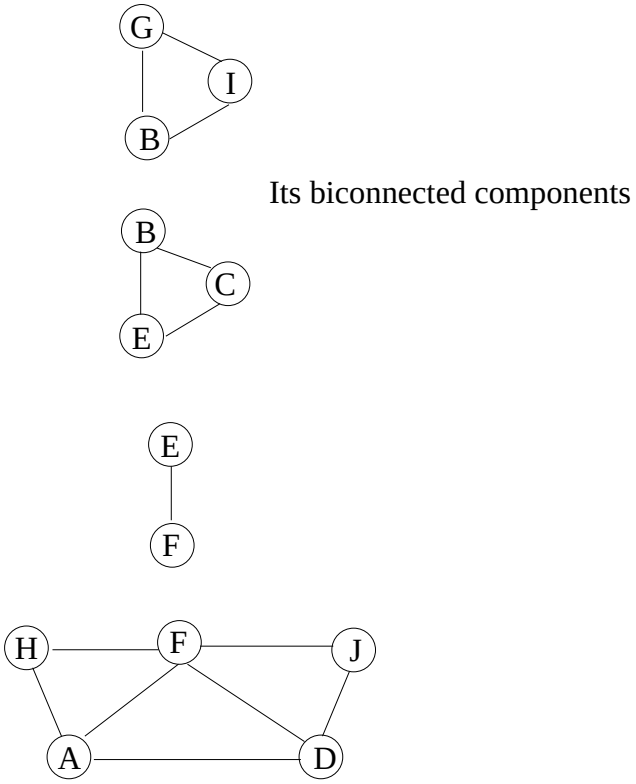
  - Note that biconnected components partition the edges (not the vertices !!) into disjoint sets

  - DFS can be used to find the biconnected components of a graph

A graph



Its biconnected components



- **A brute-force approach to find articulation points**

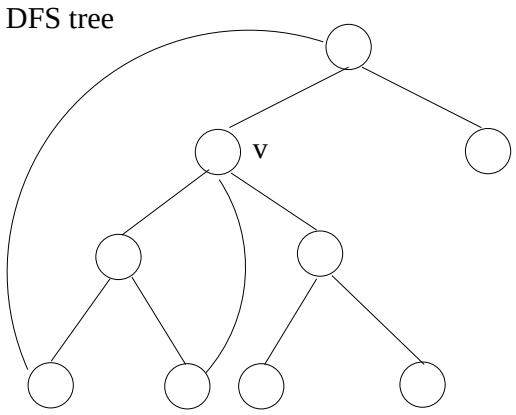
  1. Delete a vertex

  2. Apply DFS on the remaining vertices to see if the graph is connected

  3. Choose a new vertex and apply steps (1) and (2)

  - Running time: O(V(V+E))
  (not very efficient, it can be done in O(V+E) time !!!)

- **Theorem**: In a depth-first search tree, a vertex *v*, other than the root, is an articulation point **iff** *v* is not a leaf and some subtree of *v* has no back edge to a proper ancestor of *v*



v is an articulation point since the right subtree of v does not have a back edge to a proper ancestor !!

(===>) Suppose that *v* is an articulation

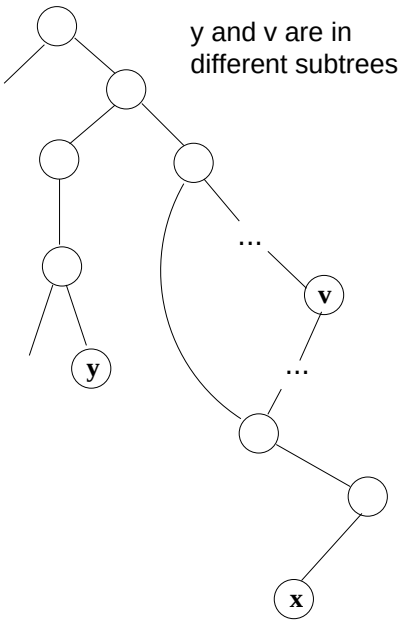
There exist *x* and *y* such that *v* is in every path from *x* to *y*

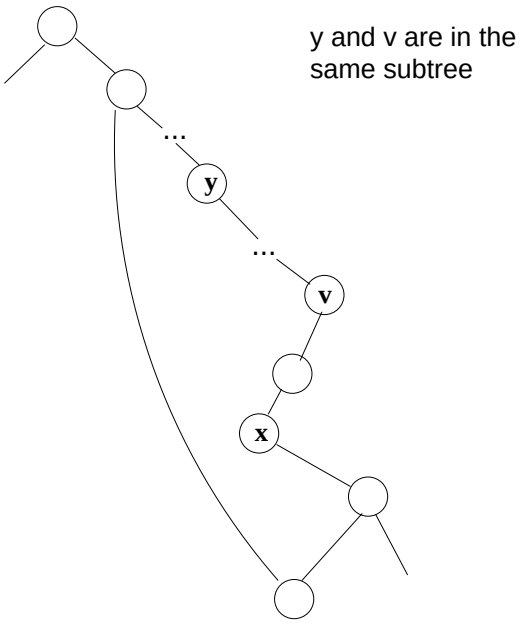
At least one of *x* and *y* must be a descendant of *v* (or *v* will not be in the path)

This implies that *v* cannot be a leaf

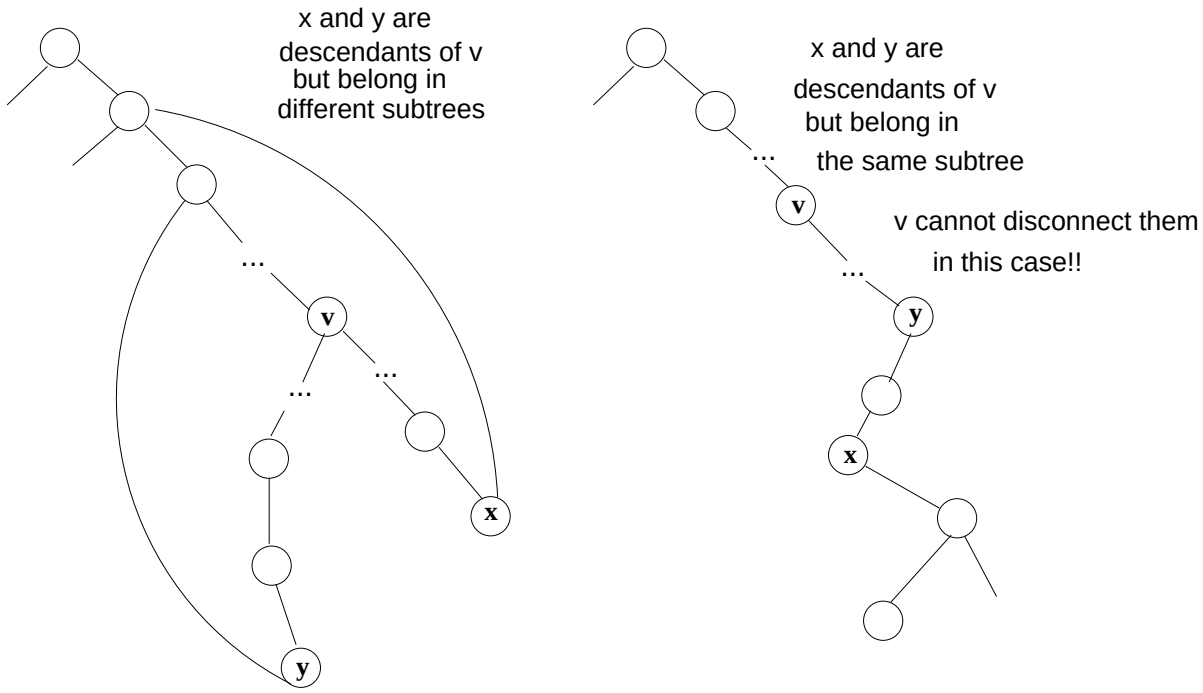
(proof by contradiction)
Suppose that every subtree has a back edge to an ancestor of *v*

Case 1: only one of the *x* and *y* is a descendant of *v*





Case 2: both *x* and *y* are descendants of *v*



Both cases lead us to the conclusion that *v* is not an articulation point **(Contradiction !!)**

(<===) Suppose that some subtree of *v* has no back edge to a proper ancestor of *v*

Since *v* is not a leaf, there exist vertices *x* and *y* such that *v* is in the path from *x* to *y*

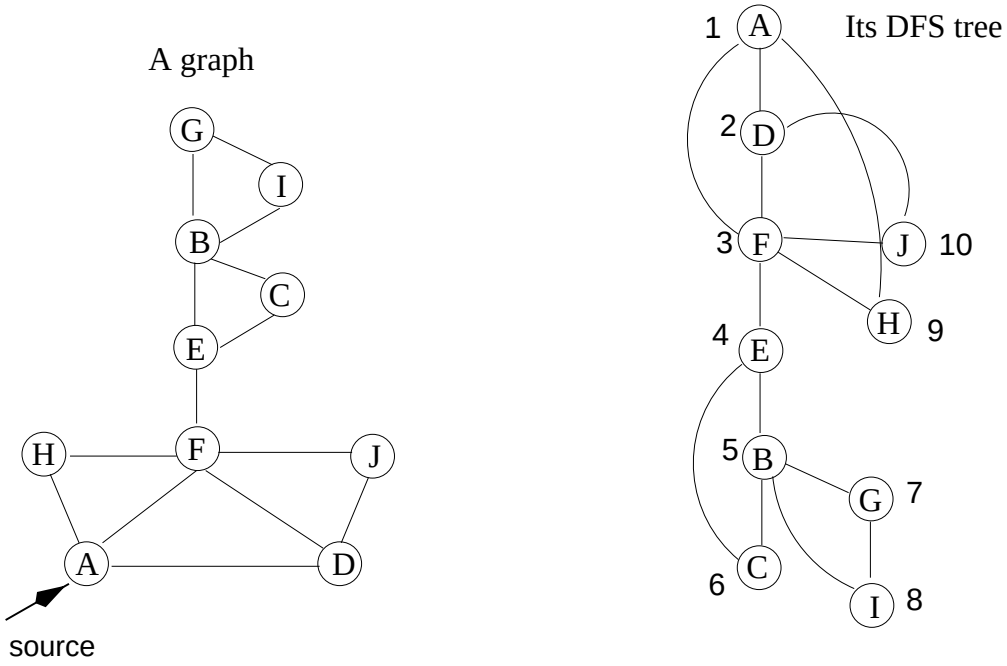
There is only one path from *x* to *y* since there are no back edges

Removing *v* will disconnect *x* from *y*; thus, *v* is an articulation point !!

- **Algorithm** (for undirected graphs)

  - Apply DFS to compute the DFS tree

  - *Key idea*: keep track of how far back in the tree one can get from each vertex by following tree edges and certain back edges



  - First, assign a number to each vertex based on the time at which the vertex is visited for the first time (*discovery[v]*)

  - Then, keep track of how back in the tree one can get from a vertex (*back[v]*)

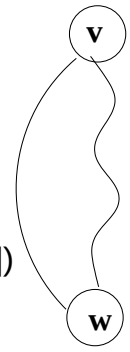
  - *back[v]* is initialized to *discovery[v]* in the beginning

  - How should we update *back[v]* and how should we detect articulation points ?

Case 1

back-edge

back[w] = min(back[w], discovery[v])



Case 2

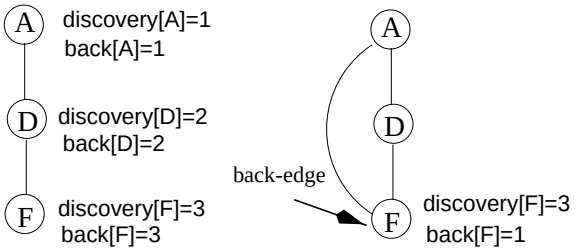
backtrack

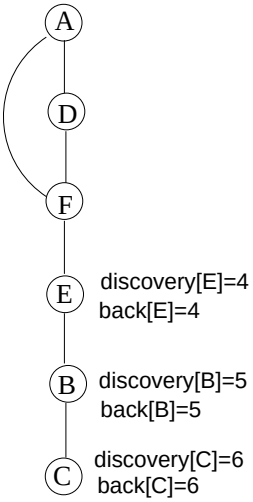
if back[w] < discovery[v]

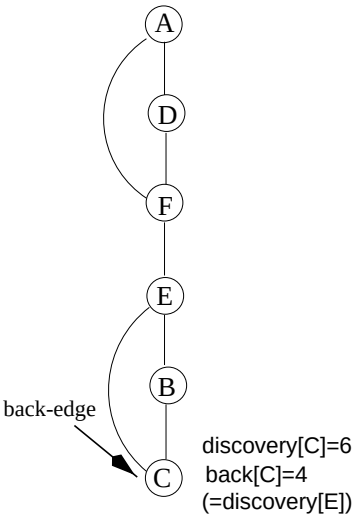
back[v] = min(back[v], back[w])

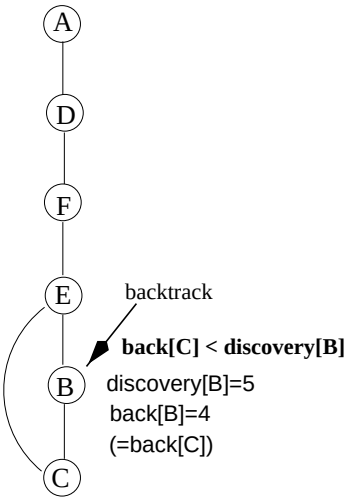
else if back[w] >= discovery[v]

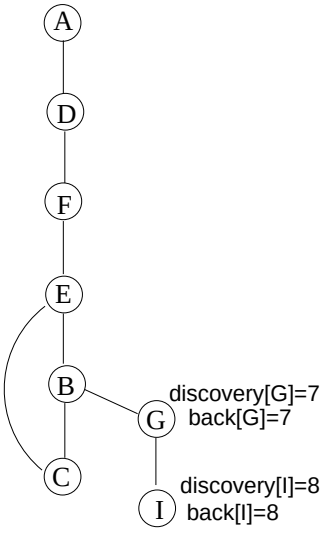
v is an articulation point !!

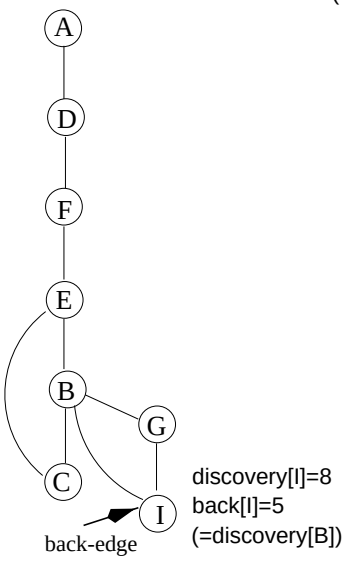


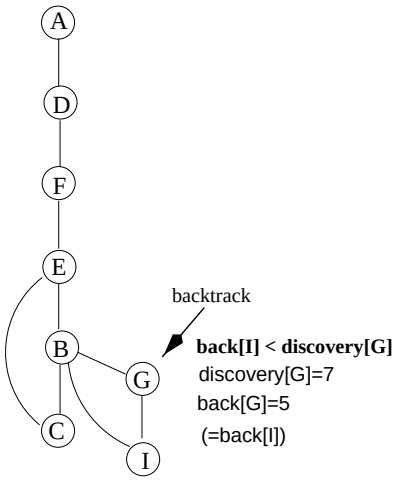


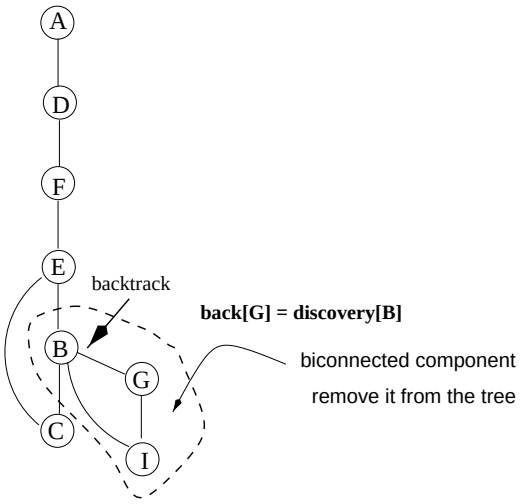


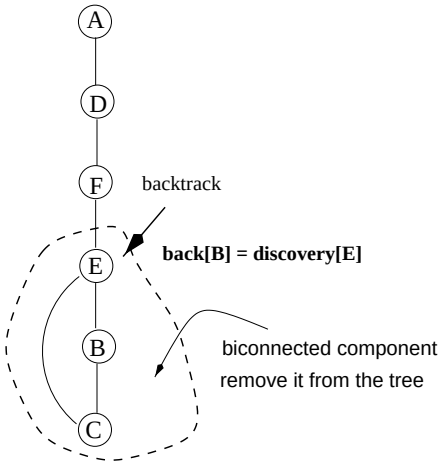


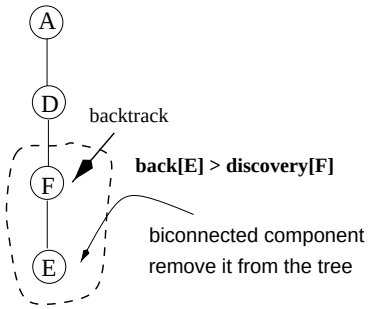


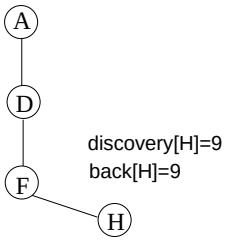


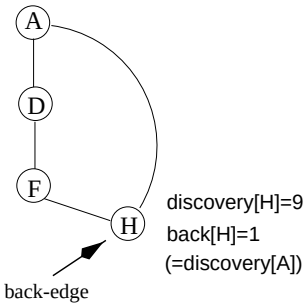

A
D
F
E
B
G
C
I
backtrack
back[I] < discovery[G]
discovery[G]=7
back[G]=5
(=back[I])

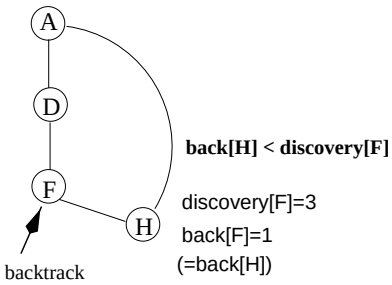

A
D
F
E
B
G
C
I
backtrack
back[G] = discovery[B]
biconnected component
remove it from the tree

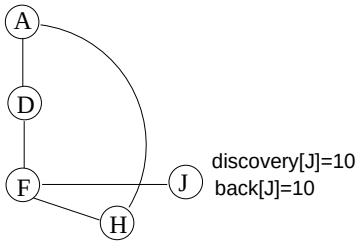

A
D
F
E
B
C
backtrack
back[B] = discovery[E]
biconnected component
remove it from the tree

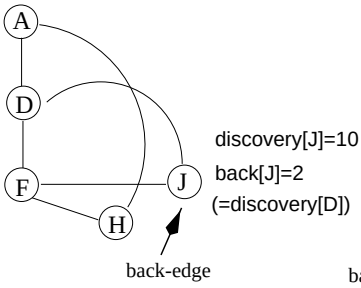

A
D
F
E
backtrack
back[E] > discovery[F]
biconnected component
remove it from the tree

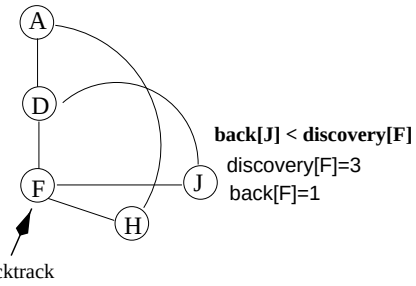

A
D
F
H
discovery[H]=9
back[H]=9

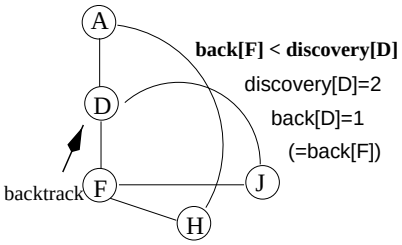

A
D
F
H
discovery[H]=9
back[H]=1
(=discovery[A])
back-edge

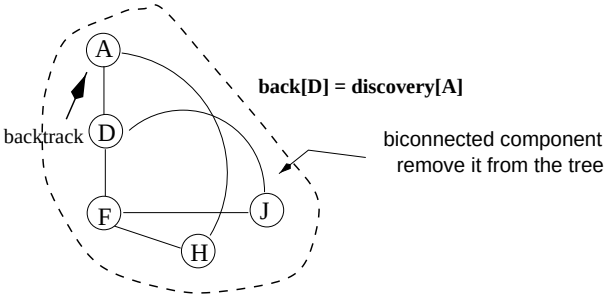

A
D
F
H
back[H] < discovery[F]
discovery[F]=3
back[F]=1
(=back[H])
backtrack


A
D
F
J
H
discovery[J]=10
back[J]=10


A
D
F
J
H
discovery[J]=10
back[J]=2
(=discovery[D])
back-edge


A
D
F
J
H
back[J] < discovery[F]
discovery[F]=3
back[F]=1
backtrack


A
D
F
J
H
back[F] < discovery[D]
discovery[D]=2
back[D]=1
(=back[F])
backtrack


A
D
F
J
H
back[D] = discovery[A]
biconnected component
remove it from the tree
backtrack