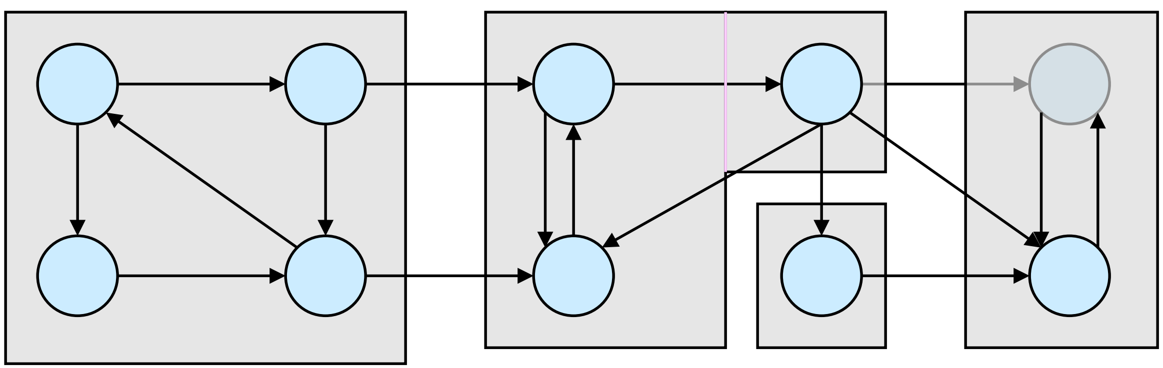
Application of DFS

- Strongly Connected Components
  - Component graph
  - Transpose of directed graph
  - Algorithm to compute SCC
- Biconnectivity
- Articulation Points
- Bridges

Strongly Connected Components

- G is strongly connected if every pair (u, v) of vertices in G is reachable from one another
- A **strongly connected component** (***SCC***) of G is a maximal set of vertices $C \subseteq V$ such that for all $u, v \in C$, both $u \rightsquigarrow v$ and $v \rightsquigarrow u$ exist
  - i.e. vertices u and v are reachable from each other

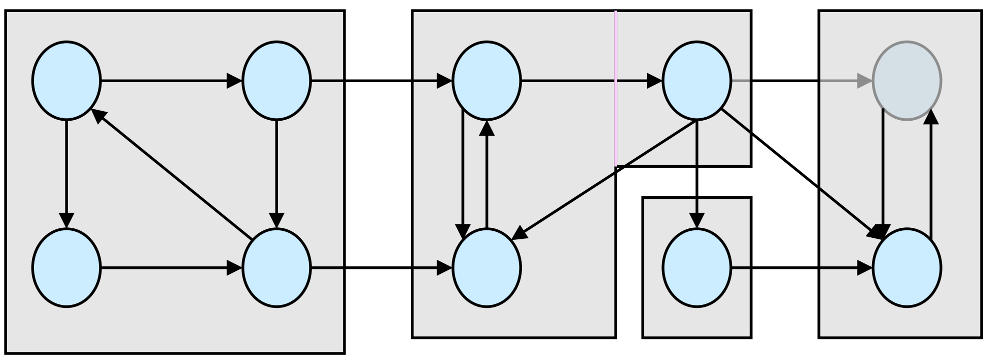


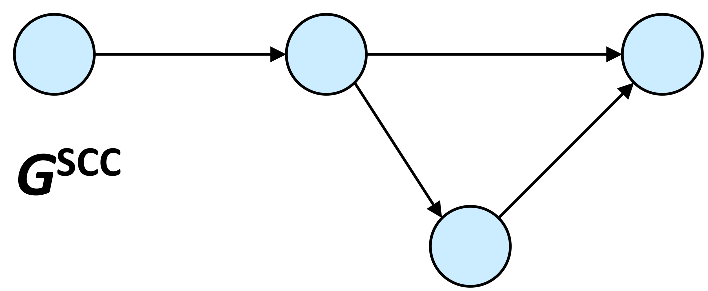
Component Graph

Component graph $G^{SCC} = (V^{SCC}, E^{SCC})$

V^{SCC} → has one vertex for each SCC in G

E^{SCC} → has an edge if there's an edge between the corresponding SCC's in G





Transpose of a Directed Graph

- **Transpose** of directed *G(V,E) is* $\boldsymbol{G^{\mathrm{T}}}$
  - $G^{\mathrm{T}} = (V, E^{\mathrm{T}})$,
    - where $E^{\mathrm{T}} = \{(u, v) : (v, u) \in E\}$.
  - G^{T} is G with all its edges reversed

- Using Adjacency list of G
  - G^{T} created in $\boldsymbol{\Theta(V + E)}$ time
- G and G^{T} have the same *strongly connected components*
  - u and v are reachable from each other in G if and only if reachable from each other in G^{T}

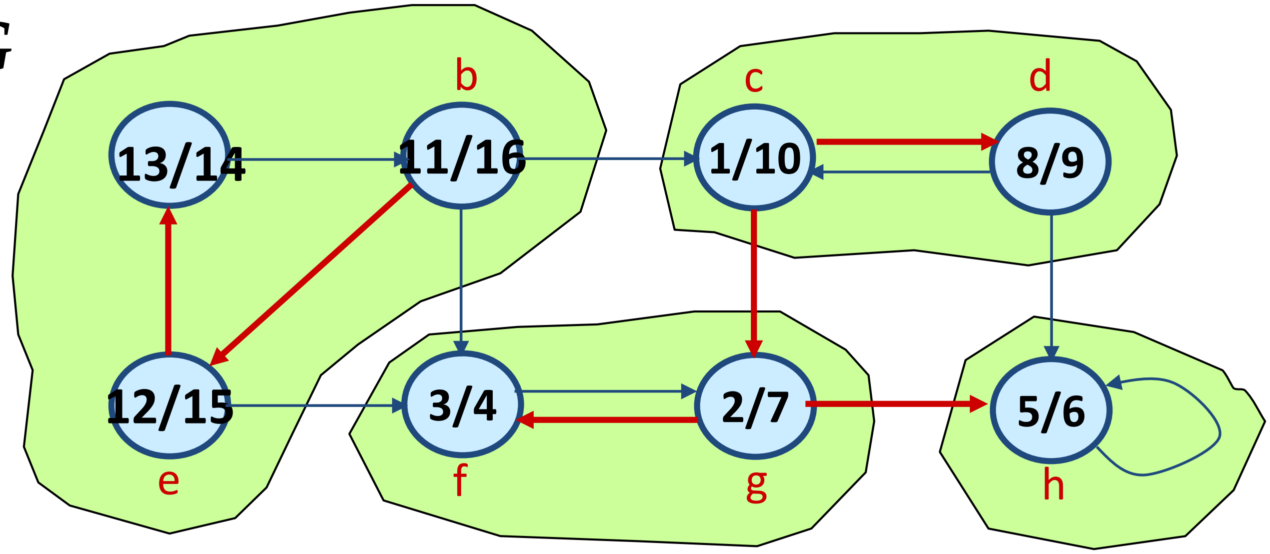
Strongly_Connected_Components(G)

- Call DFS(G) to compute finishing times $f[u]$ for all u
- Compute G^{T}
- Call DFS(G^{T}), but in the main loop, consider vertices in order of decreasing $f[u]$ (as computed in first DFS)
- Output the vertices in each tree of the depth-first forest formed in second DFS as a separate SCC

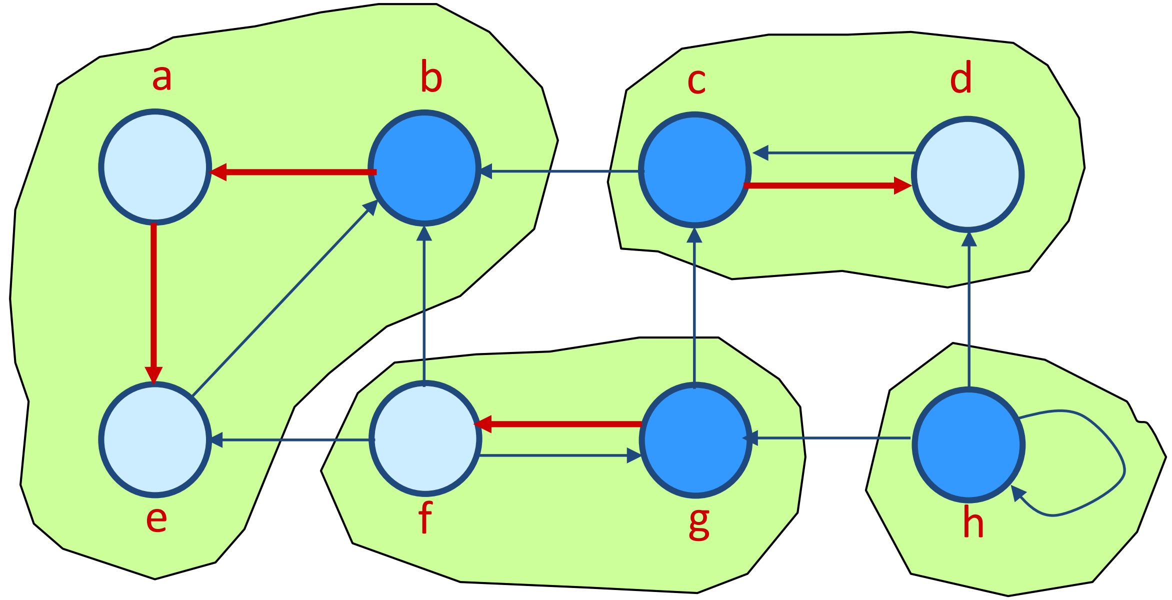
Linear running Time: $\Theta(V + E)$

Example

Graph G



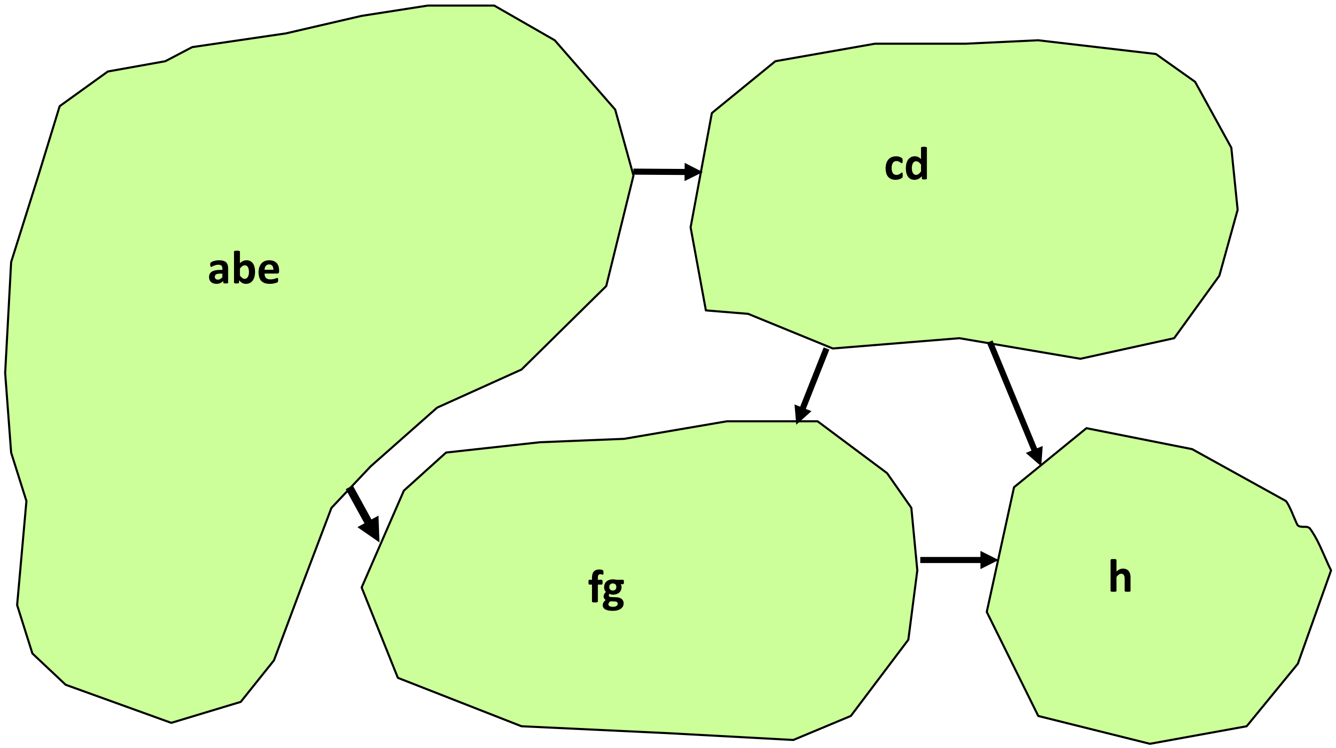
Transpose
G^T



Example

Component Graph

G^{SCC}



Point to Note:

→By considering vertices in second DFS in decreasing order of finishing times from first DFS

→actually visiting vertices of the component graph in topologically sorted order

→Since running DFS on G^T, we will not be visiting any v from a u, where v and u are in different components

Lemma

Let C and C' be distinct SCC's in $G = (V, E)$.

Suppose there is an edge $(u, v) \in E$ such that $u \in C$ and $v \in C'$

Then $f(C) > f(C')$.

Corollary

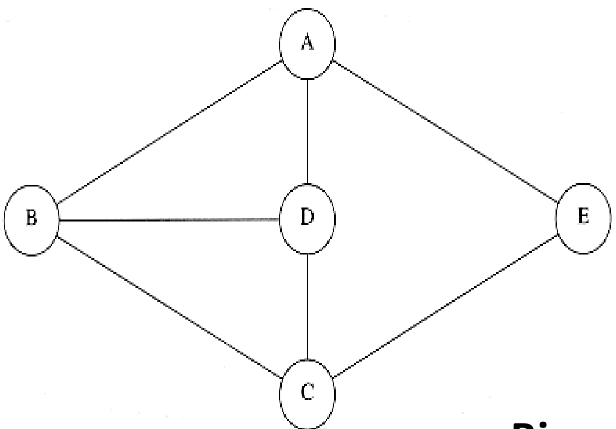
Let C and C' be distinct strongly connected components in $G = (V, E)$.

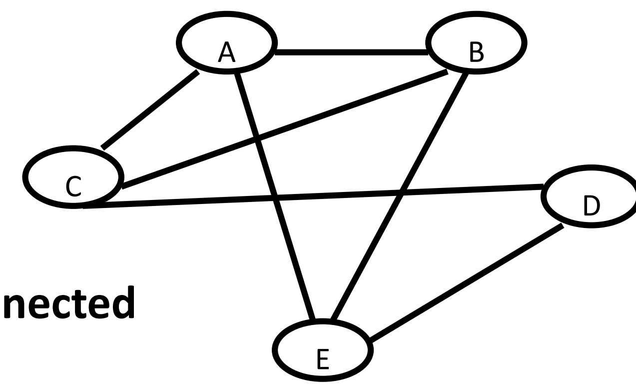
Suppose there is an edge $(u, v) \in E^T$ such that $u \in C$ and $v \in C'$

Then $f(C) < f(C')$.

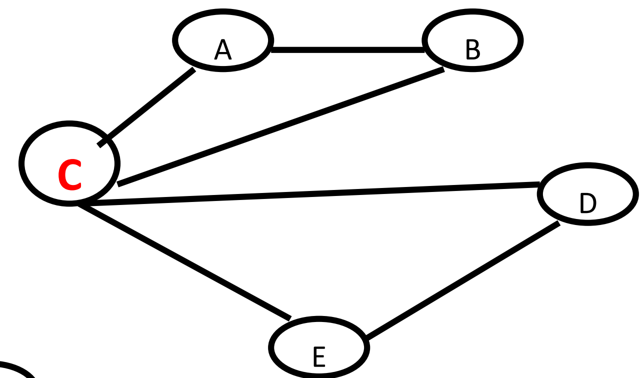
Biconnectivity

- A connected undirected graph is biconnected if there are no vertices whose removal will disconnects the rest of the graph





Biconnected



not Biconnected

Example :

--Computer→ Nodes and Links →Edges

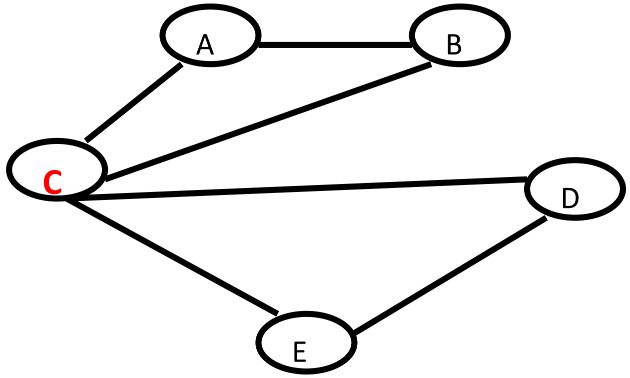
--if any computer goes down, mailing service of network mail unaffected, except at the down computer.

Articulation Points

➢ Articulation point or cut-vertex : A vertex whose removal (along with its attached arcs) makes the graph disconnected

→graph is not biconnected

Eg. **C** is an articulation point



--Relevant to computer networks
--Represent vulnerabilities in a network
--In many applications these nodes are critical

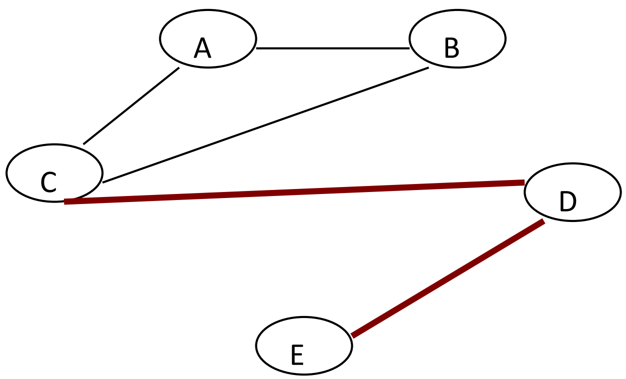
Articulation points can be computed using **depth-first search** and a special numbering of the vertices in the order of their visiting

Bridges

Bridge: An edge in a graph whose removal disconnects the graph

- a bridge is any edge in a graph, that does not lie on a cycle
- a bridge has at least one articulation point at its end
- however an articulation point is not necessarily linked in a bridge

Eg. (C,D) and (E,D)



No articulation points and No bridges→ **Biconnected graph**

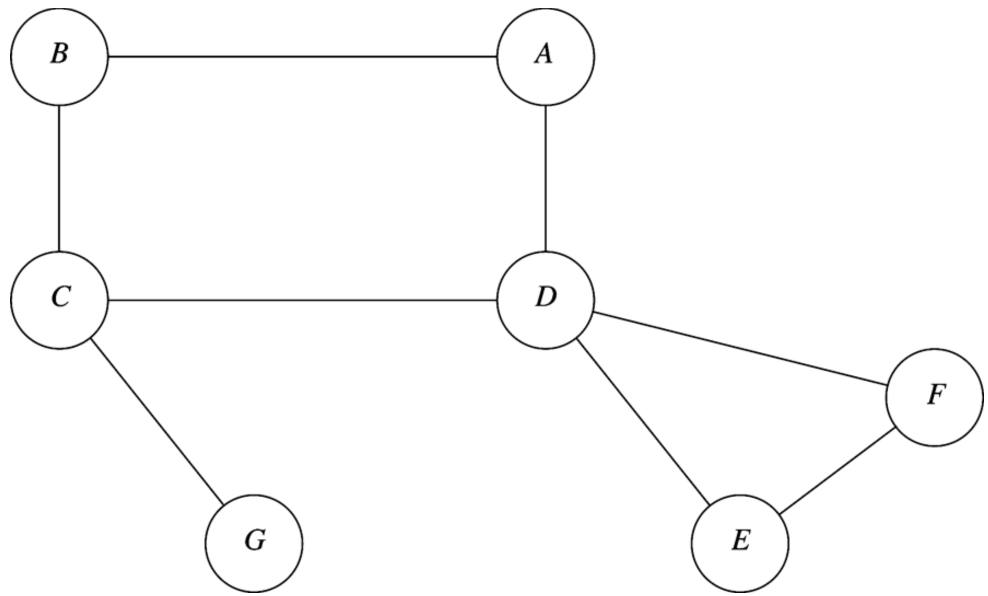
Binconnectivity: Articulation Points

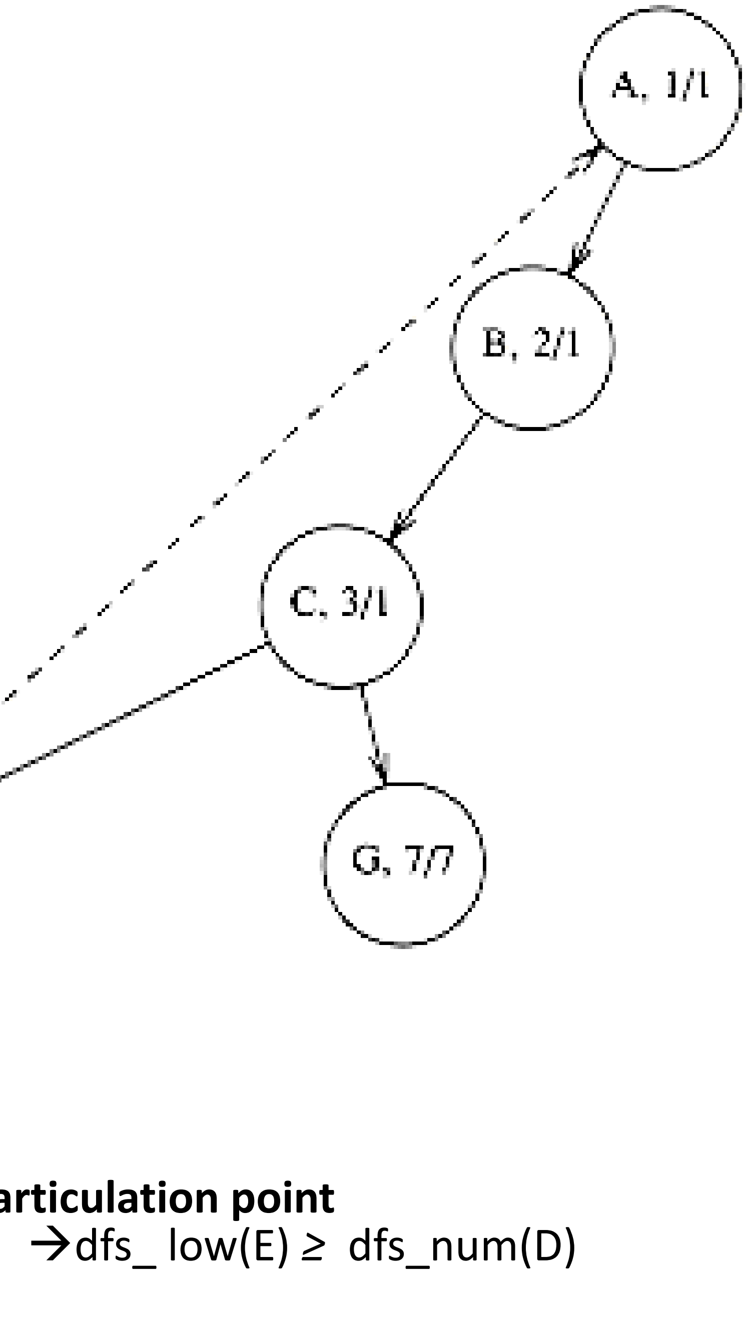
In a connected graph all articulation points can computed using DFS in a linear time:

1. Perform a DFS starting at any vertex
   - number the nodes as they are visited say: ***dfs_num(v)*** as visiting sequence of vertices ***v*** and
2. For every vertex v in the depth-first spanning tree
   - compute ***dfs_low(v) :*** the lowest-numbered vertex ***dfs_num*** that is reachable from ***v*** through a path by taking ***zero or more tree edges*** and then possibly followed by ***zero or one back edge***
3. Vertex ***v*** is an articulation point of G if and only if either
   - ***v*** is the root of DFS tree and has at least two children
   - ***v*** is not the root of DFS tree and has a child ***u*** for which no vertex in subtree rooted with ***u*** has a back edge to one of the ancesstors (in DFS tree) of ***v***

   i.e. vertex v is an articulation point if and only if **v** has some child **u** such that ***dfs_low(u)*** **≥** ***dfs_num(v)***

Binconnectivity: Articulation Points





C is an articulation point

→dsf_ low(G) ≥ dfs_num(C)

D is an articulation point

→dfs_ low(E) ≥ dfs_num(D)