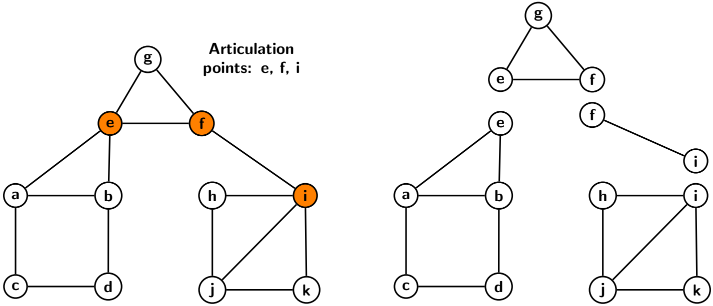
Biconnected Components



- Three articulation points split graph into four biconnected components.

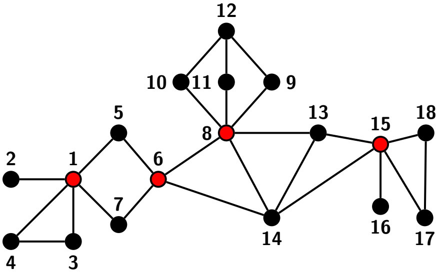
Alternative Definition of Biconnected Graphs

- A pair of edges e_1 and e_2 are related if either $e_1 = e_2$, or there is a cycle containing both edges.
- Above relation defines an equivalence relation and splits edge set into equivalence classes $E_1, E_2, \ldots, E_k$.
- Let V_i be vertex set of E_i.
- Each graph $G_i = (V_i, E_i)$ is called a biconnected component of G.

Block Cut Vertex Tree

- A biconnected component is also known as a block.
- Let B be the set of vertices in which each vertext correspond to a block.
- Let the set of articulation points in G be represented by C.
- Construct a graph of $B \cup C$ vertices as follows:
  - Join $b \in B$ to $c \in C$ if c belongs to block corresponding to b.
- The resulting graph $G = B \cup C, E_{B \cup C}$ is a tree.

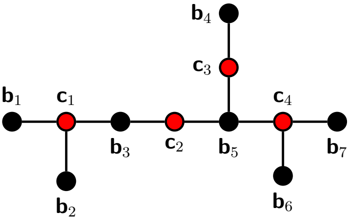
Block Cut Vertex Tree



Blocks	Vertices
b_1	1, 2
b_2	1, 3, 4
b_3	1, 5, 6, 7
b_4	8, 9, 10, 11, 12
b_5	6, 8, 13, 14, 15
b_6	15, 16
b_7	15, 17, 18

Cut vertices (shown in red):
$c_1 = 1$, $c_2 = 6$, $c_2 = 8$, $c_2 = 15$

Block Cut Vertex Tree



- c_1 belongs to b_1, b_2, b_3.
- c_2 belongs to b_3, b_5.
- c_3 belongs to b_4, b_5.
- c_4 belongs to b_5, b_6, b_7.

- Block and cut vertices are adjacent.
- No two block or no two cut vertices are adjacent.

Central Idea of Algorithm

- Central Idea behind the algorithm comes from block and cut vertx tree
- Suppose DFS enters a block B_2 from a block B_1, then it has to be through an edge (c, w), where c is a common cut vertex between blocks B_1 and B_2 and $w \in B_2$.
- Backtracking to c is possible only after completing the exploration of B_2.
- Implies that all the edges of B_2 must be on the top of the edge (c, w) in STACK.

Low Point Function

- After removing edges of B_2, DFS can explore another block in a similar fashion.
- In other words, DFS works on induced graph $G - B_2$ in identical way.
- To identify an articulation point we use a function called LOW point.
  - It is based on idea of the oldest reachable ancestor in a DFST using tree edges and at most one back edge.
- Let T represent DFS tree, and B its set of back edges.

Low Point Function

Definition

LOW point LOW[v] is the smallest of dfn[x], where x is a vertex in G that can be reached by a sequence of zero or more tree edges followed by at most one back edge.

$$\begin{aligned} LOW[v] = & MIN(\{dfn[v]\} \cup \{LOW[x] | (v, x) \in T\} \\ & \cup \{dfn[x] | (v, x) \in B\}) \end{aligned}$$

Finding Articulation Point

- Let us use following notations:
  - A tree path from v to w: $v \xrightarrow{*} w$
  - A back edge from v to w: $v \dashrightarrow w$
- Then LOW[v] = MIN $\left(\{v\} \cup \{w | v \xrightarrow{*} x \text{ and } x \dashrightarrow w\}\right)$,
- So, LOW[v] is minimum of the following three values:
  1. $v \dashrightarrow w$.
  2. DFS number of v.
  3. LOW[c] where c is child of v in DFS tree T.

Articulation Point

Theorem

Let $G = (V, E)$ be a connected graph with DFS tree T and back edges B. Then $a \in V$ is an articulation point if and only if there exists vertices $v, w \in V$ such that $(a, v) \in T$ and w is not a descendant of v in T and LOW[v] $\geq$ dfn[a].

Proof.

- Let v and w exist as stated.
- Since $(a, v) \in T$ and LOW[v] $\geq$ dfn[a], all paths from v not passing through a must remain inside subtree T_v of v.
- As w is not descendant of v, $w \notin T_v$.
- So all paths from v to w must pass through a.

□

Articulation Point

Proof continues.

- Conversely, let a is an articulation point.
- If a is root, at least two tree edges are incident at a.
  - Otherwise between any two vertices in $V - \{a\}$ there is a path avoiding a.
- If a is not the root, it has at least one ancestor w.
- One of the biconnected components containing a has all its nodes as descendants of a in T.
- In fact, all of these nodes are descendant of some child v of a in T.
- v and w can reach each other only through a.

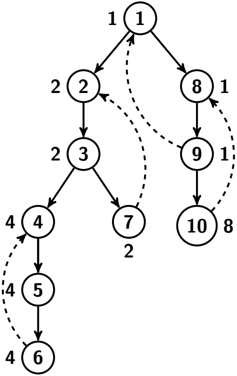
$\square$

Biconnectivity

From the above theorem, if we have values of DFS numbers and LOW points then a is an articulation point iff

1. Either a is the root of a DFST with more than 1 children, or
2. a is not the root and there is at least one child c of a such that no back edge leads from any descendant of c (including c) to a proper ancestor of a.

Computation of LOW Point



- Computation requires a single pass of DFS.
- Initially LOW[8] = dfn[8] = 8, but LOW[9] = 1 due to a back edge (9, 1).
- LOW is updated after call Bicon(9,8) completed:
  LOW[8] = MIN $\{8\} \cup \{$LOW[9]$\}$ = 1.
- LOW[10] = 8, since (10, 8) $\in$ B.
- But as LOW[10] $<$ 9, 9 cannot be an articulation point.

Computation of LOW Point

```
i = 0; // counter for dfn
S = newSTACK();
for (x ∈ V) dfn[v] = 0
for x ∈ V
    if (dfn[x] == 0) Bicon(x,0);

procedure Bicon(v,u) {
    i = i + 1;
    dfn[v] = i;
    LOW[v] = i;
    // main loop
}
```

Computation of LOW Point

```
for (w ∈ ADJ(v)) {
    if (dfn[w] == 0) { // (v,w) is tree edge
        S.push(w,v);
        Bicon(w,v); // Recursive call
        LOW[v] = min{LOW[v], LOW[w]};
        if LOW[w] ≥ dfn[v] {// Articulation point
            delete all edges from S including
            up to and including edge (v,w);
        }
    } else if (dfn[w] < dfn[v] and w ≠ u) {
        S.push(v,w); // (v,w) is back edge
        LOW[v] = min{LOW[v], dfn[w]};
    }
}
```