ITMO UNIVERSITY

How to Win Coding Competitions: Secrets of Champions

Week 5: Algorithms on Graphs 1
Lecture 4: Depth First Search with Timestamps

Maxim Buzdalov
Saint Petersburg 2016

Let's modify DFS to track the time of entering and exiting a vertex

Let's modify DFS to track the time of entering and exiting a vertex

$G = \langle V, E \rangle$
$T_{\text{in}}, T_{\text{out}} \leftarrow \{\infty\}$
$A(v) = \{u \mid (v, u) \in E\}$
$t \leftarrow 0$
procedure $\mathrm{DFS}(v)$
    $t \leftarrow t + 1$
    $T_{\text{in}}(v) \leftarrow t$
    for $u \in A(v)$ **do**
        if $T_{\text{in}}(u) = \infty$ **then** $\mathrm{DFS}(u)$ **end if**
    end for
    $t \leftarrow t + 1$
    $T_{\text{out}}(v) \leftarrow t$
end procedure

Let's modify DFS to track the time of entering and exiting a vertex

$G = \langle V, E \rangle$
$T_{\text{in}}, T_{\text{out}} \leftarrow \{\infty\}$ ▷ $T_{\text{in}}(v)$: the time of entering v
$A(v) = \{u \mid (v, u) \in E\}$ ▷ $T_{\text{out}}(v)$: the time of exiting v
$t \leftarrow 0$
procedure $\mathrm{DFS}(v)$
  $t \leftarrow t + 1$
  $T_{\text{in}}(v) \leftarrow t$
  for $u \in A(v)$ **do**
    if $T_{\text{in}}(u) = \infty$ **then** $\mathrm{DFS}(u)$ **end if**
  end for
  $t \leftarrow t + 1$
  $T_{\text{out}}(v) \leftarrow t$
end procedure

Let's modify DFS to track the time of entering and exiting a vertex

$G = \langle V, E \rangle$
$T_{in}, T_{out} \leftarrow \{\infty\}$ ▷ $T_{in}(v)$: the time of entering v
$A(v) = \{u \mid (v, u) \in E\}$ ▷ $T_{out}(v)$: the time of exiting v
$t \leftarrow 0$
procedure $\mathrm{DFS}(v)$
  $t \leftarrow t + 1$ ▷ Incrementing time
  $T_{in}(v) \leftarrow t$
  for $u \in A(v)$ **do**
    if $T_{in}(u) = \infty$ **then** $\mathrm{DFS}(u)$ **end if**
  end for
  $t \leftarrow t + 1$
  $T_{out}(v) \leftarrow t$
end procedure

Let's modify DFS to track the time of entering and exiting a vertex

```
G = ⟨V, E⟩
T_in, T_out ← {∞}                               ▷ T_in(v): the time of entering v
                                                ▷ T_out(v): the time of exiting v
A(v) = {u | (v, u) ∈ E}
t ← 0
procedure DFS(v)
    t ← t + 1                                   ▷ Incrementing time
    T_in(v) ← t                                 ▷ Marking the time of entering
    for u ∈ A(v) do
        if T_in(u) = ∞ then DFS(u) end if
    end for
    t ← t + 1
    T_out(v) ← t
end procedure
```

Let's modify DFS to track the time of entering and exiting a vertex

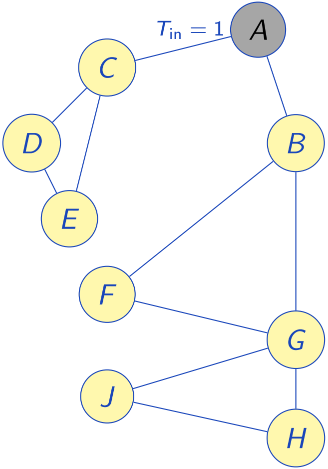
$G = \langle V, E \rangle$
$T_{\text{in}}, T_{\text{out}} \leftarrow \{\infty\}$ ▷ $T_{\text{in}}(v)$: the time of entering v
$A(v) = \{u \mid (v, u) \in E\}$ ▷ $T_{\text{out}}(v)$: the time of exiting v
$t \leftarrow 0$
procedure $\mathrm{DFS}(v)$
    $t \leftarrow t + 1$ ▷ Incrementing time
    $T_{\text{in}}(v) \leftarrow t$ ▷ Marking the time of entering
    for $u \in A(v)$ **do**
        if $T_{\text{in}}(u) = \infty$ **then** $\mathrm{DFS}(u)$ **end if** ▷ Means "not previously entered"
    end for
    $t \leftarrow t + 1$
    $T_{\text{out}}(v) \leftarrow t$
end procedure

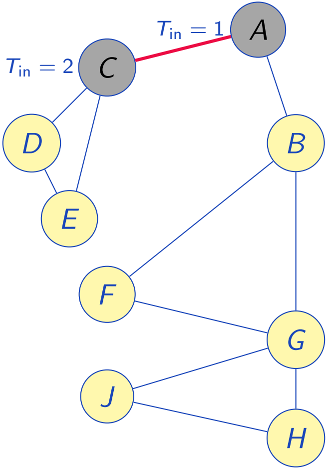
Let's modify DFS to track the time of entering and exiting a vertex

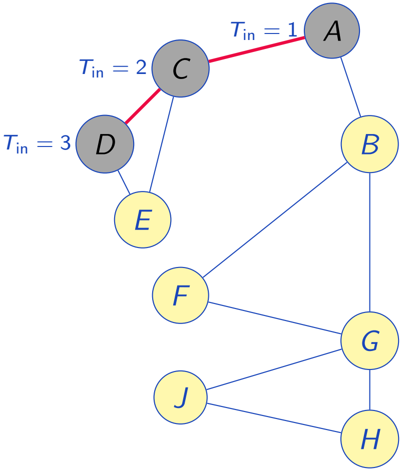
$G = \langle V, E \rangle$
$T_{in}, T_{out} \leftarrow \{\infty\}$ ▷ $T_{in}(v)$: the time of entering v
$A(v) = \{u \mid (v, u) \in E\}$ ▷ $T_{out}(v)$: the time of exiting v
$t \leftarrow 0$
procedure $\mathrm{DFS}(v)$
  $t \leftarrow t + 1$ ▷ Incrementing time
  $T_{in}(v) \leftarrow t$ ▷ Marking the time of entering
  for $u \in A(v)$ **do**
    if $T_{in}(u) = \infty$ **then** $\mathrm{DFS}(u)$ **end if** ▷ Means "not previously entered"
  end for
  $t \leftarrow t + 1$ ▷ Incrementing time
  $T_{out}(v) \leftarrow t$
end procedure

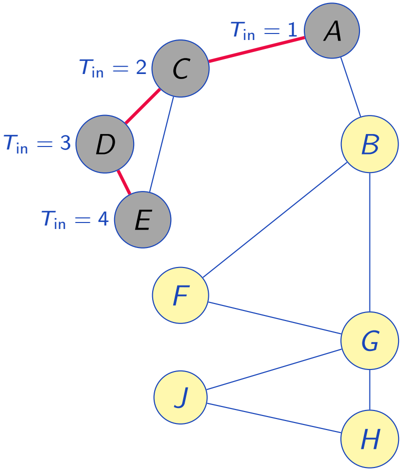
Let's modify DFS to track the time of entering and exiting a vertex

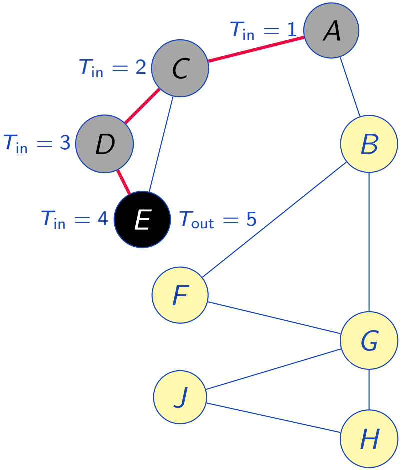
$G = \langle V, E \rangle$
$T_{\text{in}}, T_{\text{out}} \leftarrow \{\infty\}$ ▷ $T_{\text{in}}(v)$: the time of entering v
$A(v) = \{u \mid (v, u) \in E\}$ ▷ $T_{\text{out}}(v)$: the time of exiting v
$t \leftarrow 0$
procedure $\mathrm{DFS}(v)$
  $t \leftarrow t + 1$ ▷ Incrementing time
  $T_{\text{in}}(v) \leftarrow t$ ▷ Marking the time of entering
  for $u \in A(v)$ **do**
    if $T_{\text{in}}(u) = \infty$ **then** $\mathrm{DFS}(u)$ **end if** ▷ Means "not previously entered"
  end for
  $t \leftarrow t + 1$ ▷ Incrementing time
  $T_{\text{out}}(v) \leftarrow t$ ▷ Marking the time of exiting
end procedure

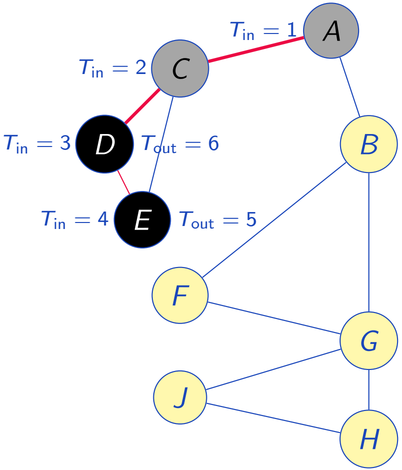

$T_{\text{in}} = 1$
A
C
D
B
E
F
G
J
H

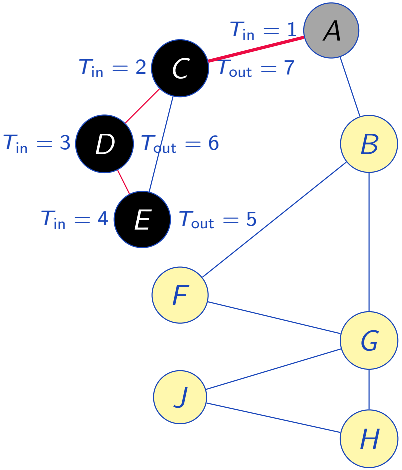

$T_{in} = 1$
A
$T_{in} = 2$
C
D
B
E
F
G
J
H

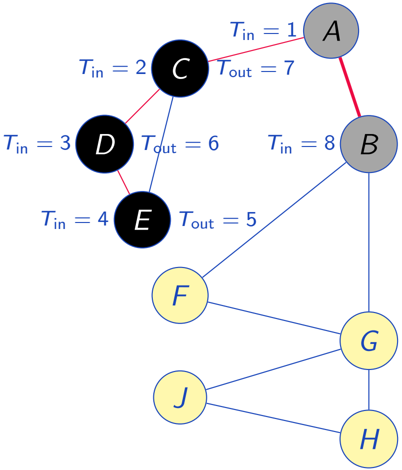

$T_{in} = 1$
A
$T_{in} = 2$
C
$T_{in} = 3$
D
B
E
F
G
J
H

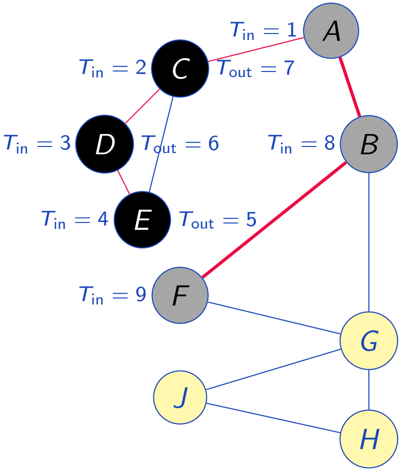

$T_{\text{in}} = 1$
A
$T_{\text{in}} = 2$
C
$T_{\text{in}} = 3$
D
$T_{\text{in}} = 4$
E
B
F
G
J
H

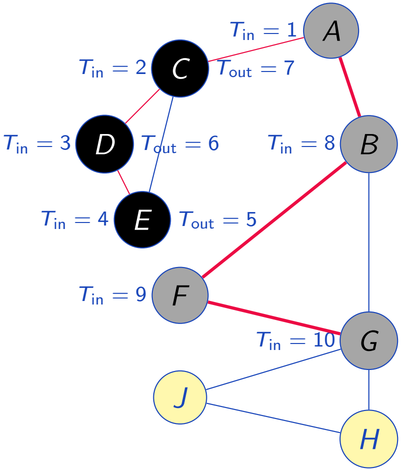

$T_{in} = 1$
A
$T_{in} = 2$
C
$T_{in} = 3$
D
B
$T_{in} = 4$
E
$T_{out} = 5$
F
G
J
H

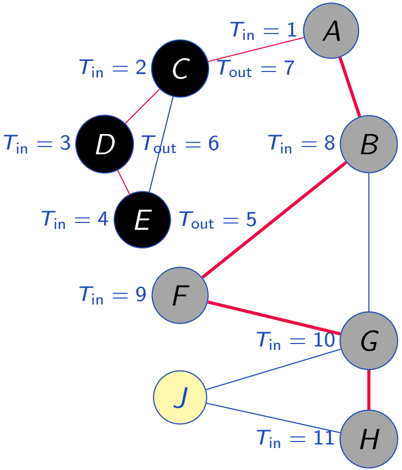

$T_{in} = 1$
A
$T_{in} = 2$
C
$T_{in} = 3$
D
$T_{out} = 6$
B
$T_{in} = 4$
E
$T_{out} = 5$
F
G
J
H

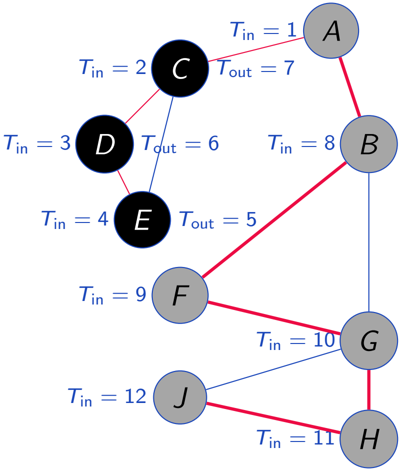

$T_{in} = 1$
A
$T_{in} = 2$
C
$T_{out} = 7$
$T_{in} = 3$
D
$T_{out} = 6$
$T_{in} = 4$
E
$T_{out} = 5$
B
F
G
J
H

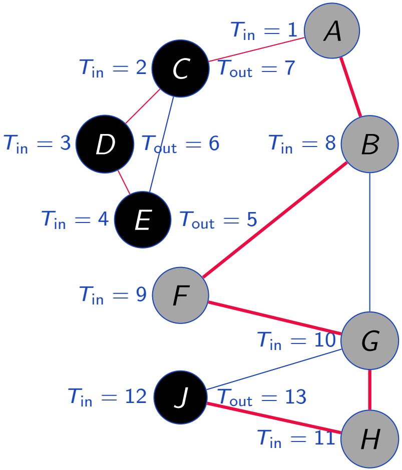

$T_{in} = 1$
A
$T_{in} = 2$
C
$T_{out} = 7$
$T_{in} = 3$
D
$T_{out} = 6$
$T_{in} = 8$
B
$T_{in} = 4$
E
$T_{out} = 5$
F
G
J
H

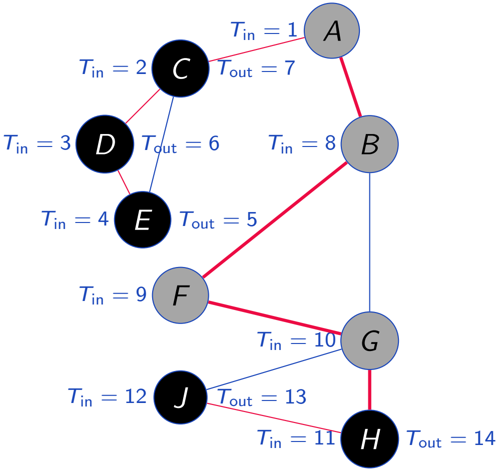

$T_{\text{in}} = 1$
A
$T_{\text{in}} = 2$
C
$T_{\text{out}} = 7$
$T_{\text{in}} = 3$
D
$T_{\text{out}} = 6$
$T_{\text{in}} = 8$
B
$T_{\text{in}} = 4$
E
$T_{\text{out}} = 5$
$T_{\text{in}} = 9$
F
G
J
H

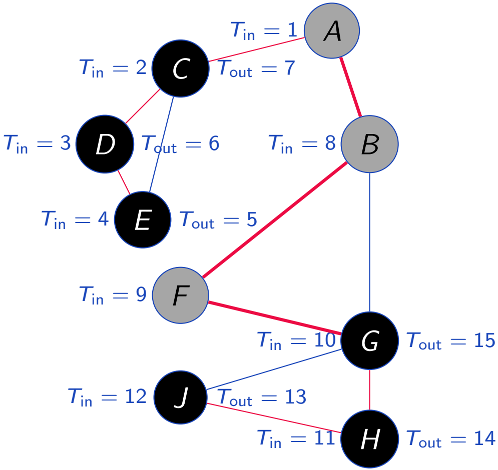

$T_{in} = 1$
A
$T_{in} = 2$
C
$T_{out} = 7$
$T_{in} = 3$
D
$T_{out} = 6$
$T_{in} = 8$
B
$T_{in} = 4$
E
$T_{out} = 5$
$T_{in} = 9$
F
$T_{in} = 10$
G
J
H

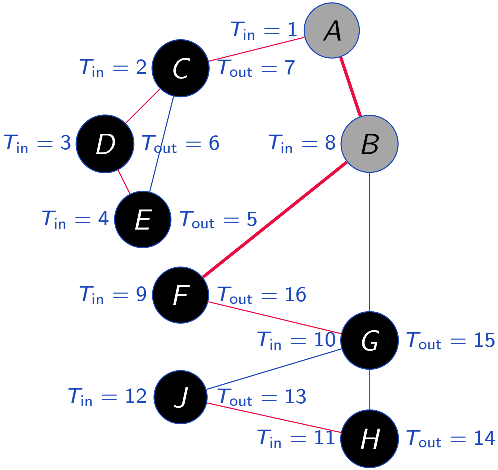

$T_{\text{in}} = 1$
A
$T_{\text{in}} = 2$
C
$T_{\text{out}} = 7$
$T_{\text{in}} = 3$
D
$T_{\text{out}} = 6$
$T_{\text{in}} = 8$
B
$T_{\text{in}} = 4$
E
$T_{\text{out}} = 5$
$T_{\text{in}} = 9$
F
$T_{\text{in}} = 10$
G
J
$T_{\text{in}} = 11$
H

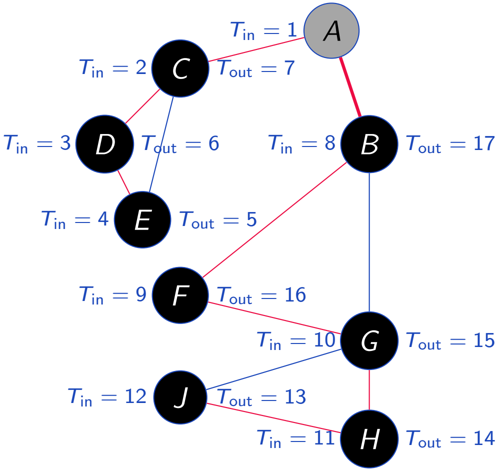

$T_{in} = 1$
A
$T_{in} = 2$
C
$T_{out} = 7$
$T_{in} = 3$
D
$T_{out} = 6$
$T_{in} = 8$
B
$T_{in} = 4$
E
$T_{out} = 5$
$T_{in} = 9$
F
$T_{in} = 10$
G
$T_{in} = 12$
J
$T_{in} = 11$
H

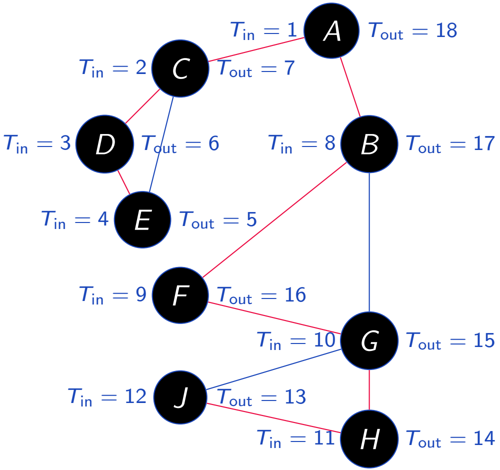

$T_{in} = 1$
A
$T_{in} = 2$
C
$T_{out} = 7$
$T_{in} = 3$
D
$T_{out} = 6$
$T_{in} = 8$
B
$T_{in} = 4$
E
$T_{out} = 5$
$T_{in} = 9$
F
$T_{in} = 10$
G
$T_{in} = 12$
J
$T_{out} = 13$
$T_{in} = 11$
H

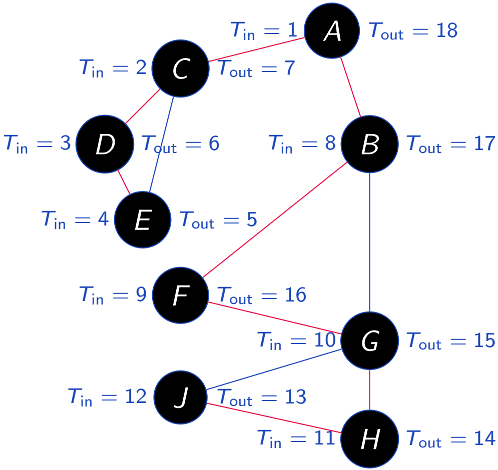

A
$T_{\text{in}} = 1$
C
$T_{\text{in}} = 2$
$T_{\text{out}} = 7$
D
$T_{\text{in}} = 3$
$T_{\text{out}} = 6$
B
$T_{\text{in}} = 8$
E
$T_{\text{in}} = 4$
$T_{\text{out}} = 5$
F
$T_{\text{in}} = 9$
G
$T_{\text{in}} = 10$
J
$T_{\text{in}} = 12$
$T_{\text{out}} = 13$
H
$T_{\text{in}} = 11$
$T_{\text{out}} = 14$


$T_{\text{in}} = 1$
A
$T_{\text{in}} = 2$
C
$T_{\text{out}} = 7$
$T_{\text{in}} = 3$
D
$T_{\text{out}} = 6$
$T_{\text{in}} = 8$
B
$T_{\text{in}} = 4$
E
$T_{\text{out}} = 5$
$T_{\text{in}} = 9$
F
$T_{\text{in}} = 10$
G
$T_{\text{out}} = 15$
$T_{\text{in}} = 12$
J
$T_{\text{out}} = 13$
$T_{\text{in}} = 11$
H
$T_{\text{out}} = 14$

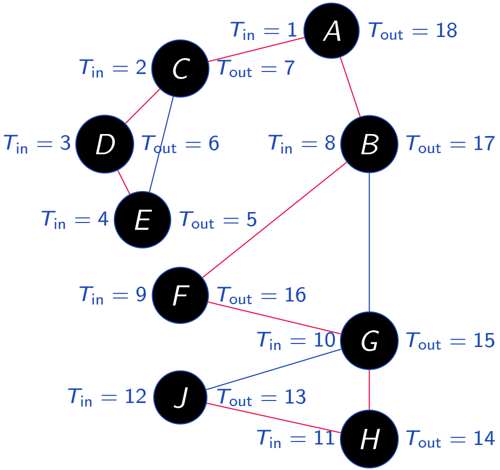

$T_{in} = 1$
A
$T_{in} = 2$
C
$T_{out} = 7$
$T_{in} = 3$
D
$T_{out} = 6$
$T_{in} = 8$
B
$T_{in} = 4$
E
$T_{out} = 5$
$T_{in} = 9$
F
$T_{out} = 16$
$T_{in} = 10$
G
$T_{out} = 15$
$T_{in} = 12$
J
$T_{out} = 13$
$T_{in} = 11$
H
$T_{out} = 14$


A
$T_{in} = 1$
C
$T_{in} = 2$
$T_{out} = 7$
D
$T_{in} = 3$
$T_{out} = 6$
B
$T_{in} = 8$
$T_{out} = 17$
E
$T_{in} = 4$
$T_{out} = 5$
F
$T_{in} = 9$
$T_{out} = 16$
G
$T_{in} = 10$
$T_{out} = 15$
J
$T_{in} = 12$
$T_{out} = 13$
H
$T_{in} = 11$
$T_{out} = 14$

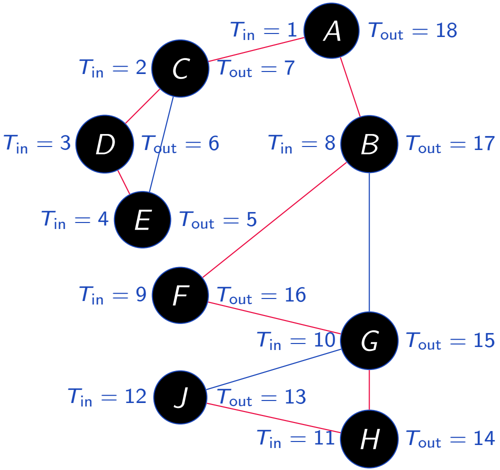

$T_{in} = 1$ A $T_{out} = 18$
$T_{in} = 2$ C $T_{out} = 7$
$T_{in} = 3$ D $T_{out} = 6$
$T_{in} = 8$ B $T_{out} = 17$
$T_{in} = 4$ E $T_{out} = 5$
$T_{in} = 9$ F $T_{out} = 16$
$T_{in} = 10$ G $T_{out} = 15$
$T_{in} = 12$ J $T_{out} = 13$
$T_{in} = 11$ H $T_{out} = 14$



- Important timestamp property:
  A is ancestor of $B \Leftrightarrow$
  $T_{\text{in}}(A) < T_{\text{in}}(B) < T_{\text{out}}(B) < T_{\text{out}}(A)$

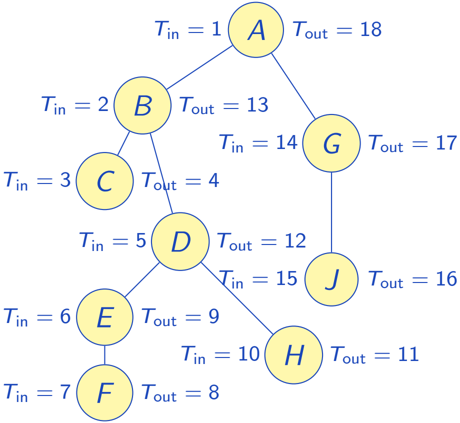


- Important timestamp property: A is ancestor of $B \Leftrightarrow$ $T_{\text{in}}(A) < T_{\text{in}}(B) < T_{\text{out}}(B) < T_{\text{out}}(A)$
- This is a fast way to determine whether a vertex is an ancestor of another one

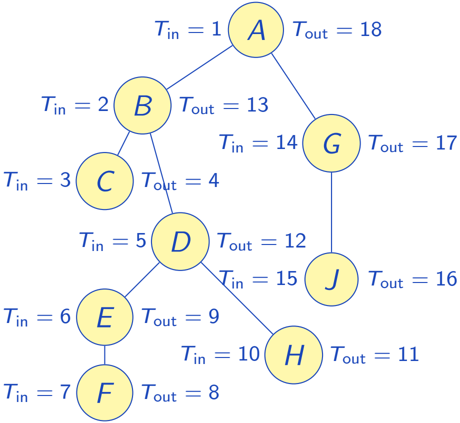


- Important timestamp property:
  A is ancestor of $B \Leftrightarrow$
  $T_{in}(A) < T_{in}(B) < T_{out}(B) < T_{out}(A)$
- This is a fast way to determine whether a vertex is an ancestor of another one
- Some examples follow where this idea is crucial

Example of working with timestamps: finding Least Common Ancestors in trees

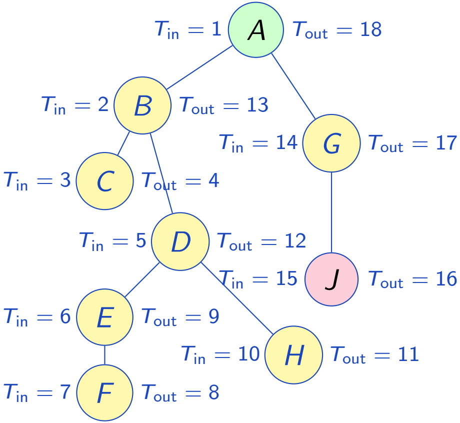


Example of working with timestamps: finding Least Common Ancestors in trees



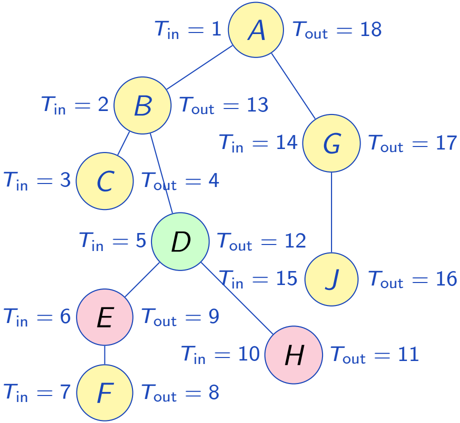
- Examples:

Example of working with timestamps: finding Least Common Ancestors in trees



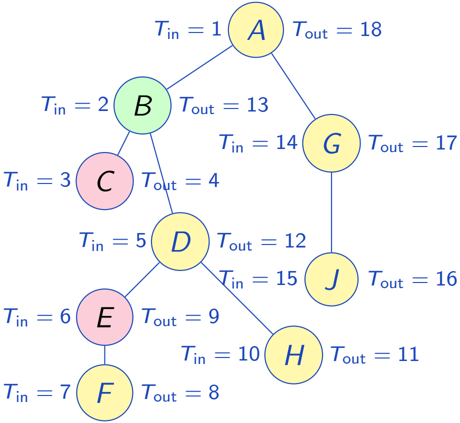
- Examples:
  - $\text{LCA}(A, J) = A$

Example of working with timestamps: finding Least Common Ancestors in trees



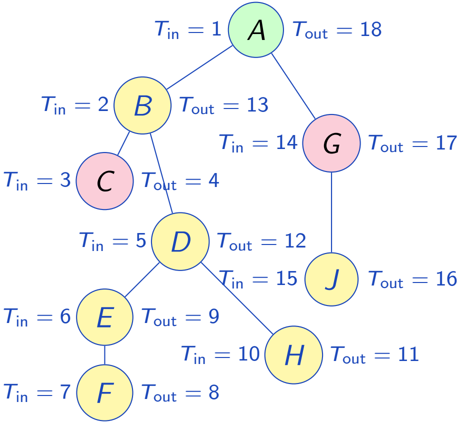
- Examples:
  - $\text{LCA}(A, J) = A$
  - $\text{LCA}(E, H) = D$

Example of working with timestamps: finding Least Common Ancestors in trees



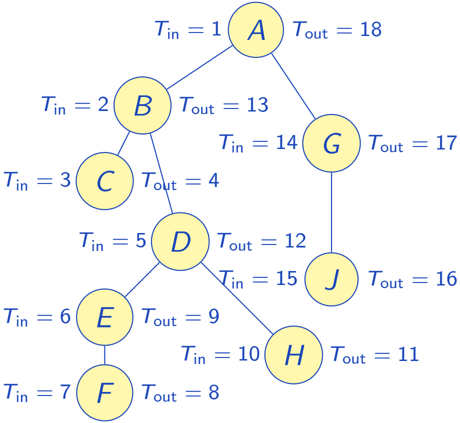
- Examples:
  - $\text{LCA}(A, J) = A$
  - $\text{LCA}(E, H) = D$
  - $\text{LCA}(C, E) = B$

Example of working with timestamps: finding Least Common Ancestors in trees



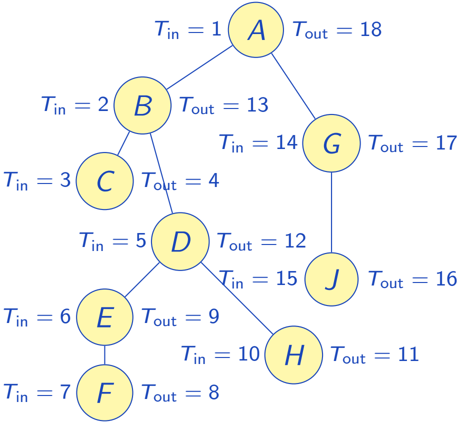
- Examples:
  - $\text{LCA}(A, J) = A$
  - $\text{LCA}(E, H) = D$
  - $\text{LCA}(C, E) = B$
  - $\text{LCA}(C, G) = A$

Example of working with timestamps: finding Least Common Ancestors in trees



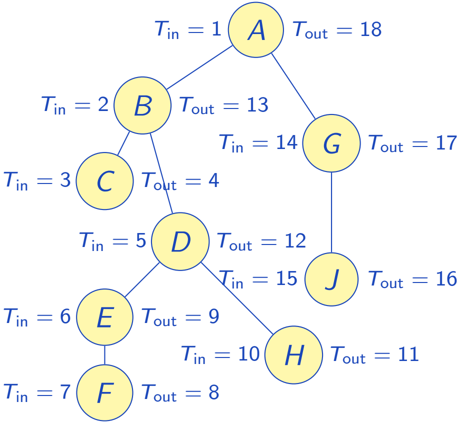
- Examples:
  - $\text{LCA}(A, J) = A$
  - $\text{LCA}(E, H) = D$
  - $\text{LCA}(C, E) = B$
  - $\text{LCA}(C, G) = A$
- Algorithm for answering $\text{LCA}(x, y)$:
  - b: the best ancestor (initially: root)
  - For every vertex z, test if it is an ancestor for both x and y
  - If it is, and b is an ancestor of z, then $b \leftarrow z$

Example of working with timestamps: finding Least Common Ancestors in trees



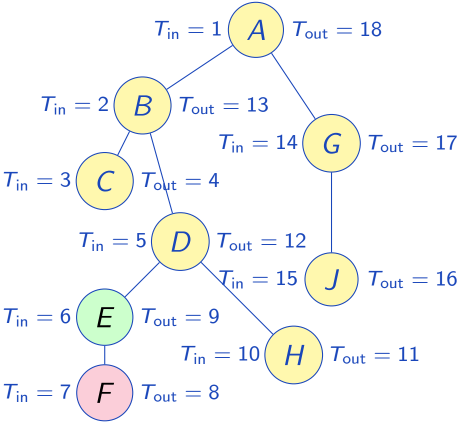
- Examples:
  - $\text{LCA}(A, J) = A$
  - $\text{LCA}(E, H) = D$
  - $\text{LCA}(C, E) = B$
  - $\text{LCA}(C, G) = A$
- Algorithm for answering $\text{LCA}(x, y)$:
  - b: the best ancestor (initially: root)
  - For every vertex z, test if it is an ancestor for both x and y
  - If it is, and b is an ancestor of z, then $b \leftarrow z$
- Runtime: $\Theta(|V|)$. Can we do it faster?

Example of working with timestamps: finding Least Common Ancestors in trees



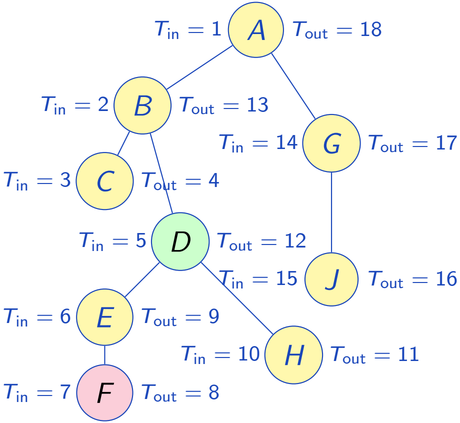
- Path compression ("binary hops"):
  - $d[v][0]$ = parent of v
  - $d[v][i]$ = 2^i-th vertex towards root

Example of working with timestamps: finding Least Common Ancestors in trees



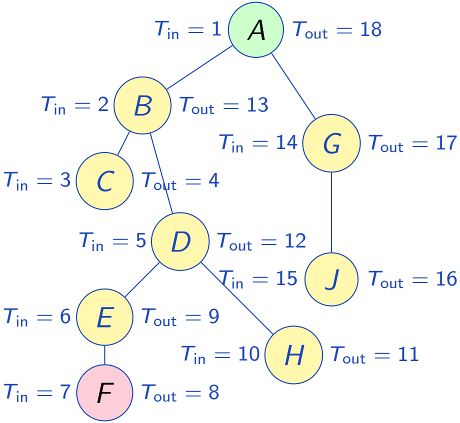
- Path compression ("binary hops"):
  - $d[v][0]$ = parent of v
  - $d[v][i]$ = 2^i-th vertex towards root
  - Example: $d[F][0]$

Example of working with timestamps: finding Least Common Ancestors in trees



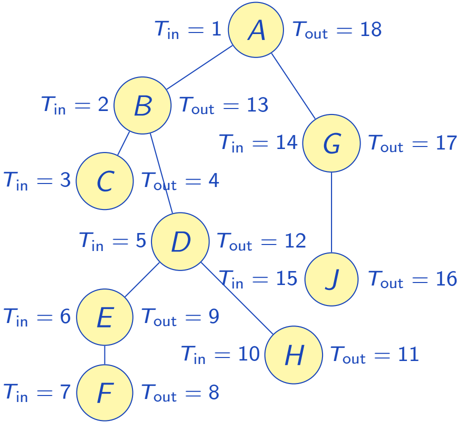
- Path compression ("binary hops"):
  - $d[v][0]$ = parent of v
  - $d[v][i]$ = 2^i-th vertex towards root
  - Example: $d[F][1]$

Example of working with timestamps: finding Least Common Ancestors in trees



- Path compression ("binary hops"):
  - d[v][0] = parent of v
  - d[v][i] = 2^i-th vertex towards root
  - Example: d[F][2]

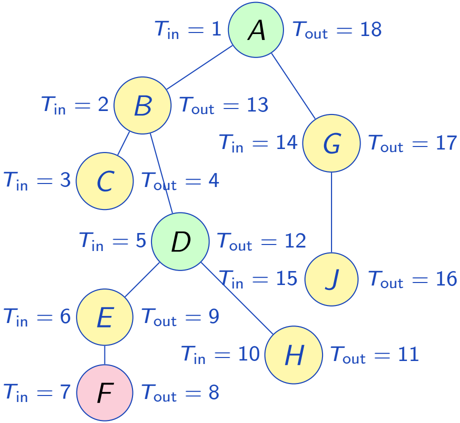
Example of working with timestamps: finding Least Common Ancestors in trees



- Path compression ("binary hops"):
  - $d[v][0]$ = parent of v
  - $d[v][i]$ = 2^i-th vertex towards root

procedure FillHops(V)
  for $v \in V$ **do**
    $d[v][0]$ = parent of v
  end for
  for $i \in [1; \log_2 |V|]$ **do**
    for $v \in V$ **do**
      $d[v][i] = d[d[v][i-1]][i-1]$
    end for
  end for
end procedure

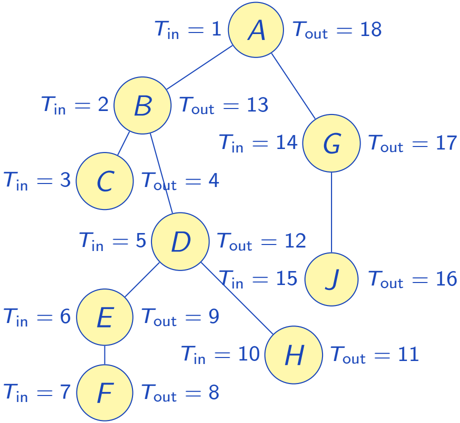
Example of working with timestamps: finding Least Common Ancestors in trees



- Path compression ("binary hops"):
  - d[v][0] = parent of v
  - d[v][i] = 2^i-th vertex towards root

procedure FillHops(V)
  for $v \in V$ **do**
    d[v][0] = parent of v
  end for
  for $i \in [1; \log_2 |V|]$ **do**
    for $v \in V$ **do**
      d[v][i] = d[d[v][$i-1$]][$i-1$]
    end for
  end for
end procedure

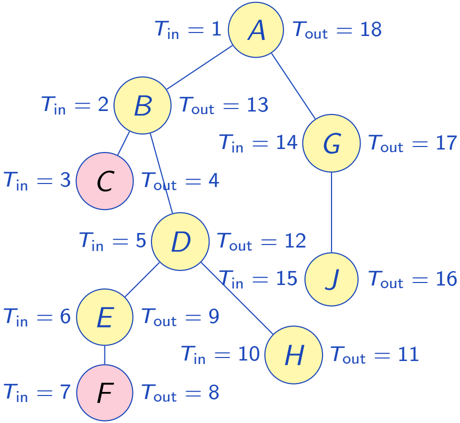
Example of working with timestamps: finding Least Common Ancestors in trees



- Path compression (“binary hops”):
  - $\mathrm{d}[v][0]$ = parent of v
  - $\mathrm{d}[v][i] = 2^i$-th vertex towards root

procedure $\mathrm{LCA}(a, b)$
  if $\textsc{IsAncestor}(a, b)$ **then return** a **end if**
  if $\textsc{IsAncestor}(b, a)$ **then return** b **end if**
  for i **from** $\log_2 |V|$ **down to** 1 **do**
    if not $\textsc{IsAncestor}(\mathrm{d}[a][i], b)$ **then**
      $a \leftarrow \mathrm{d}[a][i]$
    end if
  end for
  return $\mathrm{d}[a][0]$
end procedure

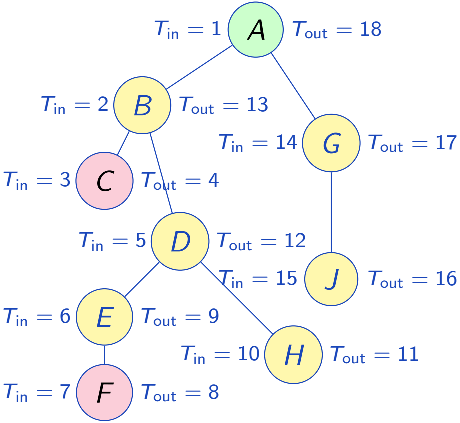
Example of working with timestamps: finding Least Common Ancestors in trees



- Path compression ("binary hops"):
  - $d[v][0]$ = parent of v
  - $d[v][i]$ = 2^i-th vertex towards root

procedure LCA(a, b)
  if IsAncestor(a, b) **then return** a **end if**
  if IsAncestor(b, a) **then return** b **end if**
  for i **from** $\log_2 |V|$ **down to** 1 **do**
    if not IsAncestor($d[a][i]$, b) **then**
      $a \leftarrow d[a][i]$
    end if
  end for
  return $d[a][0]$
end procedure

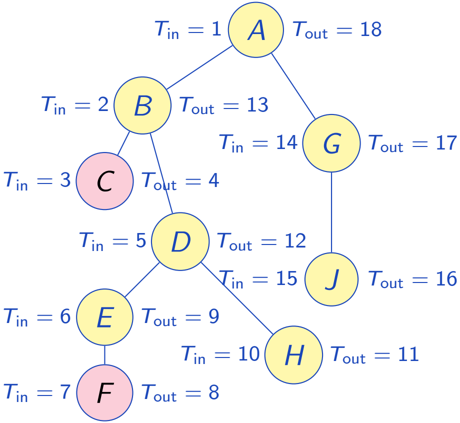
Example of working with timestamps: finding Least Common Ancestors in trees



- Path compression ("binary hops"):
  - $d[v][0]$ = parent of v
  - $d[v][i] = 2^i$-th vertex towards root

procedure LCA(a, b)
  if IsAncestor(a, b) **then return** a **end if**
  if IsAncestor(b, a) **then return** b **end if**
  for i **from** $\log_2 |V|$ **down to** 1 **do**
    if not IsAncestor($d[a][i]$, b) **then**
      $a \leftarrow d[a][i]$
    end if
  end for
  return $d[a][0]$
end procedure

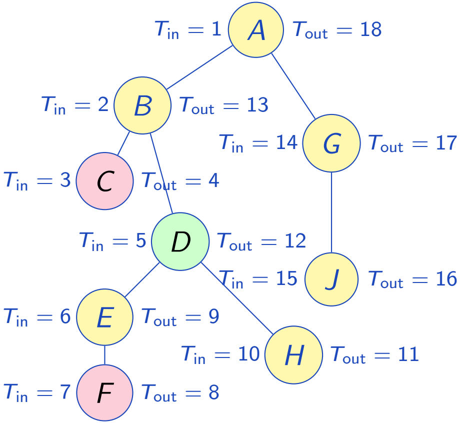
Example of working with timestamps: finding Least Common Ancestors in trees



- Path compression ("binary hops"):
  - $d[v][0]$ = parent of v
  - $d[v][i]$ = 2^i-th vertex towards root

procedure LCA(a, b)
  if IsAncestor(a, b) **then return** a **end if**
  if IsAncestor(b, a) **then return** b **end if**
  for i **from** $\log_2 |V|$ **down to** 1 **do**
    if not IsAncestor($d[a][i]$, b) **then**
      $a \leftarrow d[a][i]$
    end if
  end for
  return $d[a][0]$
end procedure

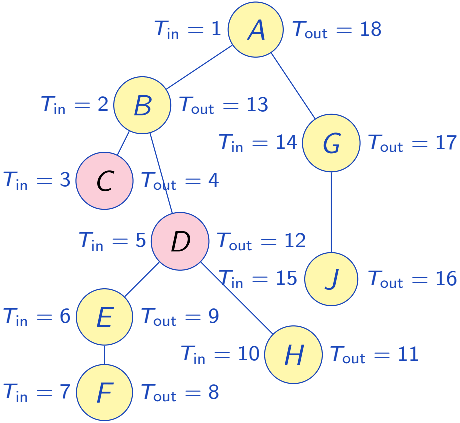
Example of working with timestamps: finding Least Common Ancestors in trees



- Path compression ("binary hops"):
  - $d[v][0]$ = parent of v
  - $d[v][i]$ = 2^i-th vertex towards root

procedure LCA(a, b)
    if IsAncestor(a, b) **then return** a **end if**
    if IsAncestor(b, a) **then return** b **end if**
    for i **from** $\log_2 |V|$ **down to** 1 **do**
        if not IsAncestor($d[a][i]$, b) **then**
            $a \leftarrow d[a][i]$
        end if
    end for
    return $d[a][0]$
end procedure

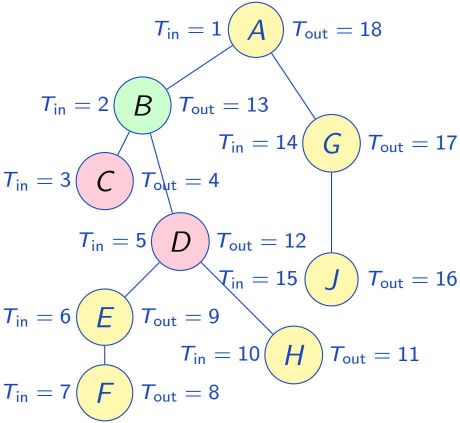
Example of working with timestamps: finding Least Common Ancestors in trees



- Path compression ("binary hops"):
  - d[v][0] = parent of v
  - d[v][i] = 2^i-th vertex towards root

procedure LCA(a, b)
    if IsAncestor(a, b) **then return** a **end if**
    if IsAncestor(b, a) **then return** b **end if**
    for i **from** $\log_2 |V|$ **down to** 1 **do**
        if not IsAncestor(d[a][i], b) **then**
            $a \leftarrow$ d[a][i]
        end if
    end for
    return d[a][0]
end procedure

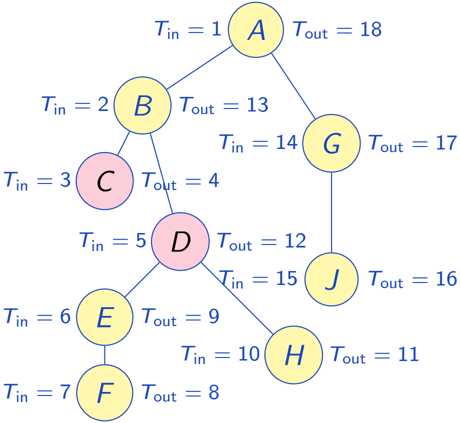
Example of working with timestamps: finding Least Common Ancestors in trees



- Path compression ("binary hops"):
  - $d[v][0]$ = parent of v
  - $d[v][i]$ = 2^i-th vertex towards root

```
procedure LCA(a, b)
    if IsAncestor(a, b) then return a end if
    if IsAncestor(b, a) then return b end if
    for i from log_2 |V| down to 1 do
        if not IsAncestor(d[a][i], b) then
            a ← d[a][i]
        end if
    end for
    return d[a][0]
end procedure
```

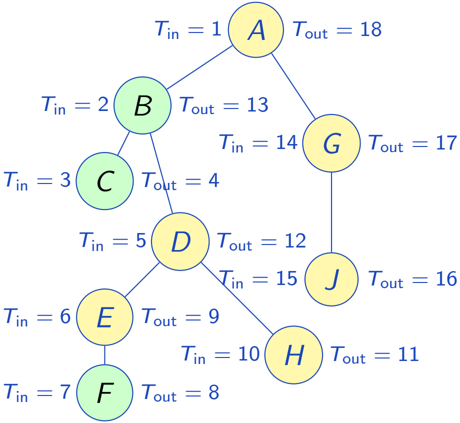
Example of working with timestamps: finding Least Common Ancestors in trees



- Path compression ("binary hops"):
  - $d[v][0]$ = parent of v
  - $d[v][i]$ = 2^i-th vertex towards root

procedure LCA(a, b)
    if IsAncestor(a, b) **then return** a **end if**
    if IsAncestor(b, a) **then return** b **end if**
    for i **from** $\log_2 |V|$ **down to** 1 **do**
        if not IsAncestor($d[a][i]$, b) **then**
            $a \leftarrow d[a][i]$
        end if
    end for
    return $d[a][0]$
end procedure

Example of working with timestamps: finding Least Common Ancestors in trees



- Path compression ("binary hops"):
  - $d[v][0]$ = parent of v
  - $d[v][i]$ = 2^i-th vertex towards root

procedure LCA(a, b)
    if IsAncestor(a, b) **then return** a **end if**
    if IsAncestor(b, a) **then return** b **end if**
    for i **from** $\log_2 |V|$ **down to** 1 **do**
        if not IsAncestor($d[a][i]$, b) **then**
            $a \leftarrow d[a][i]$
        end if
    end for
    return $d[a][0]$
end procedure

An undirected graph is edge-biconnected if the following holds:

- If any edge is removed, the graph will remain connected

An undirected graph is edge-biconnected if the following holds:

- If any edge is removed, the graph will remain connected

A bridge is an edge with the following property:

- If this edge is removed, the graph will no longer be connected

An undirected graph is edge-biconnected if the following holds:

- If any edge is removed, the graph will remain connected

A bridge is an edge with the following property:

- If this edge is removed, the graph will no longer be connected

A graph can be decomposed into edge-biconnected components and bridges.
How to do it faster than in $\Theta(|E|^2)$?

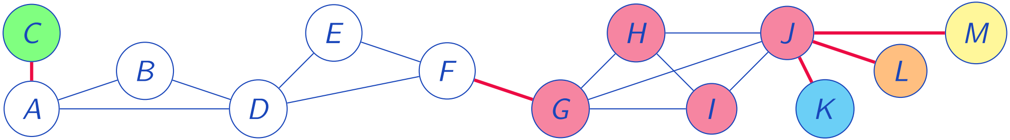
An undirected graph is edge-biconnected if the following holds:

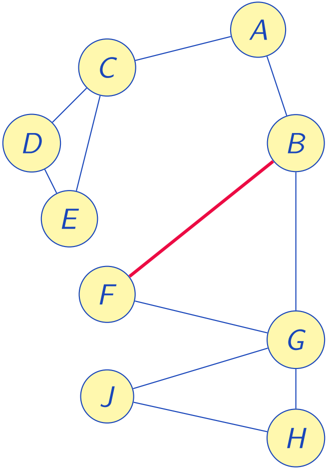
- If any edge is removed, the graph will remain connected

A bridge is an edge with the following property:

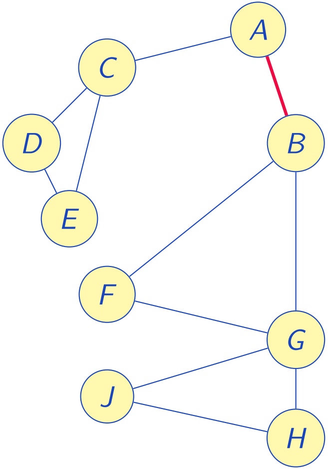
- If this edge is removed, the graph will no longer be connected

A graph can be decomposed into edge-biconnected components and bridges.
How to do it faster than in $\Theta(|E|^2)$?

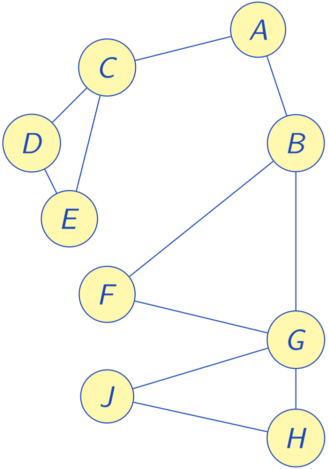




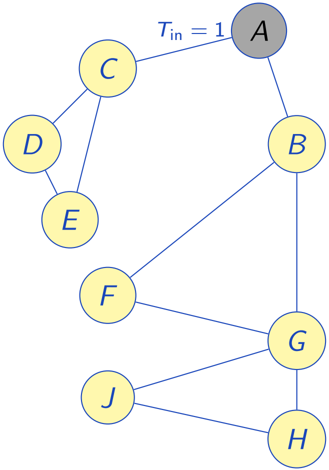
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge



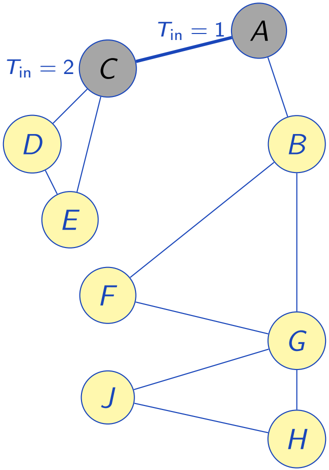
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge



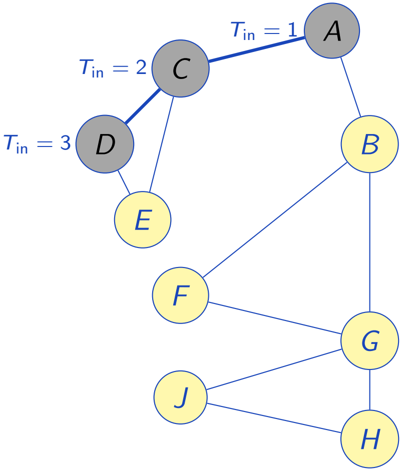
- Consider an edge *BF*
  - *B* is reachable from *F* without this edge: *BF* is not a bridge
- Consider an edge *AB*
  - *A* is not reachable from *B* without this edge: *AB* is a bridge
- An edge *XY* is a bridge, if *X* is not reachable from *Y* without this edge
  - Let's track, for each vertex v, $T_{\min}$: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{\min}(u) > T_{\text{in}}(v)$: (v, u) is a bridge



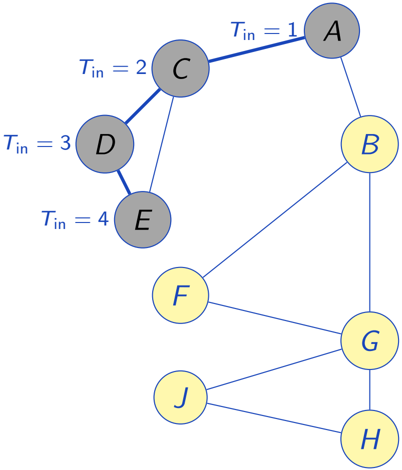
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{min}(u) > T_{in}(v)$: (v, u) is a bridge



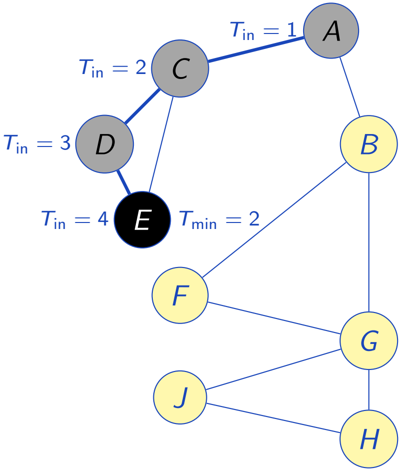
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{min}(u) > T_{in}(v)$: (v, u) is a bridge



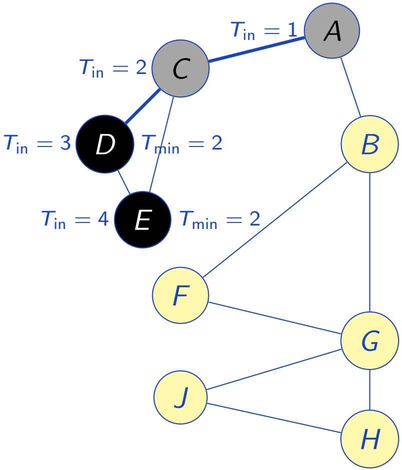
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{\text{min}}(u) > T_{\text{in}}(v)$: (v, u) is a bridge



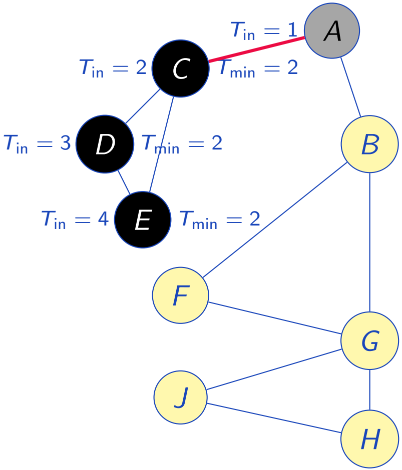
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{min}(u) > T_{in}(v)$: (v, u) is a bridge



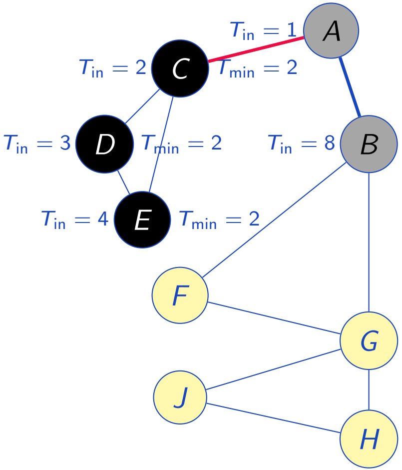
- Consider an edge *BF*
  - *B* is reachable from *F* without this edge: *BF* is not a bridge
- Consider an edge *AB*
  - *A* is not reachable from *B* without this edge: *AB* is a bridge
- An edge *XY* is a bridge, if *X* is not reachable from *Y* without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{\text{min}}(u) > T_{\text{in}}(v)$: (v, u) is a bridge



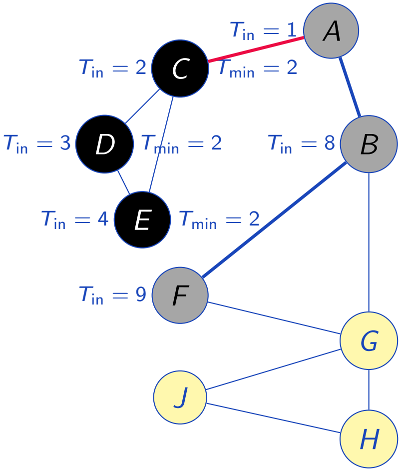
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{\text{min}}(u) > T_{\text{in}}(v)$: (v, u) is a bridge



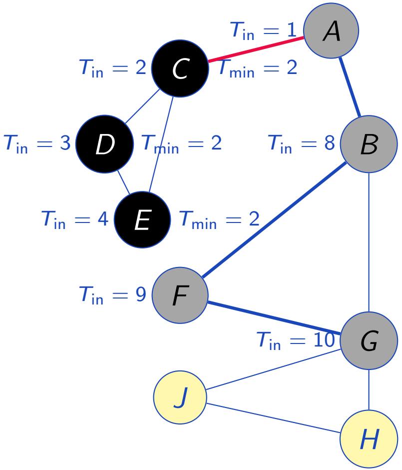
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{min}(u) > T_{in}(v)$: (v, u) is a bridge



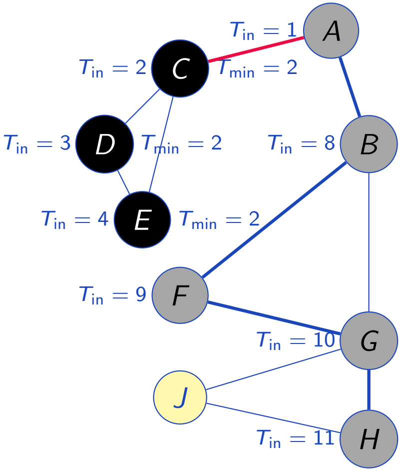
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, $T_{\min}$: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{\min}(u) > T_{\text{in}}(v)$: (v, u) is a bridge



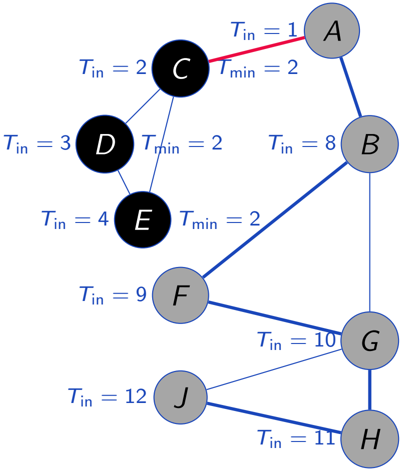
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{min}(u) > T_{in}(v)$: (v, u) is a bridge



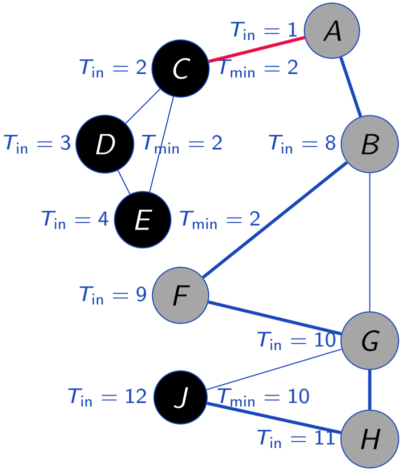
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{min}(u) > T_{in}(v)$: (v, u) is a bridge



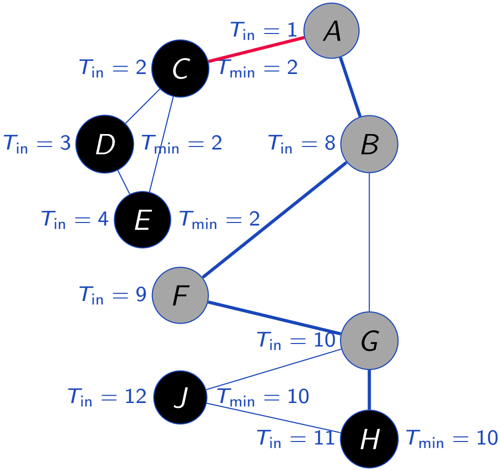
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, $T_{\min}$: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{\min}(u) > T_{\text{in}}(v)$: (v, u) is a bridge



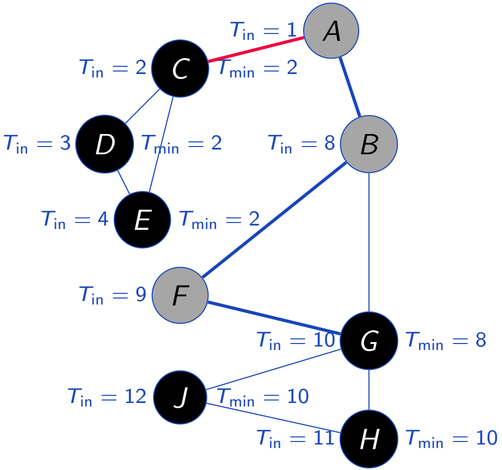
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{\text{min}}(u) > T_{\text{in}}(v)$: (v, u) is a bridge



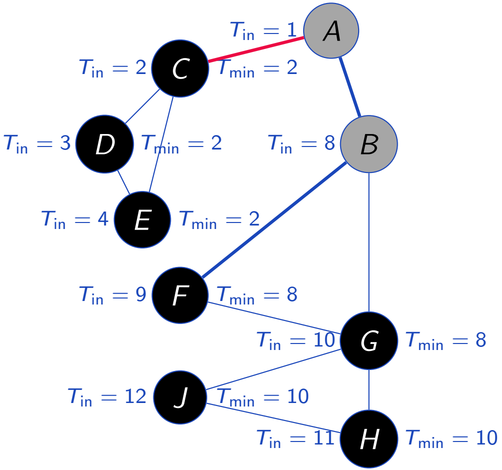
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{\text{min}}(u) > T_{\text{in}}(v)$: (v, u) is a bridge



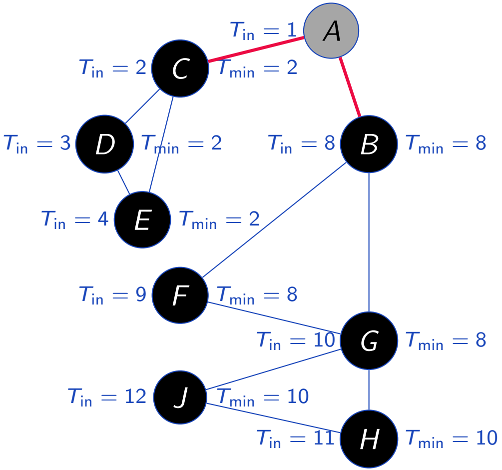
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{min}(u) > T_{in}(v)$: (v, u) is a bridge



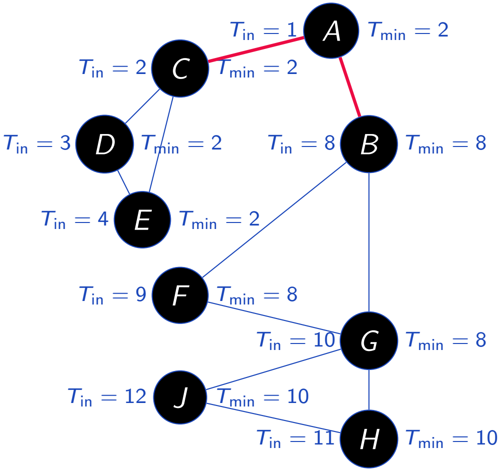
- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, $T_{\min}$: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{\min}(u) > T_{\text{in}}(v)$: (v, u) is a bridge



- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{\text{min}}(u) > T_{\text{in}}(v)$: (v, u) is a bridge



- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, $T_{\min}$: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{\min}(u) > T_{in}(v)$: (v, u) is a bridge



- Consider an edge BF
  - B is reachable from F without this edge: BF is not a bridge
- Consider an edge AB
  - A is not reachable from B without this edge: AB is a bridge
- An edge XY is a bridge, if X is not reachable from Y without this edge
  - Let's track, for each vertex v, T_{min}: the minimum T_{in} of a vertex reachable from v without following uplinks
  - $T_{\text{min}}(u) > T_{\text{in}}(v)$: (v, u) is a bridge

$G = \langle V, E \rangle$
$T_{\text{in}}, T_{\text{min}} \leftarrow \{\infty\}$
$A(v) = \{u \mid (v, u) \in E\}$
$t \leftarrow 0$
procedure $\textsc{Bridges}(v,\ p = -1)$
    $t \leftarrow t + 1;\ T_{\text{in}}(v) \leftarrow t,\ T_{\text{min}}(v) \leftarrow t$
    for $u \in A(v)$ **do**
        if $p = u$ **then continue end if**
        if $T_{\text{in}}(u) = \infty$ **then**
            $\textsc{Bridges}(u,\ v)$
            $T_{\text{min}}(v) \leftarrow \min(T_{\text{min}}(v), T_{\text{min}}(u))$
            if $T_{\text{min}}(u) > T_{\text{in}}(v)$ **then**
                $\textsc{ReportBridge}(v,\ u)$
            end if
        else
            $T_{\text{min}}(v) \leftarrow \min(T_{\text{min}}(v), T_{\text{min}}(u))$
        end if
    end for
end procedure

$G = \langle V, E \rangle$
$T_{\text{in}}, T_{\text{min}} \leftarrow \{\infty\}$
$A(v) = \{u \mid (v, u) \in E\}$
$t \leftarrow 0$
procedure BRIDGES($v,\ p = -1$)
    $t \leftarrow t + 1;\ T_{\text{in}}(v) \leftarrow t,\ T_{\text{min}}(v) \leftarrow t$
    for $u \in A(v)$ **do**
        if $p = u$ **then continue end if**
        if $T_{\text{in}}(u) = \infty$ **then**
            BRIDGES($u,\ v$)
            $T_{\text{min}}(v) \leftarrow \min(T_{\text{min}}(v), T_{\text{min}}(u))$
            if $T_{\text{min}}(u) > T_{\text{in}}(v)$ **then**
                REPORTBRIDGE($v,\ u$)
            end if
        else
            $T_{\text{min}}(v) \leftarrow \min(T_{\text{min}}(v), T_{\text{min}}(u))$
        end if
    end for
end procedure

▷ Tracking T_{min} instead of T_{out}

```
G = ⟨V, E⟩
T_in, T_min ← {∞}                                   ▷ Tracking T_min instead of T_out
A(v) = {u | (v, u) ∈ E}
t ← 0
procedure BRIDGES(v, p = −1)                        ▷ Extra parameter: the parent of v
    t ← t + 1; T_in(v) ← t, T_min(v) ← t
    for u ∈ A(v) do
        if p = u then continue end if
        if T_in(u) = ∞ then
            BRIDGES(u, v)
            T_min(v) ← min(T_min(v), T_min(u))
            if T_min(u) > T_in(v) then
                REPORTBRIDGE(v, u)
            end if
        else
            T_min(v) ← min(T_min(v), T_min(u))
        end if
    end for
end procedure
```

```
G = ⟨V, E⟩
T_in, T_min ← {∞}                                    ▷ Tracking T_min instead of T_out
A(v) = {u | (v, u) ∈ E}
t ← 0
procedure Bridges(v, p = −1)                          ▷ Extra parameter: the parent of v
    t ← t + 1; T_in(v) ← t, T_min(v) ← t
    for u ∈ A(v) do
        if p = u then continue end if
        if T_in(u) = ∞ then
            Bridges(u, v)
            T_min(v) ← min(T_min(v), T_min(u))        ▷ Updating T_min by T_min of a descendant
            if T_min(u) > T_in(v) then
                ReportBridge(v, u)
            end if
        else
            T_min(v) ← min(T_min(v), T_min(u))
        end if
    end for
end procedure
```

```
G = ⟨V, E⟩
T_in, T_min ← {∞}                                  ▷ Tracking T_min instead of T_out
A(v) = {u | (v, u) ∈ E}
t ← 0
procedure Bridges(v, p = −1)                       ▷ Extra parameter: the parent of v
    t ← t + 1; T_in(v) ← t, T_min(v) ← t
    for u ∈ A(v) do
        if p = u then continue end if
        if T_in(u) = ∞ then
            Bridges(u, v)
            T_min(v) ← min(T_min(v), T_min(u))     ▷ Updating T_min by T_min of a descendant
            if T_min(u) > T_in(v) then
                ReportBridge(v, u)
            end if
        else
            T_min(v) ← min(T_min(v), T_min(u))     ▷ Updating T_min by T_min of other vertex
        end if
    end for
end procedure
```

An undirected graph is vertex-biconnected if the following holds:

- If any vertex is removed, the graph will remain connected

An undirected graph is vertex-biconnected if the following holds:

- If any vertex is removed, the graph will remain connected

An articulation point is a vertex with the following property:

- If this vertex is removed, the graph will no longer be connected

An undirected graph is vertex-biconnected if the following holds:

- If any vertex is removed, the graph will remain connected

An articulation point is a vertex with the following property:

- If this vertex is removed, the graph will no longer be connected

A graph can be decomposed into vertex-biconnected components, connected by articulation points.

How to do it faster than in $\Theta(|V| \cdot |E|)$?

An undirected graph is vertex-biconnected if the following holds:

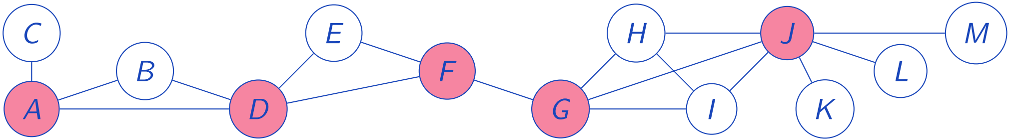
- If any vertex is removed, the graph will remain connected

An articulation point is a vertex with the following property:

- If this vertex is removed, the graph will no longer be connected

A graph can be decomposed into vertex-biconnected components, connected by articulation points.

How to do it faster than in $\Theta(|V| \cdot |E|)$?



```
G = ⟨V, E⟩
T_in, T_min ← {∞}
A(v) = {u | (v, u) ∈ E}
t ← 0
procedure ARTICULATION(v, p = −1)
    t ← t + 1; T_in(v) ← t; T_min(v) ← t; ch ← 0
    for u ∈ A(v) do
        if p = u then continue end if
        if T_in(u) = ∞ then
            ch ← ch + 1
            ARTICULATION(u, v)
            T_min(v) ← min(T_min(v), T_min(u))
            if T_min(u) ≥ T_in(v) and p ≠ −1 then
                REPORTARTICULATION(v)
            end if
        else
            T_min(v) ← min(T_min(v), T_min(u))
        end if
    end for
    if p = −1 and ch > 1 then REPORTARTICULATION(v) end if
end procedure
```

$G = \langle V, E \rangle$
$T_{in}, T_{min} \leftarrow \{\infty\}$
$A(v) = \{u \mid (v, u) \in E\}$
$t \leftarrow 0$

```
procedure ARTICULATION(v, p = −1)
    t ← t + 1; T_in(v) ← t; T_min(v) ← t; ch ← 0          ▷ Now we also track children count
    for u ∈ A(v) do
        if p = u then continue end if
        if T_in(u) = ∞ then
            ch ← ch + 1
            ARTICULATION(u, v)
            T_min(v) ← min(T_min(v), T_min(u))
            if T_min(u) ≥ T_in(v) and p ≠ −1 then
                REPORTARTICULATION(v)
            end if
        else
            T_min(v) ← min(T_min(v), T_min(u))
        end if
    end for
    if p = −1 and ch > 1 then REPORTARTICULATION(v) end if
end procedure
```

$G = \langle V, E \rangle$
$T_{\text{in}}, T_{\text{min}} \leftarrow \{\infty\}$
$A(v) = \{u \mid (v, u) \in E\}$
$t \leftarrow 0$
procedure $\textsc{Articulation}(v,\ p = -1)$
    $t \leftarrow t + 1$; $T_{\text{in}}(v) \leftarrow t$; $T_{\text{min}}(v) \leftarrow t$; $ch \leftarrow 0$ ▷ Now we also track children count
    for $u \in A(v)$ **do**
        if $p = u$ **then continue end if**
        if $T_{\text{in}}(u) = \infty$ **then**
            $ch \leftarrow ch + 1$ ▷ ...and incrementing it on every child
            $\textsc{Articulation}(u,\ v)$
            $T_{\text{min}}(v) \leftarrow \min(T_{\text{min}}(v), T_{\text{min}}(u))$
            if $T_{\text{min}}(u) \geq T_{\text{in}}(v)$ **and** $p \neq -1$ **then**
                $\textsc{ReportArticulation}(v)$
            end if
        else
            $T_{\text{min}}(v) \leftarrow \min(T_{\text{min}}(v), T_{\text{min}}(u))$
        end if
    end for
    if $p = -1$ **and** $ch > 1$ **then** $\textsc{ReportArticulation}(v)$ **end if**
end procedure

$G = \langle V, E \rangle$
$T_{\text{in}}, T_{\text{min}} \leftarrow \{\infty\}$
$A(v) = \{u \mid (v, u) \in E\}$
$t \leftarrow 0$
procedure ARTICULATION(v, $p = -1$)
    $t \leftarrow t + 1$; $T_{\text{in}}(v) \leftarrow t$; $T_{\text{min}}(v) \leftarrow t$; $ch \leftarrow 0$ ▷ Now we also track children count
    for $u \in A(v)$ **do**
        if $p = u$ **then continue end if**
        if $T_{\text{in}}(u) = \infty$ **then**
            $ch \leftarrow ch + 1$ ▷ ...and incrementing it on every child
            ARTICULATION(u, v)
            $T_{\text{min}}(v) \leftarrow \min(T_{\text{min}}(v), T_{\text{min}}(u))$
            if $T_{\text{min}}(u) \geq T_{\text{in}}(v)$ **and** $p \neq -1$ **then** ▷ Now inequality is non-strict, and root is not considered
                REPORTARTICULATION(v)
            end if
        else
            $T_{\text{min}}(v) \leftarrow \min(T_{\text{min}}(v), T_{\text{min}}(u))$
        end if
    end for
    if $p = -1$ **and** $ch > 1$ **then** REPORTARTICULATION(v) **end if**
end procedure

$G = \langle V, E \rangle$
$T_{\text{in}}, T_{\text{min}} \leftarrow \{\infty\}$
$A(v) = \{u \mid (v, u) \in E\}$
$t \leftarrow 0$
procedure $\textsc{Articulation}(v,\ p = -1)$
    $t \leftarrow t + 1;\ T_{\text{in}}(v) \leftarrow t;\ T_{\text{min}}(v) \leftarrow t;\ ch \leftarrow 0$ ▷ Now we also track children count
    for $u \in A(v)$ **do**
        if $p = u$ **then continue end if**
        if $T_{\text{in}}(u) = \infty$ **then**
            $ch \leftarrow ch + 1$ ▷ . . . and incrementing it on every child
            $\textsc{Articulation}(u,\ v)$
            $T_{\text{min}}(v) \leftarrow \min(T_{\text{min}}(v), T_{\text{min}}(u))$
            if $T_{\text{min}}(u) \geq T_{\text{in}}(v)$ **and** $p \neq -1$ **then** ▷ Now inequality is non-strict, and root is not considered
                $\textsc{ReportArticulation}(v)$
            end if
        else
            $T_{\text{min}}(v) \leftarrow \min(T_{\text{min}}(v), T_{\text{min}}(u))$
        end if
    end for
    if $p = -1$ **and** $ch > 1$ **then** $\textsc{ReportArticulation}(v)$ **end if** ▷ A root is AP iff $ch > 1$
end procedure