CSI - 3105 Design & Analysis of Algorithms
Course 6

Jean-Lou De Carufel

Fall 2019

Connected Components of $G = (V, E)$

The goal is to number the connected components as $1, 2, 3, \ldots$ such that for each vertex v,

$$ccnumber(v) = \# \text{ of the connected component that } v \text{ belongs to}$$

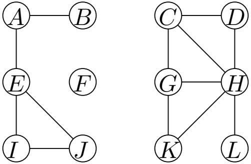
Connected Components of $G = (V, E)$

Algorithm $DFS(G)$

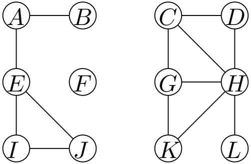
```
for all v ∈ V do
    visited(v) = false
end for
cc = 0
for all v ∈ V do
    if visited(v) = false then
        cc = cc + 1
        explore(v)
    end if
end for
```

In $exlore(v)$,

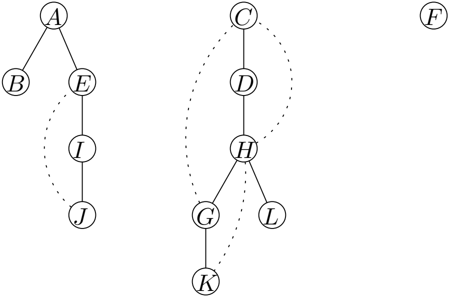
- $previsit(v) \equiv$ "$ccnumber(v) = cc$"
- $postvisit(v) \equiv$ " nil "



As usual, assume that the adjacency lists are sorted in alphabetical order.



As usual, assume that the adjacency lists are sorted in alphabetical order. We get the following DFS-forest.



$ccnumber = 1$	$ccnumber = 2$	$ccnumber = 3$
DFS-tree 1	DFS-tree 2	DFS-tree 3

Running Time of Depth-First-Search (DFS)

First for-loop : $O(|V|)$ time

Second for-loop :

Running Time of Depth-First-Search (DFS)

First for-loop : $O(|V|)$ time

Second for-loop :

- $\rightarrow$ *explore*(u) is called exactly once for each vertex u (this may be part of a recursive call)
- $\rightarrow$ time spent for *explore*(u), excluding recursive calls, is $O(1 + \mathit{degree}(u))$

Running Time of Depth-First-Search (DFS)

First for-loop : $O(|V|)$ time

Second for-loop :

- $\rightarrow$ *explore*(u) is called exactly once for each vertex u (this may be part of a recursive call)
- $\rightarrow$ time spent for *explore*(u), excluding recursive calls, is $O(1 + degree(u))$

Total time:

$$O\left(|V| + \sum_{u \in V}(1 + degree(u))\right)$$

Running Time of Depth-First-Search (DFS)

First for-loop : $O(|V|)$ time

Second for-loop :

$\rightarrow$ *explore*(u) is called exactly once for each vertex u (this may be part of a recursive call)

$\rightarrow$ time spent for *explore*(u), excluding recursive calls, is $O(1 + \mathit{degree}(u))$

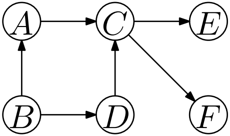
Total time:

$$O\left(|V| + \sum_{u \in V}(1 + \mathit{degree}(u))\right) = O\left(|V| + |V| + 2|E|\right) = O(|V| + |E|)$$

Directed Graphs

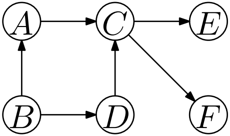
Directed Graphs

Assume that $G = (V, E)$ is directed **and acyclic**.



Directed Graphs

Assume that $G = (V, E)$ is directed **and acyclic**.

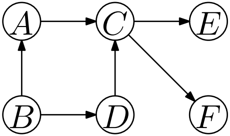


Topological Sorting (or *topological ordering*): number the vertices $1, 2, ..., n$ such that for each edge (u, v),

$$\#(u) < \#(v).$$

Directed Graphs

Assume that $G = (V, E)$ is directed **and acyclic**.



Topological Sorting (or *topological ordering*): number the vertices $1, 2, \ldots, n$ such that for each edge (u, v),

$$\#(u) < \#(v).$$

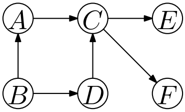
If G is cyclic, this is not possible. Do you see why?
How to compute such a numbering.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```

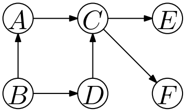


Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\ \}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while



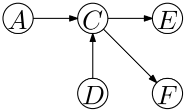
B gets number 1.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
  Choose a vertex $u \in V$ with indegree 0.
  Give u the number k.
  $k = k + 1$
  Remove u from G.
end while



B gets number 1.

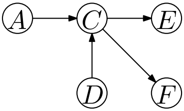
Remove B from G.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```



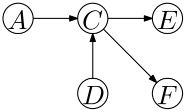
We can pick A or D.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{ \}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while



We can pick A or D.

Let us choose A.

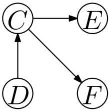
A gets number 2.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while



We can pick A or D.

Let us choose A.

A gets number 2.

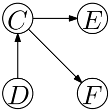
Remove A from G.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while



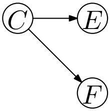
D gets number 3.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

1: $k = 1$
2: **while** $V \neq \{\,\}$ **do**
3: Choose a vertex $u \in V$ with indegree 0.
4: Give u the number k.
5: $k = k + 1$
6: Remove u from G.
7: **end while**



D gets number 3.

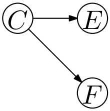
Remove D from G.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while



C gets number 4.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```

C gets number 4.

Remove C from G.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while

We can pick E or F.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```

We can pick E or F.

Let us choose E.

E gets number 5.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```

We can pick E or F.

Let us choose E.

E gets number 5.

Remove E from G.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\ \}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while

F gets number 6.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\}$ **do**
  Choose a vertex $u \in V$ with indegree 0.
  Give u the number k.
  $k = k + 1$
  Remove u from G.
end while

F gets number 6.

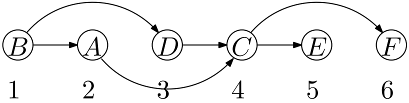
Remove F from G.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{ \}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while



Prenumbers and Postnumbers

Let $G = (V, E)$ be a directed graph. For each vertex $v \in V$, we define the following two numbers with respect to Depth-First-Search.

- $pre(v)$: the first time we visit v (the time at which $explore(v)$ is called)
- $post(v)$: the time at which $explore(v)$ is finished

Prenumbers and Postnumbers

Let $G = (V, E)$ be a directed graph. For each vertex $v \in V$, we define the following two numbers with respect to Depth-First-Search.

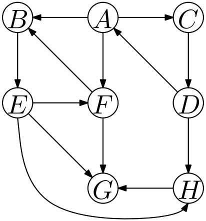
$pre(v)$: the first time we visit v (the time at which $explore(v)$ is called)

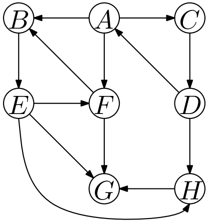
$post(v)$: the time at which $explore(v)$ is finished

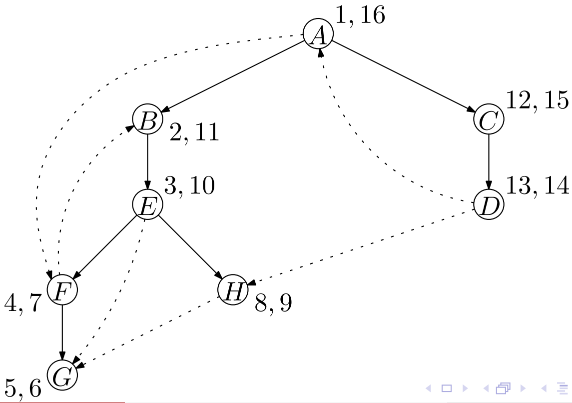
Use variable $clock$. At start, $clock = 1$.

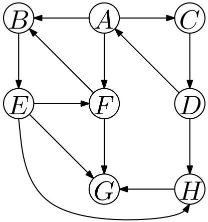
$$previsit(v) \equiv \quad \begin{aligned} & pre(v) = clock \\ & clock = clock + 1 \end{aligned}$$

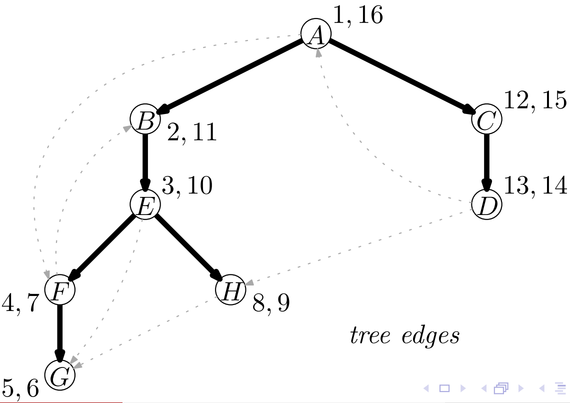
$$postvisit(v) \equiv \quad \begin{aligned} & post(v) = clock \\ & clock = clock + 1 \end{aligned}$$

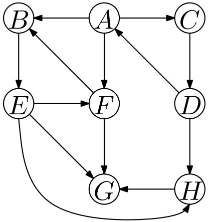

B
A
C
E
F
D
G
H

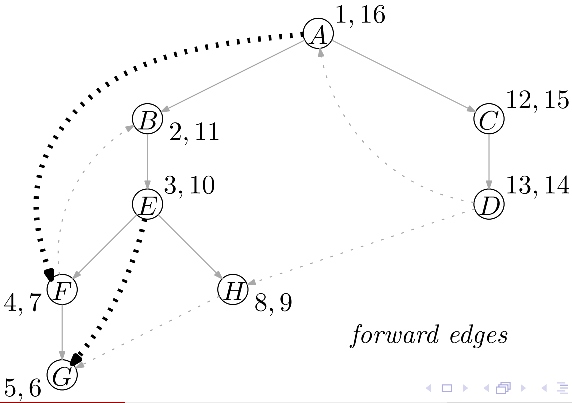

B
A
C
E
F
D
G
H

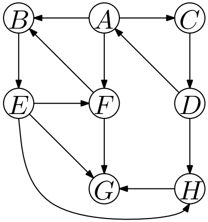

A
1, 16
B
2, 11
C
12, 15
E
3, 10
D
13, 14
F
4, 7
H
8, 9
G
5, 6

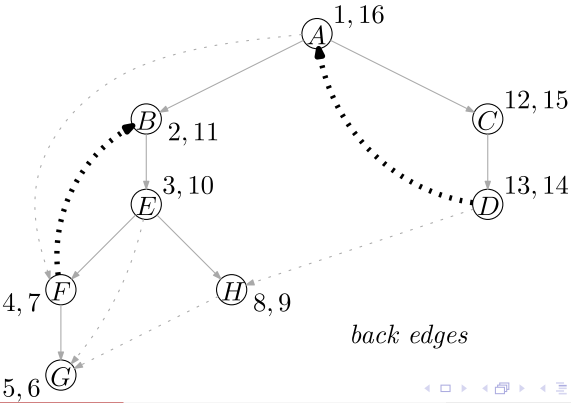

B
A
C
E
F
D
G
H

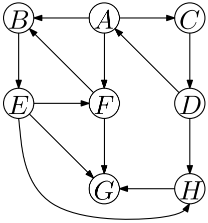

A 1, 16
B 2, 11
C 12, 15
E 3, 10
D 13, 14
F 4, 7
H 8, 9
G 5, 6
tree edges

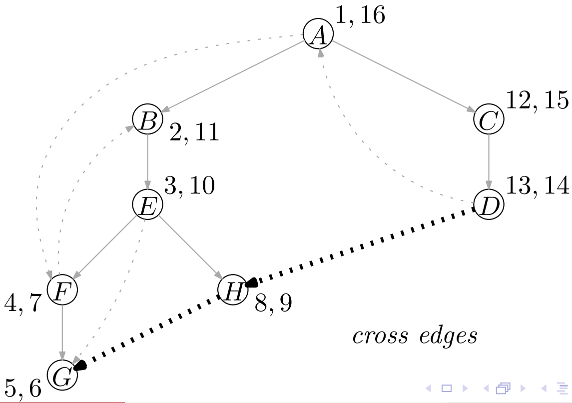

B
A
C
E
F
D
G
H


A
1, 16
B
2, 11
C
12, 15
E
3, 10
D
13, 14
F
4, 7
H
8, 9
G
5, 6
forward edges

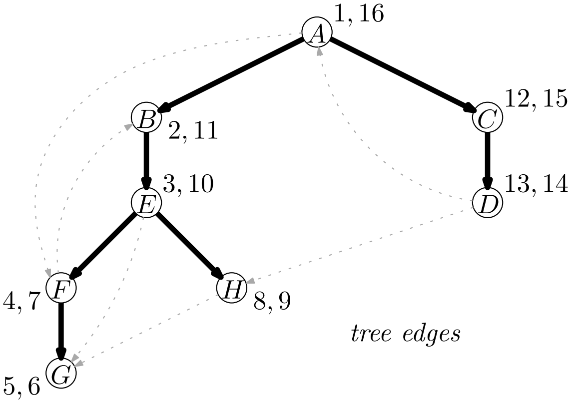

A
B
C
E
F
D
G
H


A 1, 16
B 2, 11
C 12, 15
E 3, 10
D 13, 14
F 4, 7
H 8, 9
G 5, 6
back edges


B
A
C
E
F
D
G
H


A
1, 16
B
2, 11
C
12, 15
E
3, 10
D
13, 14
F
4, 7
H
8, 9
G
5, 6
cross edges

4 Types of Edges

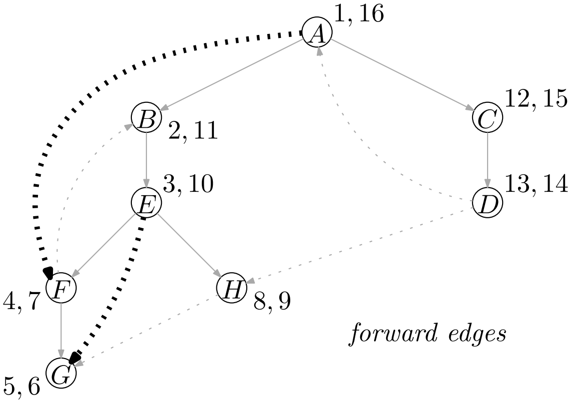


Tree edge:

- edge $v \longrightarrow u$
- $explore(u)$ is called as a recursive call within $explore(v)$

Solid edges

4 Types of Edges



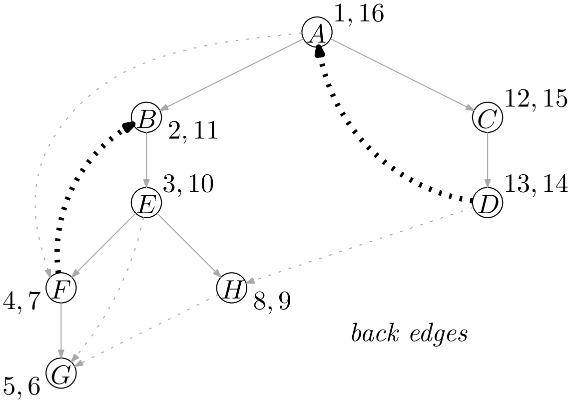
forward edges

Forward edge:

- edge $v \longrightarrow u$ where in the (solid) tree,
  - $\cdot$ u is in subtree of v
  - $\cdot$ u is not a child of v

(A, F), (E, G)

4 Types of Edges



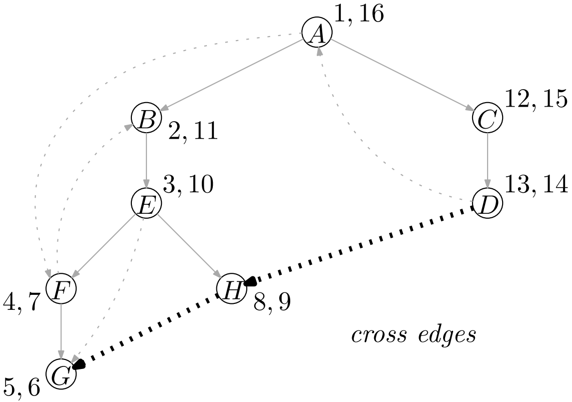
back edges

Back edge:

- edge $v \longrightarrow u$ where in the (solid) tree, $\cdot$ v is in subtree of u

(F, B), (D, A)

4 Types of Edges



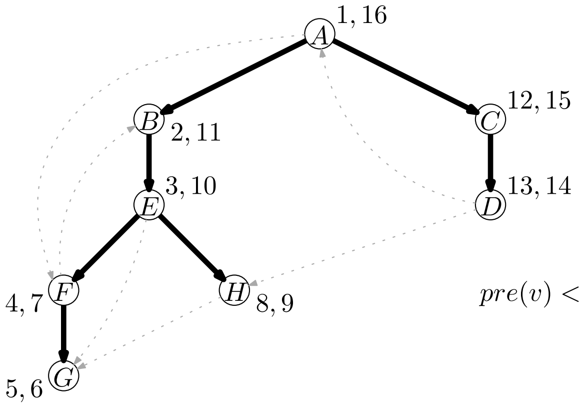
Cross edge:

- All other edges

(D, H), (H, G)

4 Types of Edges

How to decide the type of an edge?



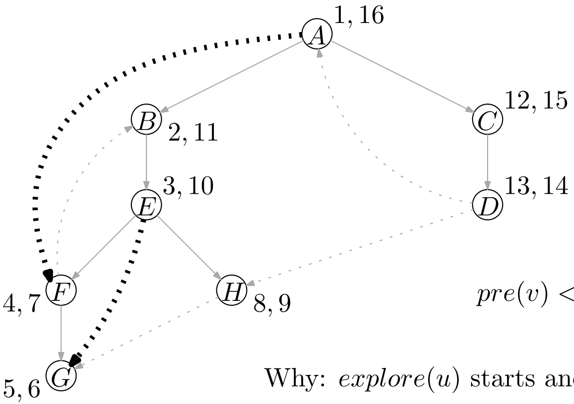
Tree edge:

These are discovered during the execution of the algorithm.

$$pre(v) < pre(u) < post(u) < post(v)$$

4 Types of Edges

How to decide the type of an edge?



Forward edge:

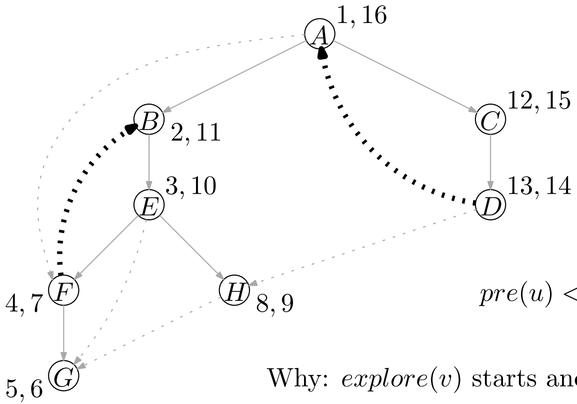
(v, u) not a tree edge

$$pre(v) < pre(u) < post(u) < post(v)$$

Why: $explore(u)$ starts and finishes within $explore(v)$.

4 Types of Edges

How to decide the type of an edge?



Back edge:

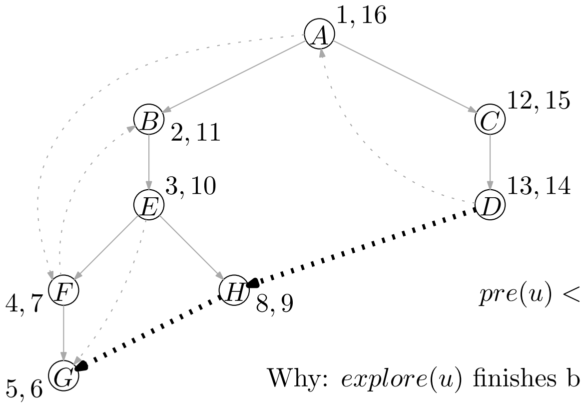
(v, u) where

$$pre(u) < pre(v) < post(v) < post(u)$$

Why: $explore(v)$ starts and finishes within $explore(u)$.

4 Types of Edges

How to decide the type of an edge?



Cross edge:

(v, u) where

$$pre(u) < post(u) < pre(v) < post(v)$$

Why: $explore(u)$ finishes before $explore(v)$ starts.

Acyclic vs Cyclic

How to decide if a directed graph has a directed cycle?

Lemma

G has a directed cycle
if and only if
DFS-forest has a back-edge.