ALGORITMI E STRUTTURE DATI

Prof. Manuela Montangero

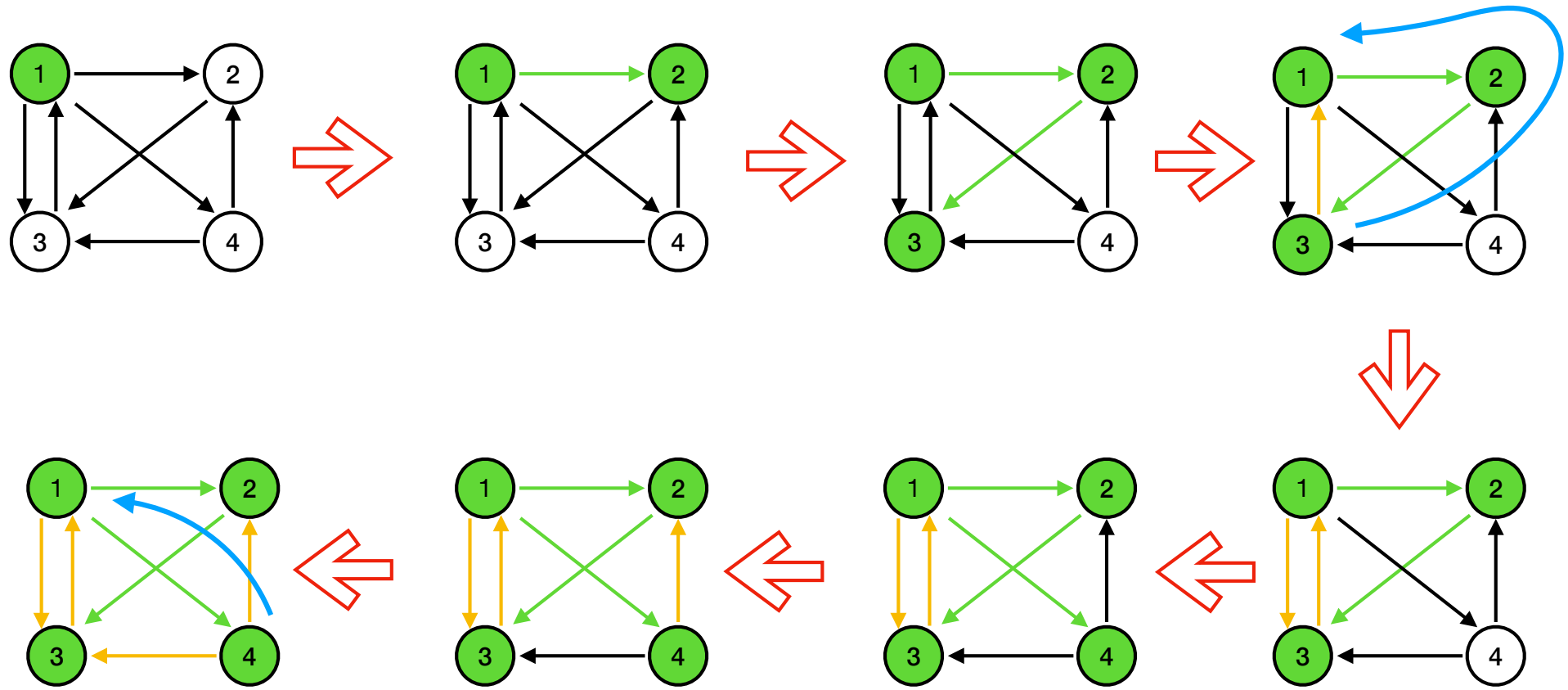
A.A. 2022/23

DFS e ricerca di un ciclo in un grafo diretto, classificazione archi

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO ORIENTATO



Gli algoritmi di vista DFS e foresta di copertura
funzionano senza modifiche anche nel caso di grafo diretto

Ricerca ciclo in un grafo orientato

TEST GRAFO ORIENTATO con CICLO:
INPUT: grafo ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

IDEA:

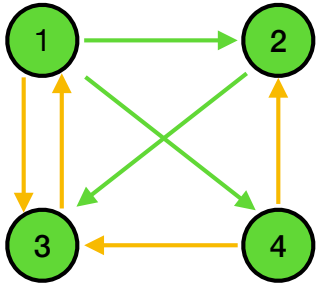
- Facciamo una DFS
- Se troviamo un back-edge, nel grafo c'e' un ciclo
- Altrimenti no

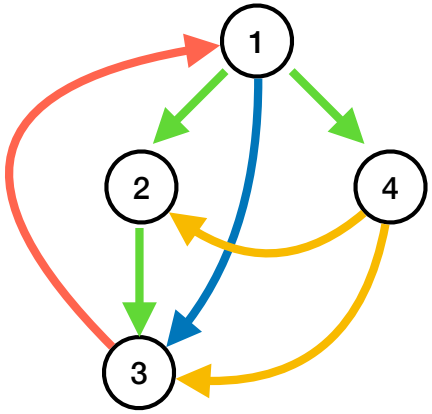
arco che porta ad un antenato dell'albero di copertura della visita DFS

?

Ricerca ciclo in un grafo orientato

TEST GRAFO ORIENTATO con CICLO:
INPUT: grafo ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti





Gli archi che non fanno parte dell'albero di copertura non indicano sempre la presenza di un ciclo!

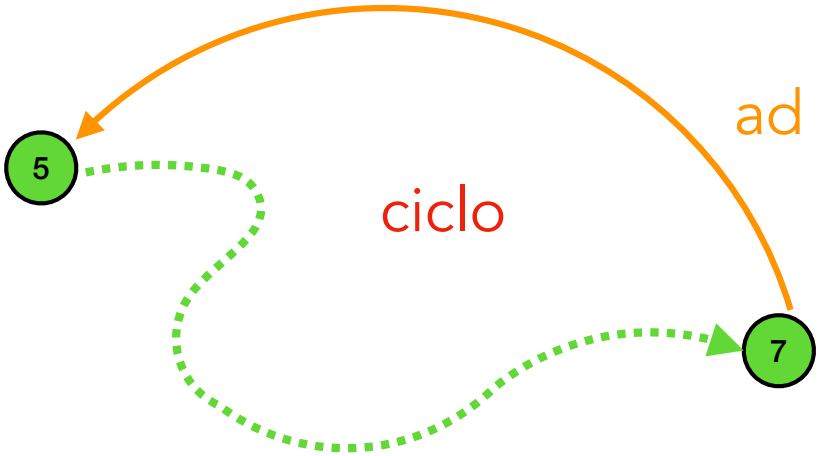
- **Tree edge**: arco dell'albero di copertura
- **Back edge**: porta ad un **antenato**
- **Forward edge**: porta ad un **discendente**
- **Cross edge**: porta ad un nodo che non è né antenato né discendente

Ricerca ciclo in un grafo orientato

TEST GRAFO ORIENTATO con CICLO:
INPUT: grafo ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

Un grafo orientato ha un ciclo
SE e SOLO SE
esiste un arco classificato come BACKEDGE in una visita DFS

(⇐)



arco che porta
ad un antenato nell'albero
di copertura

ciclo

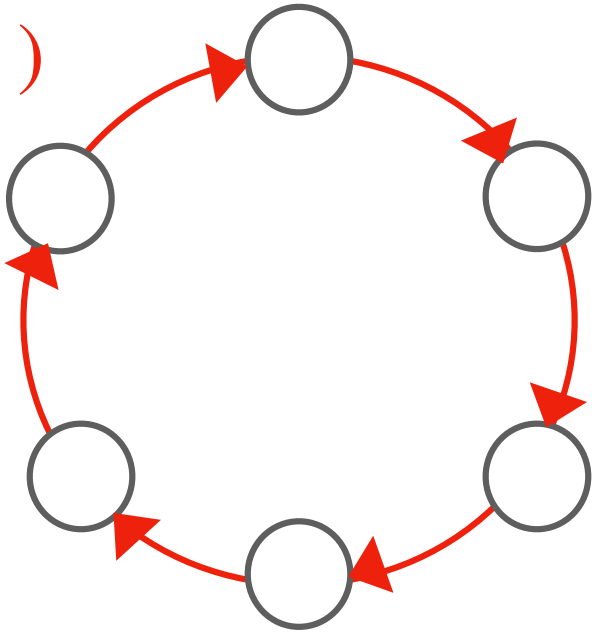
cammino diretto sull'albero
di copertura

Ricerca ciclo in un grafo orientato

TEST GRAFO ORIENTATO con CICLO:
INPUT: grafo ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

Un grafo orientato ha un ciclo
SE e SOLO SE
esiste un arco classificato come BACKEDGE in una visita DFS

($\Rightarrow$)



Il grafo ha un ciclo

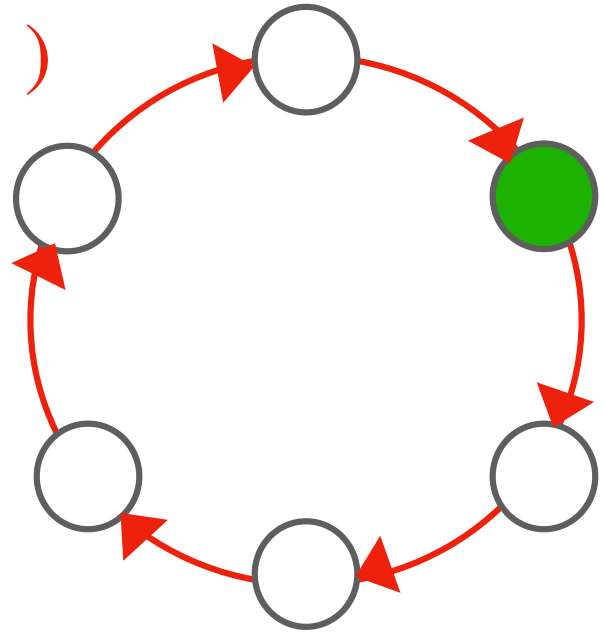
Prova del tutto analoga al caso non orientato

Ricerca ciclo in un grafo orientato

TEST GRAFO ORIENTATO con CICLO:
INPUT: grafo ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

Un grafo orientato ha un ciclo
SE e SOLO SE
esiste un arco classificato come BACKEDGE in una visita DFS

(⇒)



Primo nodo raggiunto
dalla DFS

e la DFS continua in profondità

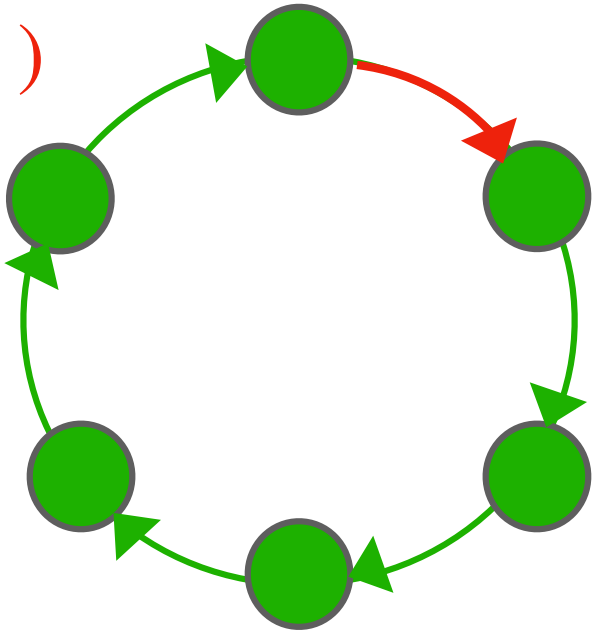
Il grafo ha un ciclo

Ricerca ciclo in un grafo orientato

TEST GRAFO ORIENTATO con CICLO:
INPUT: grafo ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

Un grafo orientato ha un ciclo
SE e SOLO SE
esiste un arco classificato come BACKEDGE in una visita DFS

(⇒)



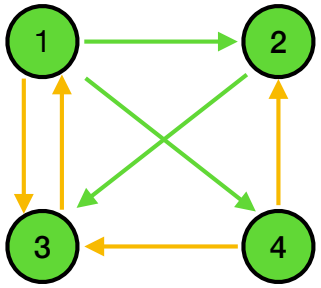
Il grafo ha un ciclo

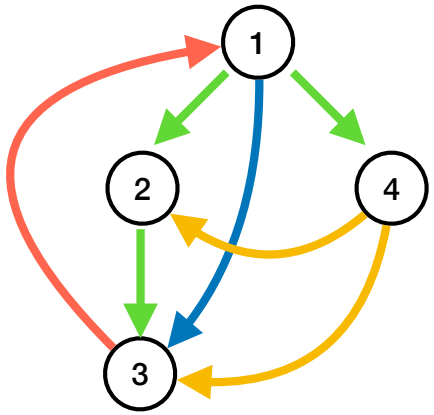
L'ultimo arco del ciclo porta ad un nodo già visitato che e' un antenato nell'albero di copertura

⇒ L'arco e' un backedge

Ricerca ciclo in un grafo orientato

TEST GRAFO ORIENTATO con CICLO:
INPUT: grafo ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

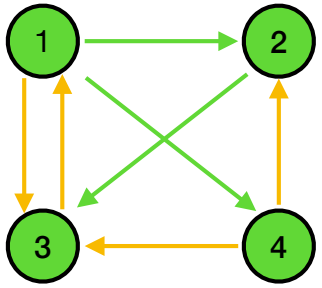


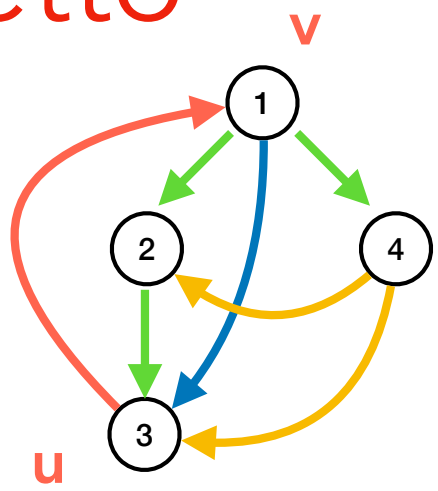


IDEA: fare una visita DFS per determinare l'esistenza di un backedge

N.B. gli archi FROWARD e CROSS non permettono di formare cicli, in quanto la direzione degli archi "è sbagliata"

Classificazione degli archi di un grafo diretto



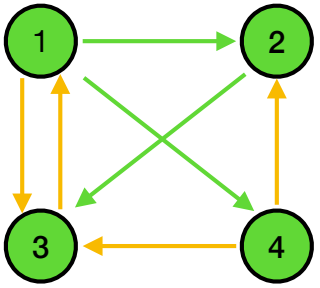


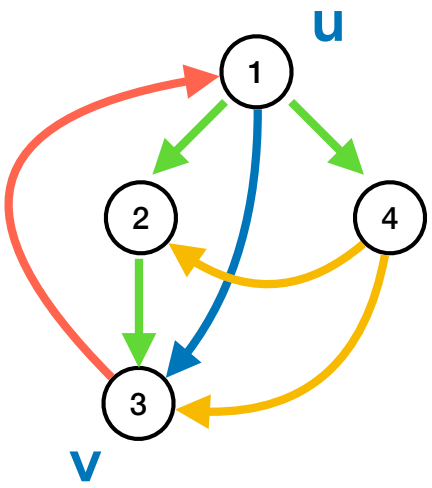
- **Back edge (u,v)**

La visita dell'antenato v È già iniziata quando si esplora (u,v), ma non ancora terminata

La visita del discendente u termina prima di quando termina la visita dell'antenato v

Classificazione degli archi di un grafo diretto



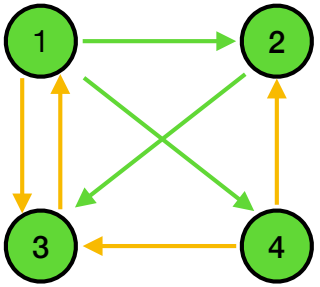


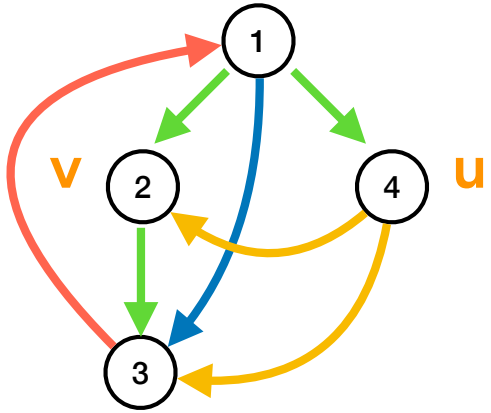
- **Forward edge (u,v)**

La visita dell'antenato u È già iniziata quando inizia la visita del discendente v

La visita del discendente v È già terminata quando si esplora (u,v)

Classificazione degli archi di un grafo diretto

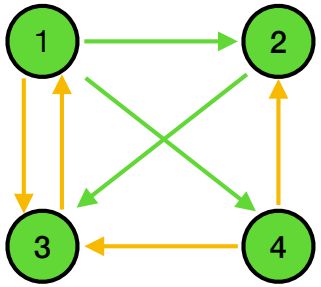


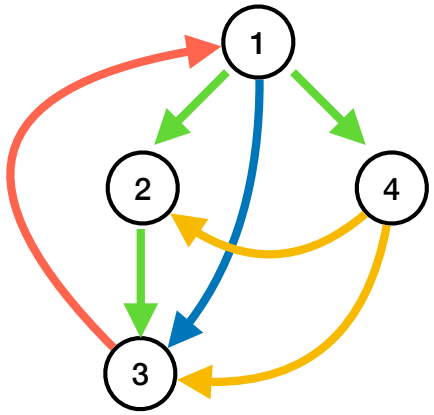


- **Cross edge (u,v)**

  La visita di v È già terminata quando inizia la visita di v,
  e quindi anche di quando si esplora (u,v)

Classificazione degli archi (grafo diretto)





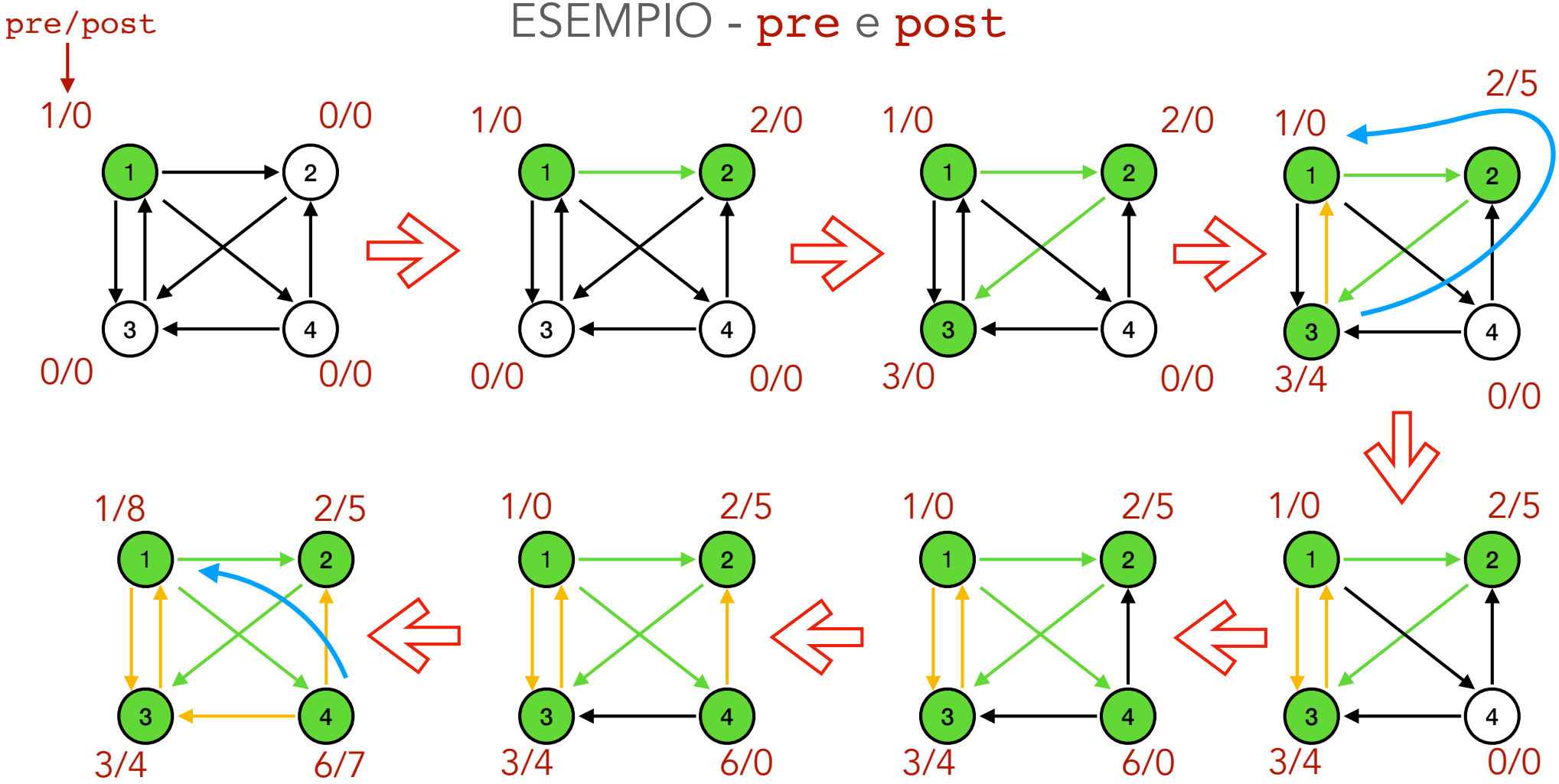
Classifichiamo gli archi DURANTE la visita DFS

Contatore `time` per tenere traccia di **due eventi** specifici:

1. Istante di tempo in cui viene "**scoperto**" un nuovo nodo (e inizia la sua visita)
   —> usiamo un array `pre[1..n]`
2. Istante di tempo in cui viene "**terminata**" la visita di un nodo
   —> usiamo un array `post[1..n]`

VISITA in PROFONDITA' (DFS)

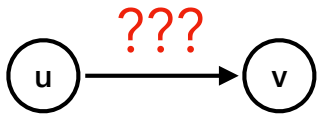
ESEMPIO - **pre** e **post**



Classificazione degli archi (grafo diretto)

INPUT: `G = (V,E)` orientato
OUTPUT: classificazione degli archi in **TREE**, **BACK**, **FORWARD** e **CROSS**

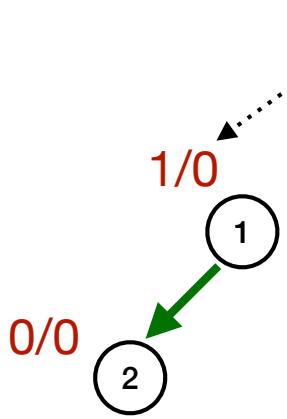
La visita DFS sta analizzando tutti gli archi `(u,v)` uscenti dal nodo `u`



- Se `pre[v] = 0`, significa che il nodo `v`
  non è ancora stato scoperto
  e che viene scoperto adesso per la
  prima volta

L'arco è **TREE**

ESEMPIO `pre[v]/post[v]`

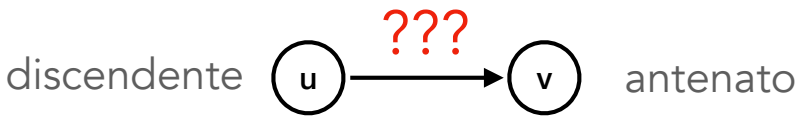


Quando la visita analizza l'arco (1,2)
si ha che `pre[2] = 0`,
quindi l'arco (1,2) viene classificato
come **TREE**

Classificazione degli archi (grafo diretto)

INPUT: `G = (V,E)` orientato
OUTPUT: classificazione degli archi in **TREE**, **BACK**, **FORWARD** e **CROSS**

La visita DFS sta analizzando tutti gli archi `(u,v)` uscenti dal nodo `u`

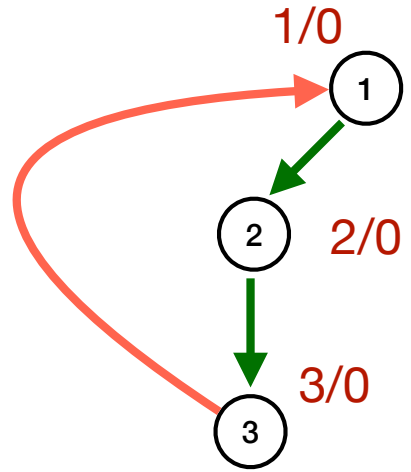


- **Back edge (u,v)**

La visita dell'antenato v È già iniziata quando si esplora (u,v), **ma non ancora terminata**

La visita del discendente u inizia DOPO la visita dell'antenato v

ESEMPIO

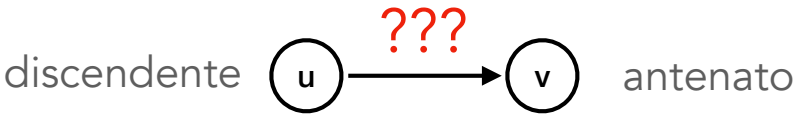


`u = 3, v = 1`
`pre[1]= 1 < 3 = pre[3]`
`post[1]= 0`

Classificazione degli archi (grafo diretto)

INPUT: `G = (V,E)` orientato
OUTPUT: classificazione degli archi in **TREE**, **BACK**, **FORWARD** e **CROSS**

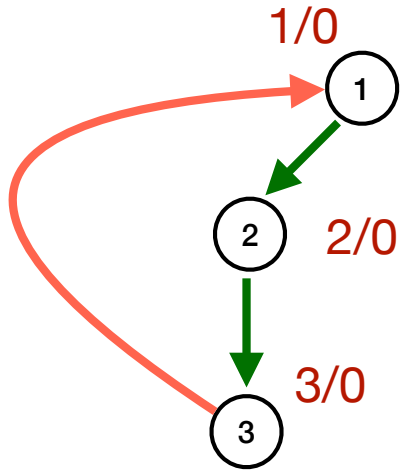
La visita DFS sta analizzando tutti gli archi `(u,v)` uscenti dal nodo `u`



- Se `pre[v] < pre[u] e post[v] = 0`, il nodo `v` è stato scoperto prima di `u`, ma la sua visita non è ancora terminata perché `post[v]` è ancora a zero.

L'arco è **BACK**

ESEMPIO

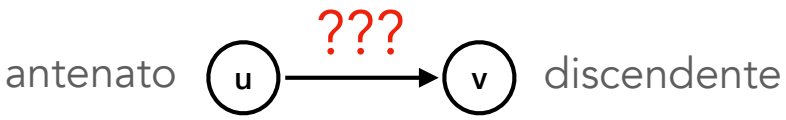


`u = 3, v = 1`
`pre[1]= 1 < 3 = pre[3]`
`post[1]= 0`

Classificazione degli archi (grafo diretto)

INPUT: `G = (V,E)` orientato
OUTPUT: classificazione degli archi in **TREE**, **BACK**, **FORWARD** e **CROSS**

La visita DFS sta analizzando tutti gli archi `(u,v)` uscenti dal nodo `u`

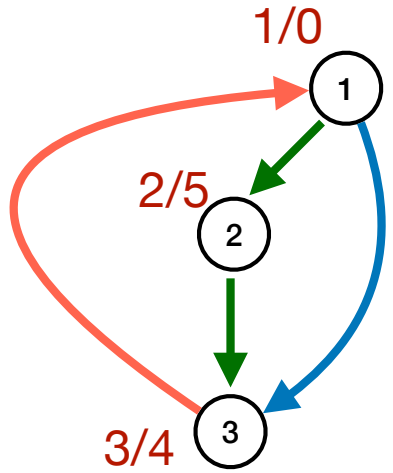


- **Forward edge (u,v)**

La visita dell'antenato u È già iniziata quando inizia la visita del discendente v

La visita del discendente v E' già terminata quando si esplora (u,v)

ESEMPIO

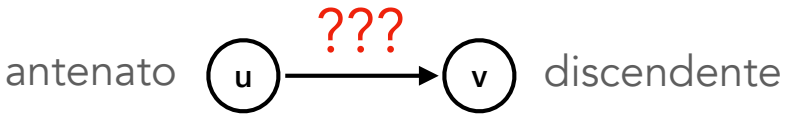


`u = 1, v = 3`
`pre[3]= 3 > 1 = pre[1]`
`post[3] > 0`

Classificazione degli archi (grafo diretto)

INPUT: `G = (V,E)` orientato
OUTPUT: classificazione degli archi in **TREE**, **BACK**, **FORWARD** e **CROSS**

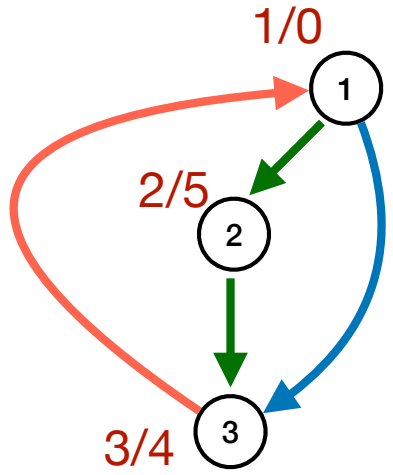
La visita DFS sta analizzando tutti gli archi `(u,v)` uscenti dal nodo `u`



- Se `pre[v] > pre[u]` e post[v] > 0: il nodo `v` è stato scoperto per la prima volta dopo di `u`, e la sua visita è già terminata, perché post[v] > 0.

L'arco è **FORWARD**

ESEMPIO



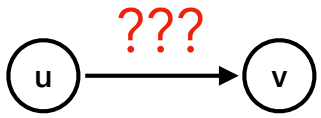

```
u = 1, v = 3
pre[3]= 3 > 1 = pre[1]
post[3] > 0
```

Classificazione degli archi (grafo diretto)

INPUT: `G = (V,E)` orientato
OUTPUT: classificazione degli archi in **TREE**, **BACK**, **FORWARD** e **CROSS**

La visita DFS sta analizzando tutti gli archi `(u,v)` uscenti dal nodo `u`

ne' antenato
ne' discendente

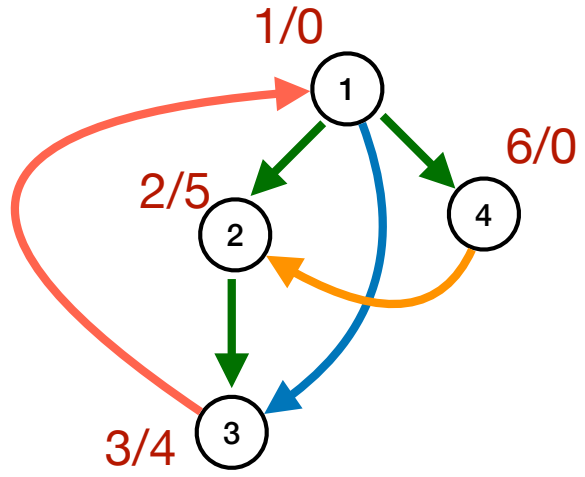


ne' antenato
ne' discendente

- **Cross edge (u,v)**

La visita di v È già terminata quando inizia la visita di v,
e quindi anche di quando si esplora (u,v)

ESEMPIO



`u = 4, v = 2`

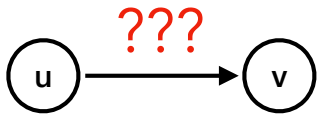
`post[2]= 5 < 6 = pre[4]`

Classificazione degli archi (grafo diretto)

INPUT: `G = (V,E)` orientato
OUTPUT: classificazione degli archi in **TREE**, **BACK**, **FORWARD** e **CROSS**

La visita DFS sta analizzando tutti gli archi `(u,v)` uscenti dal nodo `u`

ne' antenato
ne' discendente

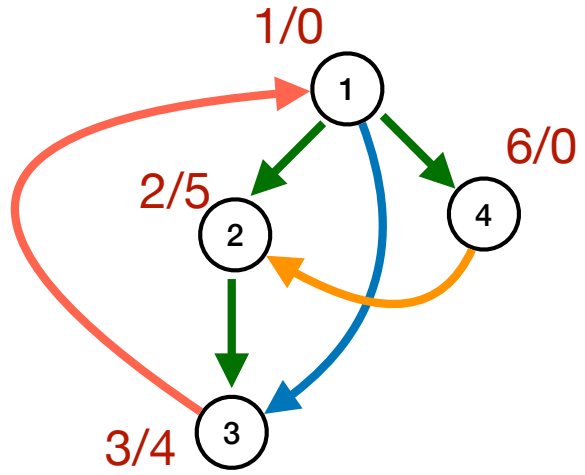


ne' antenato
ne' discendente

- Se `0 < post[v]< pre[u]`:
  il nodo `u` è stato scoperto per la prima volta dopo che la visita di `v` è terminata.

L'arco è **CROSS**

ESEMPIO



`u = 4, v = 2`

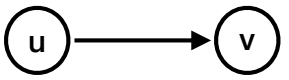
`0 < post[2]= 5 < 6 = pre[4]`

Classificazione degli archi (grafo diretto)

INPUT: G = (V,E) orientato
OUTPUT: classificazione degli archi in **TREE**, **BACK**, **FORWARD** e **CROSS**

```
DFS(G)
  for all v ∈ V
    pre[v] := 0
    post[v] := 0
  time := 0
  for all u ∈ V
    if pre[u] = 0
      then DFS-Visit(G,u)
```

```
DFS-Visit(G,u)
  time := time +1
  pre[u] := time //inizia visita di u
  for all (u,v) ∈ E // archi uscenti da u
    if pre[v] = 0
      then
        arco (u,v) è TREE
        DFS-Visit(G,v)
      else
        if post[v]=0
          // visita di v non finita
          then arco (u,v) è BACK
          else if pre[v] > pre[u]
              // visita di u iniziata prima di v e di v finita
              then arco (u,v) è FORWARD
              else arco (u,v) è CROSS
  time := time +1
  post[u] := time //termina visita di u
```



Costo computazionale $O(|V|+|E|)$

Ricerca ciclo in un grafo

TEST GRAFO ORIENTATO con CICLO:
INPUT: grafo ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

Modificare il codice della classificazione degli archi per individuare la presenza di almeno un arco BACK.

N.B. gli archi FORWARD e CROSS non permettono di formare cicli, in quanto la direzione degli archi "è sbagliata"