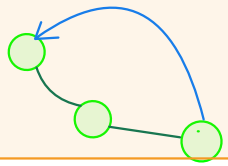
RICERCA DI UN CICLO IN UN GRAFO

Come faccio, utilizzando una DFS se ∃ un ciclo nel grafo? → SI FA UNA DFS

- Se nel grafo non c'è un ciclo (grafo aciclico) allora il grafo sarà un albero
- SE SI TROVA UN BACK EDGE ⇒ ∃ UN CICLO
- SE NON SI TROVA UN BACK EDGE ⇒ ∄ UN CICLO

BACK EDGE: arco che ci porta ad un antenato



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

GRAFO NON DIRETTO (non orientato): le coppie di E non sono ordinate

L'esplorazione di un arco ci porta in un nodo gia' visitato

GRAFO DIRETTO (orientato): le coppie di E sono ordinate

?

L'ultimo ARCO del ciclo ci porta ad un nodo già visitato ⇒ c'è un back edge ⇒ ∃ un ciclo

CHIEDERE!!

COME CAPIRE SE IL GRAFO È CONNESSO?

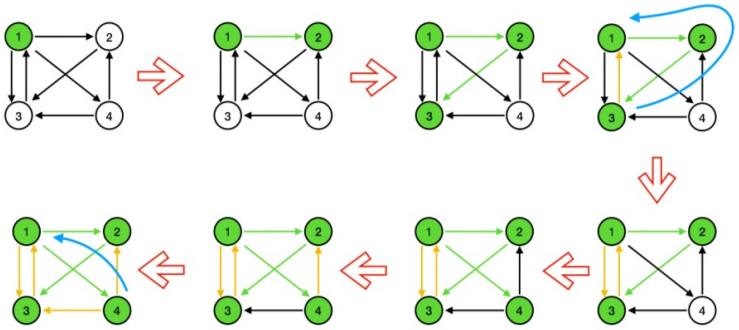
- DFS-VISIT IN UN NODO CASUALE
- CONTROLLO SE CI SONO NODI NON VISITATI NELL'ARRAY VISITED

```
For v∈V do
    if visited[v]=0
        then count=count+1 DFS-visit (G, V, count)
```

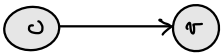
count conta le componenti non connesse

VISITA in PROFONDITA' (DFS)

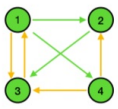
ESEMPIO - GRAFO ORIENTATO

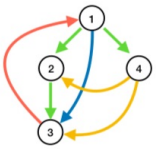


for (u,v) ∈ E do



CLASSIFICAZIONE DEGLI ARCHI DI UN GRAFO DIRETTO





La classificazione degli archi viene fatta durante una DFS

- **Back edge (u,v)**
  La visita dell'antenato v E' 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
- **Forward edge (u,v)**
  La visita dell'antenato u E' già iniziata quando inizia la visita del discendente v
  La visita del discendente v E' già terminata quando si esplora (u,v)
- **Cross edge (u,v)**
  La visita di v E' già terminata quando inizia la visita di v, e quindi anche di quando si esplora (u,v)

NON permettono di formare cicli

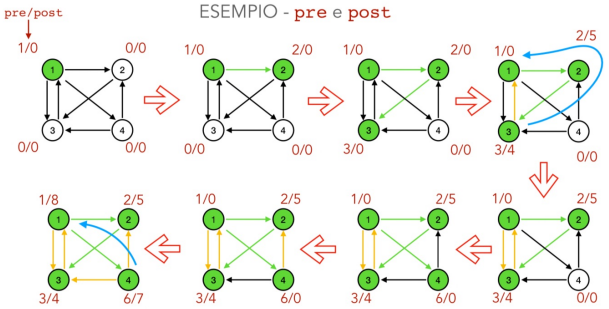
⇓

la direzione degli archi è sbagliata

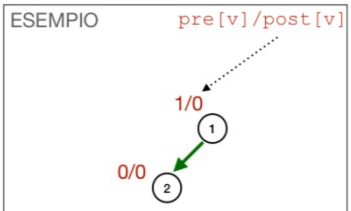
INTRODUCIAMO UN CONCETTO DI TEMPO ⇒ CONTATORE TIME INCREMENTATO QUANDO INIZIA E QUANDO TERMINA LA VISITA DI UN NODO

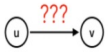
1. Istante di tempo in cui viene "**scoperto**" un nuovo nodo (e inizia la sua visita) —> usiamo un array `pre[1..n]`
2. Istante di tempo in cui viene "**terminata**" la visita di un nodo —> usiamo un array `post[1..n]`

- `pre[v]` e `post[v]` sono inizializzati a zero per ogni nodo v
- `pre[v] = 0` —> non e' ancora **iniziata** la visita di v, non è stata scoperta
- `post[v] = 0` —> non e' ancora **finita** la visita di v
- `pre[v] := time` quando v viene scoperto per la prima volta
- `post[v] := time` quando termina la visita di v
- `pre[v] < prev[u]` —> la visita di v e' **iniziata** prima della visita di u
- `post[v] < post[u]` —> la visita di v e' **finita** prima della visita di u

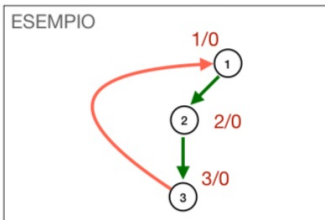


TREE ⇒ prev[v] = 0

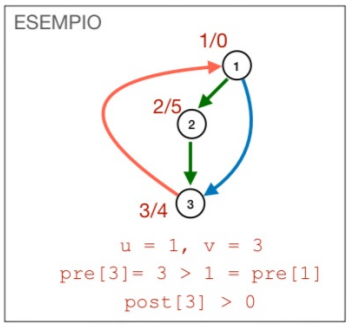


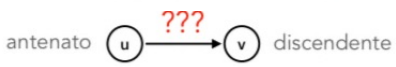


BACK ⇒ prev[v] < prev[u] & post[v] = 0

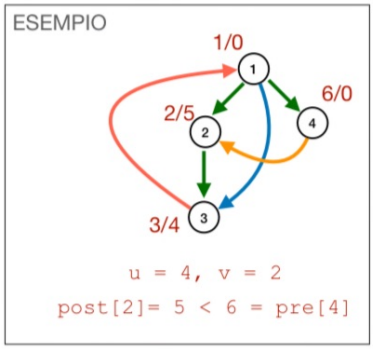


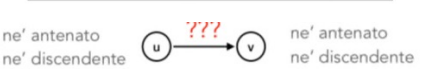
FORWARD ⇒ prev[u] < prev[v] & post[v] > 0





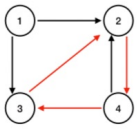
CROSS ⇒ post[u] > post[v] > 0



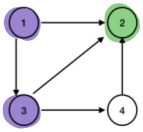


GRAFO DIRETTO ACICLICO
DAG (DIRECTED ACYCLIC GRAPH)

- É UN GRAFO DIRETTO SENZA CICLI



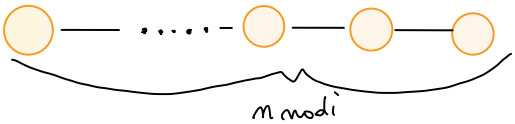
NO



SI

POZZO (SINK): non ha archi uscenti

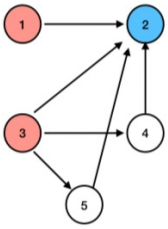
SORGENTE (source): non ha archi entranti

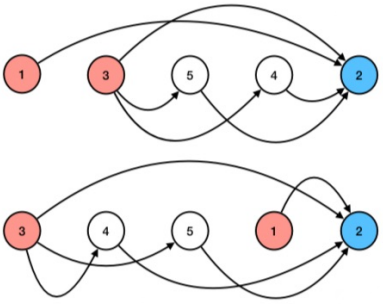


- SE SIAMO IN UN DAG CI DEVE ESSERE ALMENO 1 SORGENTE
- OPPURE NON SIAMO IN UN DAG E ABBIAMO UN CICLO

ORDINAMENTO TOPOLOGICO (linearizzazione)

Dato un DAG G=(V,E), un ordinamento topologico è un ordinamento lineare dei suoi nodi tali che, se (u,v) ∈ E, allora u viene prima di v nell'ordinamento.

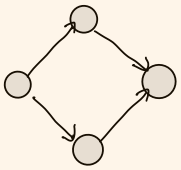




Gli archi sono sempre diretti da sinistra verso destra

LA LINEARIZZAZIONE VIENE FATTA PARTENDO DA UNA SORGENTE

ALBERO ⇓ DAG



DAG ma NON ALBERO

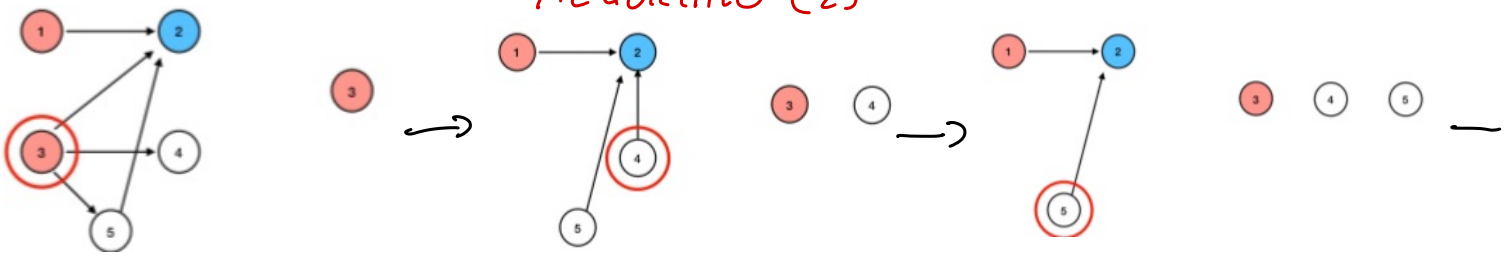
ALGORITMO PER LINEARIZZAZIONE (1)

while ci sono nodi
- Scegli una sorgente
- metti il nodo nella linearizzazione
- Togli il nodo (e i suoi archi uscenti) del grafo

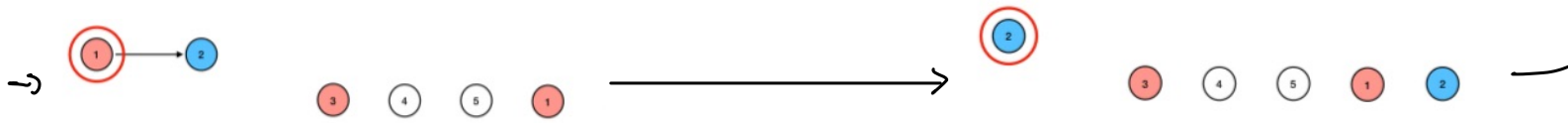
PROBLEMA: come faccio a trovare una sorgente? → Devo controllare tutti i nodi

OCCORRE fare una copia del grafo iniziale perché, per l'ultimo passo, andrò a rimuovere dei nodi perdendo il grafo originale

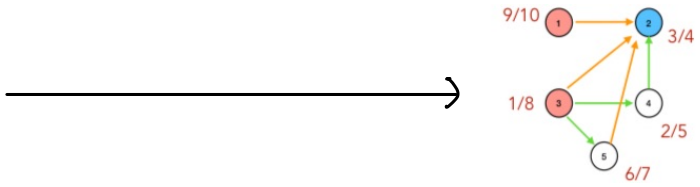
ALGORITMO (2)



INIZIO LA VISITA PARTENDO DA UNA SORGENTE



UTILIZZO UNA PILA



Ci mettiamo in ordine decrescente in base a post

7

Arco (u,v) **FORWARD**: `post[u] > post[v]` ⇨ il nodo u sta a sinistra del nodo v

Arco (u,v) **CROSS**: `post[v] < pre[u] < post[u]` ⇨ il nodo u sta a sinistra del nodo v

Arco (u,v) **TREE**: `post[u] > post[v]` ⇨ il nodo u sta a sinistra del nodo v

NOTA BENE: non possiamo avere anche BACK perché siamo in un DAG, altrimenti ci sarebbe un ciclo!

Si inseriscono i valori nella pila nel momento in cui termino la visita, estraendoli successivamente otterrò la linearizzazione desiderata

Il costo di inserimento è costante e, pertanto, non si cambia il costo di una DFS classica

Cosa cambia? Da una parte ho un'idea più semplice dell'altra. Abbiamo: