RICERCA IN PROFONDITÀ
COMPONENTI FORTEMENTE CONNESSE
ORDINAMENTO TOPOLOGICO

---

# ALGORITMI E STRUTTURE DATI

# VISITA IN PROFONDITA'

# RICERCA IN PROFONDITÀ

- La ricerca in profondità (depth-first search o DFS) è l'altro algoritmo standard di visita dei grafi
- Dato un grafo $G = (V, E)$ e un nodo $s \in V$ detto nodo sorgente la ricerca in profondità esplora tutti i nodi raggiungibili a partire da $s$.
- Se rimangono nodi non esplorati si ripete la ricerca in profondità su di essi*

* questo è fattibile anche con BFS, ma generalmente BFS si usa per trovare la distanza minima da un nodo sorgente, DFS si usa per altri scopi.

# RICERCA IN PROFONDITÀ

- Solitamente DFS salva due tempi:
  - Il tempo di scoperta d[v] di v, quando il nodo v è stato visitato per la prima volta (quando v diventa grigio)
  - Il tempo di fine visita f[v] di v, quando tutti i vicini di v sono stati visitati (quando v diventa nero)
- Questi due tempi sono poi utilizzati da altri algoritmi che hanno DFS come subroutine

# RICERCA IN PROFONDITÀ

- La ricerca in profondità può essere espressa in due modi diversi: ricorsivo e iterativo:
  - Ricorsivo è come viene solitamente presentata, ed il caso che vedremo.
- Una versione iterativa può essere ottenuta sostituendo nella BFS la coda con uno stack.

# RICERCA IN PROFONDITÀ: IDEA

- Dato un nodo $u \in V$
- Colora il nodo di grigio
- Se esiste scegli un nodo bianco adiacente e richiama ricorsivamente la visita in profondità su quel nodo
- Ripeti il punto precedente finché rimangono nodi bianchi
- Colora il nodo di nero

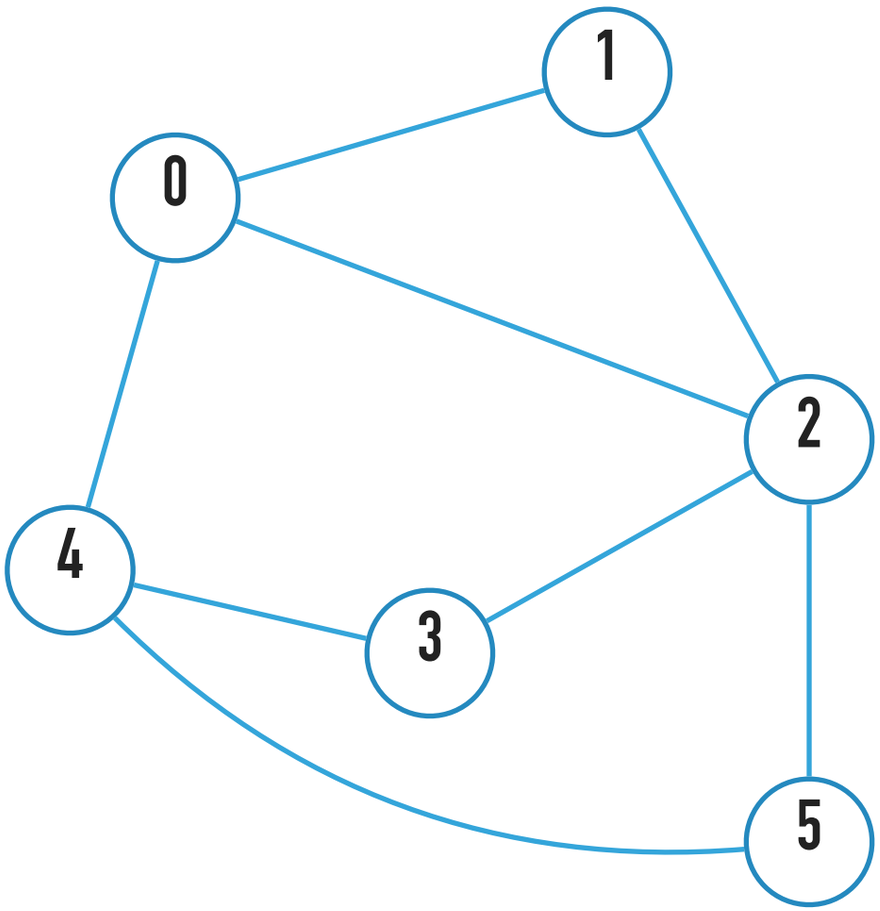
# PSEUDOCODICE: INIZIALIZZAZIONE

```
Parametri: grafo G
# inizialmente impostiamo colore e predecessore per tutti i nodi
for all v ∈ V:
    colore[v] = bianco
    predecessore[v] = None
tempo = 0 # un contatore globale per il tempo di visita dei nodi
for all u ∈ V
    if colore[u] == bianco
        DFS-VISIT(G, u) # chiamiamo la procedura di ricorsiva visita
```

# PSEUDOCODICE: PROCEDURA DFS-VISIT

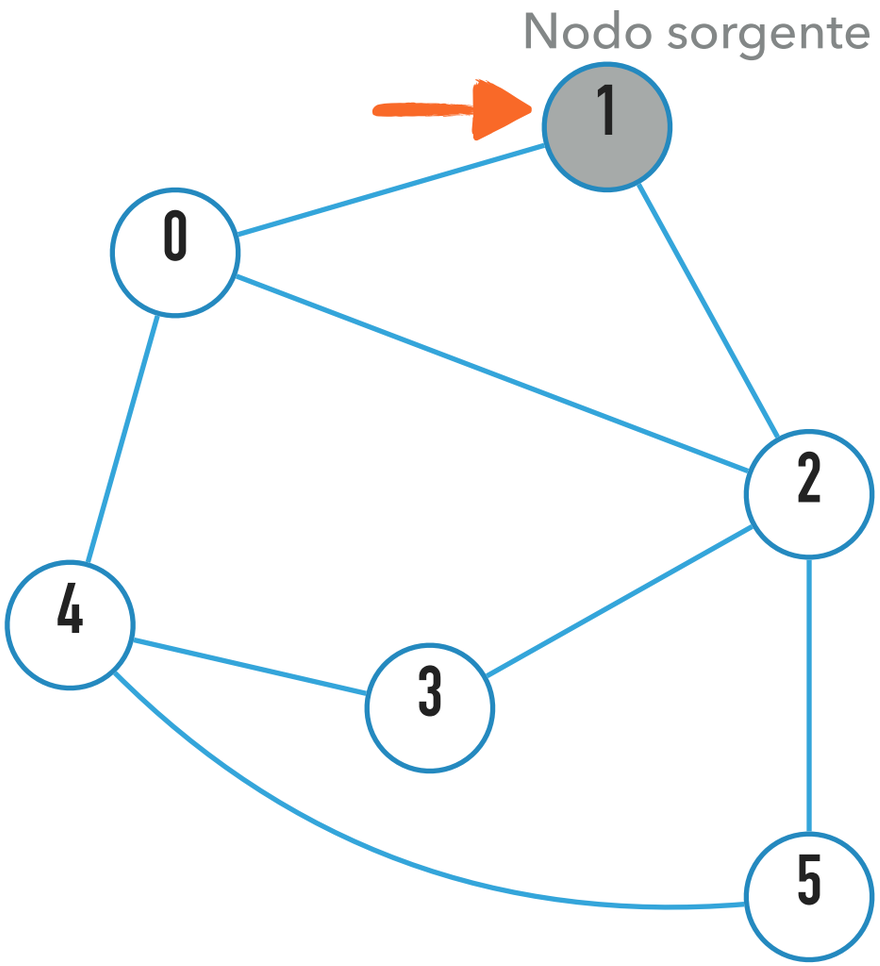
```
Parametri: grafo G, nodo u
tempo = tempo + 1 # incrementiamo il tempo globale
tempo_inizio[u] = tempo
colore[u] = grigio
for all v adiacenti a u
    if colore[v] == bianco # se è la prima volta che vediamo v
        precedessore[v] = u # ci siamo arrivati da u
        DFS-VISIT(G, v) # e ricorsivamente iniziamo la procedura di visita
colore[u] = nero # arrivati qui abbiamo visitato tutti i nodi adiacenti a u
tempo = tempo + 1
tempo_fine[u] = tempo
```

# ESEMPIO DI ESECUZIONE




Tempo **0**

# ESEMPIO DI ESECUZIONE




Stack di chiamate

NODO: 1

Iniziamo chiamando la procedura di visita sul primo nodo bianco che troviamo e lo coloriamo di grigio

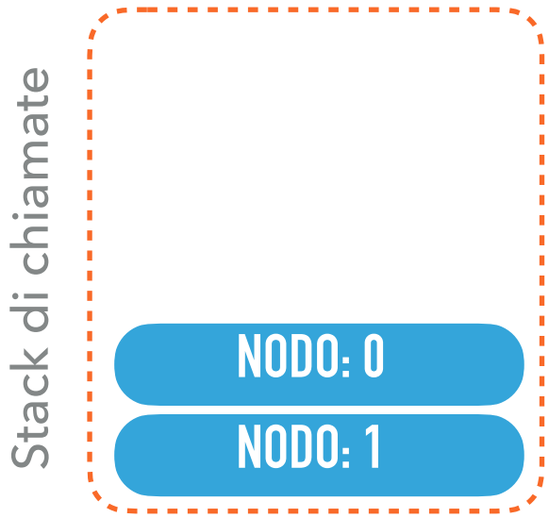
Tempo 1

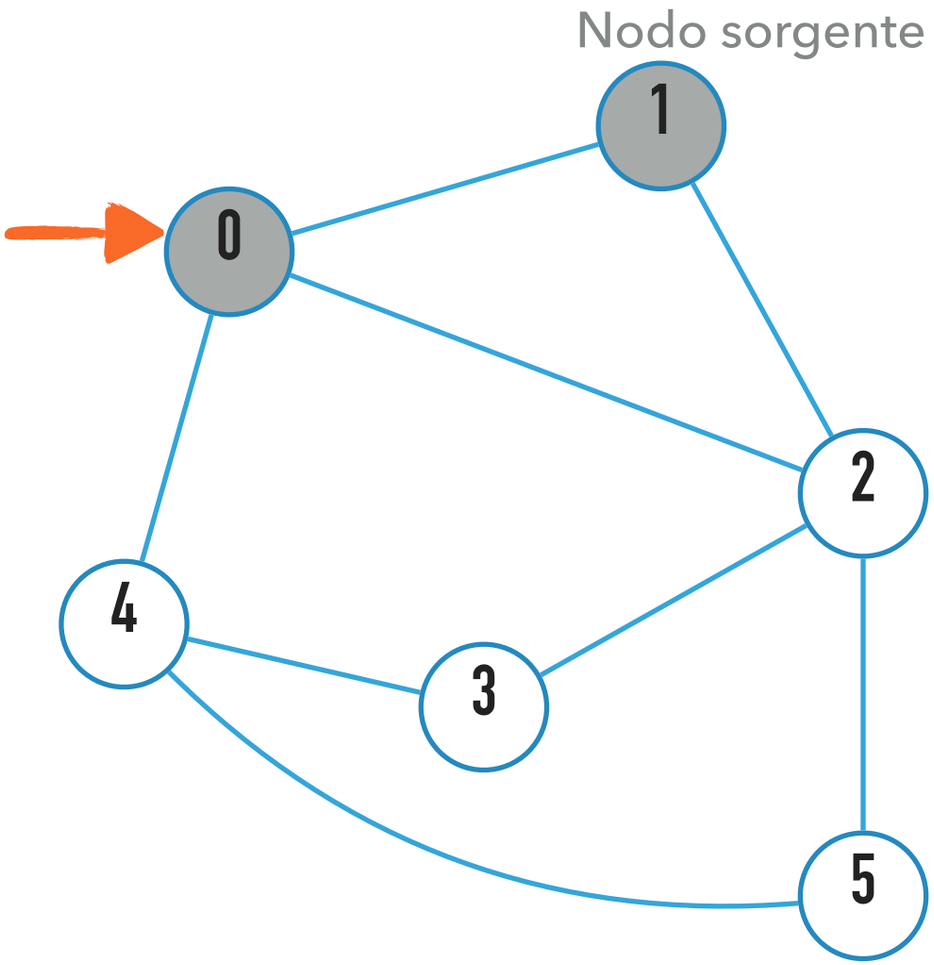
| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | | 1 | | | | |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_fine | | | | | | |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Predecessore | - | - | - | - | - | - |

# ESEMPIO DI ESECUZIONE




Chiamiamo la procedura di visita in profondità ricorsivamente su ciascuno dei nodi bianchi adiacenti quello corrente

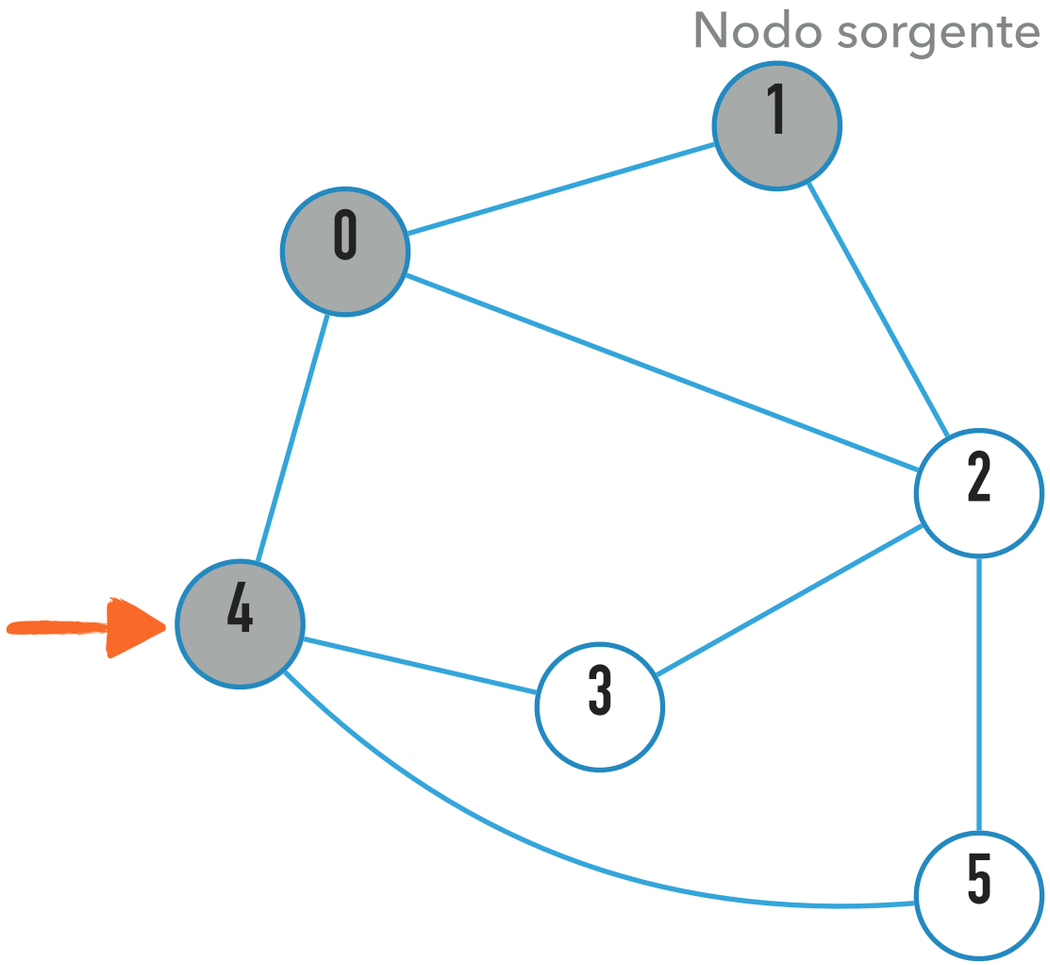
Tempo 2

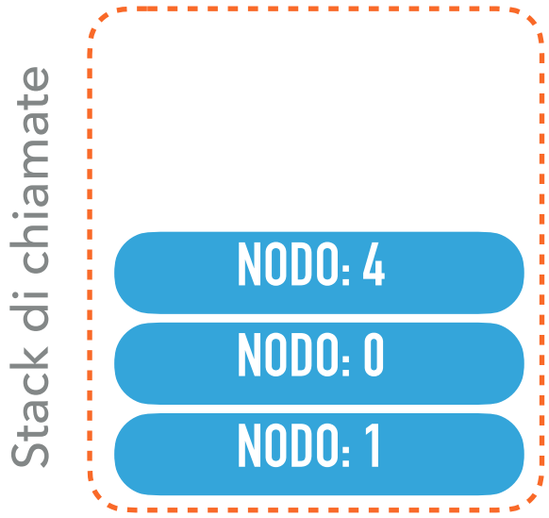
| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | 2 | 1 | | | | |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_fine | | | | | | |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Predecessore | 1 | - | - | - | - | - |

# ESEMPIO DI ESECUZIONE




Chiamiamo la procedura di visita in profondità ricorsivamente su ciascuno dei nodi bianchi adiacenti quello corrente

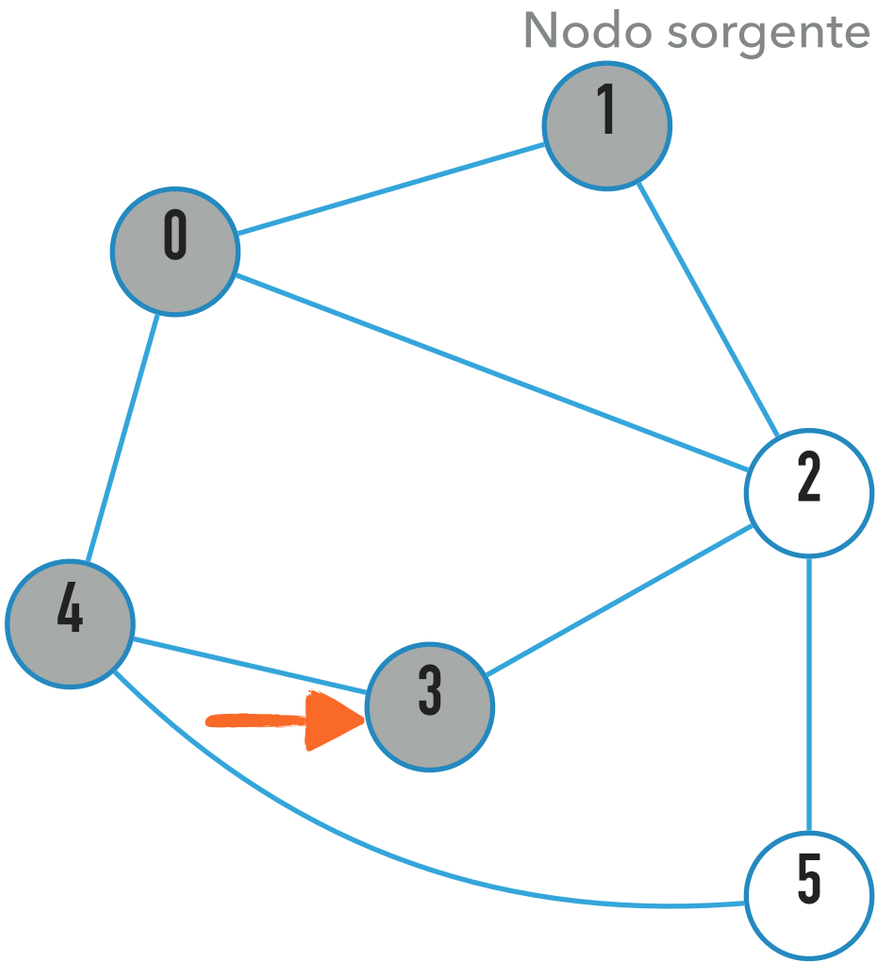
| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | 2 | 1 | | | 3 | |

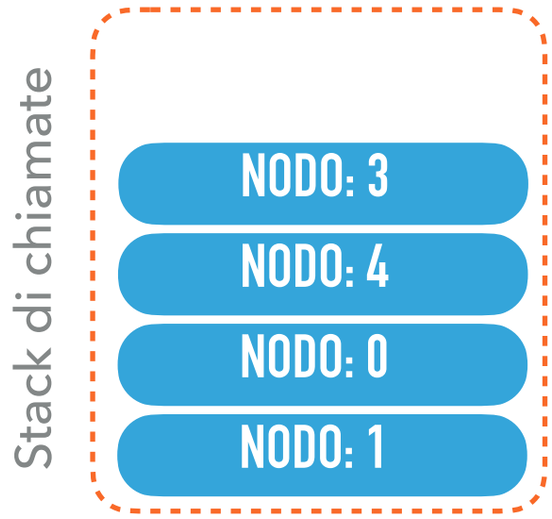
| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_fine | | | | | | |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Predecessore | 1 | - | - | - | 0 | - |

| Tempo | 3 |
|---|---|

# ESEMPIO DI ESECUZIONE



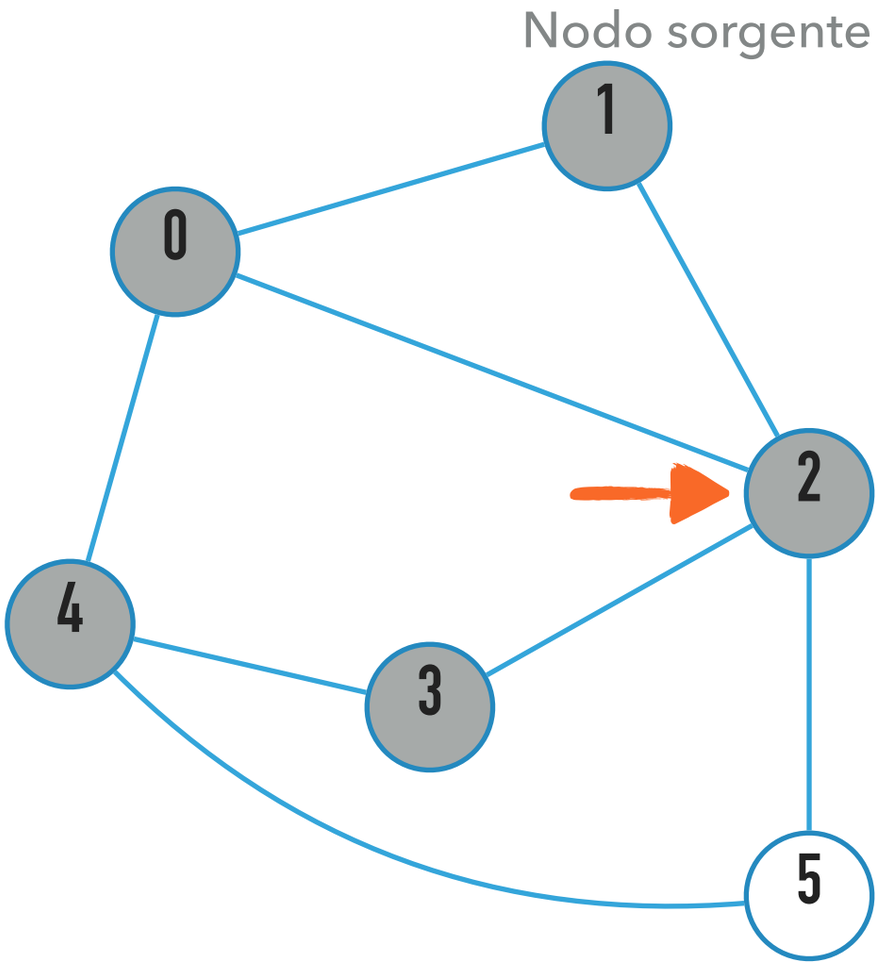




Chiamiamo la procedura di visita in profondità ricorsivamente su ciascuno dei nodi bianchi adiacenti quello corrente

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | 2 | 1 | | 4 | 3 | |
| Tempo_fine | | | | | | |
| Predecessore | 1 | - | - | 4 | 0 | - |

| Tempo | 4 |
|---|---|

# ESEMPIO DI ESECUZIONE



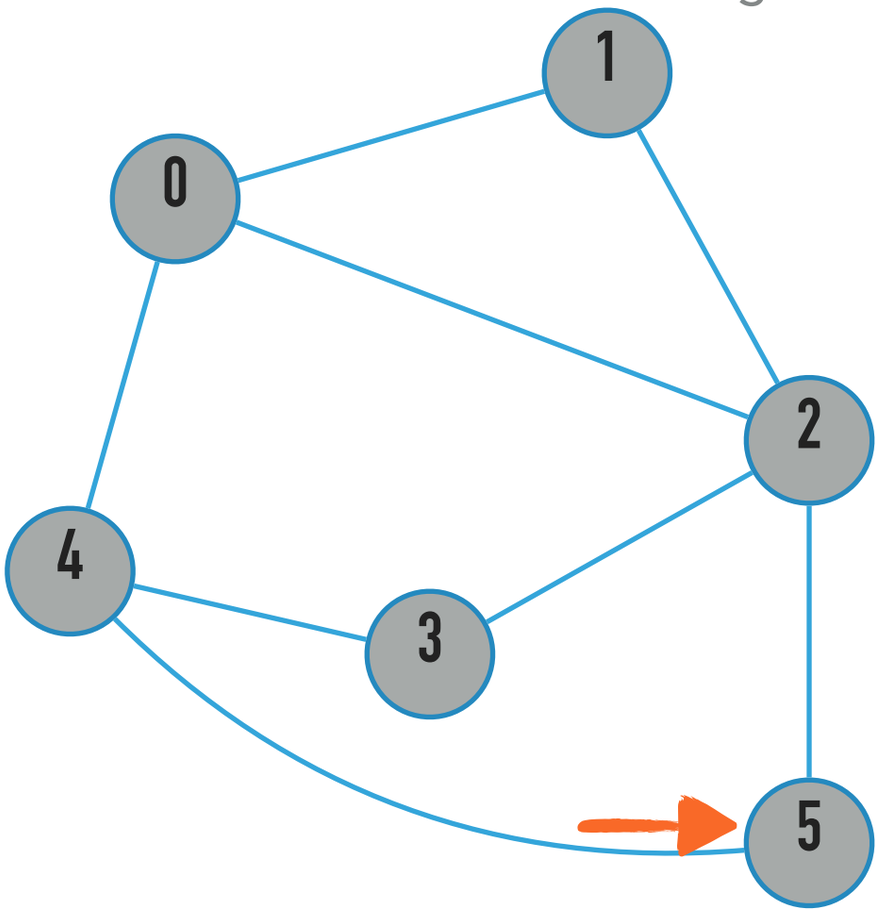

Stack di chiamate


Chiamiamo la procedura di visita in profondità ricorsivamente su ciascuno dei nodi bianchi adiacenti quello corrente

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | 2 | 1 | 5 | 4 | 3 | |
| Tempo_fine | | | | | | |
| Predecessore | 1 | - | 3 | 4 | 0 | - |

| Tempo | 5 |
|---|---|

# ESEMPIO DI ESECUZIONE




Stack di chiamate


Chiamiamo la procedura di visita in profondità ricorsivamente su ciascuno dei nodi bianchi adiacenti quello corrente

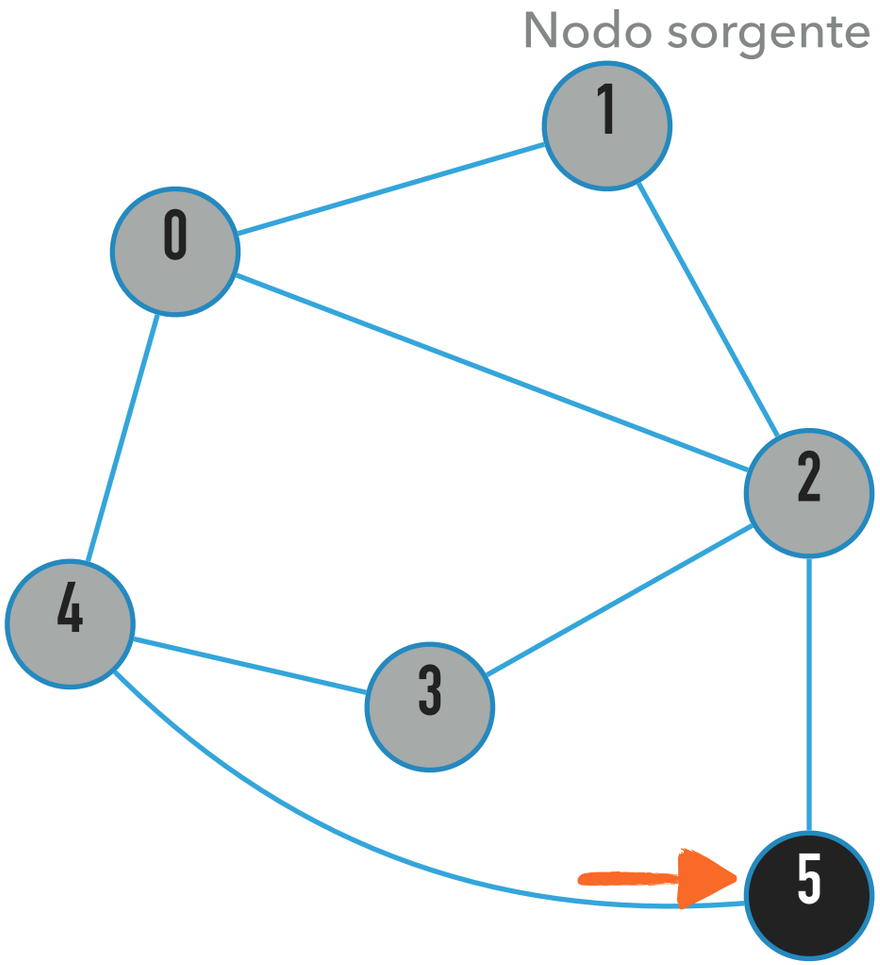
| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | 2 | 1 | 5 | 4 | 3 | 6 |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_fine | | | | | | |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Predecessore | 1 | - | 3 | 4 | 0 | 2 |

| Tempo | 6 |
|---|---|

# ESEMPIO DI ESECUZIONE




Stack di chiamate

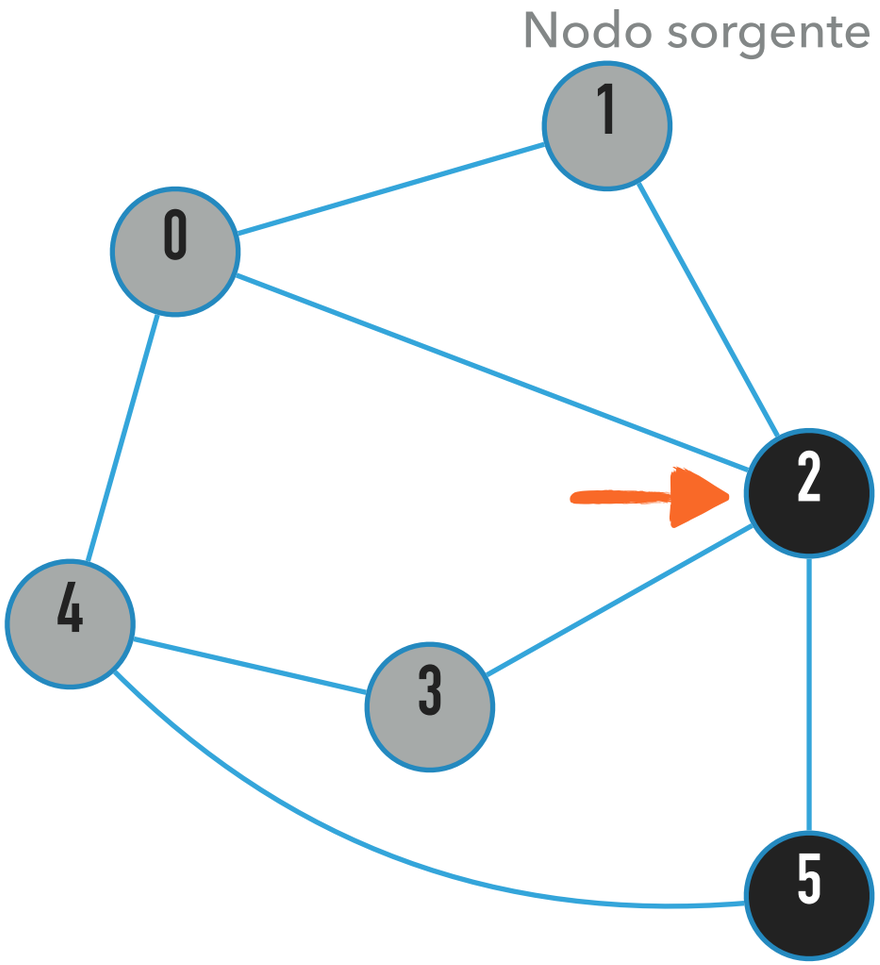
- NODO: 5
- NODO: 2
- NODO: 3
- NODO: 4
- NODO: 0
- NODO: 1

Dato che non possiamo più effettuare chiamate ricorsive, abbiamo finito di visitare il nodo corrente. aggiorniamo colore, tempo di fine e ritorniamo dalla chiamata ricorsiva

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | 2 | 1 | 5 | 4 | 3 | 6 |
| Tempo_fine | | | | | | 7 |
| Predecessore | 1 | - | 3 | 4 | 0 | 2 |

Tempo 7

# ESEMPIO DI ESECUZIONE



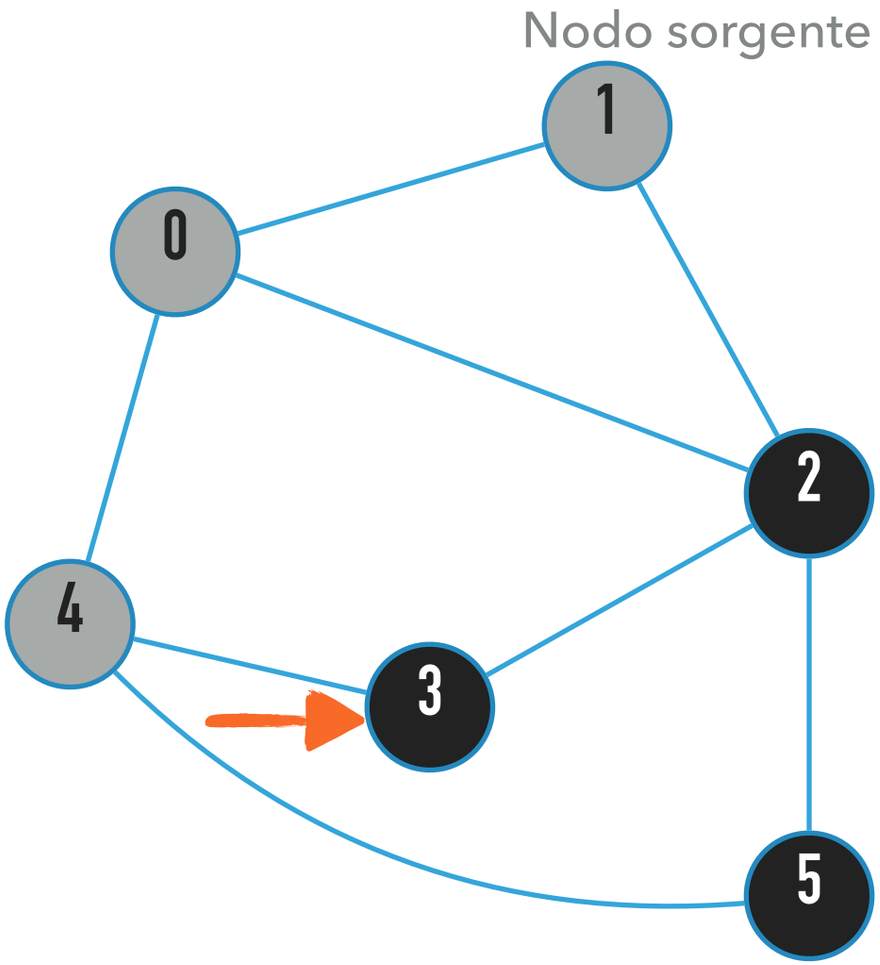

Stack di chiamate


Dato che non possiamo più effettuare chiamate ricorsive, abbiamo finito di visitare il nodo corrente. aggiorniamo colore, tempo di fine e ritorniamo dalla chiamata ricorsiva

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | 2 | 1 | 5 | 4 | 3 | 6 |
| Tempo_fine | | | 8 | | | 7 |
| Predecessore | 1 | - | 3 | 4 | 0 | 2 |

| Tempo | 8 |
|---|---|

# ESEMPIO DI ESECUZIONE




Tempo | 9

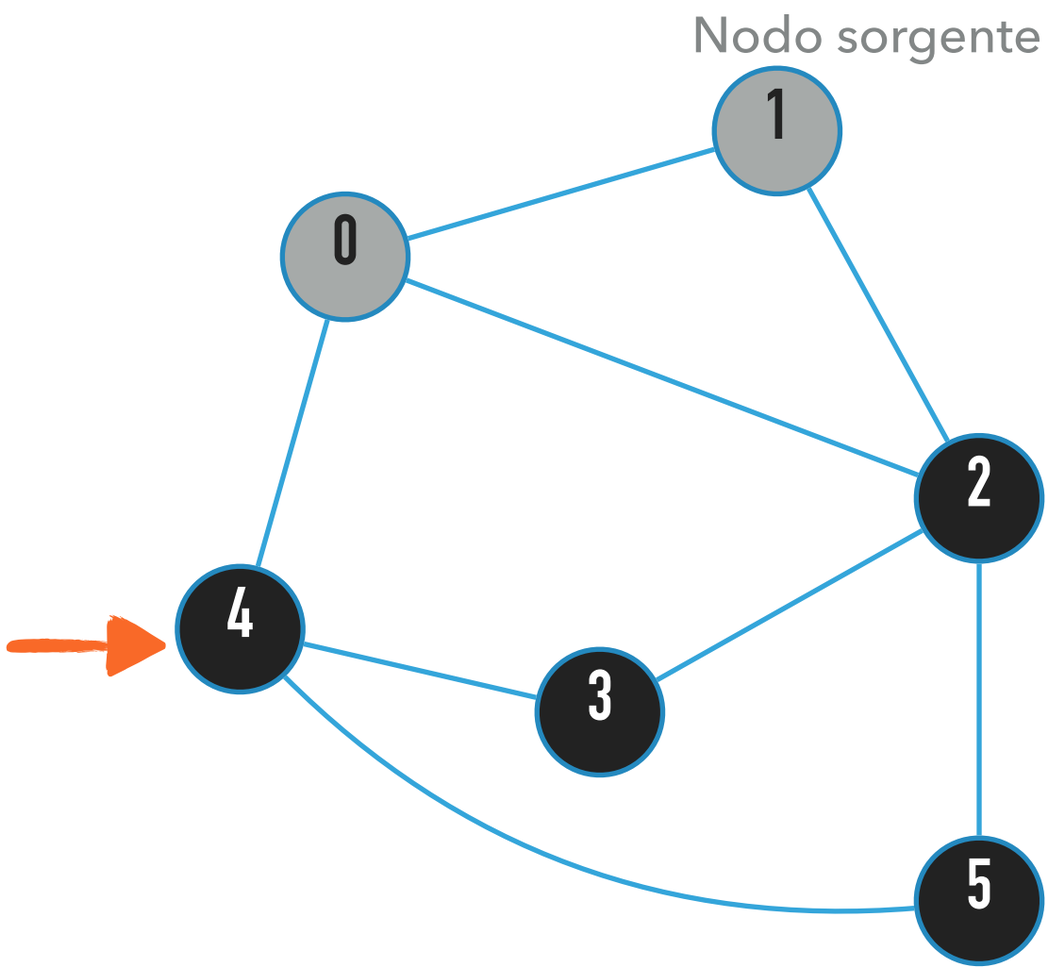
Stack di chiamate

- NODO: 3
- NODO: 4
- NODO: 0
- NODO: 1

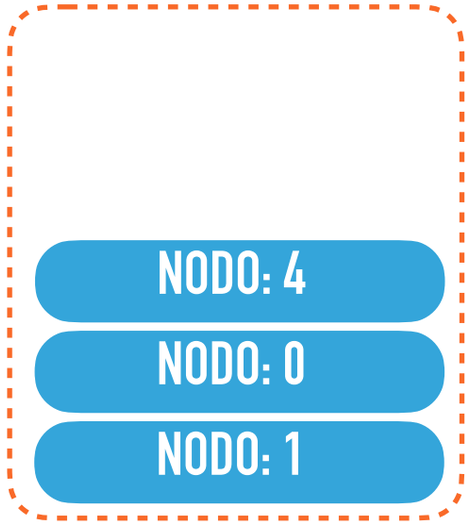
Dato che non possiamo più effettuare chiamate ricorsive, abbiamo finito di visitare il nodo corrente. aggiorniamo colore, tempo di fine e ritorniamo dalla chiamata ricorsiva

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | 2 | 1 | 5 | 4 | 3 | 6 |
| Tempo_fine | | | 8 | 9 | | 7 |
| Predecessore | 1 | - | 3 | 4 | 0 | 2 |

# ESEMPIO DI ESECUZIONE




Stack di chiamate




Dato che non possiamo più effettuare chiamate ricorsive, abbiamo finito di visitare il nodo corrente. aggiorniamo colore, tempo di fine e ritorniamo dalla chiamata ricorsiva

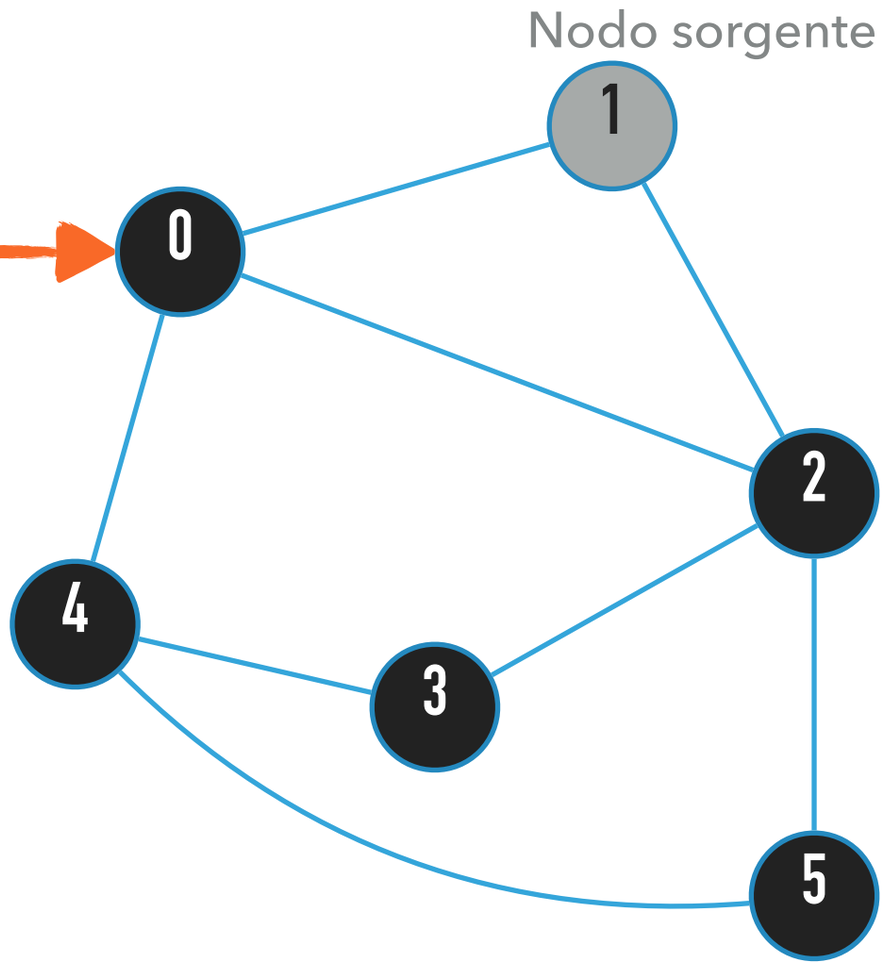
| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | 2 | 1 | 5 | 4 | 3 | 6 |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_fine | | | 8 | 9 | 10 | 7 |

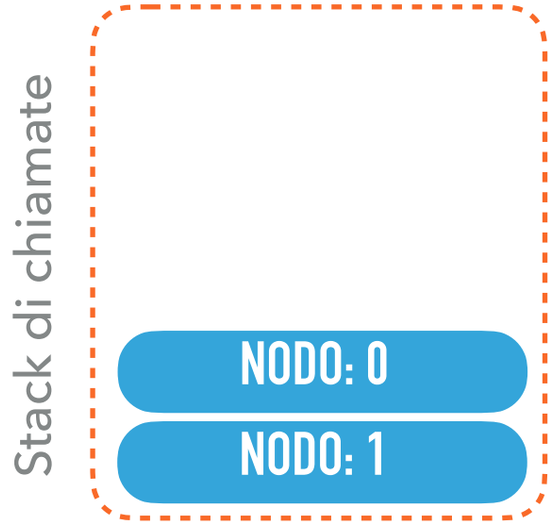
| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Predecessore | 1 | - | 3 | 4 | 0 | 2 |

| Tempo | 10 |
|---|---|

# ESEMPIO DI ESECUZIONE




Tempo 11



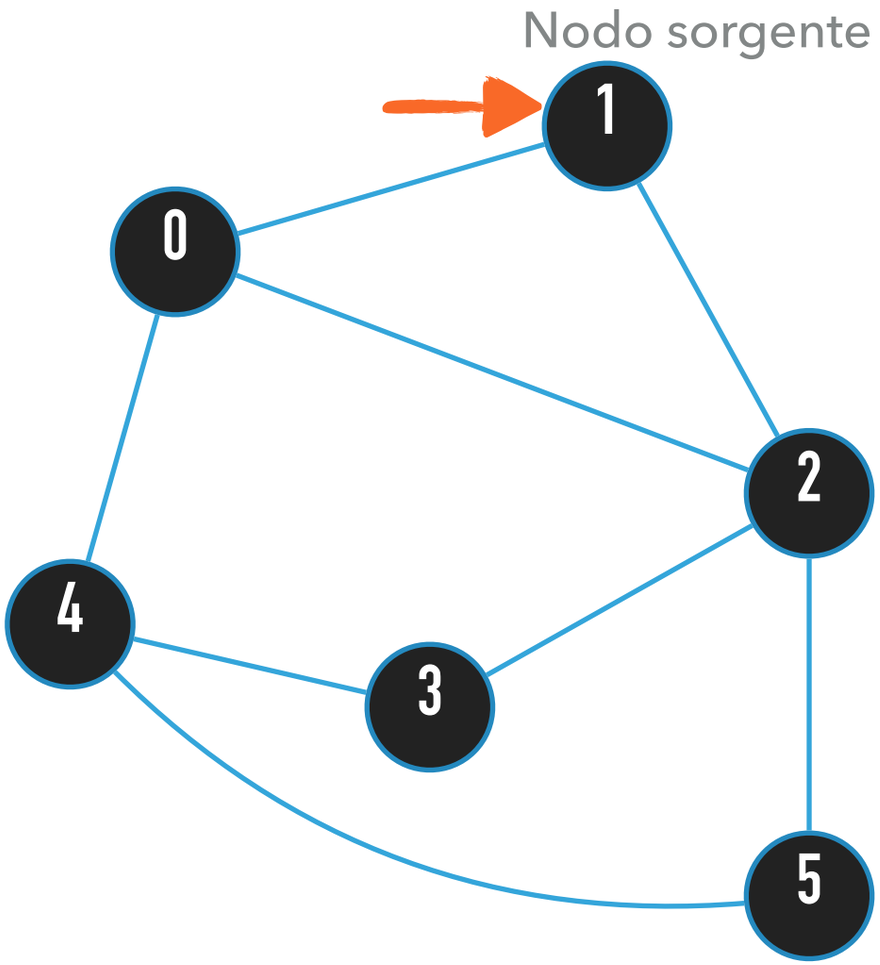

Dato che non possiamo più effettuare chiamate ricorsive, abbiamo finito di visitare il nodo corrente. aggiorniamo colore, tempo di fine e ritorniamo dalla chiamata ricorsiva

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | 2 | 1 | 5 | 4 | 3 | 6 |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_fine | 11 | | 8 | 9 | 10 | 7 |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Predecessore | 1 | - | 3 | 4 | 0 | 2 |

# ESEMPIO DI ESECUZIONE




Stack di chiamate

NODO: 1

Dato che non possiamo più effettuare chiamate ricorsive, abbiamo finito di visitare il nodo corrente. aggiorniamo colore, tempo di fine e ritorniamo dalla chiamata ricorsiva

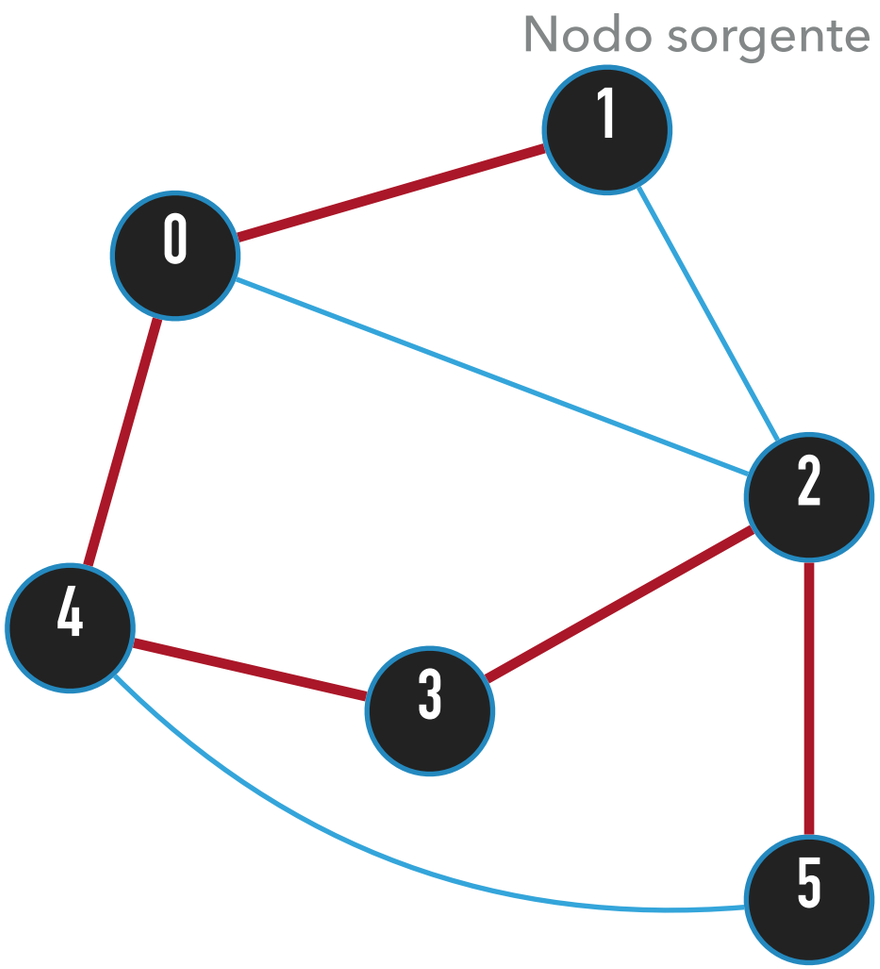
| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | 2 | 1 | 5 | 4 | 3 | 6 |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_fine | 11 | 12 | 8 | 9 | 10 | 7 |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Predecessore | 1 | - | 3 | 4 | 0 | 2 |

| Tempo | 12 |
|---|---|

# RISULTATI




Abbiamo ottenuto un albero DFS. Notate come sia diverso dall'albero BFS e come i percorsi su di esso non rappresentino, in generale, il percorso di lunghezza minima dal nodo di partenza

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_inizio | 2 | 1 | 5 | 4 | 3 | 6 |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Tempo_fine | 11 | 12 | 8 | 9 | 10 | 7 |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Predecessore | 1 | - | 3 | 4 | 0 | 2 |

# PROPRIETA DELLA VISITA IN PROFONDITA'

# ANALISI DELLA COMPLESSITÀ

- Notiamo che DFS-VISIT viene chiamata al più una volta per ogni nodo, dato che la chiamata avviene solo se il nodo viene visitato per la prima volta (era bianco) e viene immediatamente ricolorato di grigio, quindi $\Theta(V)$ chiamate a DFS-VISIT
- Su **tutte** le chiamate a DFS-VISIT, il ciclo che itera sui nodi adiacenti viene eseguito in totale $\Theta(E)$ volte
- Otteniamo quindi una complessità di $\Theta(V + E)$

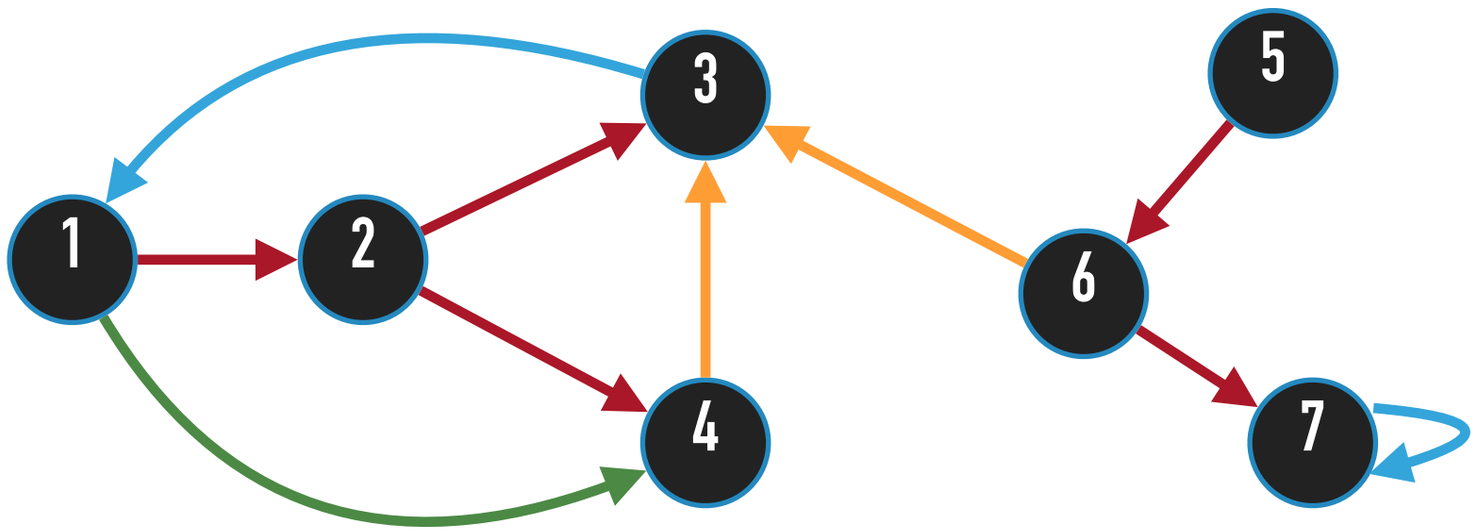
# TEMPI DI INIZIO E FINE VISITA: TEOREMA DELLE PARENTESI

- Sia G=(V,E). Dopo DFS, $\forall u, v \in V$, vale una ed una sola di:
  - $[d[u], f[u]] \cap [d[v], f[v]] = \varnothing$
  - $[d[u], f[u]] \subset [d[v], f[v]]$
  - $[d[v], f[v]] \subset [d[u], f[u]]$
- **Dimostrazione**: assumiamo $d[u] < d[v]$. Ci sono due casi:
  - $d[v] < f[u]$, ergo v è scoperto mentre u è grigio, e diventa discendente di u, e la sua visita finisce prima: $f[v] < f[u]$
  - $f[u] < d[v]$, ergo siamo nel primo caso.
- **Corollario**: $[d[v], f[v]] \subset [d[u], f[u]]$ se e solo se v è discendente di u nel DFS-tree

# TEOREMA DEL CAMMINO BIANCO

- Sia G=(V,E) su cui eseguiamo DFS. Siano $u, v \in V$.
  v è discendente di u nel DFS-tree se e solo se all'istante d[u] esiste un cammino da u a v fatto solo di nodi bianchi.
- **Dimostrazione**:
  - $\Rightarrow$: Tutti i nodi w nel cammino nel DFS-tree da u a v sono tali che d[w] > d[u] ergo al tempo d[u] sono bianchi.
  - $\Leftarrow$: Per assurdo, dato un cammino di nodi bianchi da u, sia v il primo nodo a non essere discendente di u in tale cammino, e w il predecessore. Allora w è discendente di u, e v sarà visitato prima che w diventi nero. Quindi f[v] < f[w] < f[u] e d[v] > d[w] > d[u]. Quindi v è discendente di u.

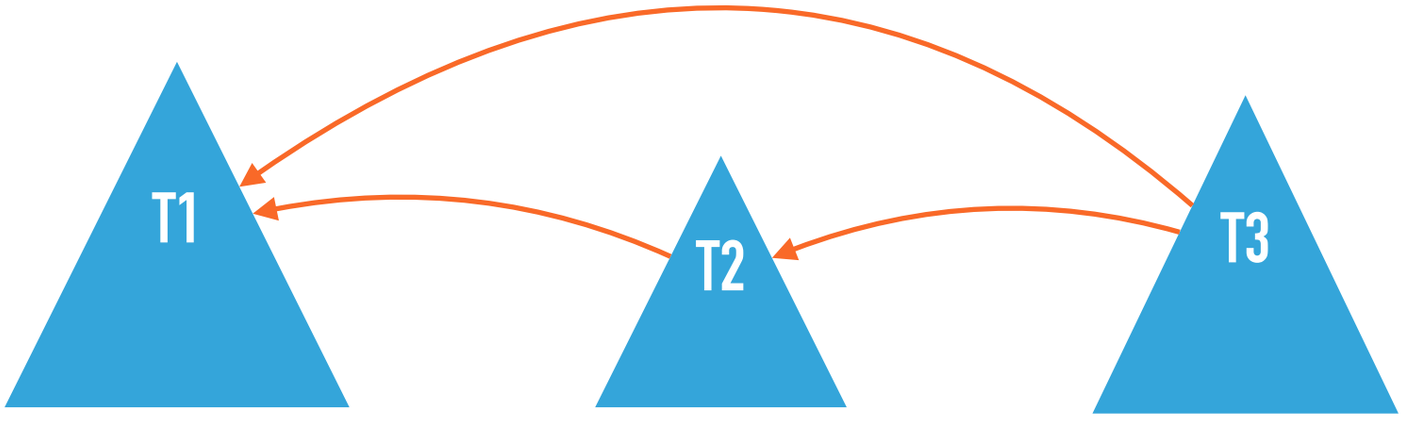
# CLASSIFICAZIONE DEGLI ARCHI




- **Archi dell'albero** (u,v): color[v]=bianco durante l'esame di (u,v)
- **Archi all'indietro** (u,v): connettono un vertice u ad un suo antenato v nell'albero - color[v]=grigio durante DFS-VISIT(u)
- **Archi in avanti** (u,v): color[v]=nero e d[v] >d[u], quindi v è discendente di u nell'albero DFS.
- **Archi di attraversamento** (u,v): color[v]=nero e d[v]<d[u]. L'arco collega parti diverse dell'albero o alberi DFS diversi della foresta DFS.

# FORESTA DFS

- L'esecuzione di DFS può generare un certo numero di alberi $T_j$ disgiunti (una per ogni chiamata di DFS-VISIT da DFS).
- Gli archi di attraversamento possono andare solo da $T_j$ a $T_i$ per i < j.



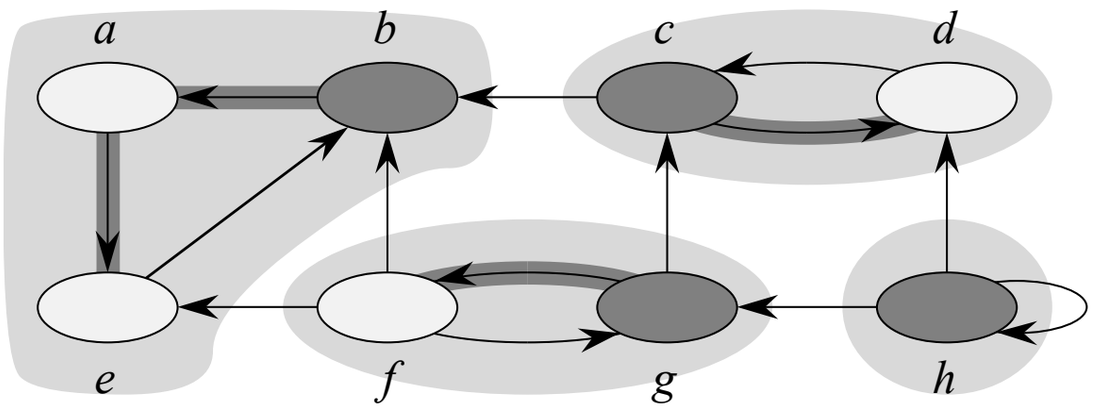
# AMPIEZZA VS PROFONDITÀ

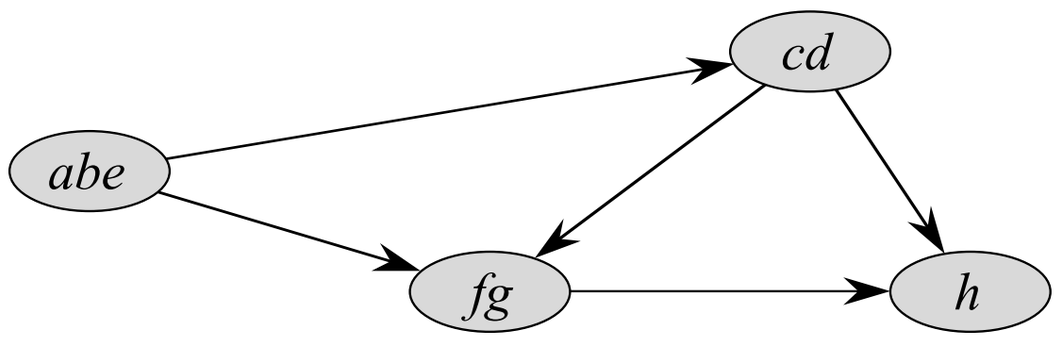
- La scelta di una o dell'altra tipologia di visita dipende dalle specifiche dell'algoritmo
- Per trovare il percorso di lunghezza minima? BFS
- Spesso per grafi diretti si utilizza DFS
- Notate come in BFS la coda sia esplicita, mentre in DFS lo stack è implicitamente fornito dalle chiamate ricorsive

# COMPONENTI FORTEMENTE CONNESSE

# COMPONENTI FORTEMENTE CONNESSE (GRAFI ORIENTATI)

- Diciamo che v è raggiungibile da u se esiste un cammino da u a v: $u \longrightarrow v$.
- Se u è raggiungibile da v e v è raggiungibile da u, diciamo che u e v sono **mutualmente raggiungibili**: $u \longleftrightarrow v$ sse $u \longrightarrow v$ e $v \longrightarrow u$
- $u \longleftrightarrow v$ è una relazione di equivalenza. Le sue classi di equivalenza sono le **componenti fortemente connesse** del grafo.
- Il grafo delle componenti fortemente connesse (scc-graph) ha un nodo per ogni scc di G, ed un arco tra $C_1$ e $C_2$ sse c'è un arco in G da un nodo u di $C_1$ ad un nodo u di $C_2$. Il **scc-graph** è aciclico. Perchè?



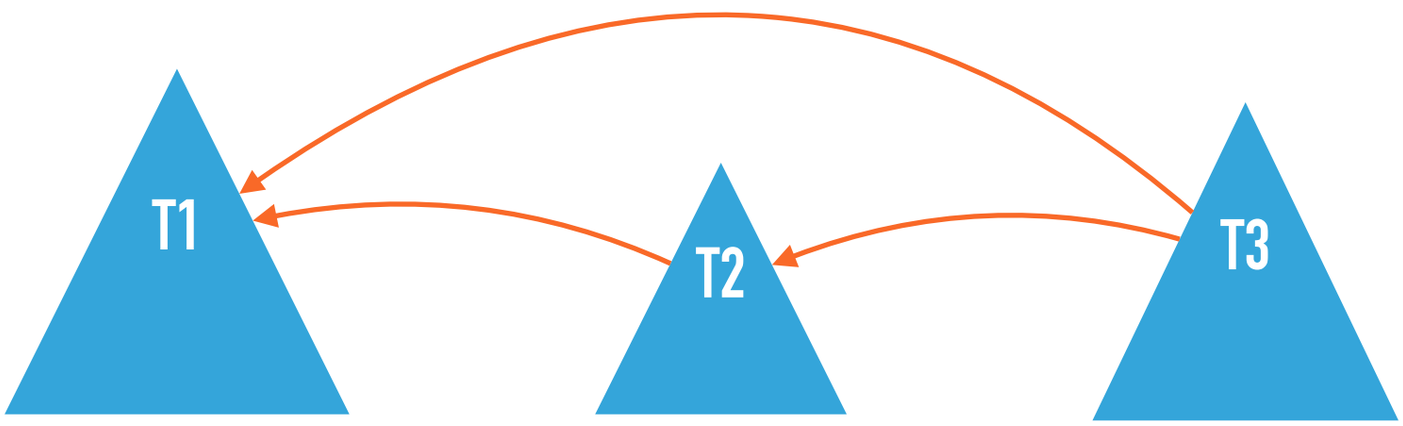



# COMPONENTI FORTEMENTE CONNESSE (GRAFI ORIENTATI)

- Il grafo trasposto di G=(V,E) è $G^T$=(V,$E^T$), con
  $E^T = \{(v,u) \mid (u,v) \in E\}$

- Se $u \longleftrightarrow v$ in G allora $u \longleftrightarrow v$ anche in $G^T$ . Le componenti fortemente connesse di G e $G^T$ sono le stesse.

- Ogni componente fortemente connessa è tutta contenuta in un singolo albero DFS $T_j$ di $G$ e in un singolo albero DFS $T_i$ di $G^T$.



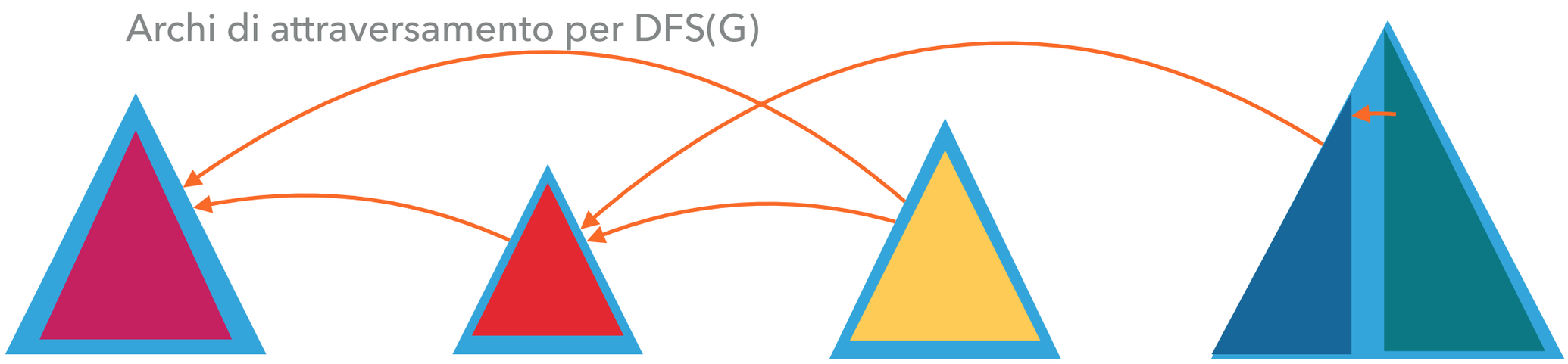
# COMPONENTI FORTEMENTE CONNESSE (GRAFI ORIENTATI)

- Siano $C_1$ e $C_2$ due componenti fortemente connesse, e sia $(u, v) \in E$ un arco di G, con $u \in C_1, v \in C_2$.
  Dopo l'esecuzione di DFS(G), vale f[$C_1$] > f[$C_2$], dove $f[C] = \max\{f[v] \mid v \in C\}$.
- Dimostrazione: si considera il caso in cui viene prima scoperta la componente $C_1$ (i nodi di $C_2$ diventano tutti discendenti del primo nodo di $C_1$ scoperto per teorema del cammino bianco) ed il caso in cui si scopre prima $C_2$ ($C_1$ è visitata quando tutti i nodi di $C_2$ sono neri).
- Siano $C_1$ e $C_2$ due componenti fortemente connesse, e sia $(u, v) \in E^T$ un arco di $G^T$, con $u \in C_1, v \in C_2$.
  Dopo l'esecuzione di DFS(G), vale f[$C_1$] < f[$C_2$].
- Se lancio DFS($G^T$) e parto prima dalla componente $C_2$ come sopra, la componente $C_2$ e la componente $C_1$ staranno in alberi DFS diversi della foresta DFS.

# ALGORITMO DI TARJAN PER LE COMPONENTI FORTEMENTE CONNESSE

- Tarjan-SCC($G$)
  DFS($G$), memorizza i tempi di fine visita f[v]
  calcola $G^T$
  DFS($G^T$), in ordine decrescente di f[v]
  **return** alberi DFS di $G^T$

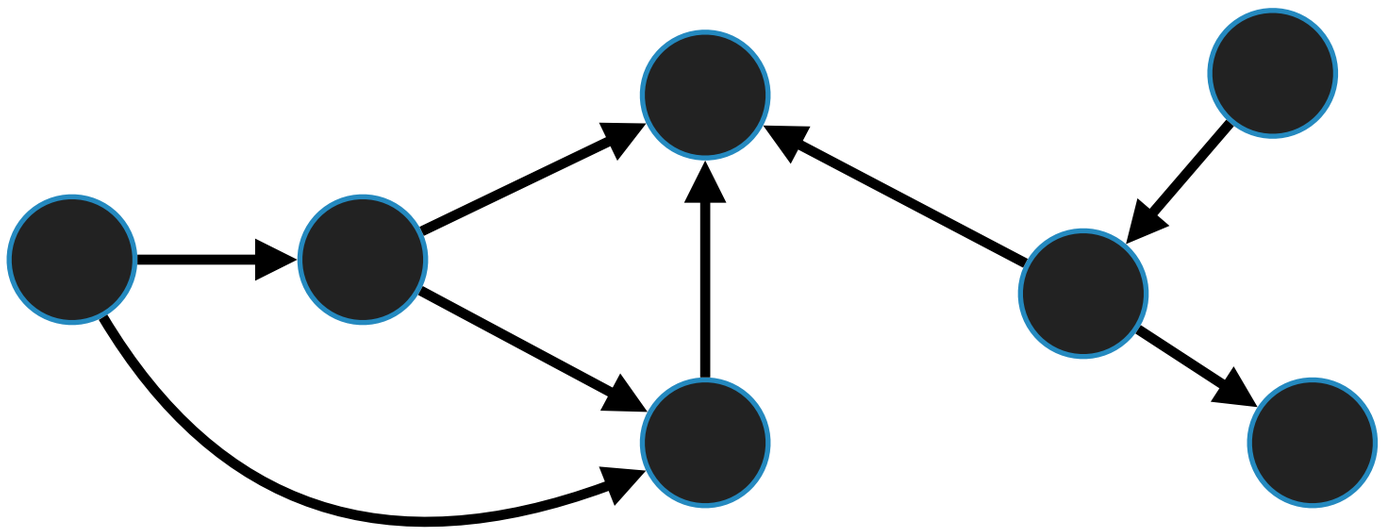
- Correttezza: per le proprietà del scc-graph, chiamando DFS su $G^T$ in ordine decrescente di f[v] di DFS($G$), le componenti connesse vengono scoperte dalle foglie alle radici nel scc-graph di $G^T$, e quindi ognuna finisce in un albero DFS diverso. Ma ogni albero contiene almeno una scc.



# ORDINAMENTO TOPOLOGICO

# GRAFI ORIENTATI ACICLICI (DAG)

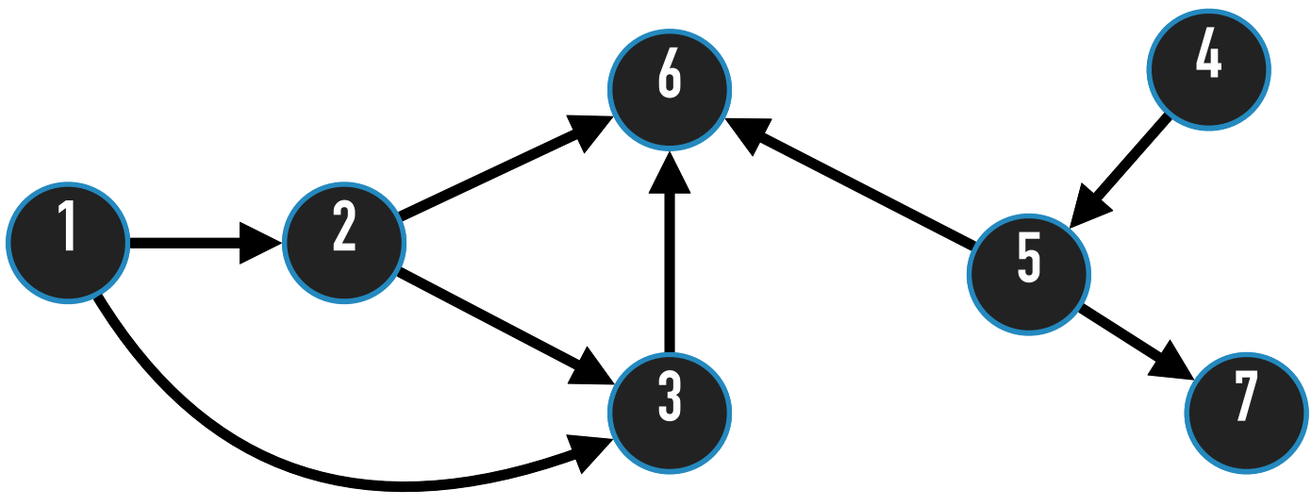
- Un grafo diretto è aciclico se non ha cicli (cammini semplici che iniziano e terminano nello stesso vertice). Tutti gli alberi sono aciclici, ma ci sono grafi aciclici che non sono alberi.
- Teorema: un grafo diretto è aciclico sse DFS non produce archi all'indietro.
  - $\Rightarrow$: (contronominale) Sia (u,v) arco all'indietro. Quindi u è discendente di v nel DFS-tree. Otteniamo un ciclo.
- $\Leftarrow$ (contronominale) sia c un ciclo e v il primo vertice del ciclo scoperto, w il suo predecessore nel ciclo. Usare il teorema del cammino bianco per mostrare che si ottiene un arco all'indietro.

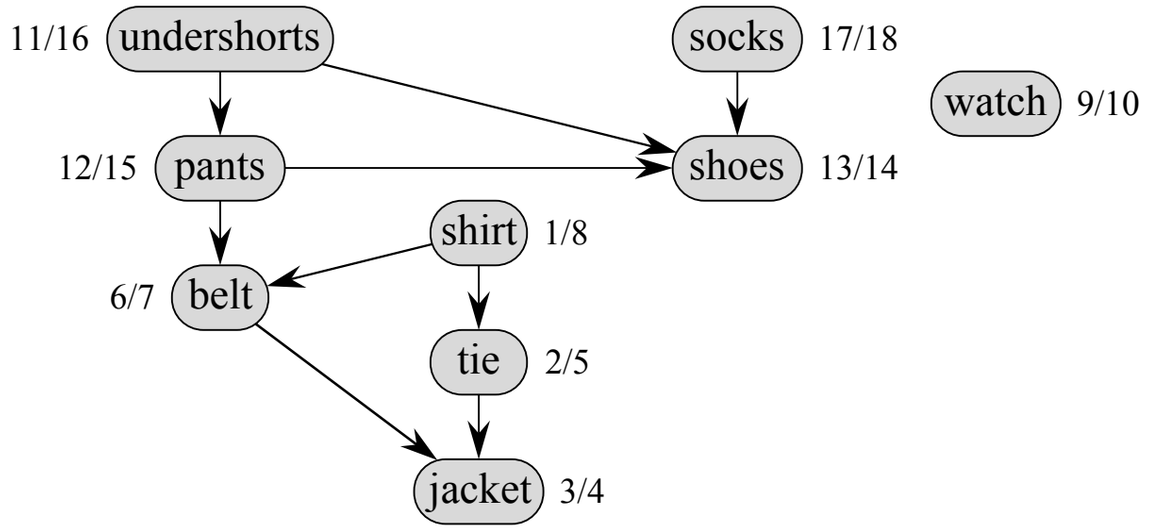


# ORDINAMENTO TOPOLOGICO

- Dato un grafo aciclico, un **ordinamento topologico** è un ordinamento dei vertici tale che, per ogni arco (u,v), u precede v nell'ordinamento.

- Topological-Sort(G)
  DFS(G)
  **return** lista dei vertici v di G in ordine decrescente di f[v]



# ORDINAMENTO TOPOLOGICO

- Topological-Sort(G)
  DFS(G)
  **return** lista dei vertici v di G in ordine decrescente di f[v]

- Correttezza: per ogni $(u, v) \in E$, mostriamo f[v] < f[u].

- Quando esaminiamo (u,v), v non può essere grigio (no archi all'indietro). Se v è bianco, diventa discendente di u e quindi f[v] < f[u], se v è nero, f[v] < d[u] < f[u].