RICERCA IN PROFONDITÀ

INFORMATICA

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, quando il nodo è stato visitato per la prima volta (quando diventa grigio)
  - Il tempo di fine visita, quando tutti i suoi vicini sono stati visitati (quando 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

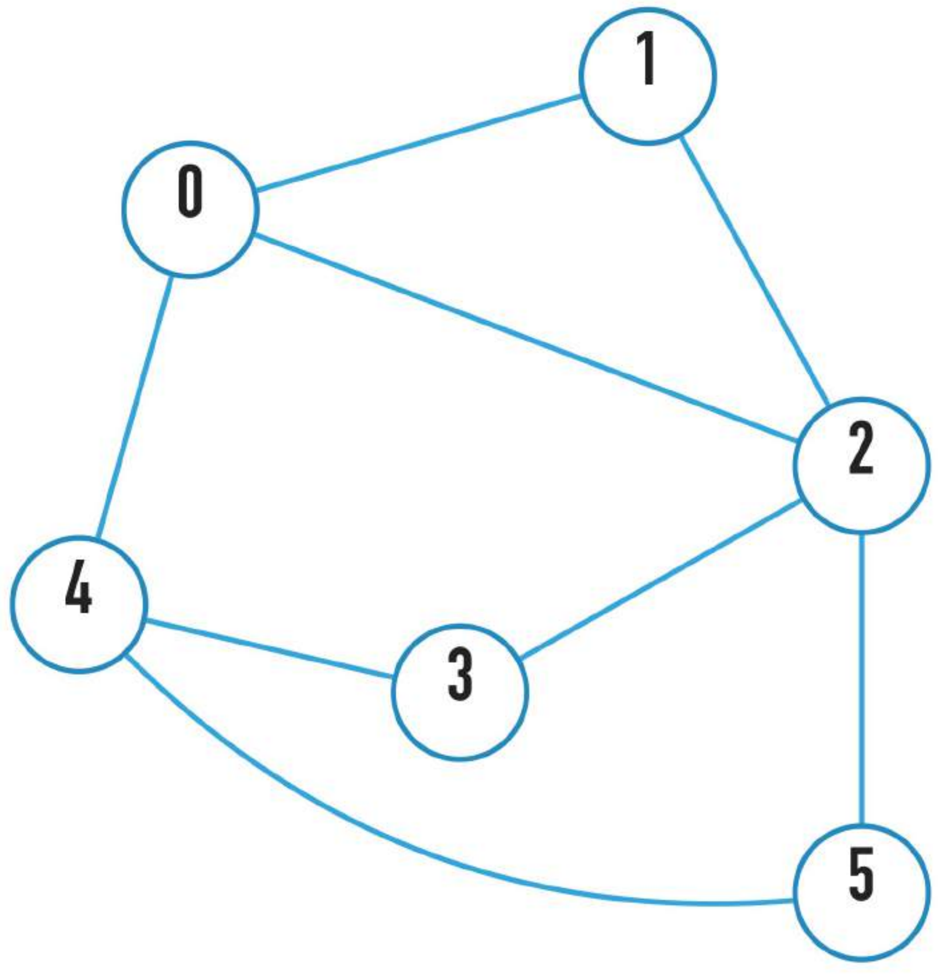
PSEUDOCODICE: INIZIALIZZAZIONE

```
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   d[u]
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   f[u]
```

ESEMPIO DI ESECUZIONE

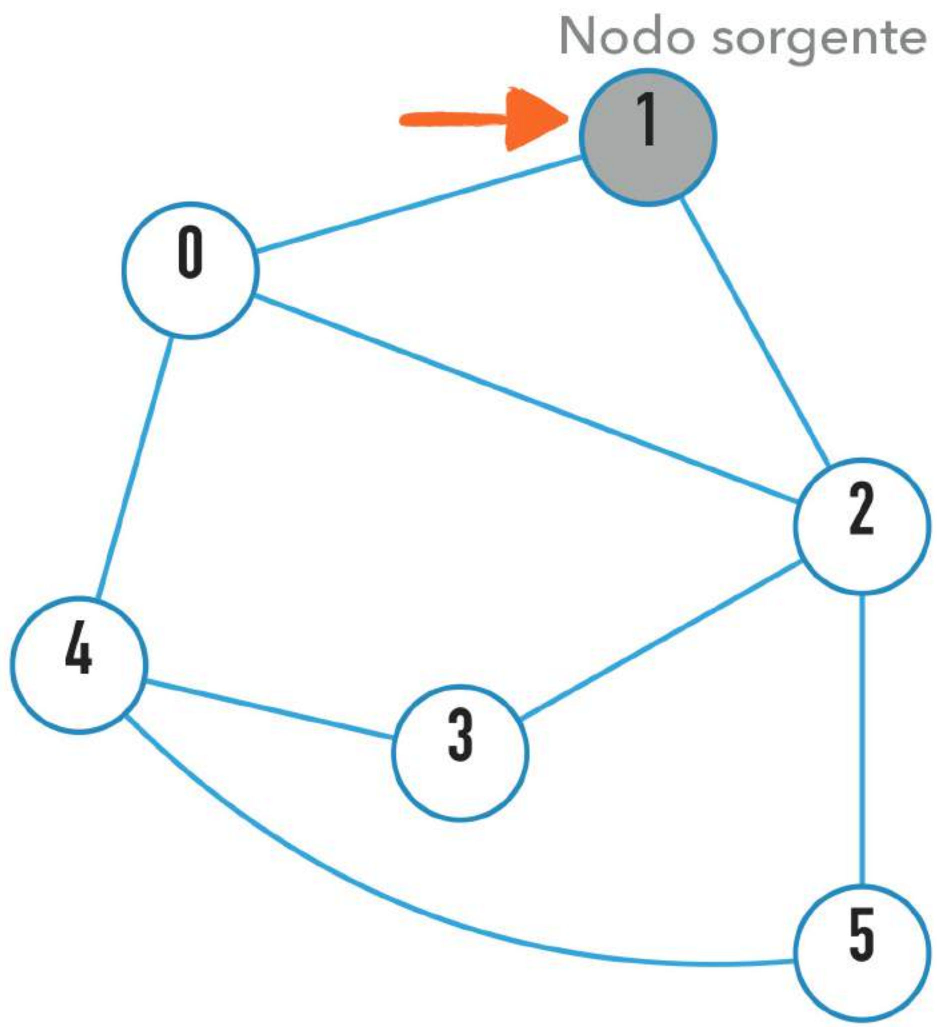


Tempo 0

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

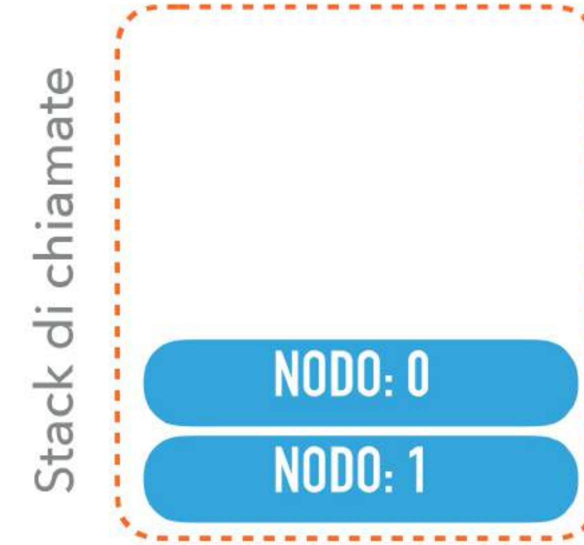
	0	1	2	3	4	5
Tempo_inizio		1				

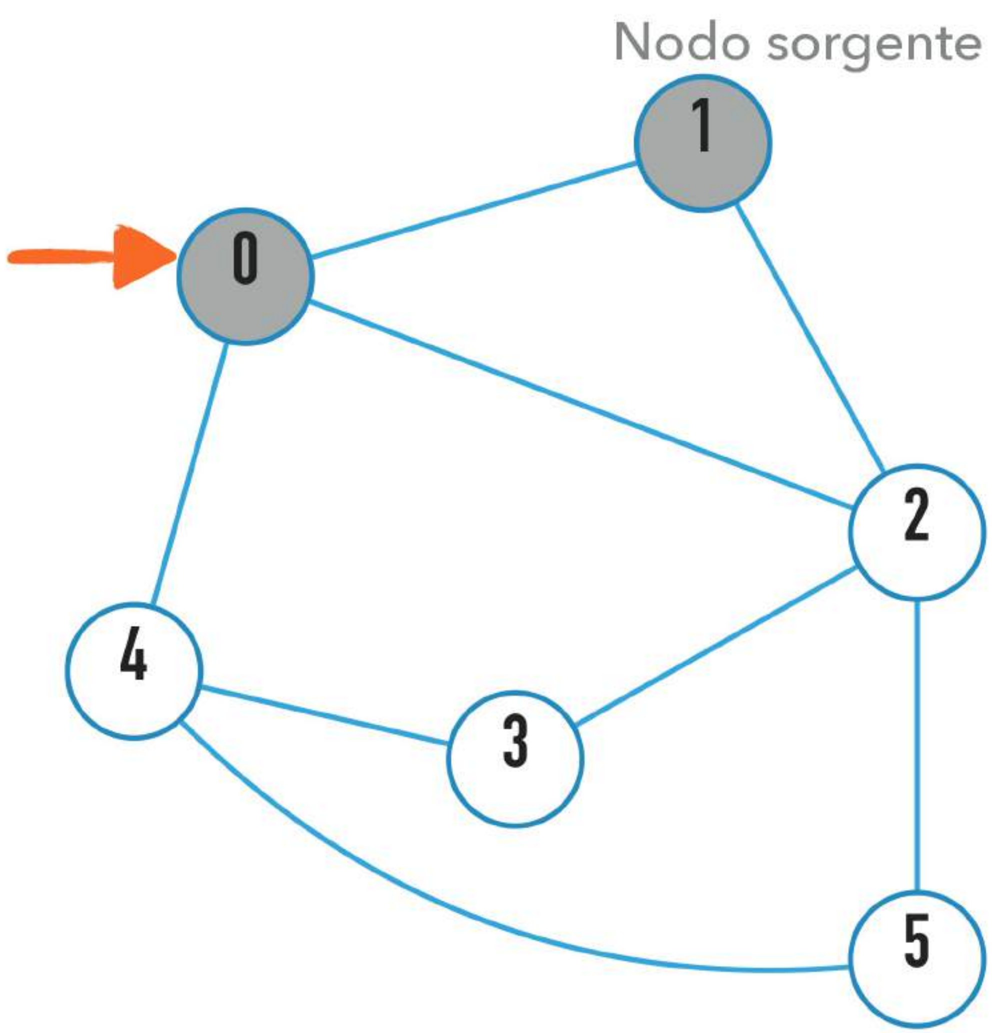
	0	1	2	3	4	5
Tempo_fine						

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

Tempo: 1

ESEMPIO DI ESECUZIONE





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

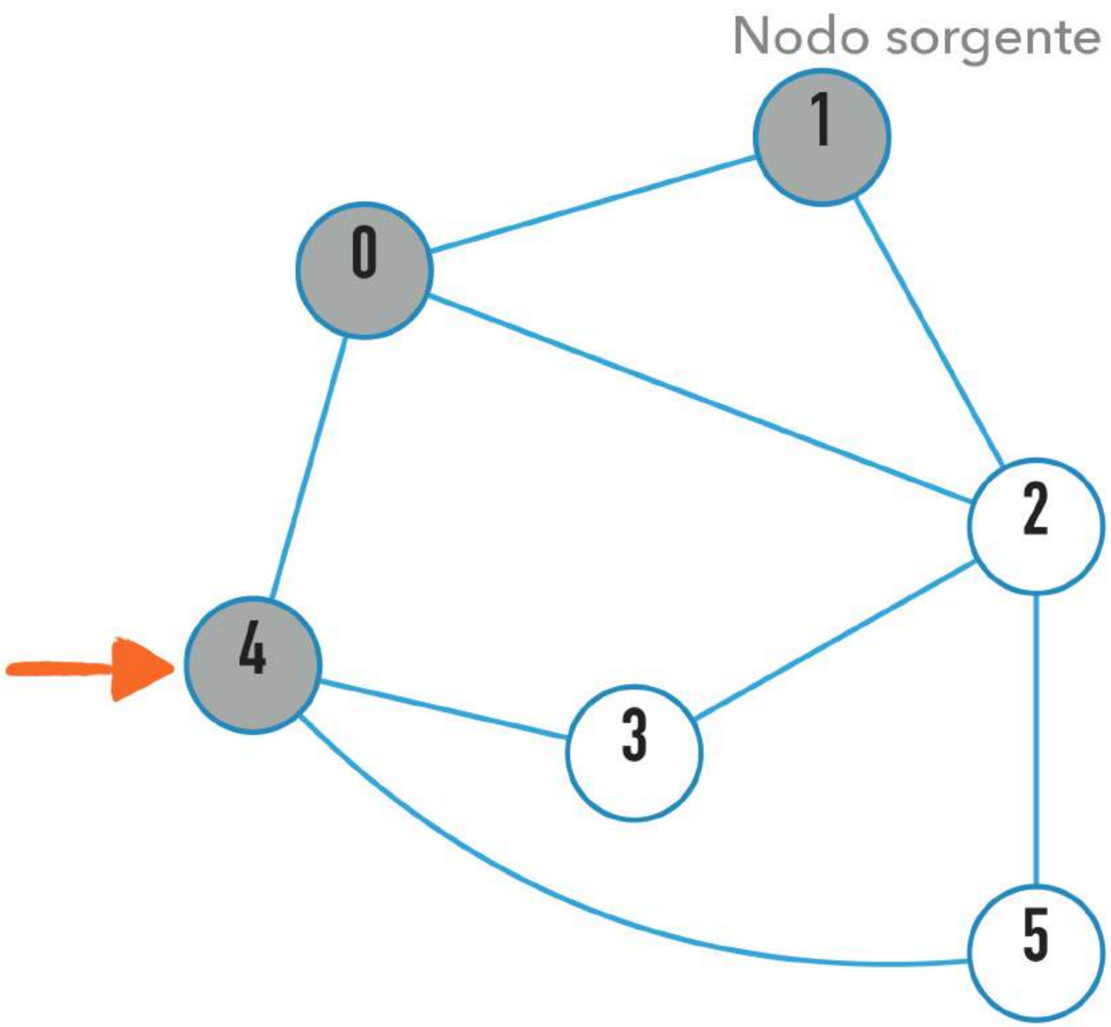
	0	1	2	3	4	5
Tempo_inizio	2	1				

	0	1	2	3	4	5
Tempo_fine						

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

Tempo 2

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						

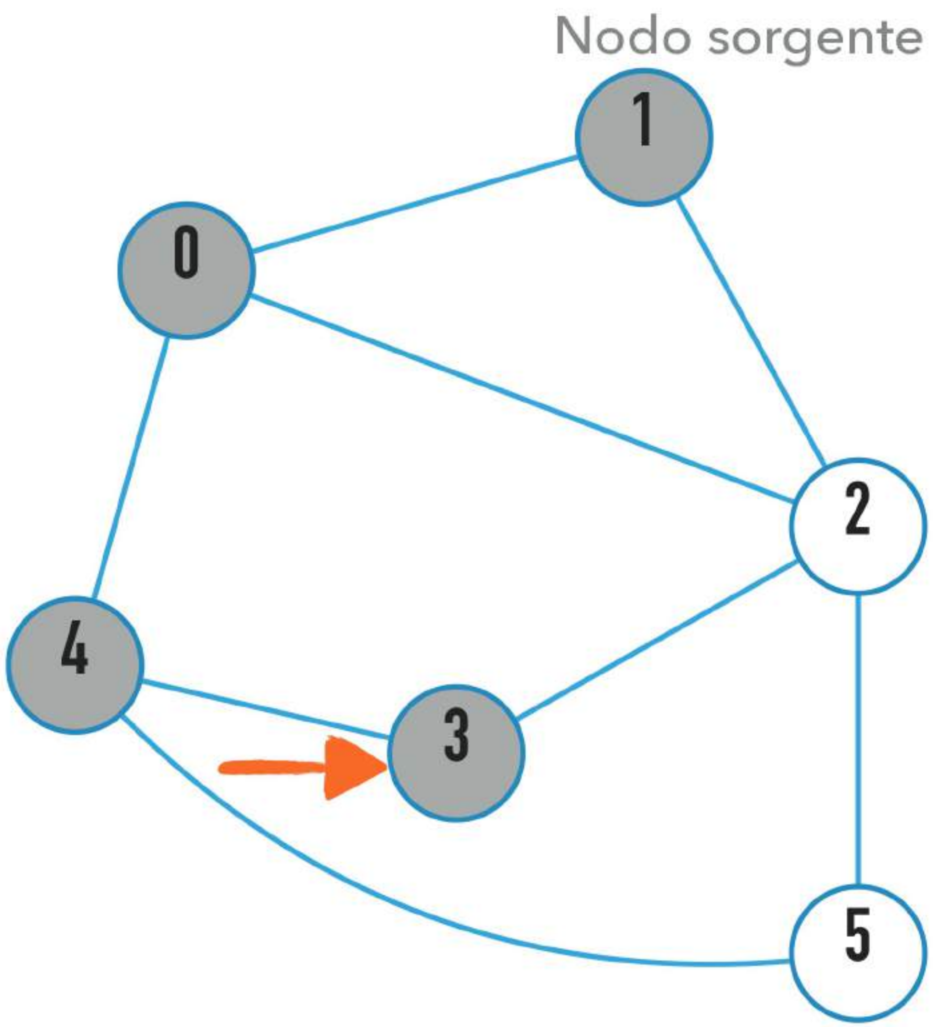
	0	1	2	3	4	5
Predecessore	1	-	-	-	0	-

Tempo 3

ESEMPIO DI ESECUZIONE

Stack di chiamate

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



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

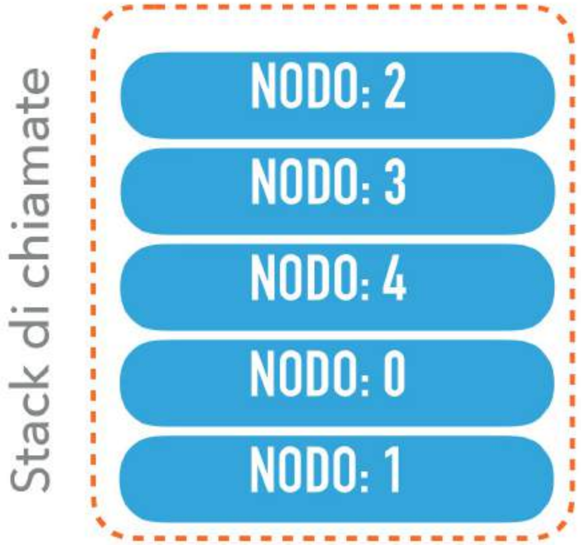
	0	1	2	3	4	5
Tempo_inizio	2	1		4	3	

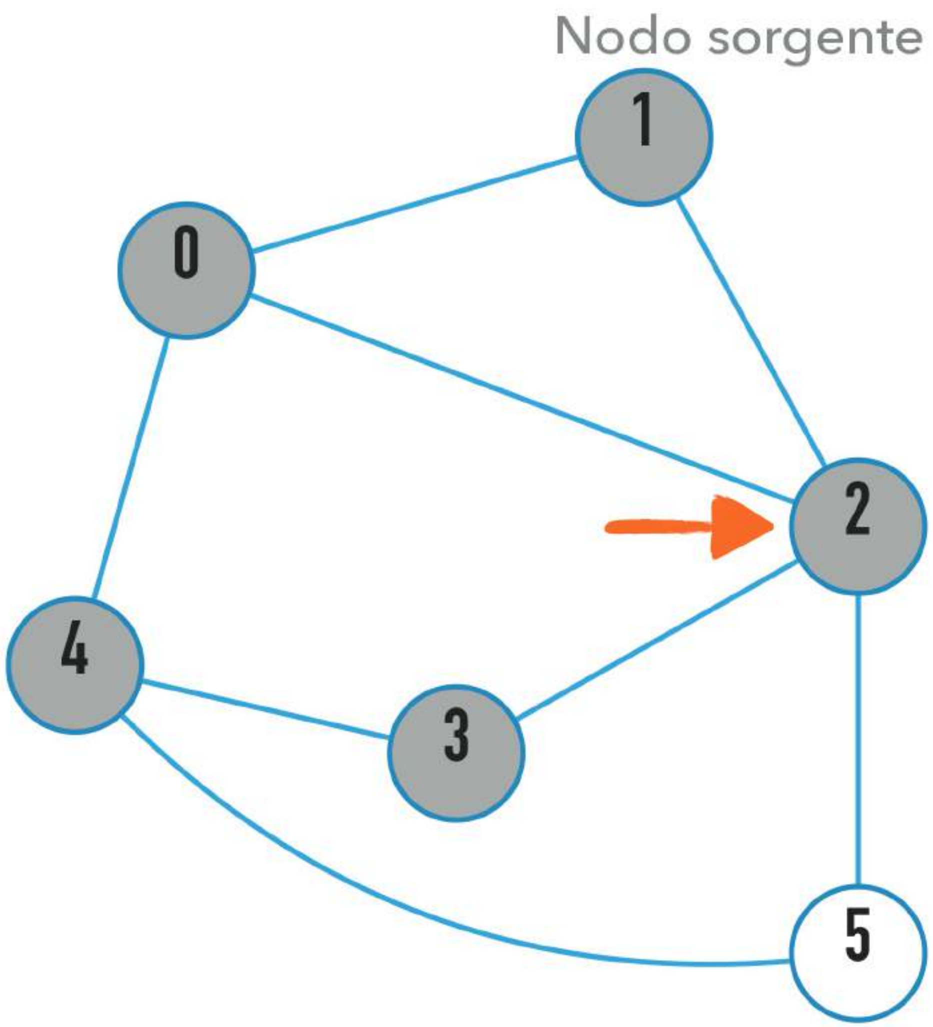
	0	1	2	3	4	5
Tempo_fine						

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

Tempo	4

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	5	4	3	

	0	1	2	3	4	5
Tempo_fine						

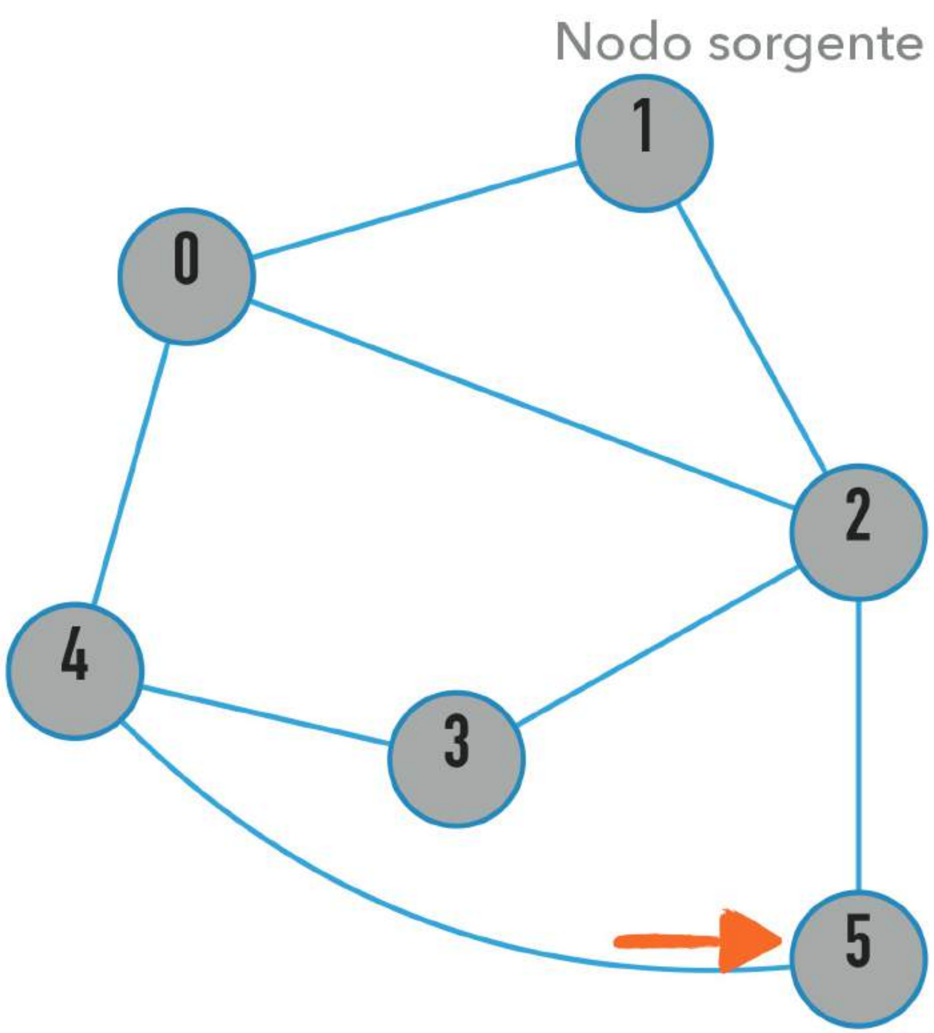
	0	1	2	3	4	5
Predecessore	1	-	3	4	0	-

Tempo 5

ESEMPIO DI ESECUZIONE

Stack di chiamate

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



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

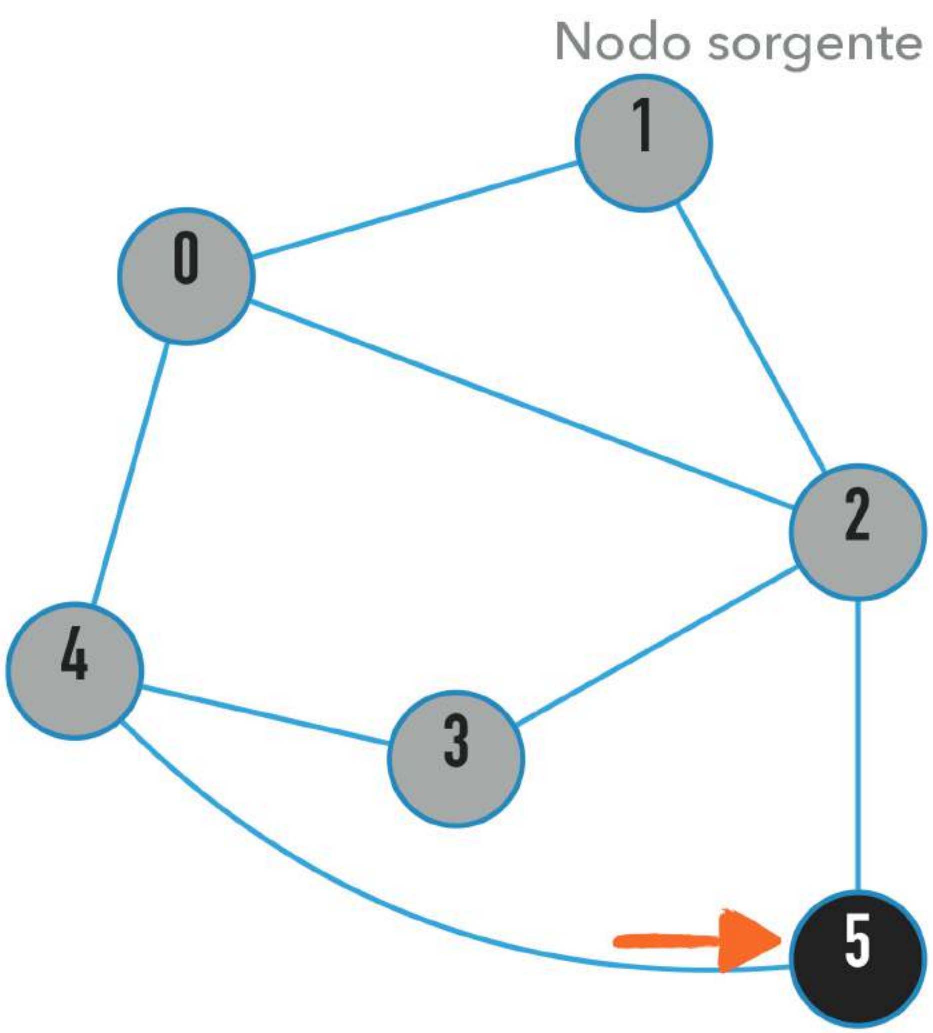
	0	1	2	3	4	5
Tempo_inizio	2	1	5	4	3	6

	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



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						7

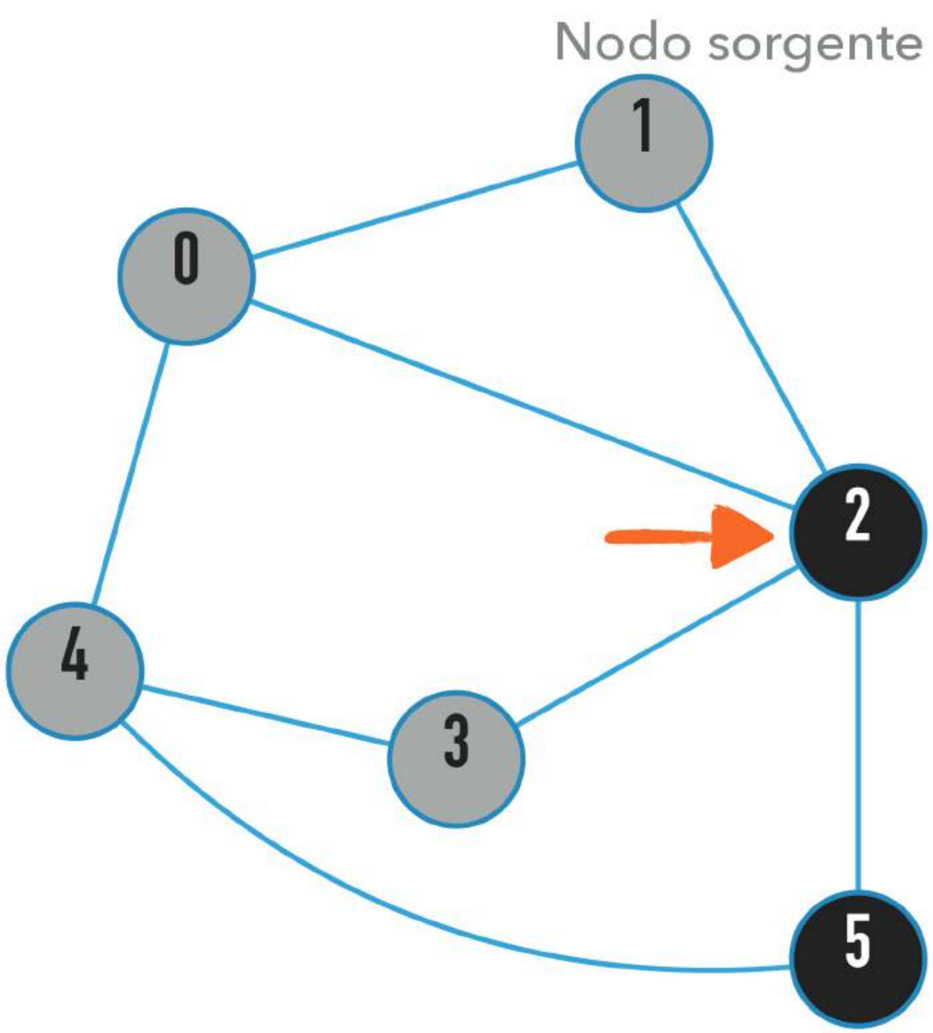
	0	1	2	3	4	5
Predecessore	1	-	3	4	0	2

Tempo	7

ESEMPIO DI ESECUZIONE

Stack di chiamate

- 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

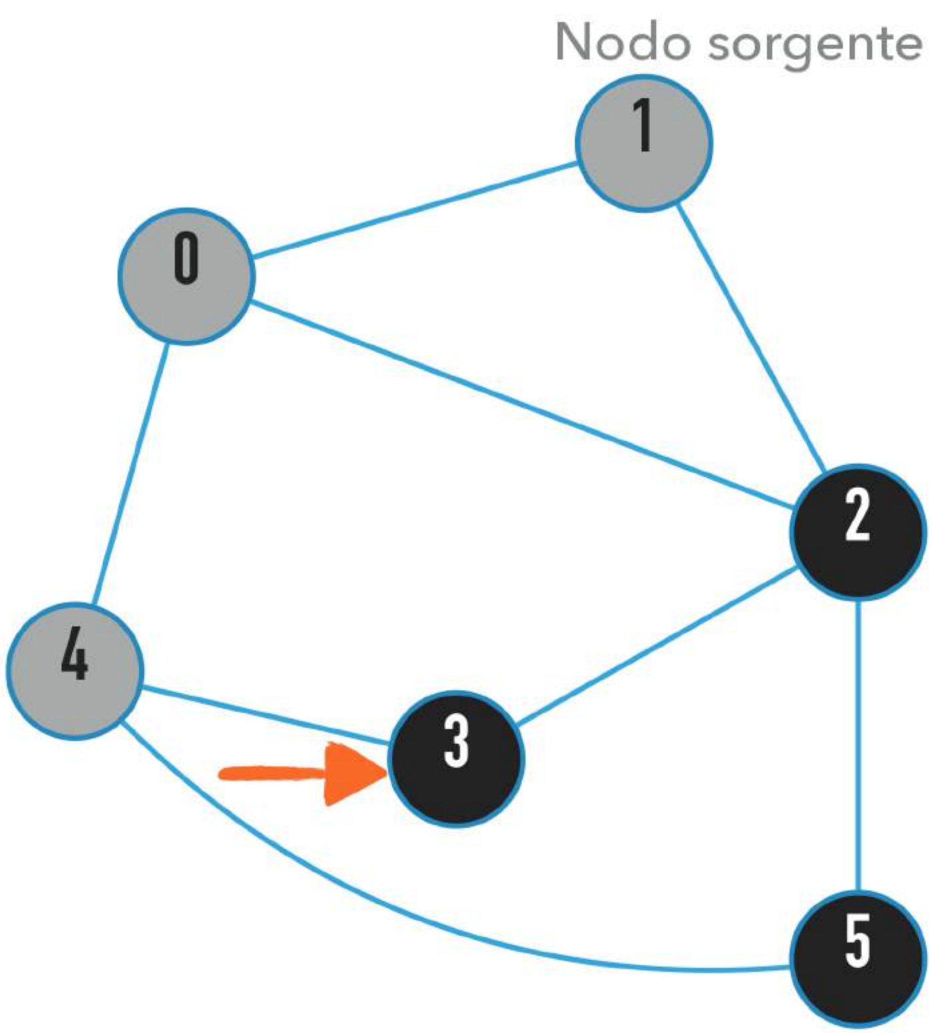
	0	1	2	3	4	5
Tempo_inizio	2	1	5	4	3	6

	0	1	2	3	4	5
Tempo_fine			8			7

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

Tempo 8

ESEMPIO DI ESECUZIONE



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

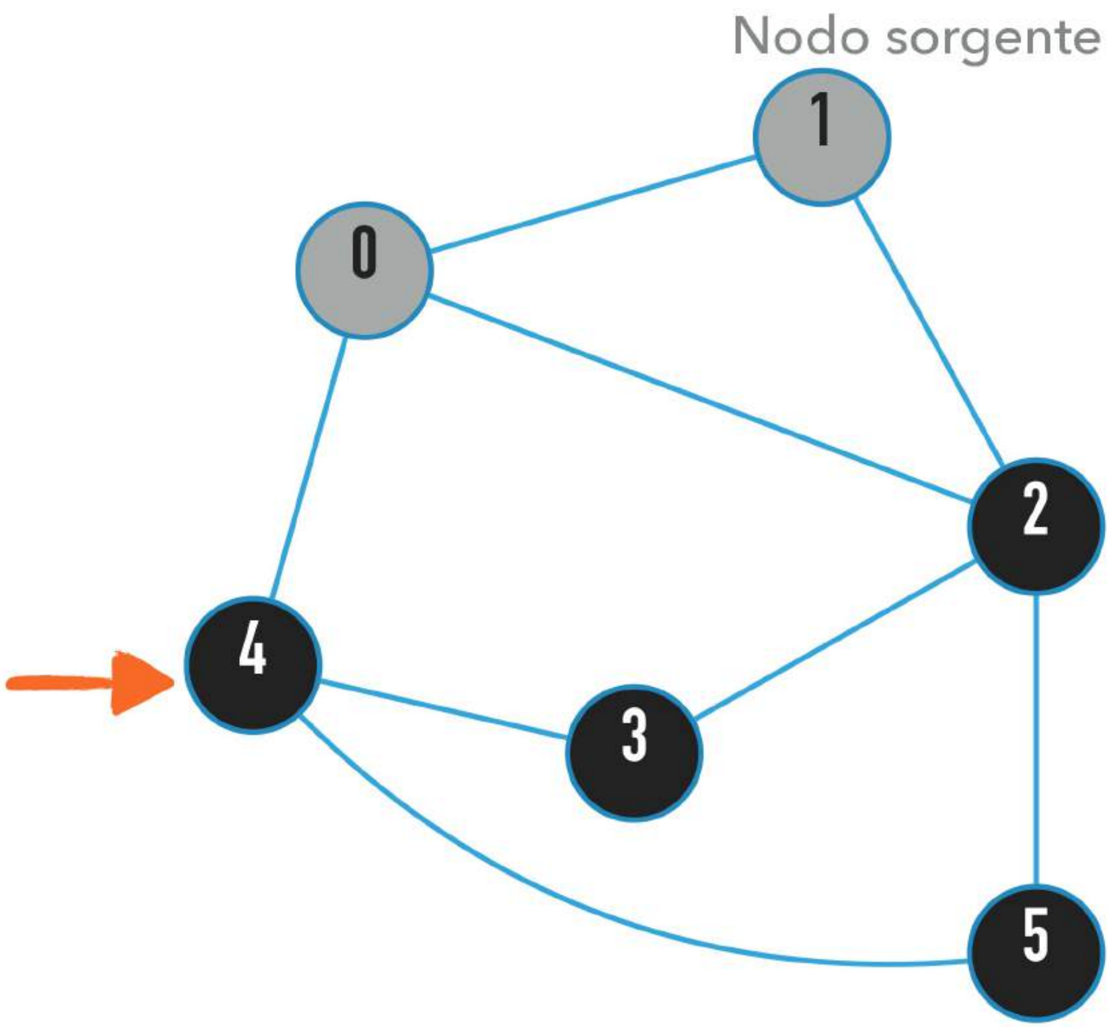
	0	1	2	3	4	5
Tempo_inizio	2	1	5	4	3	6

	0	1	2	3	4	5
Tempo_fine			8	9		7

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

Tempo	9

ESEMPIO DI ESECUZIONE



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

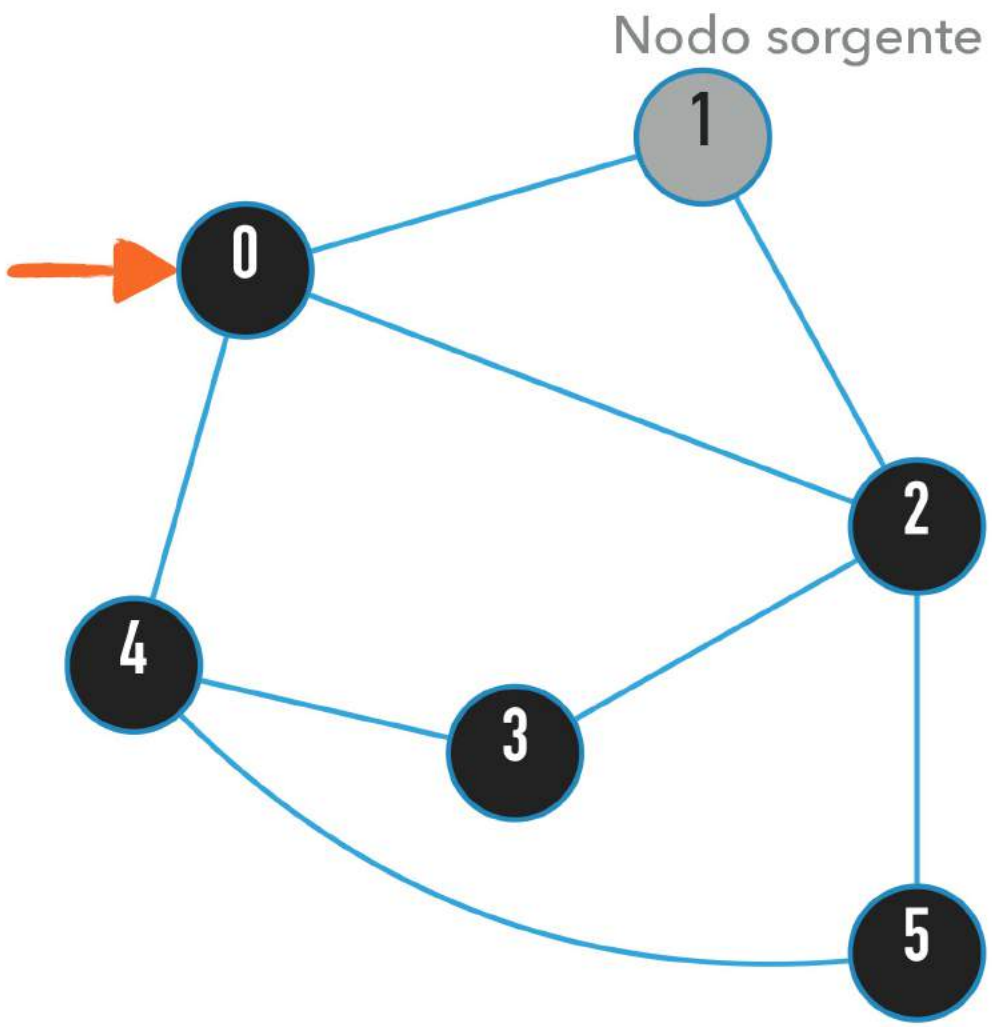
	0	1	2	3	4	5
Tempo_inizio	2	1	5	4	3	6

	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



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

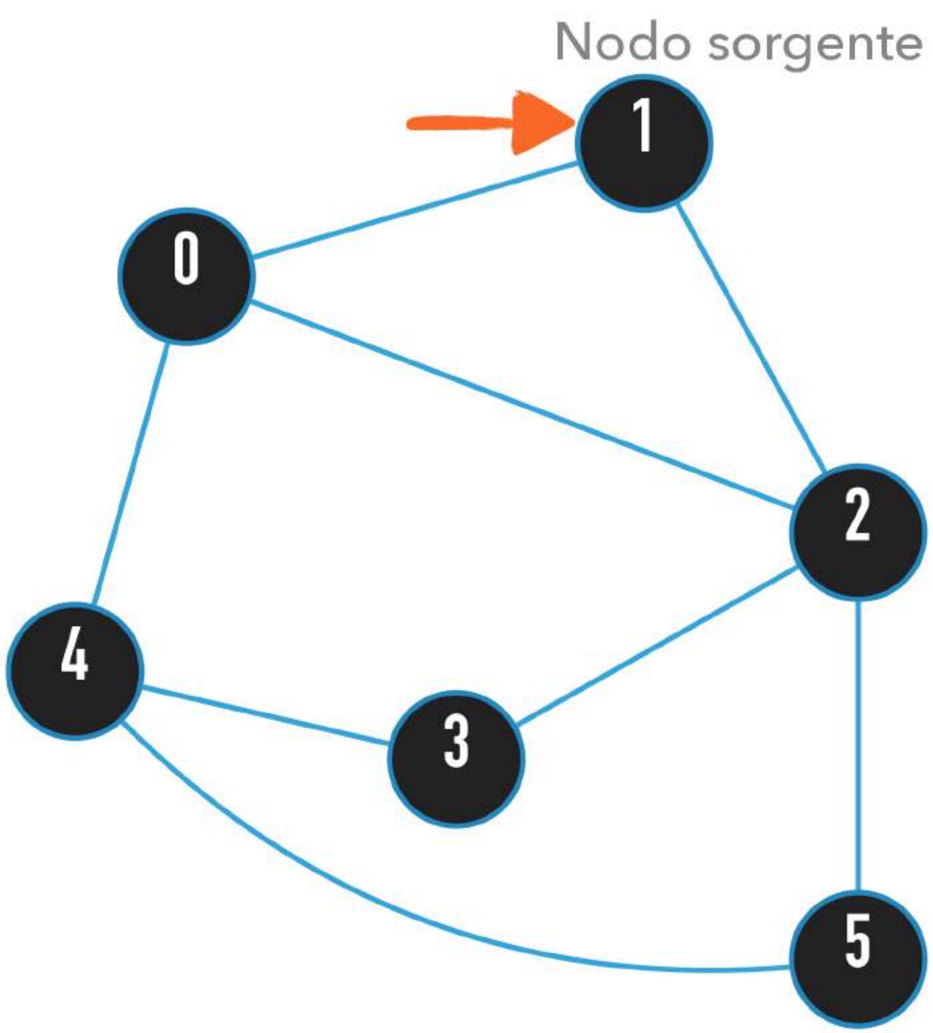
	0	1	2	3	4	5
Tempo_inizio	2	1	5	4	3	6

	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

Tempo	11

ESEMPIO DI ESECUZIONE

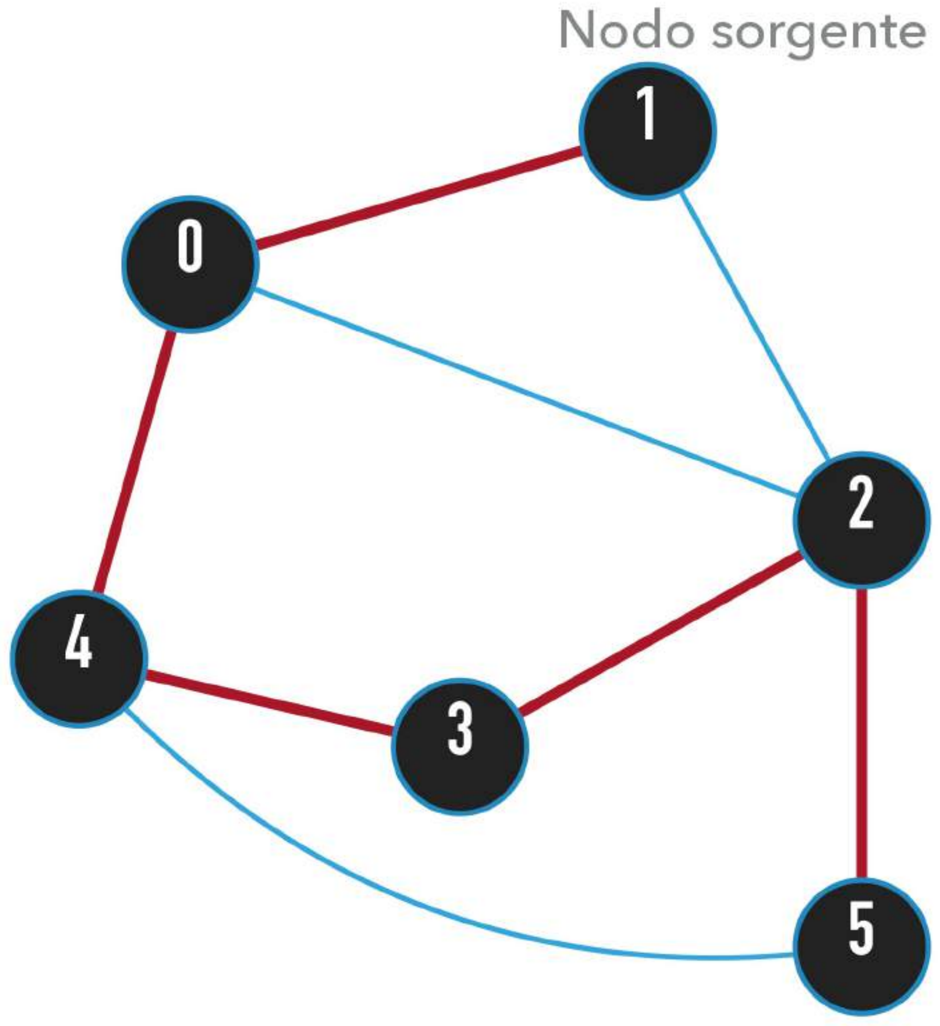


Dato che non possiamo più effettuare chiamate ricorsive, abbiamo finito di visitare il nodo corrente. aggiorniamo colore, tempo di fine e ritorniamo dalla chiamata ricorsiva

	0	1	2	3	4	5
Tempo_inizio	2	1	5	4	3	6
Tempo_fine	11	12	8	9	10	7
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

ANALISI DELLA COMPLESSITÀ

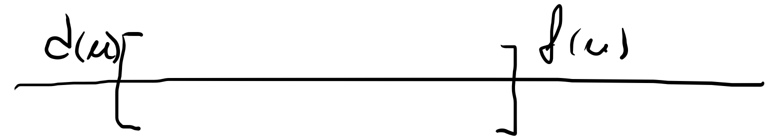
- Notiamo che DFS-VISIT viene chiamata al più una volta per ogni nodo, dato che la chiamata avviene solo se il nodo viene visitato per la prima volta (era bianco) e viene immediatamente ricolorato di grigio, quindi $\Theta(V)$ chiamate a DFS-VISIT
- Su **tutte** le chiamate a DFS-VISIT, il ciclo che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

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

DFS - PROPRIETÀ

TEOREMA delle PARENTESI



[[ ]]

] [[ ]]

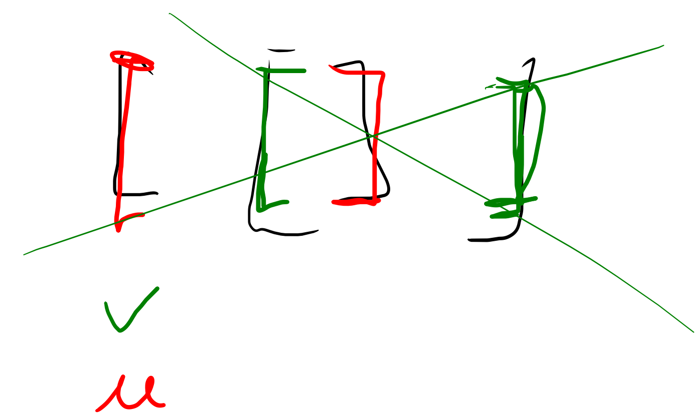
[ ]] [

Sia $G=(V,E)$. Dopo DFS, $\forall\, u,v \in V$ vale una ed una sola di

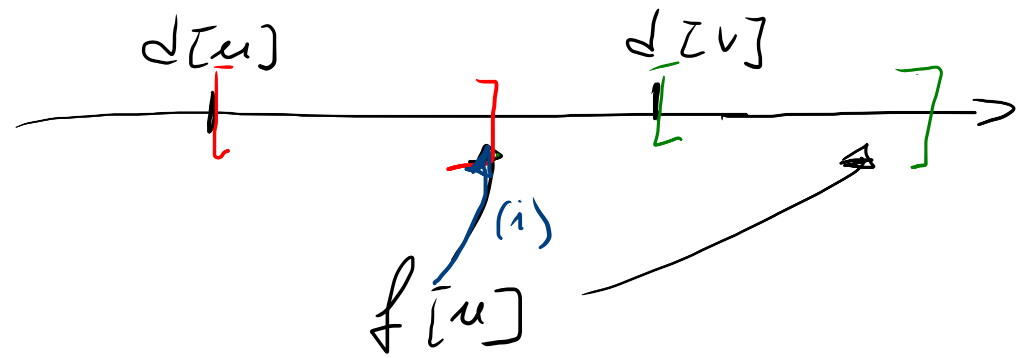
i) $[d[u], f[u]] \cap [d[v], f[v]] = \emptyset$ [ ] [ ]

ii) $[d[u], f[u]] \subseteq [d[v], f[v]]$ [[ ]]

iii) $[d[u], f[u]] \supset [d[v], f[v]]$ [[ ]]

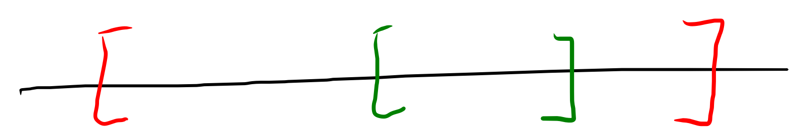


$d[u] < d[v]$



Se $f[u] < d[v] \Rightarrow$ i)

Se $f[v] > d[v] \Rightarrow \underbrace{f[v] < f[u]}_{\text{(ii)}}$



Corollario: Se $[d[v], f[v]] \subseteq [d[u], f[u]] \Rightarrow v$ è discendente di u in G_π

TEOREMA del CAMMINO BIANCO

Sia $G=(V,E)$ ed eseguiamo DFS(G). Siano $u,v \in V$

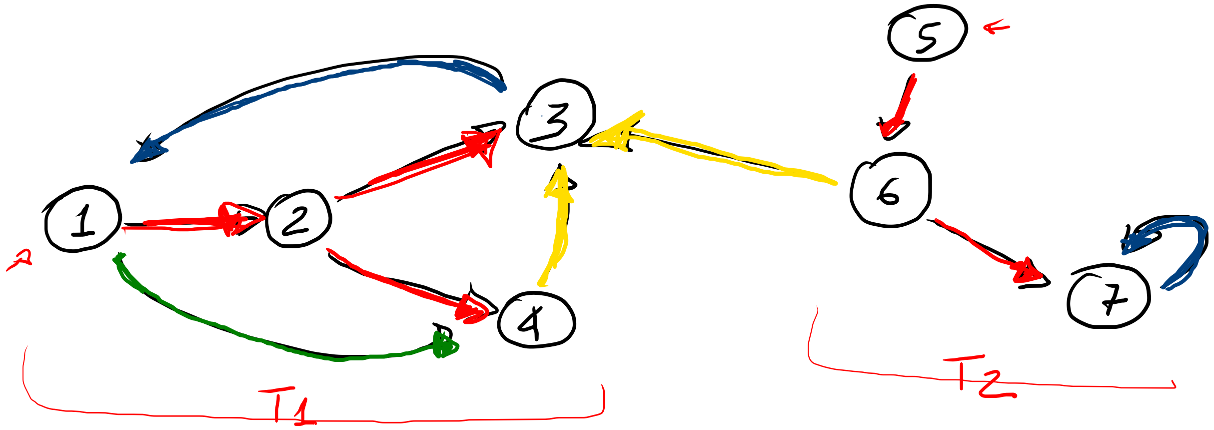
v è discendente di u in G_π <=> All'istante $d[u]$ esiste un cammino da u a v
fatto di soli nodi bianchi

"=>" in G_π $p: u \rightsquigarrow v$ => ogni vertice w di p è discendente di u,
$d[w] > d[u]$

"<=" (per assurdo) - esercizio

CLASSIFICAZIONE degli ARCHI

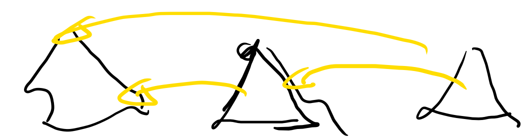
$G = (V, E)$ ORIENTATO



G_π

Siano T_i, T_j due alberi della foresta DFS.

Può esistere un arco da T_i a T_j se $i > j$



1) ARCHI DELL'ALBERO, sono in E_π $(u,v) \Rightarrow v$ è stato esplorato da u

2) ARCHI ALL'INDIETRO: connettono un vertice u ad un suo antenato v nell'albero DFS ($\Leftrightarrow$ COLOR[v] = GRIGIO quando (u,v) viene esaminato)

3) ARCO IN AVANTI: (u,v) t.c. COLOR[v] = NERO e $d[v] > d[u]$: v è discendente di u in G_π

4) ARCO DI ATTRAVERSAMENTO: (u,v) t.c. COLOR[v] = NERO $d[v] < d[u]$ collega parti diverse di un albero o alberi diversi

COMPONENTI FORTEMENTE CONNESSE (ALGORITMO di TARJAN)

Sia $G=(V,E)$ ORIENTATO

IL GRAFO G^T TRASPOSTO di G, $G^T=(V,E^T)$ $E^T=\{(v,u) \mid (u,v)\in E\}$

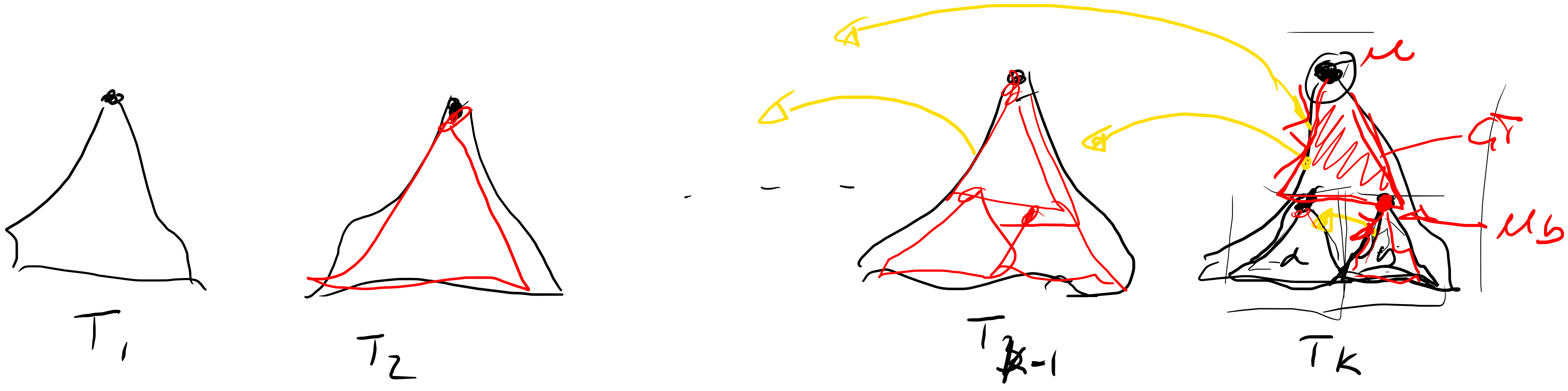
SCC(G)

DFS(G), restituisco i tempi $f[v]$ di fine visita.
CALCOLO G^T
DFS(G^T) esplorando V in ORDINE decrescente rispetto a $f[v]$
RETURN G^T_π

1) $u \overset{p}{\rightsquigarrow} v$ in G $\Rightarrow$ $v \overset{p'}{\rightsquigarrow} u$ in G^T
2) $u \leftrightsquigarrow v$ in G $\Leftrightarrow$ $u \leftrightsquigarrow v$ in G^T
3) $u \leftrightsquigarrow v$ in G $\Leftrightarrow$ $u \rightsquigarrow v$ in G and $u \rightsquigarrow v$ in G^T

in G^T se inizio una visita in T_k non posso mai lasciare T_k

FORESTA DFS di G



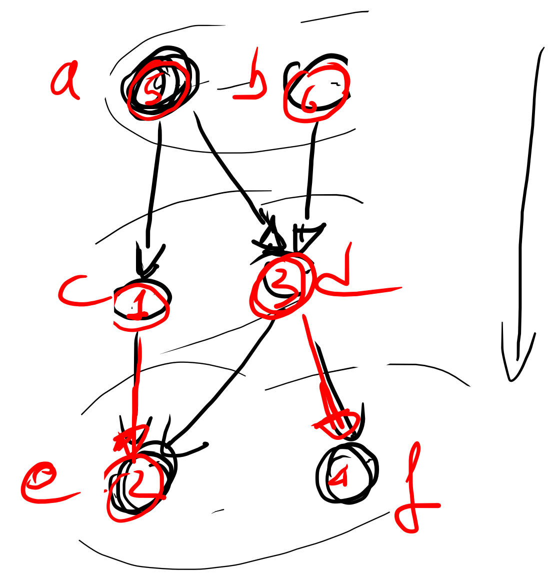
in G^T non posso uscire da T_K

ORDINAMENTO TOPOLOGICO

G DAG

Sia $v_1 \ldots v_n$ un ordinamento dei vertici t.c.

$$\boxed{(v_i, v_j) \in E \Leftrightarrow i < j}$$



$$\boxed{b, a, d, f, c, e}$$