

TECNICO SUPERIORE WEB DEVELOPER FULL STACK

TESTING FEATURES

DEVELOPMENT

#12 - Grafi: Algoritmi notevoli

Introduzione

Dopo avere definito cosa sono i grafi, come rappresentarli e come visitarli, vediamo alcuni algoritmi che, singolarmente, hanno poche applicazioni pratiche, ma che ci daranno enormi poteri successivamente.

- Identificazione delle componenti connesse nei grafi non orientati
- Identificazione dei cicli (in grafi orientati e non)
- Creazione di un ordinamento topologico (grafo orientato)
- Identificazione delle componenti fortemente connesse nei grafi orientati

Grafi: componenti connesse

Componenti (fortemente) connesse

Molti algoritmi che operano sui grafi iniziano decomponendo il grafo nei sue componenti connesse.

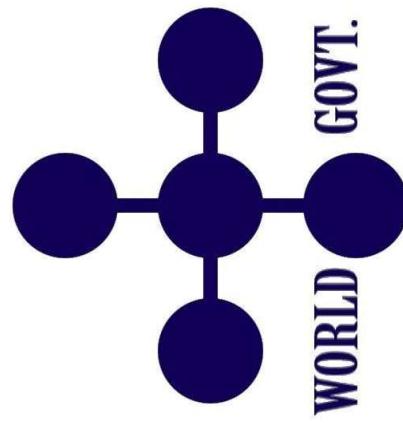
Questo genere di algoritmi sono prima eseguiti su ognuna delle componenti, e il risultato di ogni esecuzione viene ricomposto assieme agli altri.

- Componenti connesse (Connected components, CC), definite su grafi non orientati
- Componenti fortemente connesse (Strongly connected components, SCC), definite su grafi orientati

La Marina ha bisogno di te!

La marina del governo mondiale ha bisogno del vostro aiuto per identificare le ciurme che navigano nella Rotta Maggiore.

Purtroppo, chi raccoglie le informazioni non è un bravo algoritmista...



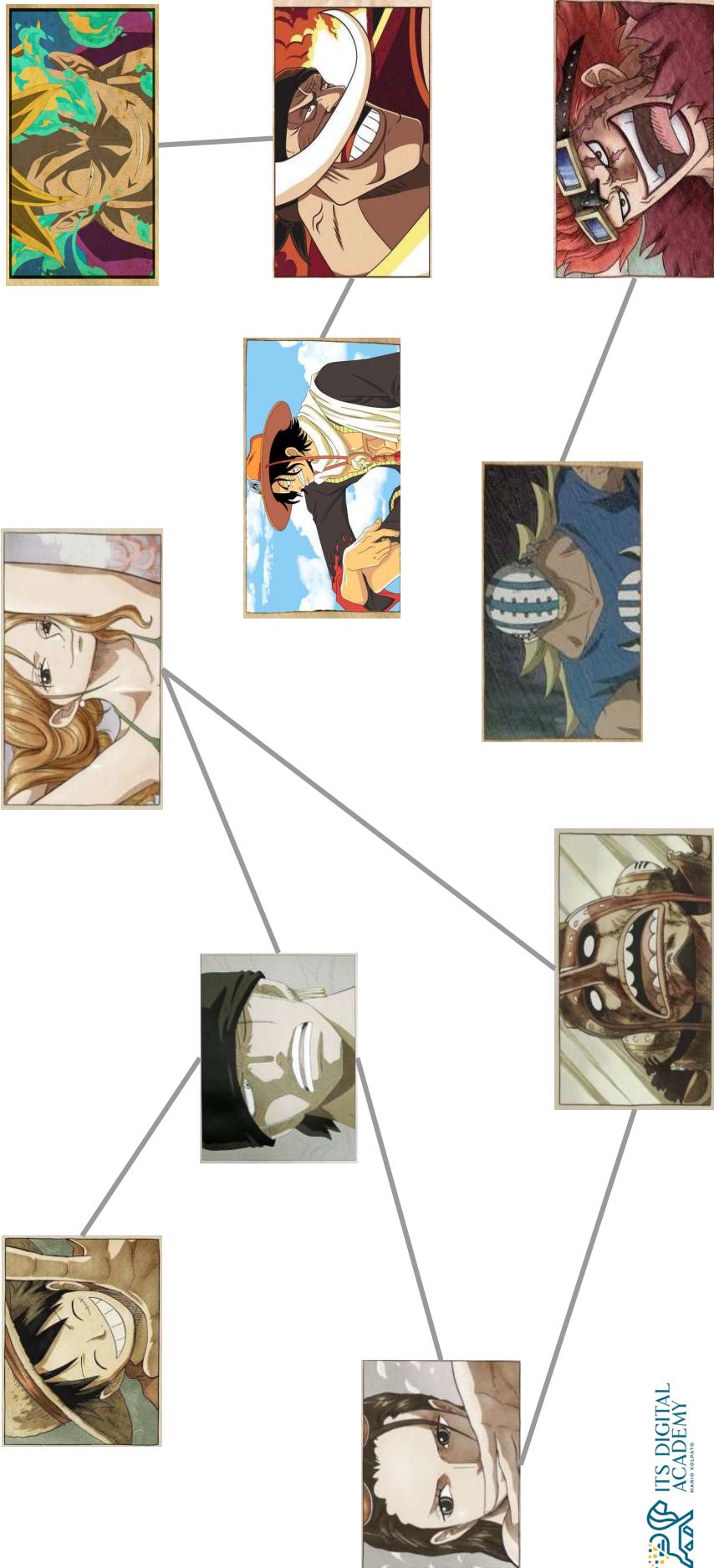
I dati della Marina

La Marina conosce i nomi di vari pirati, ma non è sempre in grado di associarli alla ciurma a cui appartengono; conosce solamente alcune delle relazioni tra pirati della stessa ciurma.

Ad esempio, sa che Nico Robin e Rolonoa Zoro appartengono alla stessa ciurma, così come Zoro e Ruffy.

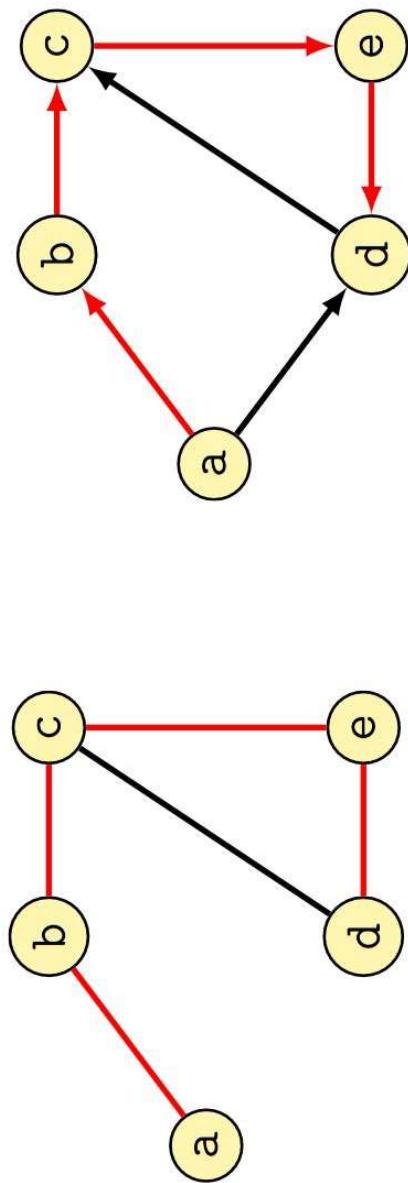
Come fare a capire quante ciurme ci sono?

7



Definizioni: Raggiungibilità

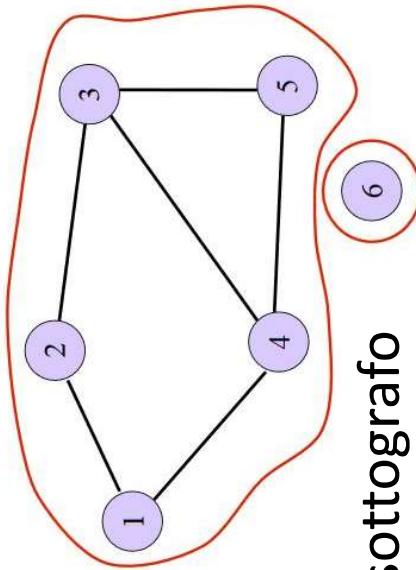
Un nodo v è raggiungibile da un nodo u se esiste almeno un cammino da u a v .



Definizioni

Un grafo non orientato $G=(V, E)$ è **connesso** se e solo se ogni suo nodo è raggiungibile da ogni altro suo nodo

Un grafo $G'=(V', E')$ è una **componente连通** di G se e solo se G' è un *sottografo连通* e *massimale* di G



G' è un **sottografo** di G ($G' \subseteq G$)
se e solo se $V' \subseteq V$ e $E' \subseteq E$

G' è **massimale** se e solo se non esiste un altro sottografo G'' di G tale che G'' è connesso e più grande di G' (i.e. $G' \subseteq G'' \subsetneq G$)

Applicazione di DFS: componenti connesse

Problemi:

- Verificare se un grafo è连通的 oppure no
- Identificare le sue componenti connesse

Soluzione:

Un grafo è连通的 se, al termine della DFS, tutti i nodi sono marcati; altrimenti, la visita deve ricominciare da capo da un nodo non marcato, identificando una nuova componente

Strutture dati necessarie:

- Un vettore id , che contiene gli identificatori delle componenti
- $id[u]$ è l'identificatore della c.c. a cui appartiene u

Identificare una componente连通子图: l'idea

1. Scelgo un nodo
2. Da quel nodo, effettuo una DFS cercando i nodi adiacenti, poi gli adiacenti degli adiacenti, ecc. che riesco a raggiungere.
3. Tutti i nodi raggiunti saranno parte della stessa componente连通子图.

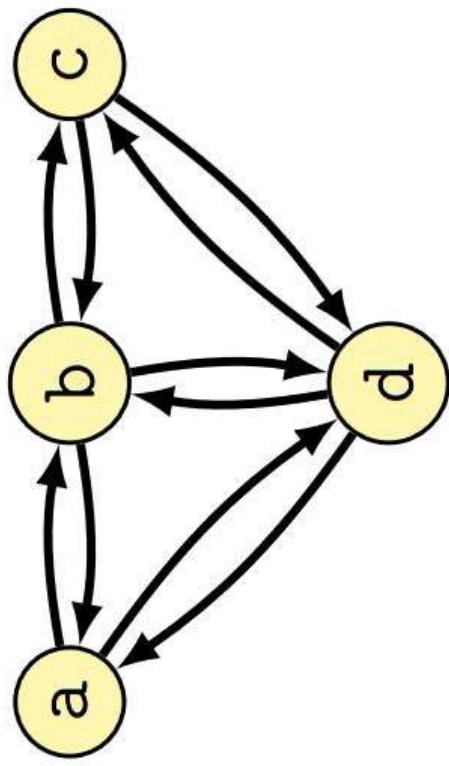
Identificare una componente连通的

12

ccdfs(GRAPH <i>G</i> , int <i>counter</i> , NODE <i>u</i> , int[] <i>id</i>)	Parametri:
<i>id</i> [<i>u</i>] = <i>counter</i> foreach <i>v</i> ∈ <i>G</i> .adj(<i>u</i>) do if <i>id</i> [<i>v</i>] == 0 then ccdfs(<i>G</i> , <i>counter</i> , <i>v</i> , <i>id</i>)	<ul style="list-style-type: none">• <i>G</i>: il grafo• counter: l'attuale CC che stiamo costruendo• <i>u</i>: il nodo in esame• id: il vettore degli identificativi delle CC

Esempio di esecuzione

Stack delle chiamate:

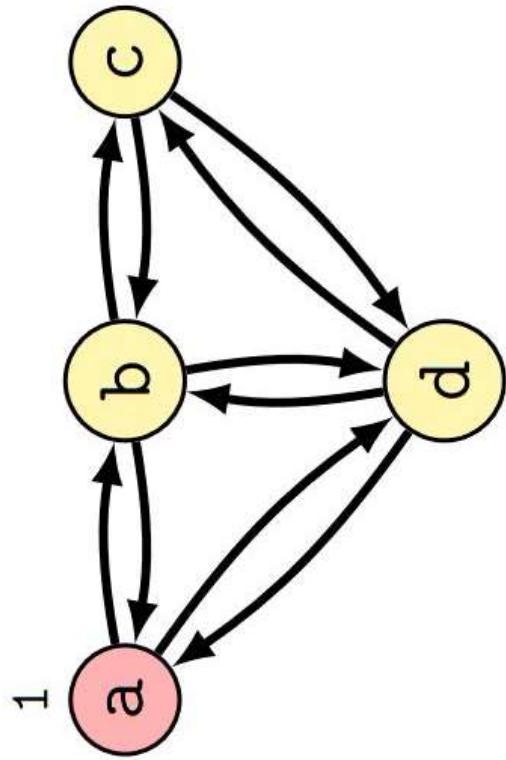


Vettore id: [0,0,0,0]

Esempio di esecuzione

14

Stack delle chiamate:

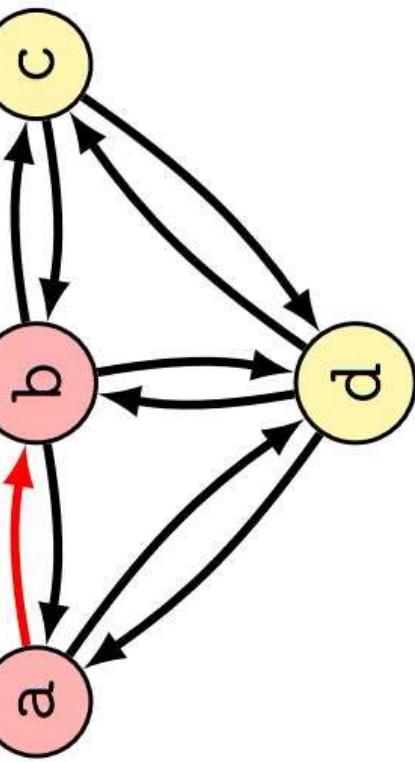


$ccdfs(G, 1, a)$

Vettore id: [1,0,0,0]

Esempio di esecuzione

Stack delle chiamate:



$\text{ccdfs}(G, 1, \text{b})$

$\text{ccdfs}(G, 1, \text{a})$

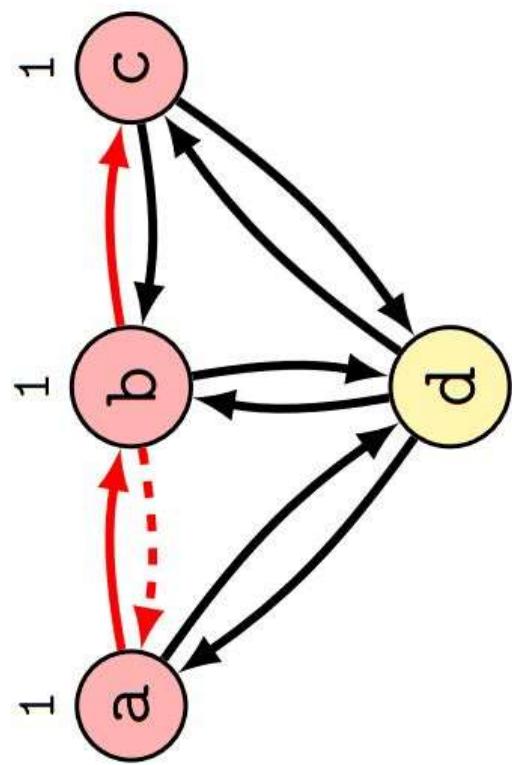
Vettore id: [1,1,0,0]

niukk
Innovation and knowledge

Esempio di esecuzione

16

Stack delle chiamate:



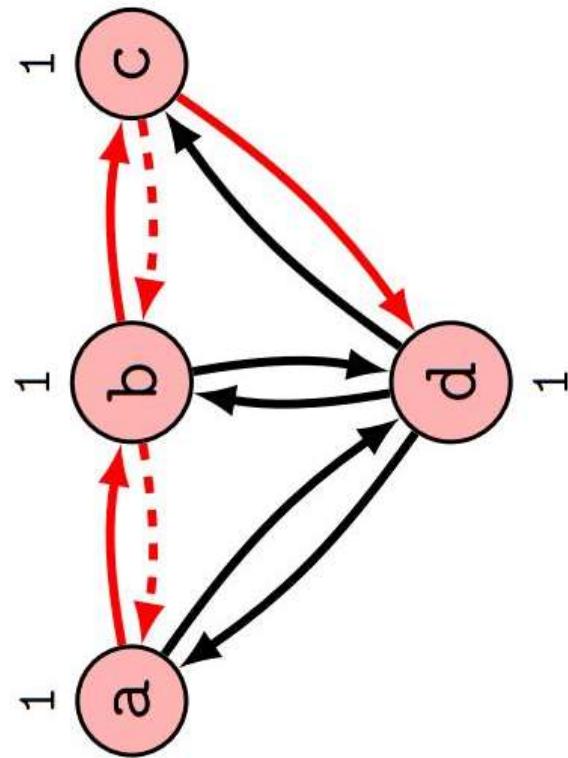
La linea tratteggiata indica che
il nodo di arrivo ha $id[u] != 0$

Vettore id: [1,1,1,0]

niukk
Innovation and knowledge

Esempio di esecuzione

Stack delle chiamate:

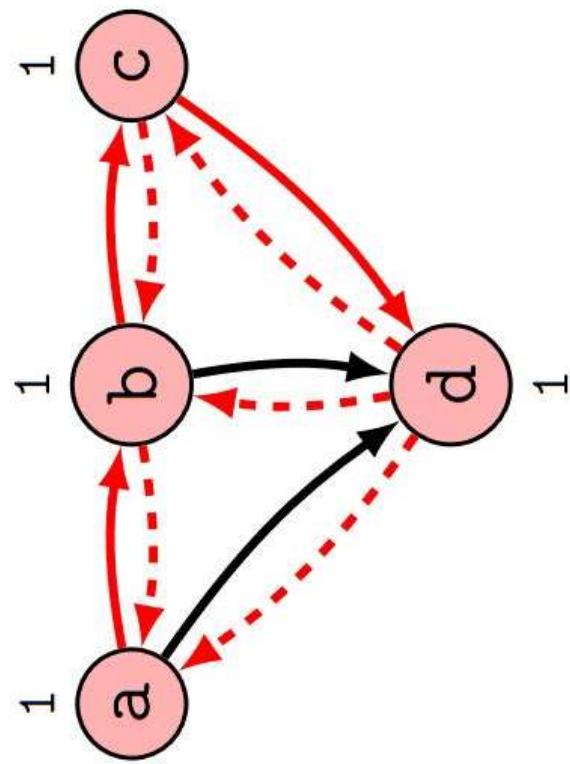


La linea tratteggiata indica che
il nodo di arrivo ha $\text{id}[u] \neq 0$

Vettore id: [1,1,1,1]

Esempio di esecuzione

Stack delle chiamate:



La linea tratteggiata indica che
il nodo di arrivo ha $\text{id}[u] \neq 0$

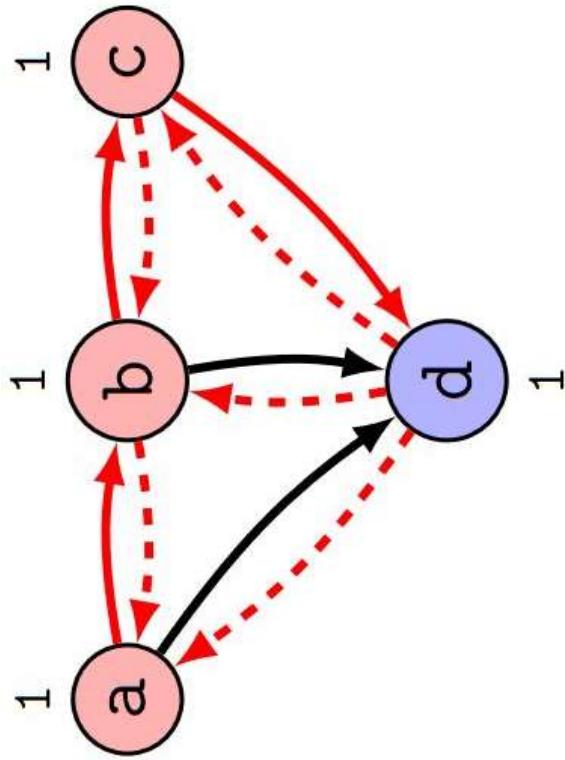
Esempio di esecuzione

Stack delle chiamate:

$\text{ccdfs}(G, 1, c)$

$\text{ccdfs}(G, 1, b)$

$\text{ccdfs}(G, 1, a)$

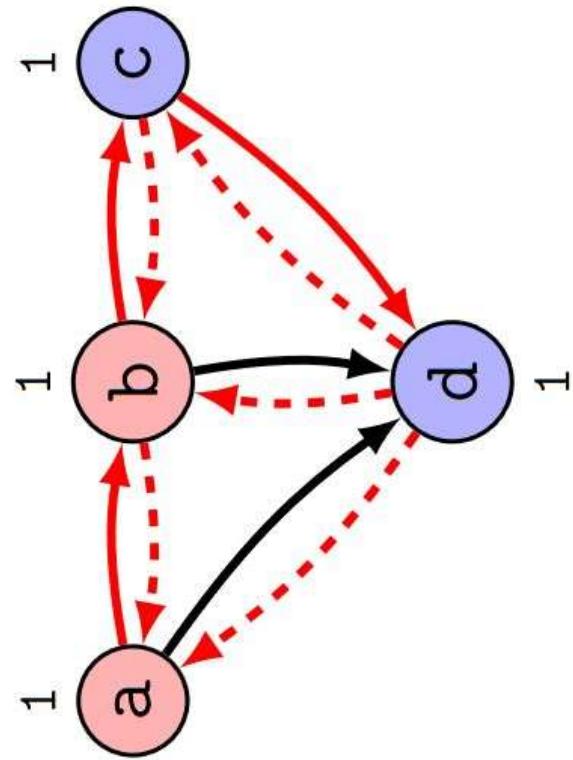


La linea tratteggiata indica che
il nodo di arrivo ha $\text{id}[u] != 0$

Vettore id: [1,1,1,1]

Esempio di esecuzione

Stack delle chiamate:



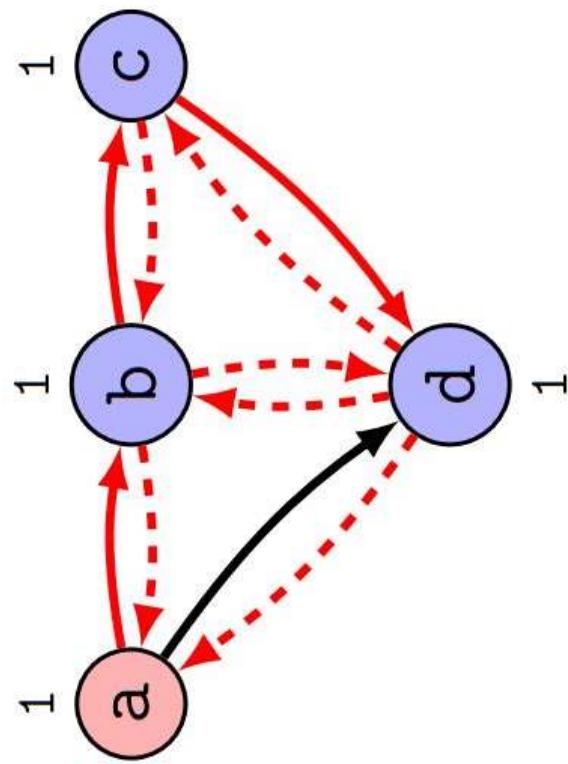
20

La linea tratteggiata indica che
il nodo di arrivo ha $\text{id}[u] \neq 0$

Vettore id: [1,1,1,1]

Esempio di esecuzione

Stack delle chiamate:

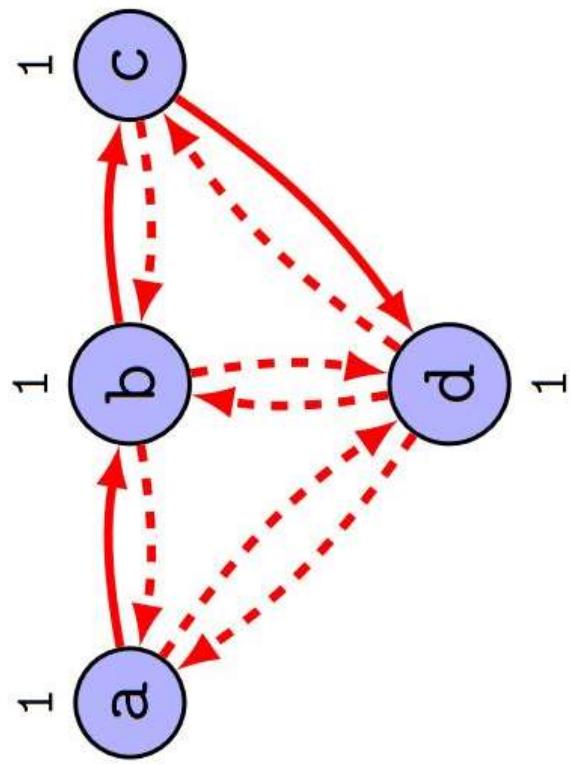


La linea tratteggiata indica che
il nodo di arrivo ha $\text{id}[u] \neq 0$

Vettore id: [1,1,1,1]

Esempio di esecuzione

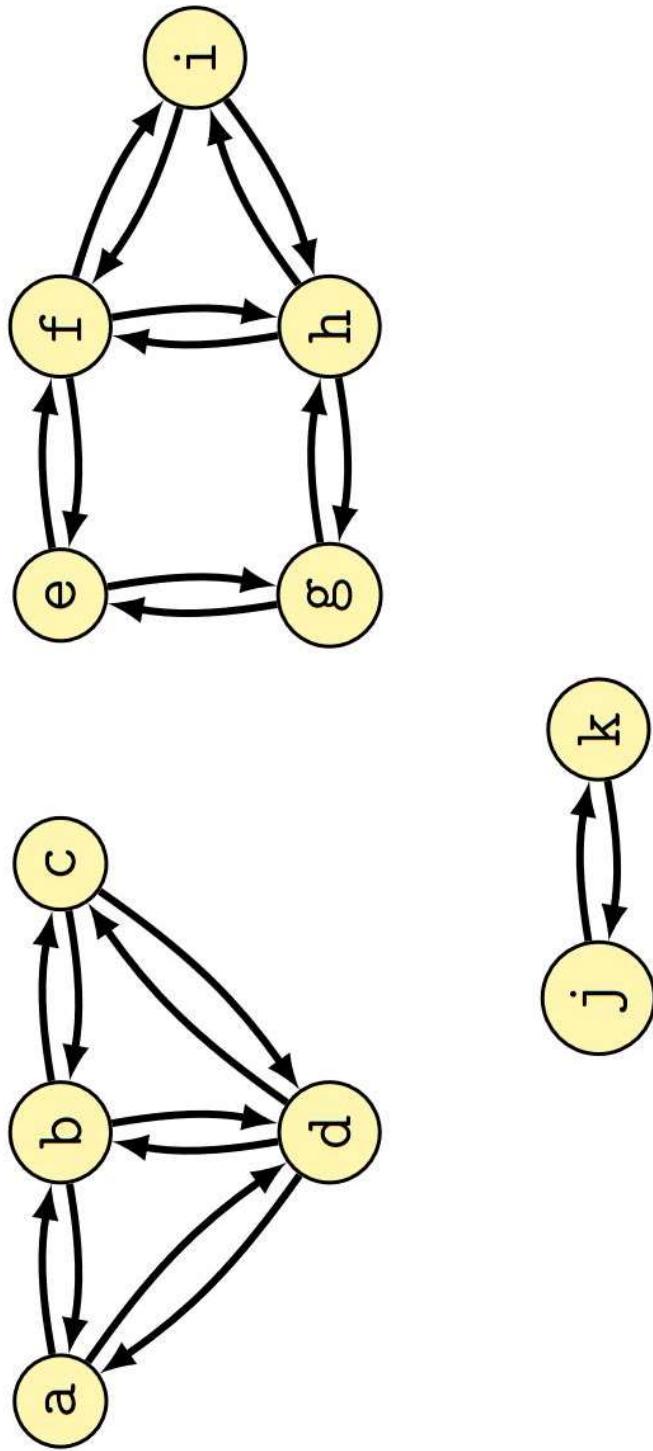
Stack delle chiamate:



La linea tratteggiata indica che
il nodo di arrivo ha $id[u] != 0$

Vettore id: [1,1,1,1]

E se il mio grafo fosse così?



Identificazione di tutte le componenti connesse

24

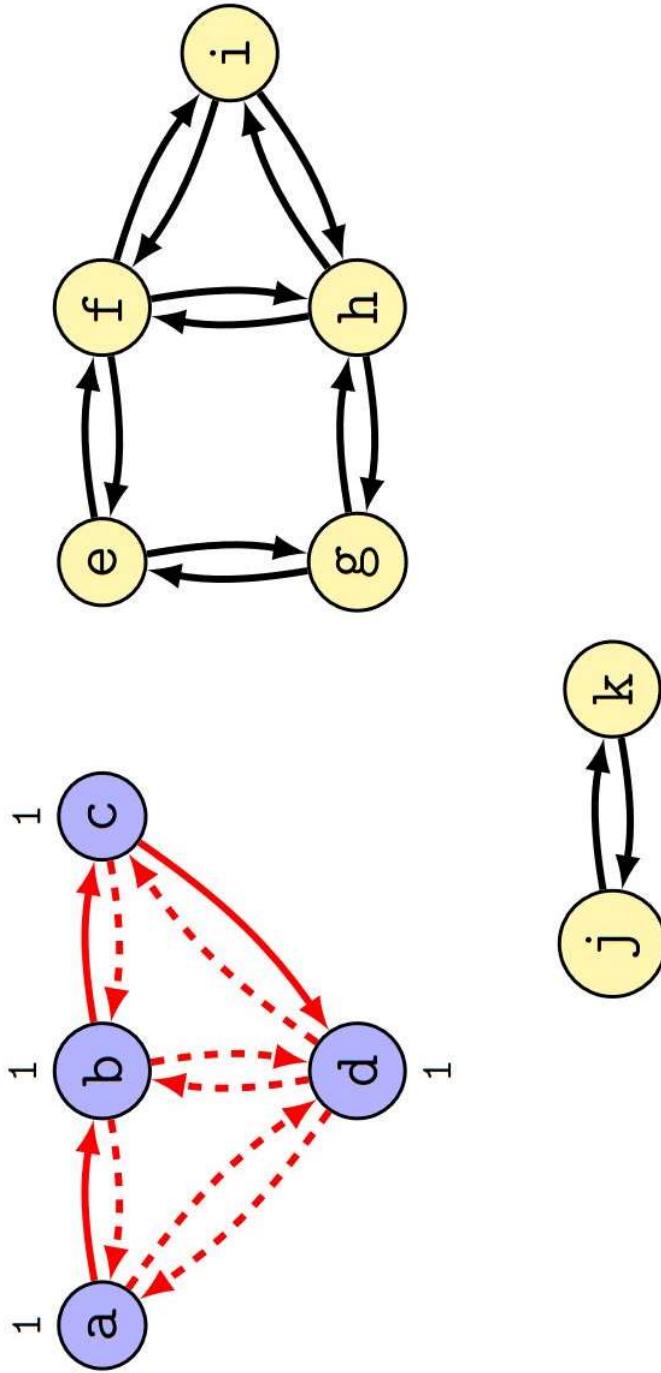
```
int[] cc(GRAPH G)
int[] id = new int[G.size()]
foreach u ∈ G.V() do
    id[u] = 0
int counter = 0
foreach u ∈ G.V() do
    if id[u] == 0 then
        counter = counter + 1
        cdfs(G, counter, u, id)
    else
        return id
return id
```

```
ccdfs(GRAPH G, int counter,
NODE u, int[] id)
id[u] = counter
foreach v ∈ G.adj(u) do
    if id[v] == 0 then
        ccdfs(G, counter, v, id)
```

Il costo computazionale di CC è
 $O(m+n)$, perché in fondo è una DFS.

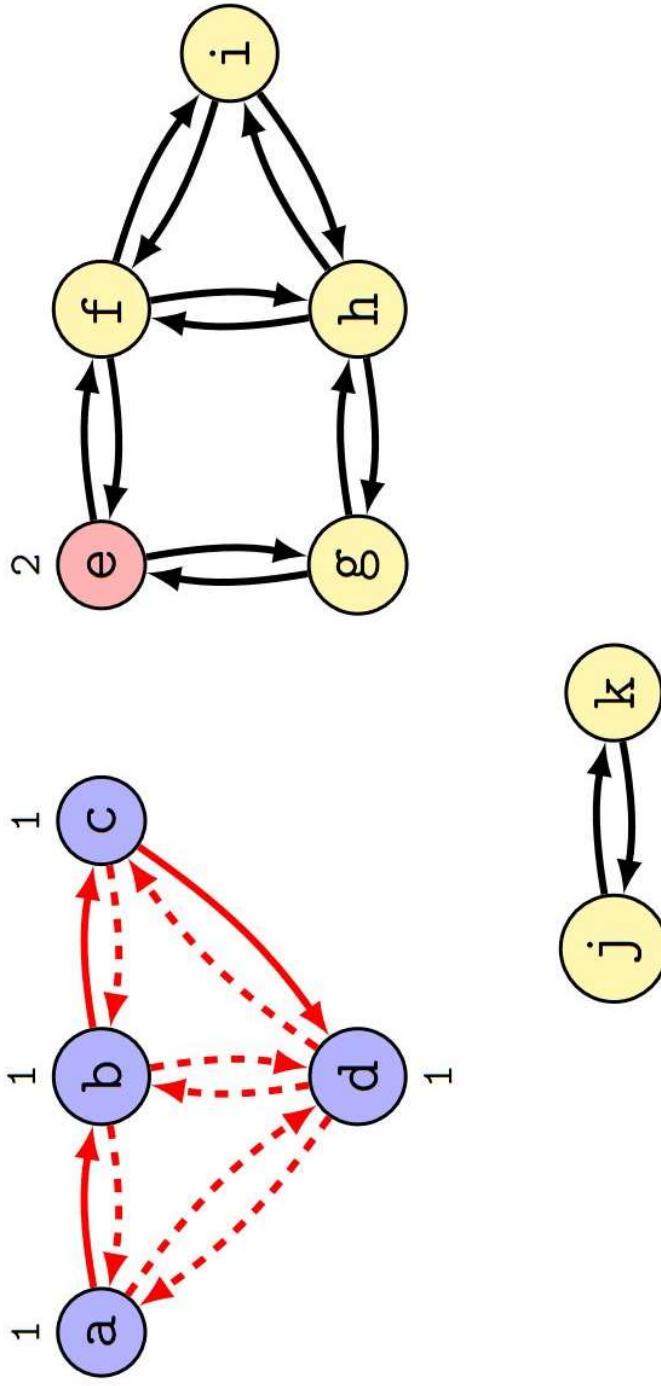
Esempio di esecuzione (continua)

25

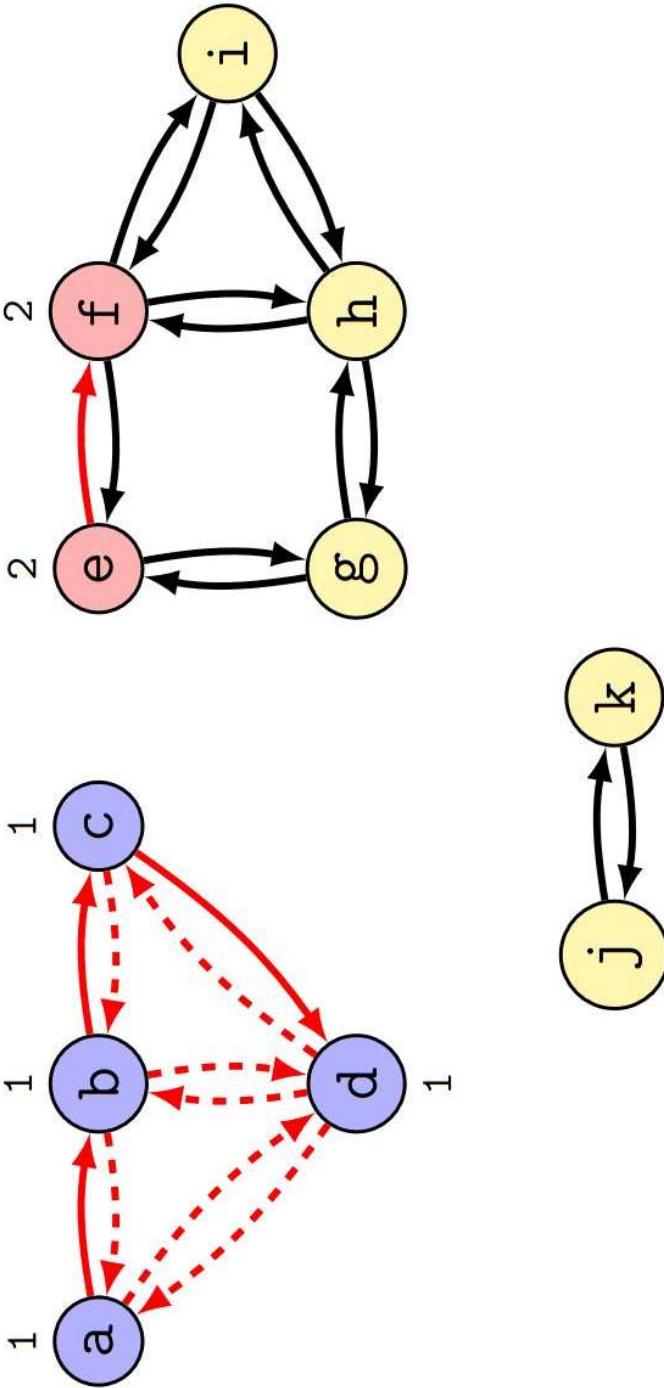


Esempio di esecuzione (continua)

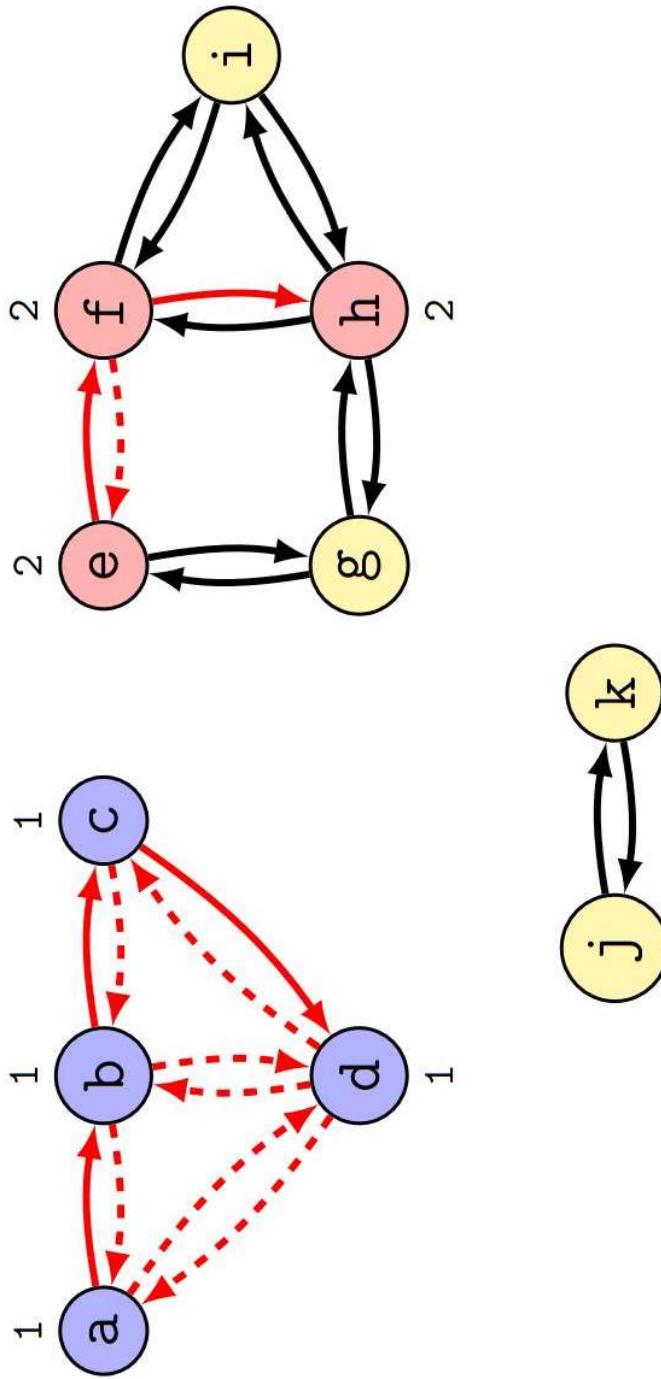
26



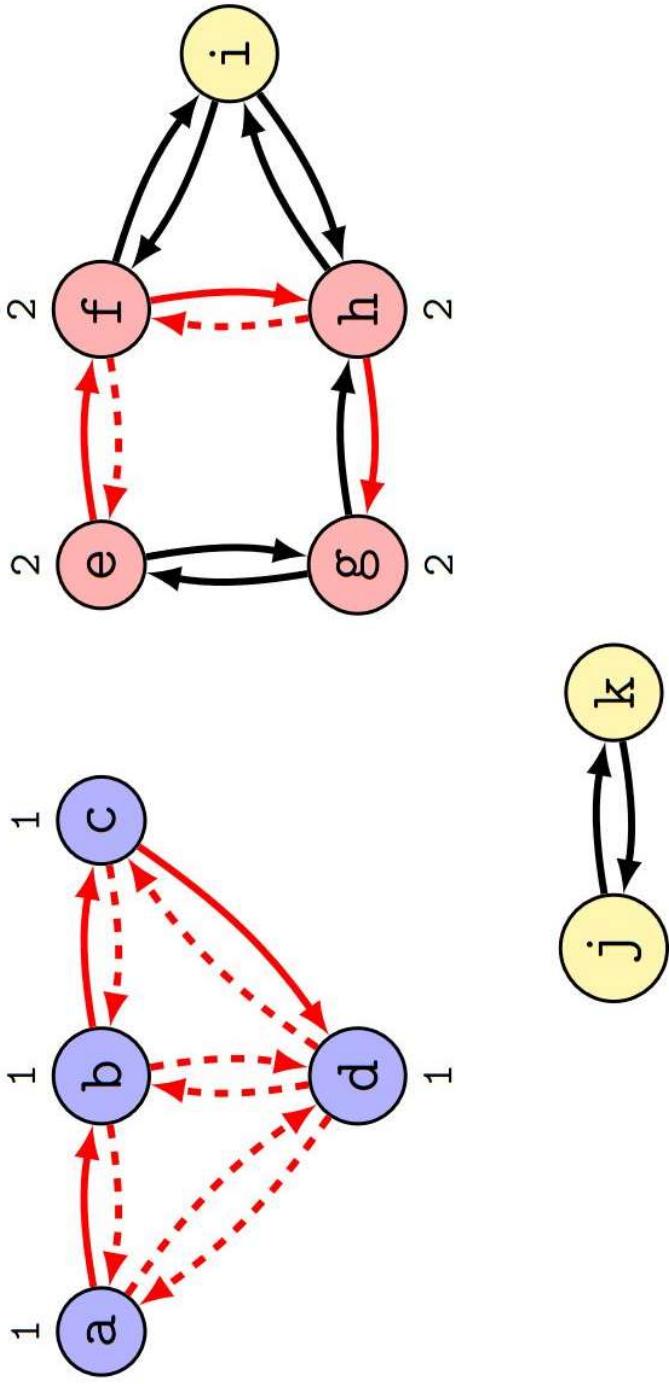
Esempio di esecuzione (continua)



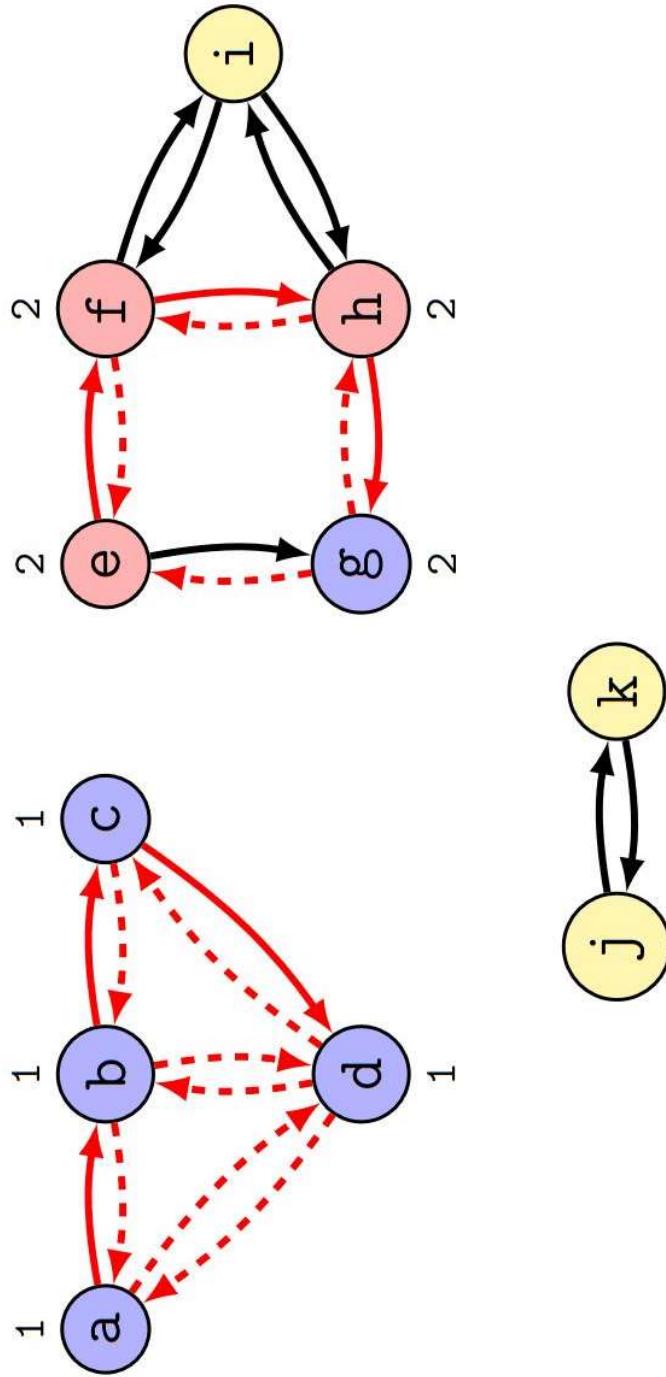
Esempio di esecuzione (continua)



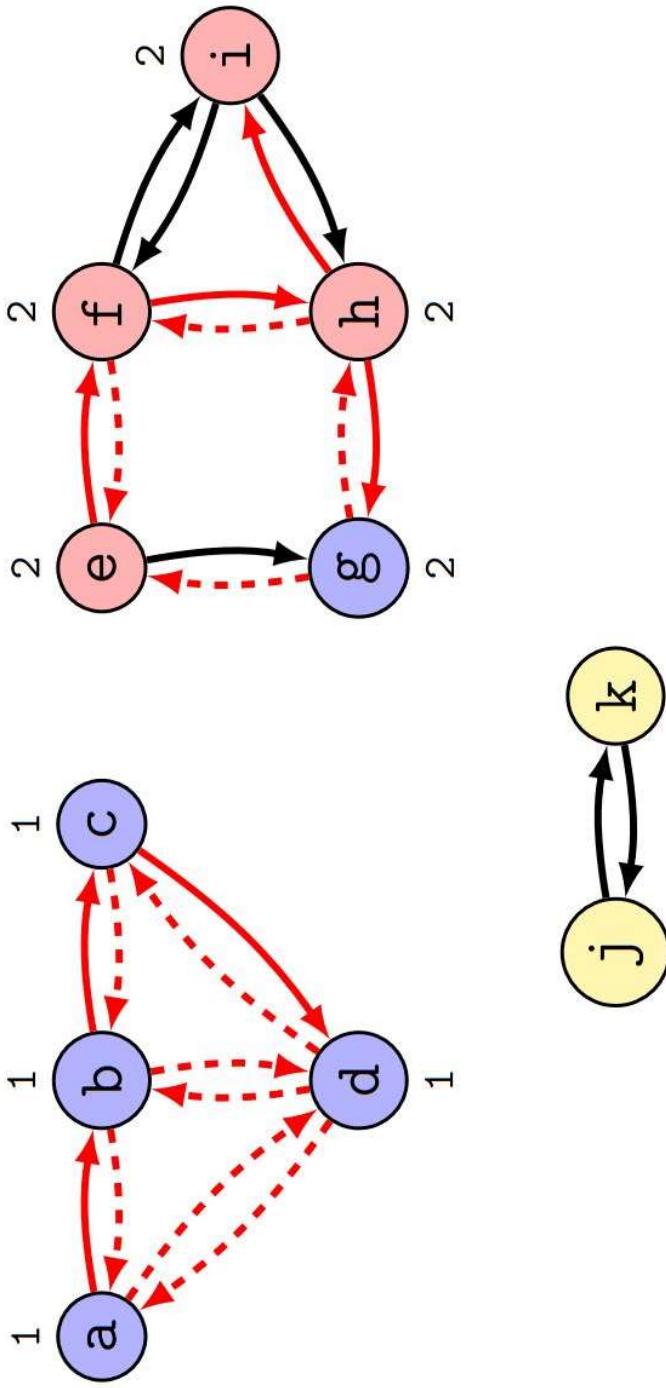
Esempio di esecuzione (continua)



Esempio di esecuzione (continua)

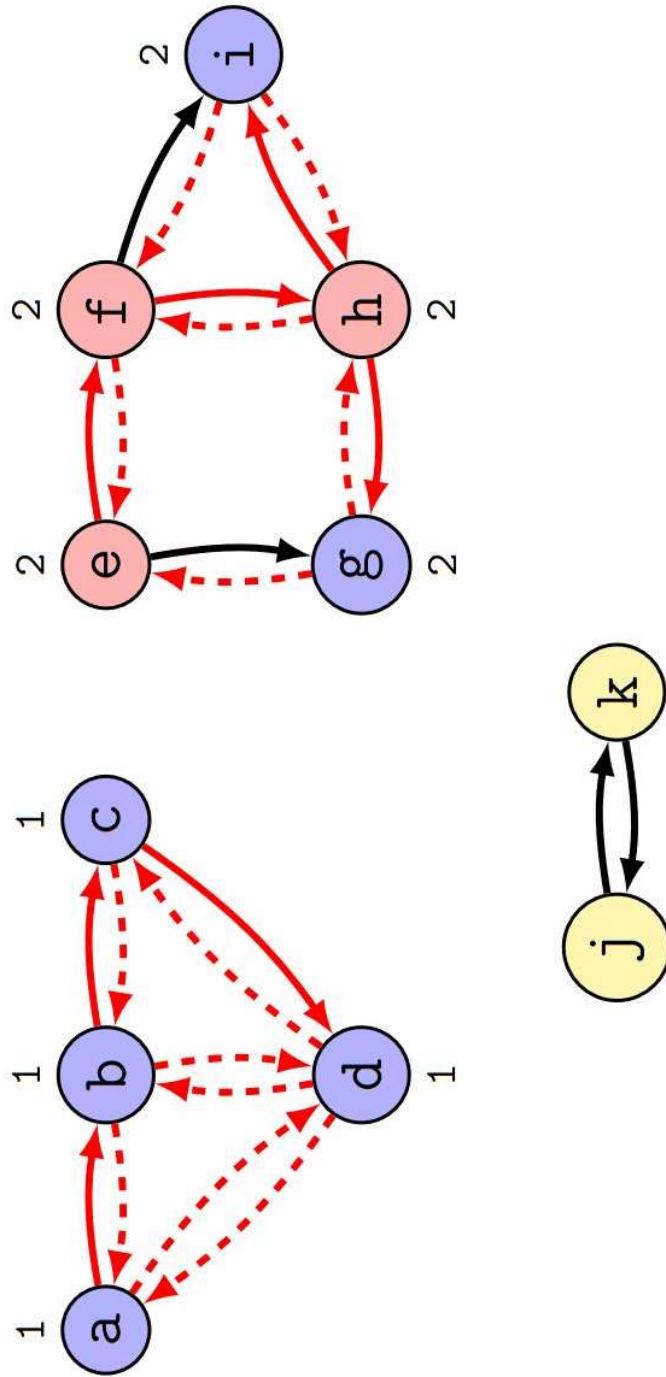


Esempio di esecuzione (continua)

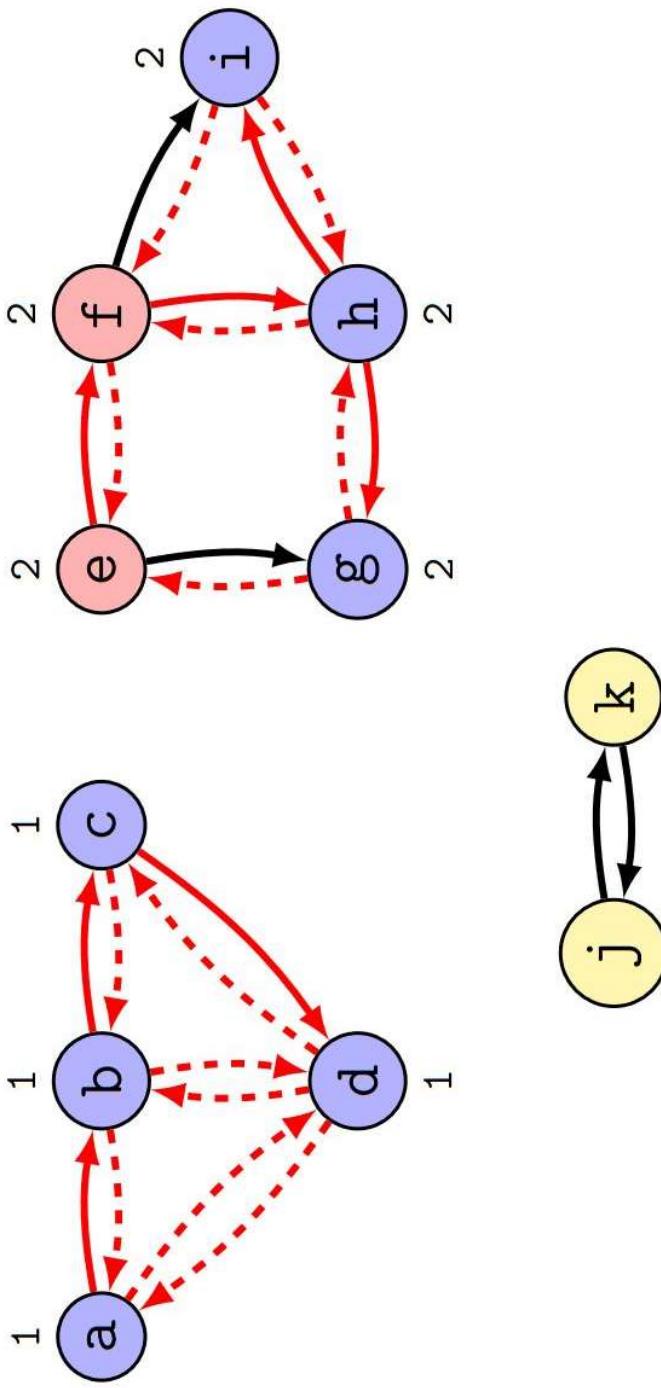


Esempio di esecuzione (continua)

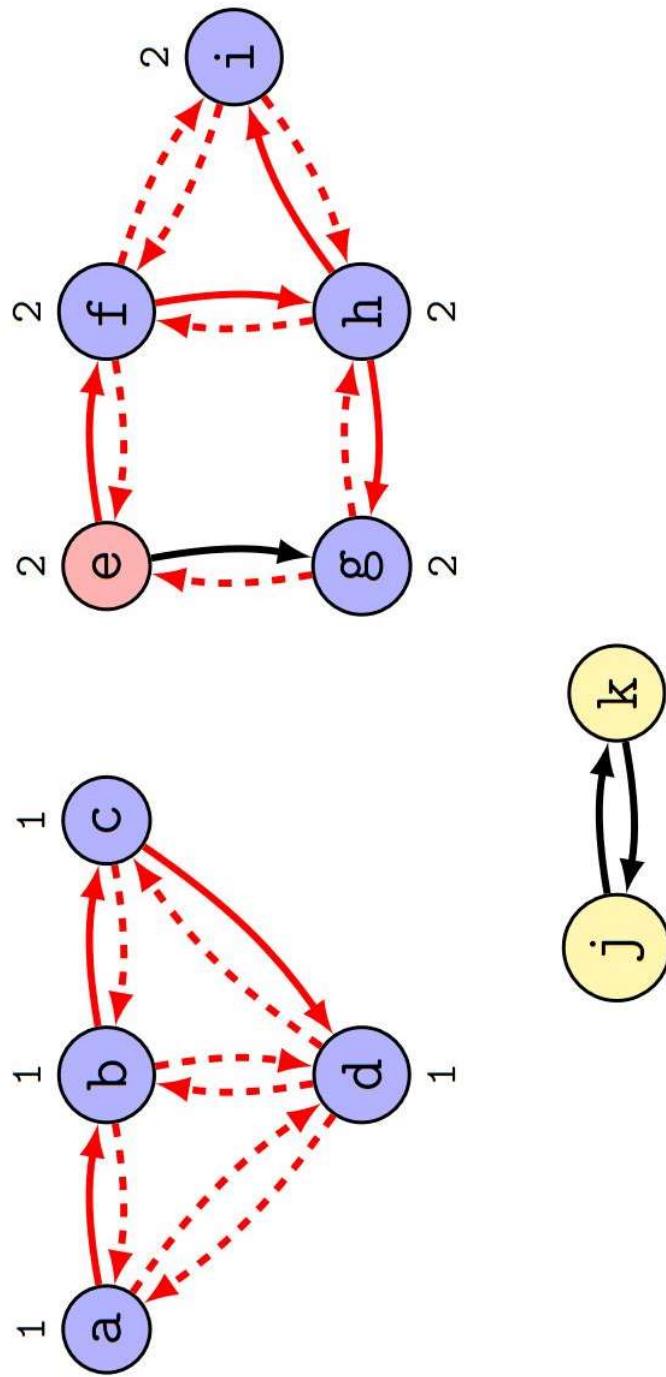
32



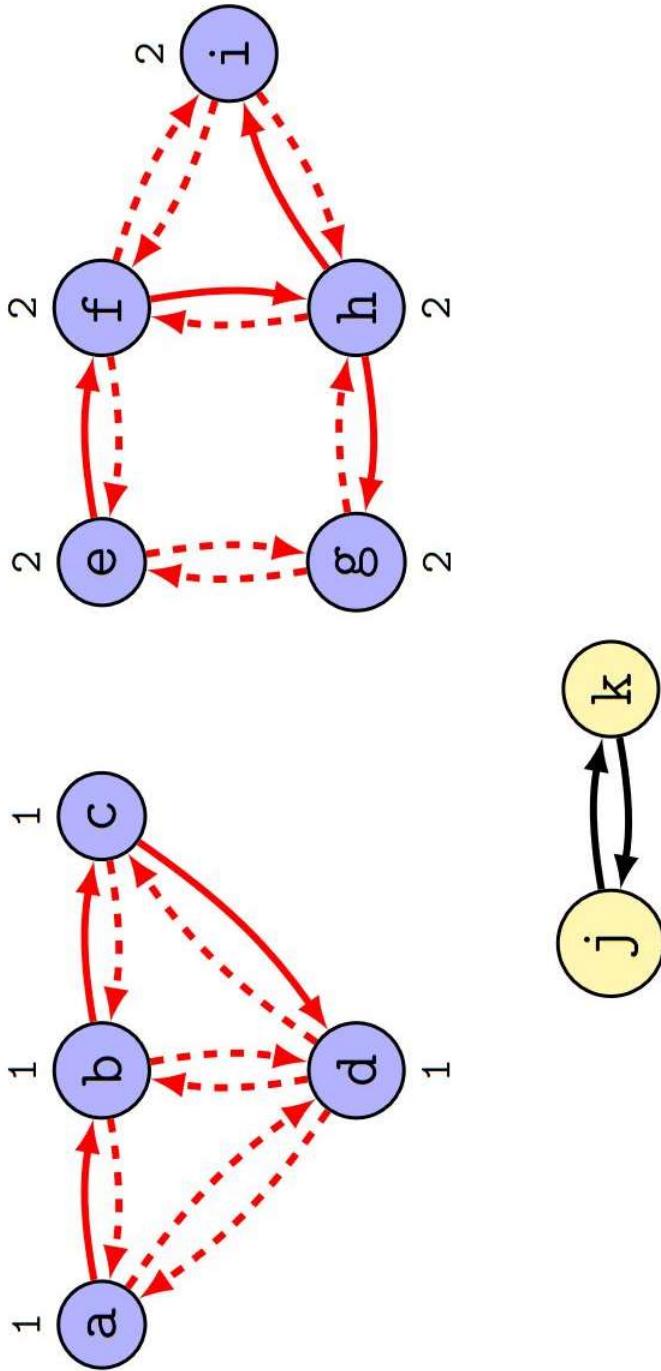
Esempio di esecuzione (continua)



Esempio di esecuzione (continua)

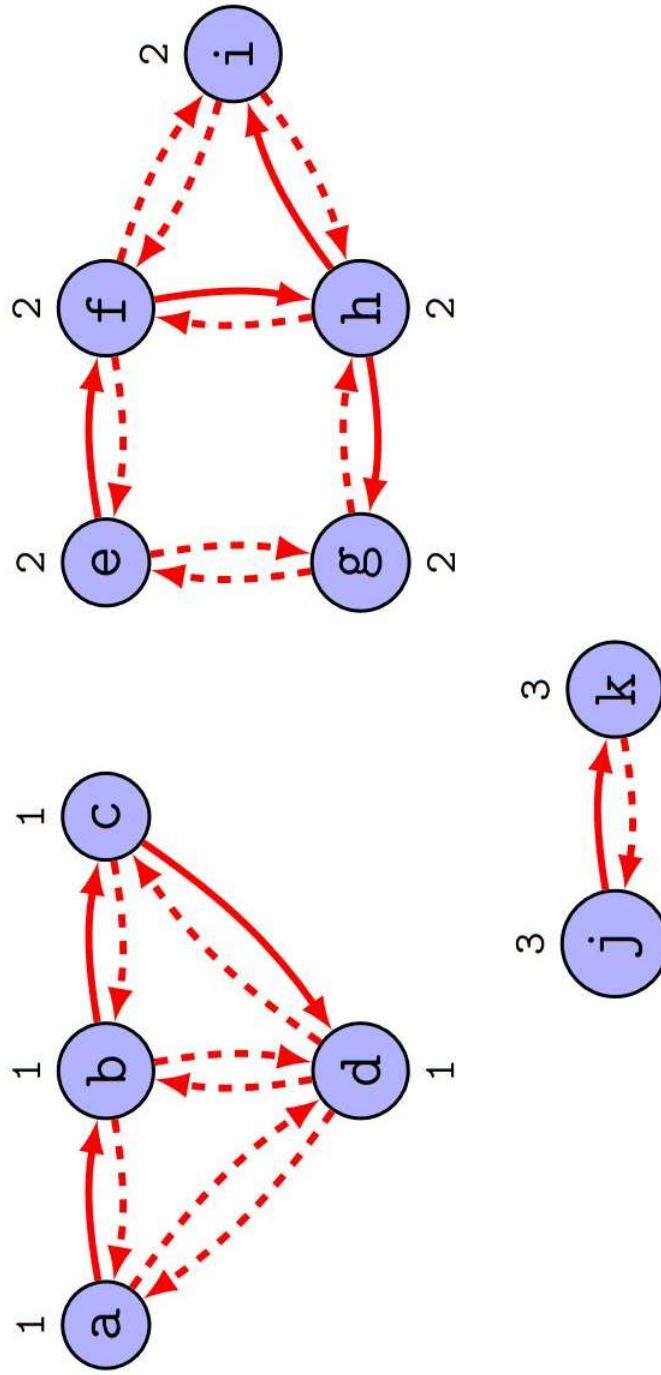


Esempio di esecuzione (continua)

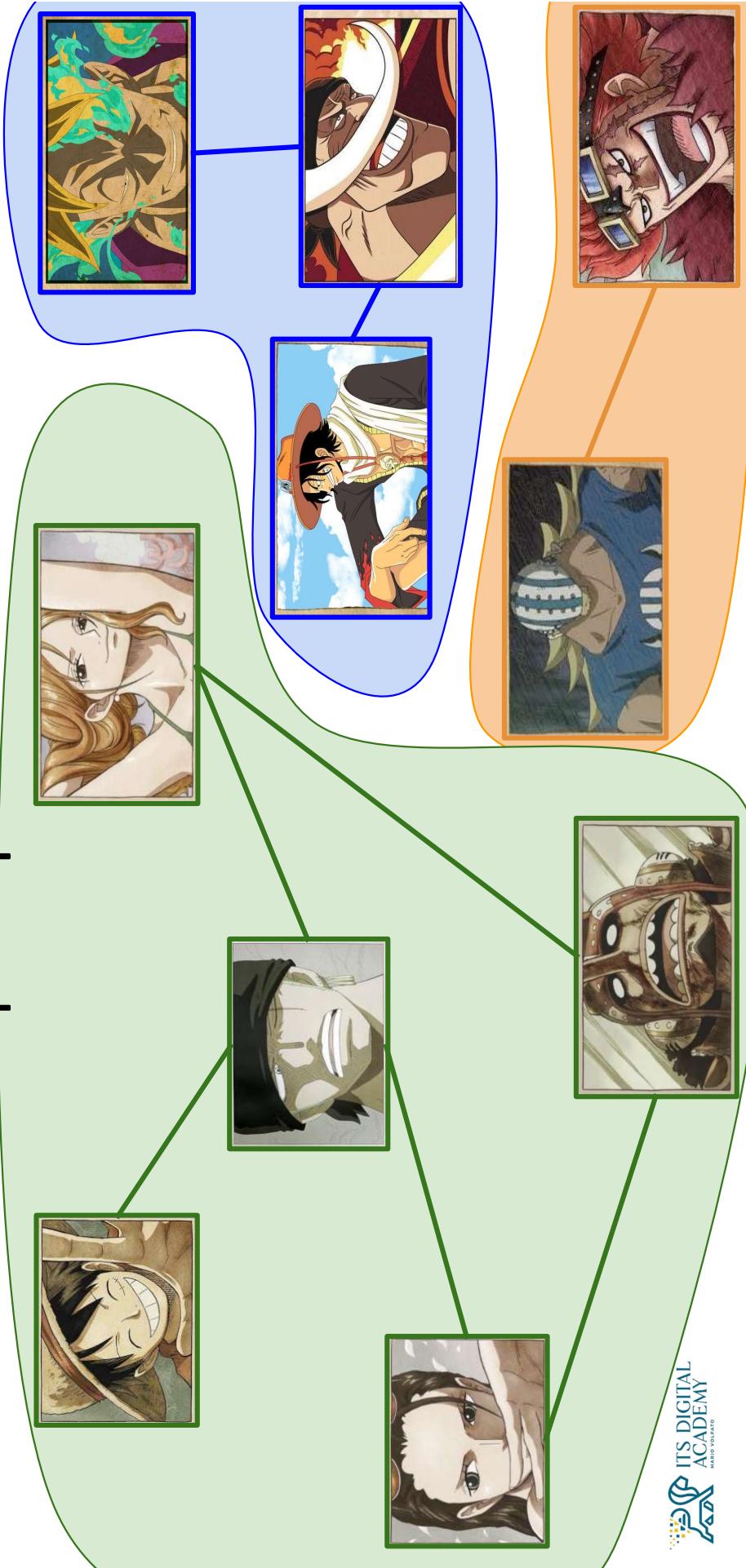


Esempio di esecuzione (continua)

36



Come fare a capire quante ci sono?

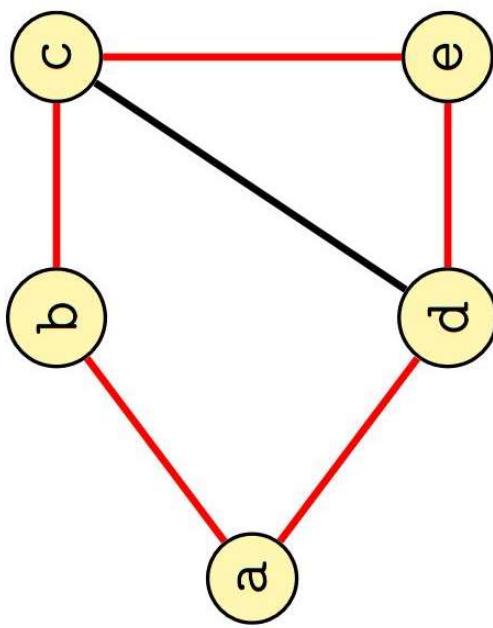


Grafi: cicli

Cicli nei grafi non orientati

In un grafo non orientato $G = (V, E)$, un ciclo C di lunghezza $k > 2$ è una sequenza di nodi u_0, u_1, \dots, u_k tale che $(u_i, u_{i+1}) \in E$ per $0 \leq i \leq k - 1$ e $u_0 = u_k$.

$k > 2$ esclude cicli semplici tra due nodi, sempre presenti nei grafi non orientati.



Grafo aciclico

Un grafo che non contiene cicli è detto **aciclico**.

Ci serve un algoritmo che, dato un grafo non orientato G , restituiscia true se G contiene un ciclo, false altrimenti.

Cicli nei grafi non orientati: l'idea

1. Parto da un nodo k , segnandolo come **visitato**;
2. Effettuo una DFS a partire da k , controllando se, scendendo in profondità, incontro nodi già marcati come **visitati**.
3. Se sì, esiste un ciclo (return true)
4. Se no?

Cicli nei grafi non orientati: lo pseudocodice

42

```
boolean hasCycleRec(GRAPH G, NODE u, NODE p, boolean[] visited)
visited[u] = true
foreach v ∈ G.adj(u) – {p} do
    if visited[v] then
        return true
    else if hasCycleRec(G, v, u, visited) then
        return true
    else
        return false
```

Il nodo p è il nodo da cui arrivo, che è sicuramente già visitato

Cicli nei grafi non orientati: l'idea

1. Parto da un nodo k , segnandolo come **visitato**;
2. Effettuo una DFS a partire da k , controllando se, scendendo in profondità, incontro nodi già marcati come **visitati**.
3. Se sì, esiste un ciclo (return true)
4. Se no? Non c'è un ciclo in questa componente, ma *potrebbe esistere in un'altra*. Quindi devo ricominciare a cercare tra i nodi non visitati.
5. Se non ho mai restituito true prima, alla fine restituisco false.

Cicli nei grafi non orientati: lo pseudocodice

44

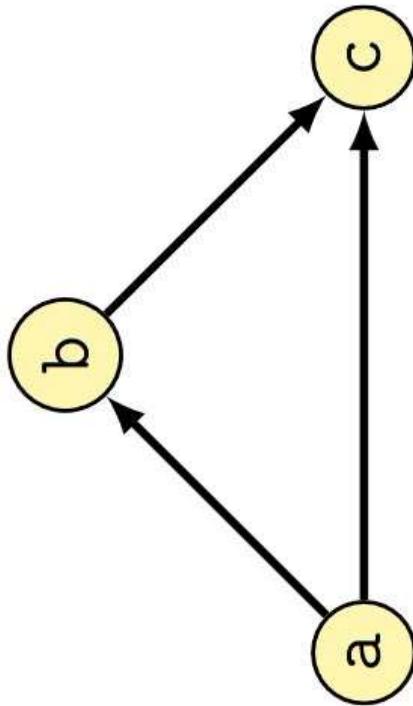
```
boolean hasCycle(GRAPH G)


---

boolean[] visited = new boolean[G.size()]
foreach u ∈ G.v() do
    [ ] visited[u] = false
foreach u ∈ G.v() do
    if not visited[u] then
        [ ] if hasCyclerec(G, u, null, visited) then
            [ ] return true
    [ ]
return false
```

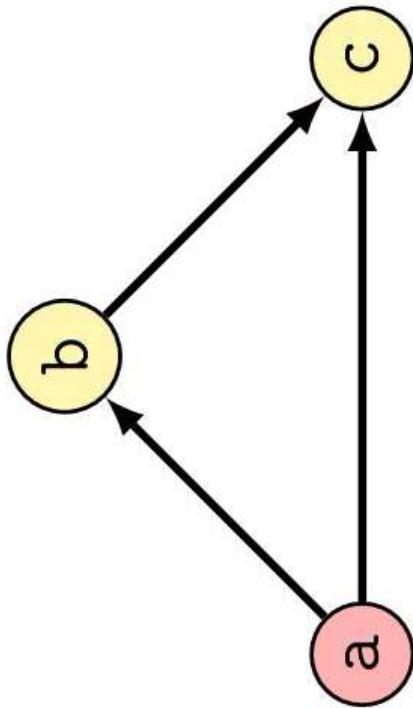
Cicli nei grafi orientati: il problema

In un grafo orientato, individuare un ciclo è più complesso:
effettuando una DFS analoga a quella dei grafi non orientati,
infatti, non so se un eventuale nodo già visitato è parte di un
ciclo oppure no.



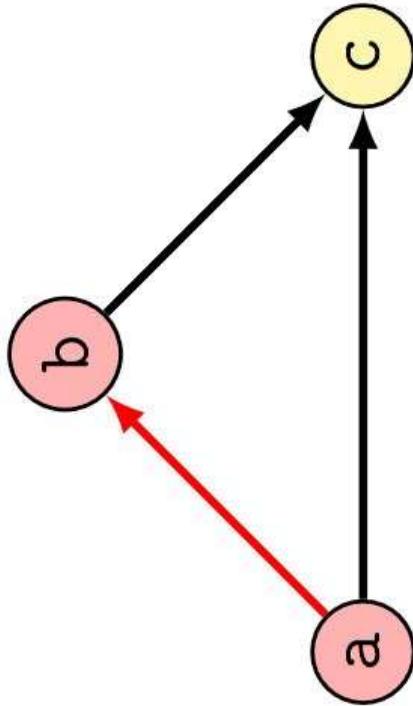
Cicli nei grafi orientati: il problema

In un grafo orientato, individuare un ciclo è più complesso:
effettuando una DFS analoga a quella dei grafi non orientati,
infatti, non so se un eventuale nodo già visitato è parte di un
ciclo oppure no.



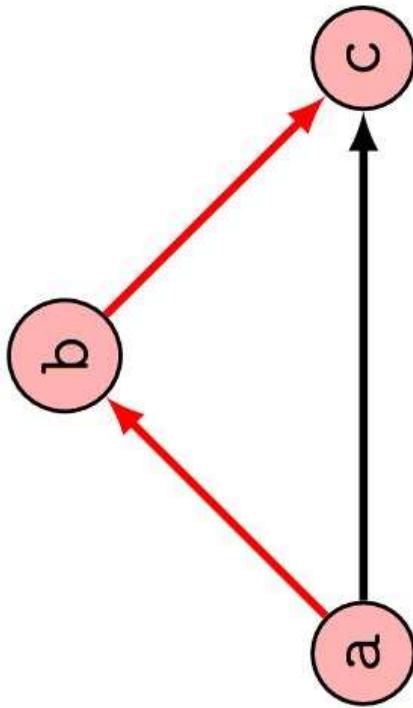
Cicli nei grafi orientati: il problema

In un grafo orientato, individuare un ciclo è più complesso:
effettuando una DFS analoga a quella dei grafi non orientati,
infatti, non so se un eventuale nodo già visitato è parte di un
ciclo oppure no.



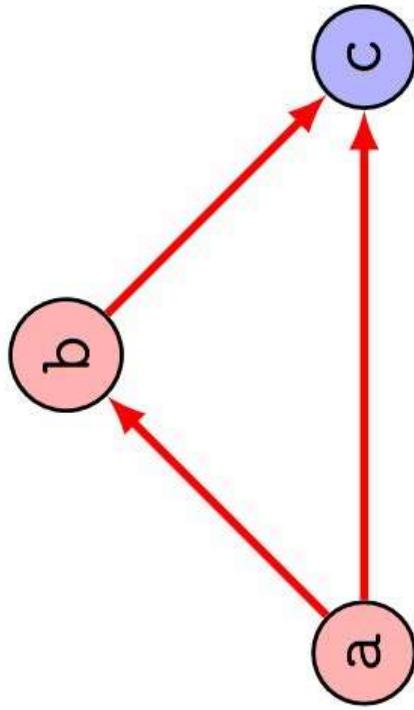
Cicli nei grafi orientati: il problema

In un grafo orientato, individuare un ciclo è più complesso:
effettuando una DFS analoga a quella dei grafi non orientati,
infatti, non so se un eventuale nodo già visitato è parte di un
ciclo oppure no.



Cicli nei grafi orientati: il problema

In un grafo orientato, individuare un ciclo è più complesso:
effettuando una DFS analoga a quella dei grafi non orientati,
infatti, non so se un eventuale nodo già visitato è parte di un
ciclo oppure no.



Cicli nei grafi orientati: l'idea

Semplificando molto, possiamo procedere così:

1. Parto da un nodo k , segnandolo come in corso di visita
2. Effettuo una DFS a partire da k , controllando se, scendendo in profondità, incontro nodi già completamente visitati, ossia marcati come visitati e non in corso di visita.
3. Se sì, esiste un ciclo (return true)
4. Se no? Non c'è un ciclo in questa componente, ma potrebbe esistere in un'altra. Quindi devo ricominciare a cercare tra i nodi non visitati.
5. Se non ho mai restituito true prima, alla fine restituisco false.

Cicli nei grafi Orientati: l'idea

Per marcare un nodo come in corso di visita o visitato, usiamo due array di appoggio, dt e ft , che conterranno il tempo di inizio (discovery time) e di fine visita (finish time):

1. inizialmente tutti i nodi hanno $dt[u] = 0$ e $ft[u] = 0$;
2. quando scopro un nodo, segno il suo dt ;
3. quando ho concluso la visita di un nodo (ho effettuato DFS su tutti i suoi adiacenti), segno il suo ft .
4. se durante la visita incontro un nodo che ha $dt[u] > dt[v]$ e $ft[v] == 0$ allora è un nodo in corso di visita;

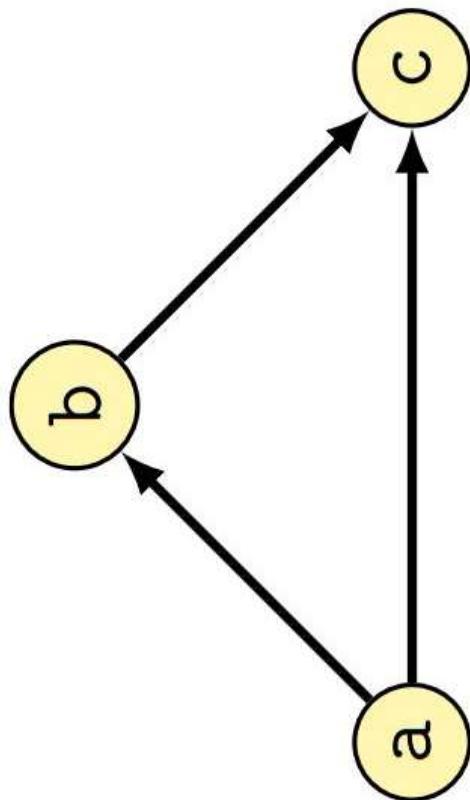
Ciò significa che ho trovato un ciclo.

Cicli nei grafi orientati: lo pseudocodice

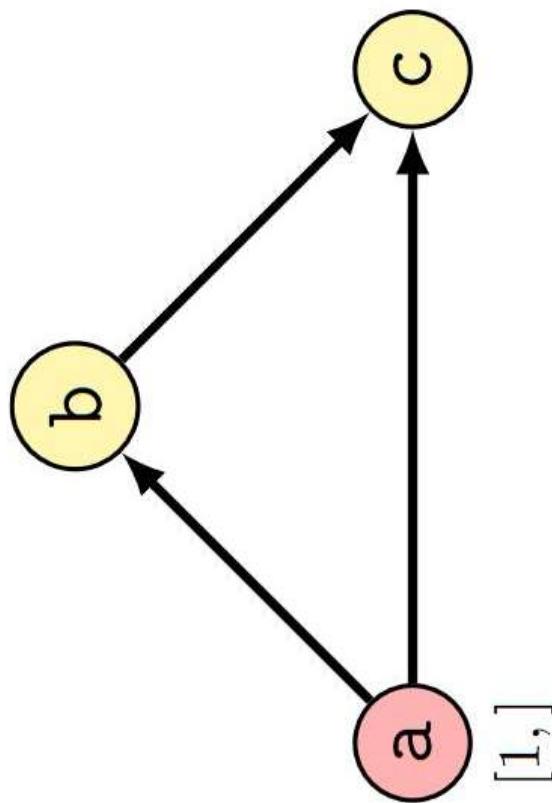
```
boolean hasCycle(GRAPH G, NODE u, int &time, int[] dt, int[] ft)
time = time + 1; dt[u] = time
foreach v ∈ G.adj(u) do
    if dt[v] == 0 then
        if hasCycle(G, v, time, dt, ft) then
            return true
    else if dt[u] > dt[v] and ft[v] == 0 then
        return true
    time = time + 1; ft[u] = time
return false
```

NB: time è una variabile globale.

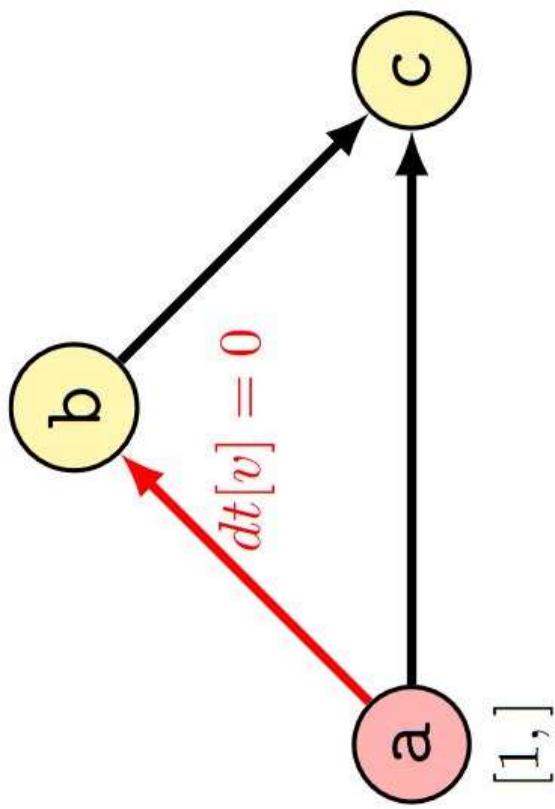
Cicli nei grafi orientati: esempio senza ciclo



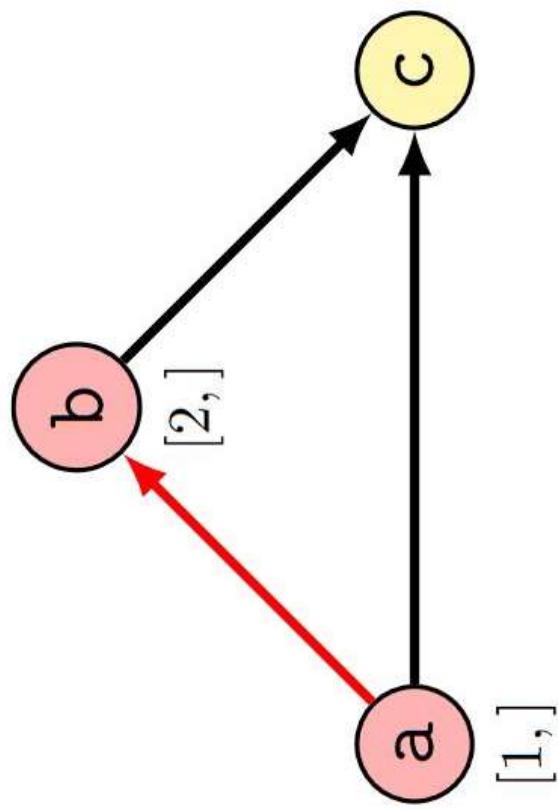
Cicli nei grafi orientati: esempio senza ciclo



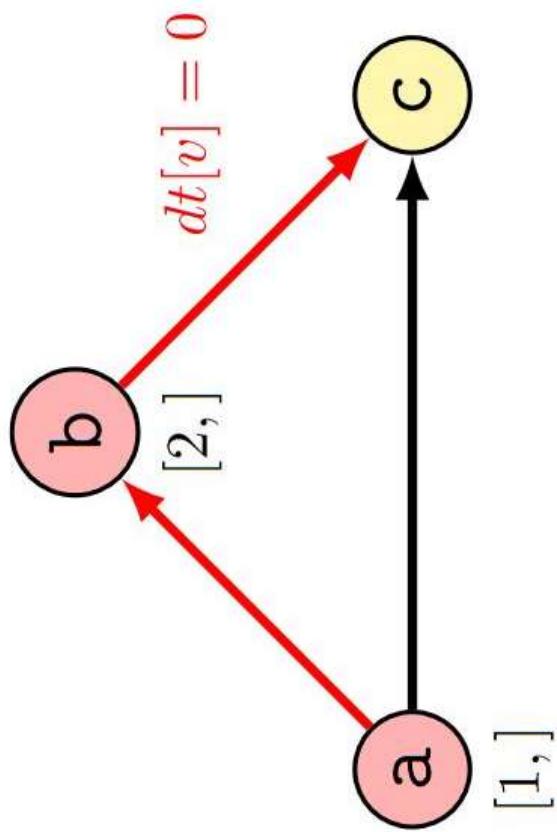
Cicli nei grafi orientati: esempio senza ciclo



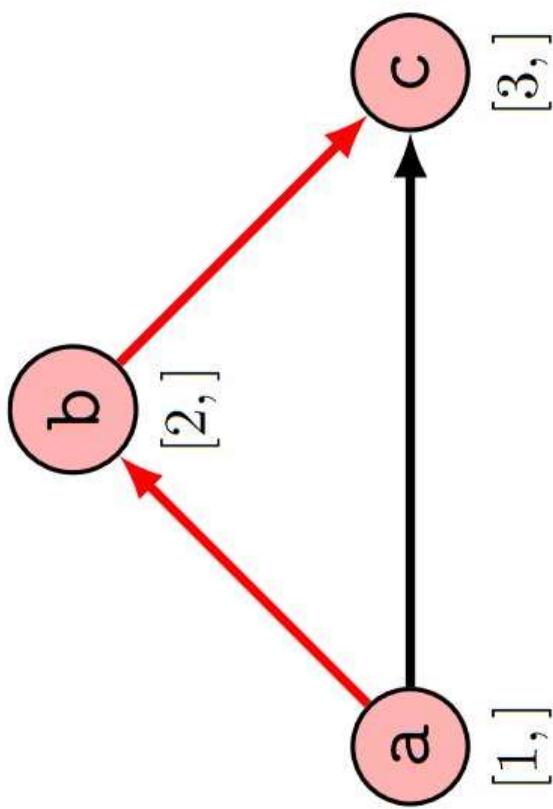
Cicli nei grafi orientati: esempio senza ciclo



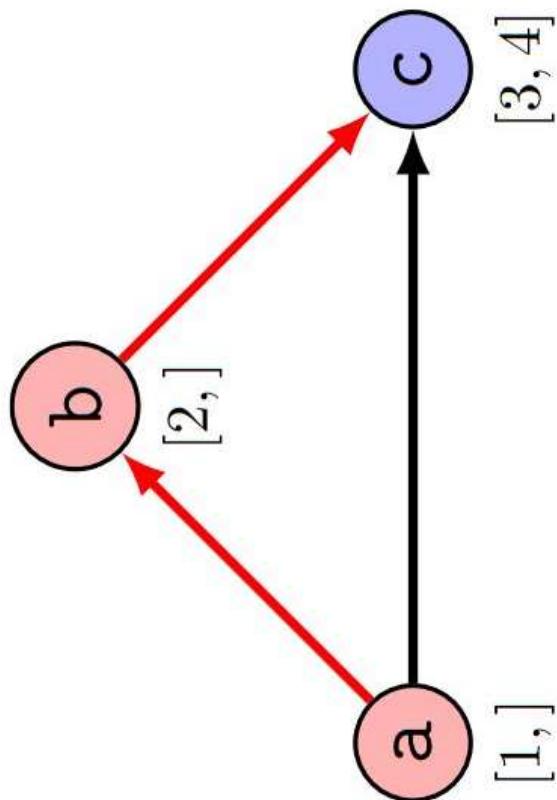
Cicli nei grafi orientati: esempio senza ciclo



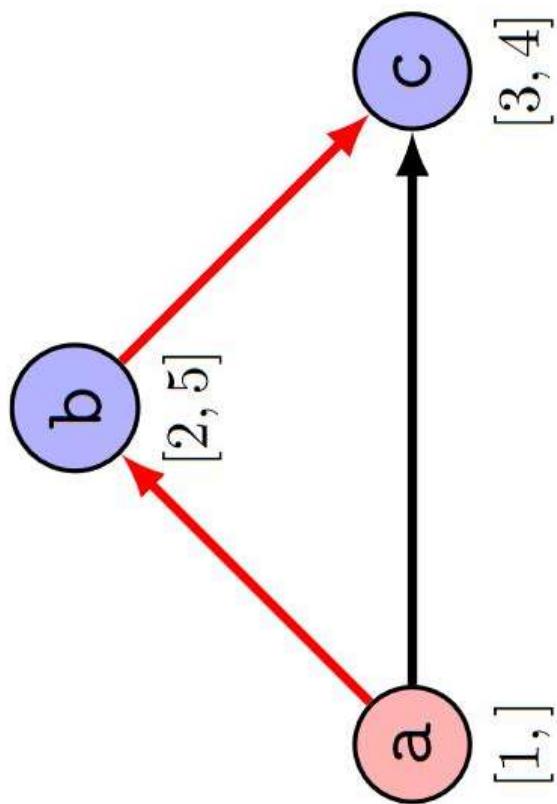
Cicli nei grafi orientati: esempio senza ciclo



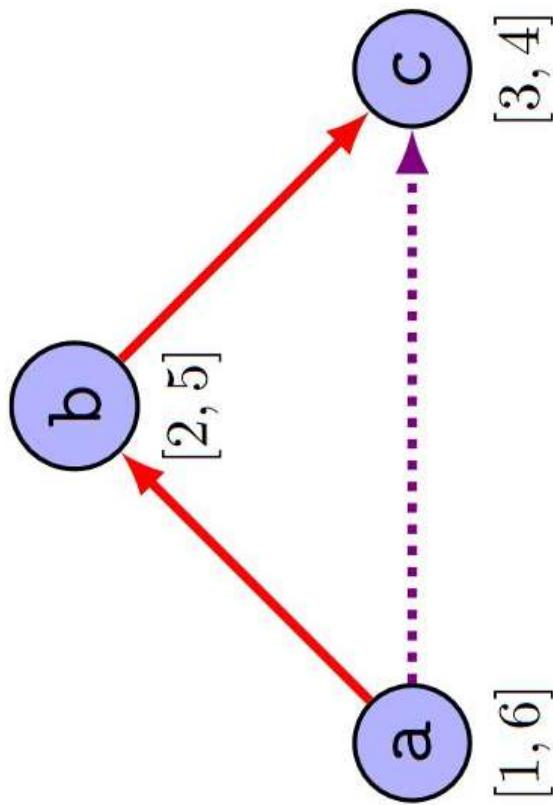
Cicli nei grafi orientati: esempio senza ciclo



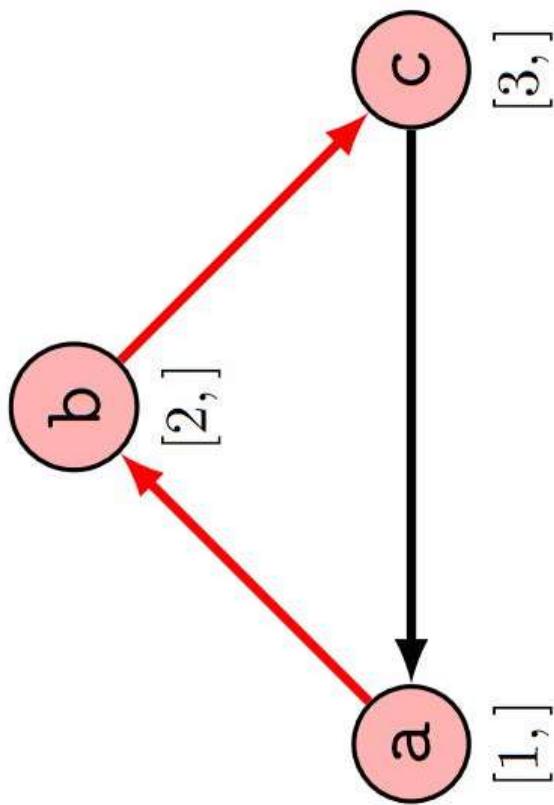
Cicli nei grafi orientati: esempio senza ciclo



Cicli nei grafi orientati: esempio senza ciclo

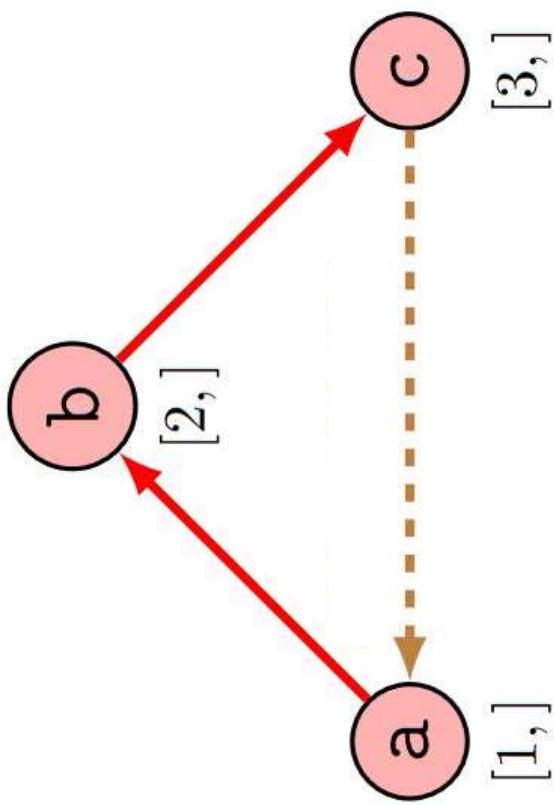


Cicli nei grafi orientati: esempio con ciclo



Fino a c è uguale all'esempio precedente.

Cicli nei grafi orientati: esempio con ciclo



Qui trovo un nodo che ha $dt[a] < dt[c]$ e $ft[a] == 0$, quindi c'è un ciclo.

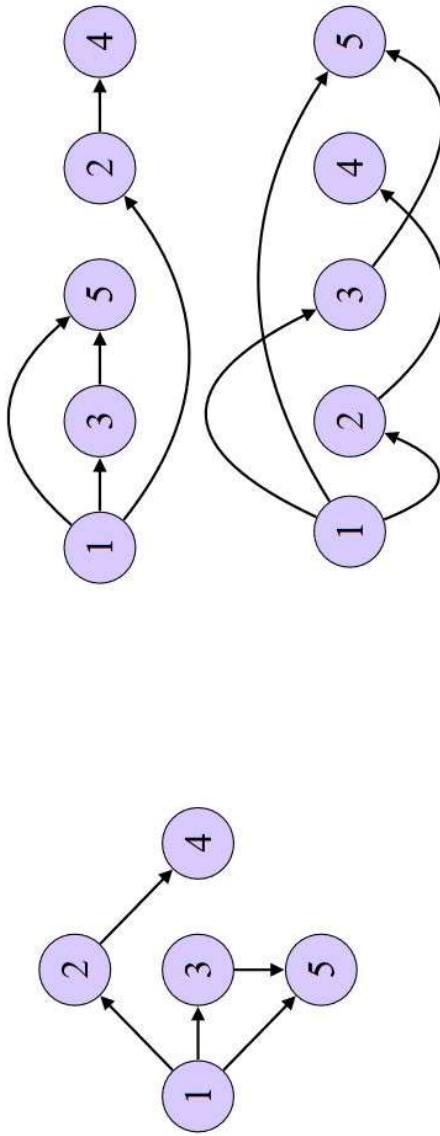
Grafi: ordinamento topologico

Ordinamento topologico: definizione

65

Dato un DAG G , un ordinamento topologico di G è un ordinamento lineare dei suoi nodi tale che se $(u, v) \in E$, allora u appare prima di v nell'ordinamento.

- Esistono più ordinamenti topologici
- Se il grafo contiene un ciclo, non esiste un ordinamento topologico.



Che problema è?

66

Quello dell'ordinamento topologico è sostanzialmente un problema di “trovare quale attività può essere eseguita”, ed ha diversi ambiti di applicazione:

- Ordine di valutazione delle celle in uno spreadsheet
- Risoluzione delle dipendenze nei gestori di pacchetti software
- Risoluzione degli ordini dei task in una WBS (ricordate GPOI?)
- “Dipendenza” delle risorse in una catena di forniture

Definizione del problema

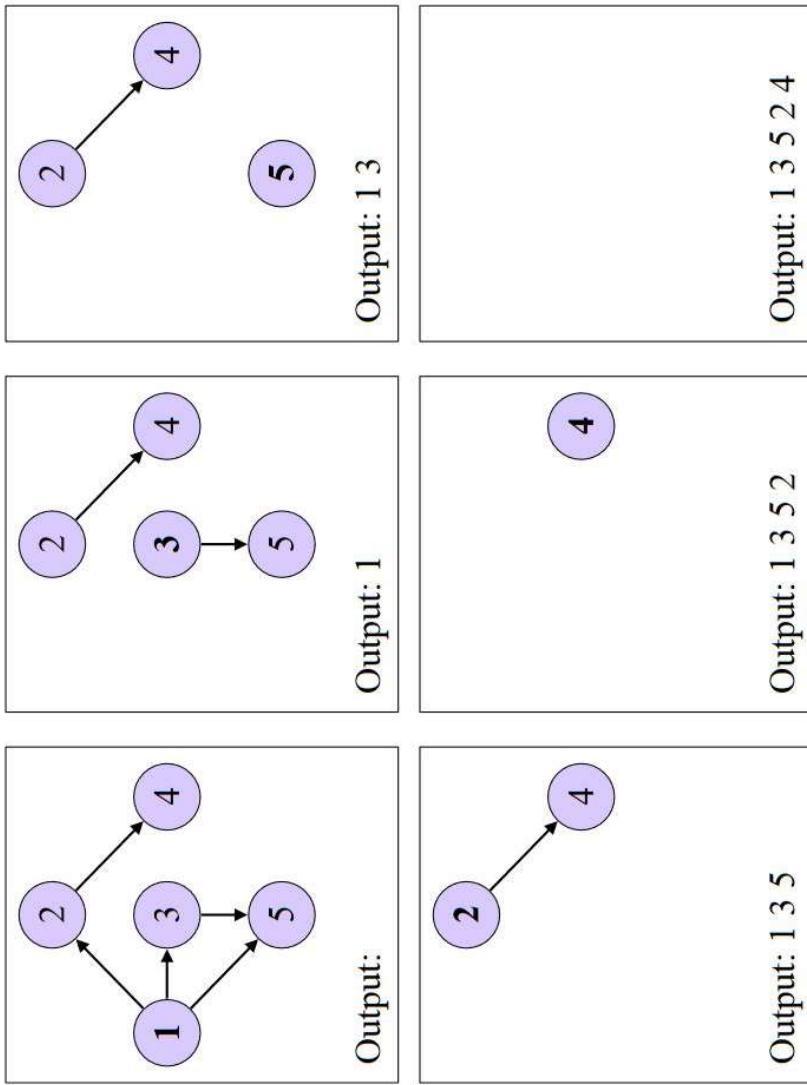
Scrivere un algoritmo che prende in input un DAG e ritorna un ordinamento topologico per esso, ossia una possibile sequenza di selezione dei nodi.

Soluzione semplice da capire... ...ma complicata da implementare.

1. Trovo un nodo senza archi entranti;
2. Aggiungo questo nodo nell'ordinamento e lo rimuovo dal grafo, insieme a tutti i suoi archi;
3. Ripeto questa procedura fino a quando tutti i nodi sono stati rimossi.

Esempio

69

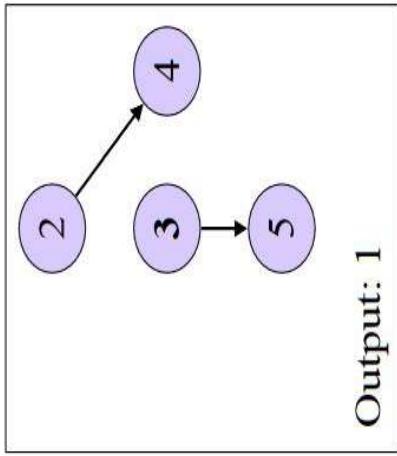


Alcune considerazioni

Alcuni passaggi sono “ambigui”, nel senso che si sarebbero potute fare altre scelte.

In questo caso, ad esempio, avremmo potuto scegliere il nodo 2 al posto del 3, e avere la sequenza {1, 2, 3, 4, 5}, ma anche {1, 2, 4, 3, 5}.

Certamente, non sarebbe stata valida {1, 2, 5, 4, 3}, perché 5 non sarebbe stato più in ordine (visto che esiste l’arco (3,5), 3 deve sempre apparire prima di 5).



L'idea migliore (usando DFS)

1. Scelgo un nodo k ;
2. Effettuo una DFS a partire da k , dove l'operazione di visita consiste nell'aggiungere il nodo in testa ad una lista, "a tempo di fine" (quando ho concluso la visita);
3. Se ci sono altri nodi non ancora visitati, ricomincio dal punto 1.
4. Al termine, restituisco la lista di nodi così ottenuta.
5. La sequenza dei nodi, ordinati per tempo decrescente di fine, sarà un ordinamento topologico.

Funziona, ma perché funziona?

Perché funziona?

Quando un nodo è marcato come “visitato”, significa che tutti i suoi discendenti sono stati scoperti e aggiunti alla lista.

Aggiungendolo in testa alla lista, il nodo è in ordine corretto (non ci sono più discendenti che possano venire inseriti prima di lui).

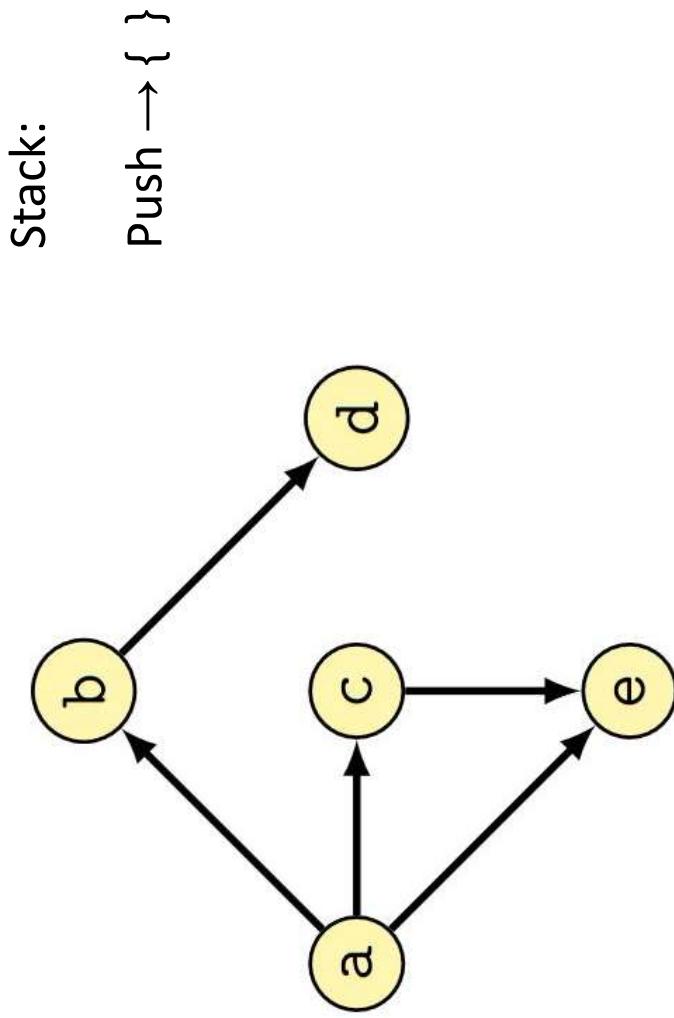
Ordinamento topologico: lo pseudocodice

73

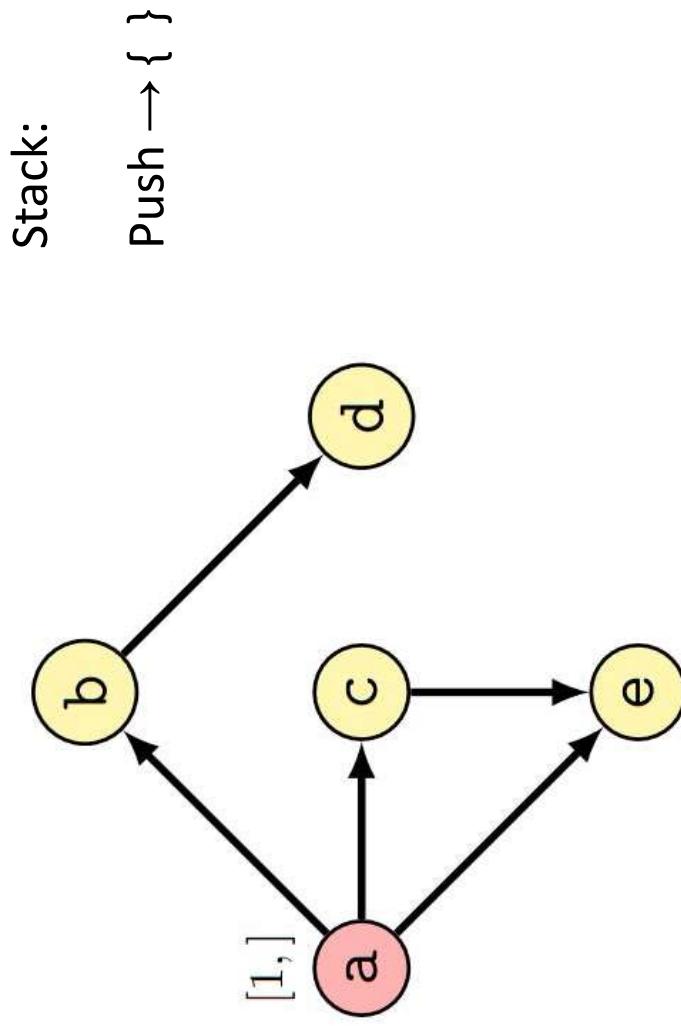
```
STACK topSort(GRAPH G)
STACK S = Stack()
boolean[] visited = boolean[G.size()]
foreach u ∈ G.V() do visited[u] = false
foreach u ∈ G.V() do
    if not visited[u] then
        ts-dfs(G, u, visited, S)
return S
```

```
ts-dfs(GRAPH G, NODE u, boolean[ ] visited, STACK S)
visited[u] = true
foreach v ∈ G.adj(u) do
    if not visited[v] then
        ts-dfs(G, v, visited, S)
S.push(u)
```

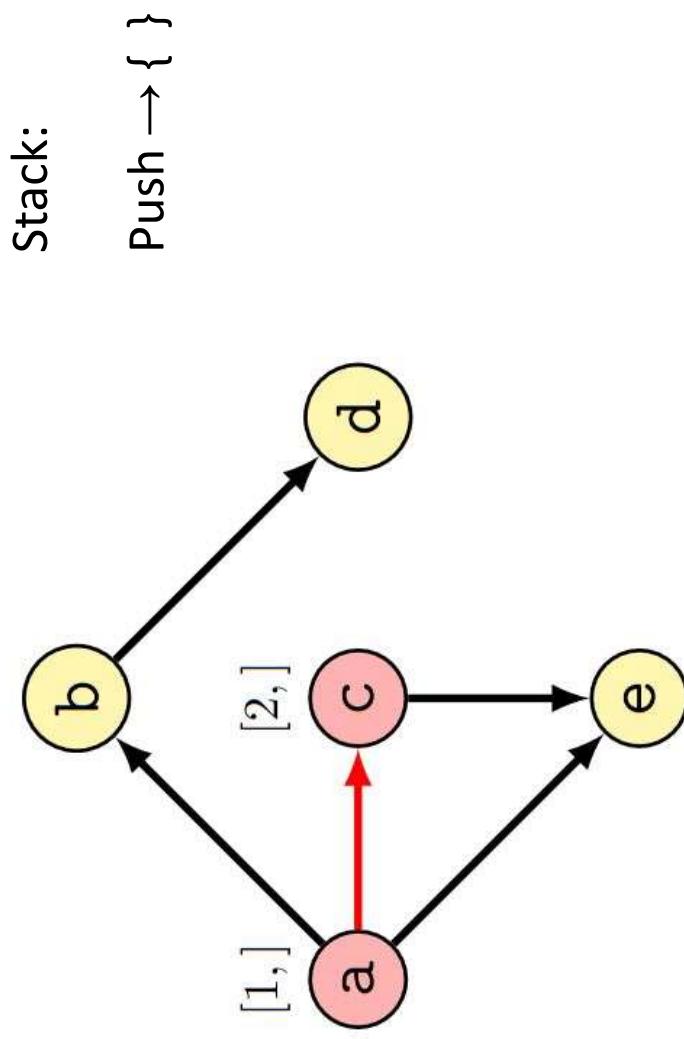
Esempio di esecuzione



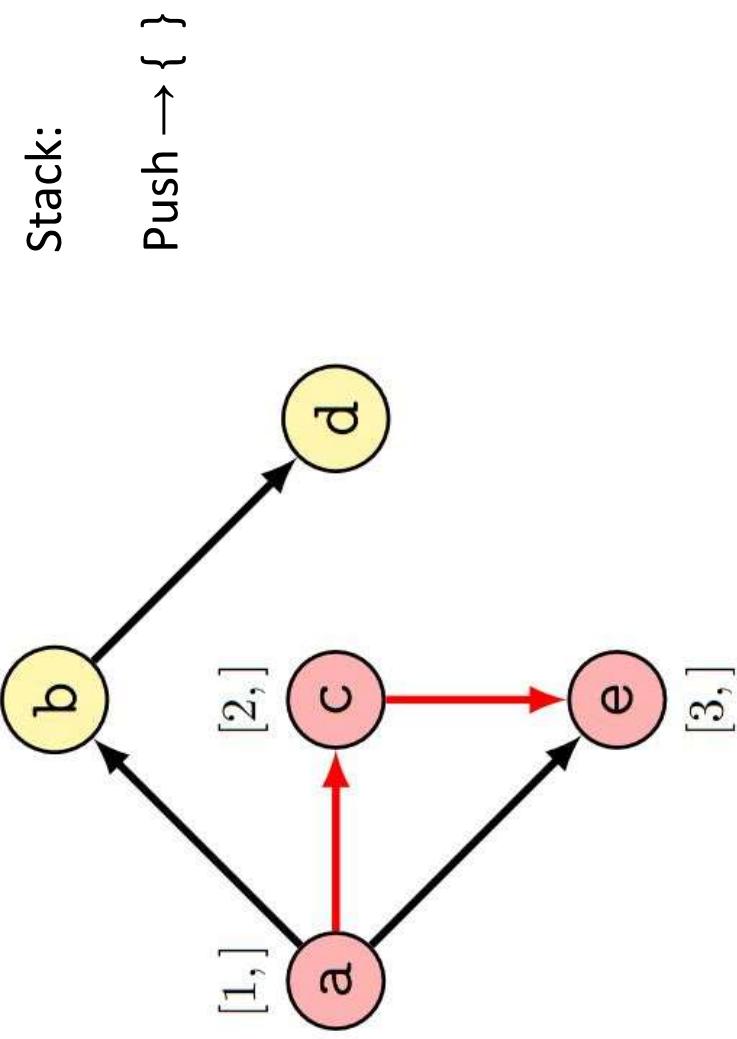
Esempio di esecuzione



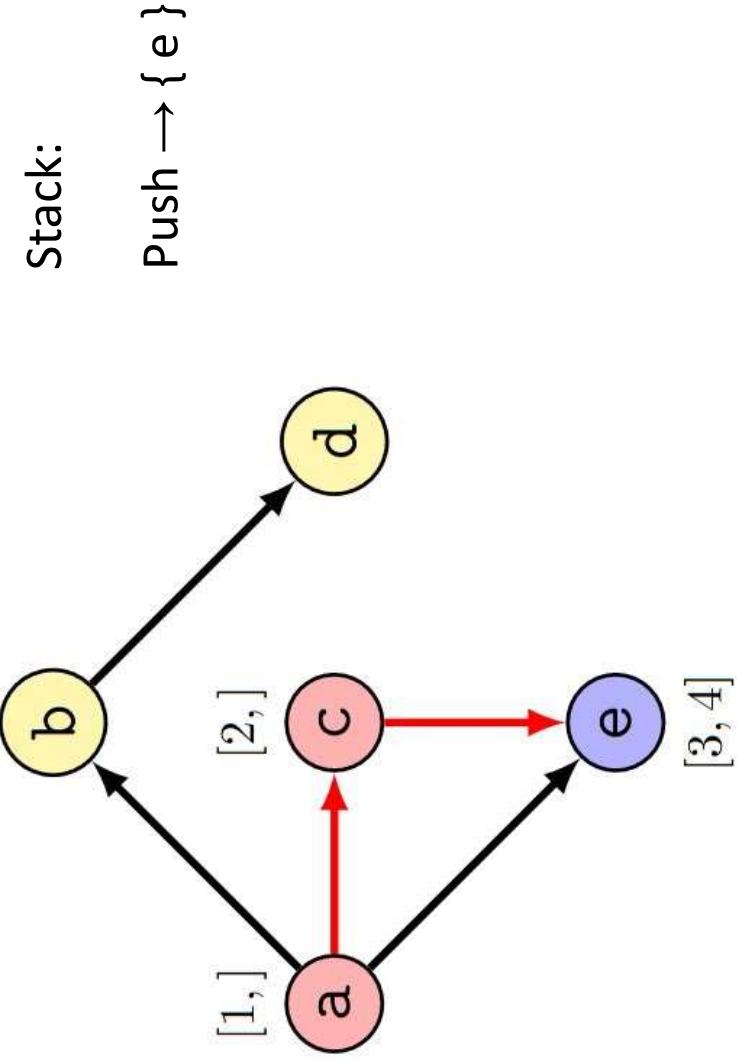
Esempio di esecuzione



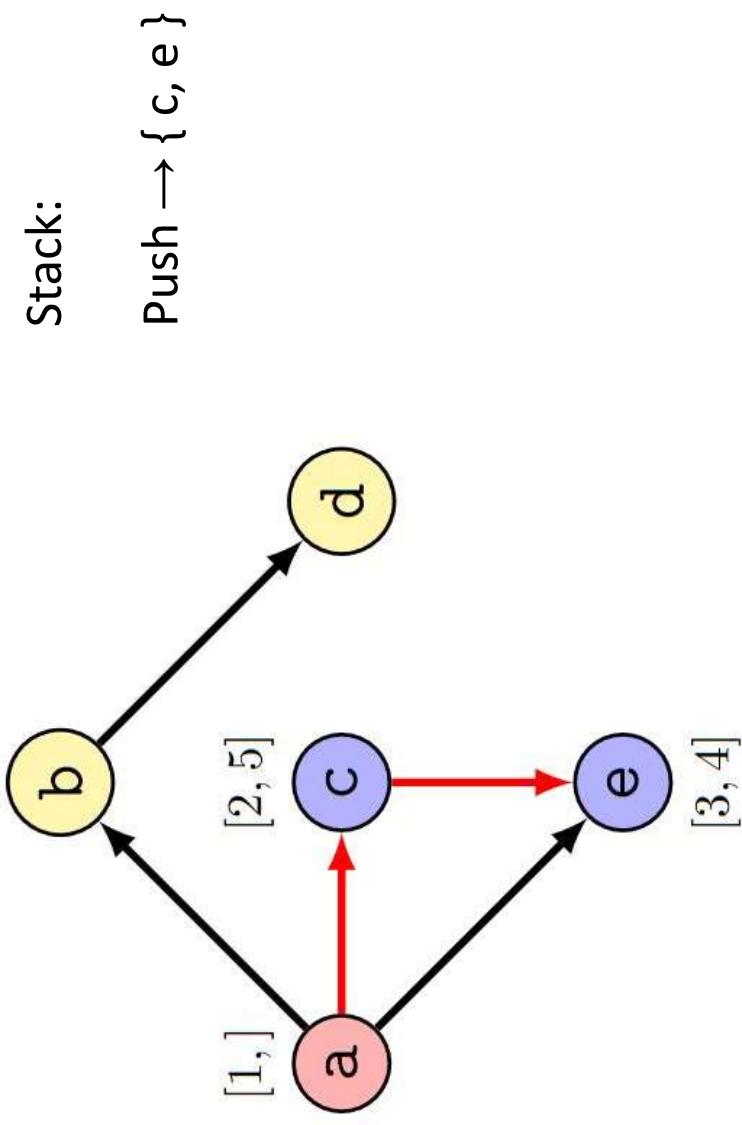
Esempio di esecuzione



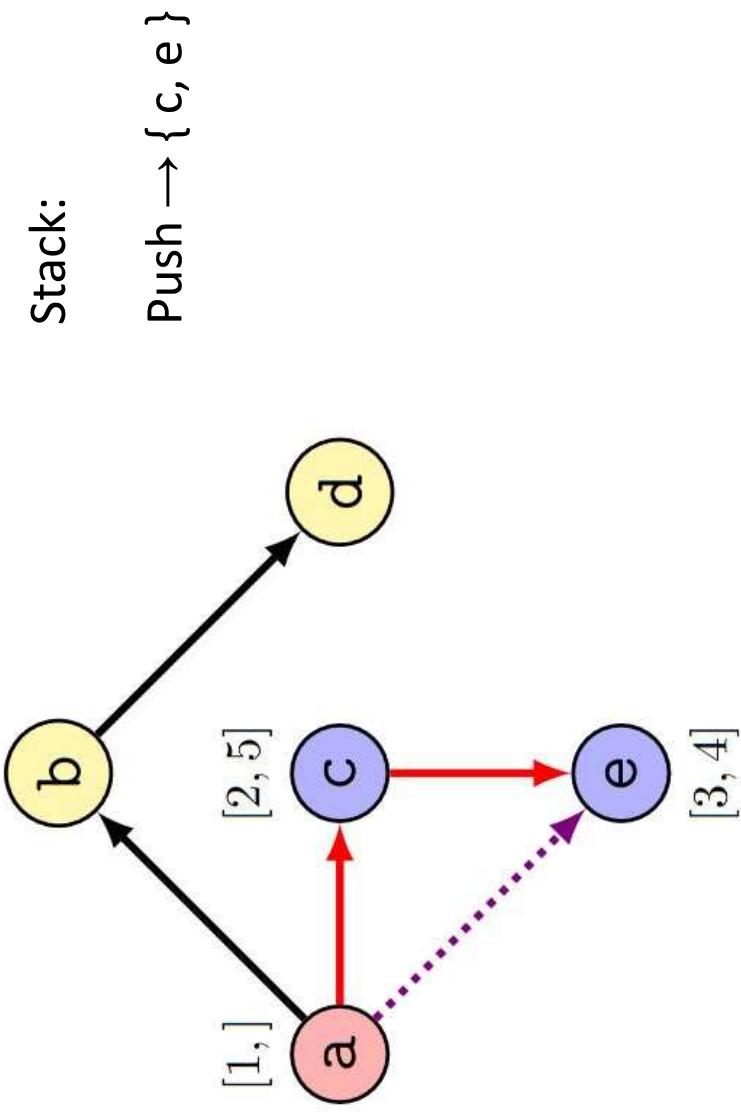
Esempio di esecuzione



Esempio di esecuzione



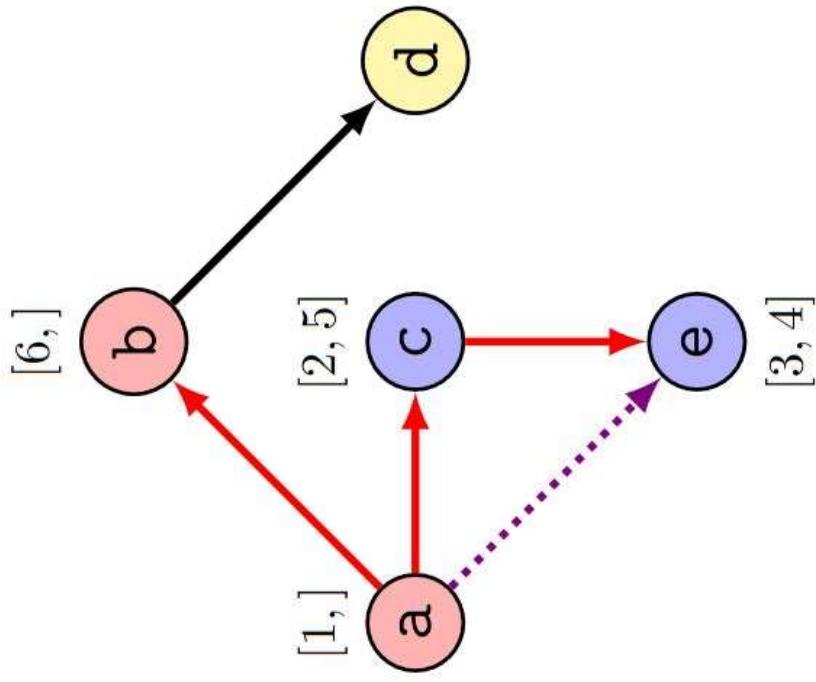
Esempio di esecuzione



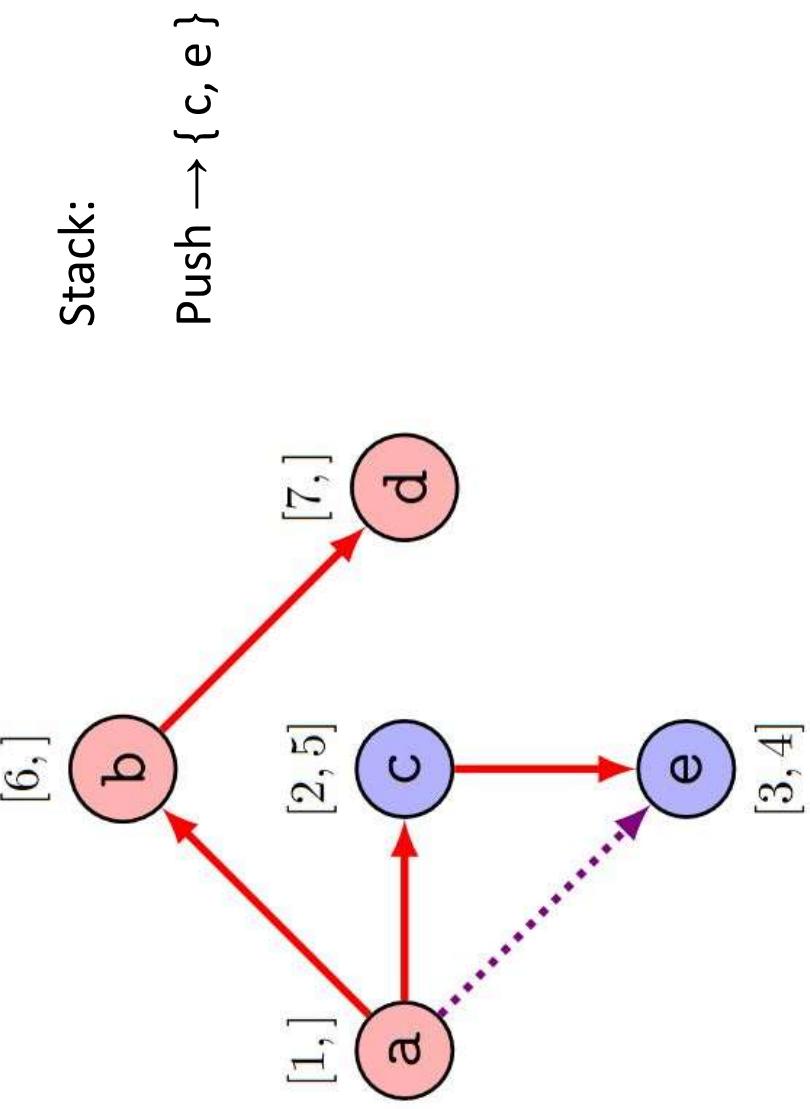
Esempio di esecuzione

Stack:

Push $\rightarrow \{ c, e \}$



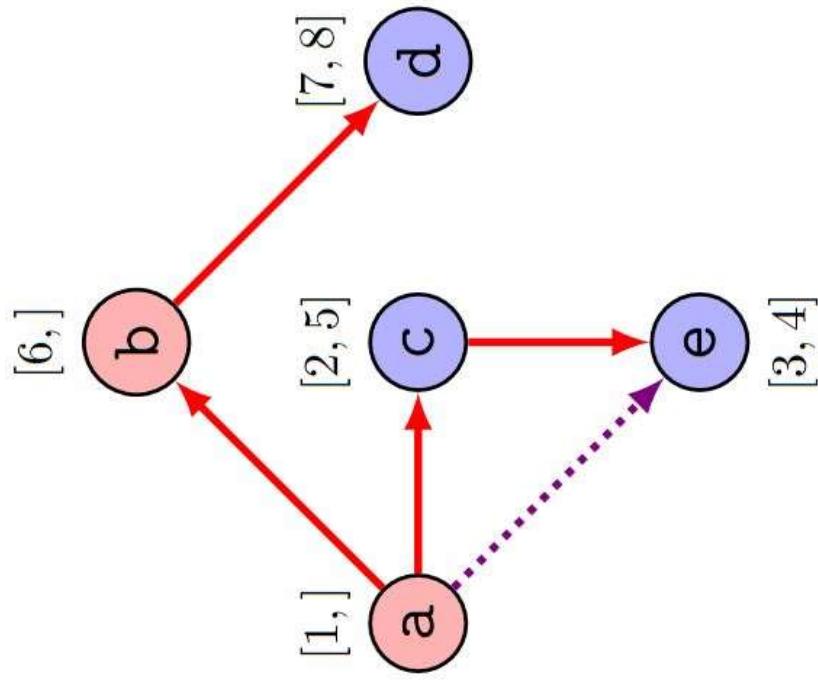
Esempio di esecuzione



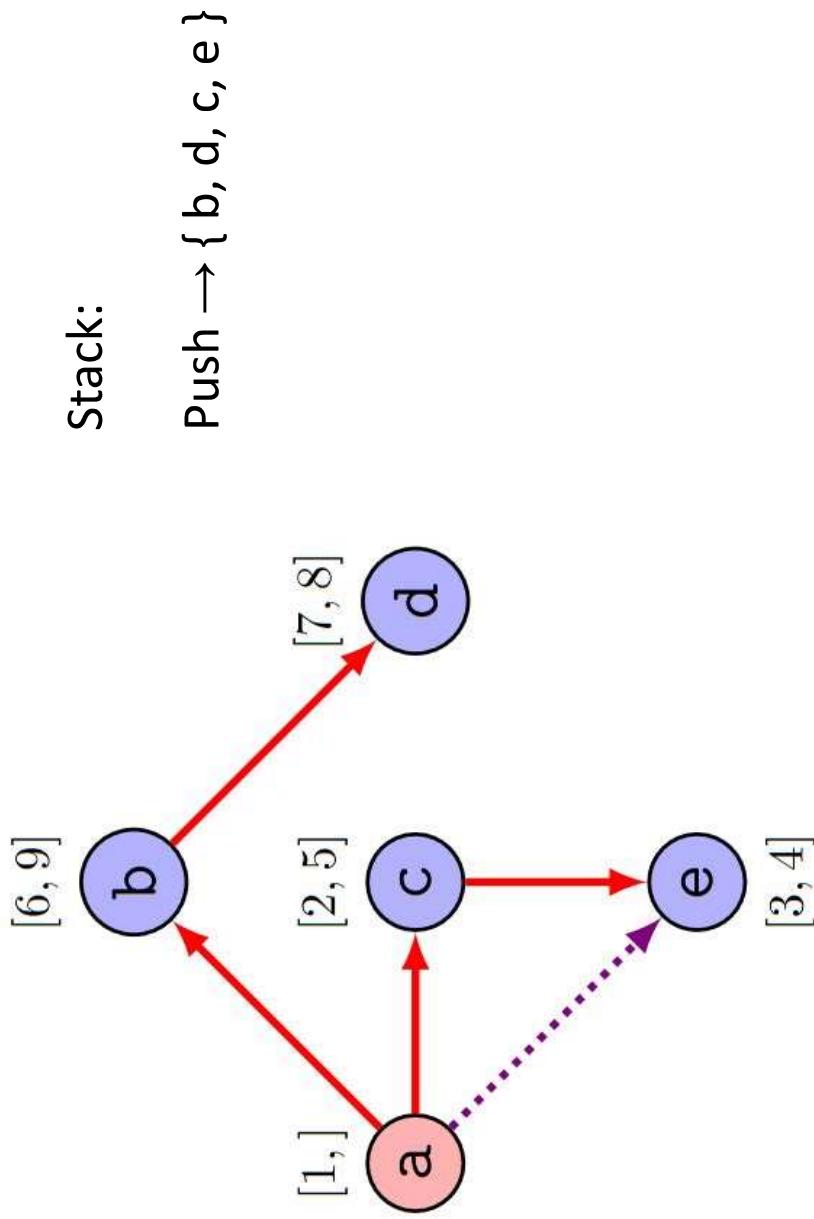
Esempio di esecuzione

Stack:

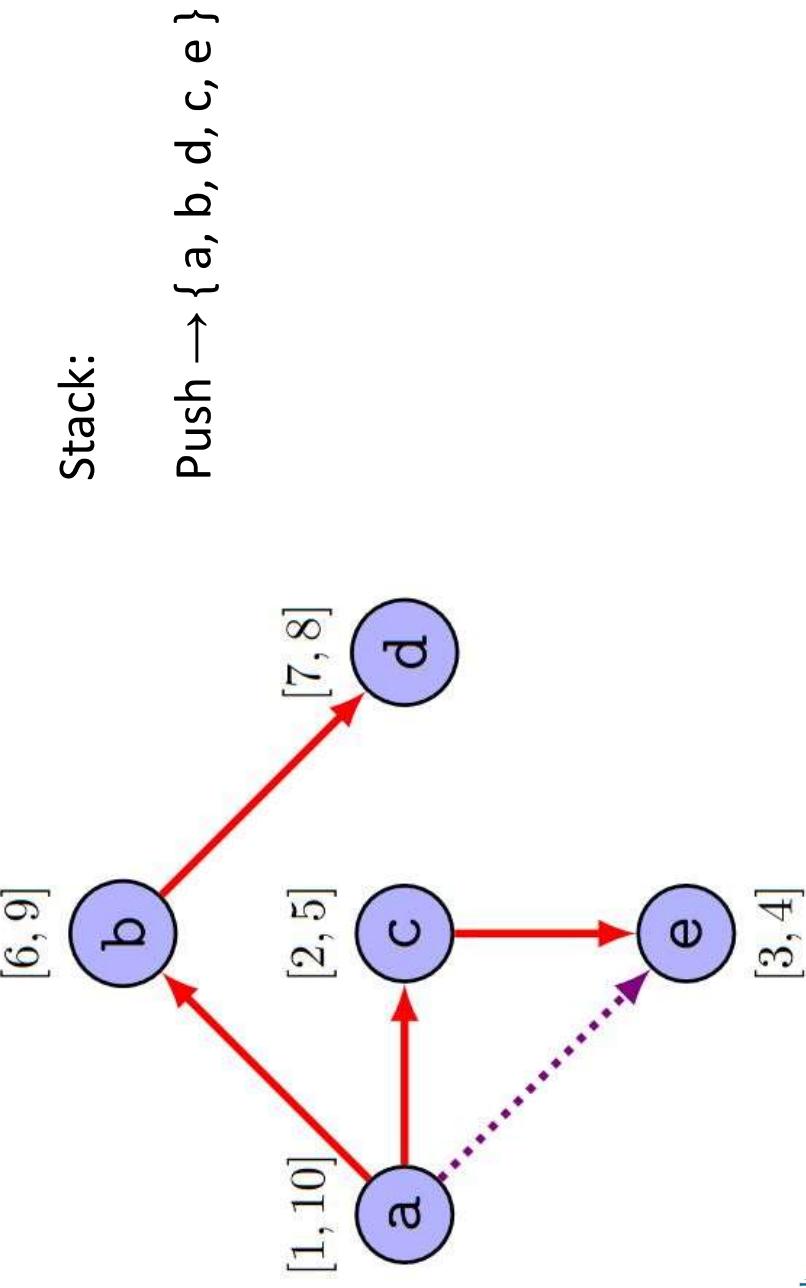
Push $\rightarrow \{ d, c, e \}$



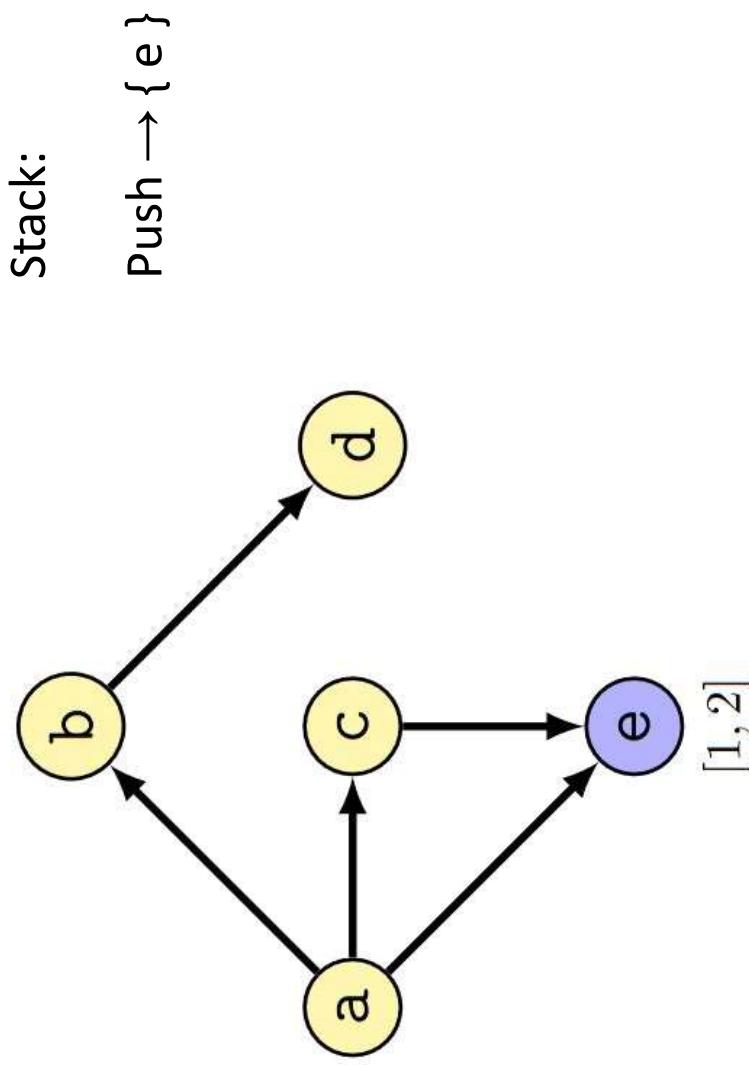
Esempio di esecuzione



Esempio di esecuzione



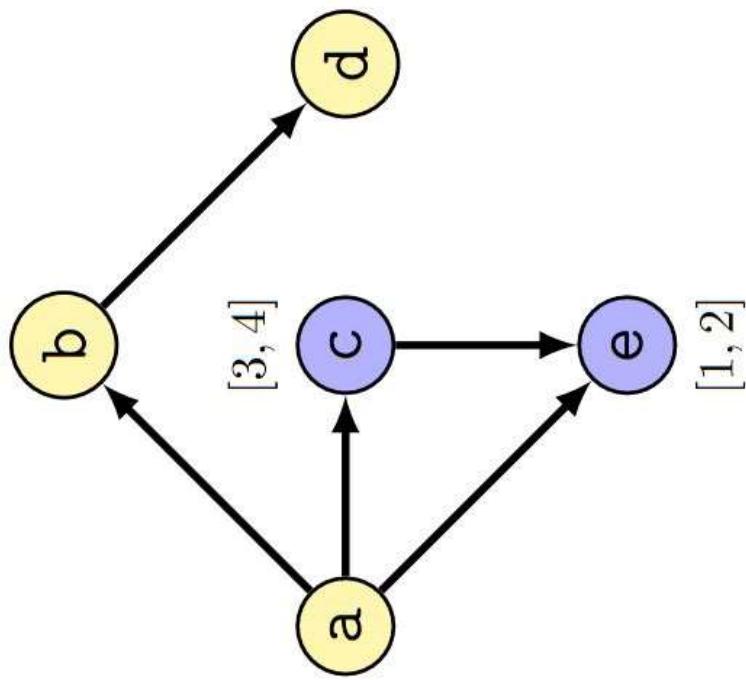
Esempio di esecuzione alternativa



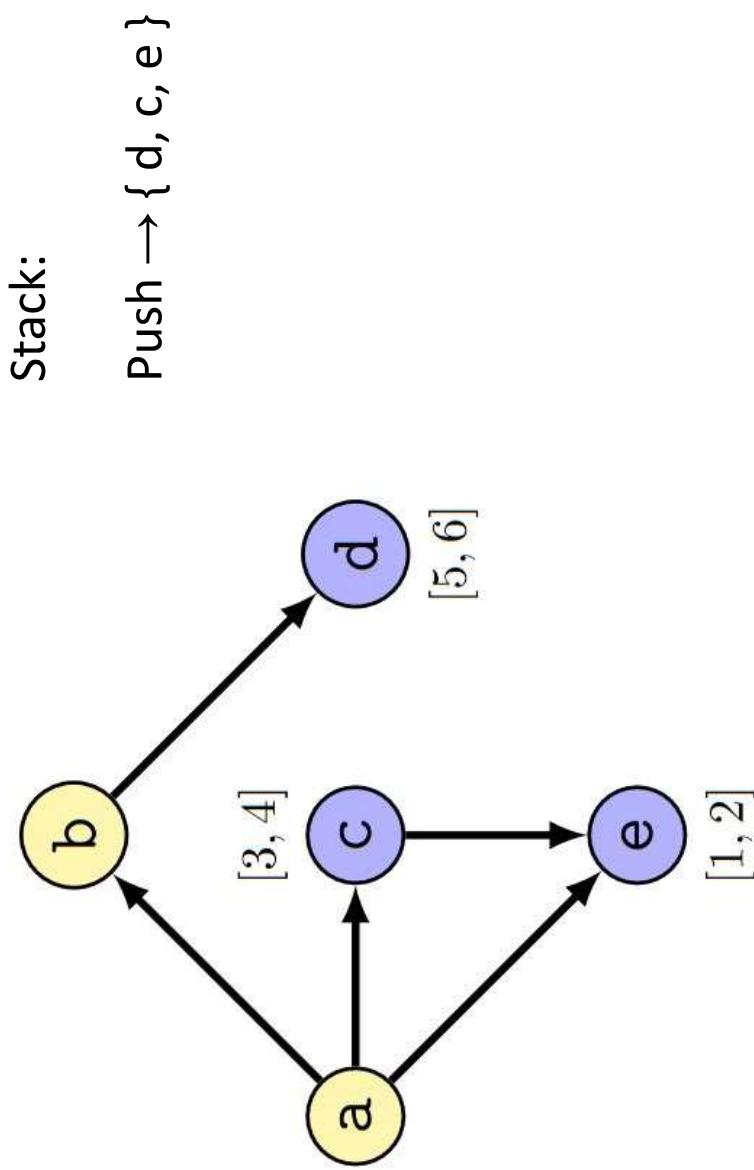
Esempio di esecuzione alternativa

Stack:

Push $\rightarrow \{ c, e \}$



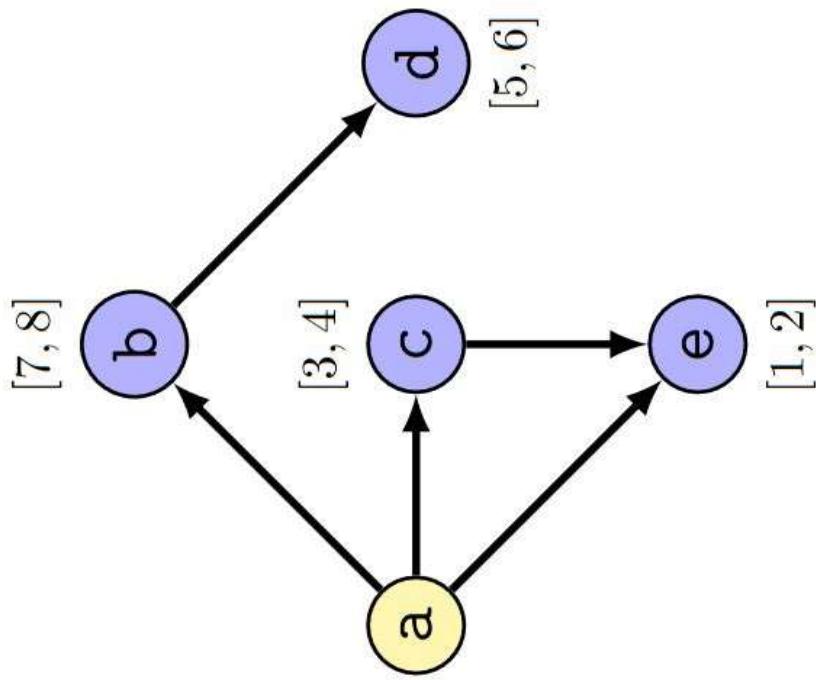
Esempio di esecuzione alternativa



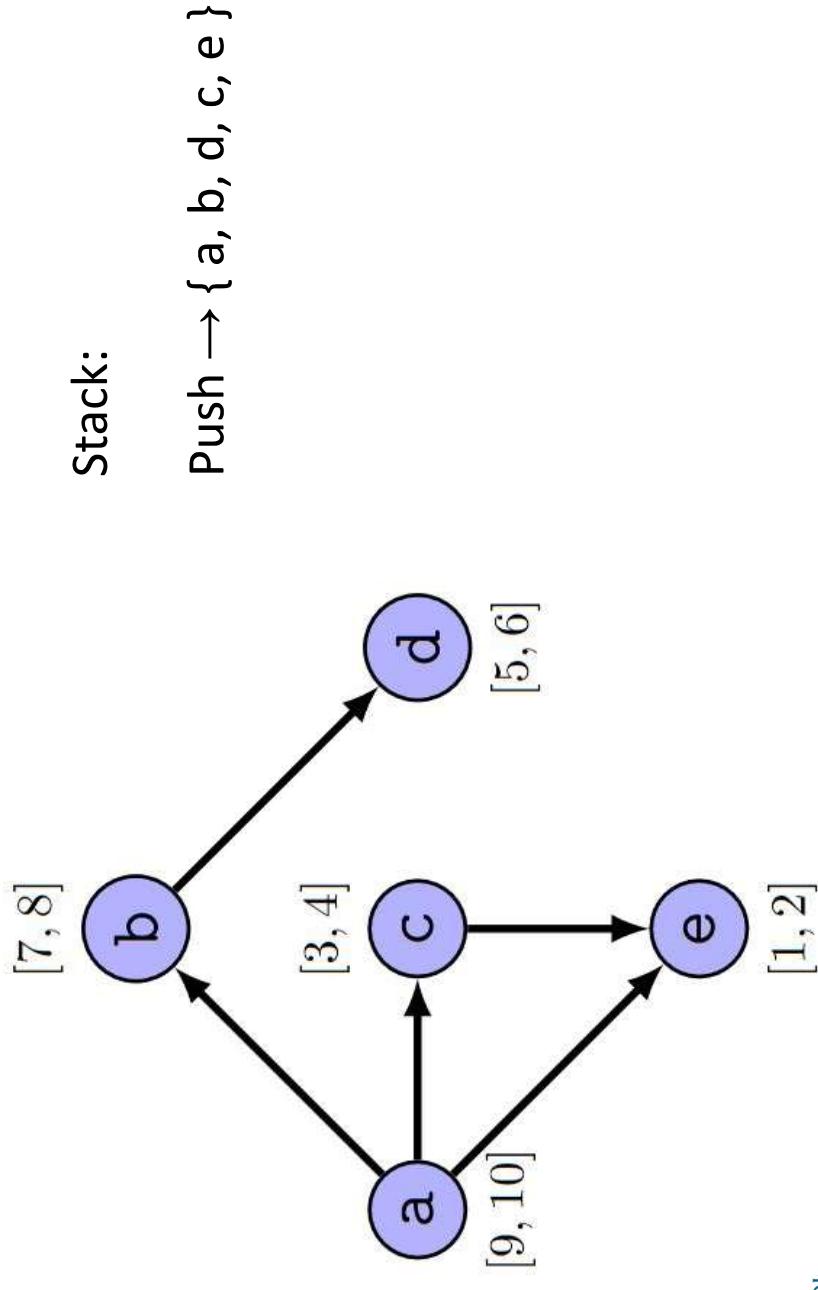
Esempio di esecuzione alternativa

Stack:

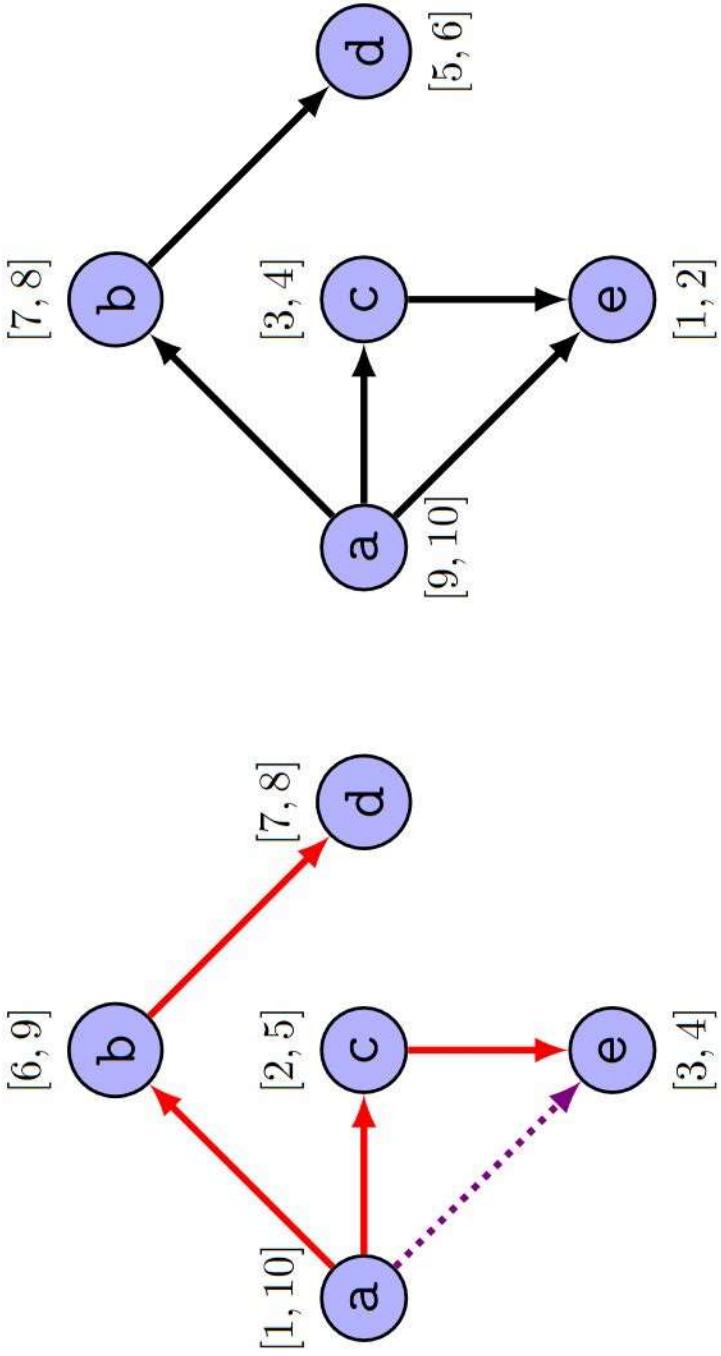
Push $\rightarrow \{ b, d, c, e \}$



Esempio di esecuzione alternativa



Esecuzioni a confronto



Perché tutto questo? Alcuni esempi:

- Trovare sottografi densi all'interno di un grafo più grande, ad esempio gruppi di persone che sono più strettamente collegati tra loro in un enorme gruppo di dati. Tipo Facebook o LinkedIn quando vi propongono “persone che potresti conoscere”.
- Modelli stradali: una rete stradale è un grafo diretto. Se il grafo non è SCC, i veicoli potrebbero entrare in una zona e non poterne mai uscire: dato che un programma di navigazione recupera solo una parte di tutta la mappa mondiale, potreste avere aree dove avete mappato i collegamenti interni ma non le strade che portano al di fuori di una certa area.

<https://stackoverflow.com/questions/11212676/what-are-strongly-connected-components-used-for>

Esercizi



Nel gioco dello Shanghai, un insieme di n bastoncini vengono lanciati su un tavolo, con il risultato che i bastoncini si sovrastano l'uno con l'altro. Un bastoncino può essere rimosso se nessun altro bastoncino lo sovrasta.

Una volta rimosso, è possibile che i bastoncini che erano sovrastati da esso possano essere rimossi. E' anche possibile tuttavia che ad un certo punto nessun bastoncino possa essere rimosso, in quanto sovrastato da altri bastoncini.

Sono dati in input due vettori X e Y , entrambi di dimensione m , contenenti numeri da 1 a n . I vettori vanno interpretati in questo modo: per ogni indice i , il bastoncino $X[i]$ sovrasta il bastoncino $Y[i]$.

Scrivere un algoritmo che prenda in input i vettori X , Y oltre alle dimensioni n ed m , e restituisca true se è possibile rimuovere tutti i bastoncini presenti, false altrimenti. Se è possibile, stampare un possibile piano di rimozione.