Definizione

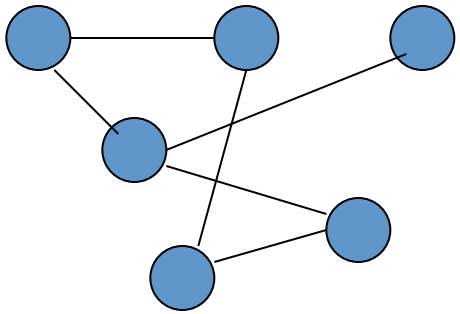
- Un grafo ***G = (V,E)*** è costituito da un insieme di vertici ***V*** ed un insieme di archi ***E*** ciascuno dei quali connette due vertici in ***V*** detti estremi dell'arco.
- Un grafo è orientato quando vi è un ordine tra i due estremi degli archi.

Definizione

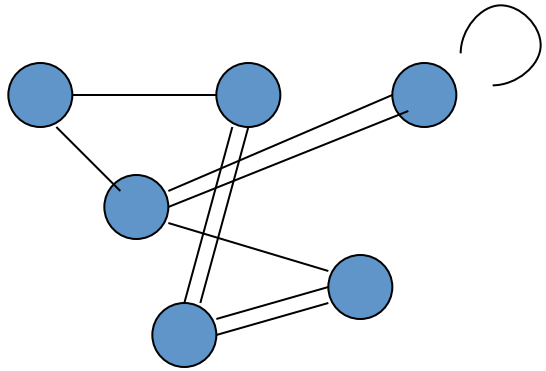
- Un *cappio* è un arco i cui estremi coincidono.
- Un grafo non orientato è *semplice* se
  - non ha cappi;
  - non ci sono due archi con gli stessi estremi.
- In caso contrario si parla di *multigrafo*.
- Salvo indicazione contraria noi assumeremo sempre che un grafo sia semplice.

Esempi

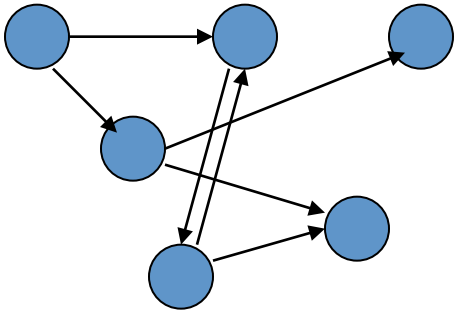
grafo semplice

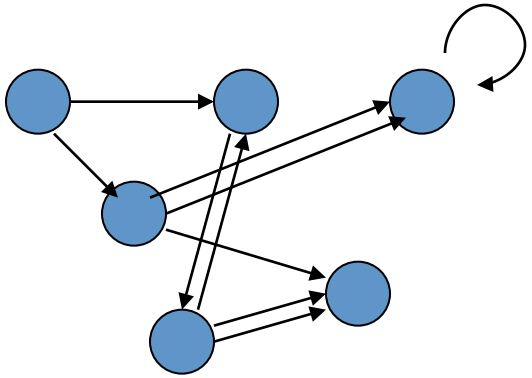


multigrafo



non orientato





orientato

Archi e orientamento

- Se un grafo è semplice possiamo identificare un arco con la coppia dei suoi estremi: $e = (u,v) \in E$.

- Quando $e = (u,v) \in E$ diciamo che l'arco e è *incidente* in u e in v.

- Se il grafo è orientato la coppia (u,v) è ordinata. In questo caso diciamo che l'arco e *esce* da u ed *entra* in v.

Grado di un vertice

- Il *grado* $\delta(v)$ del vertice v è il numero di archi incidenti in v.
- Se il grafo è orientato $\delta(v)$ si suddivide in
  - *grado entrante* $\delta^-(v)$: numero di archi entranti in v (in-degree)
  - *grado uscente* $\delta^+(v)$: numero di archi uscenti da v (out-degree)
- Se $e = (u,v) \in E$ diciamo che il vertice v è *adiacente* al vertice u.
  Se il grafo non è orientato la relazione di adiacenza è simmetrica.

Cammini e cicli

- Un cammino π di lunghezza k dal vertice u al vertice v in un grafo $G = (V,E)$ è una sequenza di $k+1$ vertici

$$\pi = x_0, x_1, \ldots, x_k$$

tali che $x_0 = u$, $x_k = v$ e $(x_{i-1}, x_i) \in E$ per $i = 1,\ldots,k$.

Il cammino $\pi = x_0$ ha lunghezza $k = 0$.

- Se $k > 0$ e $x_0 = x_k$ diciamo che il cammino è *chiuso*

- Un *cammino semplice* è un cammino i cui vertici $x_0, x_1, \ldots, x_k$ ($k > 0$) sono tutti distinti con la possibile eccezione di $x_0 = x_k$, nel qual caso esso è un *ciclo*.

- Un ciclo di lunghezza $k = 1$ è un *cappio*. Un *grafo aciclico* è un grafo che non contiene cicli.

Raggiungibilità e connessione

- Quando esiste almeno un cammino dal vertice ***u*** al vertice ***v*** diciamo che il vertice ***v*** è *accessibile* o *raggiungibile* da ***u***.

- Un grafo non orientato si dice *connesso* se esiste almeno un cammino tra ogni coppia di vertici.

- Le *componenti connesse* di un grafo sono le classi di equivalenza dei suoi vertici rispetto alla relazione di accessibilità.

Connessione forte

- Un grafo orientato si dice *fortemente connesso* se esiste almeno un cammino da ogni vertice ***u*** ad ogni altro vertice ***v***

- Le *componenti fortemente connesse* di un grafo orientato sono le classi di equivalenza dei suoi vertici rispetto alla relazione di mutua accessibilità.

Completezza

- Un grafo si dice completo se esiste un arco per ogni coppia di vertici.
- Il grafo completo con *n* vertici è denotato K_n
- Un grafo $G = (V,E)$ è *bipartito* se esiste una partizione (V_1, V_2) dell'insieme dei vertici V, tale che tutti gli archi hanno come estremi un vertice di V_1 ed un vertice di V_2.
- Un grafo bipartito è completo ($K_{n,m}$) se esiste un arco per ogni coppia (della bipartizione) di vertici.

Sottografi

- Un *sottografo* del grafo $G = (V,E)$ è un grafo $G' = (V',E')$ tale che $V' \subseteq V$ e $E' \subseteq E$.

- Il sottografo di $G = (V,E)$ *indotto* da $V' \subseteq V$ è il grafo $G' = (V',E')$ tale che
$$E' = \{ (u,v) : (u,v) \in E \text{ e } u,v \in V' \}.$$

Rappresentazione dei Grafi

- Vi sono due modi standard per rappresentare un grafo $G = (V,E)$: con le liste delle adiacenze o con la matrice delle adiacenze.

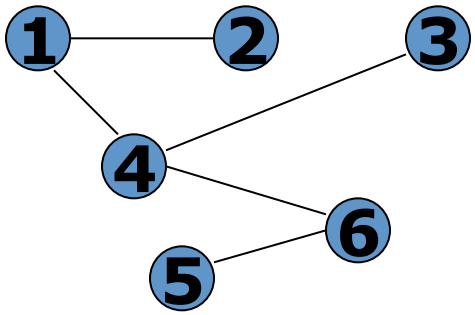
- La rappresentazione di $G = (V,E)$ mediante *liste delle adiacenze* è costituita da una lista $Adj[u]$ per ogni vertice $u \in V$ in cui vengono memorizzati i vertici adiacenti al vertice u (ossia tutti i vertici v tali che $(u,v) \in E$).

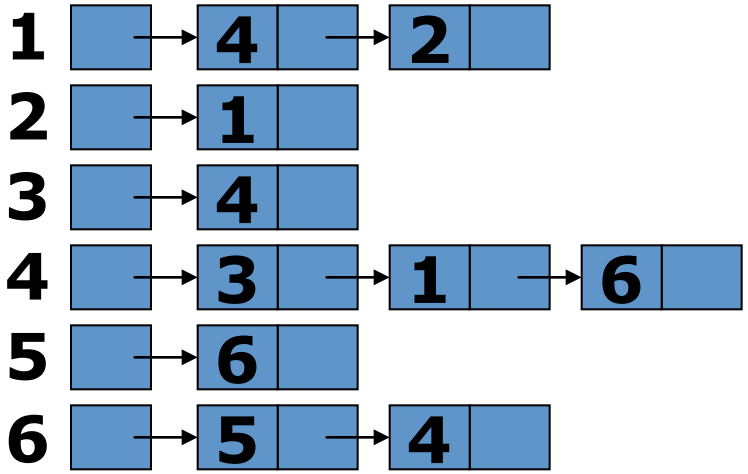
Rappresentazione dei Grafi

- Nella rappresentazione di $G = (V,E)$ mediante *matrice delle adiacenze* assumiamo che i vertici siano numerati $1,2,\ldots,|V|$ in qualche modo arbitrario. La rappresentazione è quindi costituita da una matrice booleana $A = (a_{ij})$ tale che

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{se } (i,j) \notin E \end{cases}$$

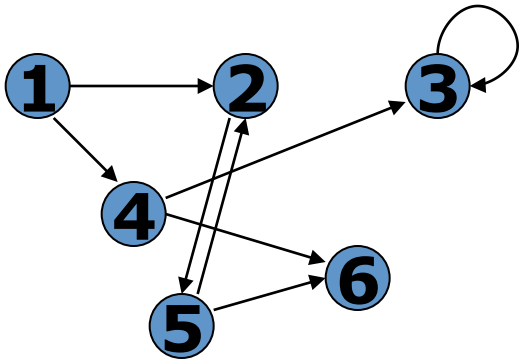
Esempio: grafo non orientato

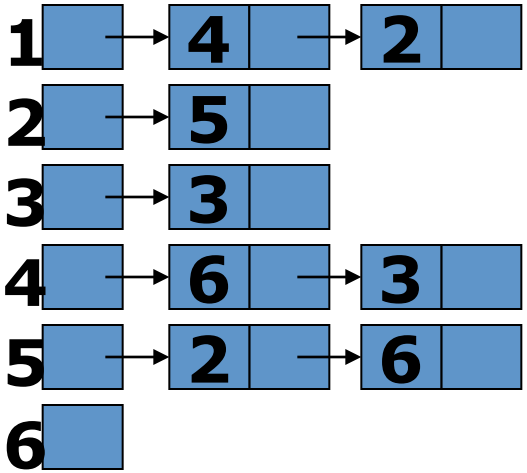




	1	2	3	4	5	6
1	0	1	0	1	0	0
2	1	0	0	0	0	0
3	0	0	0	1	0	0
4	1	0	1	0	0	1
5	0	0	0	0	0	1
6	0	0	0	1	1	0

Esempio: grafo orientato





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

Complessità spaziale della rappresentazione dei grafi

- Con liste di adiacenza: $O(|V|+|E|)$. Perché?
- Con matrice di adiacenza: $O(|V|^2)$. Perché?

Visita di un grafo: BFS

- Dato un grafo $G = (V,E)$ ed un vertice particolare $s \in V$, detto *sorgente*, la *visita in ampiezza* partendo dalla sorgente s visita sistematicamente il grafo per scoprire tutti i vertici che sono raggiungibili da s.

- Nel contempo calcola la *distanza* di ogni vertice del grafo dalla sorgente s (lunghezza minima di un cammino dalla sorgente al vertice).

- Essa produce anche un *albero di ricerca in ampiezza*, detto *albero* BF, i cui rami sono cammini di lunghezza minima.

Visita di un grafo: BFS

- La visita viene detta *in ampiezza* perché l'algoritmo espande uniformemente la frontiera tra i vertici scoperti e quelli non ancora scoperti. Ovvero scopre tutti i vertici a distanza ***k*** da ***s*** prima di scoprire quelli a distanza ***k*+1** .

- Per mantenere traccia del punto a cui si è arrivati nell'esecuzione dell'algoritmo i vertici sono colorati
  - ***bianco*** (vertici non ancora raggiunti),
  - ***grigio*** (vertici raggiunti che stanno sulla frontiera) e
  - ***nero*** (vertici raggiunti che non stanno sulla frontiera)

Visita di un grafo: BFS

- I vertici adiacenti ad un vertice nero possono essere soltanto neri o grigi mentre quelli adiacenti ad un vertice grigio possono essere anche bianchi

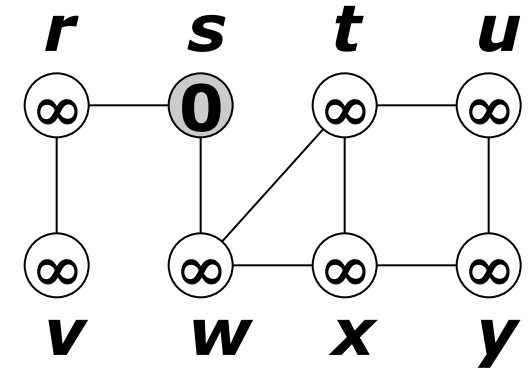
- L'algoritmo costruisce un albero, detto albero BF, che all'inizio contiene soltanto la radice ***s***. Quando viene scoperto un vertice bianco ***v*** mediante un arco **(*u,v*)** che lo connette ad un vertice ***u*** scoperto precedentemente, il vertice ***v*** e l'arco **(*u,v*)** vengono aggiunti all'albero. Il vertice ***u*** viene detto *padre* di ***v.***

- Il seguente algoritmo di vista in ampiezza assume che il grafo sia rappresentato con liste di adiacenza ed usa una coda ***Q*** di vertici in cui memorizza la frontiera.

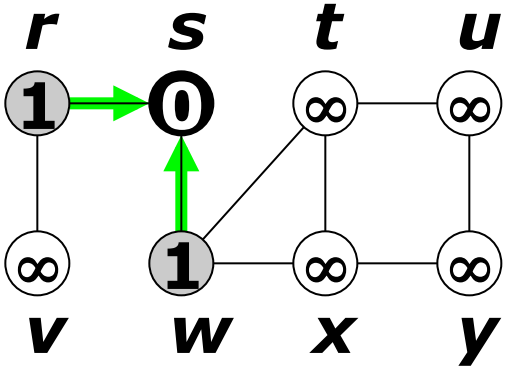
L'algoritmo: BFS

```
BFS(G, s)
  for v ∈ V[G] do
    color[v] ← bianco, d[v] ← ∞, p[v] ← nil
  color[s] ← grigio, d[s] ← 0
  Enqueue(Q, s)

  while not Empty(Q) do
    u ← First(Q)
    for v ∈ Adj[u] do
      if color[v] = bianco then
        color[v] ← grigio, d[v] ← d[u] + 1, p[v] ← u
        Enqueue(Q,v)
    Dequeue(Q)
    color[u] ← nero
```

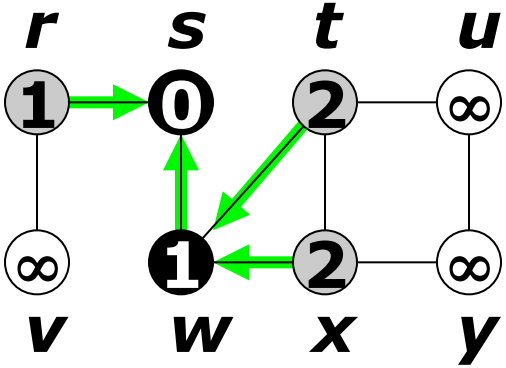
BFS: Esempio



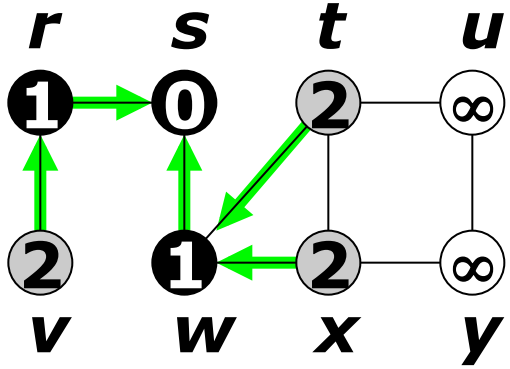
Q | s |
0



Q | w | r |
1 1



Q | r | t | x |
1 2 2



Q | t | x | v |
2 2 2

BFS: Complessità

- Valutiamo la complessità di ***BFS*** in funzione del numero di vertici **|*V*|** e archi **|*E* |** del grafo ***G* = (*V,E*)**.

- L'inizializzazione richiede tempo ***O*(|*V*|)** dovendo percorrere tutti i vertici del grafo.

- Dopo l'inizializzazione nessun vertice viene più colorato ***bianco*** e questo ci assicura che ogni vertice verrà inserito nella coda al più una sola volta. Quindi il ciclo **while** viene eseguito al più ***n*** volte.

- Il tempo richiesto per il ciclo **while** è dunque ***O*(|*V*|)** più il tempo richiesto per eseguire i cicli **for** interni.

BFS: Complessità

- I cicli **for** interni percorrono una sola volta le liste delle adiacenze di ciascun vertice visitato.

- Siccome la somma delle lunghezze di tutte le liste è $O(|E|)$ possiamo concludere che la complessità è

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

Con abuso di notazione, scriveremo anche

$$O(V+E)$$

BFS: Riassumendo

- ***BFS*** è un algoritmo per la visita in ampiezza di un grafo (orientato o no) ***G = (V,E)*** a partire da un vertice sorgente ***s,*** la cui complessità è ***O(|V|+|E|)***
  (E se il grafo è rappresentato mediante matrice di adiacenza?).

- Produce:

  - Una funzione ***d:V->{0,1,…, |V|-1}*** che assegna ad ogni nodo raggiungibile dal vertice sorgente ***s*** la sua distanza da ***s***

  - Un albero BF radicato in ***s***, rappresentato mediante una funzione ***p*** che assegna ad ogni ***v*** nodo il suo genitore ***p(v)*** nell'albero.

Visita di un grafo: DFS

- Nella *visita in profondità* a partire da un dato vertice, si esplorano sempre gli archi uscenti dall'ultimo vertice ***u*** raggiunto.

- Se viene scoperto un nuovo vertice ***v,*** ci si sposta su tale vertice. Se tutti gli archi uscenti da un vertice ***u*** portano a vertici già scoperti si torna indietro e si riprende esplorando gli altri archi uscenti dal vertice da cui ***u*** è stato scoperto.

- Si continua finché tutti i vertici raggiungibili dal vertice iniziale scelto sono stati scoperti. Se non tutti i vertici del grafo sono stati raggiunti, il procedimento viene ripetuto partendo da un qualunque vertice non ancora raggiunto.

Visita di un grafo: DFS

- Quando viene scoperto un nuovo vertice ***v*** visitando la lista delle adiacenze di un vertice ***u*** si memorizza un puntatore da ***v*** ad ***u***.

- Si costruisce così, in generale, non un albero, ma una "foresta" (insieme di alberi disgiunti).

- Come nella BFS, i vertici sono colorati di

  - ***bianco*** (vertici non ancora raggiunti)
  - ***grigio*** (vertici *scoperti*)
  - ***nero*** (vertici *finiti*, la cui lista delle adiacenze è stata completamente esplorata).

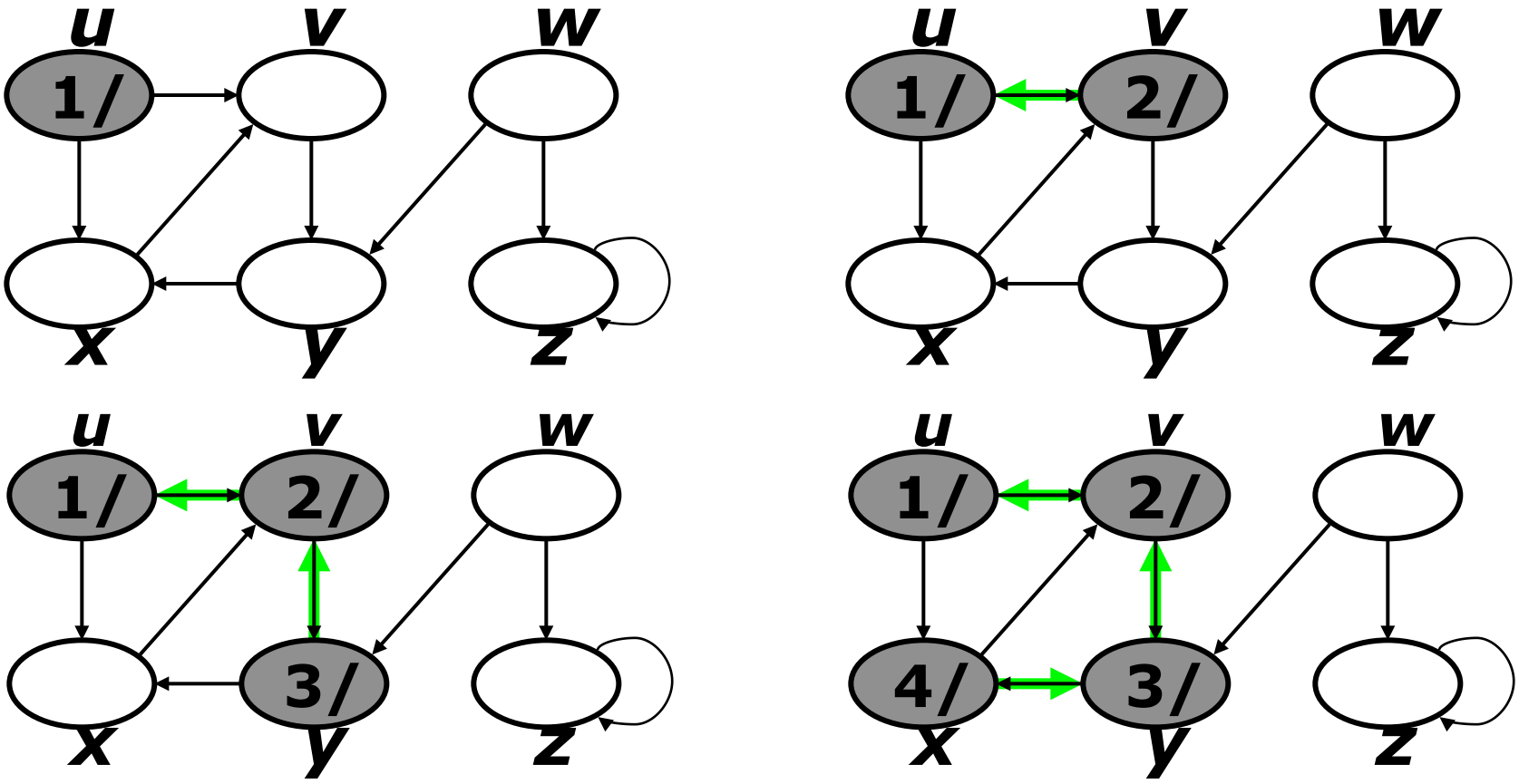
DFS: marcatempi

- La DFS utilizza due *marcatempi* su ogni vertice ***u***.
  - ***d*[*u*]:** tempo di scoperta, quando il vertice è colorato di ***grigio***
  - ***f*[*u*]:** tempo di fine visita, quando il vertice è colorato di ***nero***.

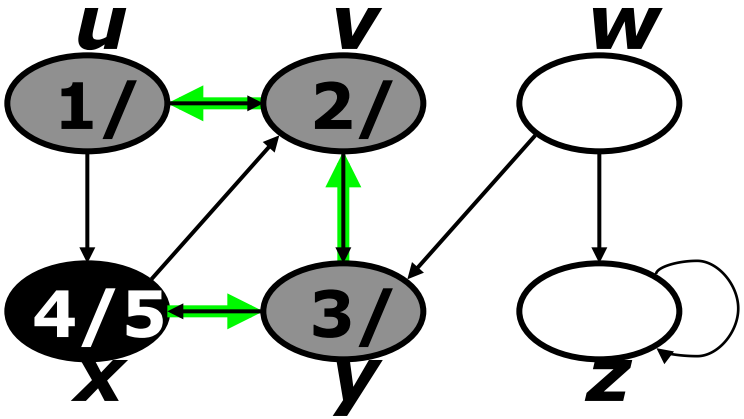
L'algoritmo: DFS

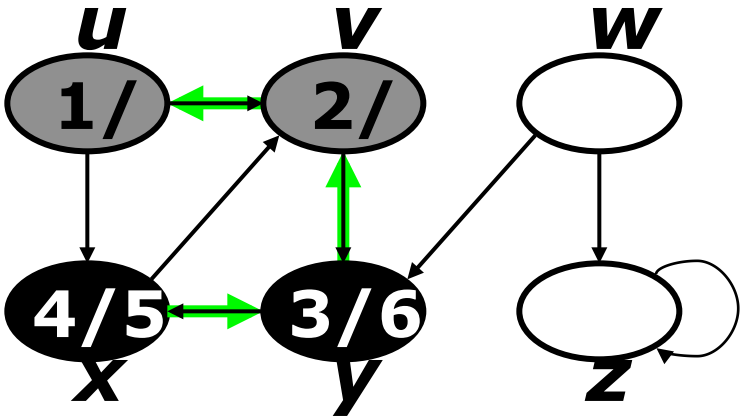
```
DFS(G)
  for v ∈ V[G] do
    color[v] ← bianco, p[v] ← nil
  t ← 1
  for v ∈ V[G] do
    if color[v] = bianco then
      DFSric(v, t)

DFSric(u, t)
  color[u] ← grigio, d[u] ← t , t ← t + 1
  for v ∈ Adj[u] do
    if color[v] = bianco then
      p[v] ← u
      DFSric(v, t)
  color[u] ← nero, f[u] ← t , t ← t + 1
```

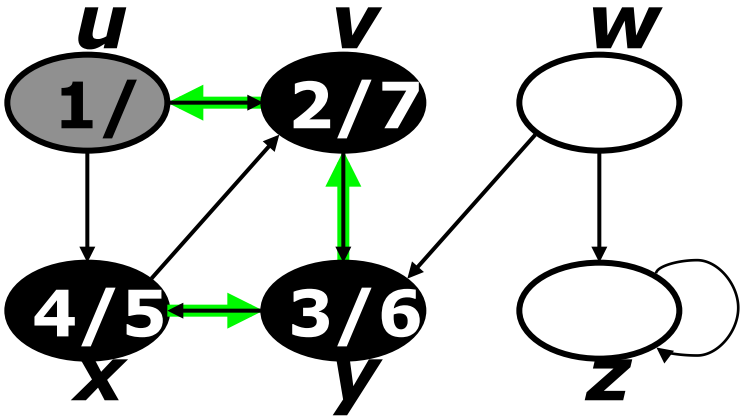
DFS: Esempio

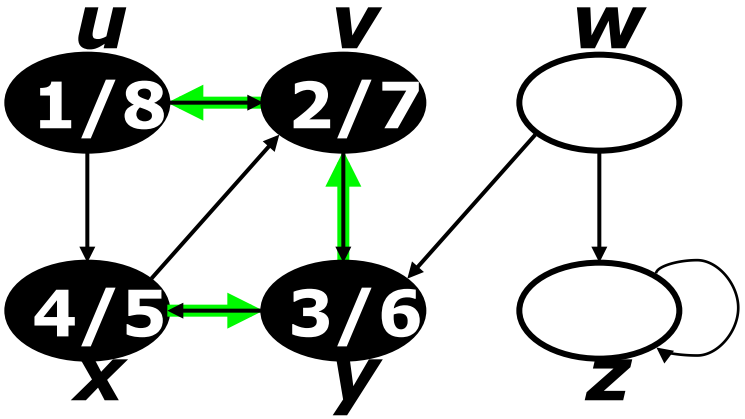


DFS: Esempio

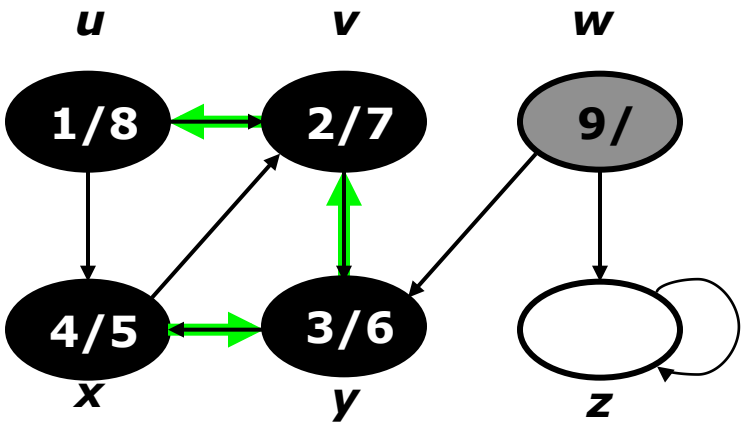

u
v
w
1/
2/
4/5
3/
x
y
z

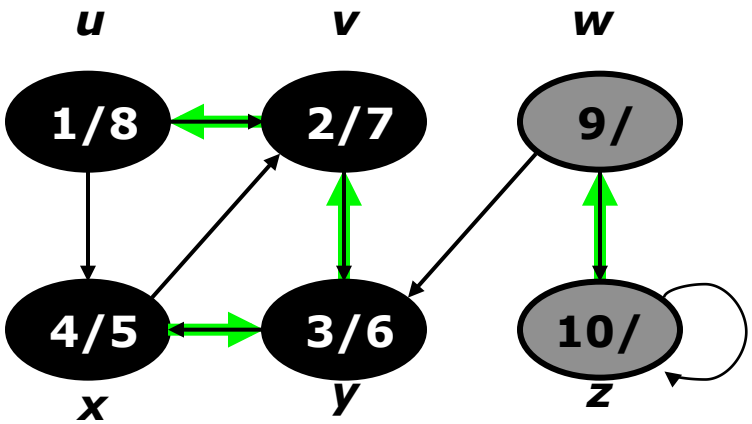

u
v
w
1/
2/
4/5
3/6
x
y
z

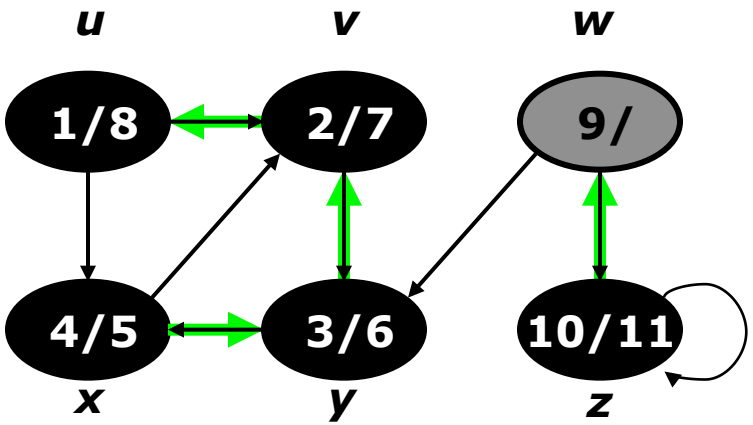

u
v
w
1/
2/7
4/5
3/6
x
y
z

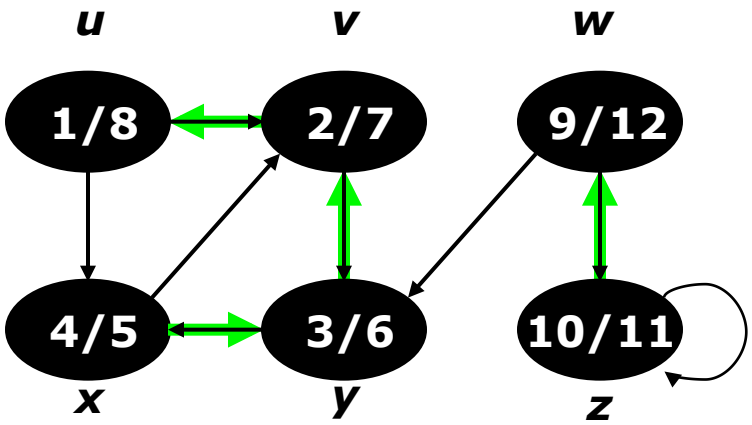

u
v
w
1/8
2/7
4/5
3/6
x
y
z

DFS: Esempio


u
v
w
1/8
2/7
9/
4/5
3/6
x
y
z


u
v
w
1/8
2/7
9/
4/5
3/6
10/
x
y
z


u
v
w
1/8
2/7
9/
4/5
3/6
10/11
x
y
z


u
v
w
1/8
2/7
9/12
4/5
3/6
10/11
x
y
z

DFS: complessità

- I due cicli **for** di ***DFS*** richiedono tempo $O(|V|)$.
- La funzione ***DFSric*** viene richiamata solo su vertici bianchi che vengono subito colorati grigio. Essa viene quindi richiamata al più una sola volta per ogni vertice. Il ciclo interno percorre la lista delle adiacenze dei vertici.
- Siccome la somma delle lunghezze di tutte le liste delle adiacenze è $\Theta(|E|)$ l'intero algoritmo ha complessità $O(|V| + |E|)$.

DFS: proprietà delle parentesi

- Se si rappresenta la scoperta di ogni vertice ***u*** con una parentesi aperta $($_***u*** e la fine con una parentesi chiusa _***u***$)$ si ottiene una sequenza bilanciata di parentesi. Ossia, per ogni coppia di vertici ***u*** e ***v*** ci sono quattro possibilità:
  a) $($_***u*** .... _***u***$)$ .... $($_***v*** .... _***v***$)$
  b) $($_***v*** .... _***v***$)$ .... $($_***u*** .... _***u***$)$
  c) $($_***u*** .... $($_***v*** .... _***v***$)$ .... _***u***$)$
  d) $($_***v*** .... $($_***u*** .... _***u***$)$ .... _***v***$)$

Proprietà delle parentesi: dimostrazione

- Assumiamo che $d[u] < d[v]$ (l'altro caso è simmetrico). Se $f[u] < d[v]$ allora:

$$(_u \ldots\ (_u) \ldots (_v \ldots\ _v)$$

- Se $f[u] > d[v]$ allora quando v viene scoperto u è grigio e siccome v è stato scoperto dopo di u la sua lista delle adiacenze verrà completamente esplorata prima di riprendere l'esplorazione di quella di u. Quindi v viene finito prima di u e pertanto

$$(_u \ldots (_v \ldots\ _v) \ldots\ _u)$$

Altre Proprietà

- Discendenti: Il vertice ***v*** è discendente del vertice ***u*** in un albero della foresta di ricerca in profondità se e solo se:

$$(_u \ldots (_v \ldots _v) \ldots _u).$$

- Cammino bianco: Il vertice ***v*** è discendente del vertice ***u*** in un albero della foresta di ricerca in profondità se e solo se nell'istante in cui ***u*** viene scoperto esiste un cammino da ***u*** a ***v*** i cui vertici sono tutti bianchi (cammino bianco)

Classificazione degli archi

- *archi d'albero*: archi **(*u*,*v*)** con ***v*** scoperto visitando le adiacenze di ***u*** (da grigio a bianco)
- *archi all'indietro*: archi **(*u*,*v*)** con ***u* = *v*** oppure ***v*** ascendente di ***u*** in un albero della foresta di ricerca in profondità (da grigio a grigio)
- *archi in avanti*: archi **(*u*,*v*)** con ***v*** discendente di ***u*** in un albero della foresta (da grigio a nero)
- *archi trasversali*: archi **(*u*,*v*)** in cui ***v*** ed ***u*** appartengono a rami o alberi distinti della foresta (da grigio a nero)

DFS

- *Cicli nel grafo*

Ordinamento Topologico

- La ricerca in profondità si può usare per ordinare topologicamente un *grafo orientato aciclico* (detto anche *DAG*: *D*irected *A*cyclic *G*raph).

- Un *ordinamento topologico* di un grafo orientato aciclico $\boldsymbol{G = (V,E)}$ è un ordinamento (permutazione) dei suoi vertici tale che per ogni arco $\boldsymbol{(u,v) \in E}$ il vertice $\boldsymbol{u}$ precede il vertice $\boldsymbol{v}$.

L'algoritmo: TopSort

```
TP_DFS(G)
  for v ∈ V[G] do
    color[v] ← bianco, p[v] ← nil
  t ← 1
  TP← nil    // lista concatenata vuota
  for v ∈ V[G] do
    if color[v] = bianco then TP_DFSric(v, t)
  return TP;

TP_DFSric(u, t)
  color[u] ← grigio,   d[u] ← t,   t ← t + 1
  for v ∈ Adj[u] do
    if color[v] = bianco then  p[v] ← u, TP_DFSric(v, t)
  color[u] ← nero, f[u] ← t,   t ← t + 1
  TP ← u+TP   //inserimento in testa alla lista concatenata
```

L'algoritmo TopSort: Complessità e Correttezza

- *Complessità*. La stessa di ***DFS***, ossia

$$O(|V| + |E|).$$

- *Correttezza* Dal momento che un grafo è un DAG se e solo se non ci sono archi all'indietro allora, quando un arco $(u,v) \in E$ viene ispezionato da DFS il vertice v non può essere grigio, quindi o è nero ($f[v]$ già assegnato) oppure è bianco ($d[v]$ da assegnare e vale la proprietà delle parentesi).

Topological Sort: Esercizi

- *Aciclicità grafi non orientati*.

- *Algoritmo Alternativo:* rimozione continua vertici con in-degree 0.

Componenti Fortemente Connesse

- La ricerca in profondità si può usare anche per calcolare le componenti fortemente connesse di un *grafo orientato*.

- Una *componente fortemente connessa* (***cfc***) di un grafo orientato $G = (V,E)$ è un insieme massimale di vertici $U \subseteq V$ tale che per ogni $u,v \in U$ esiste un cammino da u a v ed un cammino da v ad u.

Componenti Fortemente Connesse

L'algoritmo per il calcolo delle componenti fortemente connesse si compone di tre fasi:

a) usa la ricerca in profondità in ***G*** per ordinare i vertici in ordine di tempo di fine ***f*** decrescente (come per l'ordinamento topologico): **O(|V|+|E|)**
b) calcola il grafo trasposto G^T del grafo ***G***: **O(|V|+|E|)**
c) esegue una ricerca in profondità in G^T usando l'ordine dei vertici calcolato nella prima fase nel ciclo principale: **O(|V|+|E|)**

Alla fine gli alberi della ricerca in profondità in G^T rappresentano le componenti fortemente connesse.

Proprietà dei cammini

Proprietà dei cammini. Se due vertici ***u*** e ***v*** sono in una stessa ***cfc*** allora tale ***cfc*** contiene anche i vertici di tutti i cammini da ***u*** a ***v***.

Dimostrazione. Sia ***w*** un vertice di un cammino da ***u*** a ***v***. Allora vi è un cammino da ***u*** a ***w*** ed un cammino da ***w*** a ***v***. Siccome ***u*** e ***v*** stanno nella stessa ***cfc*** esiste anche un cammino da ***v*** a ***u*** che concatenato al cammino da ***w*** a ***v*** forma un cammino da ***w*** a ***u***. Quindi ***u*** e ***w*** sono nella stessa ***cfc***.

Proprietà degli alberi

Proprietà degli alberi. Una ricerca in profondità mette nello stesso albero tutti i vertici di una ***cfc***.

Dimostrazione. Sia ***u*** il primo vertice scoperto in una ***cfc***. Quando viene scoperto ***u*** tutti gli altri vertici di tale ***cfc*** sono bianchi. Esiste quindi un cammino bianco da ***u*** ad ogni altro vertice di tale ***cfc***. Per la proprietà del cammino bianco ognuno di tali vertici è discendente di ***u*** e quindi appartiene allo stesso albero.

Grafo delle componenti

Definizione: Dato un grafo orientato **G**, il *grafo delle componenti fortemente connesse* di **G** è il grafo orientato **H** avente come vertici le componenti fortemente connesse di **G** e un arco da una ***cfc*** C ad una ***cfc*** C' se e solo se in **G** vi è un arco che connette un vertice di C ad un vertice di C'.

Proprietà: Il grafo **H** delle componenti fortemente connesse di **G** è aciclico.