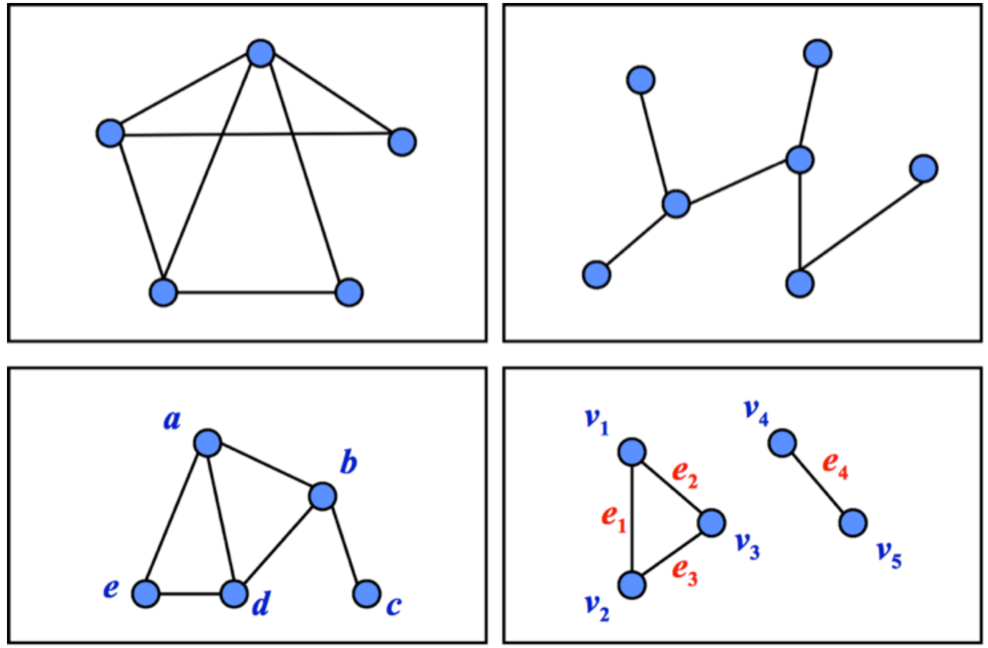
Algoritmi e Strutture Dati

Capitolo 9 - Grafi

Alberto Montresor
Università di Trento

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Esempi di grafi



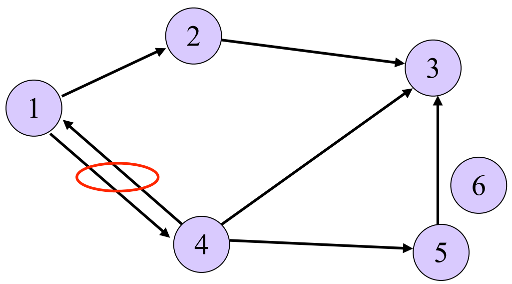
Problemi sui grafi

- **Visite**
  - Visite in ampiezza (numero di Erdös)
  - Visite in profondità (ordinamento topologico, componenti (fortemente) connesse)
- **Cammini minimi**
  - Da singola sorgente
  - Fra tutte le coppie di vertici
- **Alberi di connessione minimi**
- **Problemi di flusso**
- **....**

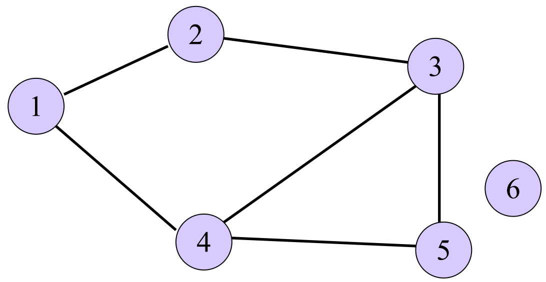
Grafi orientati e non orientati: definizione

- **Un *grafo orientato* G è una coppia (V, E) dove:**
  - Insieme finito dei vertici V
  - Insieme degli archi E: relazione binaria tra vertici

- **Un *grafo non orientato* G è una coppia (V, E) dove:**
  - Insieme finito dei vertici V
  - Insieme degli archi E: coppie non ordinate



V = {1, 2, 3, 4, 5, 6 }
E = { (1,2), (1,4), (2,3), (4,3), (5,3), (4,5), (4,1) }

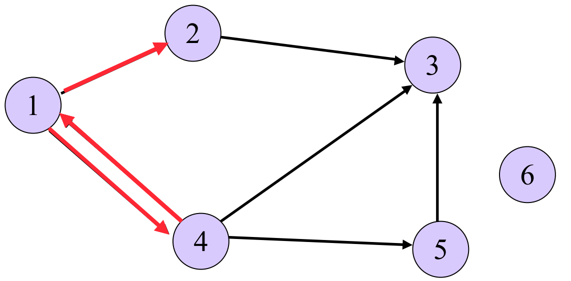


V = {1, 2, 3, 4, 5, 6 }
E = { [1,2], [1,4], [2,3], [3,4], [3,5], [4,5] }

Definizioni: incidenza e adiacenza

- **In un grafo orientato**
  - un arco (u,v) si dice *incidente* da u in v
- **In un grafo non orientato**
  - la relazione di incidenza tra vertici è simmetrica
- **Un vertice v si dice *adiacente* a u se e solo se $(u, v) \in E$**

(1,2) è incidente da 1 a 2
(1,4) è incidente da 1 a 4
(4,1) è incidente da 4 a 1



2 è adiacente ad 1
3 è adiacente a 2, 4, 5
1 è adiacente a 4 e viceversa
2 non è adiacente a 3,4
6 non è adiacente ad alcun vertice

© Alberto Montresor

Rappresentazione grafi

- **Poniamo**
  - $n = |V|$ numero di nodi
  - $m = |E|$ numero di archi
- **Matrice di adiacenza**
  - Spazio richiesto $O(n^2)$
  - Verificare se il vertice u è adiacente a v richiede tempo $O(1)$
  - Elencare tutti gli archi costa $O(n^2)$
- **Liste / vettori di adiacenza**
  - Spazio richiesto $O(n+m)$
  - Verificare se il vertice u è adiacente a v richiede tempo $O(n)$
  - Elencare tutti gli archi costa $O(n+m)$

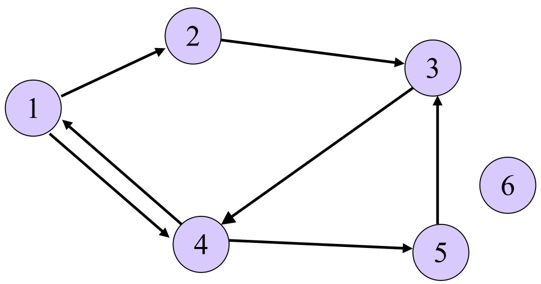
© Alberto Montresor

Matrice di adiacenza: grafo orientato o non orientato

$$m_{uv} = \begin{cases} 1, & \text{se } (u,v) \in E, \\ 0, & \text{se } (u,v) \notin E. \end{cases}$$

Spazio: n^2

$$\begin{array}{c c} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

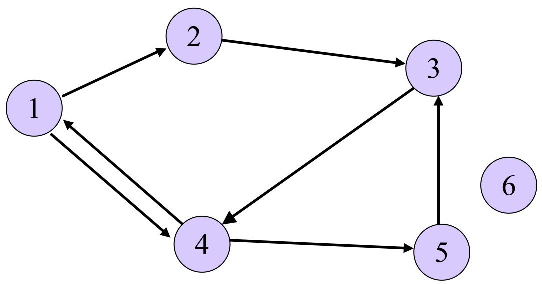


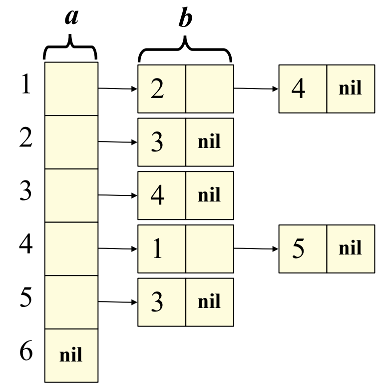
© Alberto Montresor

Liste di adiacenza: grafo orientato

$G.adj(u) = \{ v \mid (u,v) \in E \}$

Spazio: $a \cdot n + b \cdot m$



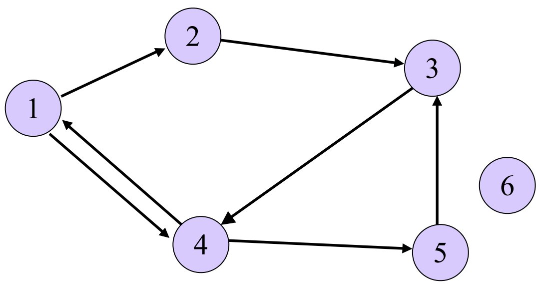


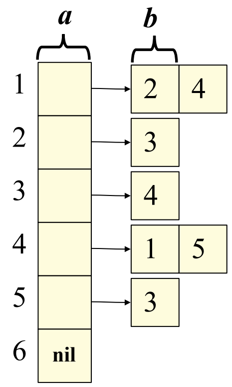
© Alberto Montresor

Vettore di adiacenza: grafo orientato

$G.adj(u) = \{\, v \mid (u,v) \in E \,\}$

Spazio: $a \cdot n + b \cdot m$



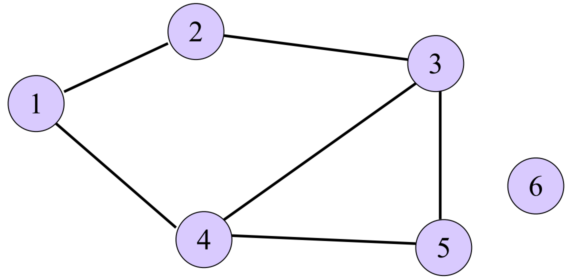


© Alberto Montresor

Matrice di adiacenza: grafo non orientato

$$m_{uv} = \begin{cases} 1, & \text{se } [u,v] \in E, \\ 0, & \text{se } [u,v] \notin E. \end{cases}$$

Spazio: n^2 o $n(n+1)/2$



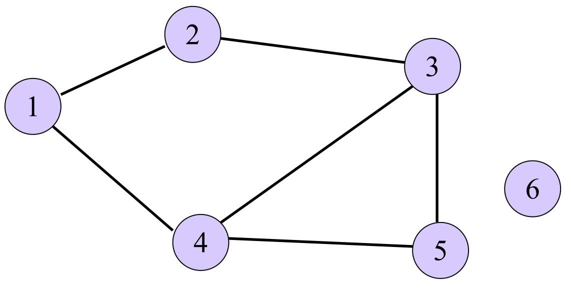
$$\begin{array}{c|c} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

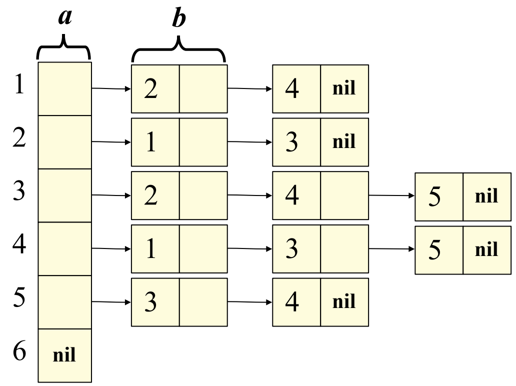
© Alberto Montresor

Liste/vettore di adiacenza: grafo non orientato

$G.adj(u) = \{\, v \mid [u,v] \in E$ **or** $[v,u] \in E\}$

Spazio: $a \cdot n + 2 \cdot b \cdot m$



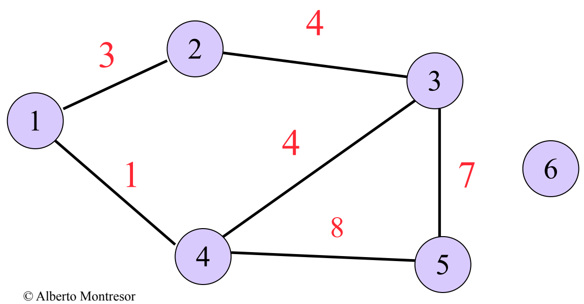


© Alberto Montresor

Grafi pesati

- **In alcuni casi ogni arco ha un *peso* (*costo*, *guadagno*) associato**
  - Il peso può essere determinato tramite una funzione di costo p: $V \times V \rightarrow$ **R**, dove **R** è l'insieme dei numeri reali
  - Quando tra due vertici non esiste un arco, il peso è infinito

$$m_{uv} = \begin{cases} p_{uv}, & \text{se } (u,v) \in E, \\ +\infty \text{ (oppure } -\infty) & \text{se } (u,v) \notin E. \end{cases}$$



$$\begin{array}{c|c} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{pmatrix} * & 3 & * & 1 & * & * \\ 3 & * & 4 & * & * & * \\ * & 4 & * & 4 & 7 & * \\ 1 & * & 4 & * & 8 & * \\ * & * & 7 & 8 & * & * \\ * & * & * & * & * & * \end{pmatrix} \end{array}$$

© Alberto Montresor

Specifica

```
GRAPH
  Graph( )                              % Crea un grafo vuoto
  insertNode(NODE u)                    % Aggiunge il nodo u al grafo
  insertEdge(NODE u, NODE v)            % Aggiunge l'arco (u, v) al grafo
  deleteNode(NODE u)                    % Rimuove il nodo u dal grafo
  deleteEdge(NODE u, NODE v)            % Rimuove l'arco (u, v) nel grafo
  SET adj(NODE u)                       % Restituisce l'insieme dei nodi adiacenti ad u
  SET V()                               % Restituisce l'insieme di tutti i nodi
```

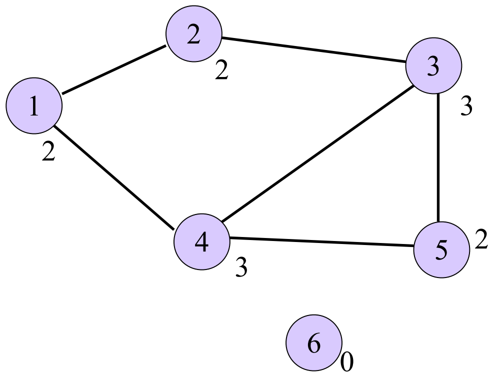
```
foreach u ∈ G.V() do
    foreach v ∈ G.adj(u) do
        fai un'operazione sull'arco (u, v)
```

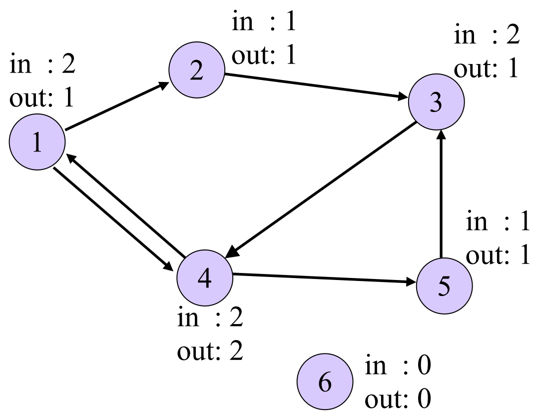
- **Complessità**
  - $O(n+m)$ liste di adiacenza
  - $O(n^2)$matrice di adiacenza
  - $O(m)$ "operazioni"

© Alberto Montresor

Definizioni: Grado

- **In un grafo non orientato**
  - il *grado* di un vertice è il numero di archi che partono da esso
- **In un grafo orientato**
  - il *grado entrante* (*uscente*) di un vertice è il numero di archi incidenti in (da) esso

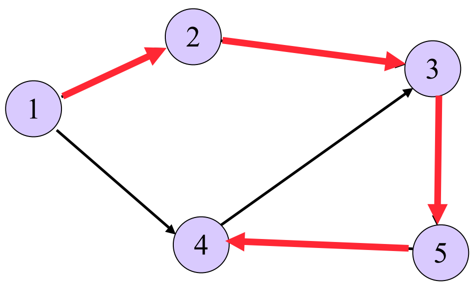




© Alberto Montresor

Definizioni: Cammini

- **In un grafo orientato $G=(V,E)$**
  - un *cammino* di lunghezza k è una sequenza di vertici $u_0, u_1, \dots, u_k$ tale che $(u_i, u_{i+1}) \in E$ per $0 \leq i \leq k-1$
- **In un grafo non orientato $G=(V,E)$**
  - una *catena* di lunghezza k è una sequenza di vertici $u_0, u_1, \dots, u_k$ tale che $[u_i, u_{i+1}] \in E$ per $0 \leq i \leq k-1$



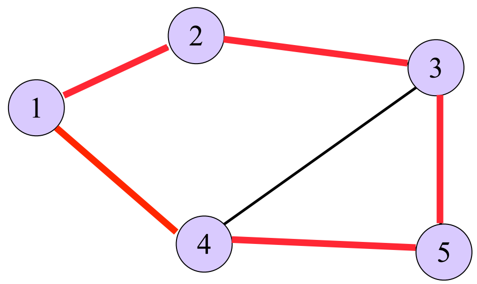
Esempio: **1, 2, 3, 5, 4** è una catena nel grafo con lunghezza 4

Un cammino (catena) si dice *semplice* se tutti i suoi vertici sono distinti (compaiono una sola volta nella sequenza)

© Alberto Montresor

Definizioni: Cicli

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



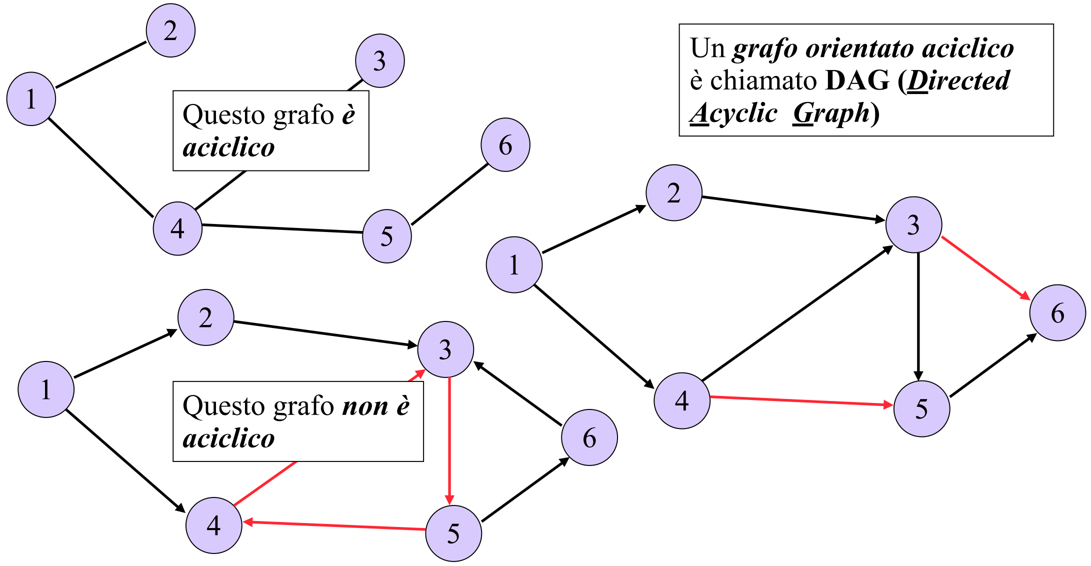
Esempio: **1, 2, 3, 5, 4, 1** è un circuito con lunghezza 5

Un ciclo (circuito) si dice *semplice* se tutti i suoi vertici sono distinti (tranne ovviamente il primo / l'ultimo)

© Alberto Montresor

Definizioni: Grafi aciclici

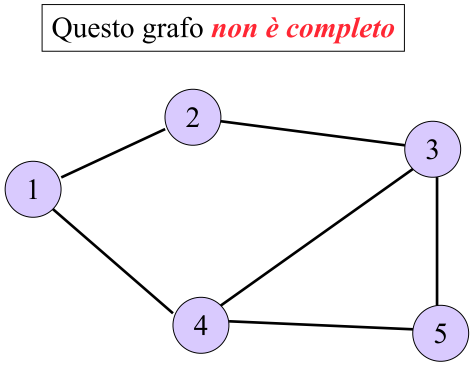
- **Un grafo senza cicli è detto aciclico**

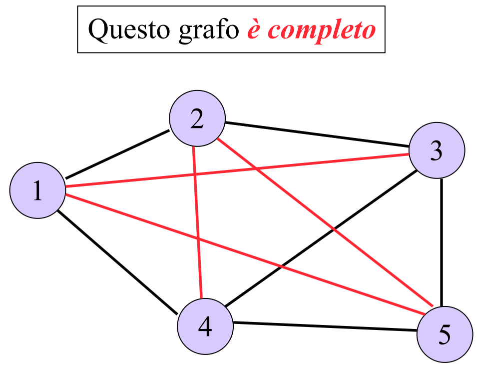


© Alberto Montresor

Definizioni: Grafo completo

- **Un *grafo completo* è un grafo che ha un arco tra ogni coppia di vertici.**



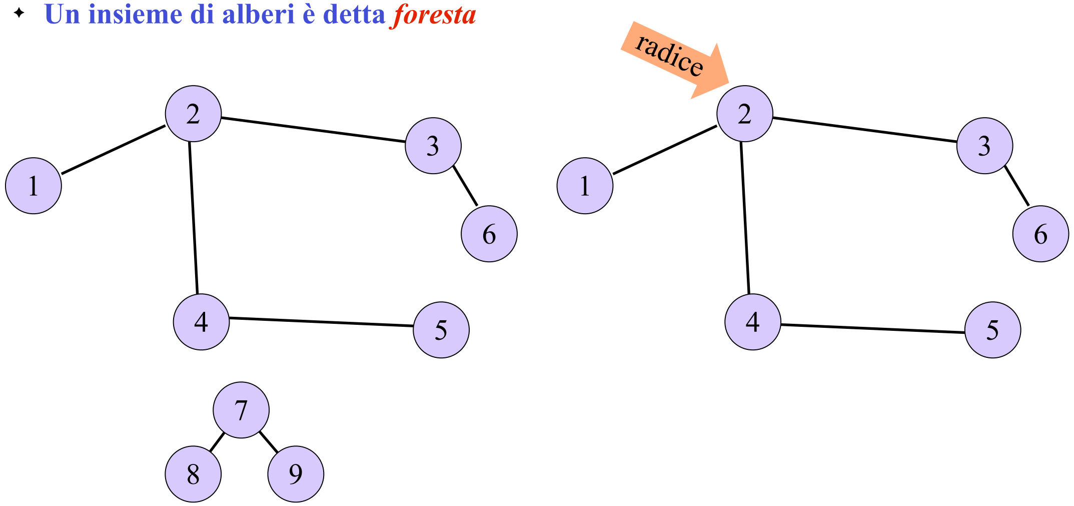


$$m = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

© Alberto Montresor

Definizioni: Alberi

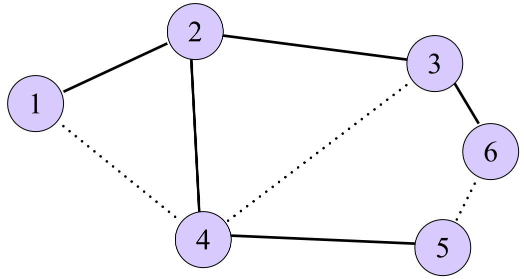
- **Un *albero libero* è un grafo non orientato connesso, aciclico**
- **Se qualche vertice è detto radice, otteniamo un *albero radicato***
- **Un insieme di alberi è detta *foresta***



© Alberto Montresor

Definizioni: Alberi di copertura

- **In un grafo non orientato $G=(V, E)$**
  - un albero di copertura T è un albero libero $T = (V, E')$ composto da tutti i nodi di V e da un sottoinsieme degli archi ($E' \subseteq E$), tale per cui tutte le coppie di nodi del grafo sono connesse da una sola catena nell'albero.



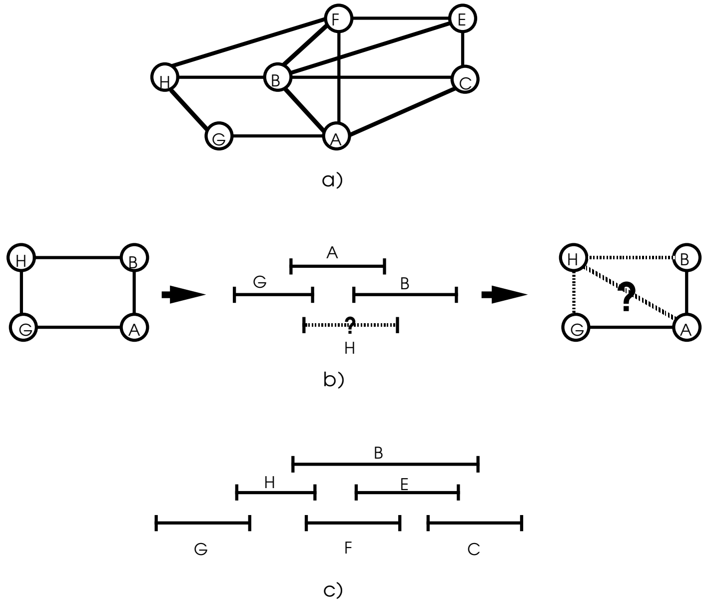
© Alberto Montresor

Un esempio di utilizzo dei grafi

- **Sherlock Holmes indaga sulla morte del duca McPollock, ucciso da un'esplosione nel suo maniero:**
  - **Watson**: "Ci sono novità, Holmes: pare che il testamento, andato distrutto nell'esplosione, fosse stato favorevole ad una delle sette 'amiche' del duca"
  - **Holmes**: "Ciò che è più strano, è che la bomba sia stata fabbricata appositamente per essere nascosta nell'armatura della camera da letto, il che fa supporre che l'assassino abbia necessariamente fatto più di una visita al castello"
  - **Watson**: "Ho interrogato personalmente le sette donne, ma ciascuna ha giurato di essere stata nel castello una sola volta nella sua vita.
    - (1) Ann ha incontrato Betty, Charlotte, Felicia e Georgia;
    - (2) Betty ha incontrato Ann, Charlotte, Edith, Felicia e Helen;
    - (3) Charlotte ha incontrato Ann, Betty e Edith;
    - (4) Edith ha incontrato Betty, Charlotte, Felicia;
    - (5) Felicia ha incontrato Ann, Betty, Edith, Helen;
    - (6) Georgia ha incontrato Ann e Helen;
    - (7) Helen ha incontrato Betty, Felicia e Georgia.

    Vedete, Holmes, che le testimonianze concordano. Ma chi sarà l'assassino?"
  - **Holmes**: "Elementare, Watson: ciò che mi avete detto individua inequivocabilmente l'assassino!"

Un esempio di utilizzo dei grafi



Problema: Visita grafi

- **Definizione del problema**
  - Dato un grafo G=(V, E) ed un vertice r di V (detto *sorgente* o *radice*), visitare ogni vertice raggiungibile nel grafo dal vertice r
  - Ogni nodo deve essere visitato una volta sola
- **Visita in ampiezza (breadth-first search)**
  - Visita i nodi "espandendo" la frontiera fra nodi scoperti / da scoprire
  - Esempi: Cammini più brevi da singola sorgente
- **Visita in profondità (depth-first search)**
  - Visita i nodi andando il "più lontano possibile" nel grafo
  - Esempi: Componenti fortemente connesse, ordinamento topologico

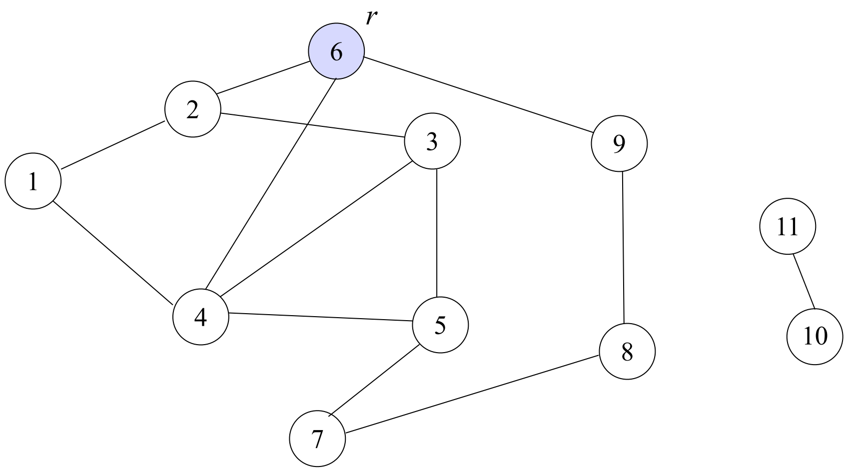
Visita: attenzione alle soluzioni "facili"

- **Prendere ispirazione dalla visita degli alberi**
- **Ad esempio:**
  - utilizziamo una visita BFS basata su coda
  - trattiamo i "vertici adiacenti" come se fossero i "figli"

```
visita(GRAPH G, NODE r)
    QUEUE S ← Queue()
    S.enqueue(r)
    while not S.isEmpty() do
        NODE u ← S.dequeue()
        { esamina il nodo u }
        foreach v ∈ G.adj(u) do
            S.enqueue(v)
```

Sbagliato

Esempio di visita - errata

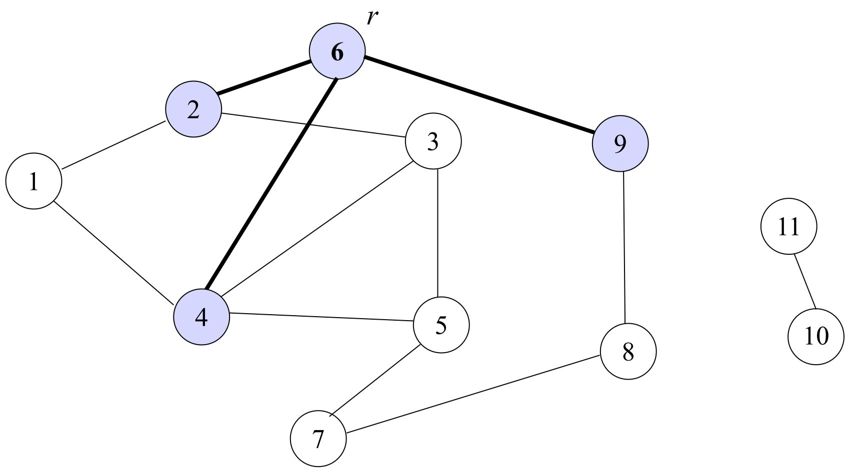


`Coda:{6}`

© Alberto Montresor

Esempio di visita - errata

`u = 6`

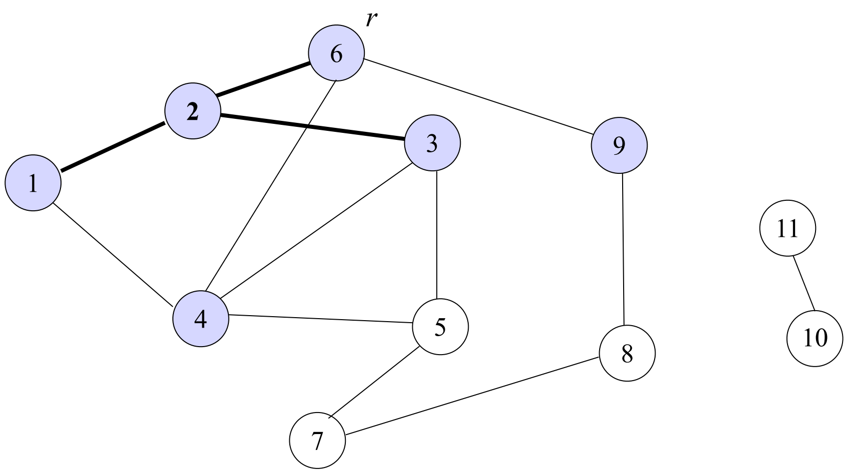


`Coda:{2,4,9}`

© Alberto Montresor

Esempio di visita - errata

`u = 2`

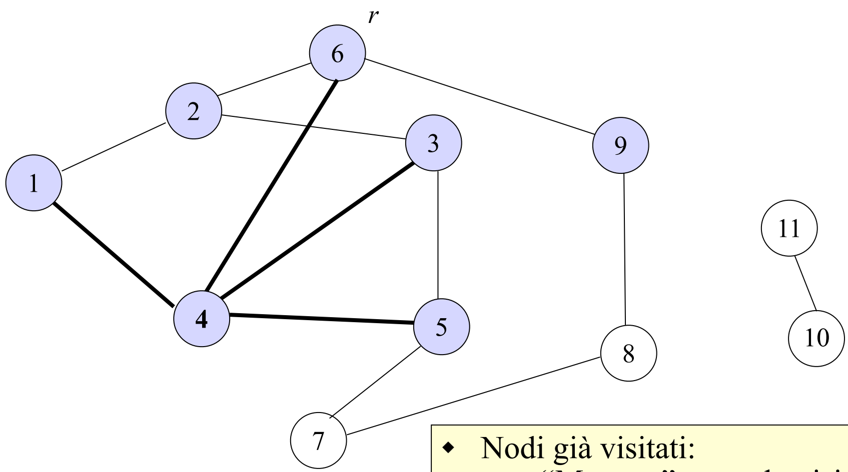


`Coda:{4,9,3,1,6}`

© Alberto Montresor

Esempio di visita - errata

`u = 4`



`Coda:{9,3,1,6,5,6,1,3}`

- Nodi già visitati:
  - "Marcare" un nodo visitato in modo che non possa essere visitato di nuovo
  - Bit di marcatura: nel vertice, array separato, etc.

© Alberto Montresor

Algoritmo generico per la visita

```
visita(GRAPH G, NODE r)
    SET S ← Set()
    S.insert(r)
    { marca il nodo r come "scoperto" }
    while S.size() > 0 do
        NODE u ← S.remove()
        { esamina il nodo u }
        foreach v ∈ G.adj(u) do
            { esamina l'arco (u, v) }
            if v non è già stato scoperto then
                { marca il nodo v come "scoperto" }
                S.insert(v)
```

- S è l'insieme *frontiera*
- Il funzionamento di insert() e remove() non è specificato

© Alberto Montresor

Visita in ampiezza (breadth first search, BFS)

Cosa vogliamo fare?

- **Visitare i nodi a distanze crescenti dalla sorgente**
  - visitare i nodi a distanza k prima di visitare i nodi a distanza $k+1$
- **Generare un albero BF (breadth-first)**
  - albero contenente tutti i vertici raggiungibili da r e tale che il cammino da r ad un nodo nell'albero corrisponde al cammino più breve nel grafo
- **Calcolare la distanza minima da s a tutti i vertici raggiungibili**
  - numero di archi attraversati per andare da r ad un vertice

© Alberto Montresor

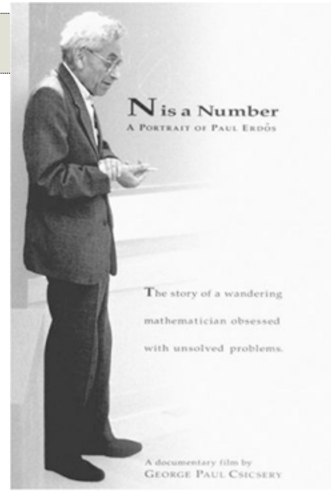
Visita in ampiezza (breadth first search, BFS)

```
bfs(GRAPH G, NODE r)
    QUEUE S ← Queue()
    S.enqueue(r)
    boolean[] visitato ← new boolean[1 ... G.n]
    foreach u ∈ G.V() − {r} do visitato[u] ← false
    visitato[r] ← true
    while not S.isEmpty() do
        NODE u ← S.dequeue()
        { esamina il nodo u }
        foreach v ∈ G.adj(u) do
            { esamina l'arco (u, v) }
            if not visitato[v] then
                visitato[v] ← true
                S.enqueue(v)
```

- Insieme S gestito tramite una coda
- *visitato[v]* corrisponde alla marcatura del nodo *v*

© Alberto Montresor

Applicazione BFS: Numero di Erdös



- **Paul Erdös (1913-1996)**
  - Matematico
  - Più di 1500 articoli, con più di 500 co-autori
- **Numero di Erdös**
  - Erdös ha *erdös* = 0
  - I co-autori di Erdös hanno *erdös* = 1
  - Se X ha scritto una pubblicazione scientifica con un co-autore con *erdös* $= k$, ma non con un co-autore con *erdös* $< k$, X ha *erdös*$=k+1$
  - Chi non è raggiunto da questa definizione ha erdös = +∞
- **Vediamo un'applicazione di BFS per calcolare il numero di Erdös**

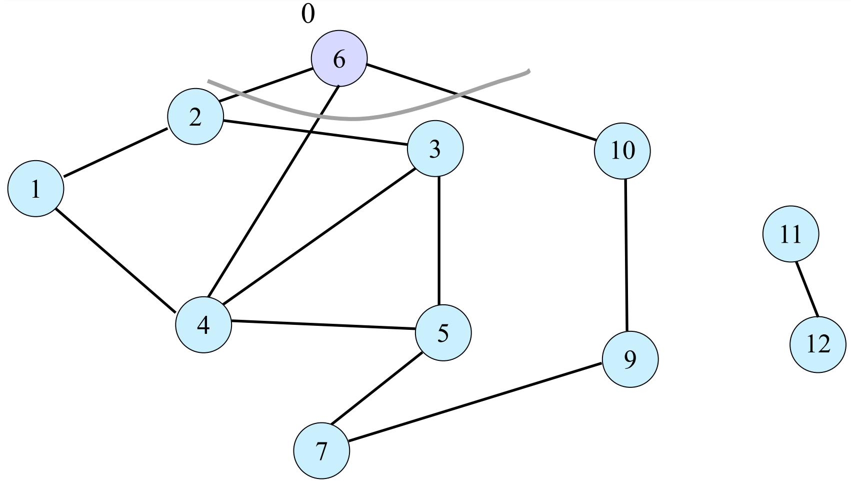
© Alberto Montresor

Calcolo del numero di Erdös

erdos(GRAPH G, NODE r, **integer**[] *erdős*, NODE[] p)

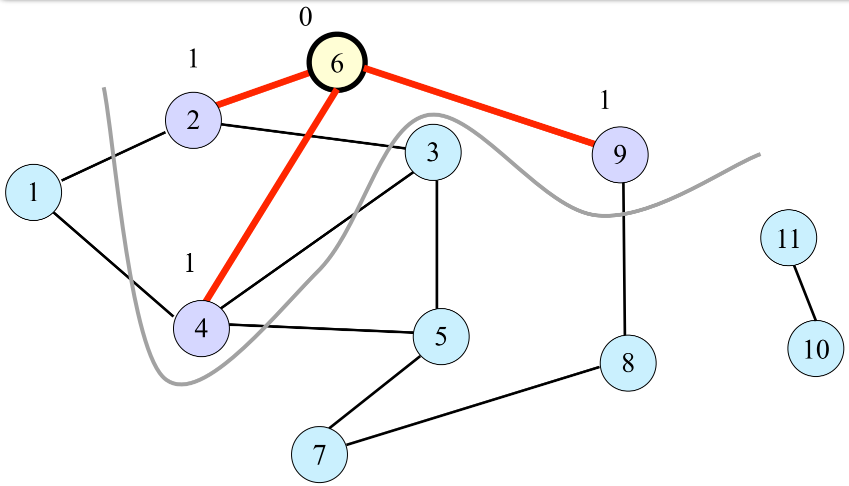
- QUEUE $S \leftarrow$ Queue()
- S.enqueue(r)
- **foreach** $u \in G$.V() $- \{r\}$ **do** $erdős[u] = \infty$
- $erdős[r] \leftarrow 0$
- $p[r] \leftarrow$ **nil**
- **while not** S.isEmpty() **do**
  - NODE $u \leftarrow S$.dequeue()
  - **foreach** $v \in G$.adj(u) **do** % Esamina l'arco (u, v)
    - **if** $erdős[v] = \infty$ **then** % Il nodo u non è già stato scoperto
      - $erdős[v] \leftarrow erdős[u] + 1$
      - $p[v] \leftarrow u$
      - S.enqueue(v)

Esempio



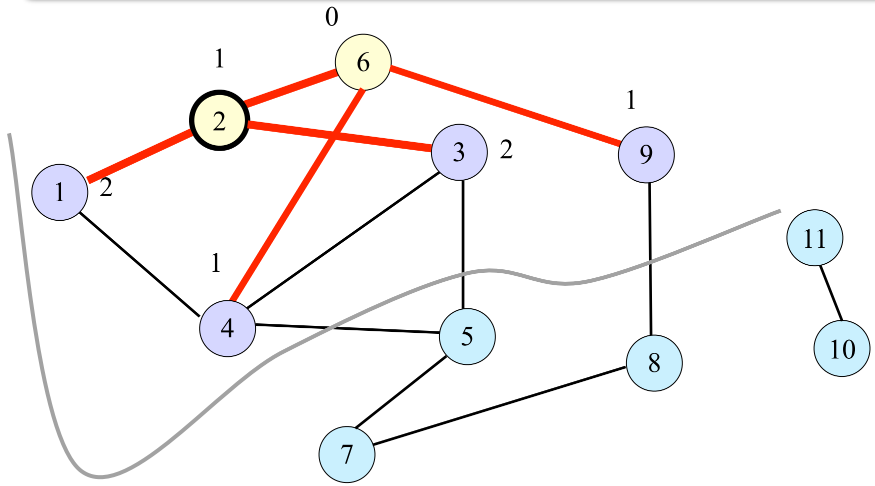
Coda:{6}

Esempio



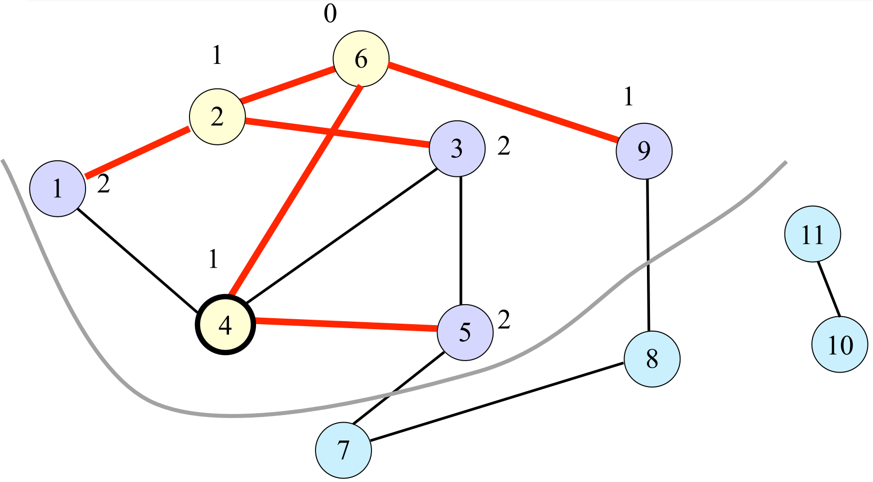
Coda:{2,4,9}

Esempio



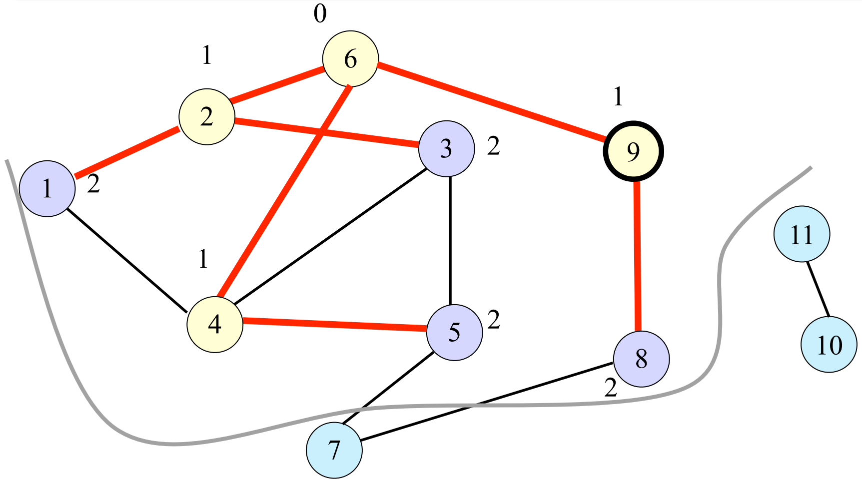
Coda:{4,9,3,1}

Esempio



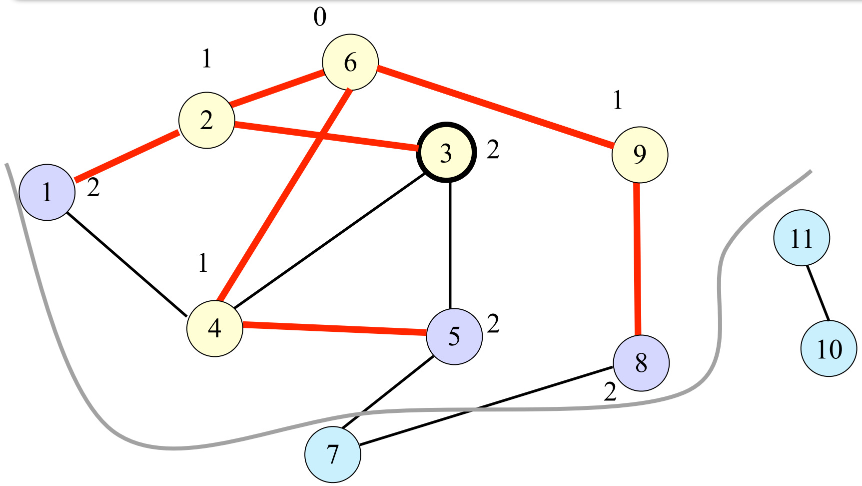
Coda:{9,3,1,5}

Esempio



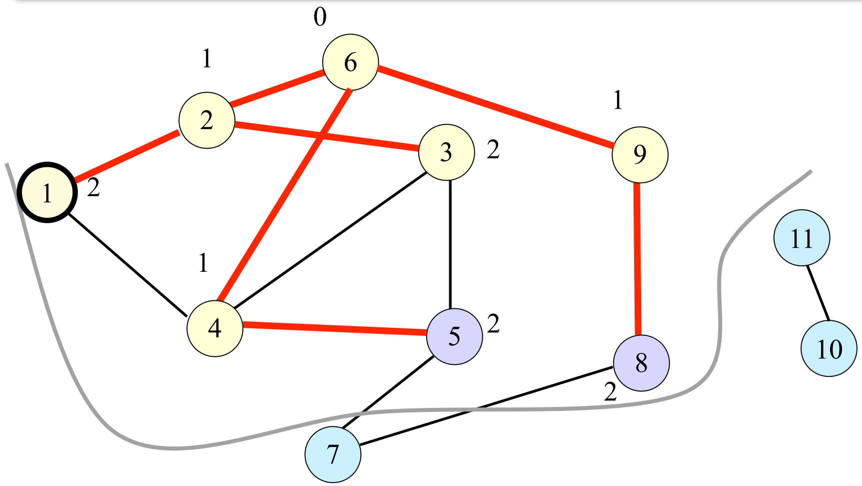
Coda:{3,1,5,8}

Esempio



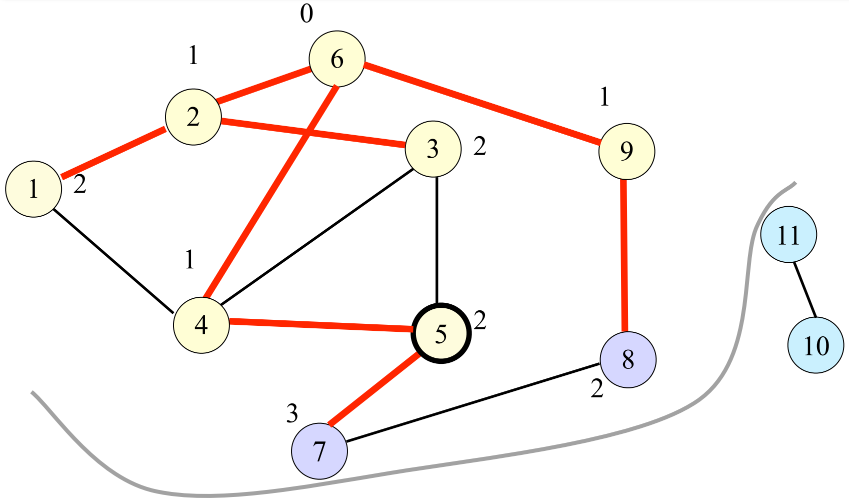
Coda:{1,5,8}

Esempio



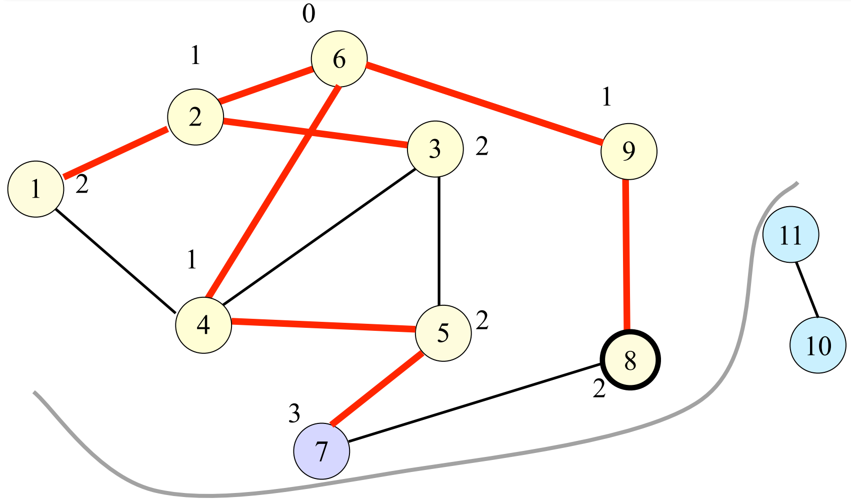
Coda:{5,8}

Esempio



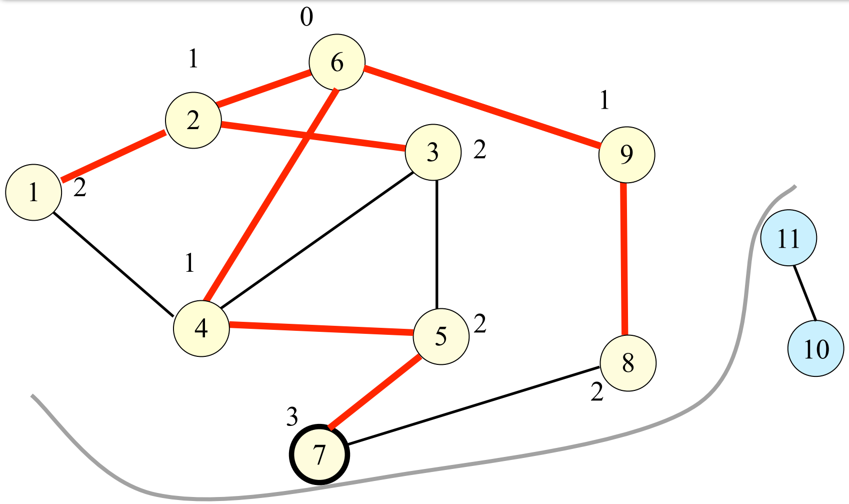
Coda:{8,7}

Esempio



Coda:{7}

Esempio



Coda:{}

Albero dei cammini BFS

- **La visita BFS può essere utilizzata per ottenere il cammino più breve fra due vertici (numero di archi)**
  - Albero di copertura di *G* radicato in *r*
  - Memorizzato tramite vettore dei padri *p*
  - Figli di *u* - nodi *v* tali che *(u,v)* ∈ *E* e *v* non è ancora visitato

if $erdős[v] = \infty$ **then**
  $erdős[v] \leftarrow erdős[u] + 1$
  $p[v] \leftarrow u$
  S.enqueue(v)

stampaCammino(Node r, Node s, Node[] p)
  if $r = s$ **then print** s
  else if $p[s] =$ **nil then**
    print "nessun cammino da r a s"
  else
    stampaCammino$(r, p[s], p)$
    print s

Algoritmo generico per la visita

- **Alcune definizioni**
  - L'albero T contiene i *vertici visitati*
  - $S \subseteq T$ contiene i *vertici aperti*: vertici i cui archi uscenti non sono ancora stati percorsi
  - $T\text{-}S \subseteq T$ contiene i *vertici chiusi*: vertici i cui archi uscenti sono stati tutti percorsi
  - $V\text{-}T$ contiene i vertici non visitati
  - Se u si trova lungo il cammino che va da r al nodo v, diciamo che:
    - u è un antenato di v
    - v è un discendente di u
- **Alcune cose da notare:**
  - I nodi vengono visitati al più una volta (marcatura)
  - Tutti i nodi raggiungibili da r vengono visitati
  - Ne segue che T contiene esattamente tutti i nodi raggiungibili da r

Visita in profondità (depth first search, DFS)

- **Visita in profondità**
  - E' spesso una "subroutine" della soluzione di altri problemi
  - Utilizzata per coprire l'intero grafo, non solo i nodi raggiungibili da una singola sorgente (diversamente da BFS)
- **Output**
  - Invece di un albero, una *foresta* DF (depth-first) $G_\pi=(V, E_\pi)$
    - Contenente un insieme di alberi DF
- **Struttura di dati**
  - Ricorsione al posto di una pila esplicita

```
dfs(GRAPH G, NODE u, boolean[] visitato)
    visitato[u] ← true
(1) { esamina il nodo u (caso previsita) }
    foreach v ∈ G.adj(u) do
        { esamina l'arco (u, v) }
        if not visitato[v] then
            dfs(G, v, visitato)
(2) { esamina il nodo u (caso postvisita) }
```

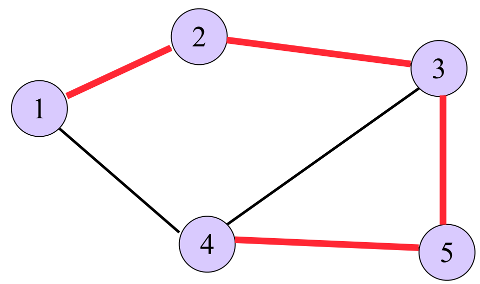

Componenti connesse e fortemente connesse

- **Terminologia**
  - Componenti connesse (connected components, CC)
  - Componenti fortemente connesse (strongly connected components, SCC)
- **Motivazioni**
  - Molti algoritmi che operano sui grafi iniziano decomponendo il grafo nelle sue componenti
  - L'algoritmo viene poi eseguito su ognuna delle componenti
  - I risultati vengono poi ricomposti assieme

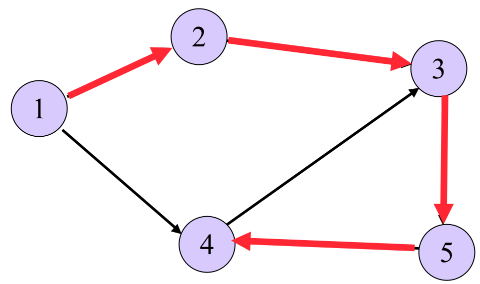
Definizioni: Raggiungibilità

- **In grafo orientato (non orientato)**
  - Se esiste un cammino (catena) c tra i vertici u e v, si dice che v è *raggiungibile* da u tramite c

1 è raggiungibile da 4 e viceversa

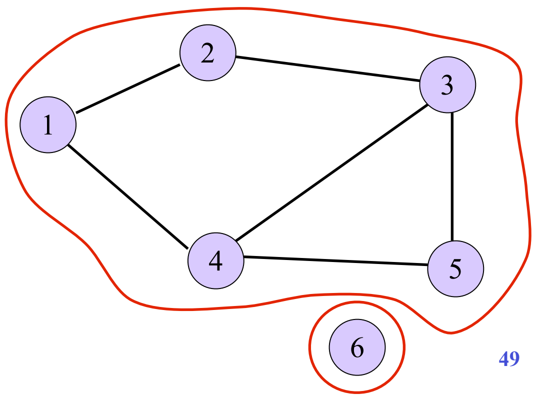


4 è raggiungibile da 1 ma non viceversa



Definizioni: Grafi connessi e componenti connesse

- **In un grafo non orientato G**
  - G è *connesso* ⇔ esiste un cammino da ogni vertice ad ogni altro vertice
  - Un grafo $G' = (V', E')$ è una *componente connessa* di G ⇔ è un sottografo di G connesso e massimale
- **Definizioni**
  - G' è un *sottografo* di G ($G' \subseteq G$) se e solo se $V' \subseteq V$ e $E' \subseteq E$
  - G' è *massimale* ⇔ non esiste un sottografo G'' di G che sia connesso e "più grande" di G', ovvero tale per cui $G' \subseteq G'' \subseteq G$



© Alberto Montresor

Applicazioni DFS: Componente connesse

- **Problema**
  - Verificare se un grafo non orientato è connesso
  - Identificare le componenti connesse di cui è composto
- **Soluzione**
  - Un grafo è connesso se, al termine della DFS, tutti i nodi sono stati marcati
  - Altrimenti, una singola passata non è sufficiente e la visita deve ripartire da un nodo non marcato, scoprendo una nuova porzione del grafo
- **Strutture dati**
  - Vettore *id* degli identificatori di componente
  - *id*[u] è l'identificatore della componente connessa a cui appartiene u

© Alberto Montresor

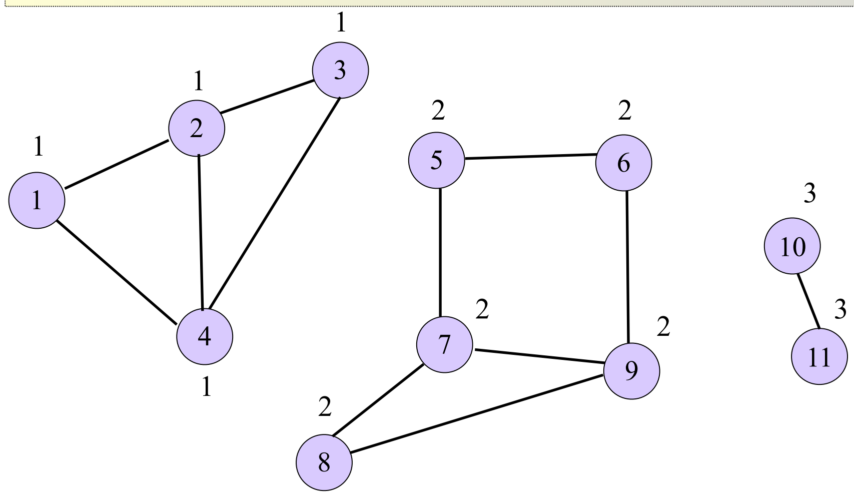
Componenti connesse

```
integer[] cc(GRAPH G, STACK S)
    integer[] id ← new integer[1 . . . G.n]
    foreach u ∈ G.V() do id[u] ← 0
    integer counter ← 0
    while not S.isEmpty() do
        u ← S.pop()
        if id[u] = 0 then
            counter ← counter + 1
            ccdfs(G, counter, u, id)
    return id

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

© Alberto Montresor

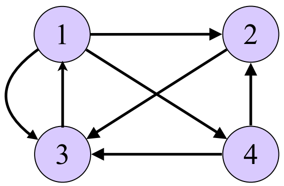
Componenti connesse

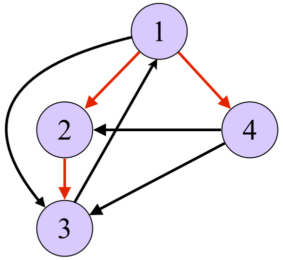


© Alberto Montresor

Alberi di copertura DFS

- **La visita DFS genera l'albero (foresta) dei cammini DFS**
  - Tutte le volte che viene incontrato un arco che connette un nodo marcato ad uno non marcato, esso viene inserito nell'albero T
- **Gli archi non inclusi in *T* possono essere divisi in tre categorie durante la visita:**
  - se l'arco è esaminato passando da un nodo di T ad un altro nodo che è suo antenato in T, è detto *arco all'indietro*
  - se l'arco è esaminato passando da un nodo di T ad un suo discendente (che non sia figlio) in T è detto *arco in avanti*
  - altrimenti, è detto *arco di attraversamento*





© Alberto Montresor

Schema DFS

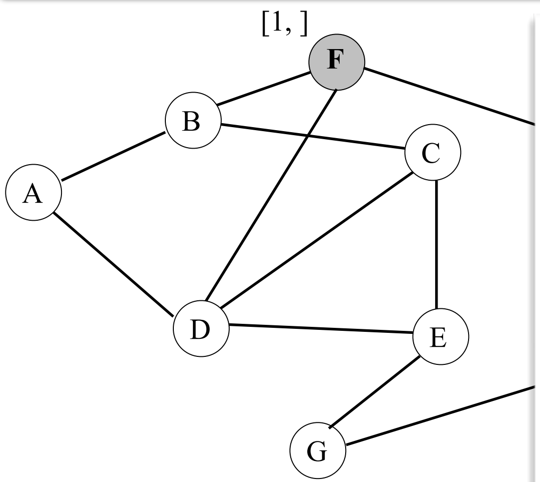
- **Variabili globali**
  - *time* orologio
  - *dt* discovery time
  - *ft* finish time

dfs-schema(GRAPH G, NODE u)

esamina il nodo u prima (caso *pre-visita*)
$time \leftarrow time + 1;\quad dt[u] \leftarrow time$
foreach $v \in G.\mathsf{adj}(u)$ **do**
  esamina l'arco (u, v) di qualsiasi tipo
  if $dt[v] = 0$ **then**
    esamina l'arco (u, v) in T
    dfs-schema(g, v)
  else if $dt[u] > dt[v]$ **and** $ft[v] = 0$ **then**
    esamina l'arco (u, v) all'indietro
  else if $dt[u] < dt[v]$ **and** $ft[v] \neq 0$ **then**
    esamina l'arco (u, v) in avanti
  else
    esamina l'arco (u, v) di attraversamento
esamina il nodo u dopo (caso *post-visita*)
$time \leftarrow time + 1;\quad ft[u] \leftarrow time$

© Alberto Montresor

Esempio

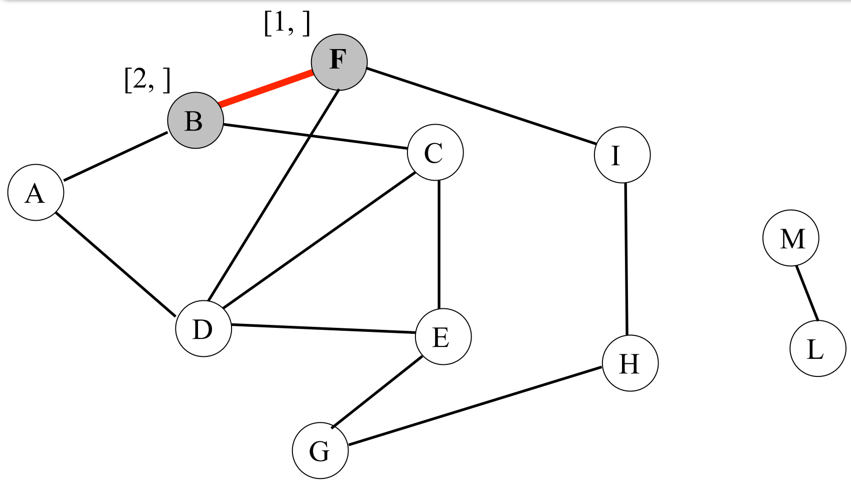


dfs-schema(GRAPH G, NODE u)

esamina il nodo u prima (caso *pre-visita*)
$time \leftarrow time + 1;\quad dt[u] \leftarrow time$
foreach $v \in G.\mathsf{adj}(u)$ **do**
  esamina l'arco (u, v) di qualsiasi tipo
  if $dt[v] = 0$ **then**
    esamina l'arco (u, v) in T
    dfs-schema(g, v)
  else if $dt[u] > dt[v]$ **and** $ft[v] = 0$ **then**
    esamina l'arco (u, v) all'indietro
  else if $dt[u] < dt[v]$ **and** $ft[v] \neq 0$ **then**
    esamina l'arco (u, v) in avanti
  else
    esamina l'arco (u, v) di attraversamento
esamina il nodo u dopo (caso *post-visita*)
$time \leftarrow time + 1;\quad ft[u] \leftarrow time$

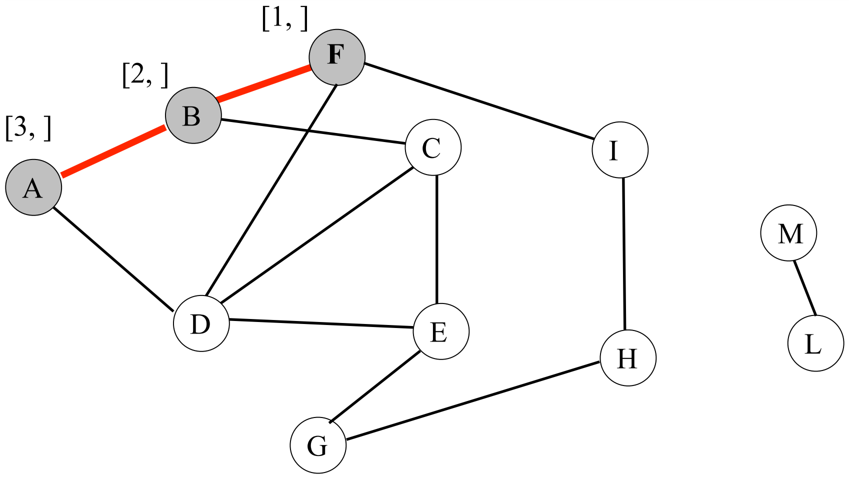
© Alberto Montresor

Esempio



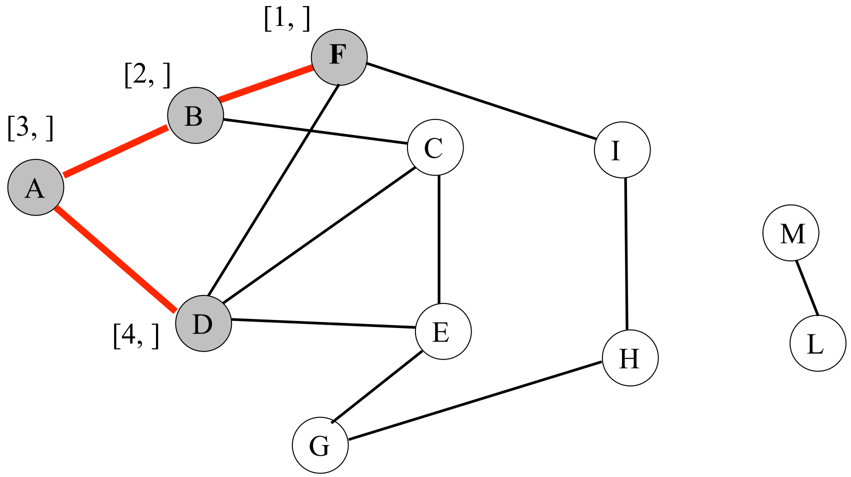
© Alberto Montresor

Esempio



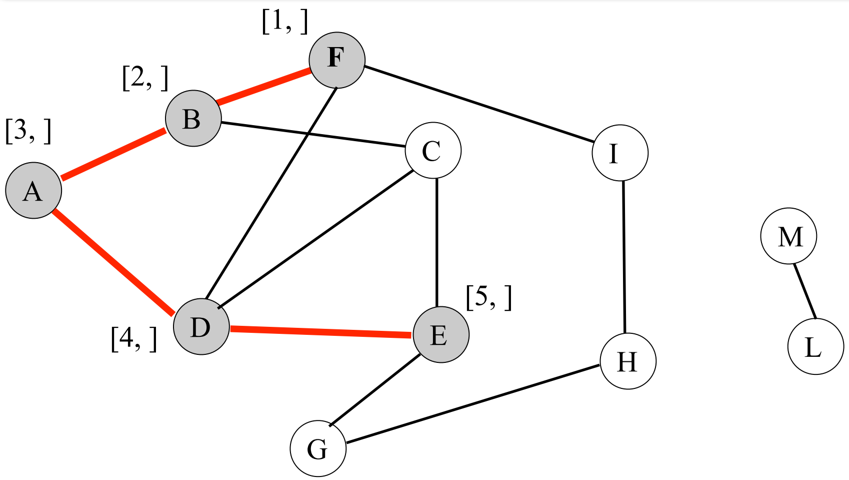
© Alberto Montresor

Esempio



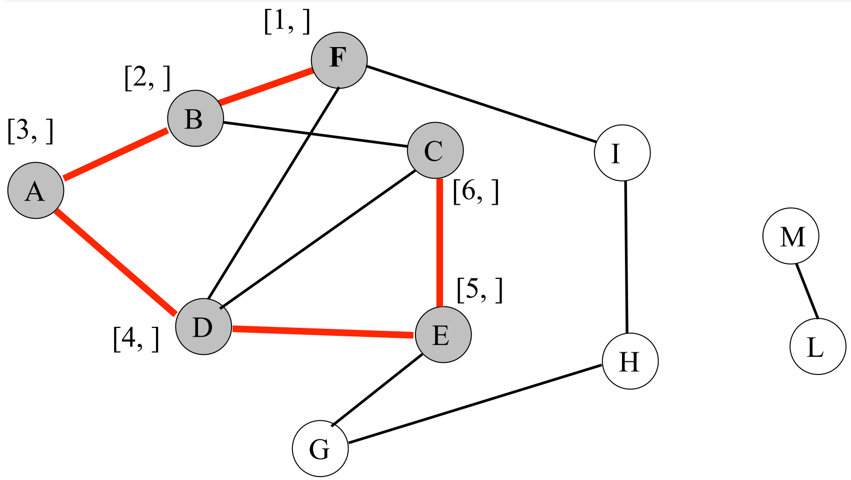
© Alberto Montresor

Esempio



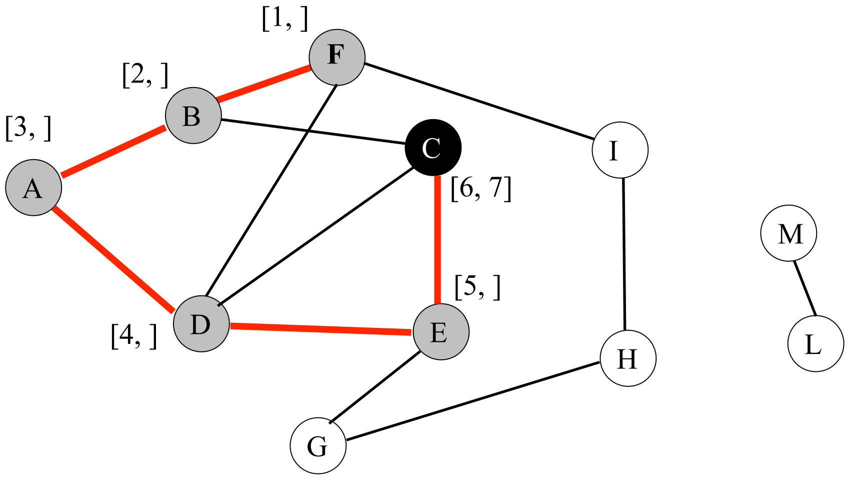
© Alberto Montresor

Esempio



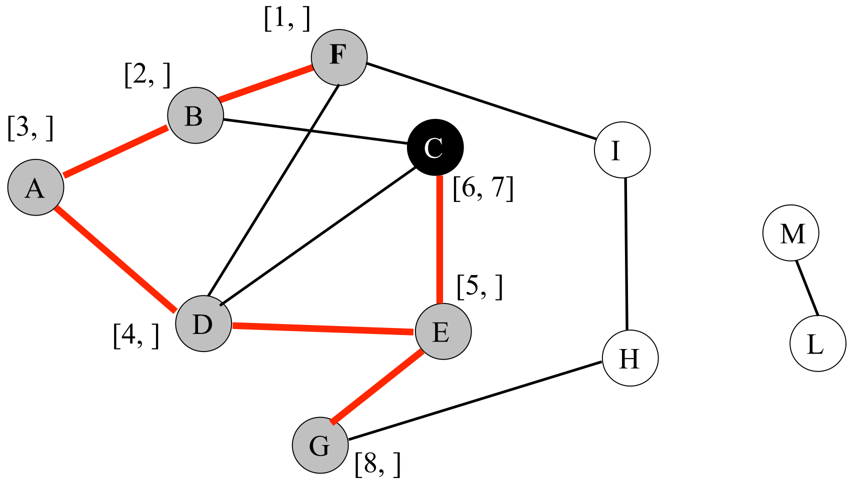
© Alberto Montresor

Esempio



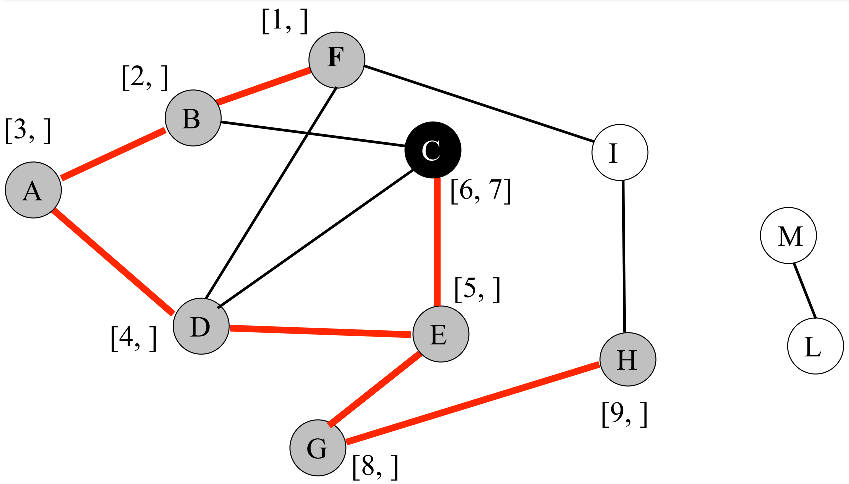
© Alberto Montresor

Esempio



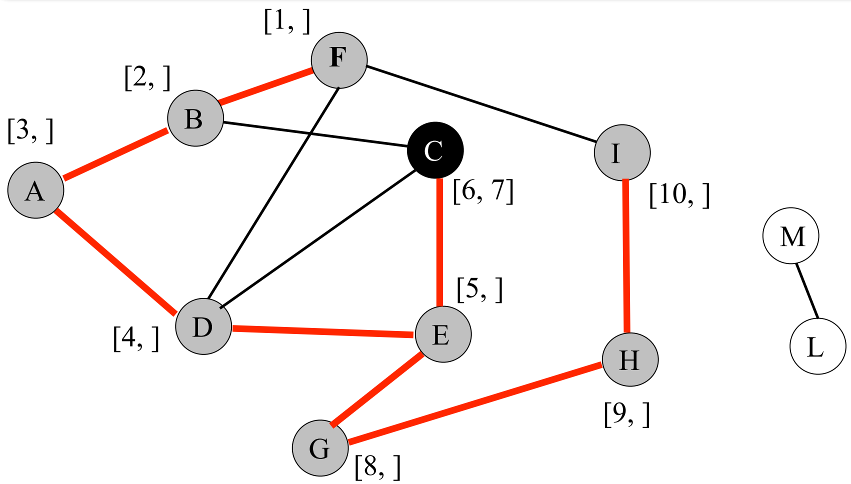
© Alberto Montresor

Esempio



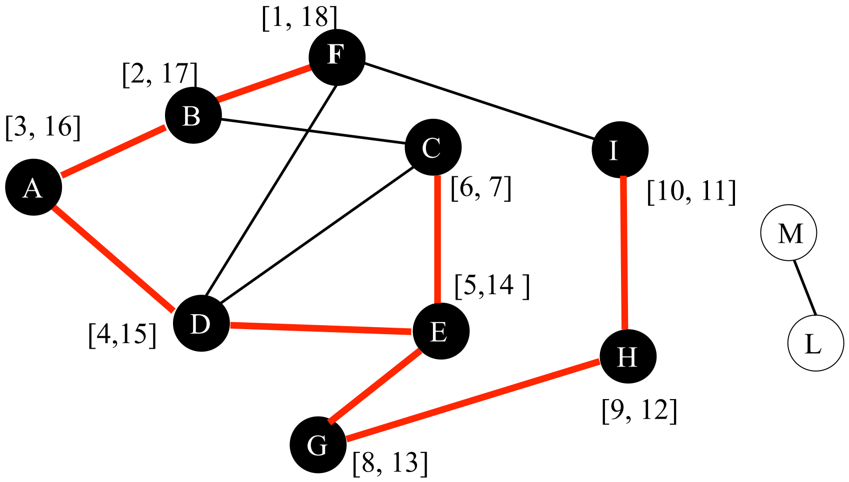
© Alberto Montresor

Esempio



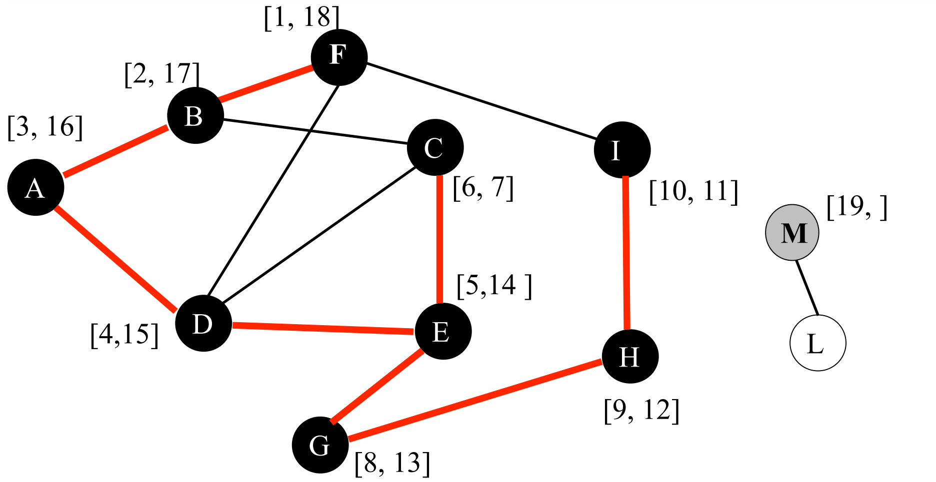
© Alberto Montresor

Esempio



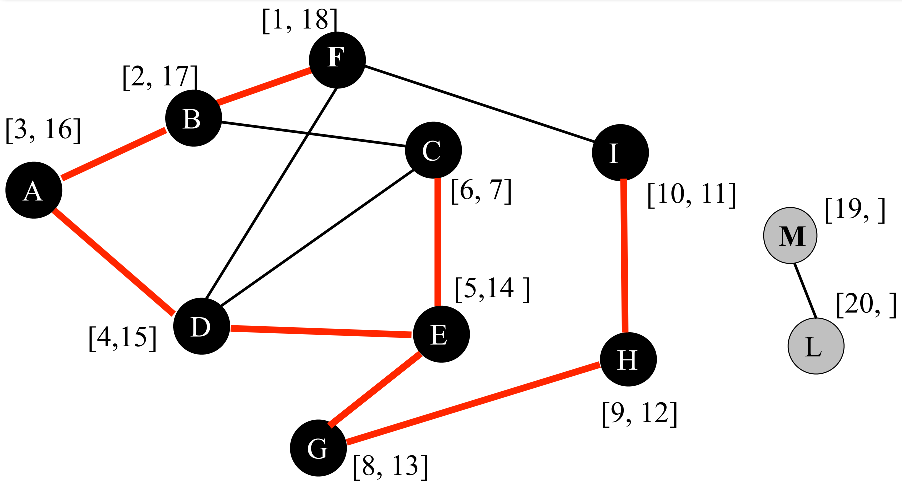
© Alberto Montresor

Esempio



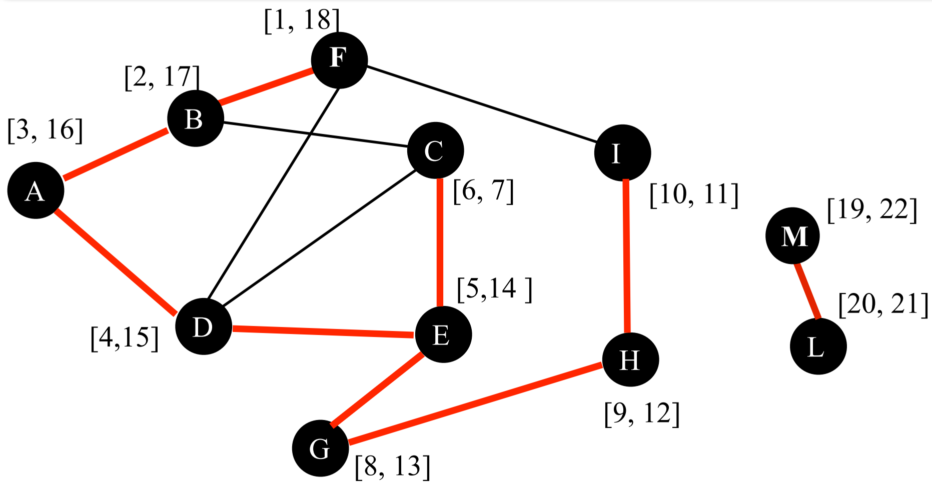
© Alberto Montresor

Esempio



© Alberto Montresor

Esempio

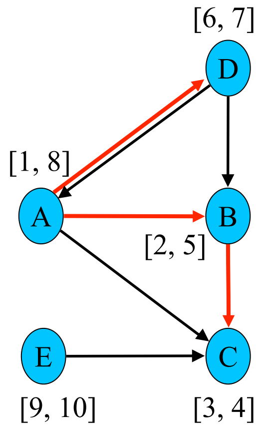


© Alberto Montresor

Classificazione degli archi

dfs-schema(GRAPH G, NODE u)

esamina il nodo u prima (caso *pre-visita*)
$time \leftarrow time + 1;\ \ dt[u] \leftarrow time$
foreach $v \in G.\mathsf{adj}(u)$ **do**
  esamina l'arco (u, v) di qualsiasi tipo
  if $dt[v] = 0$ **then**
    esamina l'arco (u, v) in T
    dfs-schema(g, v)
  else if $dt[u] > dt[v]$ **and** $ft[v] = 0$ **then**
    esamina l'arco (u, v) all'indietro
  else if $dt[u] < dt[v]$ **and** $ft[v] \neq 0$ **then**
    esamina l'arco (u, v) in avanti
  else
    esamina l'arco (u, v) di attraversamento
esamina il nodo u dopo (caso *post-visita*)
$time \leftarrow time + 1;\ \ ft[u] \leftarrow time$



© Alberto Montresor

Classificazione degli archi

- **Cosa serve la classificazione?**
  - E' possibile dimostrare alcune proprietà, che poi possono essere sfruttate negli algoritmi
- **Esempio:**
  - DAG non hanno archi all'indietro (dimostrare)

boolean ciclico(GRAPH G, NODE u)

$time \leftarrow time + 1;\ \ dt[u] \leftarrow time$
foreach $v \in G.\mathsf{adj}(u)$ **do**
  if $dt[v] = 0$ **then**
    if ciclico(G, v) **then return true**
  else if $dt[u] > dt[v]$ **and** $ft[v] = 0$ **then**
    return true
$time \leftarrow time + 1;\ \ ft[u] \leftarrow time$
return false;

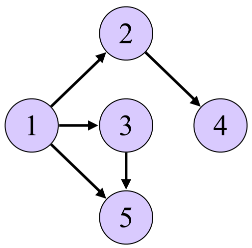
Variabile *time*:

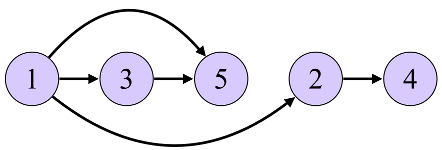
- variabile globale, oppure
- passata per riferimento)

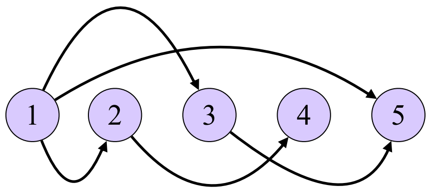
© Alberto Montresor

Ordinamento topologico

- **Dato un DAG *G* (direct acyclic graph), un ordinamento topologico su *G* è un ordinamento lineare dei suoi vertici tale per cui:**
  - se *G* contiene l'arco *(u,v)*, allora *u* compare prima di *v* nell'ordinamento
  - Per transitività, ne consegue che se *v* è raggiungibile da *u*, allora *u* compare prima di *v* nell'ordinamento
  - Nota: possono esserci più ordinamenti topologici







© Alberto Montresor

Ordinamento topologico

- **Problema:**
  - Fornire un algoritmo che dato un grafo orientato aciclico, ritorni un ordinamento topologico
- **Soluzioni**
  - Diretta (INEFFICIENTE!)
  - Basata su DFS

Trovare un vertice che non abbia alcun arco incidente in ingresso

Stampare questo vertice e rimuoverlo, insieme ai suoi archi

Ripetere la procedura finché tutti i vertici risultano rimossi

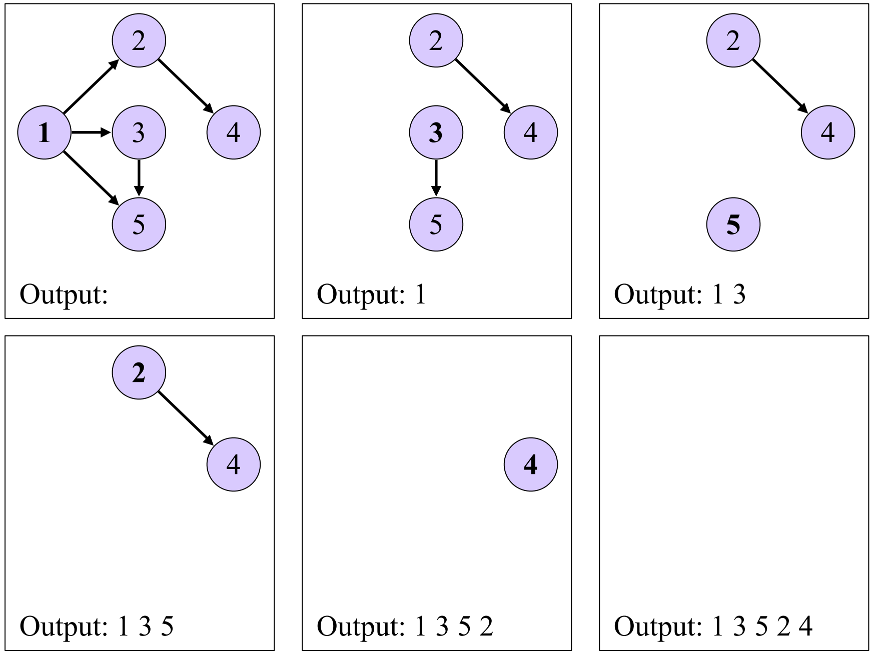
Esercizio: scrivere lo pseudocodice per questo algoritmo

Qual è la complessità?

- con matrici di adiacenza
- con liste di adiacenza

© Alberto Montresor

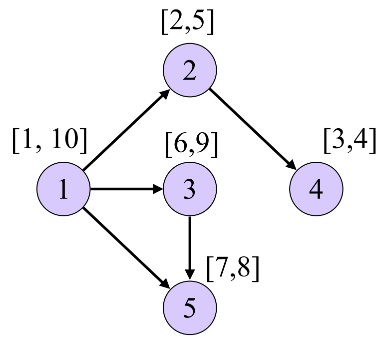
Soluzione diretta



© Alberto Montresor

Ordinamento topologico basato su DFS

- **Algoritmo**
  - Si effettua una DFS
  - L'operazione di visita consiste nell'aggiungere il vertice alla testa di una lista"at finish time"
  - Si restituisce la lista di vertici
- **Output**
  - Sequenza ordinata di vertici, in ordine inverso di finish time
- **Perché funziona?**



© Alberto Montresor

Ordinamento topologico basato su DFS - Liste

```
topSort(GRAPH G, SEQUENCE L)
    boolean[] visitato ← boolean[1 ... G.n]
    foreach u ∈ G.V() do visitato[u] ← false
    foreach u ∈ G.V() do
        if not visitato[u] then
            ts-dfs(G, u, visitato, L)

ts-dfs(GRAPH G, NODE u, boolean[] visitato, SEQUENCE L)
    visitato[u] ← true
    foreach v ∈ G.adj(u) do
        if not visitato[v] then
            ts-dfs(G, v, visitato, L)
    L.insert(L.head(), u)
```

© Alberto Montresor

Ordinamento topologico basato su DFS - Stack

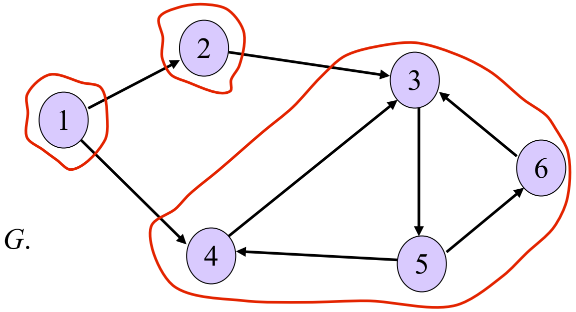
```
topSort(GRAPH G, STACK S)
    boolean[] visitato ← boolean[1 ... G.n]
    foreach u ∈ G.V() do visitato[u] ← false
    foreach u ∈ G.V() do
        if not visitato[u] then
            ts-dfs(G, u, visitato, S)

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

© Alberto Montresor

Definizioni: Grafi fortemente connessi e componenti fortemente connesse

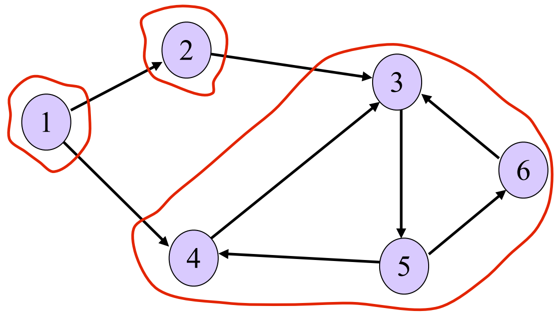
- **In un grafo orientato G**
  - G è *fortemente connesso* ⇔ esiste un cammino da ogni vertice ad ogni altro vertice
  - Un grafo $G' = (V', E')$ è una *componente fortemente connessa* di G ⇔ è un sottografo di G fortemente connesso e massimale
- **Definizioni**
  - G' è un *sottografo* di G ($G' \subseteq G$) se e solo se $V' \subseteq V$ e $E' \subseteq E$
  - G' è *massimale* ⇔ non esiste un sottografo G'' di G che sia fortemente connesso e "più grande" di G', ovvero tale per cui $G' \subseteq G'' \subseteq G$.



© Alberto Montresor

Soluzione errata

- **Proviamo ad utilizzare l'algoritmo per le componenti connesse?**
  - Risultati variano a seconda del nodo da cui si parte



© Alberto Montresor

Componenti fortemente connesse

- **Algoritmo (Kosaraju, 1978)**
  1. Effettua una DFS di G
  2. Calcola il grafo trasposto G^T
  3. Effettua una DFS di G^T esaminando i vertici in ordine inverso di tempo di fine
  4. Fornisci i vertici di ogni albero della foresta depth-first prodotta al passo 3. come una diversa SCC
- **Grafo trasposto**
  - Dato un grafo $G = (V, E)$, il grafo trasposto $G^T = (V, E^T)$ è formato dagli stessi nodi, mentre gli archi hanno direzioni invertite: i.e,
    - $E^T = \{(u,v) \mid (v,u) \in E\}$
- **Costo computazione**
  - $O(m+n)$

© Alberto Montresor

Componenti fortemente connesse - algoritmo

integer[ ] scc(GRAPH G)

STACK $S \leftarrow$ topSort(G) % Prima visita

GRAPH $G^T \leftarrow$ Graph() % Calcolo grafo trasposto

foreach $u \in G$.V() **do** G^T.insertNode(u)

foreach $u \in G$.V() **do**

  foreach $v \in G$.adj(u) **do**

    G^T.insertEdge(v, u)

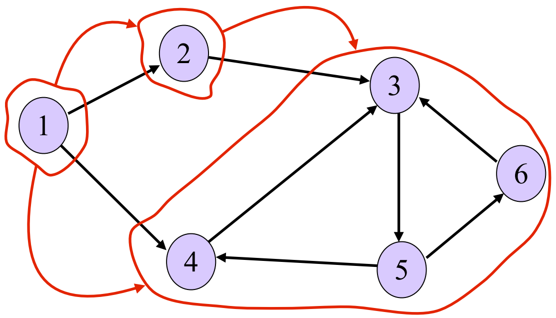
return cc(G^T, S) % Seconda visita

© Alberto Montresor

Componenti fortemente connesse - dimostrazione correttezza

- **Domanda**
  - Che rapporto c'è fra le SCC del grafo G e del suo trasposto G^T ?
- **Grafo delle componenti $G^c = (V^c, E^c)$**
  - $V^c = \{C_1, C_2, \ldots, C_k\}$, dove C_i è l'i-esima componente fortemente connessa di G;
  - $E^c = \{(C_i, C_j) : \exists (u_i, u_j) \in E : u_i \in C_i, u_j \in C_j\}$

Il grafo delle componenti è aciclico?

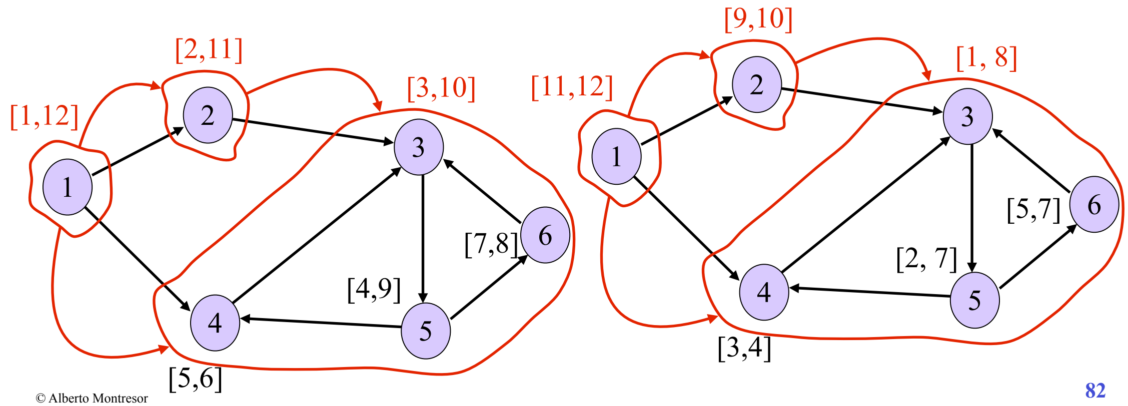


Componenti fortemente connesse - dimostrazione correttezza

- **Si può estendere il concetto di *dt* e *ft* al grafo delle componenti**

$$dt(C) = \min\{dt[u] : u \in C\}$$
$$ft(C) = \max\{ft[u] : u \in C\}$$

- **Teorema**
  - Siano C e C' due componenti distinte nel grafo orientato $G = (V, E)$. Supponiamo che esista un arco $(C, C') \in E^c$. Allora $ft(C) > ft(C')$

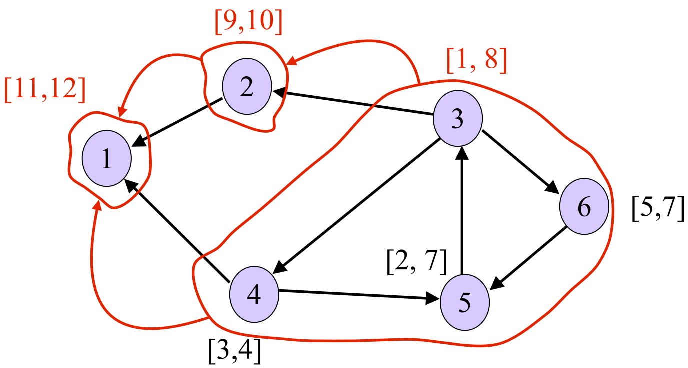


Componenti fortemente connesse - dimostrazione correttezza

- **Si può estendere il concetto di *dt* e *ft* al grafo delle componenti**

$$dt(C) = \min\{dt[u] : u \in C\}$$
$$ft(C) = \max\{ft[u] : u \in C\}$$

- **Corollario**
  - Siano C e C' due componenti distinte nel grafo orientato $G = (V, E)$. Supponiamo che esista un arco $(u,v) \in E^T$ con $u \in C$, $v \in C'$. Allora $ft(C) < ft(C')$
    - $(u,v) \in E^T \Rightarrow$
      $(v,u) \in E \Rightarrow$
      $(C', C) \in E^c \Rightarrow$
      $ft(C) < ft(C')$



Componenti fortemente connesse

- **Algoritmo di Tarjan (1972)**
  - Tarjan, R. E. "Depth-first search and linear graph algorithms", *SIAM Journal on Computing* 1(2): 146–160 (1972)
  - Algoritmo con costo $O(m+n)$ come Kosaraju
  - E' preferito a Kosaraju in quanto necessita di una sola visita e non richiede il grafo trasposto