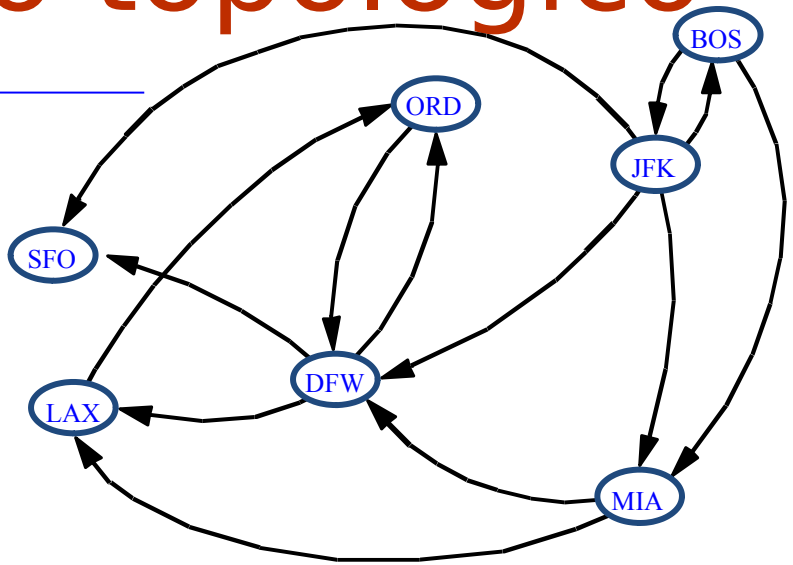
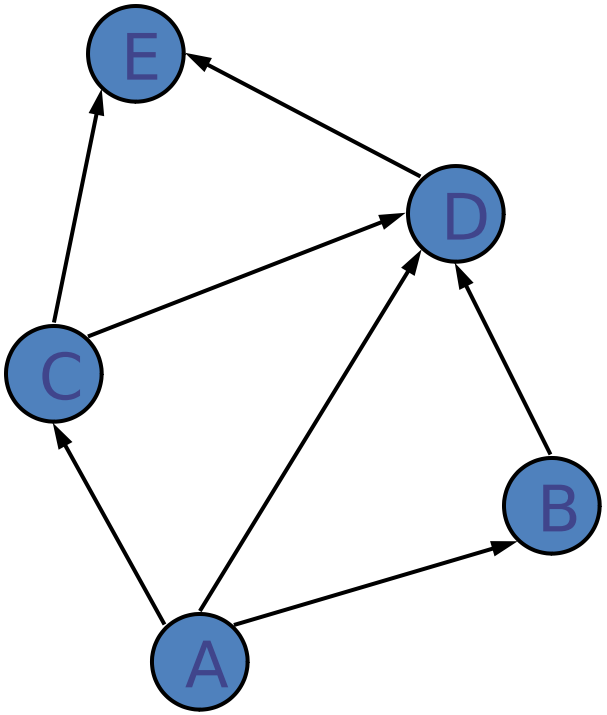
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

SCC, Chiusura transitiva Ordinamento topologico

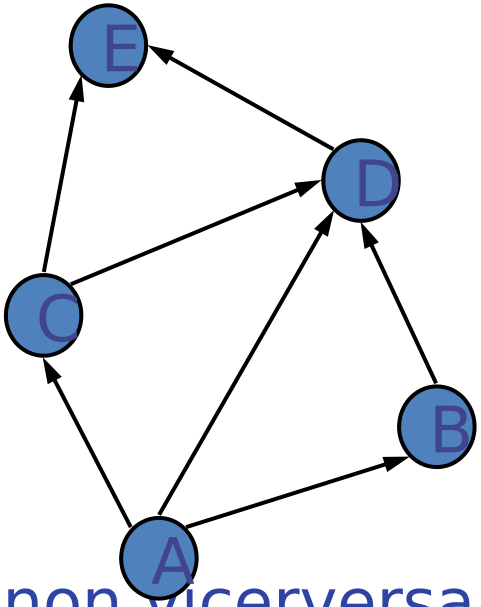


Grafi diretti

- Digrafo (digraph)
  → grafo diretto
- Applicazioni
  - Reti viarie
  - Trasporti
  - Attività di pianificazione



Proprietà

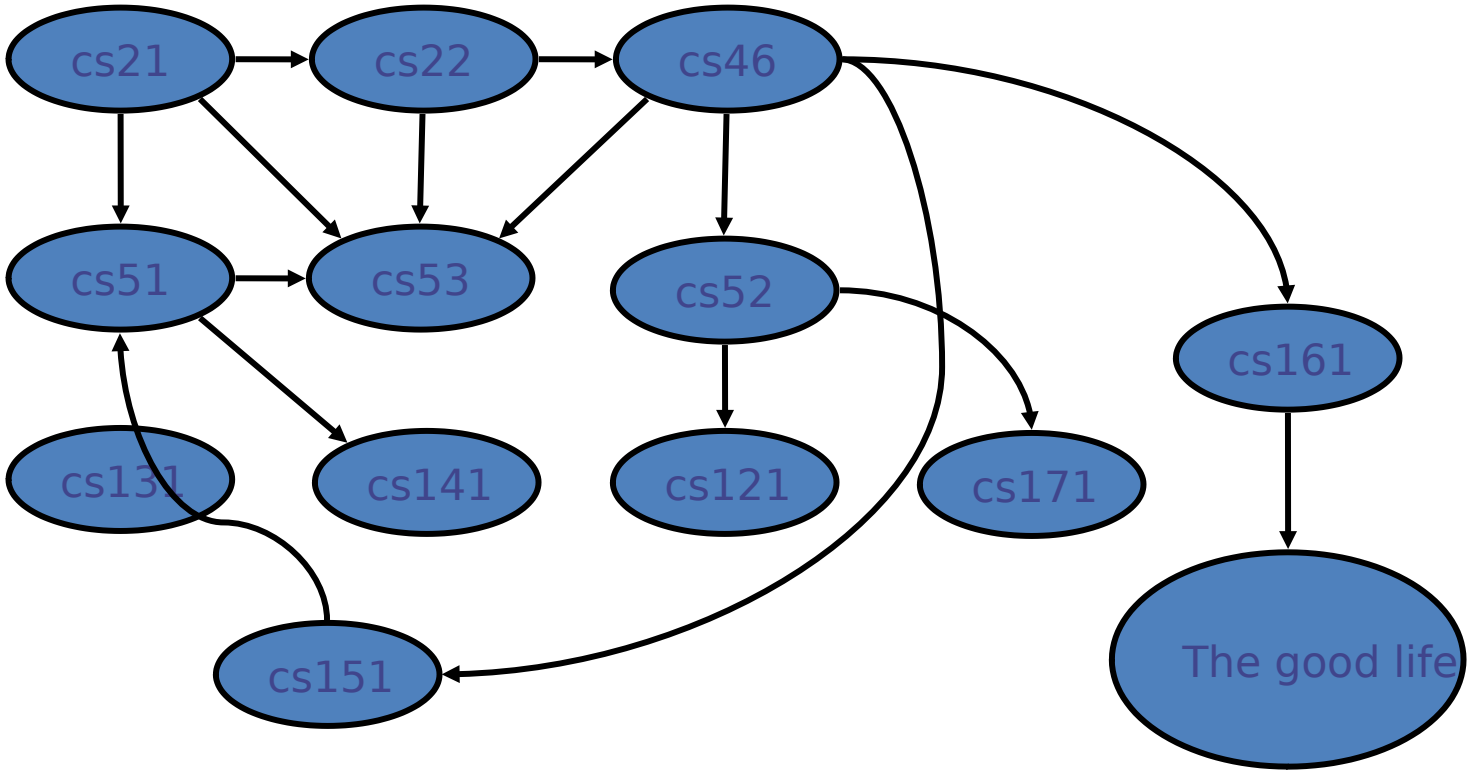


- Grafo G=(V,E) tale che
  - Ogni arco ha una direzione:
  - L'arco (a,b) va da a verso b, ma non viceversa

  Se G è semplice, $m <= n(n-1)$
- Può essere utile mantenere liste di adiacenza separate per archi entranti e uscenti

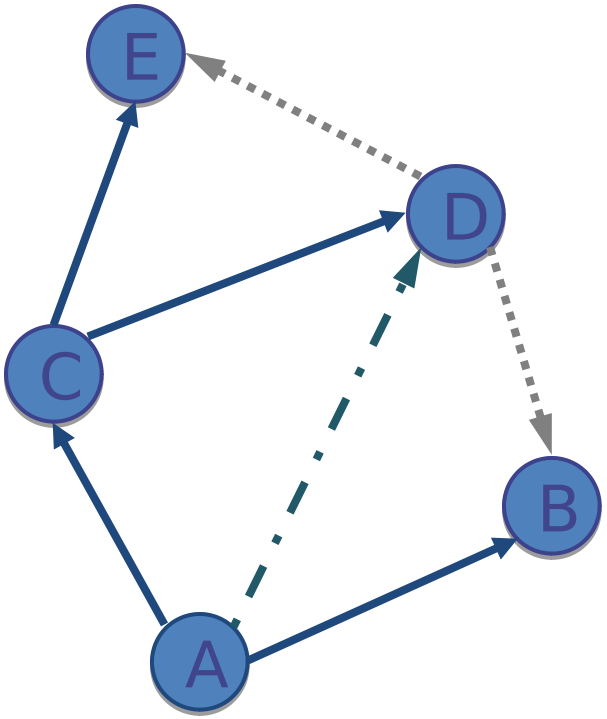
Esempio di applicazione

- Scheduling di lavori: l'arco (a,b) significa che il a va completato prima dell'inizio di b



DFS diretta

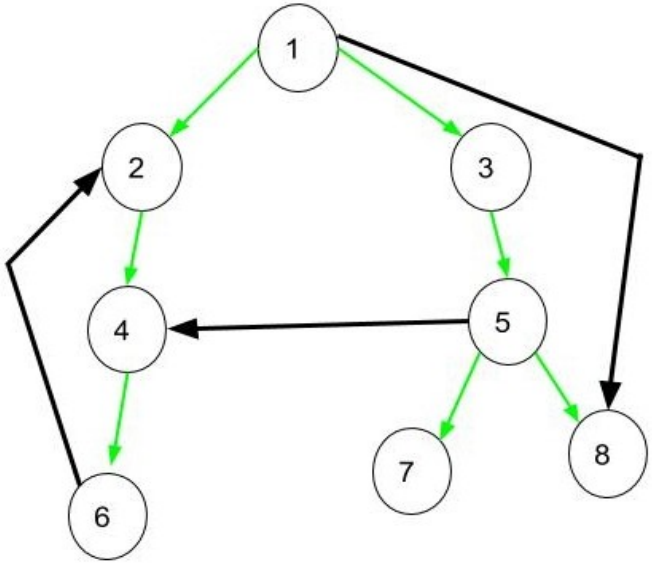
- La DFS in generale funziona anche per grafi diretti
- Gli archi sono attraversati soltanto nella loro direzioni
- 4 tipi di archi nella DFS diretta
  - Archi discovery
  - Archi back
  - Archi forward
  - Archi cross
- Una DFS a partire da un vertice s calcola il sottoinsieme dei vertici raggiungibili da s



DFS diretta

4 tipi di archi nella DFS diretta

- Archi discovery: archi presenti nell'albero costruito da DFS (archi verdi)
- Archi back: (u,v) t.c. v è antenato di u ma non fa parte dell'albero DFS; es. arco da 6 a 2 [presenza di archi back segnala che ci sono cicli nel grafo]
- Archi forward: (u,v) t.c. v è discendente ma non fa parte dell'albero DFS; es. arco da1 a 8
- Archi cross: arco che connette due nodi che non hanno relazioni di antenato o discendente fra loro; es. arco da 5 a 4



Ordine visita dei nodi iniziando DFS da nodo 1: 1,2,4,6,3,5,7,8

Caratterizzazione degli archi

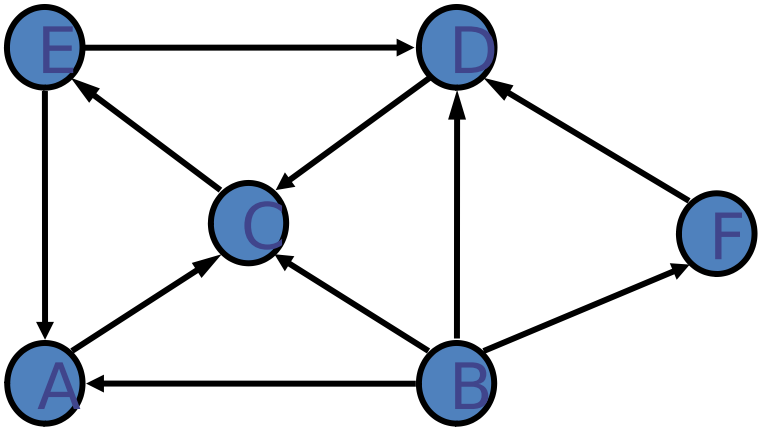
```
Algorithm sweep_aux(Graph.Node node, int time) {
        if(node.state != Graph.Node.Status.UNEXPLORED)
            return;

        node.state = Graph.Node.Status.EXPLORING;
        node.timestamp = time;
        for(Graph.Node cur : node.outEdges) {
            print("\t" + node.value + "(" + node.timestamp + ")->" + cur.value + "(" + cur.timestamp +
")");

            if (cur.state == Graph.Node.Status.EXPLORED) {
                if (node.timestamp < cur.timestamp)
                    System.out.println("FORWARD");
                else
                    System.out.println("CROSS");
                }
                else if (cur.state == Graph.Node.Status.EXPLORING)
                    System.out.println("BACK");
                else {
                    System.out.println("TREE");
                    sweep_aux(cur, time + 1);
                }
            }
            node.state = Graph.Node.Status.EXPLORED;
            return;
     }
```

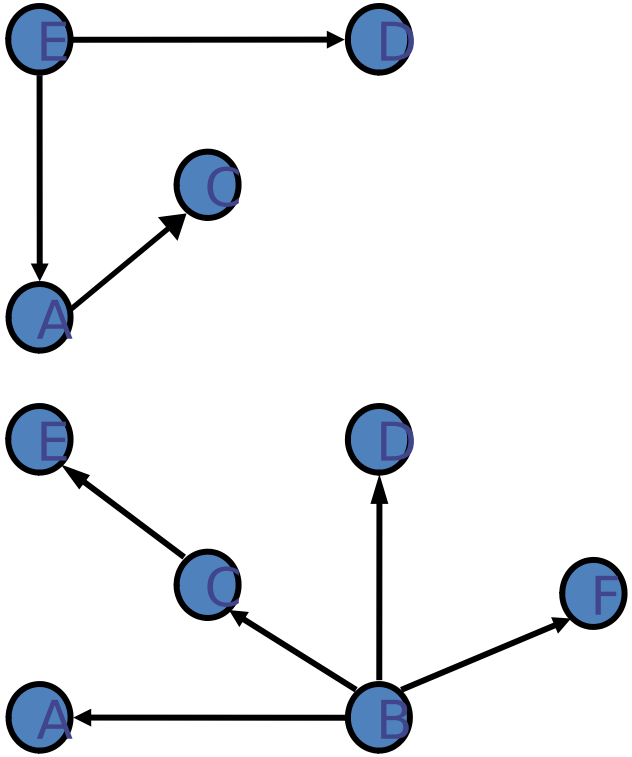
DDFS - pseudocodice

```
DDFS(Graph.Node nd, List forward) {
        if(nd.state == Graph.Node.Status.EXPLORING)
           return;

        if(nd.state == Graph.Node.Status.EXPLORED)
           return;

        nd.state = Graph.Node.Status.EXPLORING;

        for(Graph.Node cur : nd.outEdges)
           DDFS(cur, forward);

        nd.state = Graph.Node.Status.EXPLORED;
        forward.addFirst(nd);
        return forward;
}
```

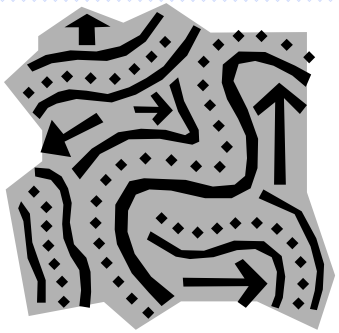
Raggiungibilità

Albero DFS con radice in v: vertici raggiungibili da v usando cammini diretti

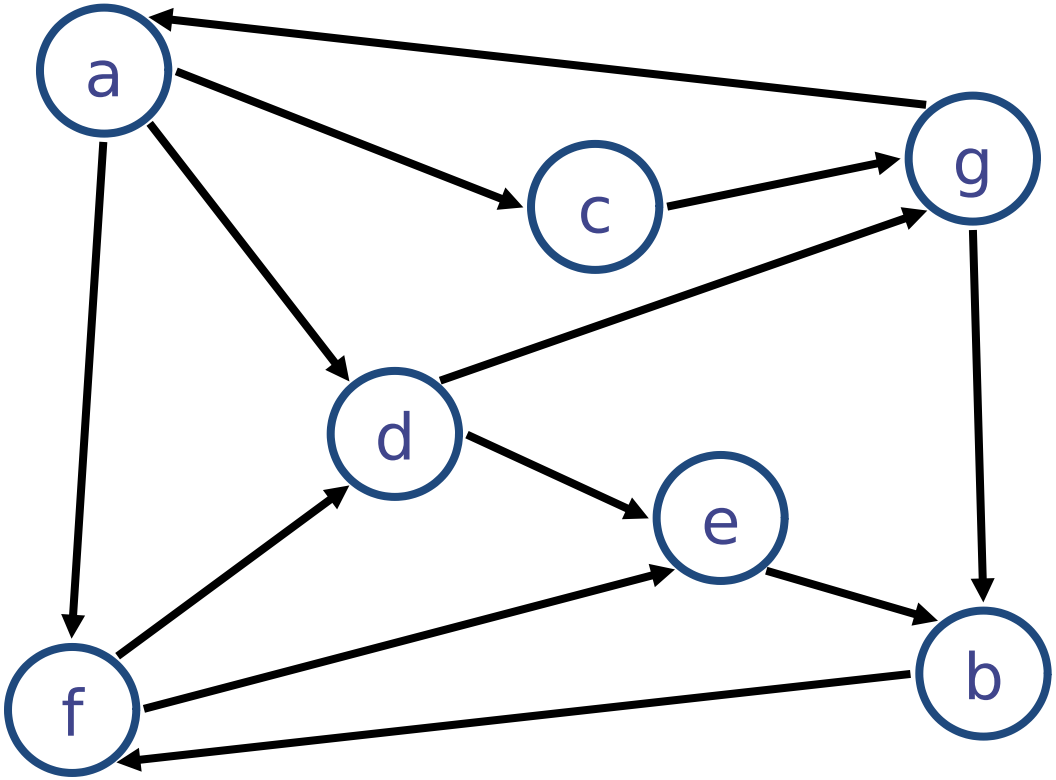




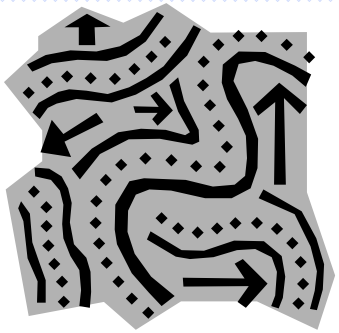
Connettività **forte**



- Da ***ogni*** vertice è possibile raggiungere ***ogni*** altro vertice

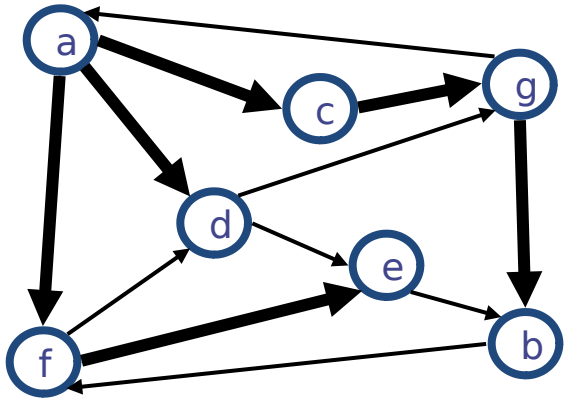


Connettività forte: algoritmo

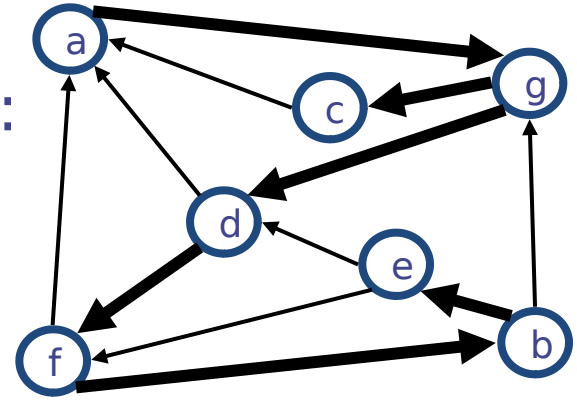


- Scegli un vertice v di G
- DFS(v, G)
  - Se esiste w non visitato --> return “no”
- Sia G’ uguale a G ma con gli archi invertiti
- DFS(v, G’) from v in G’
  - Se esiste w non visitato --> return “no”
    - Else, return “yes”
- Complessità: O(n+m)

G:



G’:



Connettività forte - pseudocodice/1

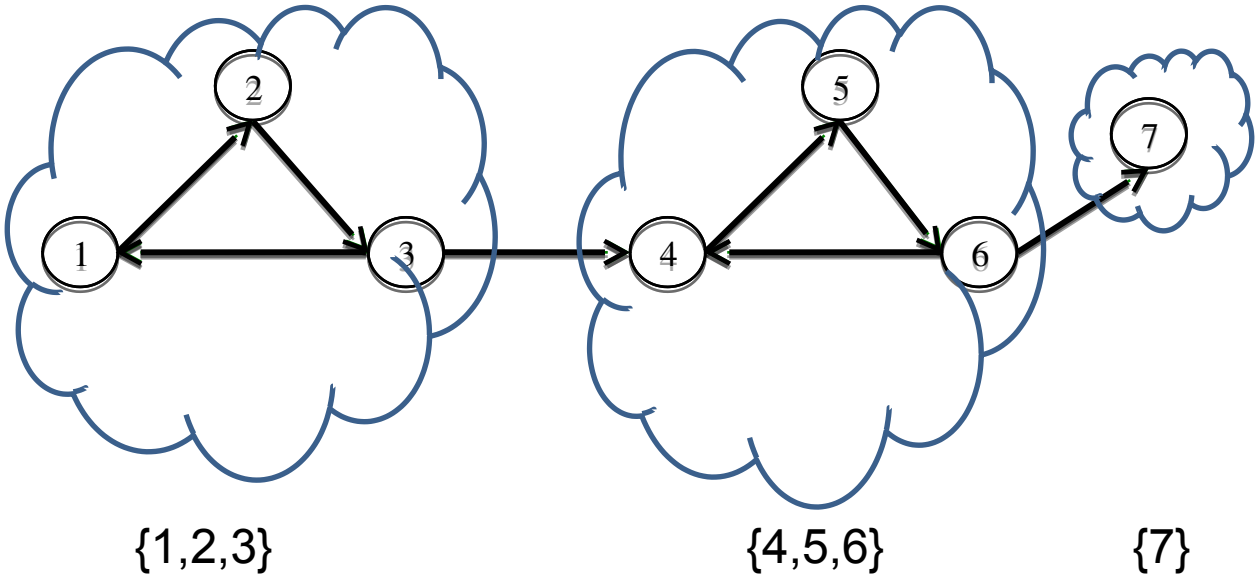
```
transposedDFS(Graph.Node nd, List reverse) {
        if(nd.state == Graph.Node.Status.EXPLORING)
            return;

        if(nd.state == Graph.Node.Status.EXPLORED)
            return;

        nd.state = Graph.Node.Status.EXPLORING;
        for(Graph.Node cur : nd.inEdges)
            transposedDFS(cur, reverse);

        reverse.addLast(nd);
        nd.state = Graph.Node.Status.EXPLORED;
    }
```

Connettività forte - pseudocodice/2

```
stronglyConnectedComponent(Graph g, Node v, List ret) {

        // First DFS
        List forward = new List();
        DDFS(v, forward); // forward contiene i nodi raggiungibili da v

        for(Graph.Node n: forward)
                n.state = Graph.Node.Status.UNEXPLORED;

        // Second DFS on the transposed graph
        List reverse = new List();
        transposedDFS(v, reverse); // reverse → nodi da cui è possibile raggiungere v

        for(Graph.Node cur : reverse)
                if forward.contains(cur) // I nodi presenti in entrambe le liste appartengono alla CC di v
                        ret.add(cur);
                else
                        cur.state = Graph.Node.Status.UNEXPLORED; // non fa parte della CC di cur
         return ret; // Nodi facenti parte della componente fortemente connessa di v
}
```

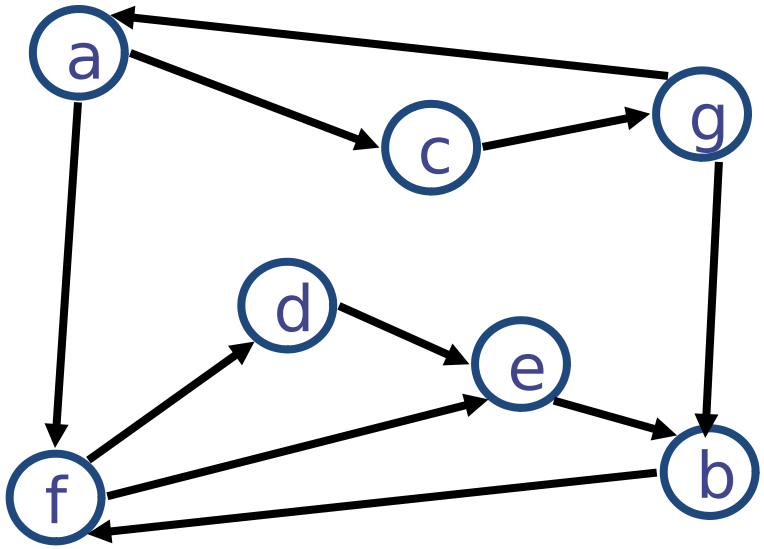
Componenti fortemente connesse

- Sottografi massimali che sono fortemente connessi



Componenti fortemente connesse

- Sottografi massimali che sono fortemente connessi

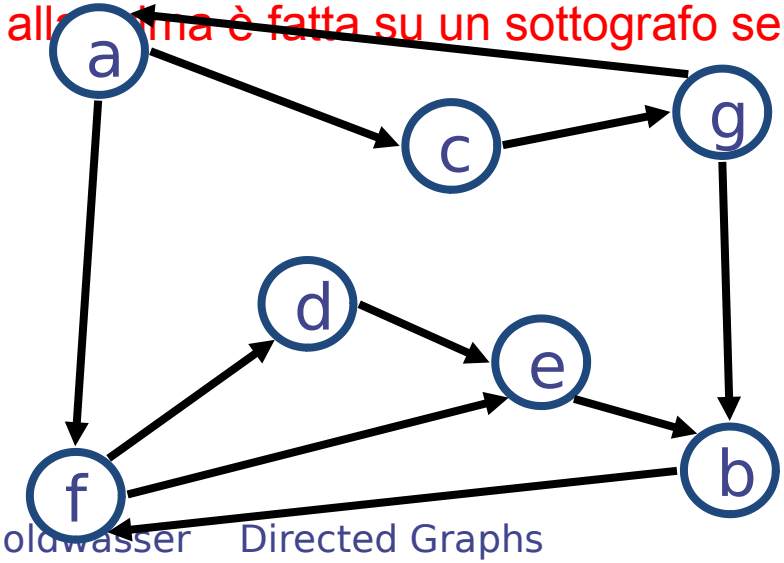


{ a , c , g }

{ f , d , e , b }

Componenti fortemente connesse

- Applica DFS da nodo v su G e su G' (grafo con archi invertiti): intersezione dei due insiemi di nodi trovati determina la componente connessa a cui appartine v : se questa include tutti i nodi il grafo è fortemente connesso → fine
- Altrimenti: trova un nodo w che non appartiene alla componente fortemente connessa di v e ripeti DFS a partire da w
- Ripeti fino a quando tutti i nodi sono in qualche componente
- Costo è O(n(n+m)): nel caso peggiore devo eseguire O(n) DFS e ciascuna costa O(m+n) (NOTA: questa stima è approssimata: non tiene conto che ogni DFS successiva alla prima è fatta su un sottografo sempre più piccolo)

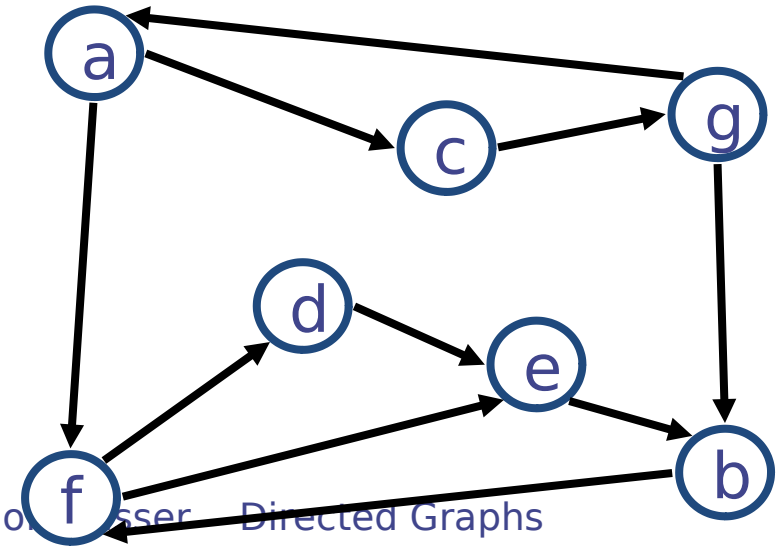


{ a , c , g }

{ f , d , e , b }

Componenti fortemente connesse

- Applica DFS da nodo v su G e su G' (grafo con archi invertiti): se grafo è fortemente connesso fine
- Altrimenti: trova un nodo w che non appartiene alla componente fortemente connessa di v e ripeti DFS a partire da w
- Ripeti fino a quando tutti i nodi sono in qualche componente
- Costo è O(n(n+m)): nel caso peggiore devo eseguire O(n) DFS e ciascuna costa O(m+n)
- Algoritmo di Kosaraju (vedi libro ed esercitazione su grafi diretti): O(n+m)



{ a , c , g }

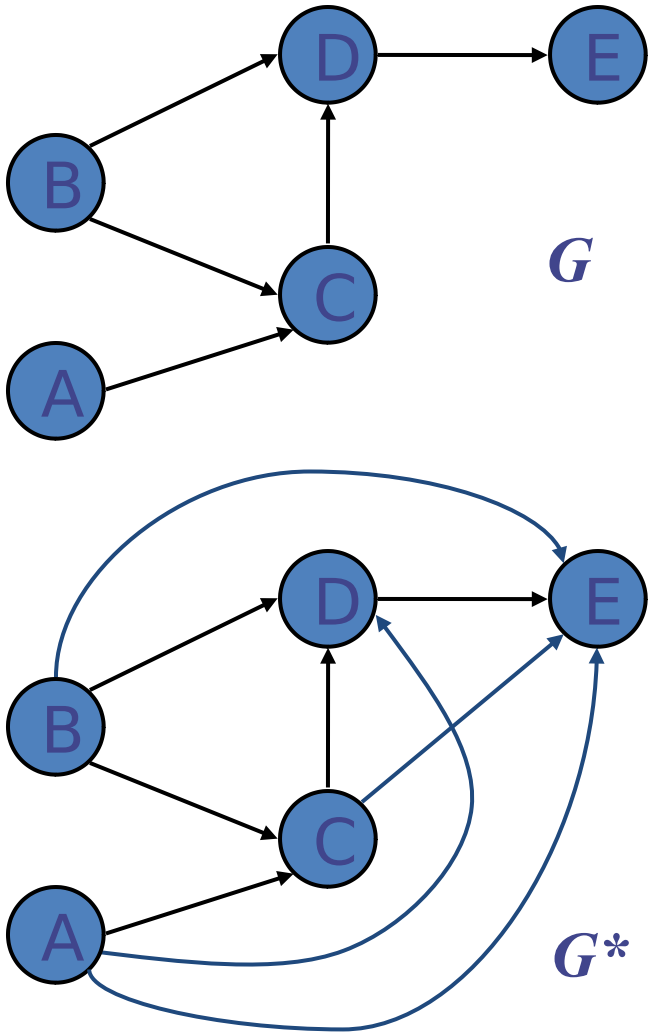
{ f , d , e , b }

Componenti fortemente connesse - pseudocodice

```
Algorithm SCC(Graph g) {
      g.resetStatus();
      for(Graph.Node n : g.getNodes())
            if(n.state == Graph.Node.Status.UNEXPLORED) {
                  List ret = new List();
                  stronglyConnectedComponent(Graph g, Node v, List ret);
                  print(“Componente connessa: ”, ret);
            }
}
```

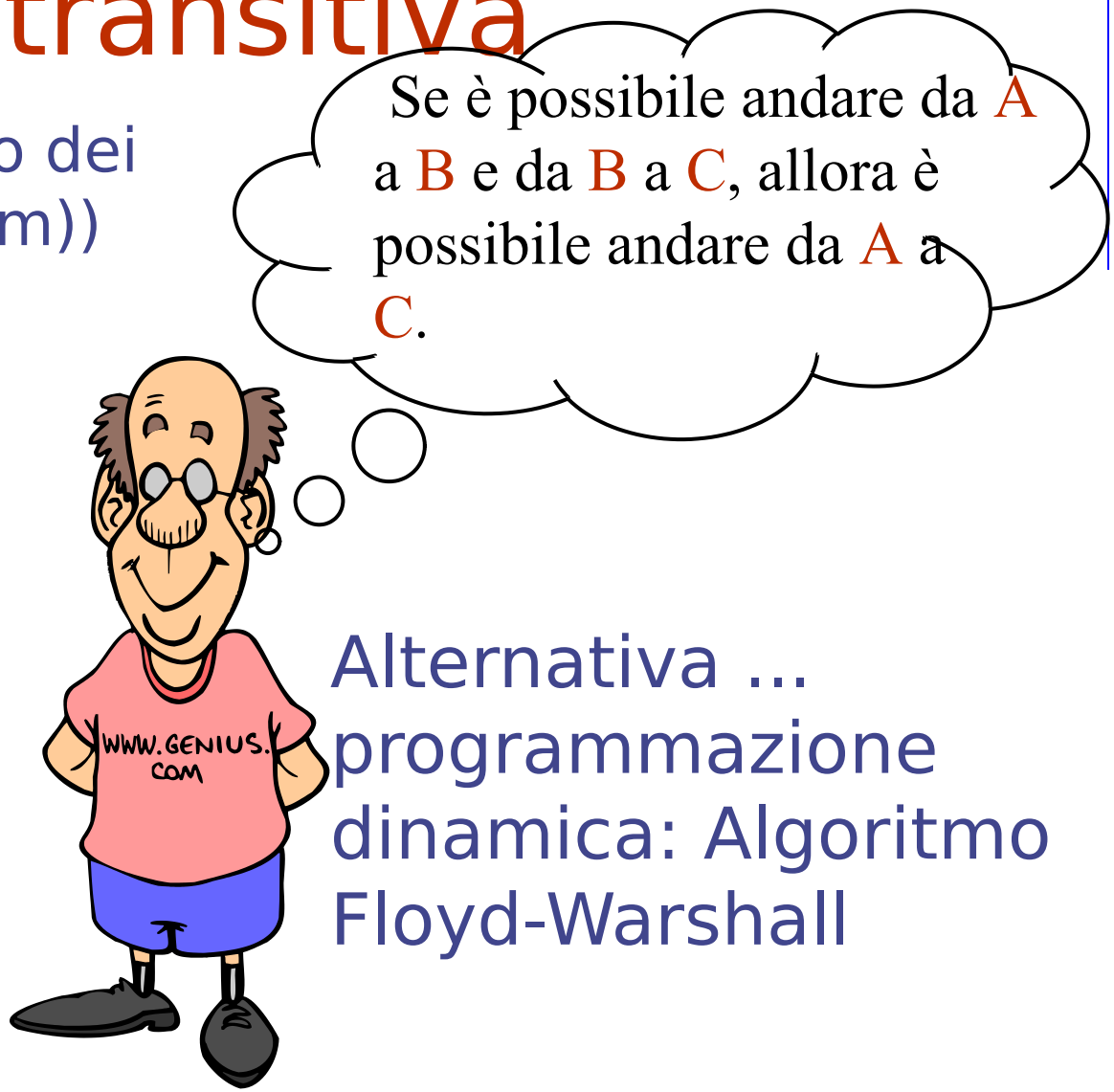
Chiusura transitiva

- Dato un digrafo G, la chiusure transitiva di G è il digrafo G^* tale che
  - G^* ha gli stessi vertici di G
    - Se G ha un cammino diretto da u a v ($u \longrightarrow v$), G^* ha un arco diretto da u a v
- Chiusura transitiva → informazione di raggiungibilità



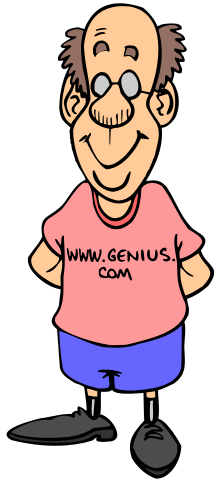
Calcolo della chiusura transitiva

- DFS da ognuno dei vertici O(n(n+m))
- Sfruttare la matrice di adiacenza
  - $O(n^{c+1})$
  - c è almeno 2.37

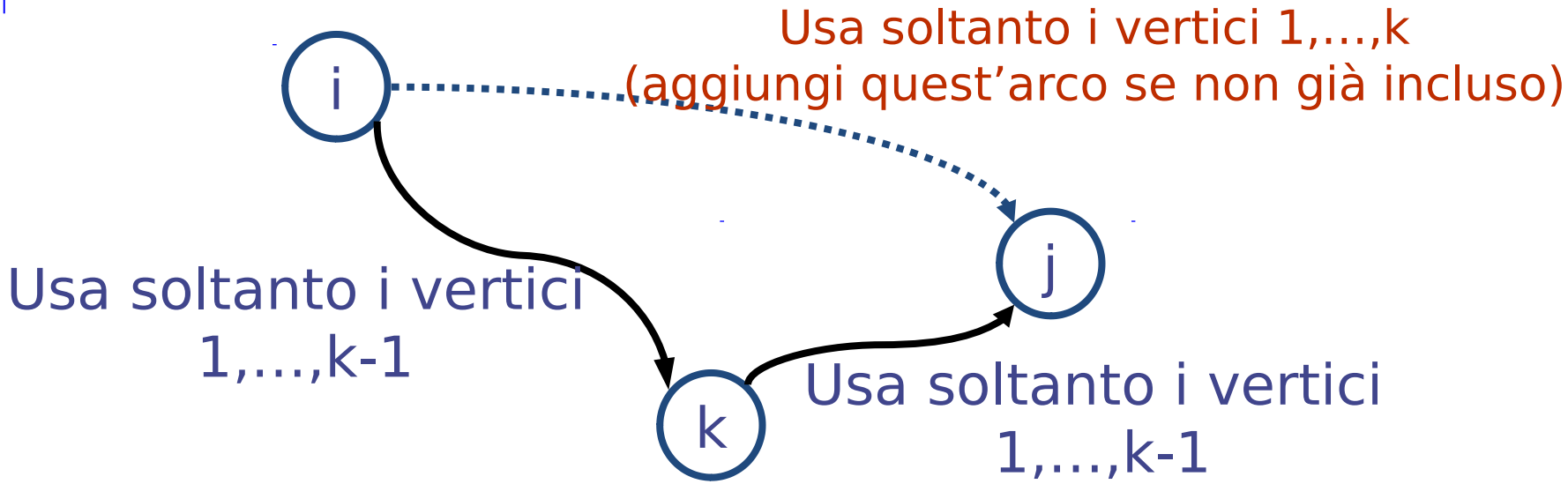


Alternativa ... programmazione dinamica: Algoritmo Floyd-Warshall

Floyd-Warshall chiusura transitiva



- Idea #1: numera i vertici 1, 2, …, n.
- Idea #2: considera i cammini che usano 1, 2, …, k, come vertici intermedi:



Algoritmo Floyd-Warshall

Arco diretto da i a j in G_k se

- Arco diretto (i, j) in G_{k-1} oppure:
- Archi diretti (i, k) e (k, j) in G_{k-1}

} Programmazione dinamica

Algorithm FloydWarshall(G): G_k = grafo usato ad iterazione k

$G_0 = G$

for k = 1 to n {

    $G_k = G_{k-1}$

    for (i, j con $i \neq k$ e $j \neq k$)

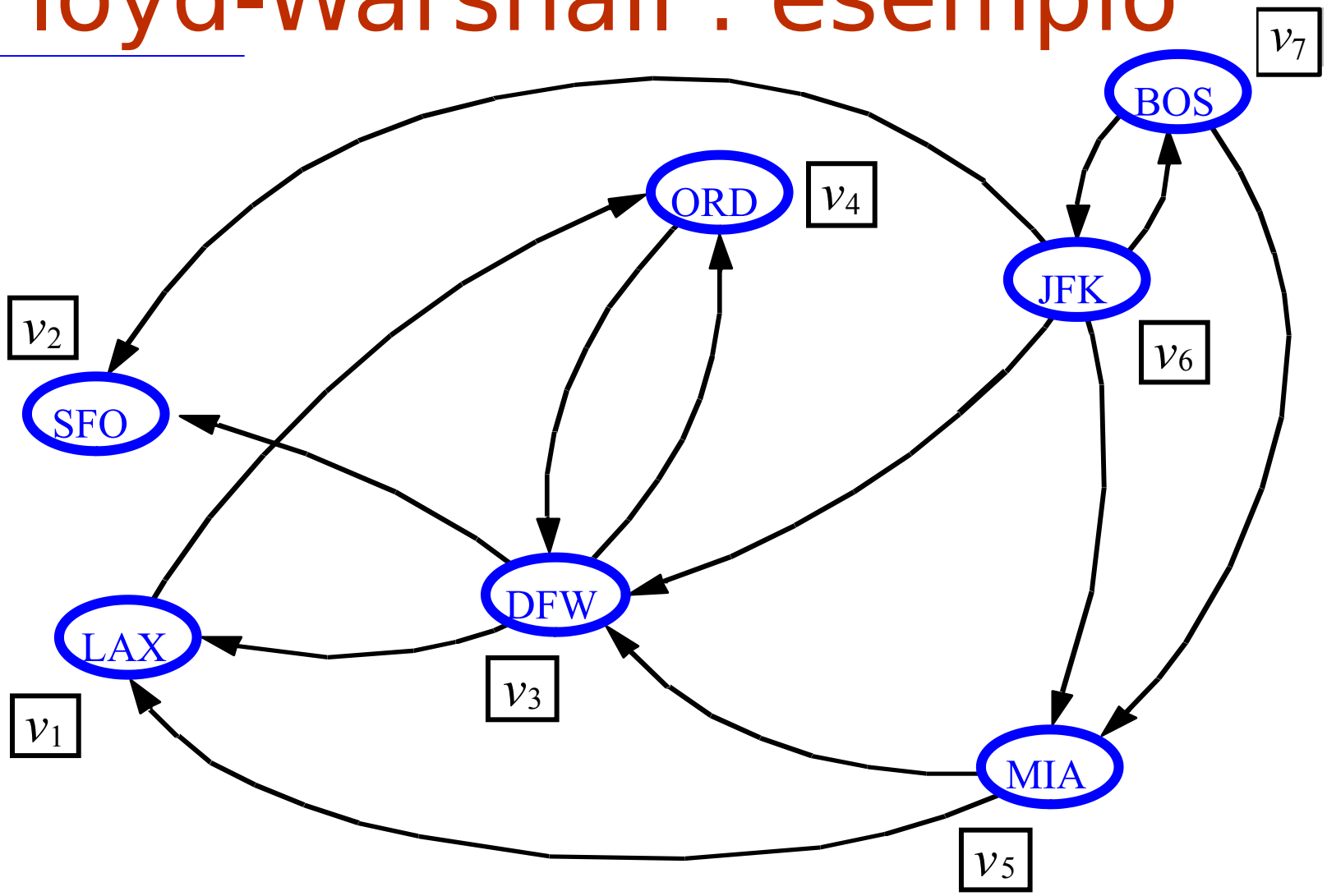
        if ((i, j) non appartiene a G_{k-1})

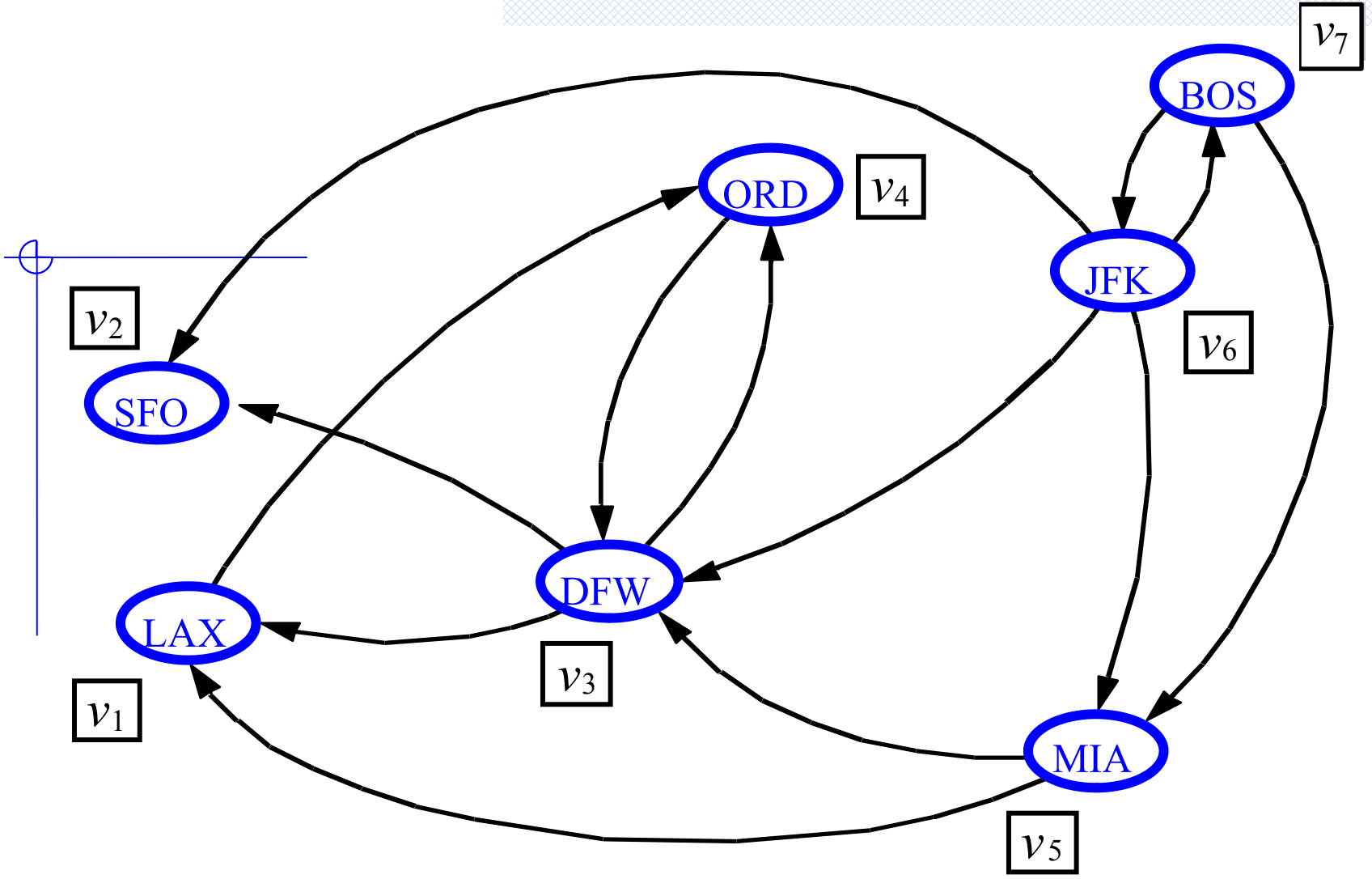
            if ((i, k) Є G_{k-1} and (k, j) Є G_{k-1})

                <Aggiungi (i, j) a G_k>

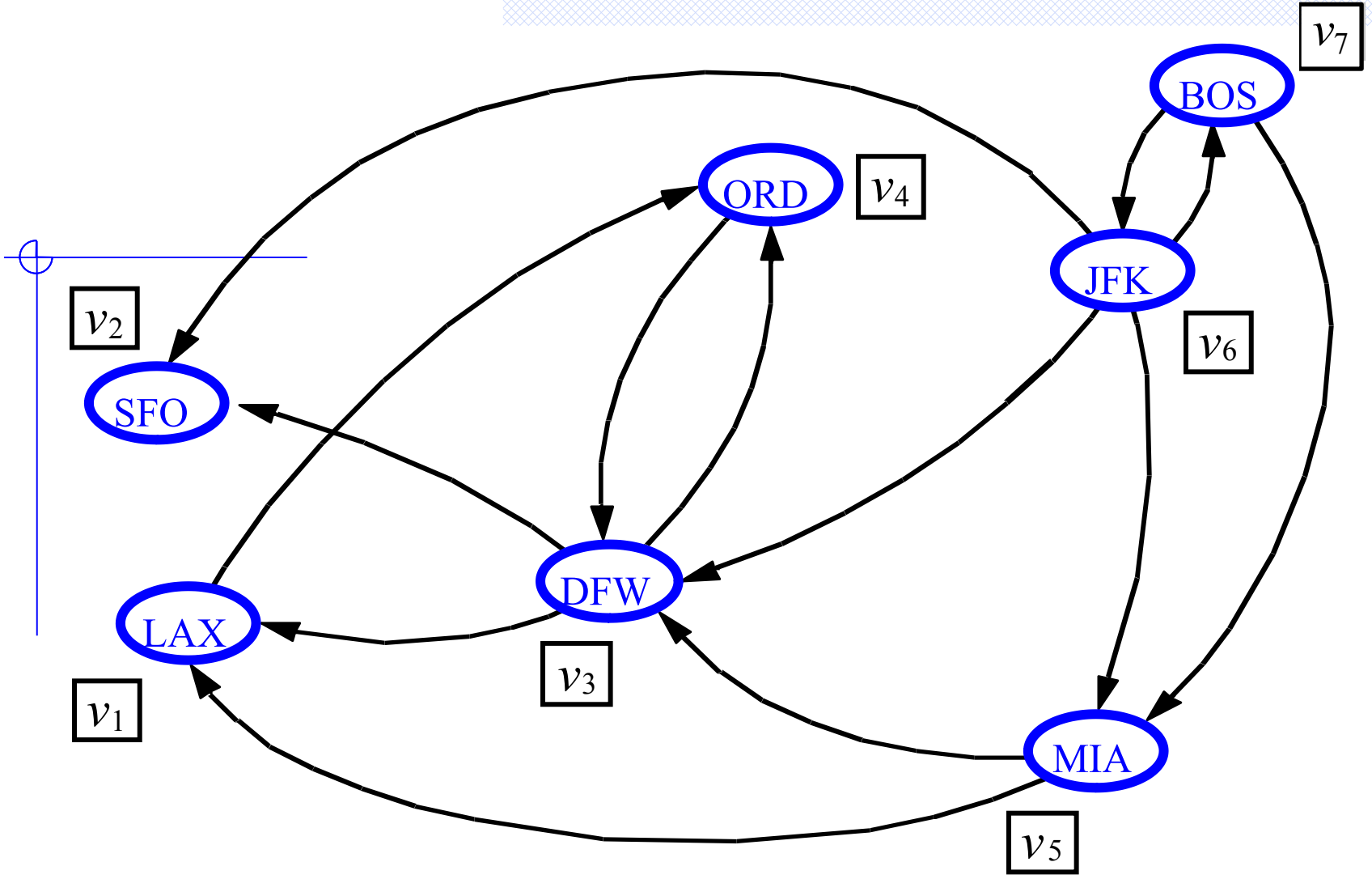
Determinare il costo computazionale

Floyd-Warshall : esempio



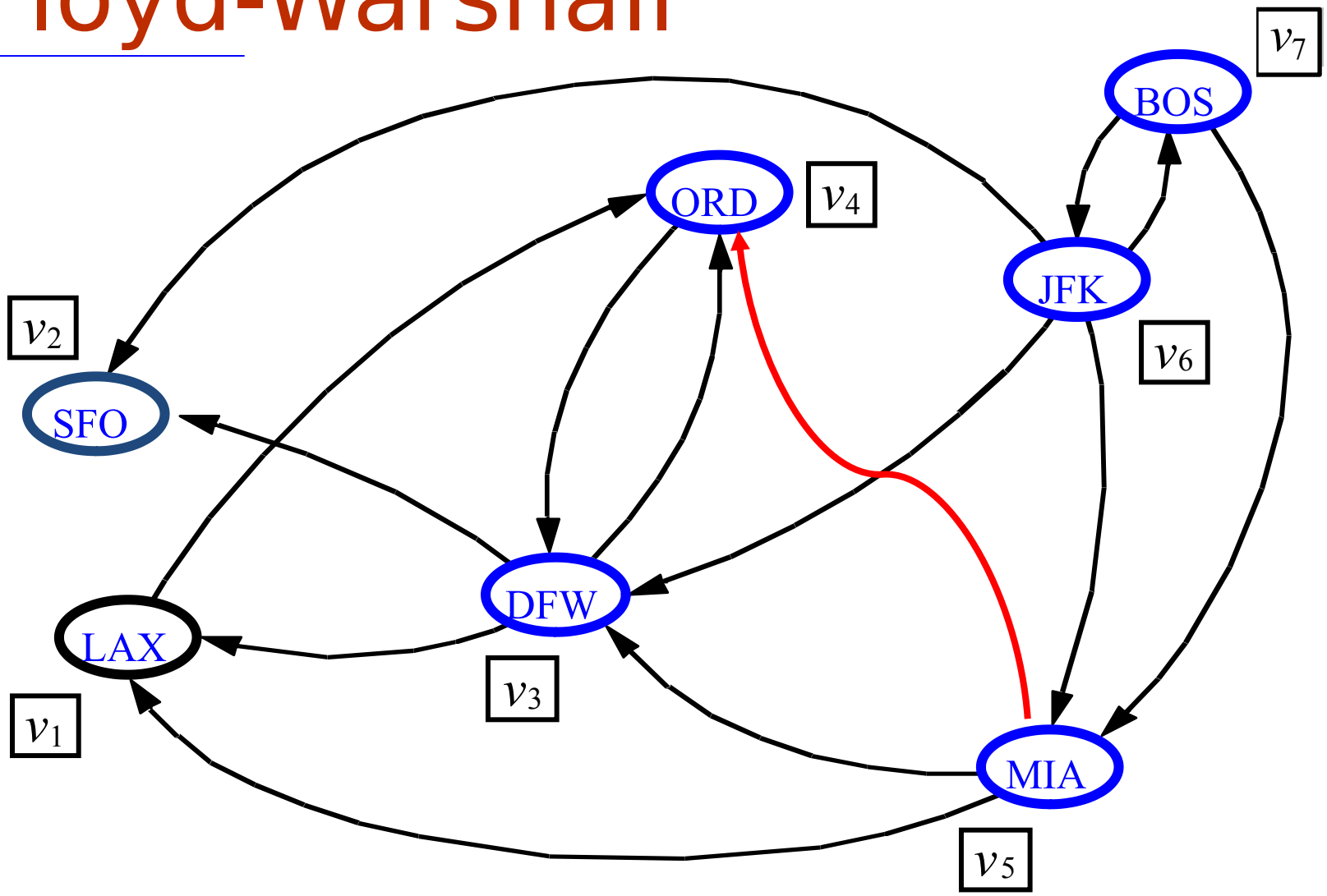


Per k=1 cerchiamo cammini che passano per il nodo 1
Dato che (2,1) e (1,2) non sono archi non aggiungiamo nulla che connette 2
Possiamo raggiungere 4 da 3 passando per 1; ma esiste già un cammino diretto

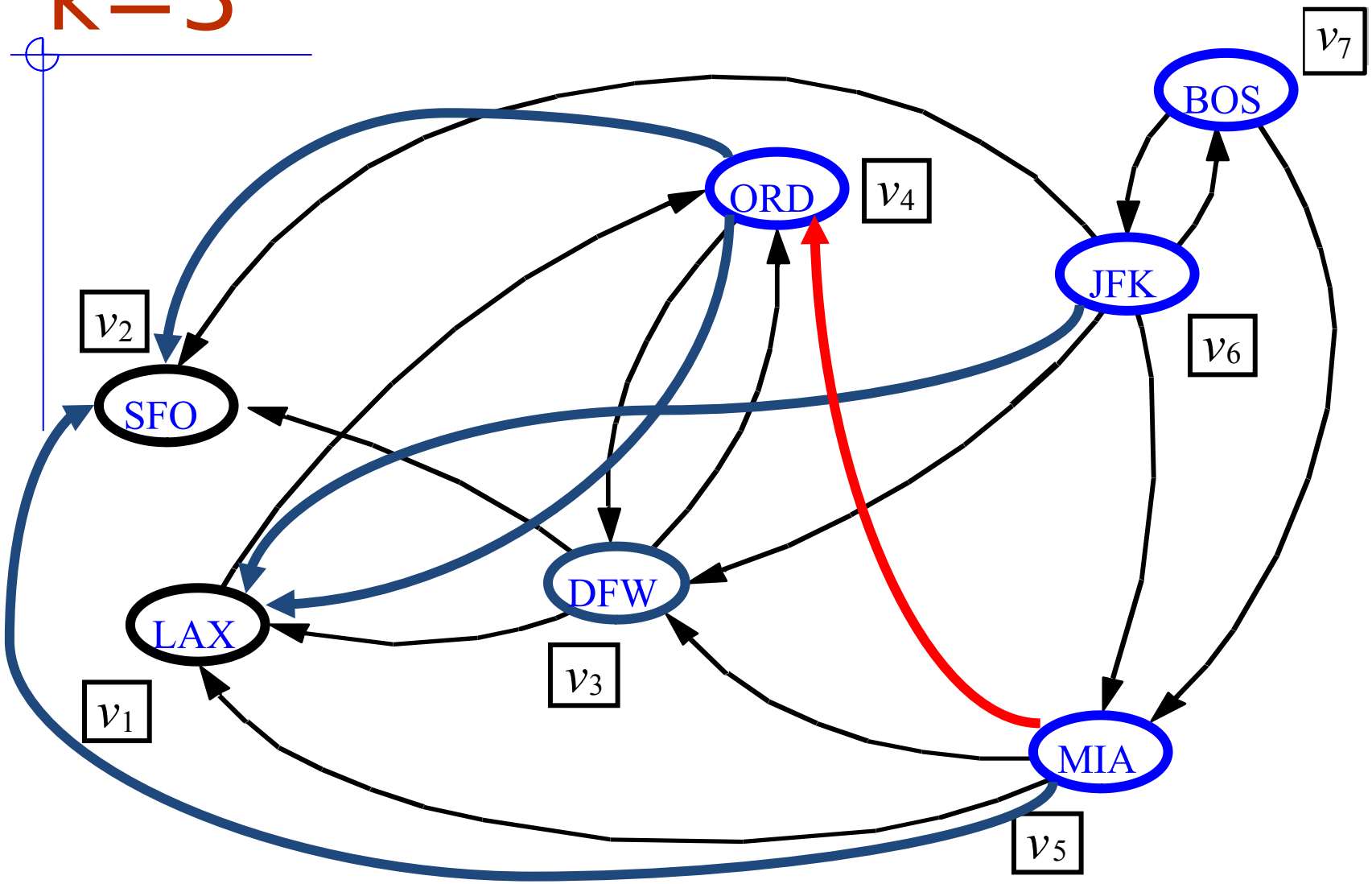


Per k=1 cerchiamo cammini che passano per il nodo 1
Dato che (5,1) e (1,4) sono archi e non esiste cammino da 5 a 4 allora
Abbiamo trovato un cammino da 5 a 4!

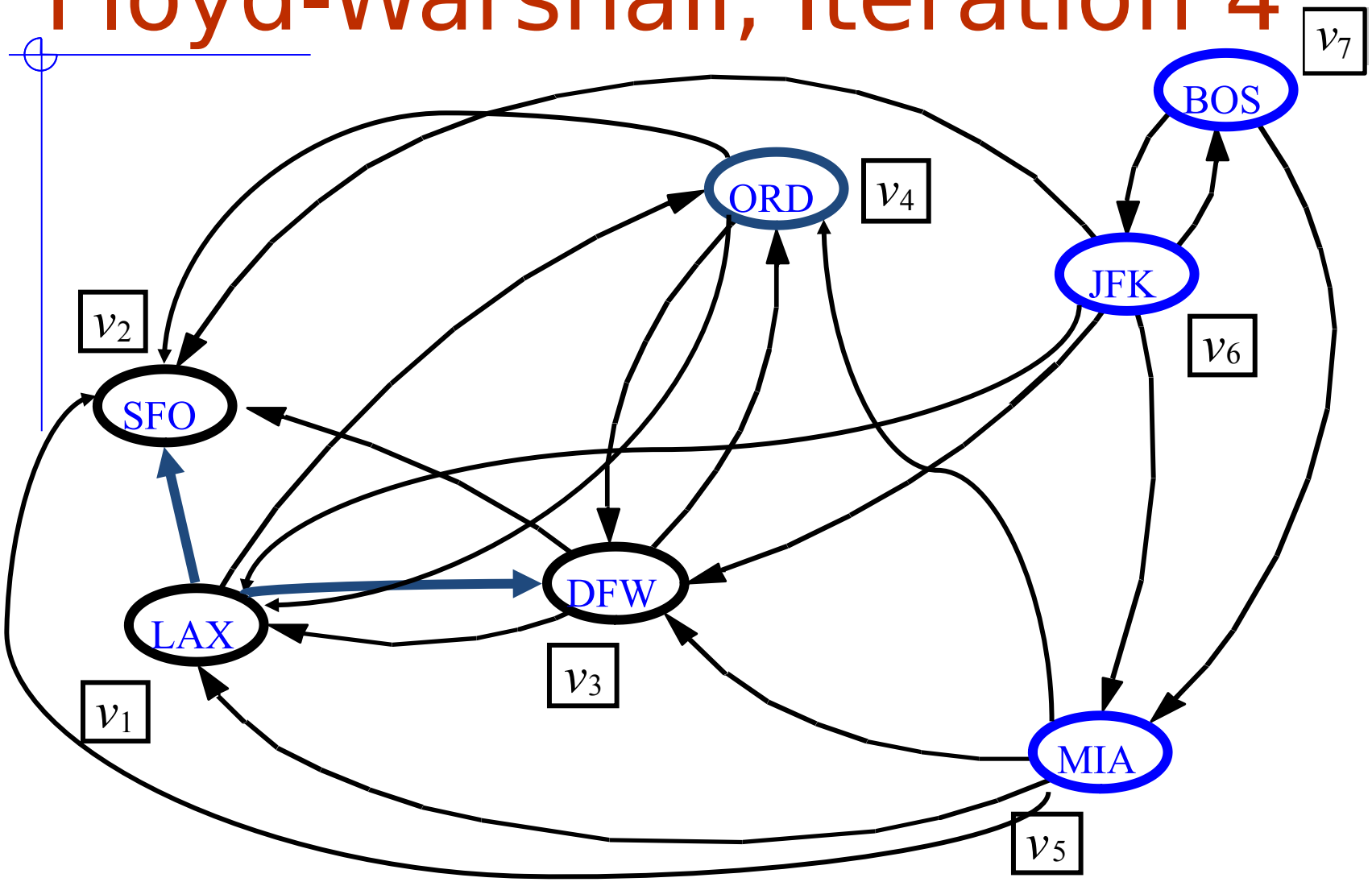
Floyd-Warshall



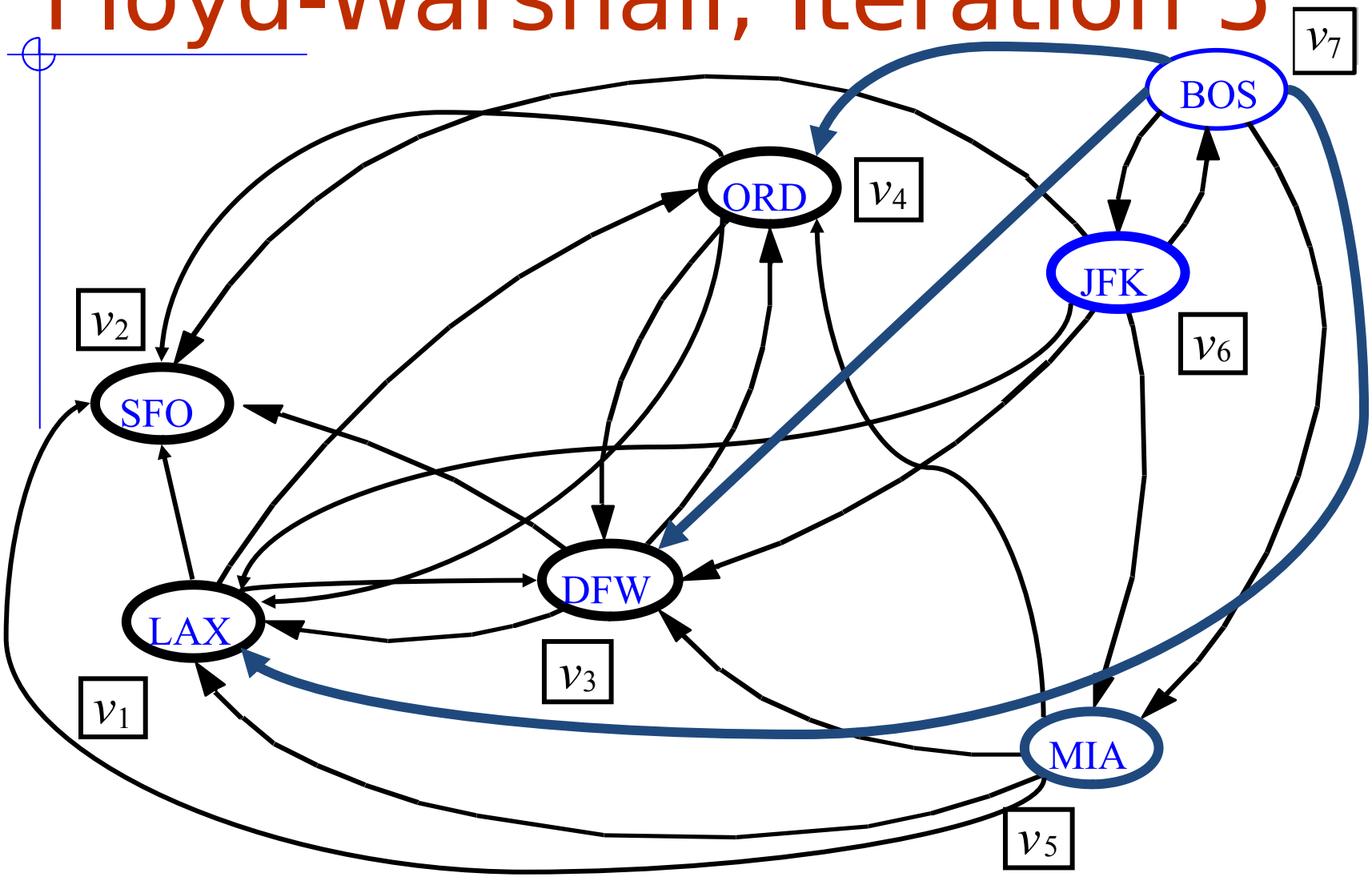
Floyd-Warshall, Iteration k=3



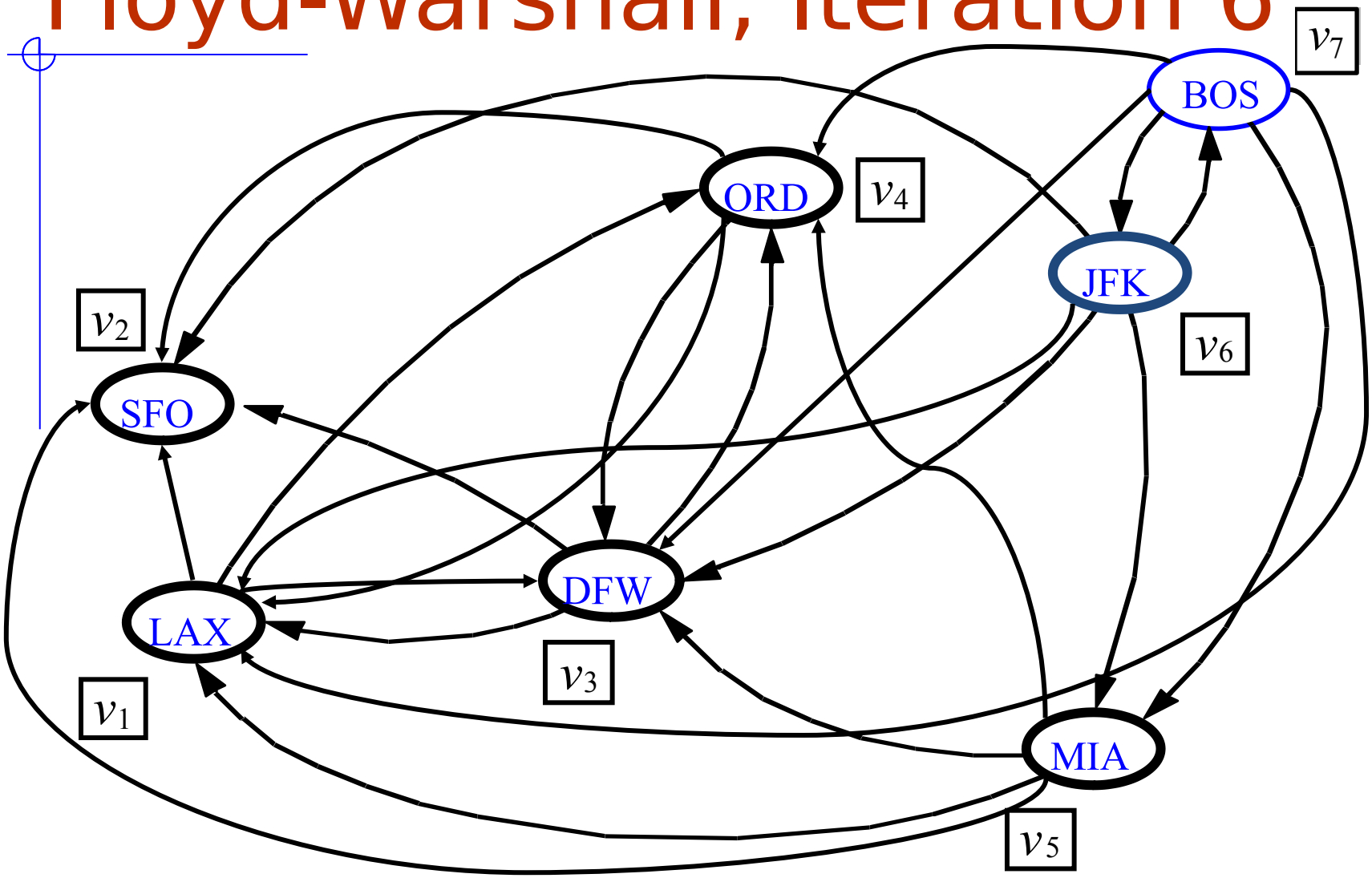
Floyd-Warshall, Iteration 4



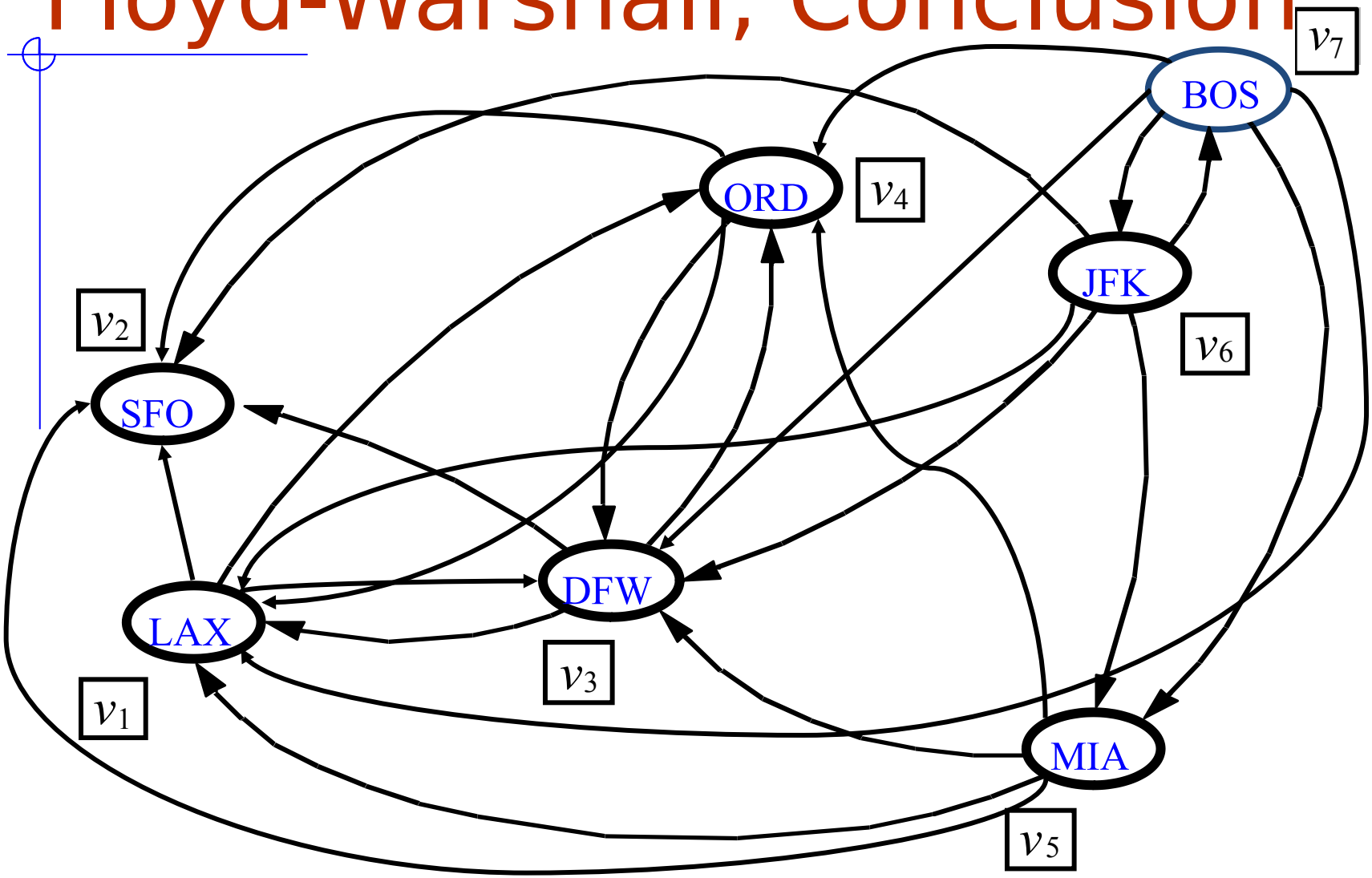
Floyd-Warshall, Iteration 5



Floyd-Warshall, Iteration 6

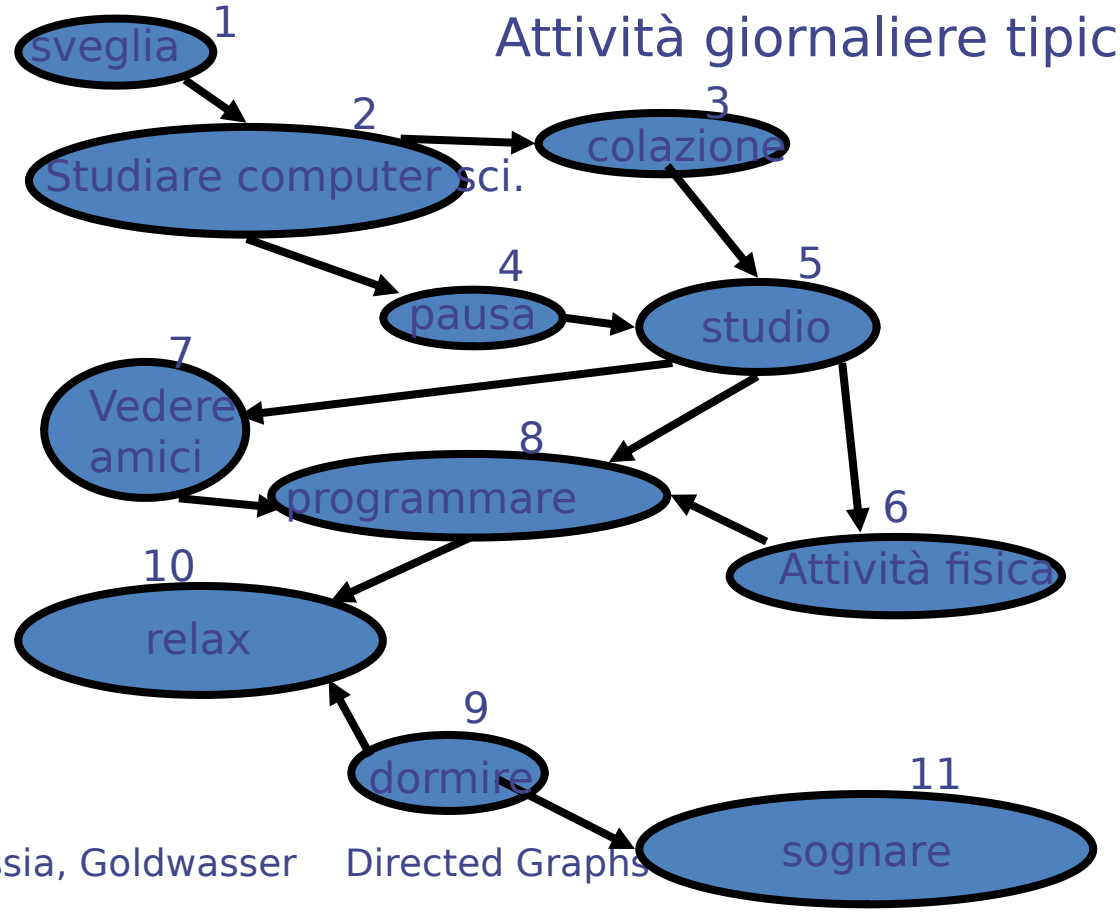


Floyd-Warshall, Conclusion



Ordinamento topologico

- Numera i vertici, in modo tale che (u,v) in E implichi u < v

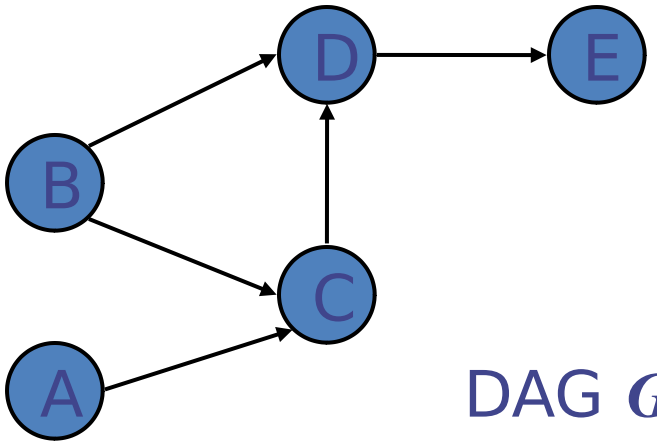


DAG e Ordinamento Topologico

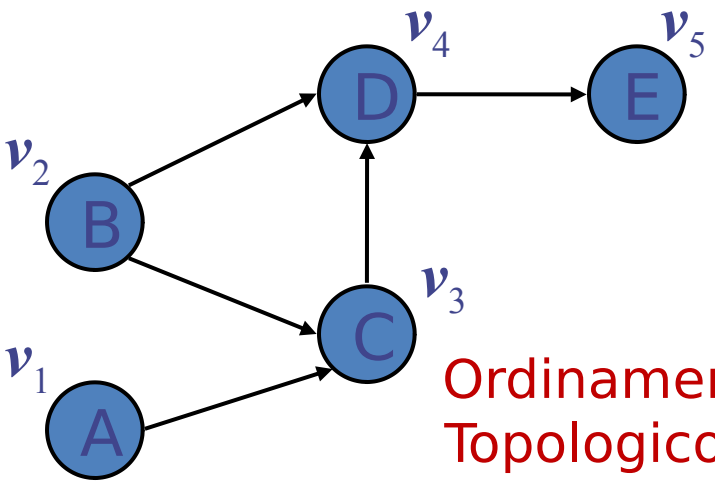
- **Grafo diretto aciclico (DAG): digrafo privo di cicli diretti**
- Ordinamento topologico: numerazione $v_1, \ldots, v_n$ dei vertici tale che per ogni arco (v_i, v_j), abbiamo $i < j$
- Esempio: se il grafo rappresenta le precedenze tra task, un ordinamento topologico è un sequenziamento dei task che soddisfa i vincoli di precedenza

Teorema

Un digrafo ammette un ordinamento topologico se e solo se è un DAG



DAG G



Ordinamento Topologico di G

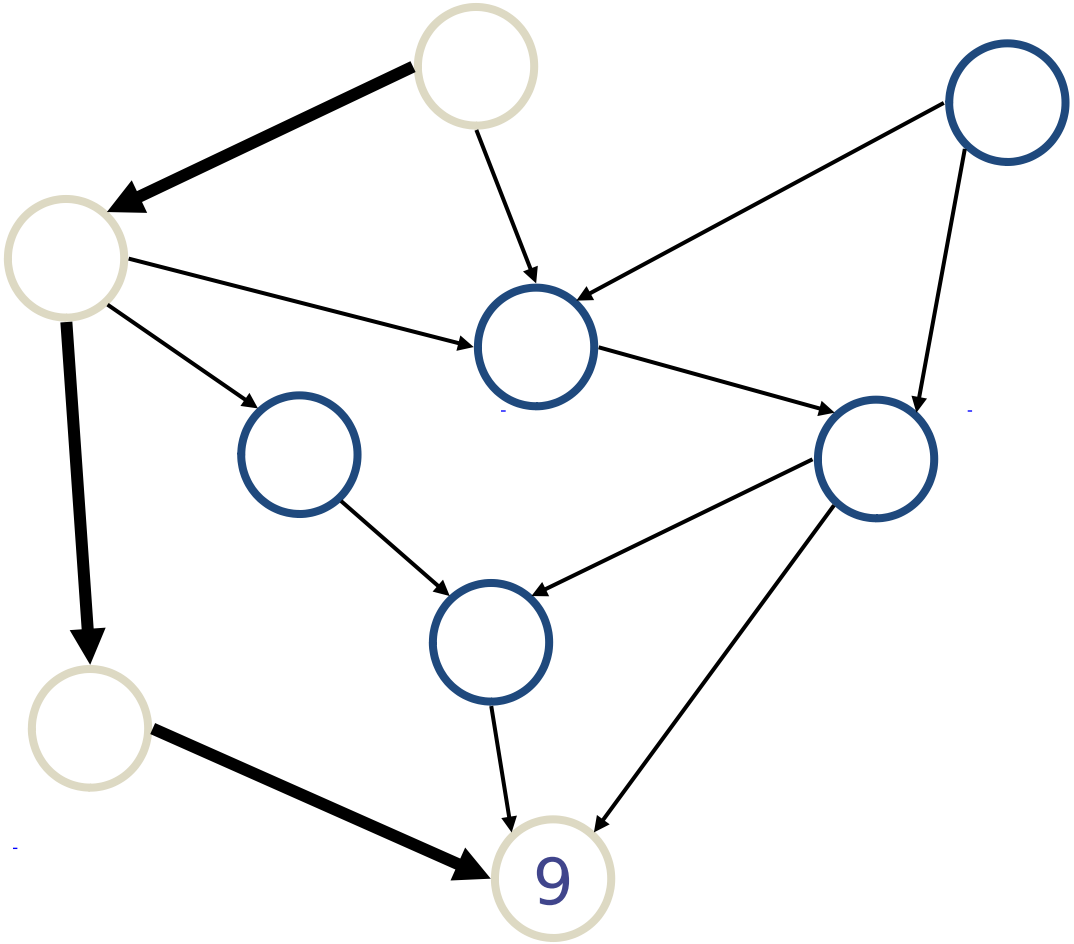
Algoritmo per l'ordinamento topologico

- Algoritmo diverso da quello suggerito dalla dimostrazione del teorema precedente e presente nel libro

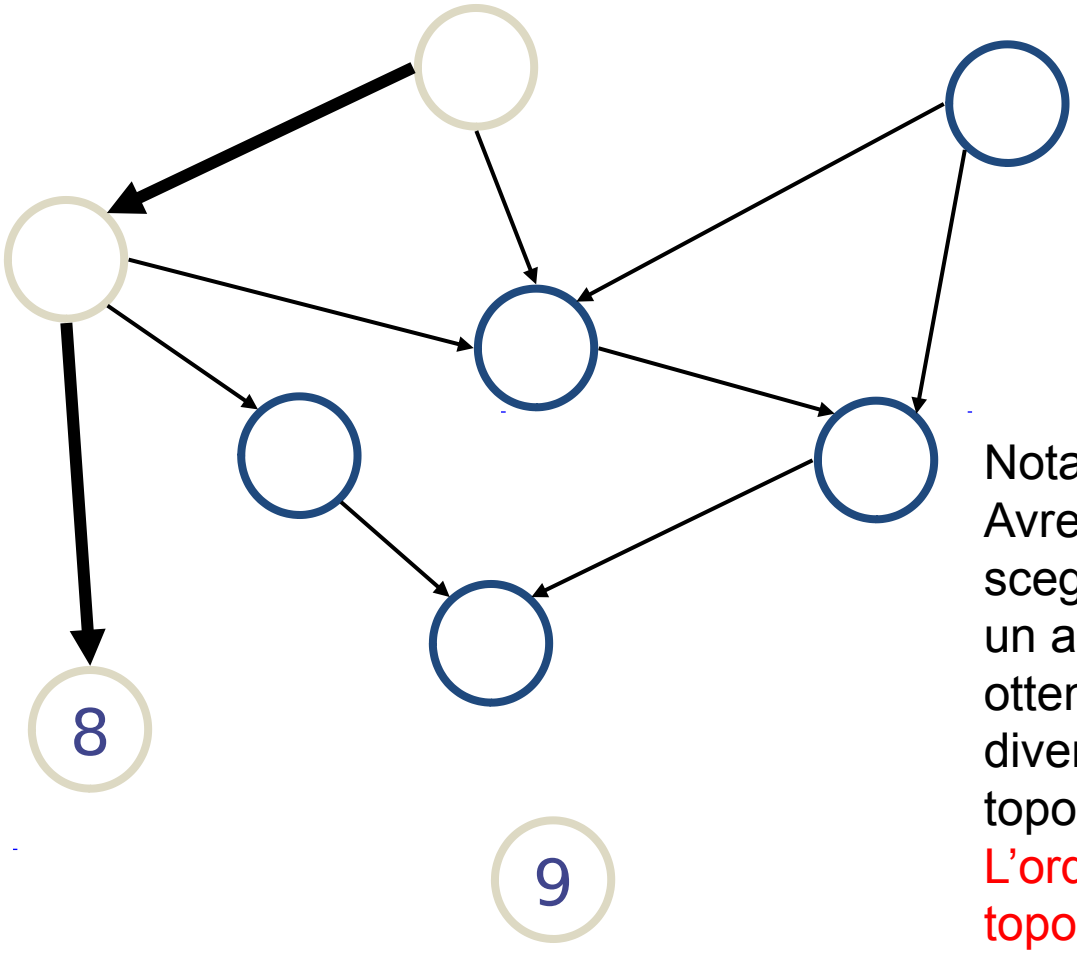
```
Algorithm TopologicalSort(G)
    H = G      // Copia temporanea di G
    n = G.numVertices()
    while H is not empty do
     Sia v un vertice senza archi uscenti
     <etichetta di > v = n
     n = n − 1
     Rimuovi v da H
```

- Complessità: O(n + m)

Esempio 9 vertici

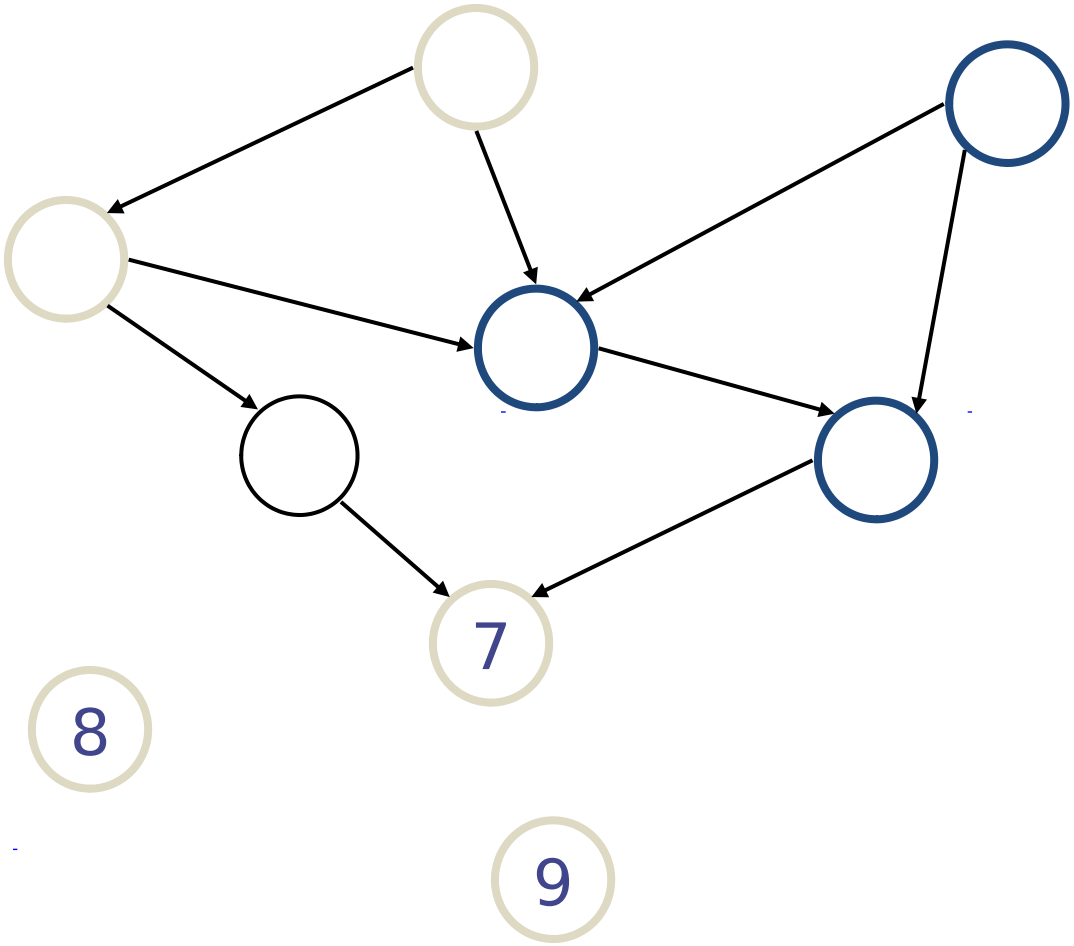


Esempio

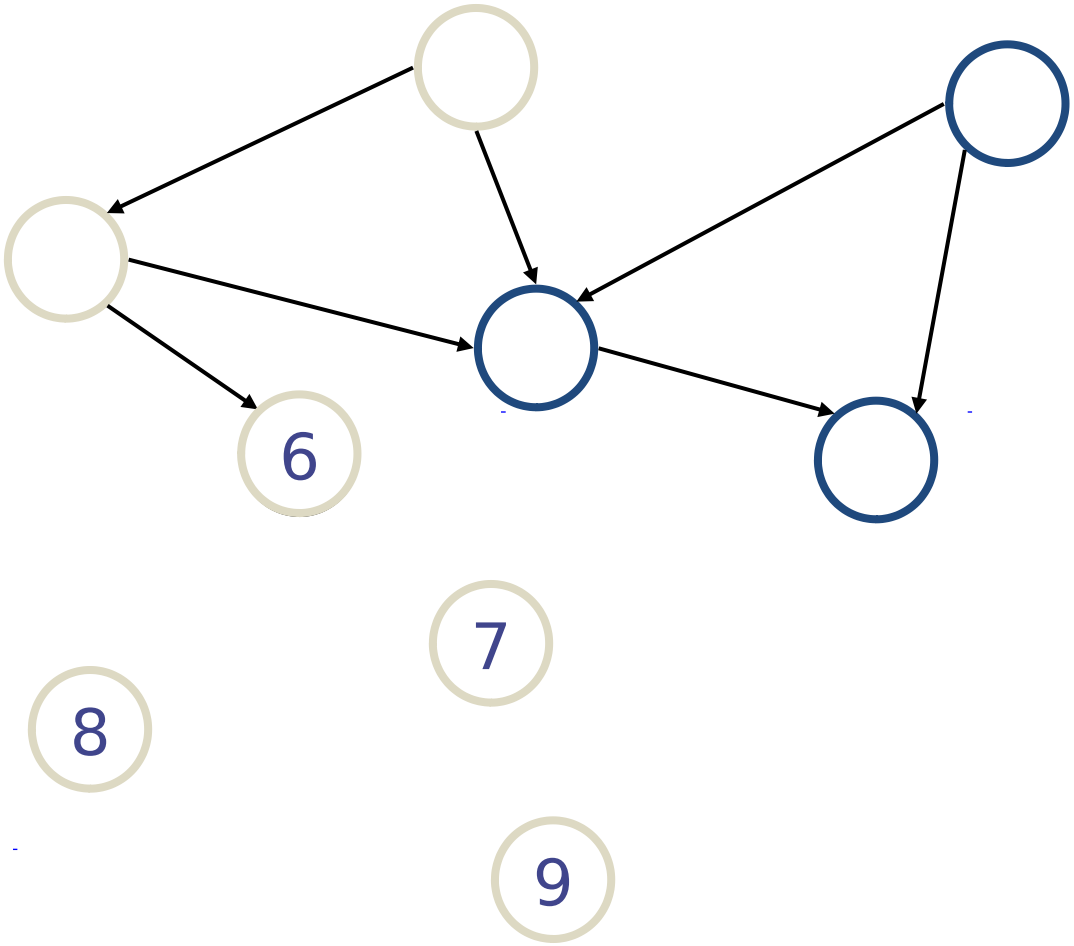


Nota:
Avrei potuto scegliere anche un altro vertice ottenendo un diverso ordine topologico
L'ordine topologico non è unico

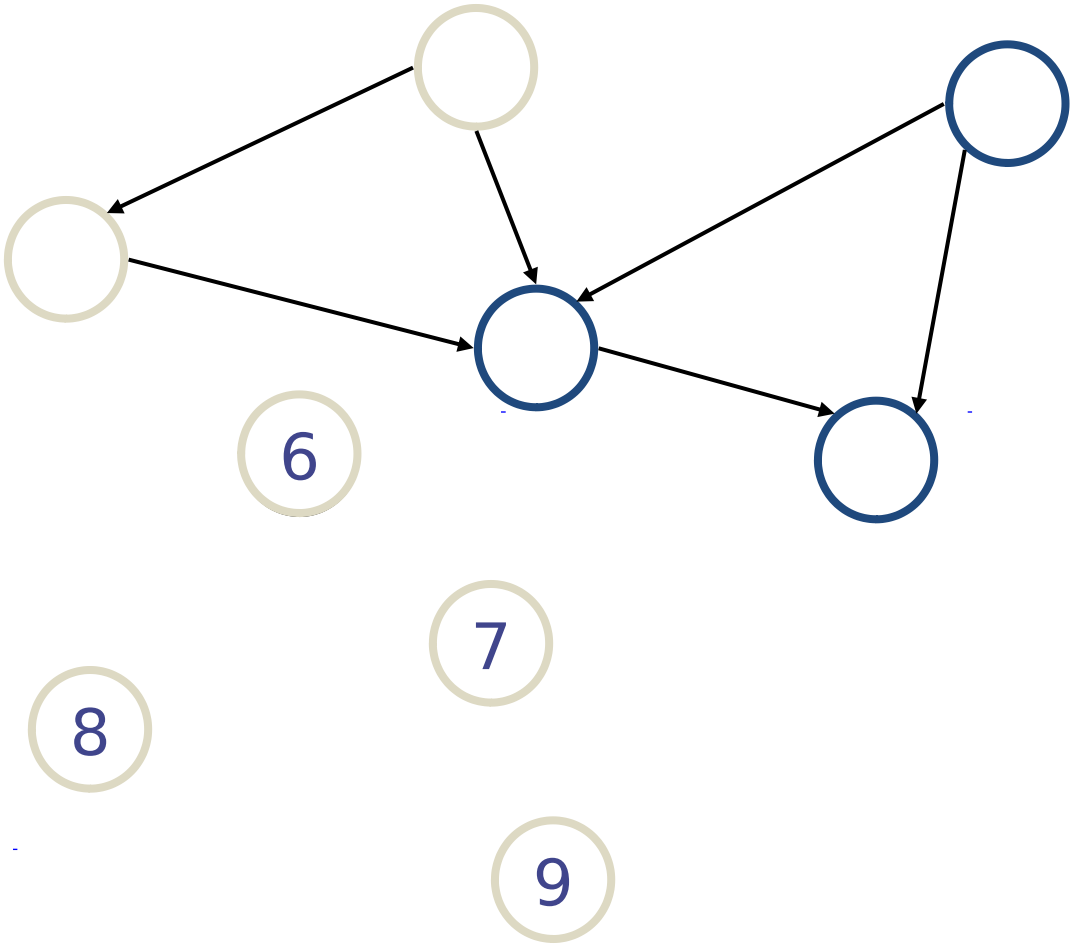
Esempio



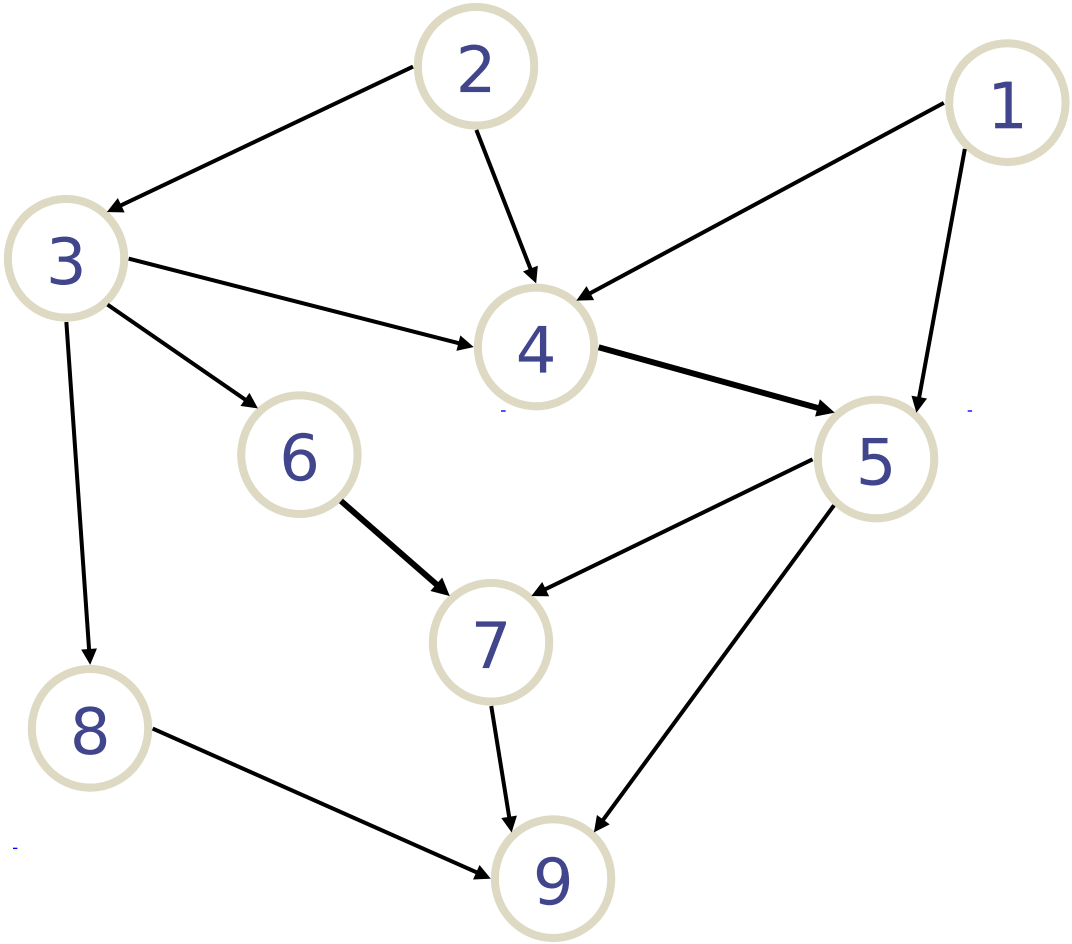
Esempio



Esempio



Esempio



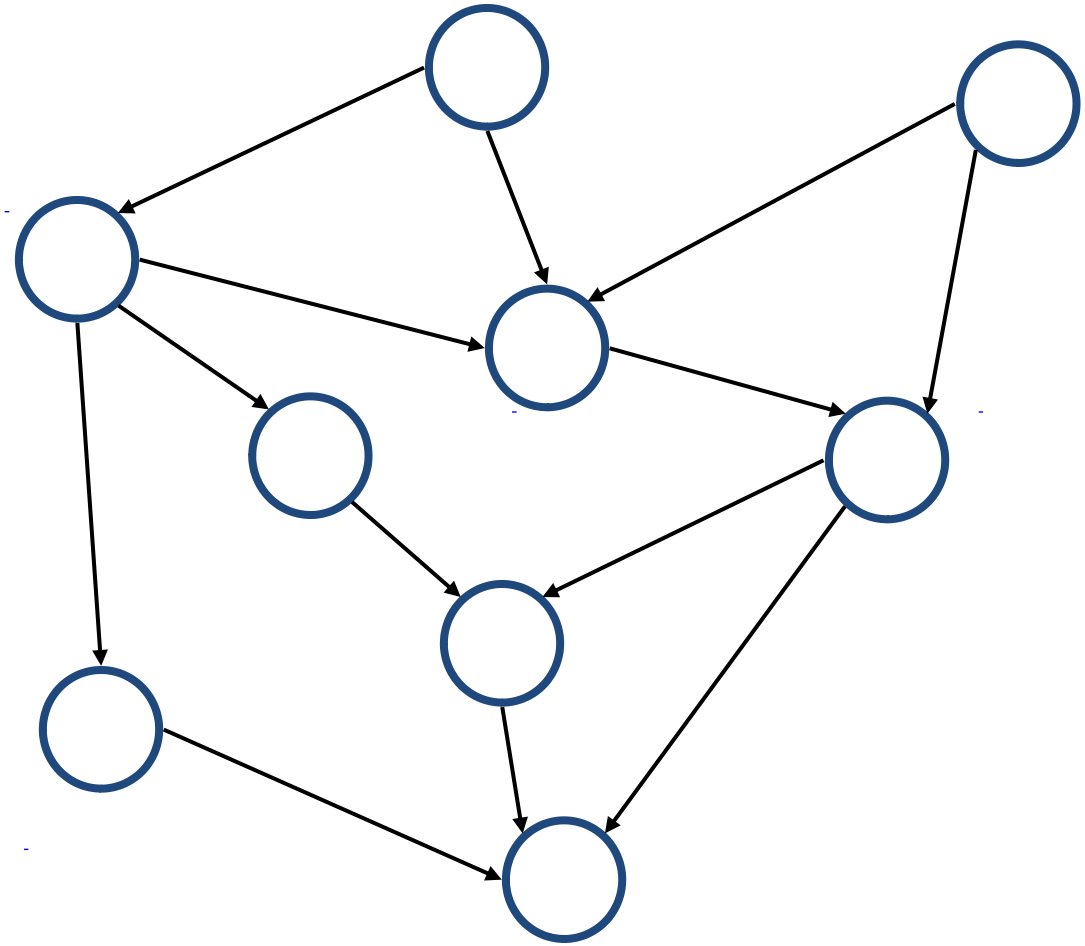
Implementazione con DFS

- Si simula l'algoritmo usando depth-first search
- Costo: O(n+m).

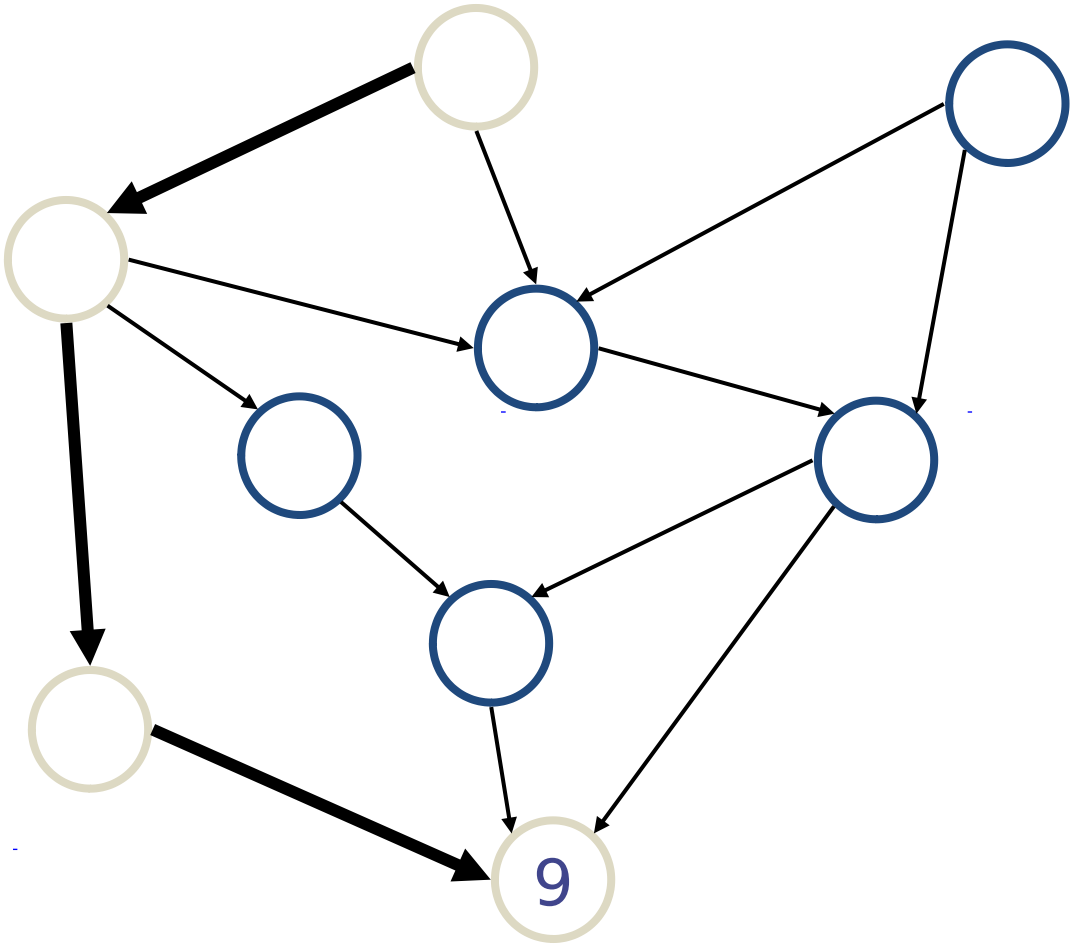
```
Algorithm topologicalDFS(G)
    Input dag G
    Output topological ordering of G
  n = G.numVertices()
  for all u ∈ G.vertices()
        setLabel(u, UNEXPLORED)
  for all v ∈ G.vertices()
        if getLabel(v) = UNEXPLORED
            topologicalDFS(G, v)
```

```
Algorithm topologicalDFS(G, v)
    Input graph G and a start vertex v of G
    Output labeling of the vertices of G in the connected component of v

setLabel(v, EXPLORED)
for all e ∈ G.outEdges(v)
        { outgoing edges }
        w = opposite(v,e)
        if getLabel(w) = UNEXPLORED
            { e is a discovery edge }
            topologicalDFS(G, w)
        else
            { e è o  forward o cross arco }
  Label v with topological number n
  n = n – 1 // Variabile globale
```

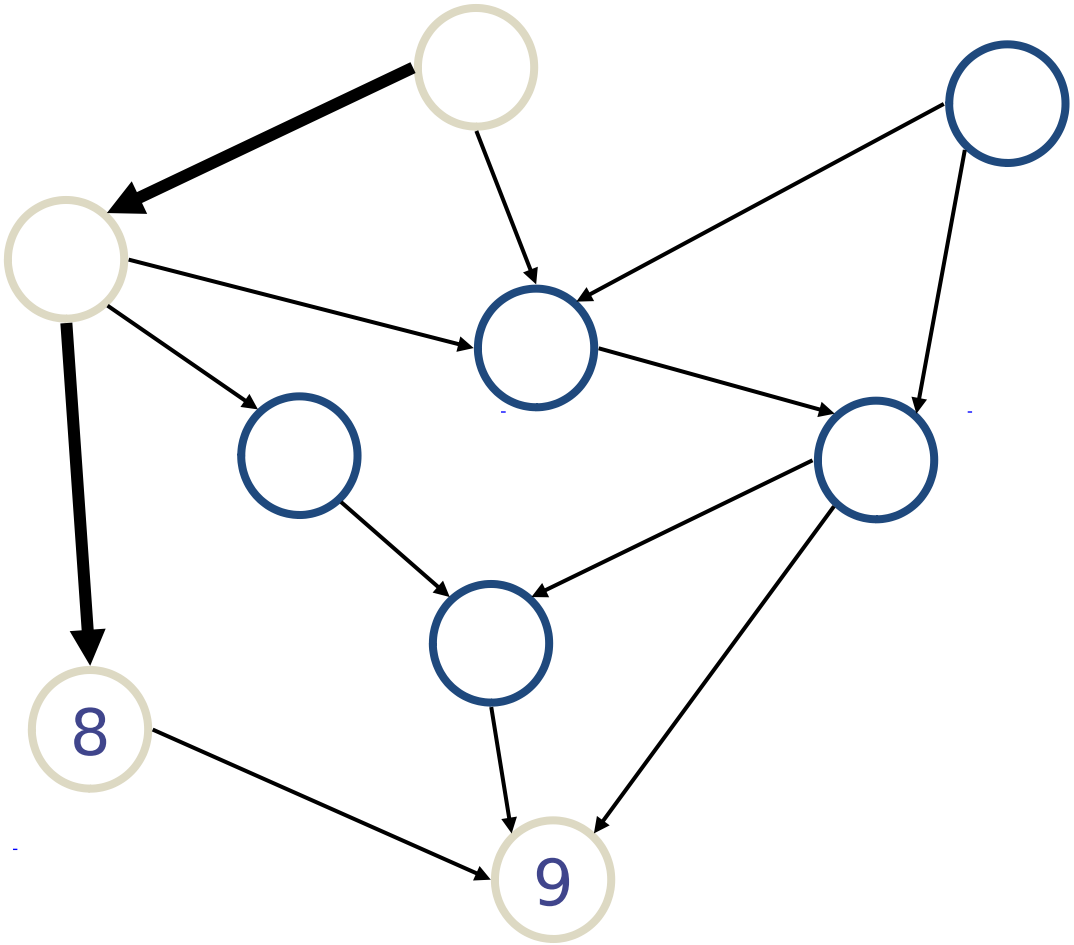
Esempio



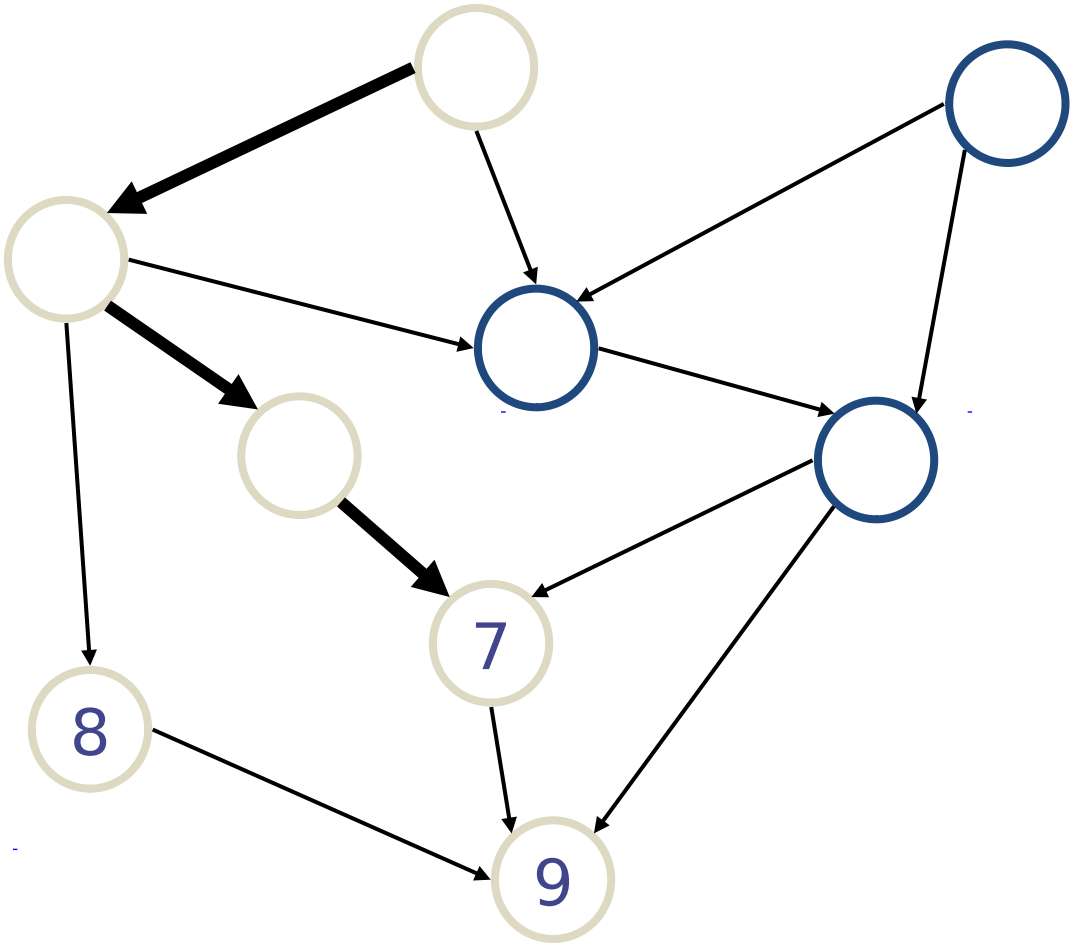
Esempio



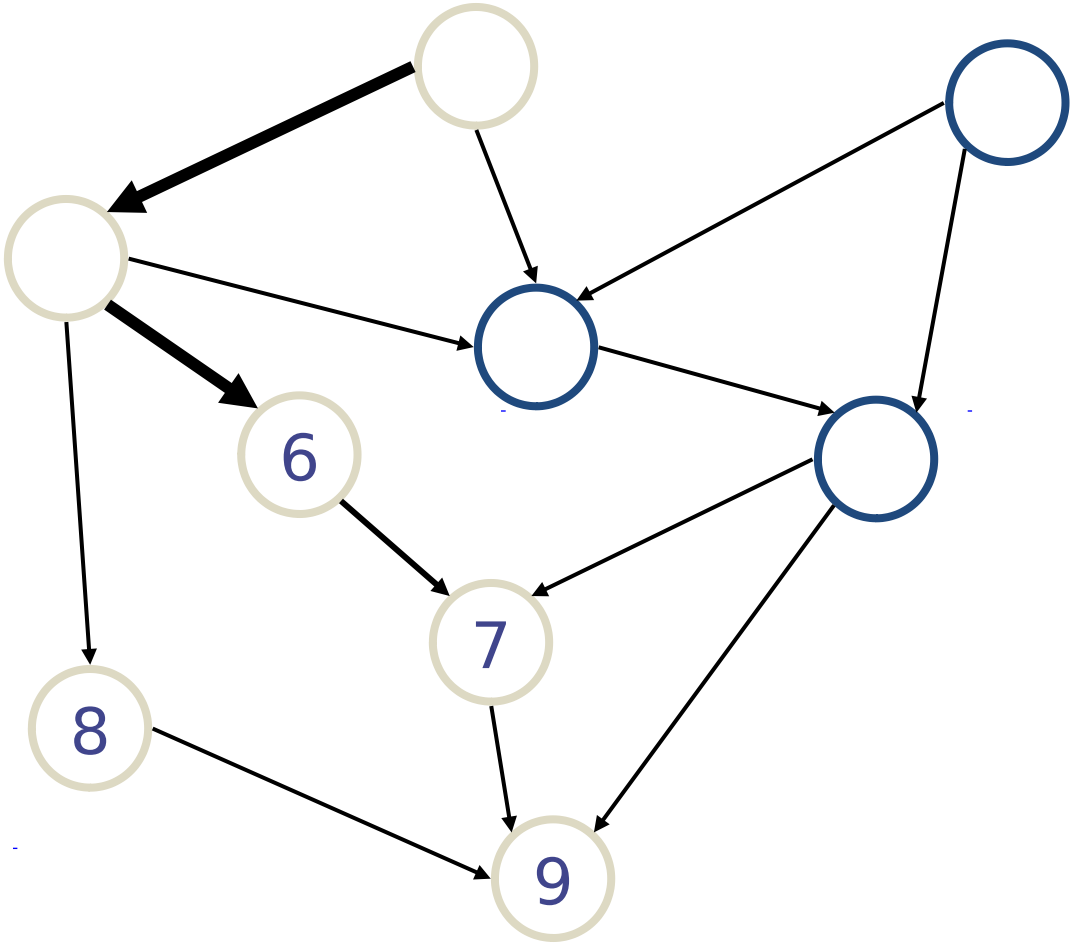
Esempio



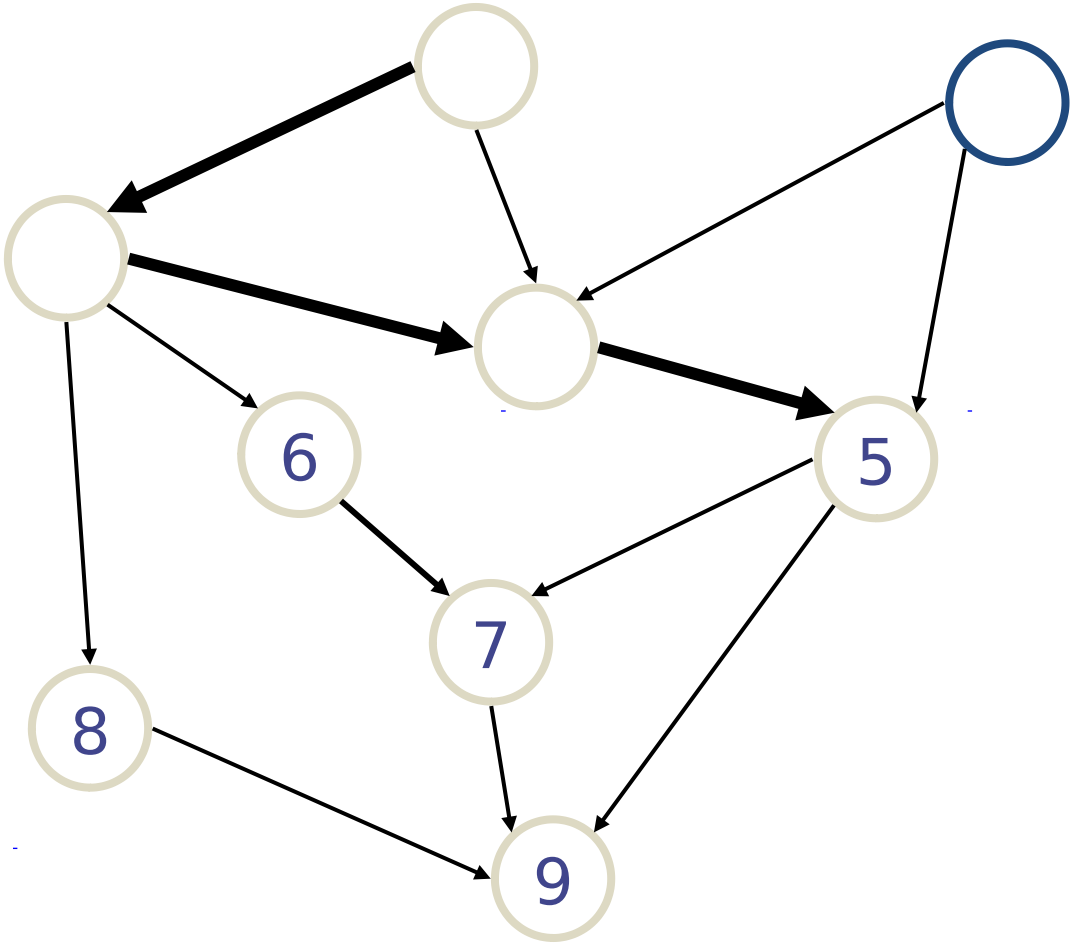
Esempio



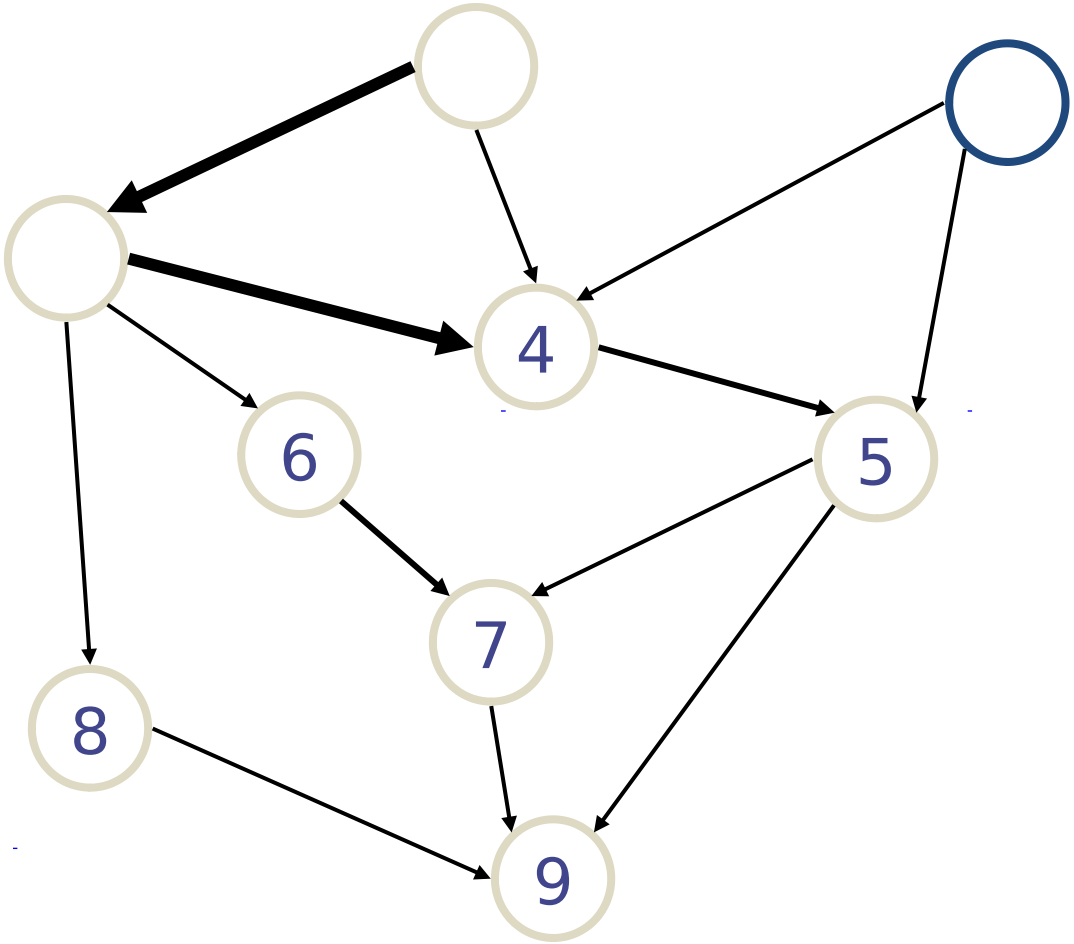
Esempio



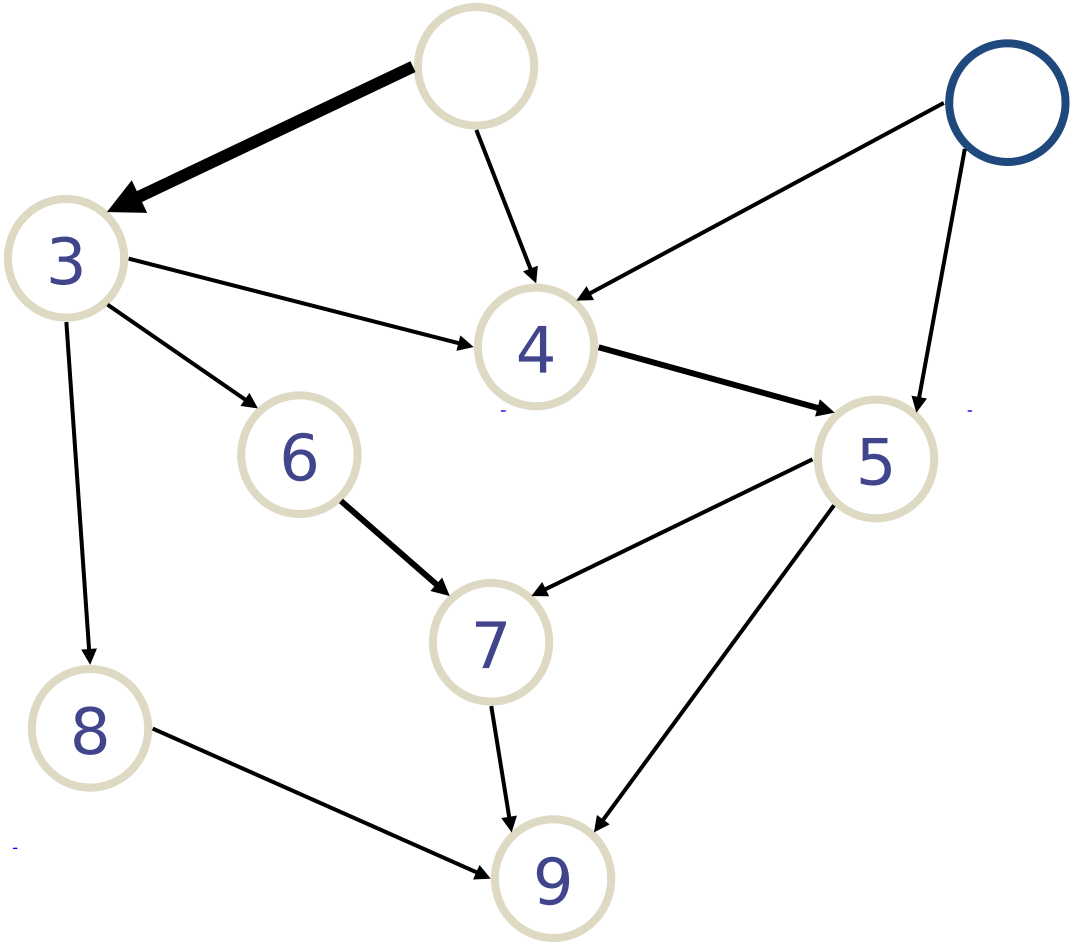
Esempio



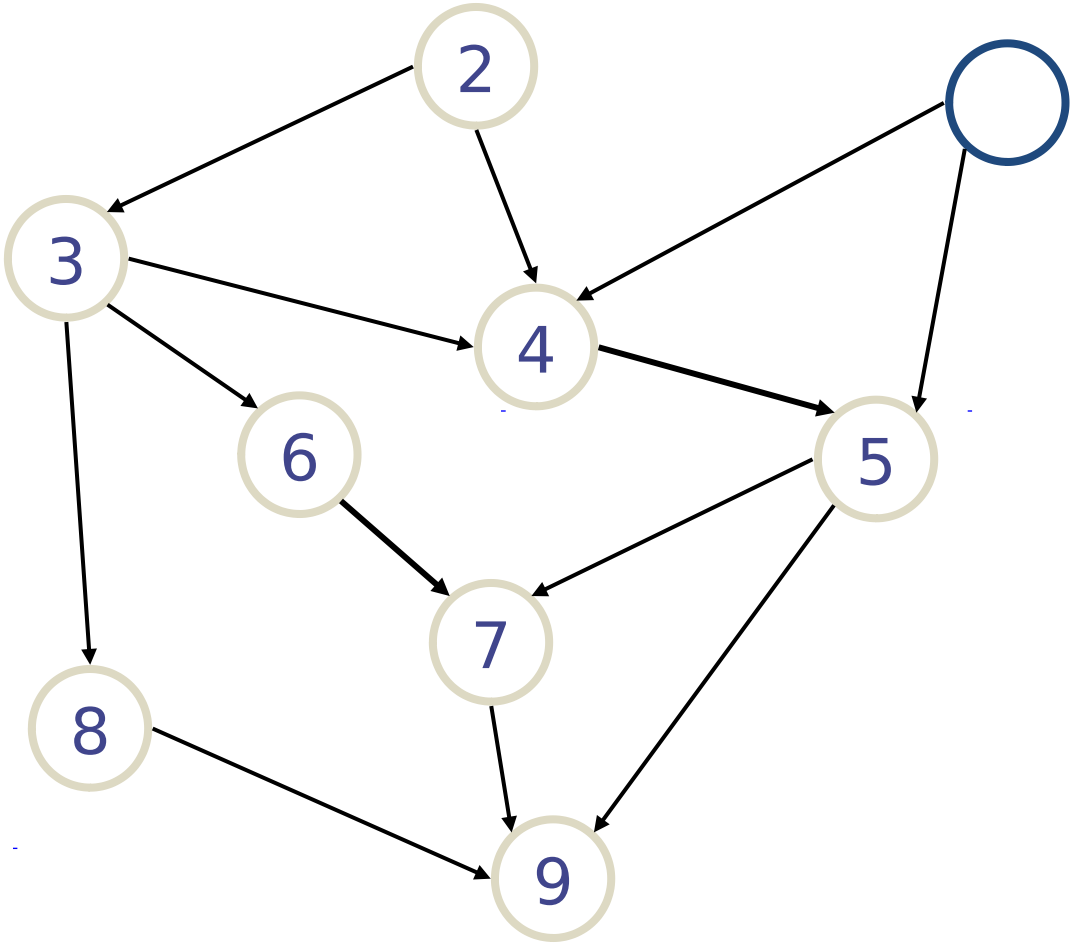
Esempio



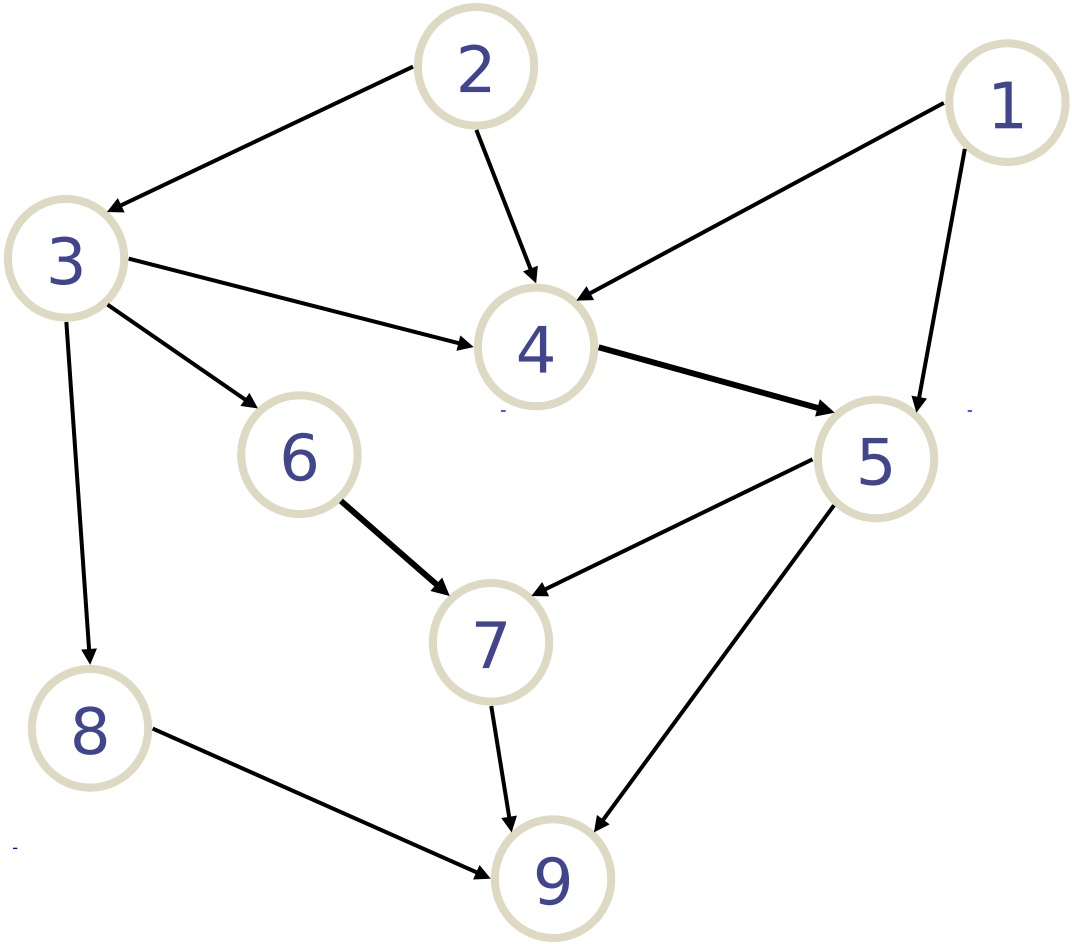
Esempio



Esempio



Esempio



Esempio

Esercizio: trovare l'ordinamento topologico se si inizia la DFS da questo nodo