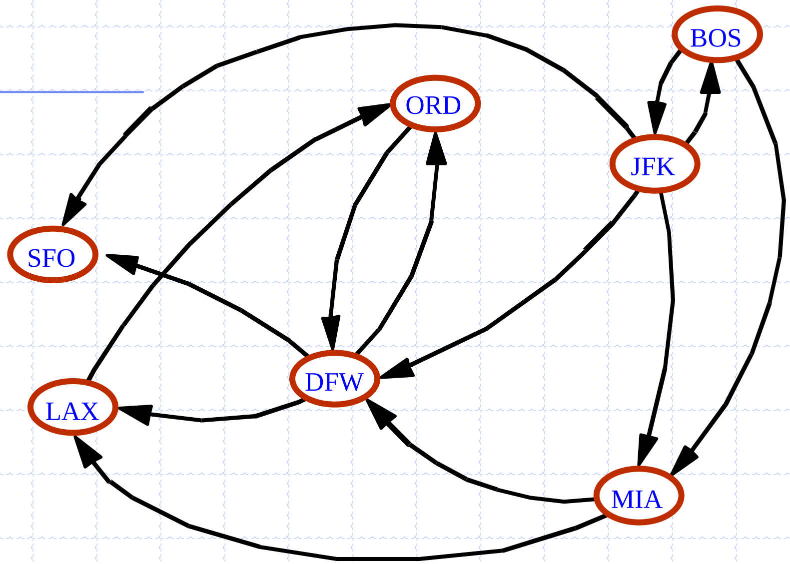
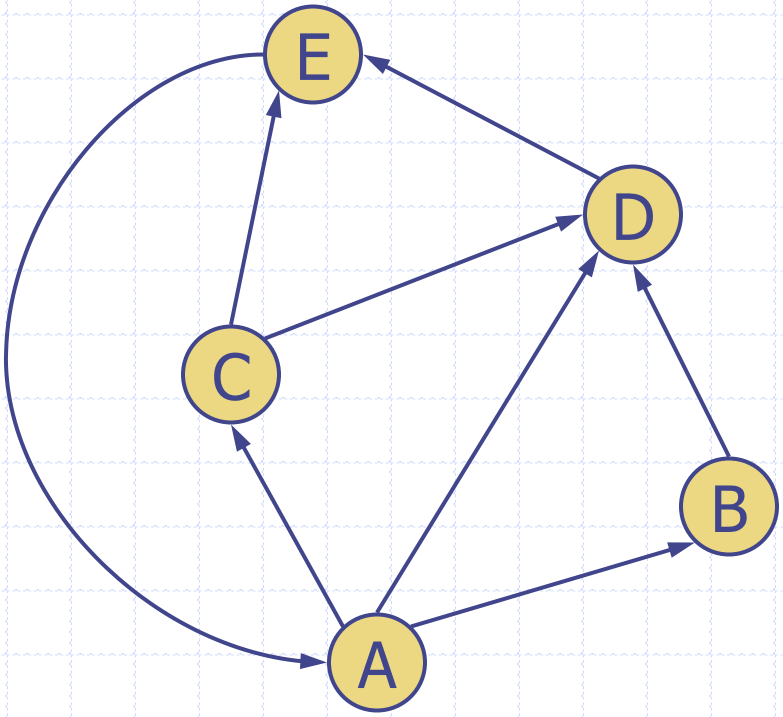
Grafi diretti

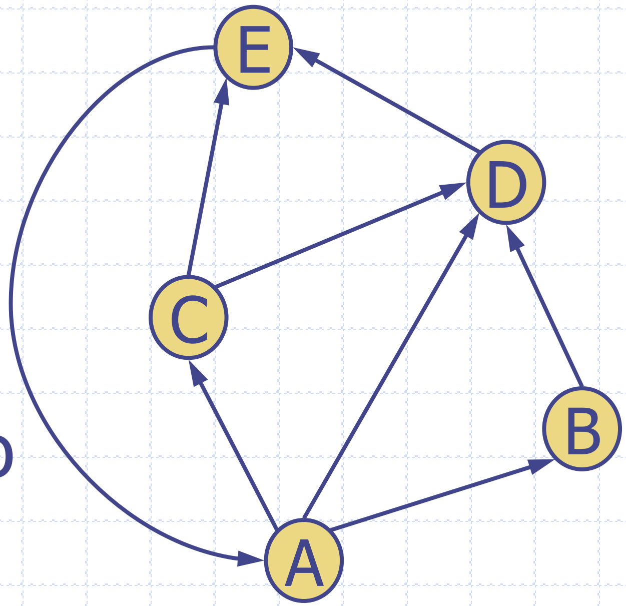


Grafi orientati (digrafi) (§ 13.4)

- Un **grafo orientato** (o **grafo diretto**, o **digrafo**) è un grafo i cui spigoli sono tutti orientati (o diretti)
  - digrafo: contrazione di "directed graph"
- Applicazioni
  - strade a senso unico
  - voli
  - vincoli di precedenza nella pianificazione di attività



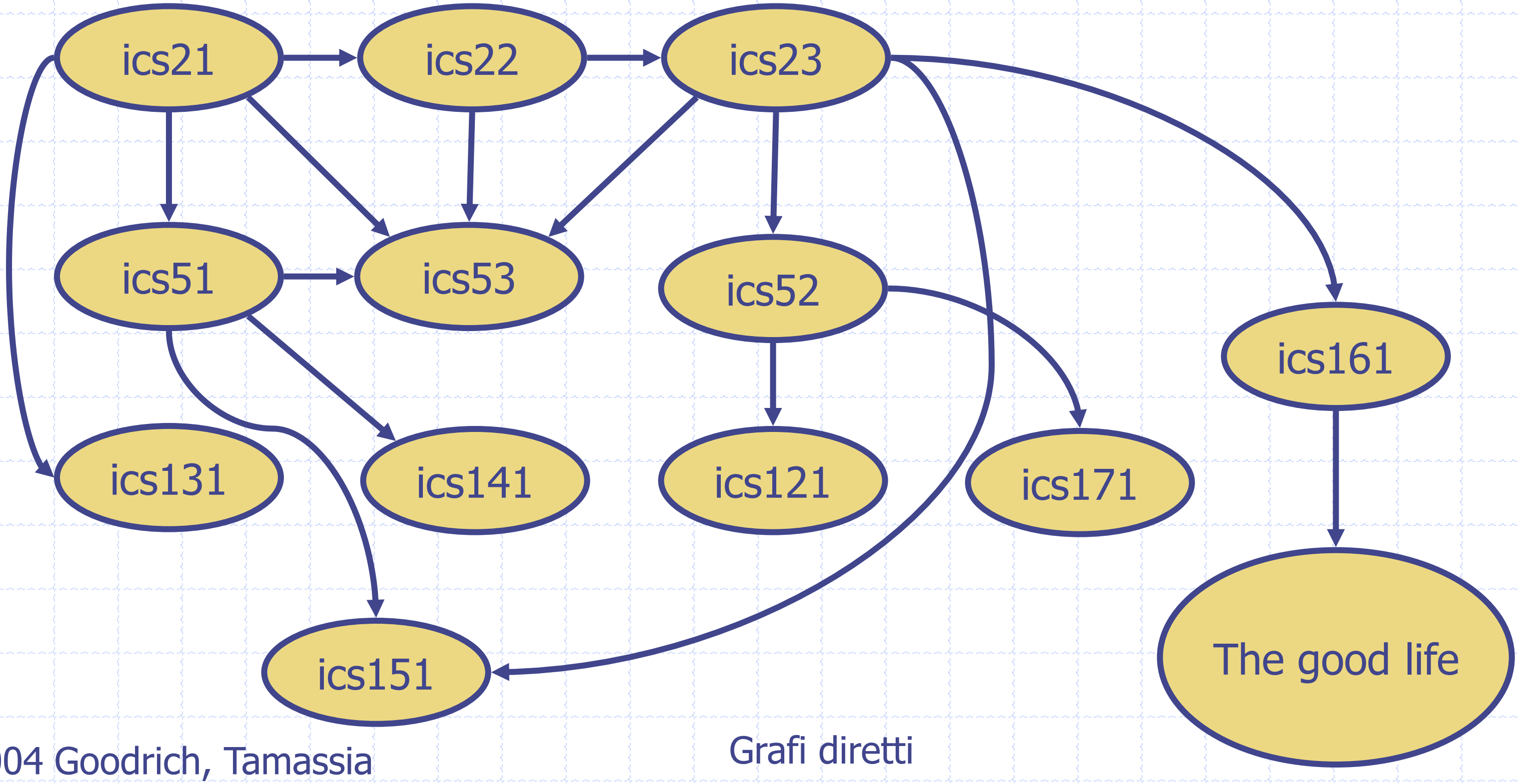
Proprietà dei digrafi



- Grafi tali che ciascuno spigolo "va" in una sola direzione
  - lo spigolo (*a*,*b*) va da *a* a *b*, ma non da *b* ad *a*.
- Se il grafo è semplice, $m \leq n*(n-1)$.
- Per ciascun nodo, se manteniamo gli spigoli entranti ed uscenti in due liste di adiacenza separate, possiamo elencare tali spigoli in tempo ottimale.

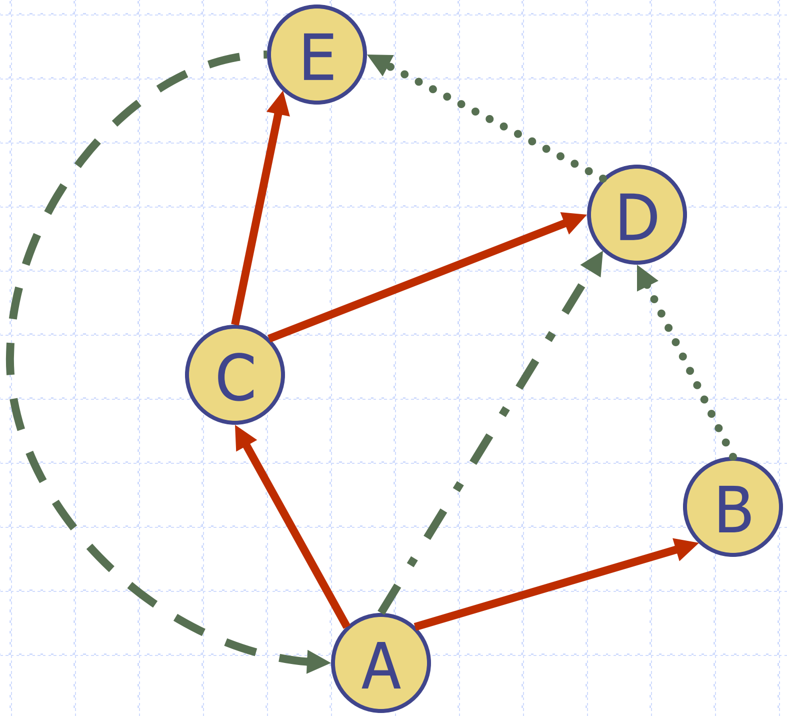
Applicazione dei digrafi

- Pianificazione (scheduling): lo spigolo (*a*,*b*) significa che l'attività (task) *a* deve essere completata prima che *b* inizi



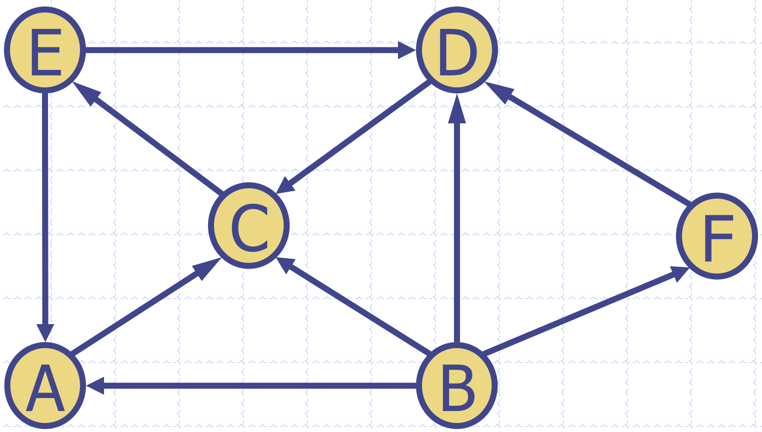
DFS diretta

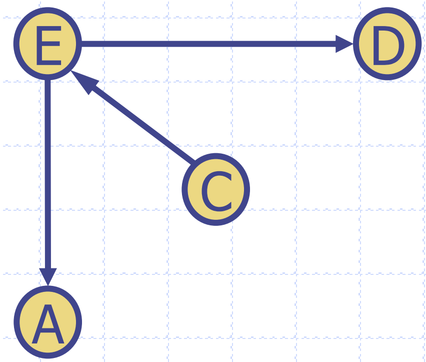
- Possiamo specializzare gli algoritmi di attraversamento (DFS e BFS) ai digrafi stabilendo di percorrere gli spigoli solo nella direzione del loro orientamento
- Nella DFS diretta ci sono quattro tipi di spigoli
  - spigoli discovery
  - spigoli back
  - spigoli forward
  - spigoli cross
- Una DFS diretta che inizia dal vertice s determina i vertici raggiungibili da s

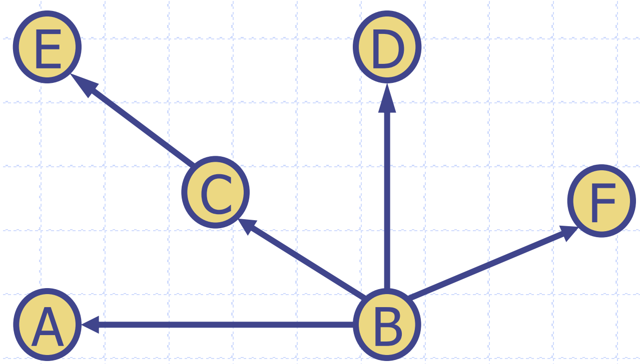


Raggiungibilità

- Albero DFS radicato in v: vertici raggiungibili da v attraverso percorsi orientati

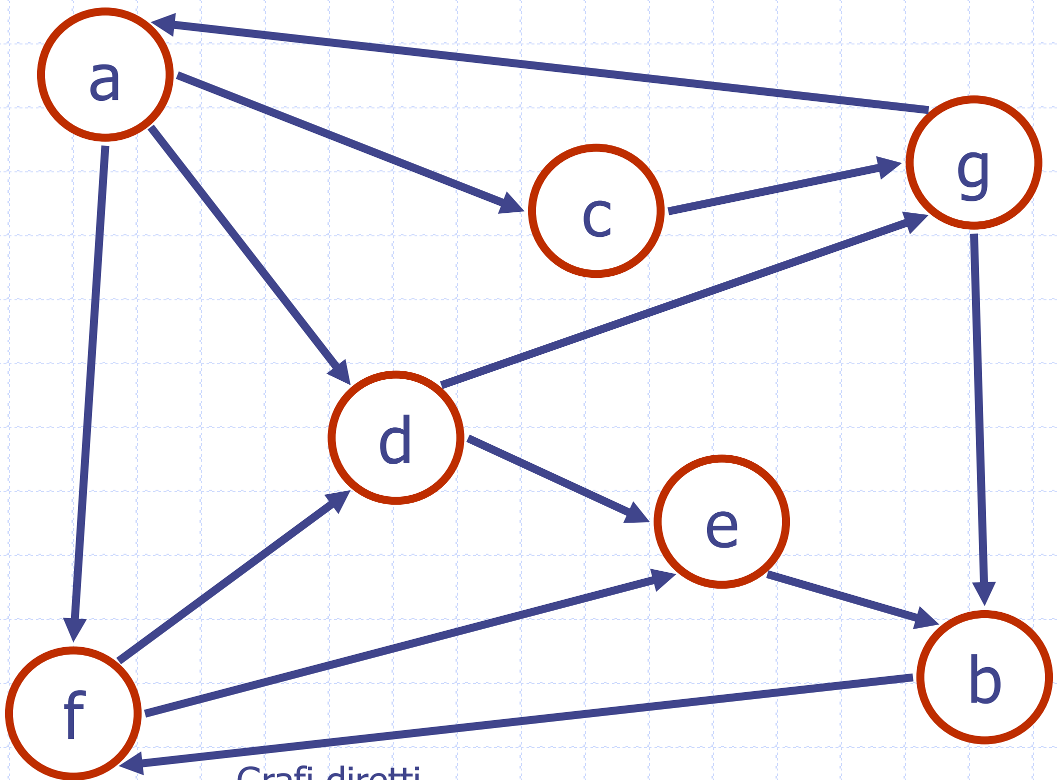


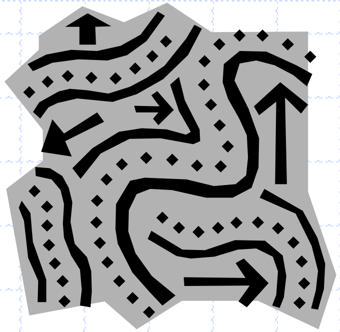




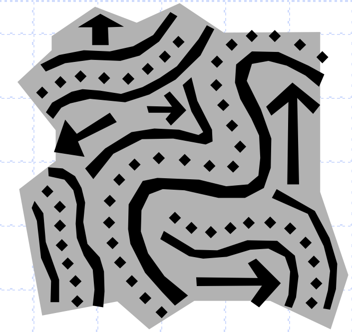
Connettività forte

- Ciascun vertice può raggiungere tutti gli altri vertici attraverso percorsi orientati



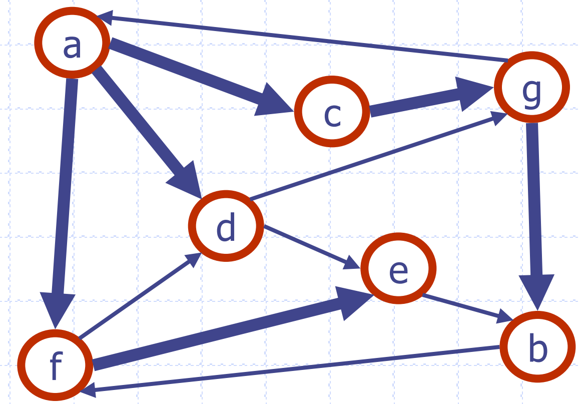


Verifica di connettività forte

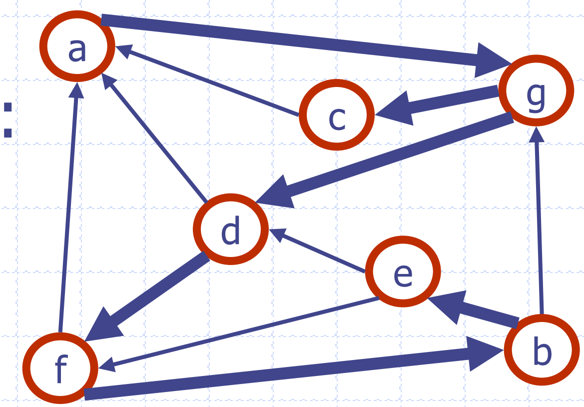


- Scegli un (qualunque) vertice v in G
- Esegui una DFS su G a partire da v
  - se esiste un vertice w non visitato, allora G non è fortemente connesso
- Sia G′ il trasposto di G (spigoli tutti "rovesciati")
  - G è fortemente connesso se e solo G′ è fortemente connesso
- Esegui una DFS su G′ a partire da v
  - se esiste un vertice w non visitato, allora G non è fortemente connesso
  - altrimenti, G è fortemente connesso
- Tempo di esecuzione: O(n+m)

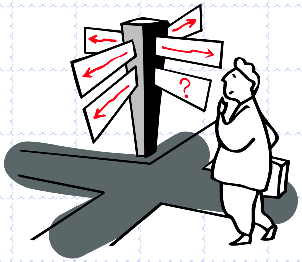
G:



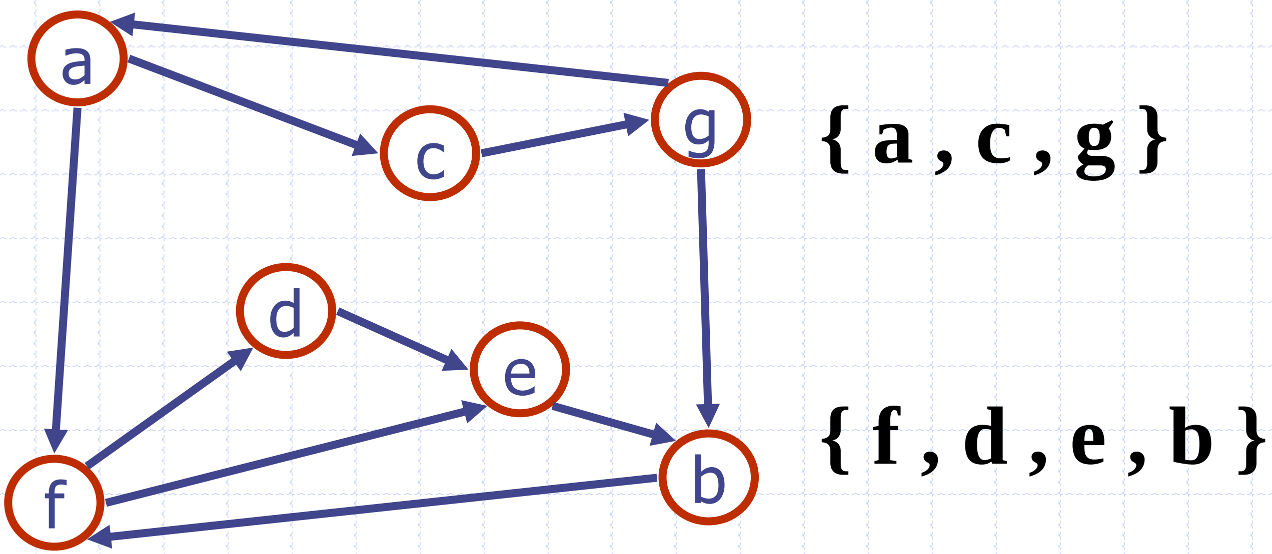
G′:



Componenti fortemente connesse

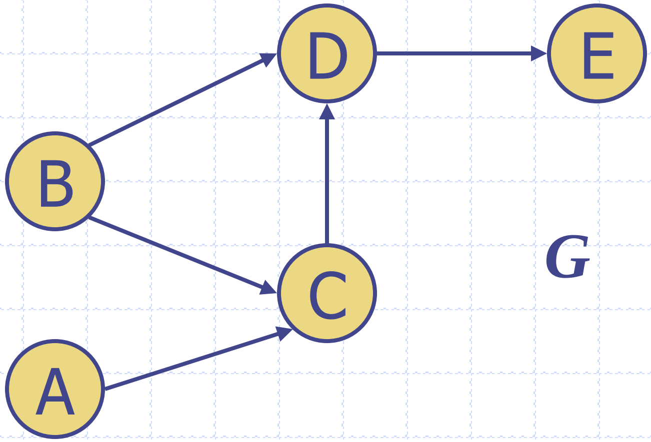


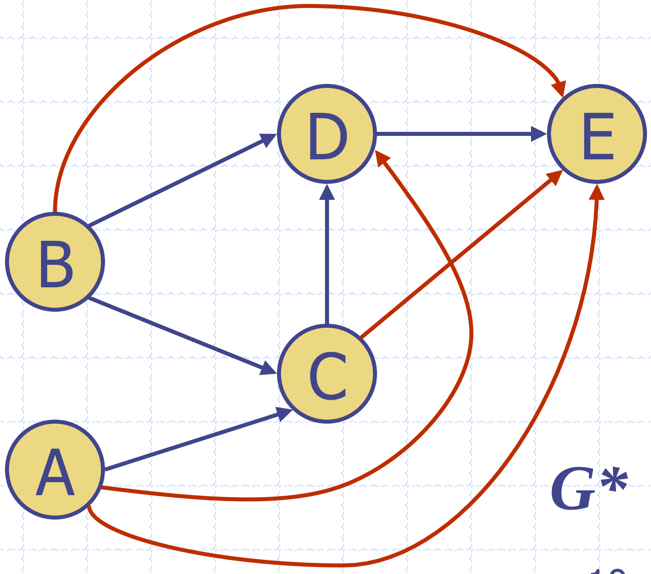
- Sottografi massimali in cui per ciascun vertice esiste un percorso orientato che lo collega a tutti gli altri vertici del sottografo
- Possono essere calcolate in tempo O(n+m) usando due DFS "aumentate"



Chiusura transitiva

- Dato un digrafo G, la chiusura transitiva di G è un digrafo G^* tale che
  - G^* ha gli stessi vertici di G
  - se G ha un percorso orientato da u a v ($u \neq v$), allora G^* ha uno spigolo orientato da u a v
- La chiusura transitiva fornisce informazioni esplicite di raggiungibilità in un digrafo

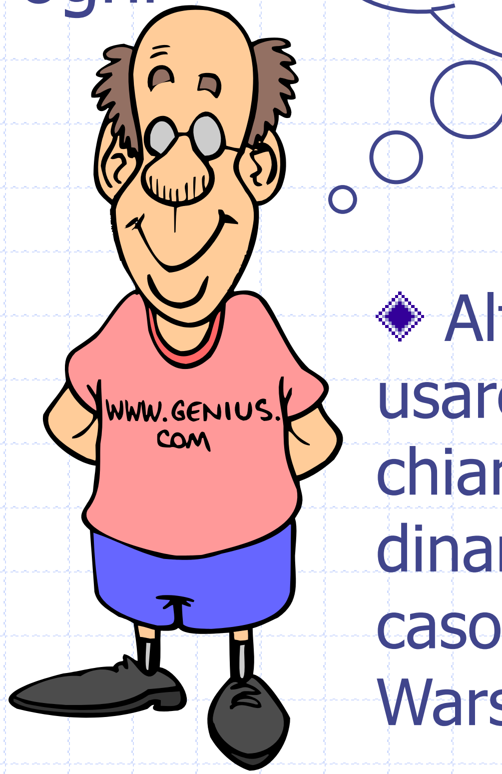




Calcolo della chiusura transitiva

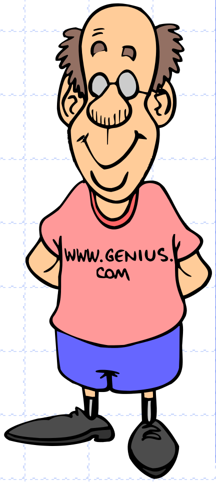
- Possiamo eseguire n DFS, iniziando da ogni vertice
  - O(n(n+m))

Se esiste un modo per andare da A a B e da B a C, allora ne esiste uno per andare da A a C.

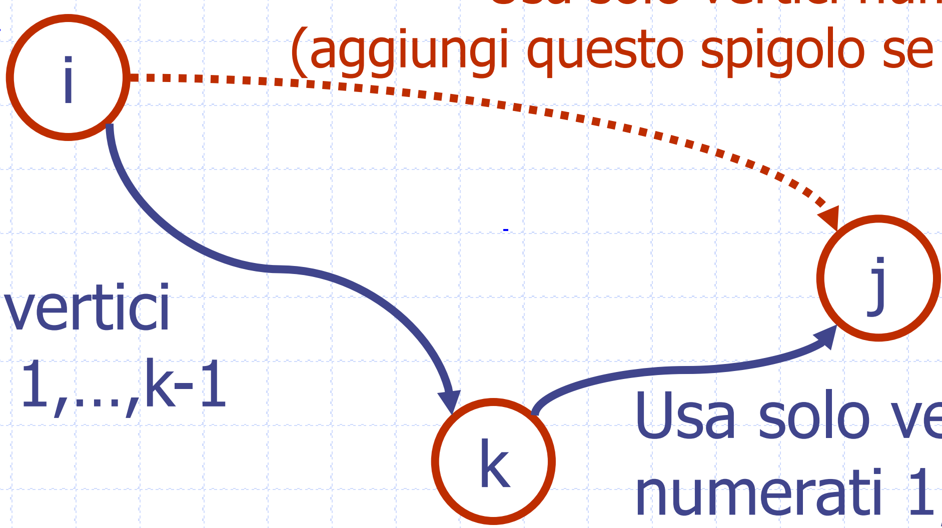


- Alternativamente, si può usare una tecnica algoritmica chiamata “programmazione dinamica,” come fatto nel caso dell’algoritmo di Floyd-Warshall

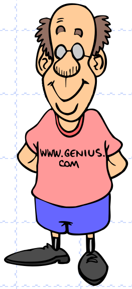
Chiusura transitiva con Floyd-Warshall



- Idea #1: Numera i vertici con 1, 2, …, n
- Idea #2: Considera percorsi che usano come vertici intermedi solo vertici numerati 1, 2, …, k



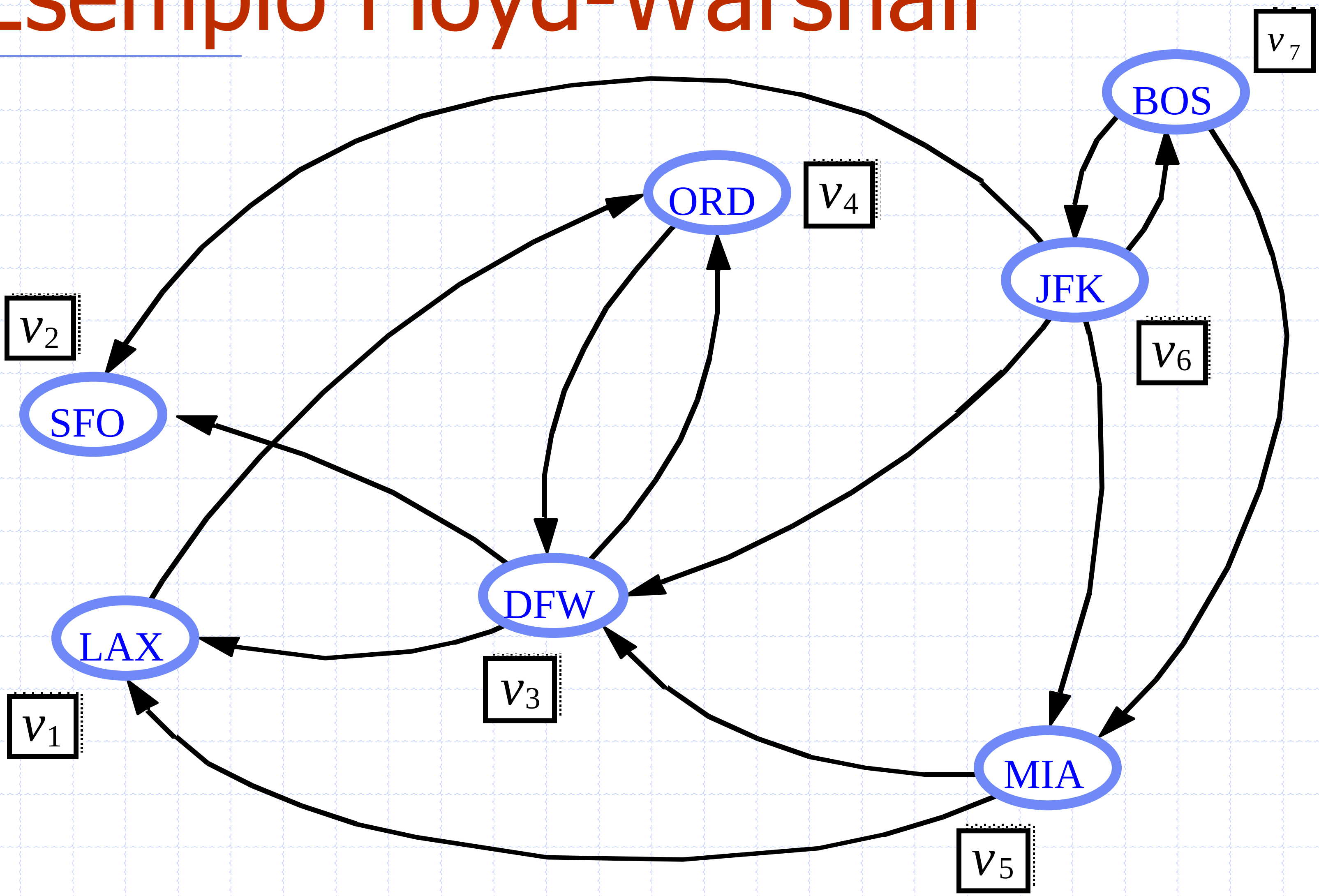
Algoritmo di Floyd-Warshall



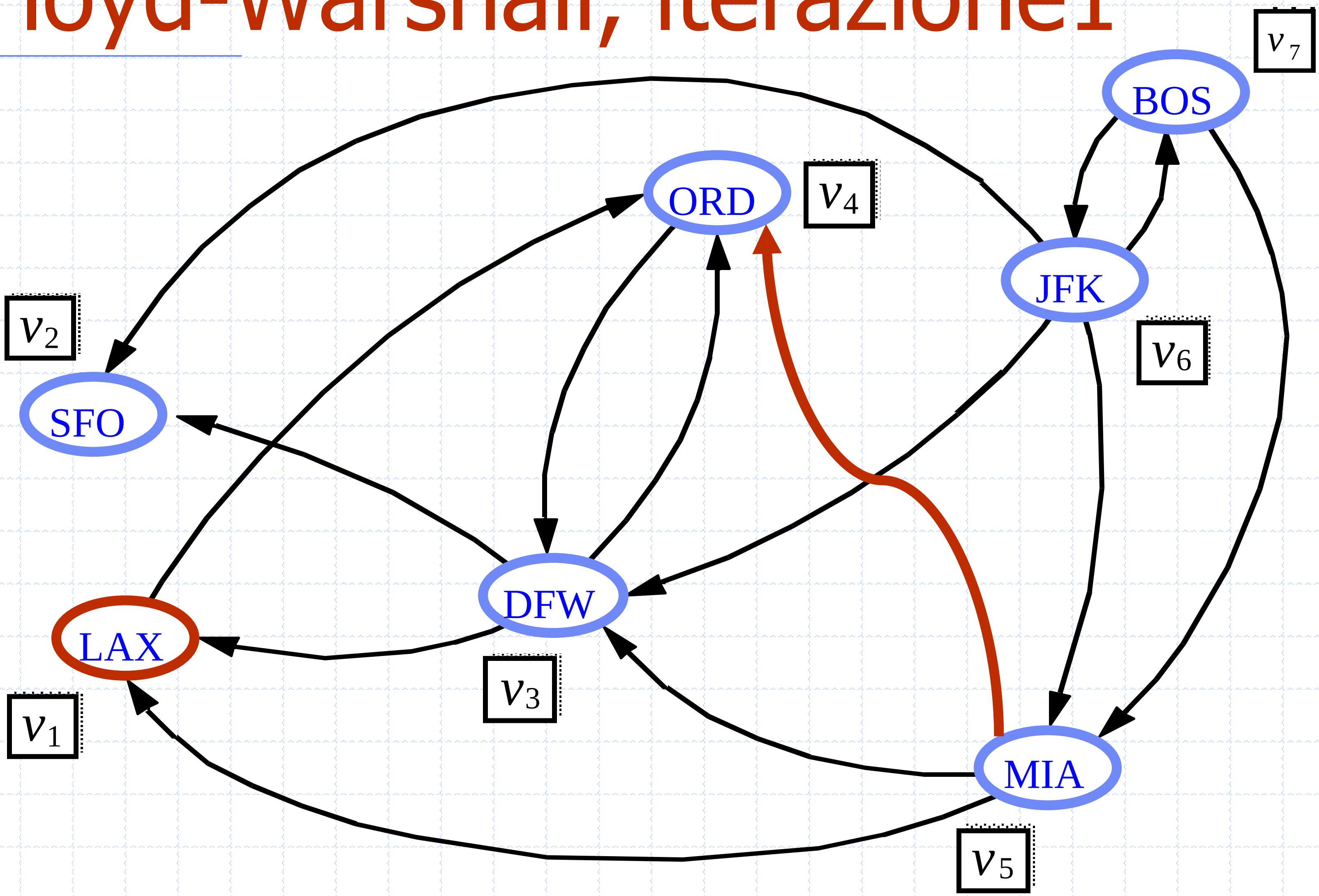
- denomina vertici di G come $v_1, \ldots, v_n$ e calcola una serie di digrafi $G_0, \ldots, G_n$
  - $G_0 = G$
  - G_k ha uno spigolo orientato (v_i, v_j) se G ha un percorso orientato da v_i a v_j con vertici intermedi nell'insieme $\{v_1, \ldots, v_k\}$
- Per definizione $G_n = G^*$
- Nella fase k, viene calcolato il digrafo G_k a partire da G_{k-1}
- Tempo di esecuzione $O(n^3)$ se areAdjacent viene eseguito in $O(1)$ (ad esempio, usando una matrice di adiacenza)

Algorithm *FloydWarshall*(G)
  Input digrafo G
  Output la chiusura transitiva G^* di G
  $i \leftarrow 1$
  for all $v \in G.vertices()$
    denota v come v_i
    $i \leftarrow i + 1$
  $G_0 \leftarrow G$
  for $k \leftarrow 1$ **to** n **do**
    $G_k \leftarrow G_{k-1}$
    for $i \leftarrow 1$ **to** n $(i \neq k)$ **do**
      for $j \leftarrow 1$ **to** n $(j \neq i, k)$ **do**
        if $G_{k-1}.areAdjacent(v_i, v_k) \wedge$
            $G_{k-1}.areAdjacent(v_k, v_j)$
          if $\neg G_k.areAdjacent(v_i, v_j)$
            $G_k.insertDirectedEdge(v_i, v_j, k)$
    return G_n

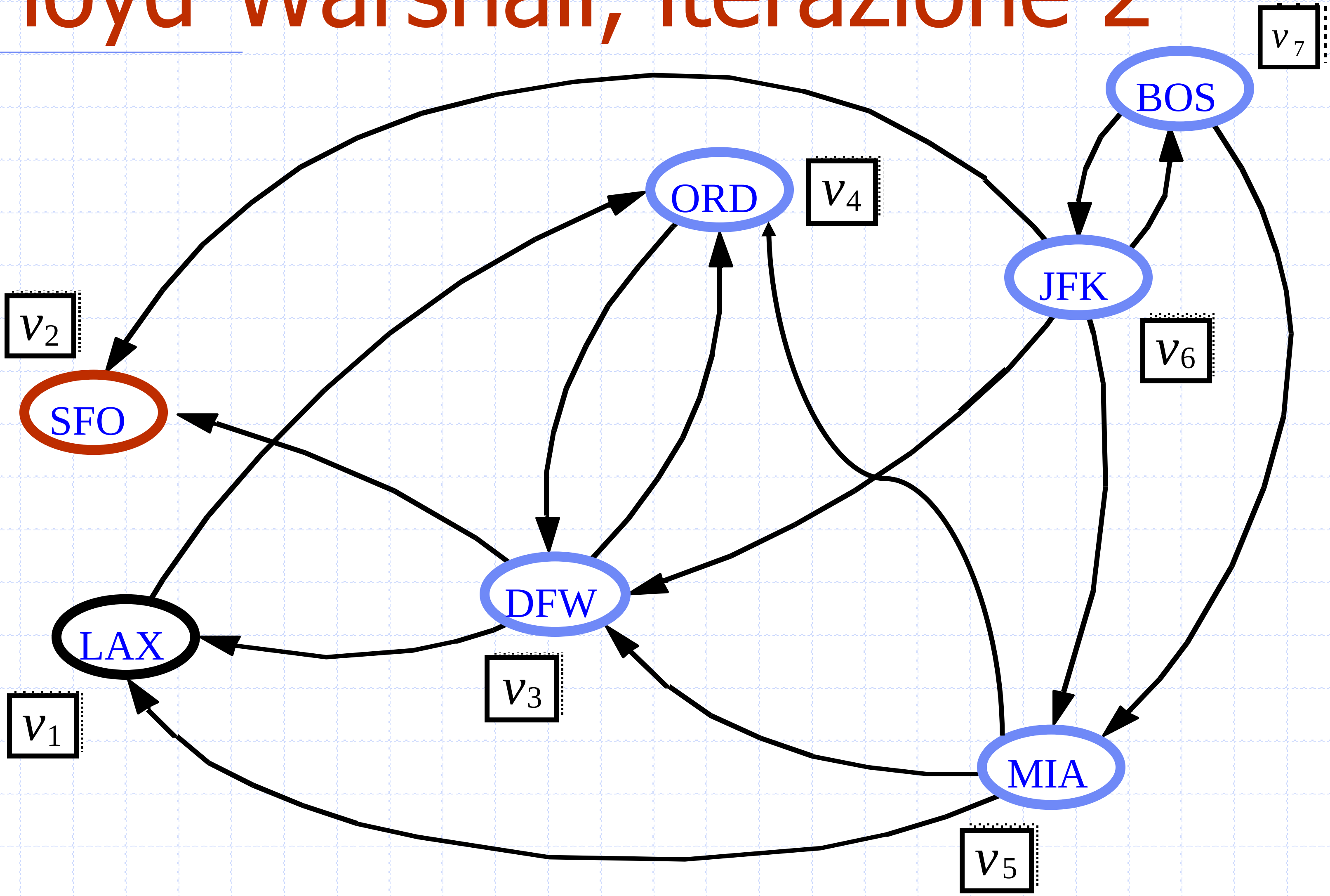
Esempio Floyd-Warshall



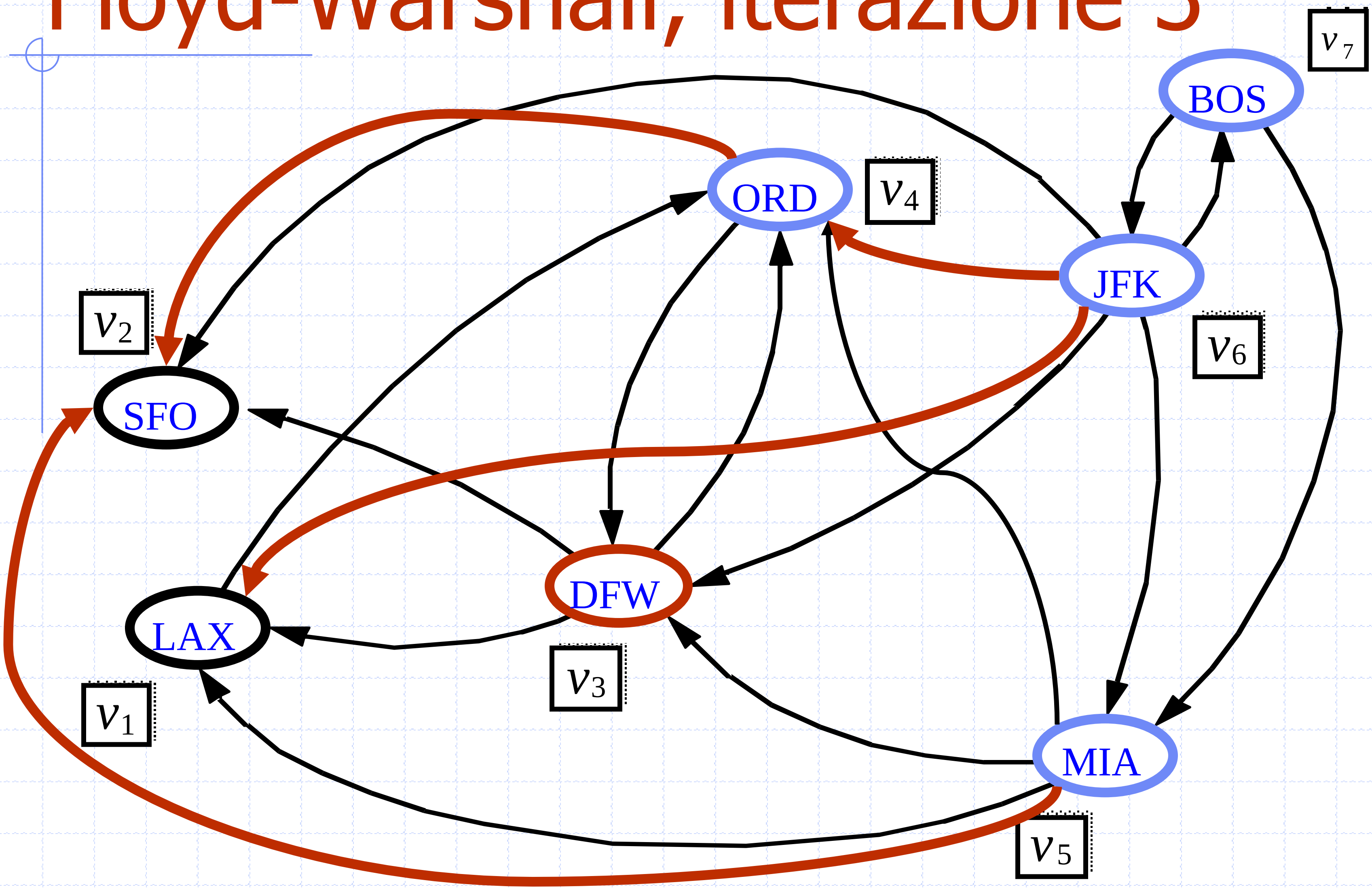
Floyd-Warshall, iterazione1



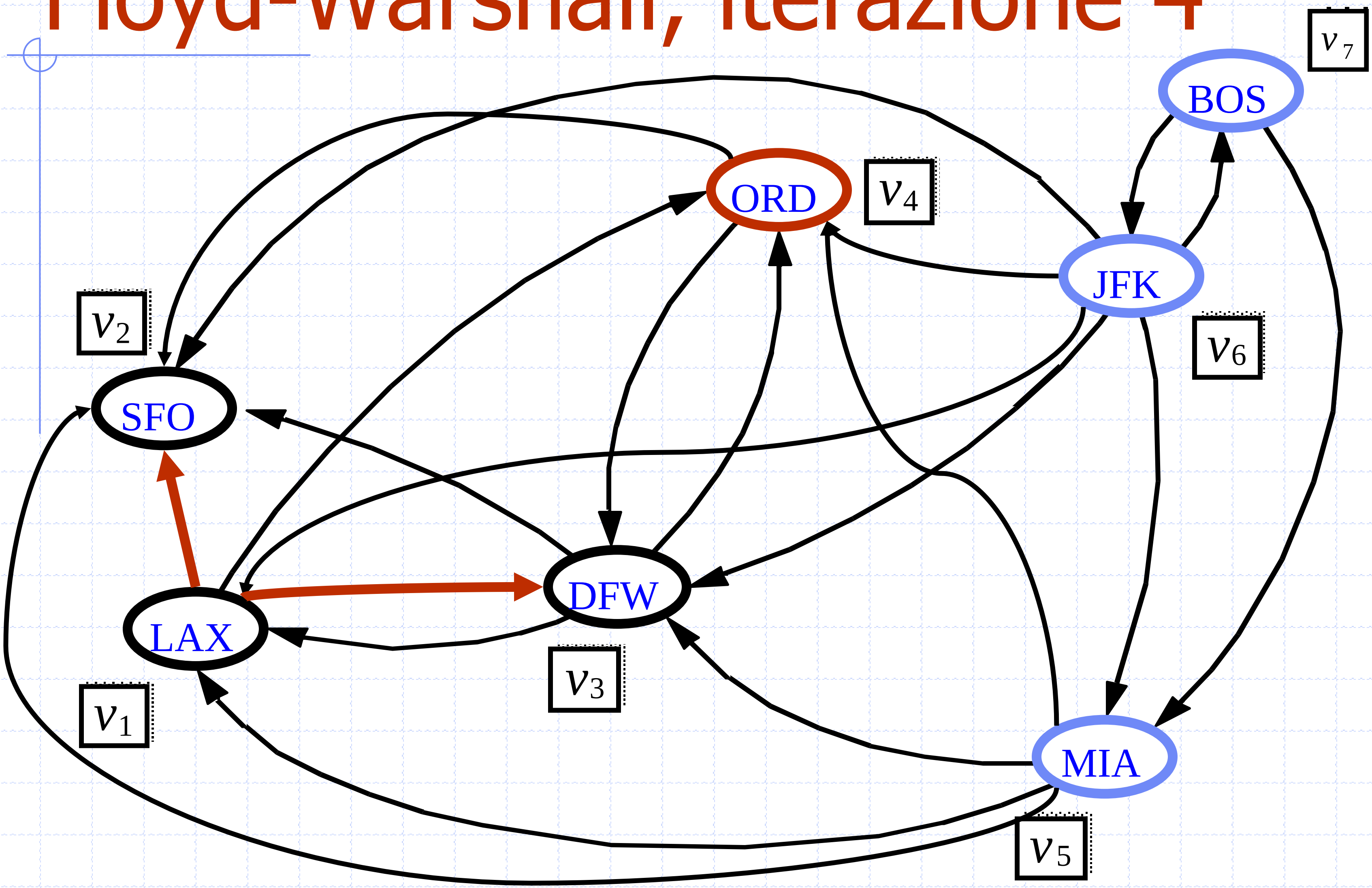
Floyd-Warshall, iterazione 2



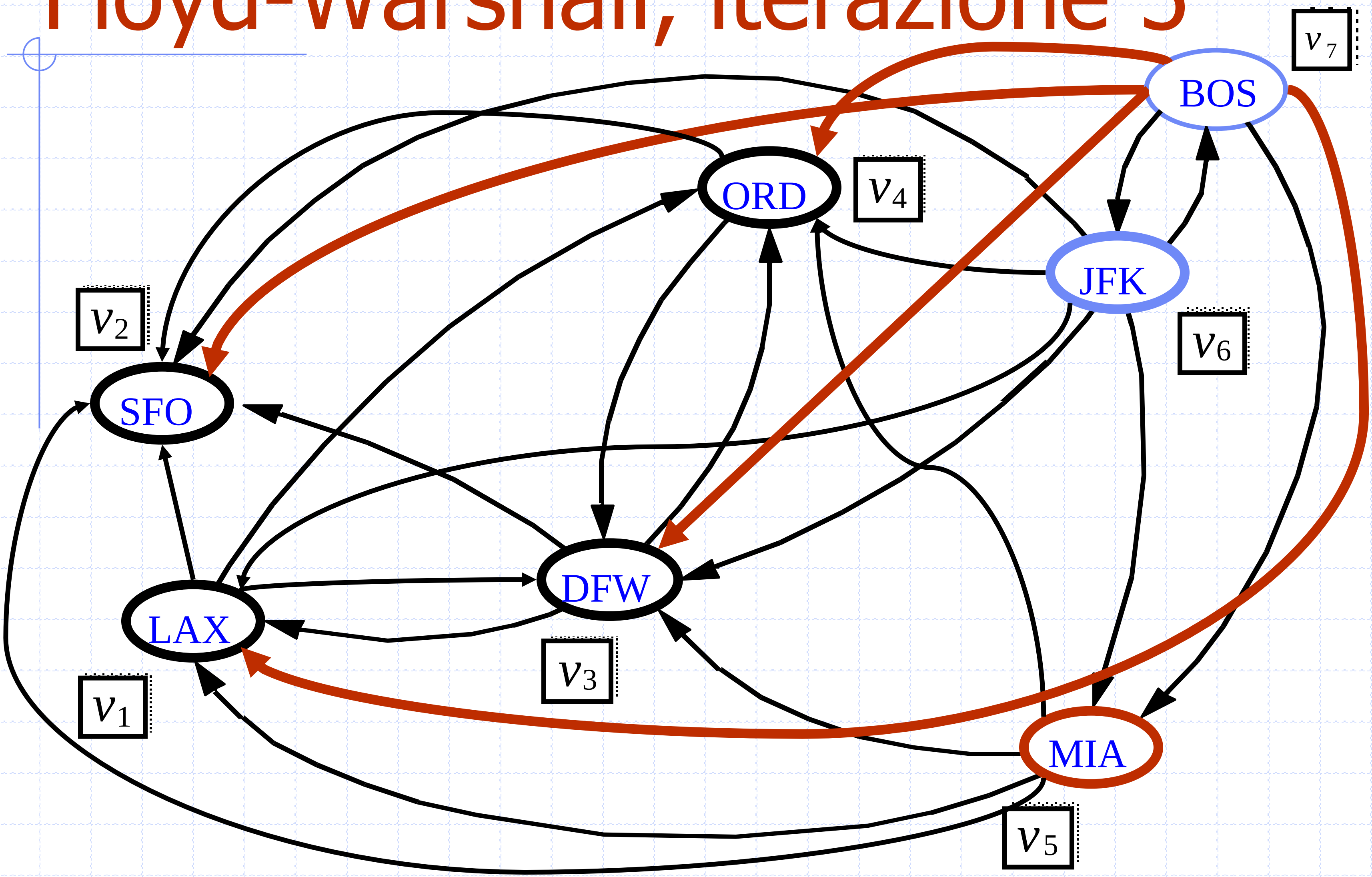
Floyd-Warshall, iterazione 3



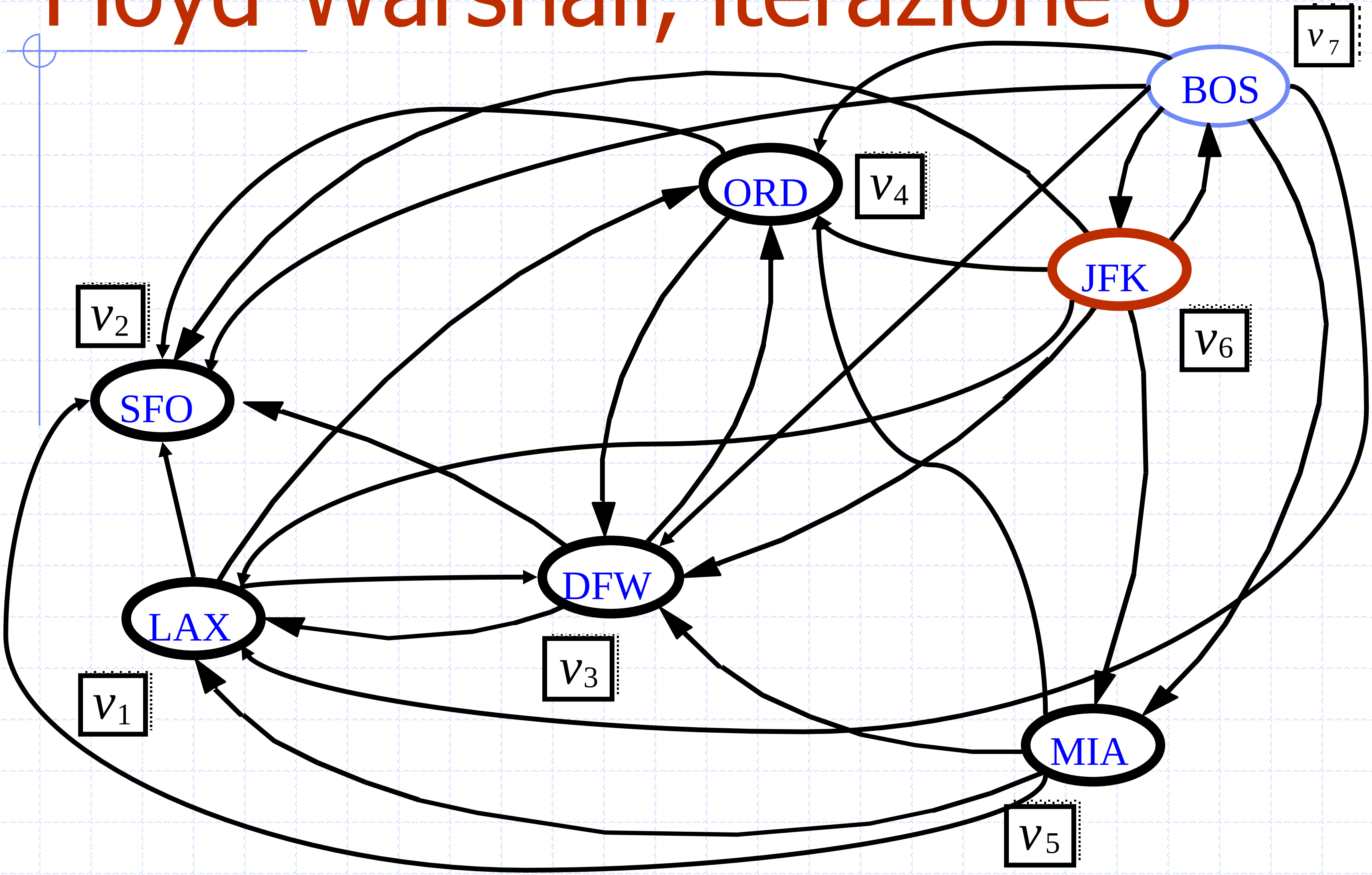
Floyd-Warshall, iterazione 4



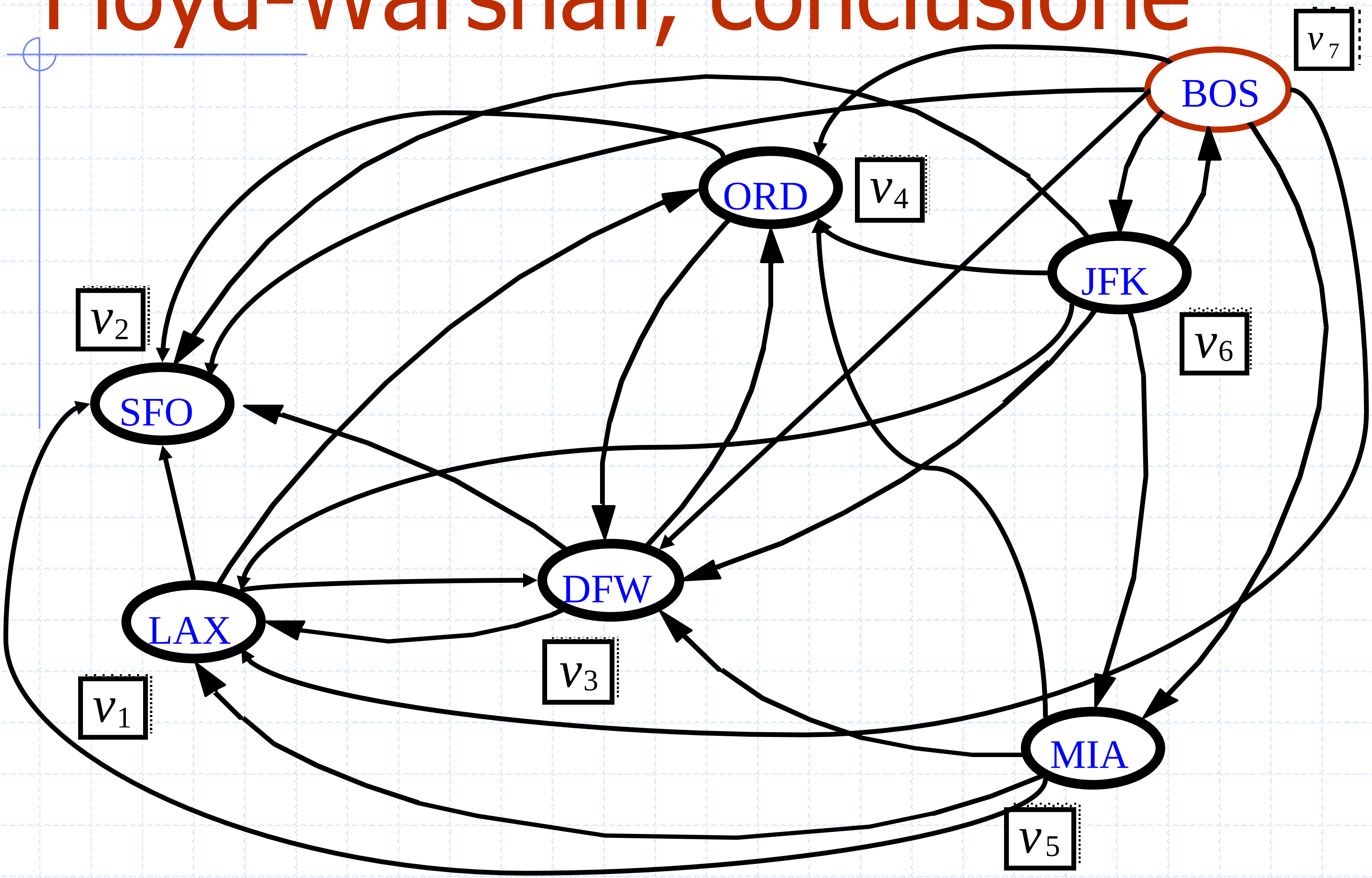
Floyd-Warshall, iterazione 5



Floyd-Warshall, iterazione 6



Floyd-Warshall, conclusione



DAG e ordinamento topologico

- Un grafo diretto aciclico è un digrafo che non ha cicli orientati (directed acyclic graph, DAG)
- Un ordinamento topologico di un digrafo è una numerazione

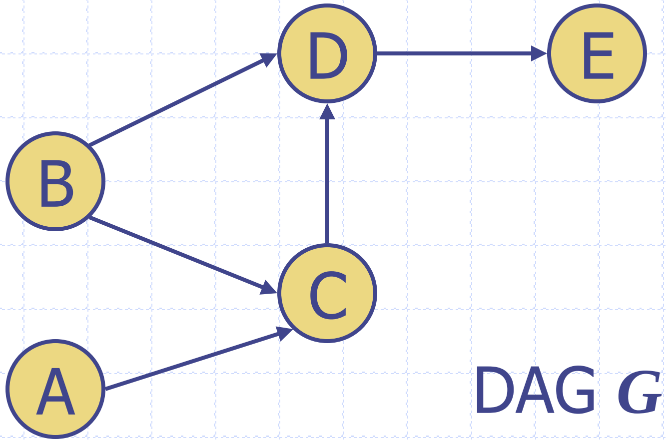
$$v_1, \ldots, v_n$$

dei vertici tale che per ogni spigolo (v_i, v_j) risulta $i < j$

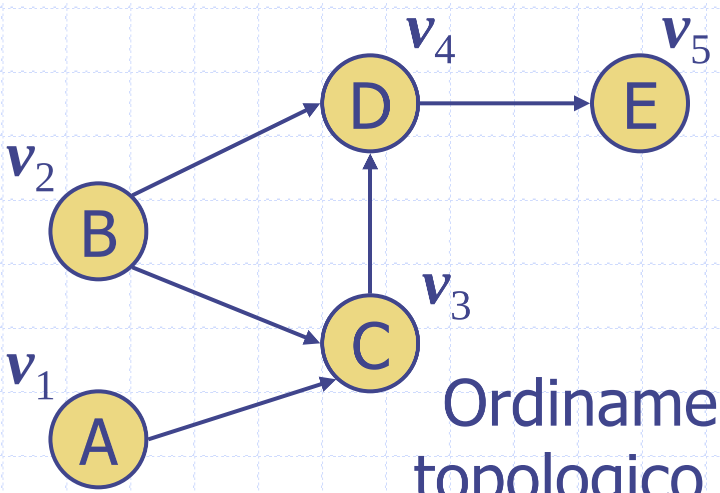
  - Esempio: in un DAG di pianificazione di attività un ordinamento topologico è un sequenziamento delle attività che soddisfa tutti i vincoli di precedenza

Teorema

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



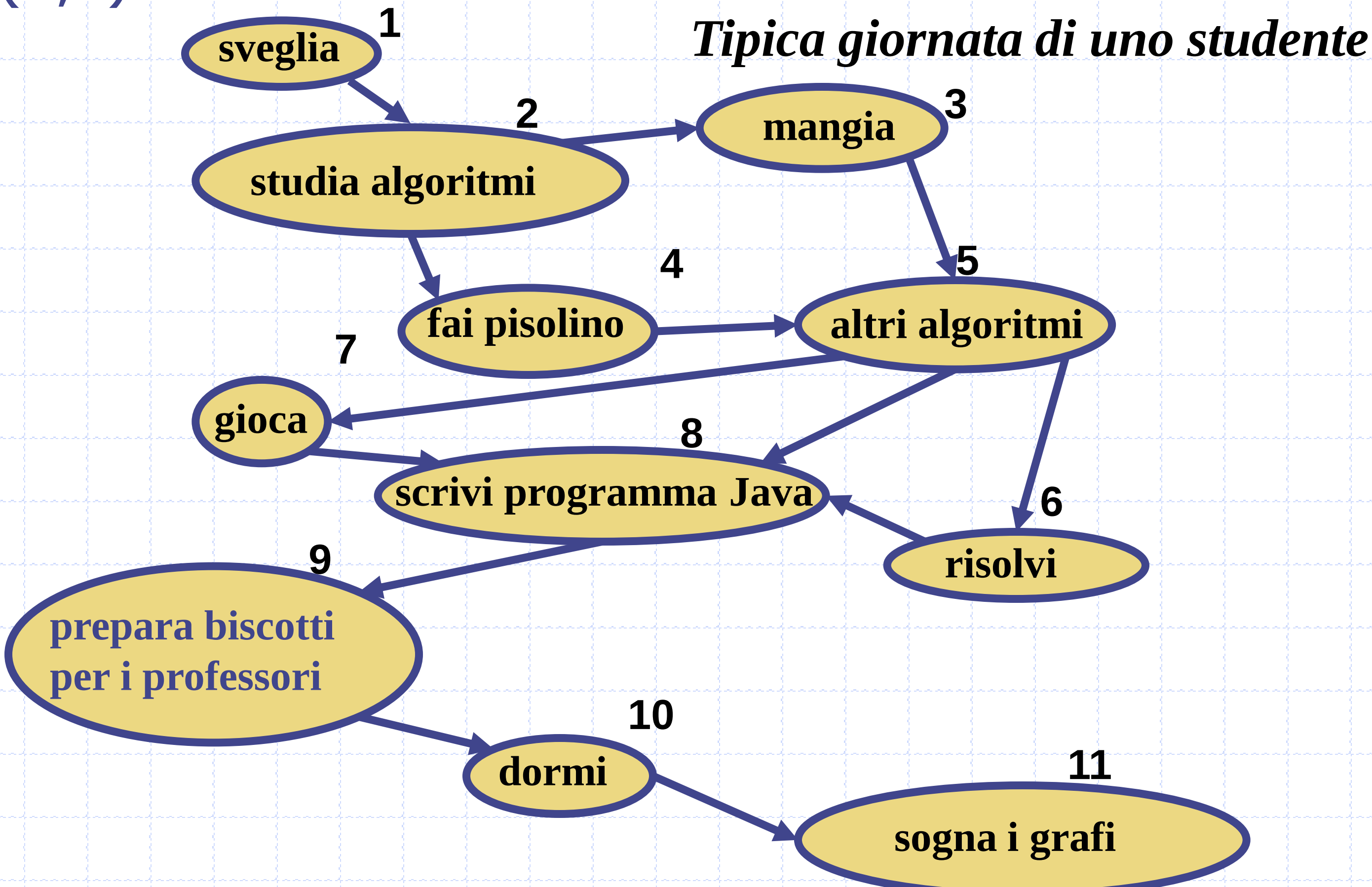
DAG G



Ordinamento topologico di G

Ordinare topologicamente

- Numera i vertici in modo tale che se esiste lo spigolo (u,v) allora u < v



Algoritmo per l'ordinamento topologico

- N.B.: questo algoritmo è differente da quello sul testo Goodrich-Tamassia

```
Method TopologicalSort(G)
    H ← G            // copia temporanea di G
    n ← G.numVertices()
    while H è non vuoto do
        trova un pozzo v // esiste sempre?
        etichetta v ← n
        n ← n - 1
        rimuovi v da H // anche gli spigoli incidenti
```

- Tempo di esecuzione: $O(n + m)$. Come?

Algoritmo di ordinamento topologico basato su DFS

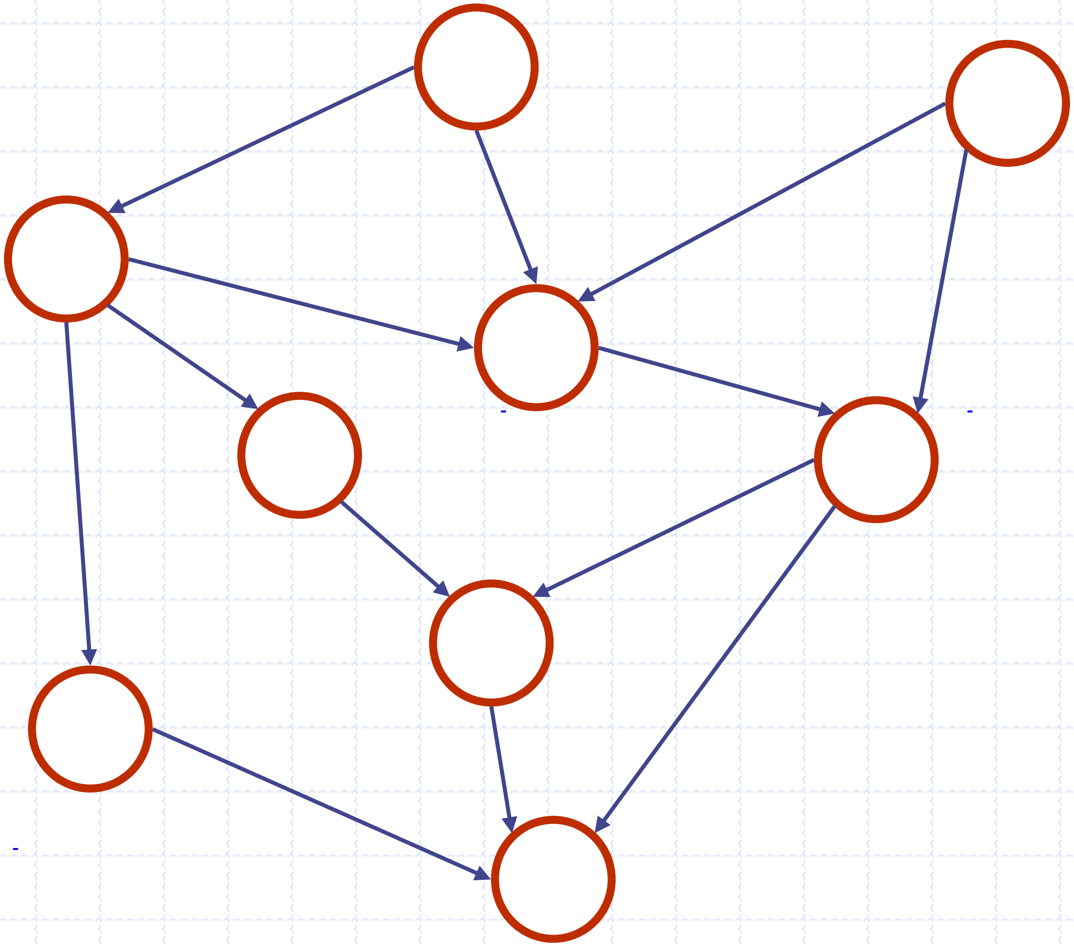
- Si raggiunge l'obiettivo usando una DFS

```
Algorithm topologicalDFS(G)
   Input DAG G
   Output ordinamento topologico di G
  n ← G.numVertices()
  for all u ∈ G.vertices()
    setLabel(u, UNEXPLORED)
  for all e ∈ G.edges()
    setLabel(e, UNEXPLORED)
  for all v ∈ G.vertices()
    if getLabel(v) = UNEXPLORED
      topologicalDFS(G, v)
```

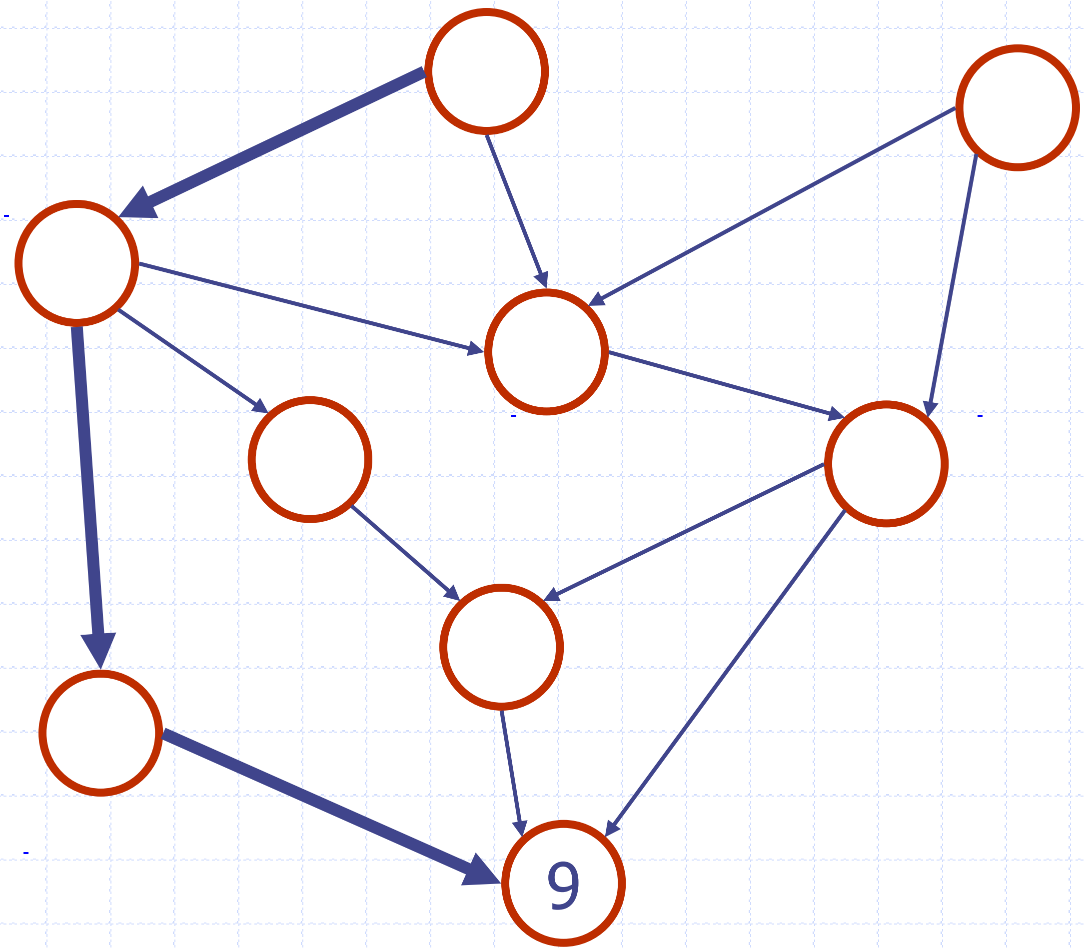
- tempo O(n+m)

```
Algorithm topologicalDFS(G, v)
  Input grafo G e vertice iniziale v di G
  Output etichettatura dei vertici di G
      nella componente connessa di v
  setLabel(v, VISITED)
  for all e ∈ G.incidentEdges(v)
    if getLabel(e) = UNEXPLORED
      w ← opposite(v,e)
      if getLabel(w) = UNEXPLORED
        setLabel(e, DISCOVERY)
        topologicalDFS(G, w)
      else
        { e è uno spigolo forward o cross }
  etichetta v con il numero n
  n ← n – 1 // side effect!
```

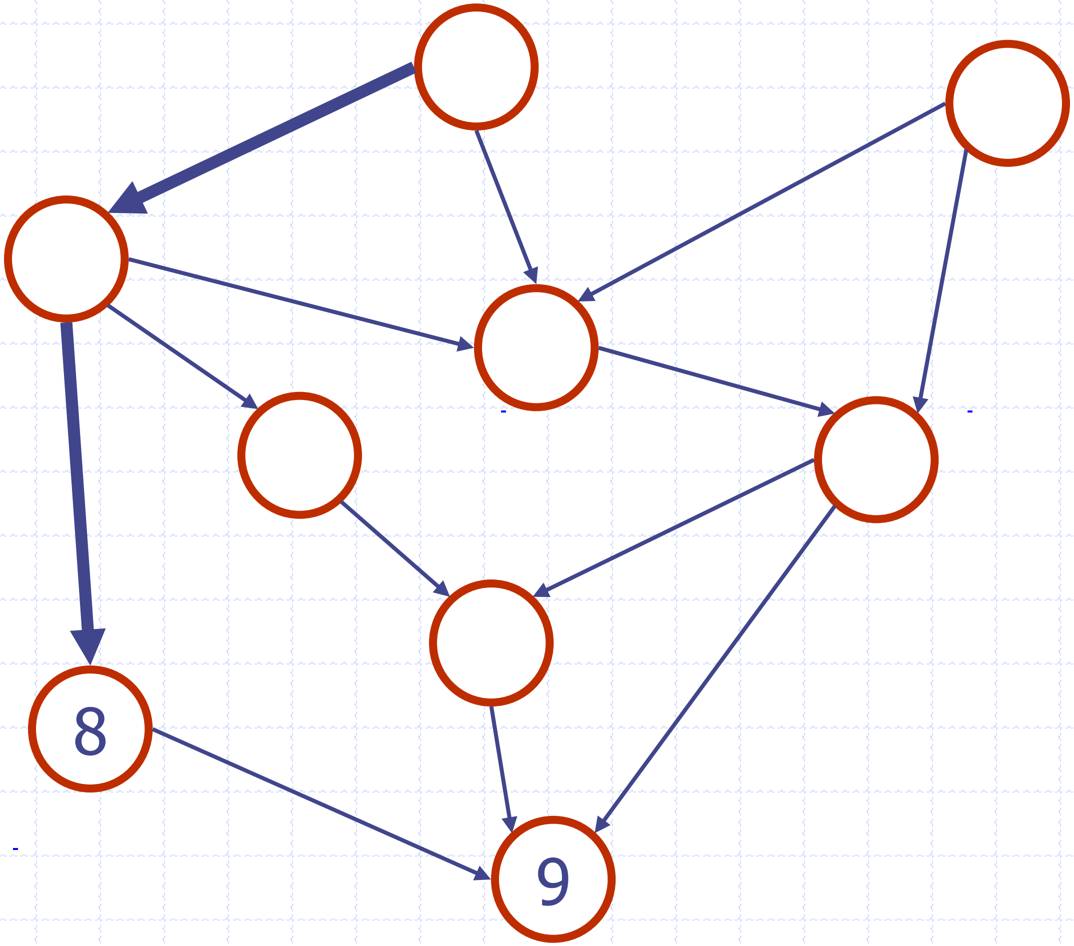
Esempio di ordinamento topologico



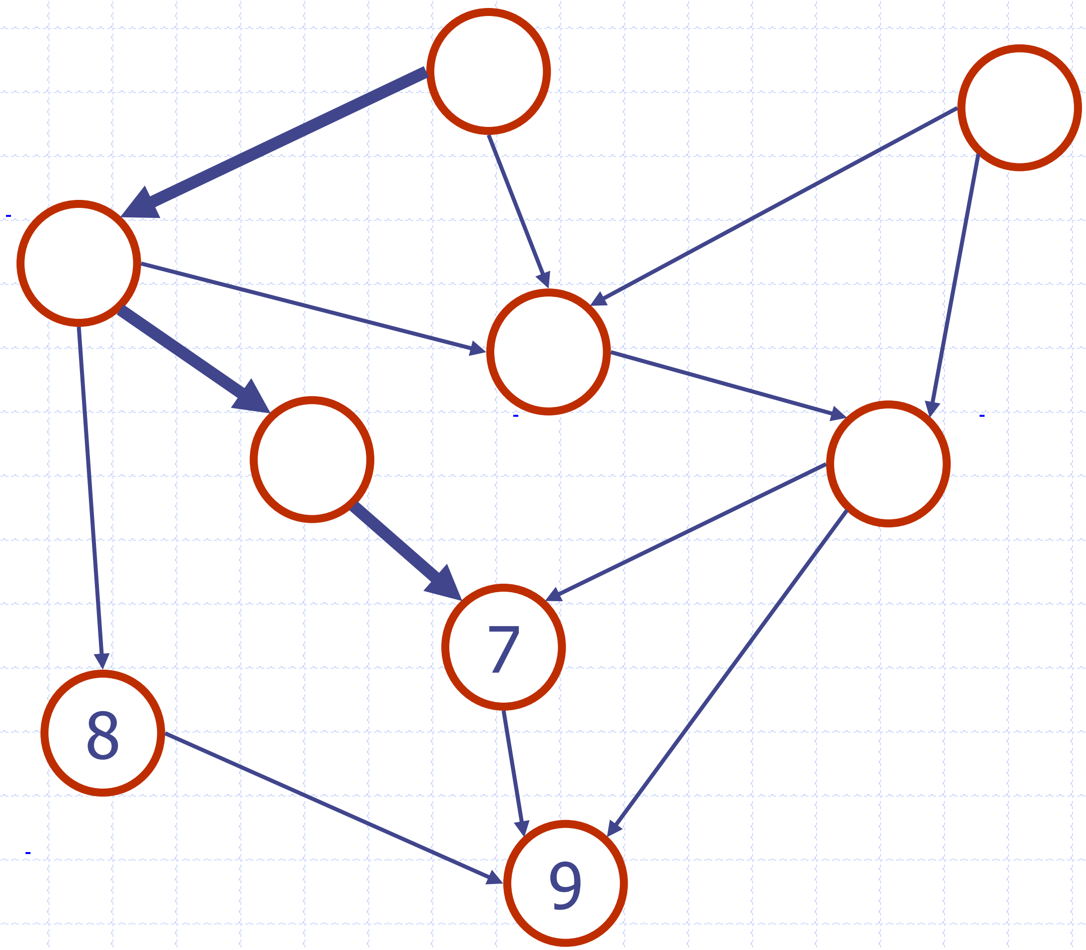
Esempio di ordinamento topologico



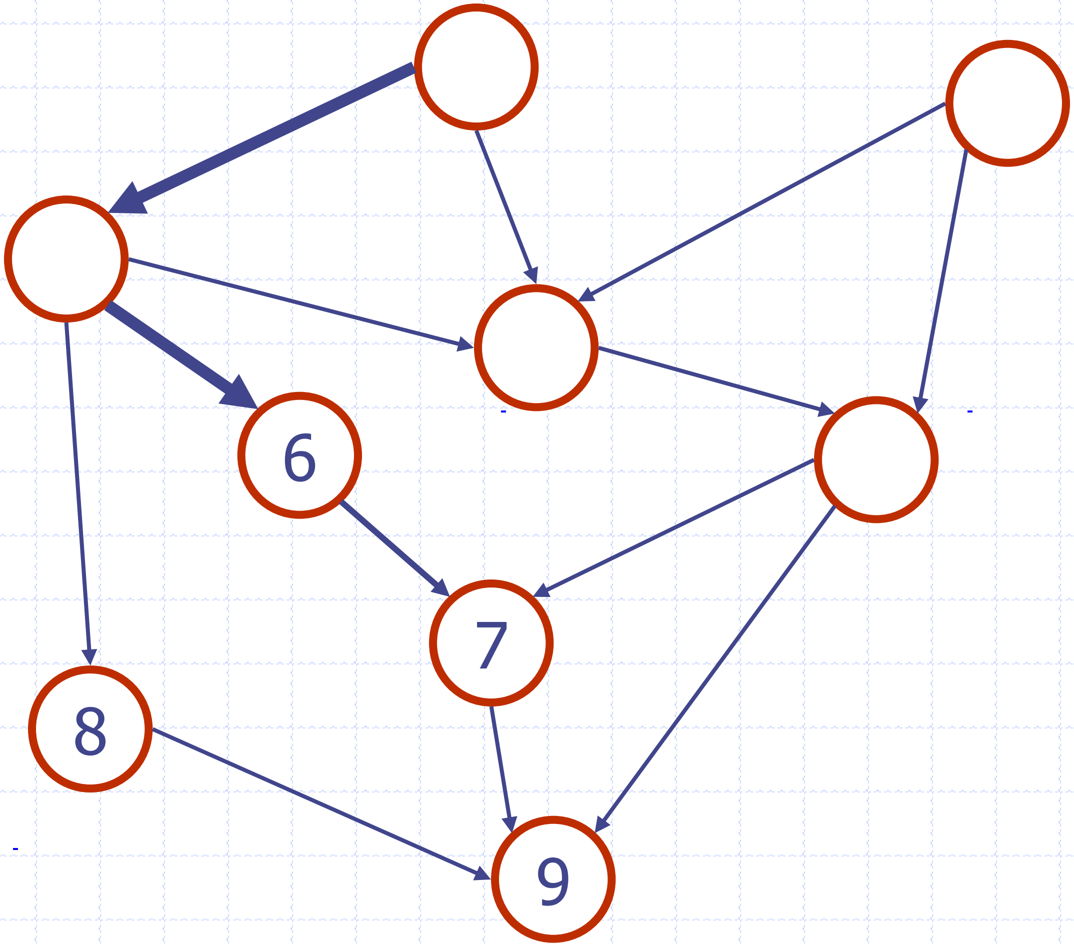
Esempio di ordinamento topologico



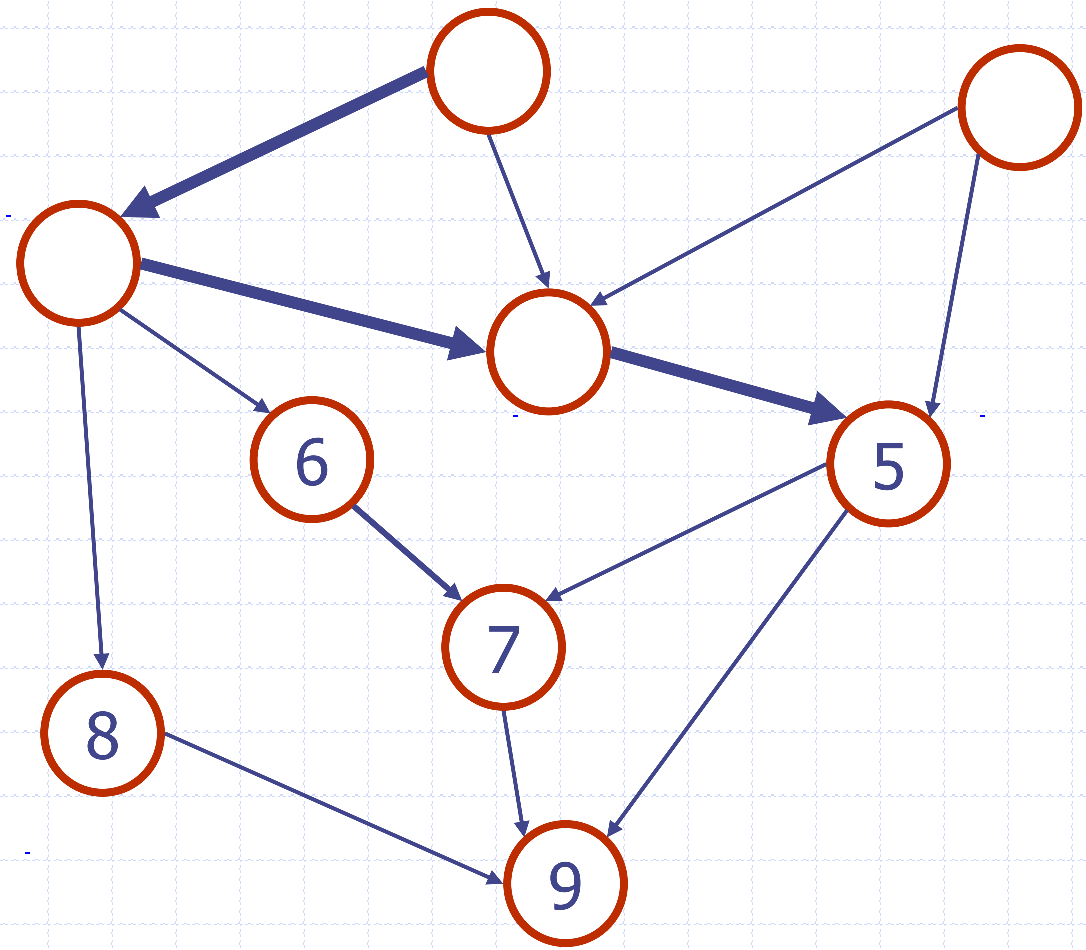
Esempio di ordinamento topologico



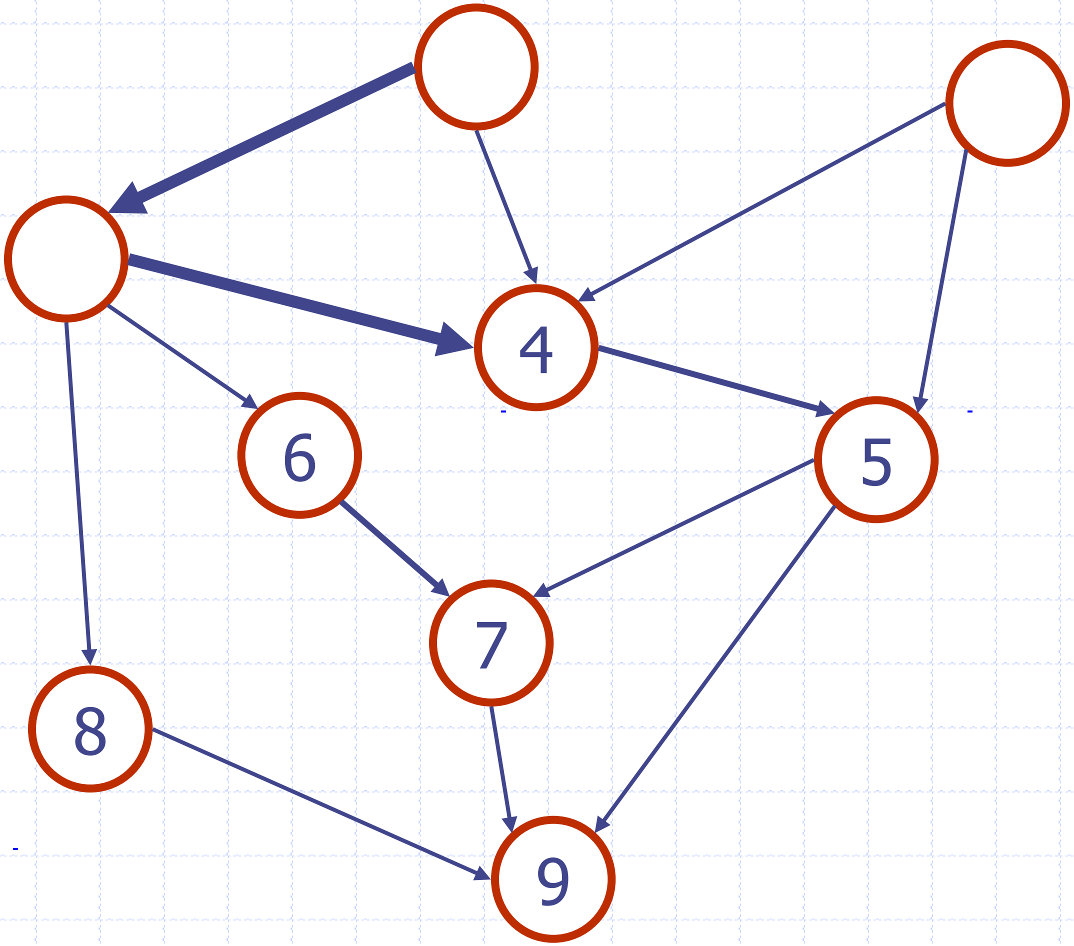
Esempio di ordinamento topologico



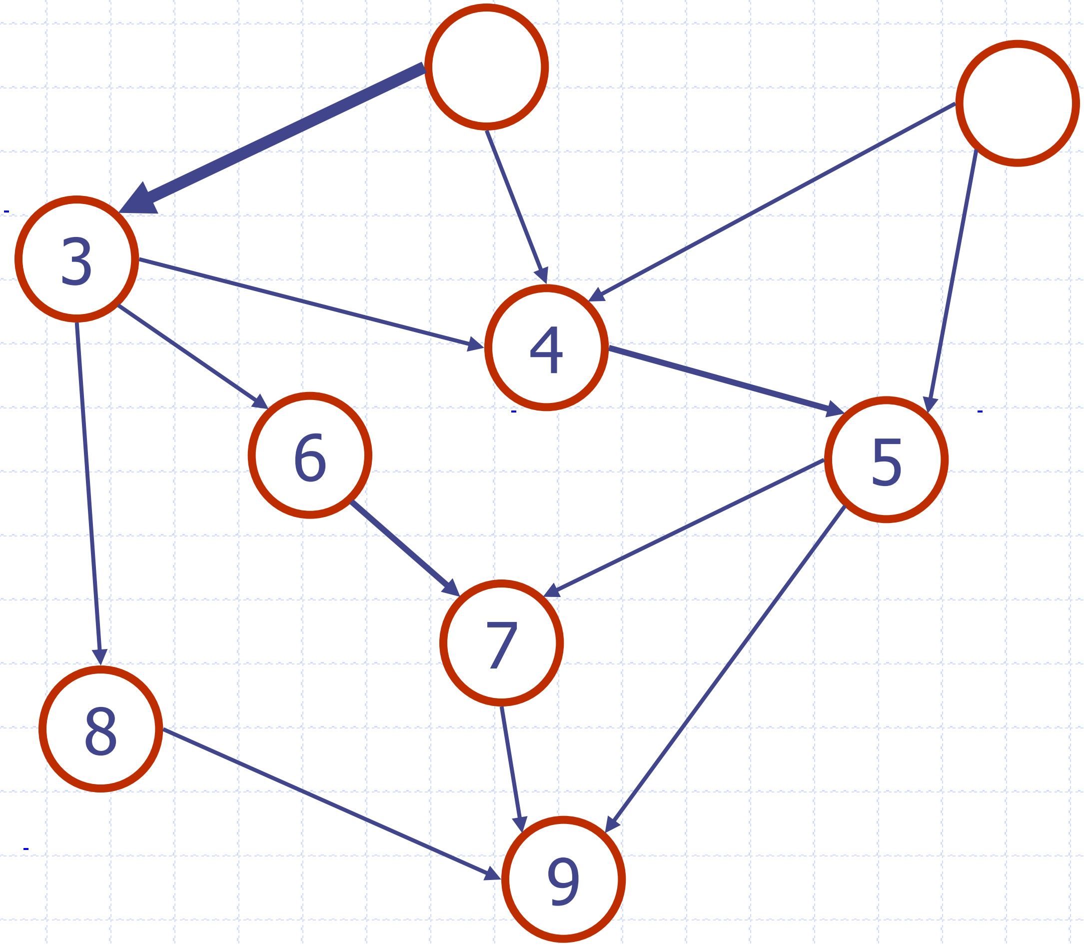
Esempio di ordinamento topologico



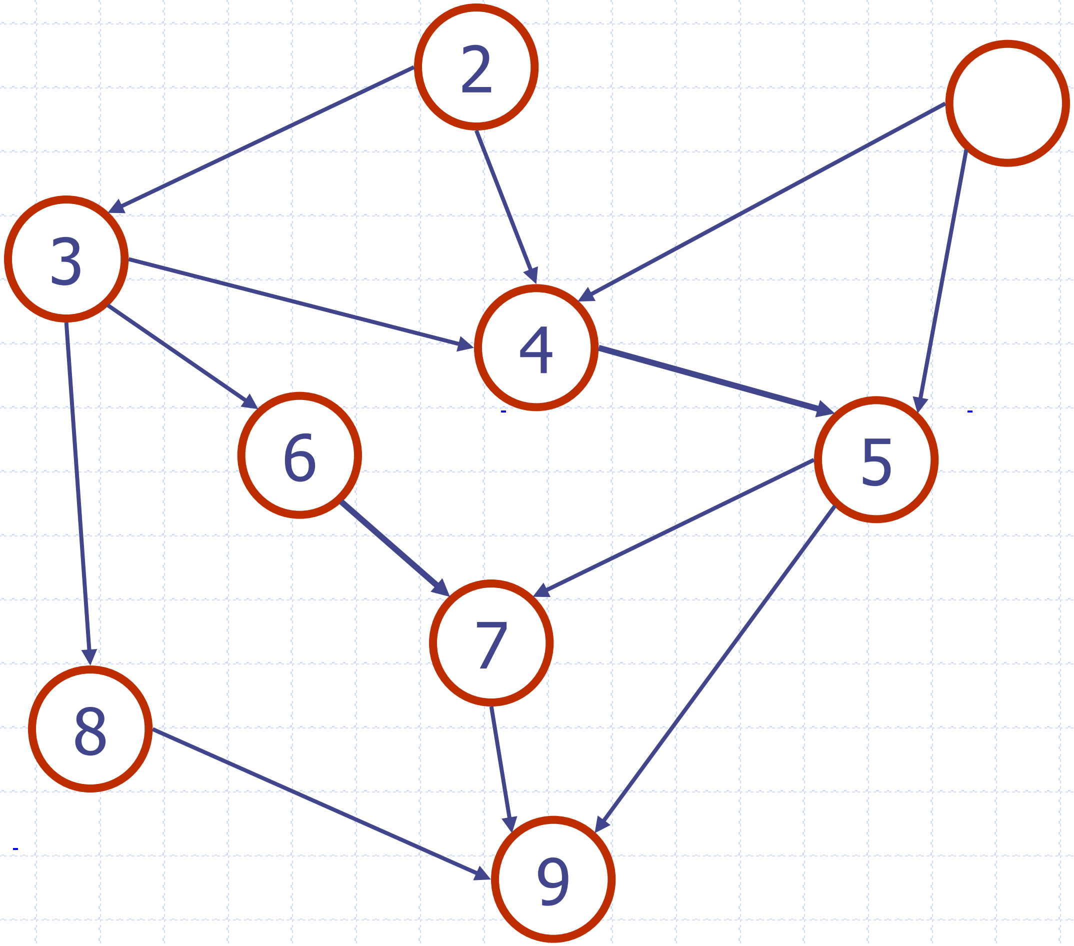
Esempio di ordinamento topologico



Esempio di ordinamento topologico



Esempio di ordinamento topologico



Esempio di ordinamento topologico