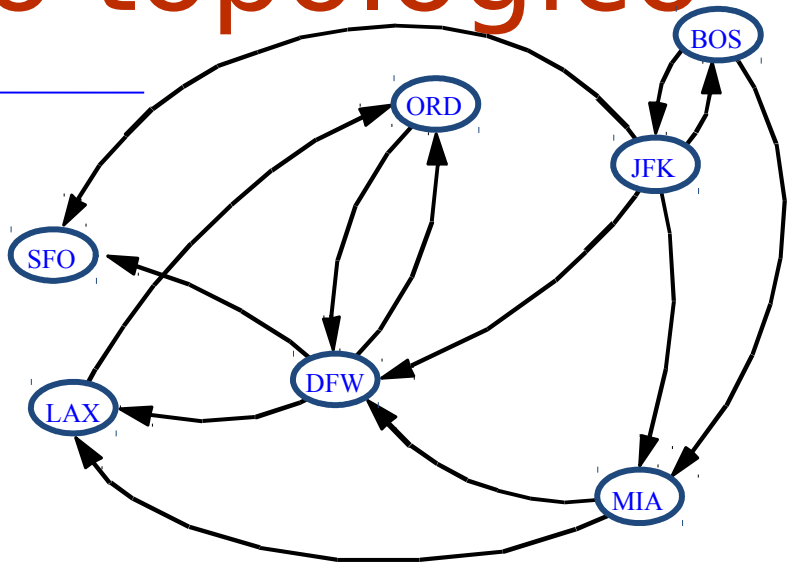
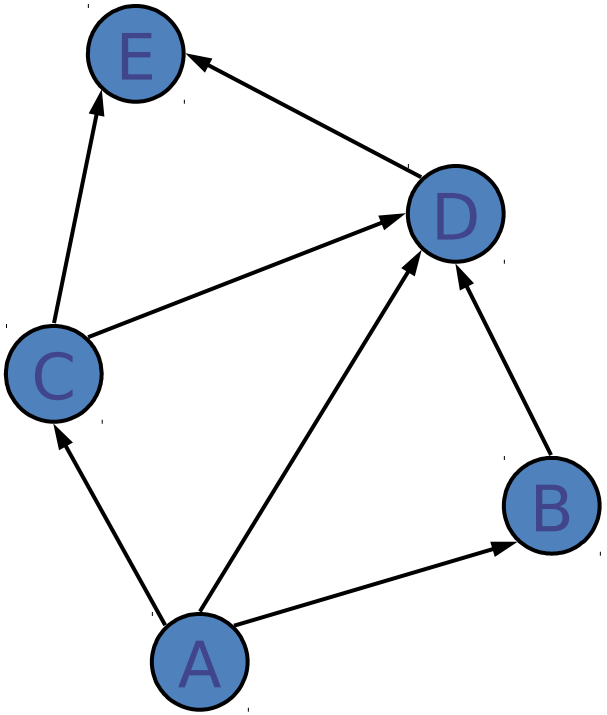
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

SCC, Chiusura transitiva Ordinamento topologico

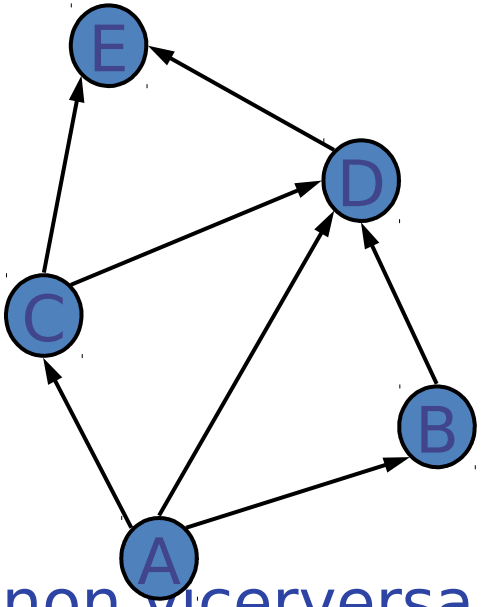


Grafi diretti

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



Proprietà

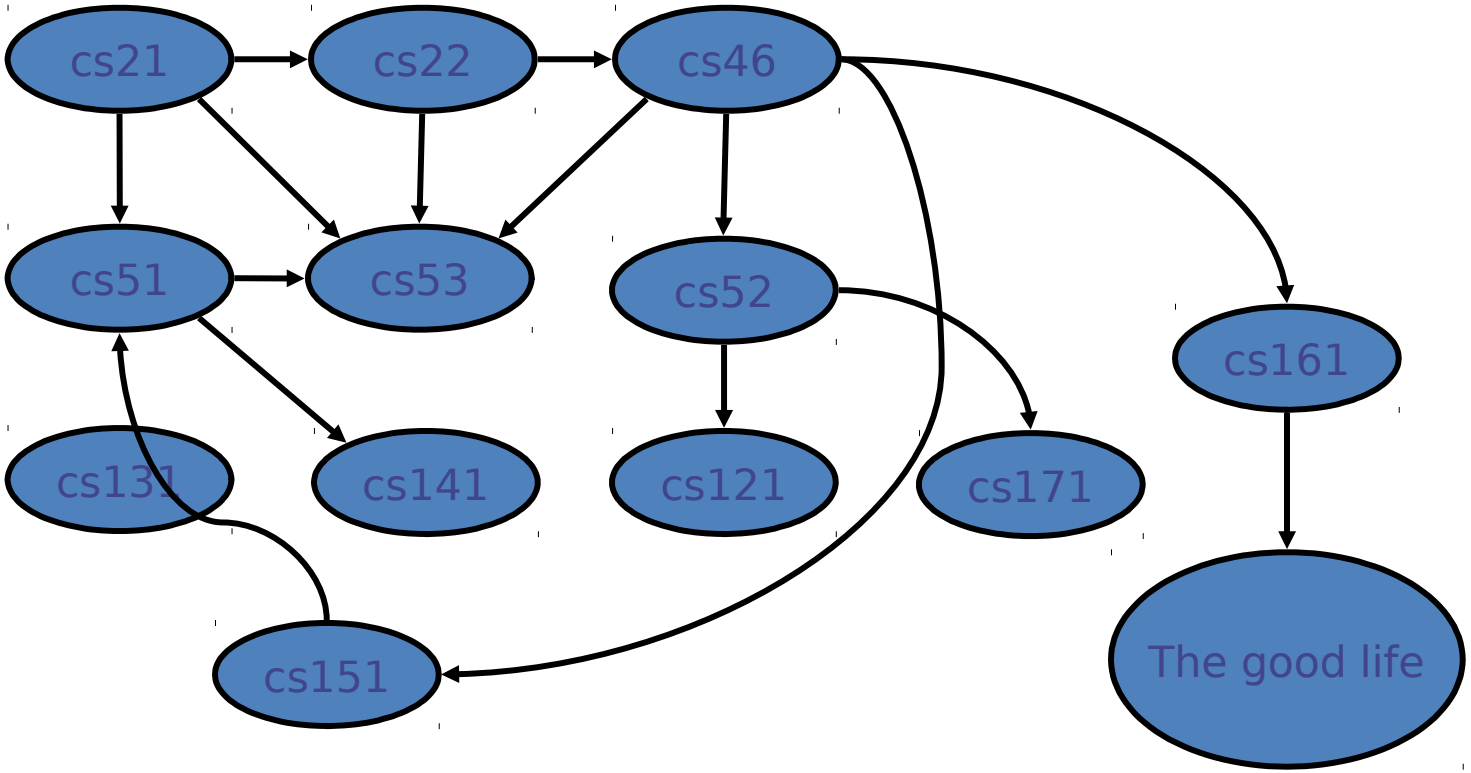


- Grafo G=(V,E) tale che
  - Ogni arco ha una direzione:
  - L'arco (a,b) va da a verso b, ma non viceversa

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

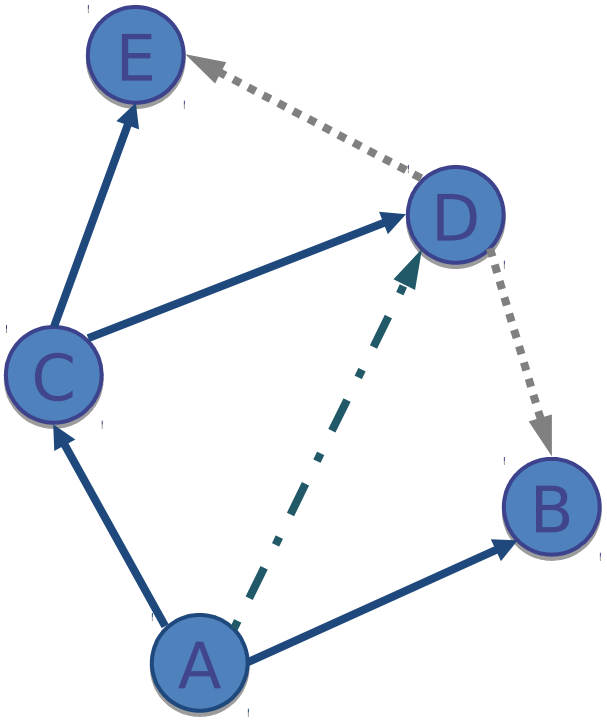
Esempio di applicazione

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



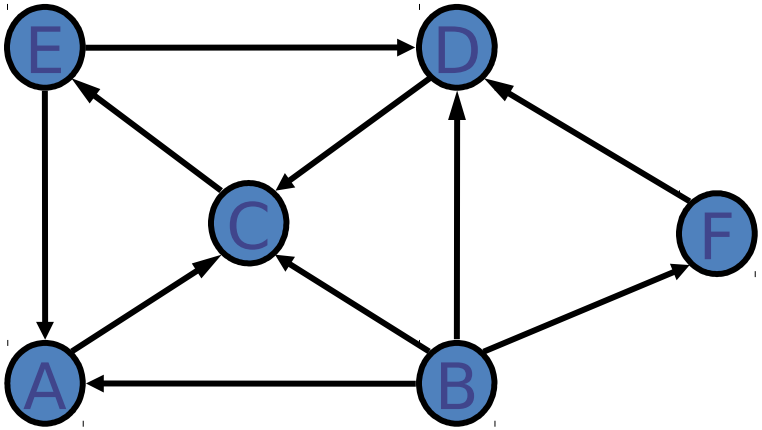
DFS diretta

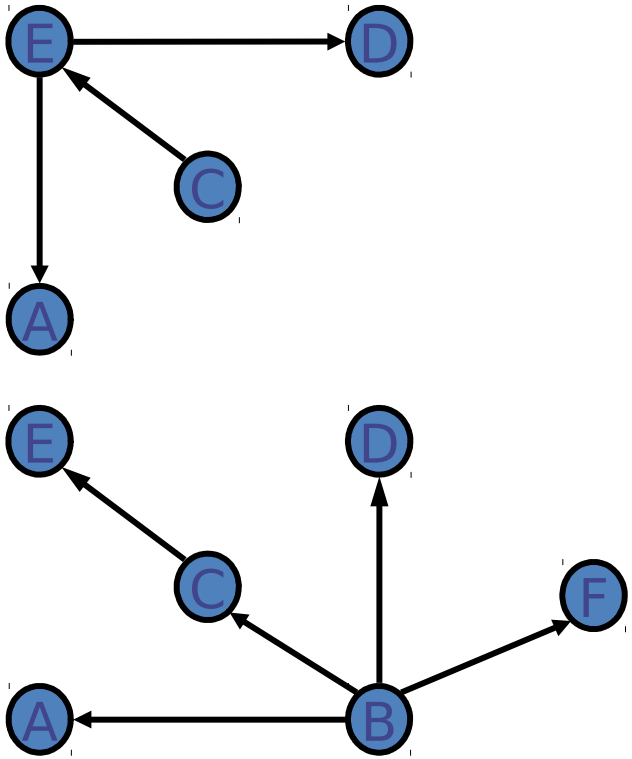
- La DFS in generale funziona anche per grafi diretti
- Gli archi sono attraversati soltanto nella loro direzioni
- 4 tipi di archi nella DFS diretta
  - Archi discovery
  - Archi back
  - Archi forward
  - Archi cross
- Una DFS a partire da un vertice s calcola il sottoinsieme dei vertici raggiungibili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



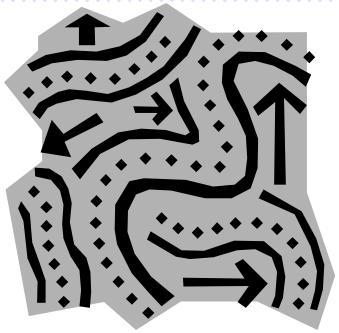
Raggiungibilità

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

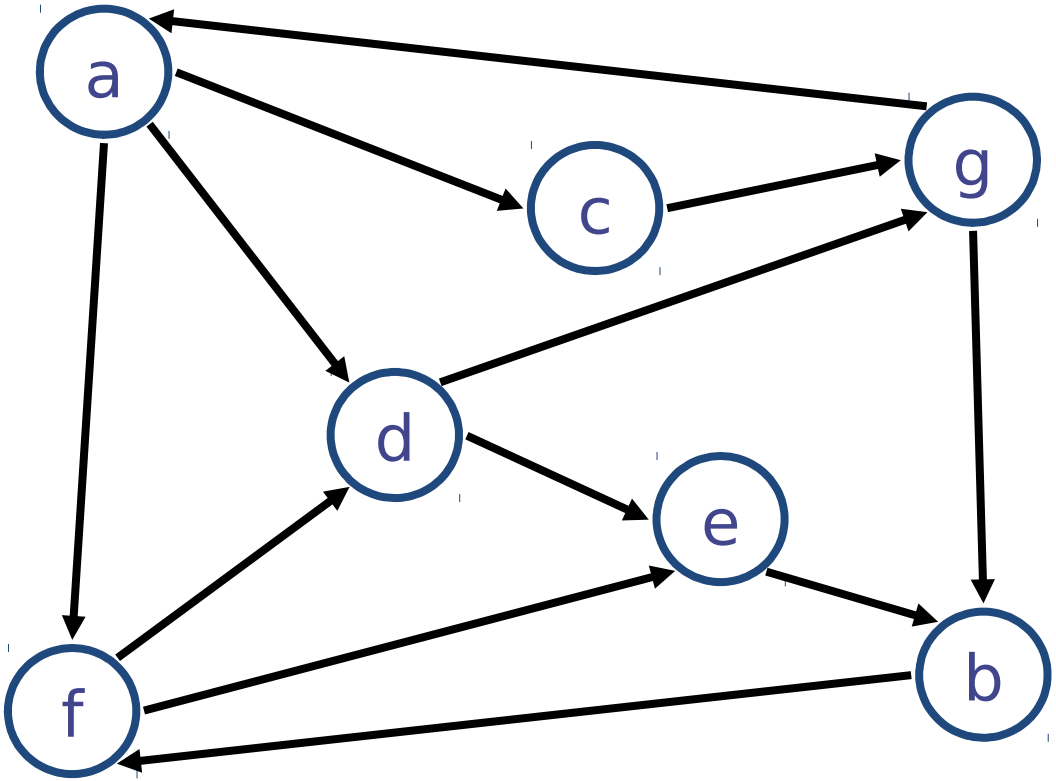




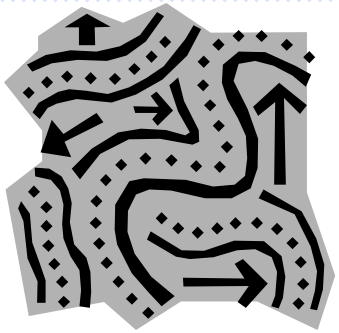
Connettività forte



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

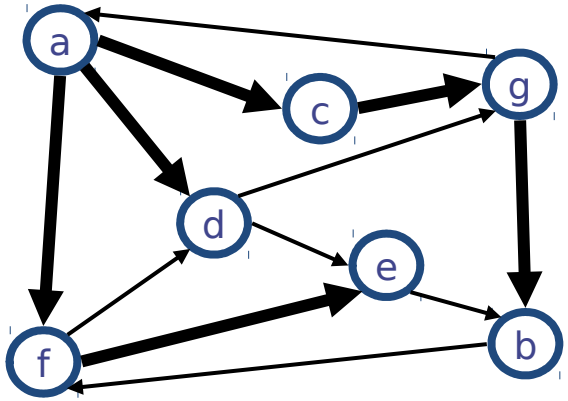


Connettività forte: algoritmo

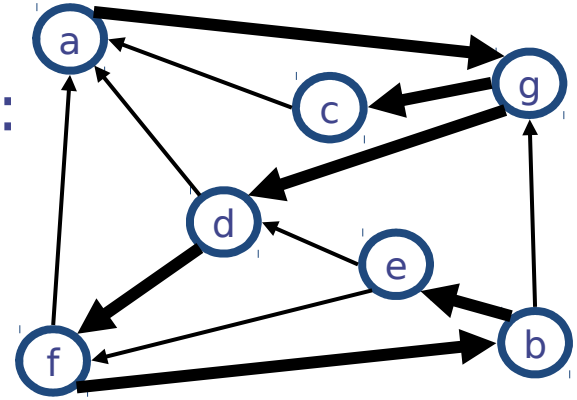


- Scegli un vetice v di G
- DFS(v, G)
  - Se esiste w non visitato --> return “no”
- Sia G’ uguale a G ma con gli archi invertiti
- DFS(v, G’) from v in G’
  - Se esiste w non visitato --> return “no”
    - Else, return “yes”
- Complessità: O(n+m)

G:

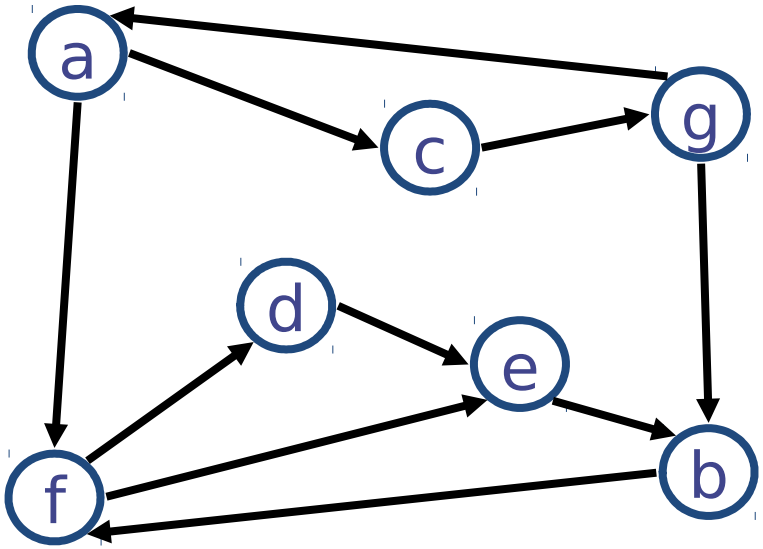


G’:



Componenti fortemente connesse

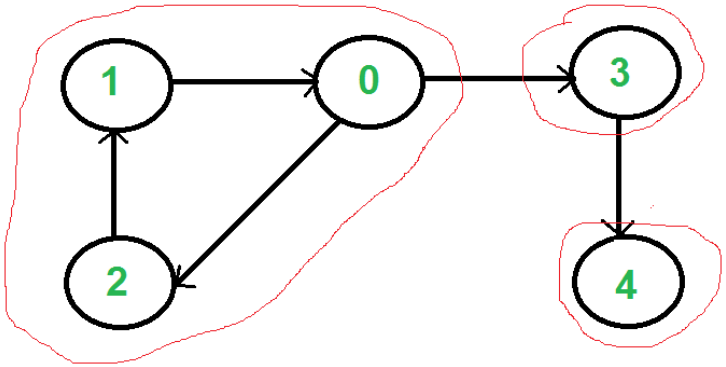
- Sottografi massimali che sono fortemente connessi
- Si può eseguire con complessità O(n+m) usando la DFS



{ a , c , g }

{ f , d , e , b }

Algoritmo di Kosaraju



- Idea dell'algoritmo:
  - Prima DFS del grafo
    - Vertici inseriti in una pila s
  - Seconda DFS sul grafo tras
    - Per ogni v in S:
      - DFS(v, G^T)
      - l'insieme dei nodi trovati è una nuova componente connessa
  - Complessità: O(n + m)
- Perché funziona: il principio è lo stesso della verifica di connettività forte

Algoritmo di Kosaraju: pseudocodice

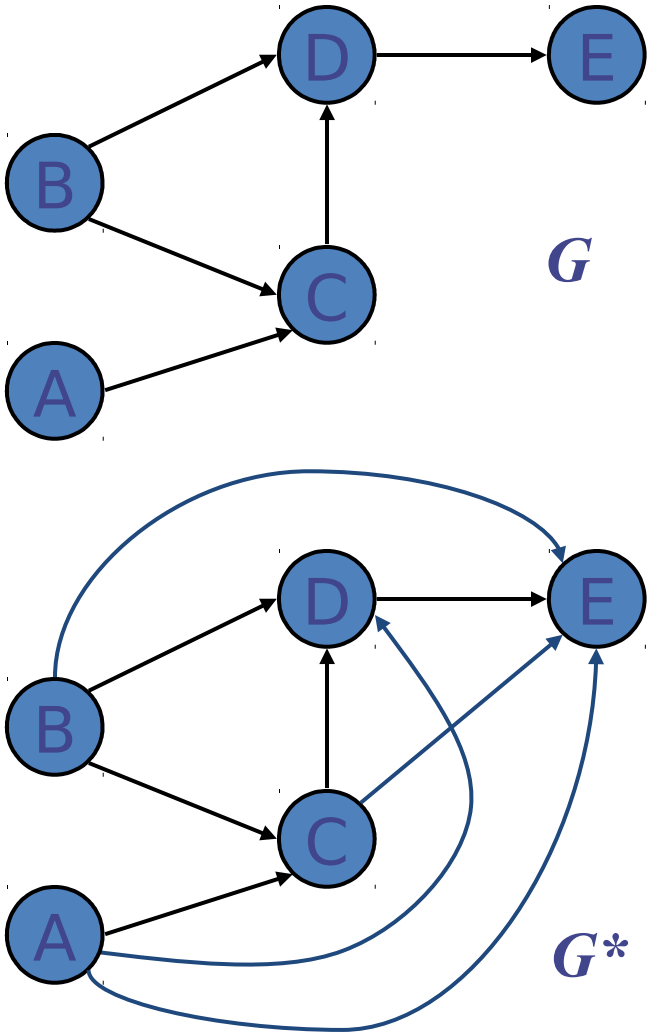
```
DDFS(v, Stack s) {
    if (v.state != UNEXPLORED)
        return;
    v.state = EXPLORING;
    for(cur in v.outEdges) {
        DDFS(cur, s);
    }
    v.state = EXPLORED;
    s.push(v);
}
```

```
SCC(G) {
    forest = new List() // Lista componenti connesse
    // Prima DFS
    stack = new Stack();
    for(v in G) {
        if(v.state == UNEXPLORED)
            DDFS(v, stack);
    }
    // Reset dello stato dei nodi
    for(v in G)
        v.state = UNEXPLORED;
    // Seconda DFS sul grafo trasposto
    for(v in stack) { // Nota: vertici estratti dalla pila
        if(v.state == UNEXPLORED) {
            ret = new List();
            transposedDFS(G, v, ret);
            forest.add(ret);
        }
    }
}
```

```
transposeDFS(G, n, lista) {
        // Effettua DFS sul grafo trasposto di G a partire da n
        // Inserisce i nodi trovati in lista
        // Nota: non è necessario trasporre effettivamente il grafo
}
```

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 ($u \neq v$), G^* ha un arco diretto da u a v
- Chiusura transitiva → informazione di raggiungibilità



Calcolo della chiusura transitiva

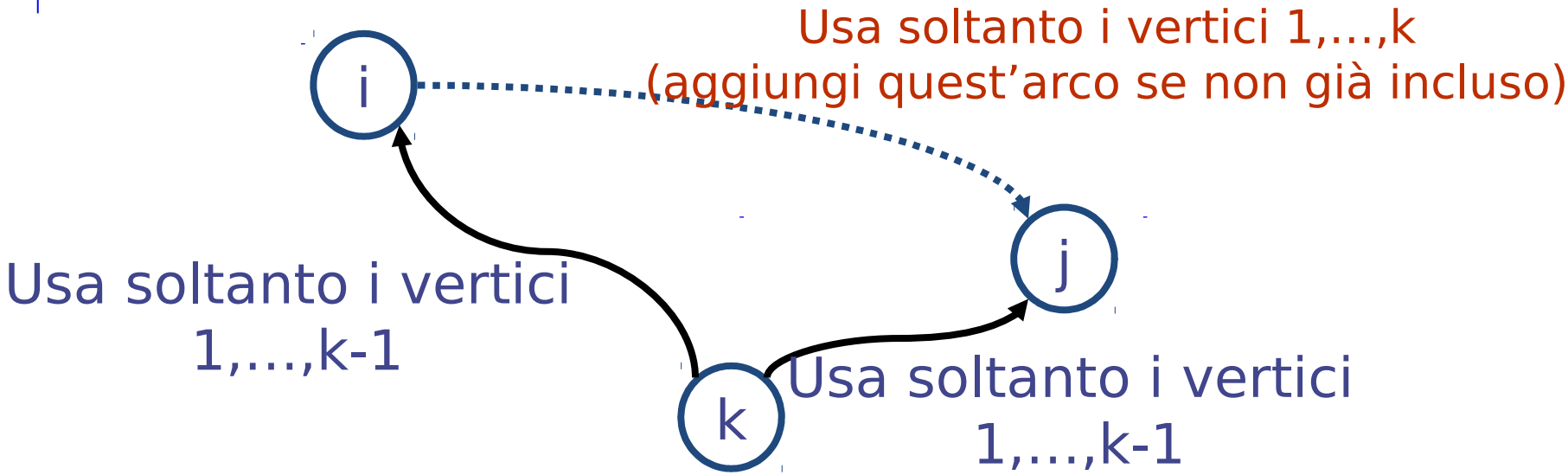
- DFS da ognuno dei vertici
- O(n(n+m))
- Sfruttare la matrice di adiacenza
  - $O(n^{c+1})$
  - c è almeno 2.37

Se è possibile andare da A a B e da B a C, allora è possibile andare da A a C.

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

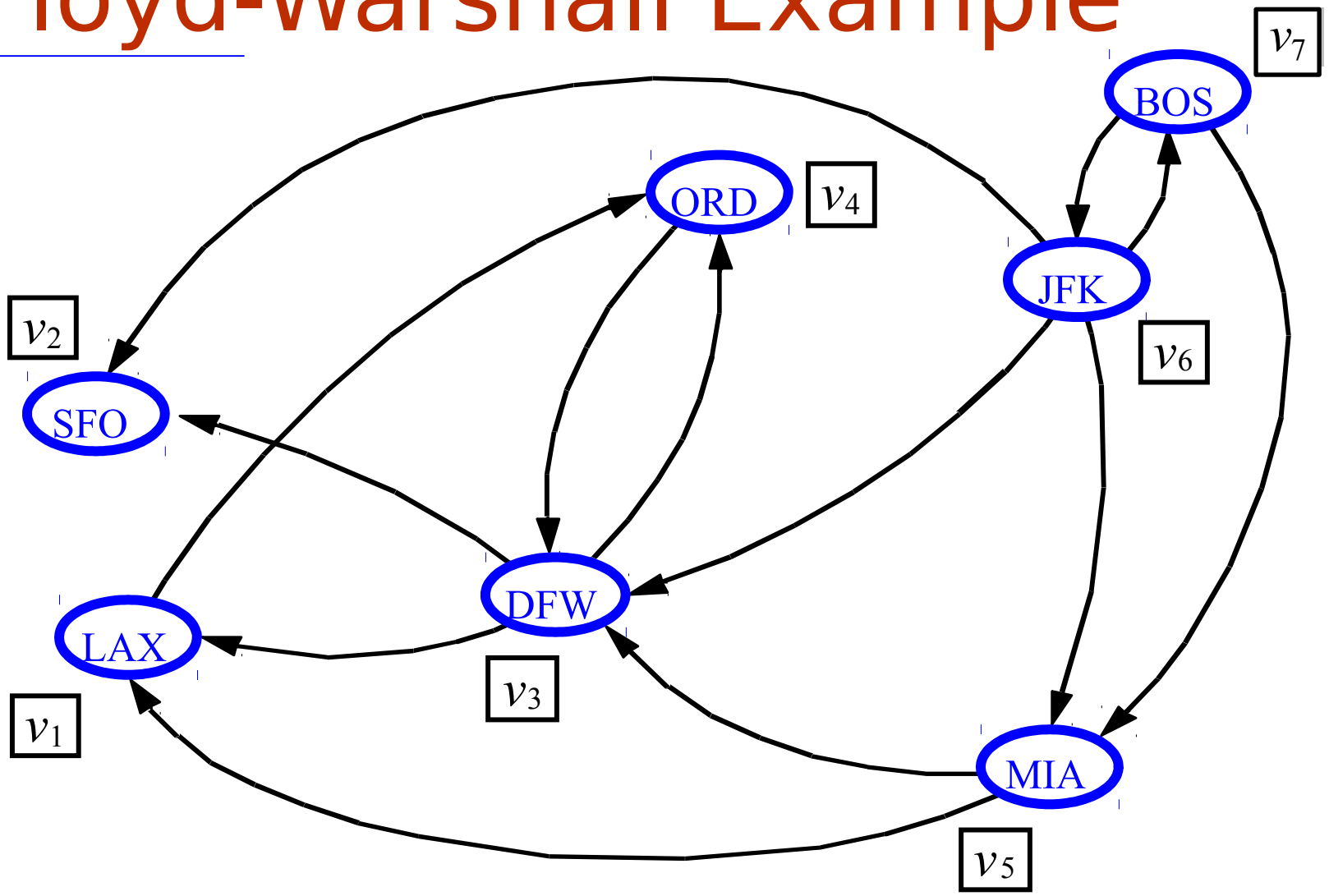
- Arco diretto (i, j) in G_{k-1} oppure:
- Archi diretti (i, k) e (k, j) in G_{k-1}

} Programmazione dinamica

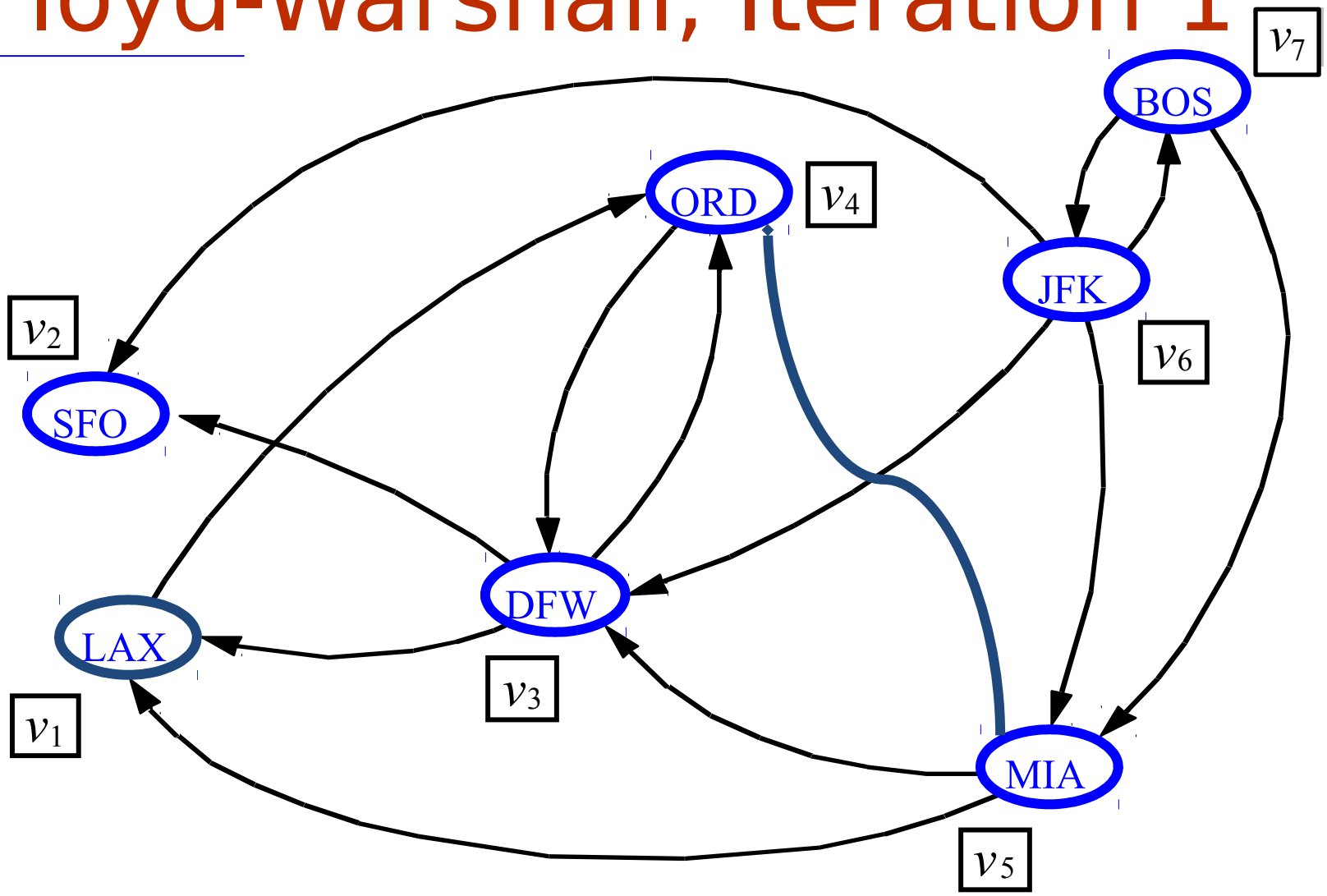
Algorithm FloydWarshall(G)
$G_0 = G$
for k = 1 to n {
    $G_k = G_{k-1}$
    for (i, j con $i \neq k$ e $j \neq k$)
        if ((i, j) non appartiene a G_{k-1})
            if ((i, k) $\in G_{k-1}$ and (k, j) $\in G_{k-1}$)
                <Aggiungi (i, k) a G_k>

Determinare il costo computazionale

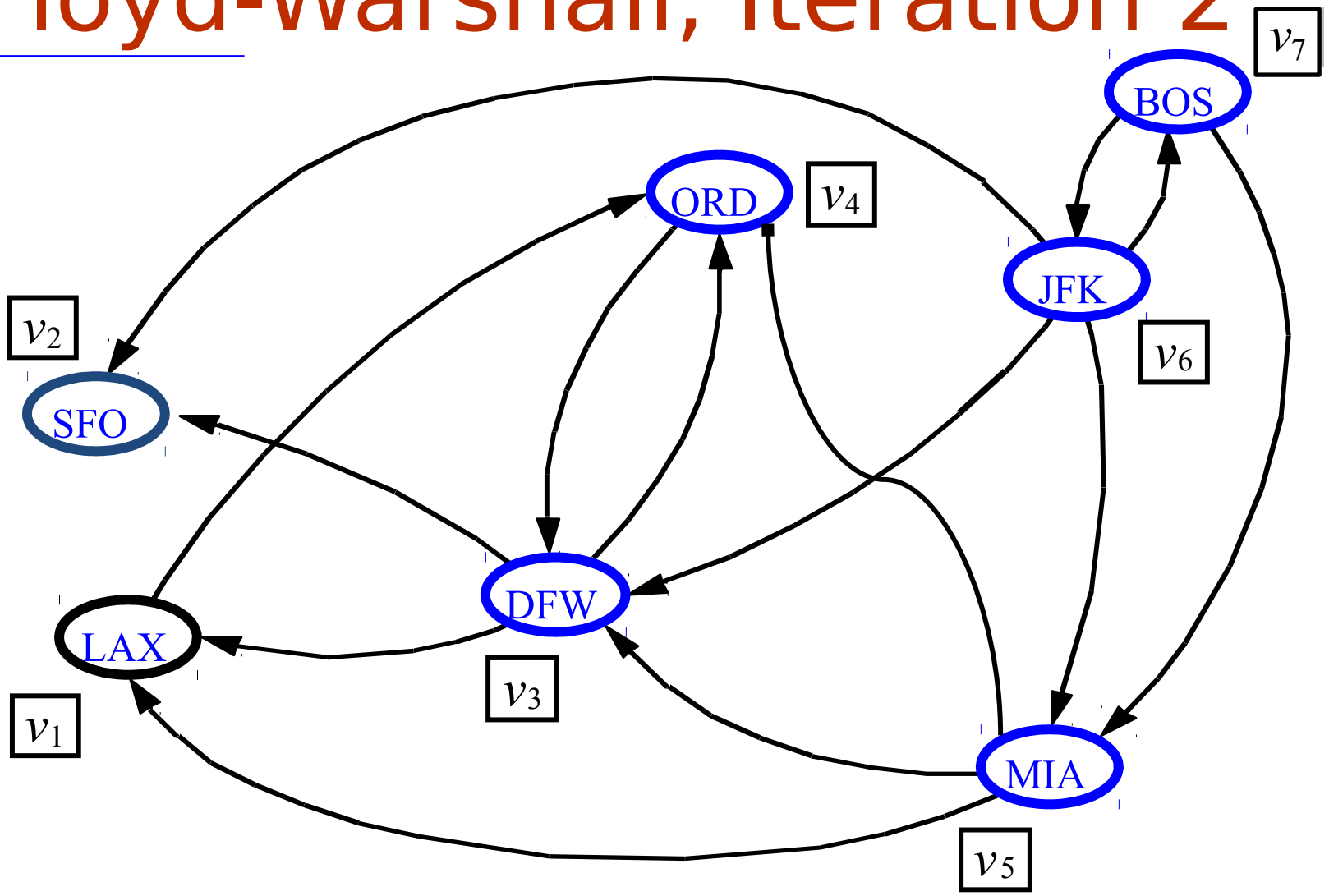
Floyd-Warshall Example



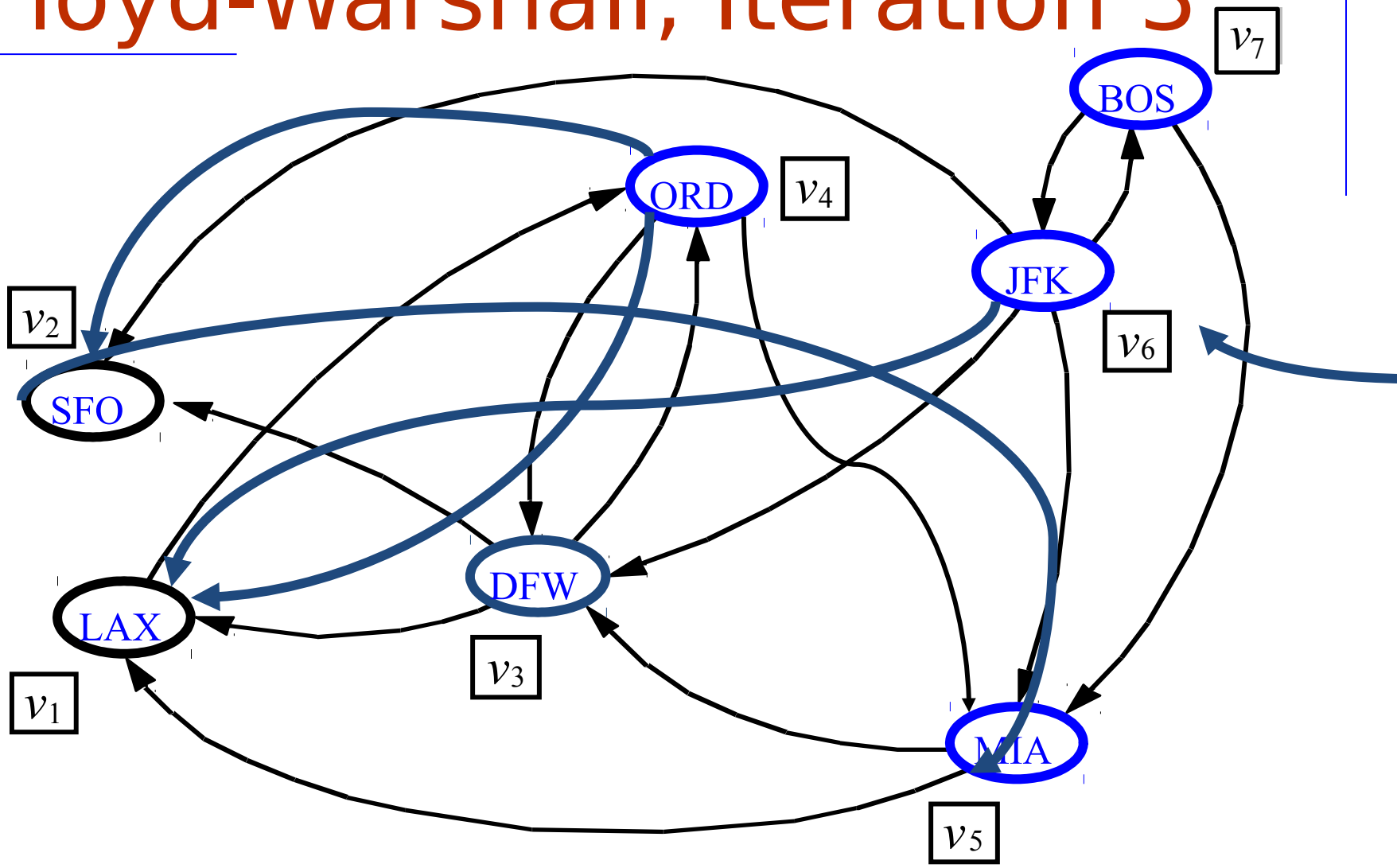
Floyd-Warshall, Iteration 1



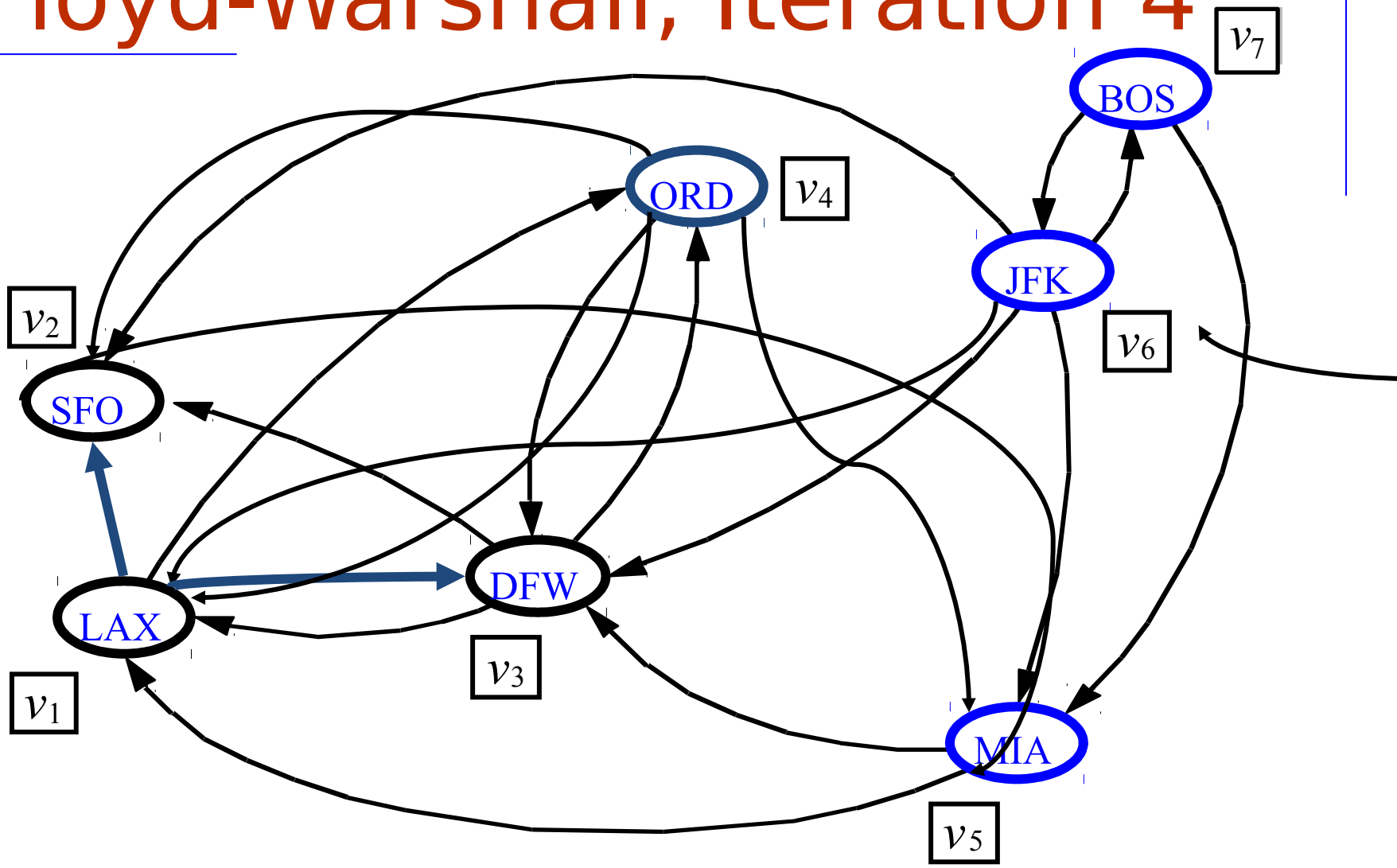
Floyd-Warshall, Iteration 2



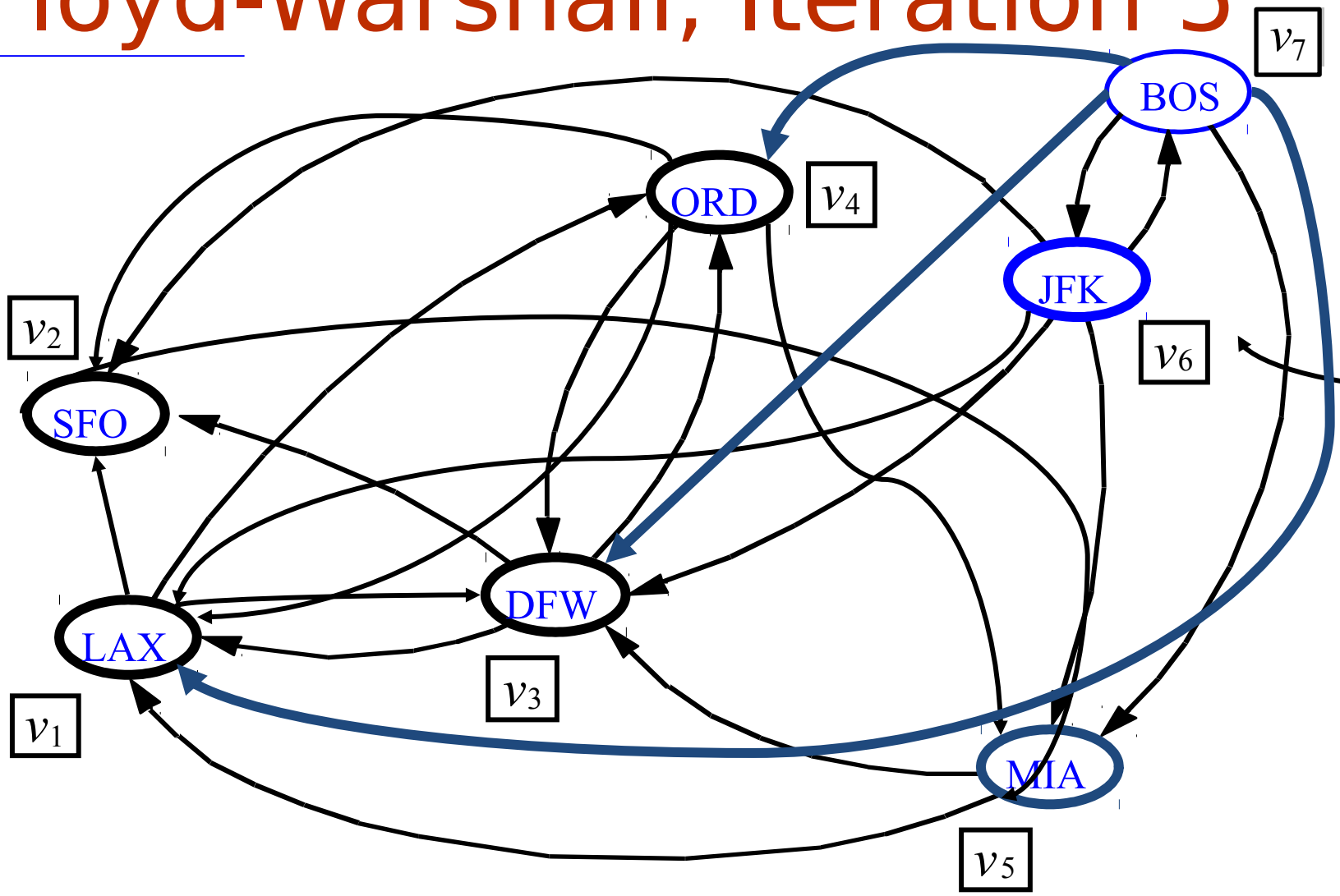
Floyd-Warshall, Iteration 3



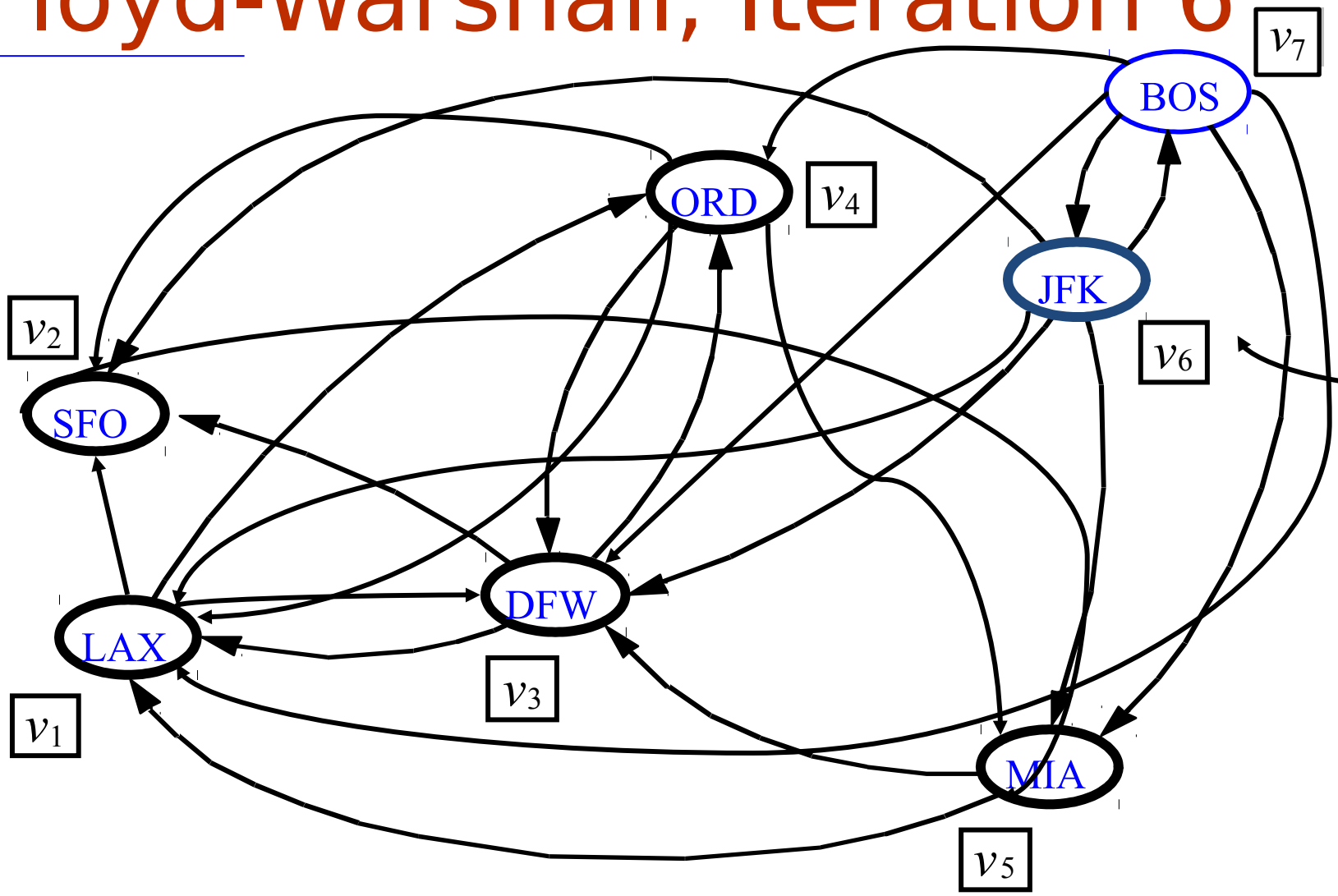
Floyd-Warshall, Iteration 4



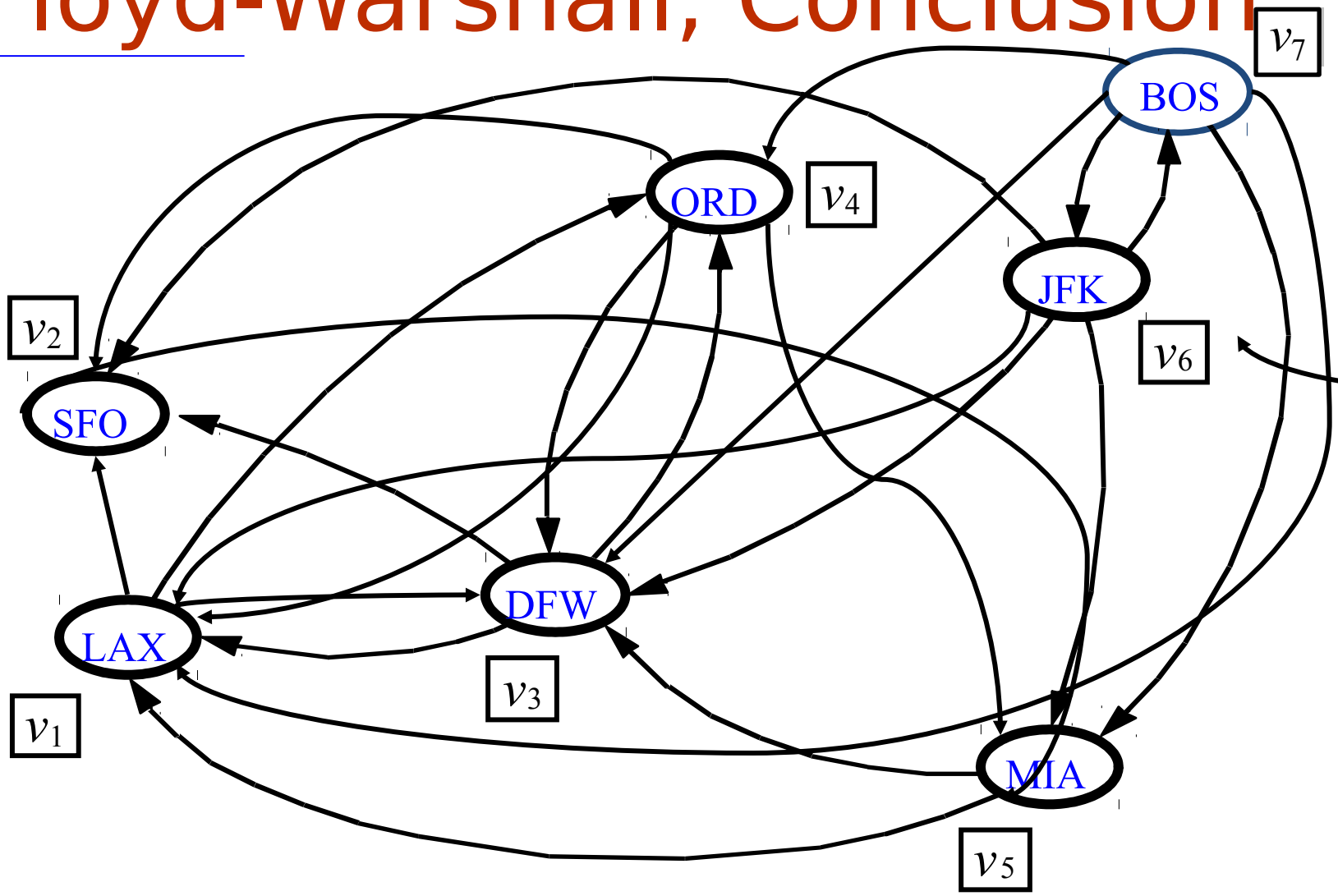
Floyd-Warshall, Iteration 5



Floyd-Warshall, Iteration 6



Floyd-Warshall, Conclusion



DAGs e Ordinamento Topologico

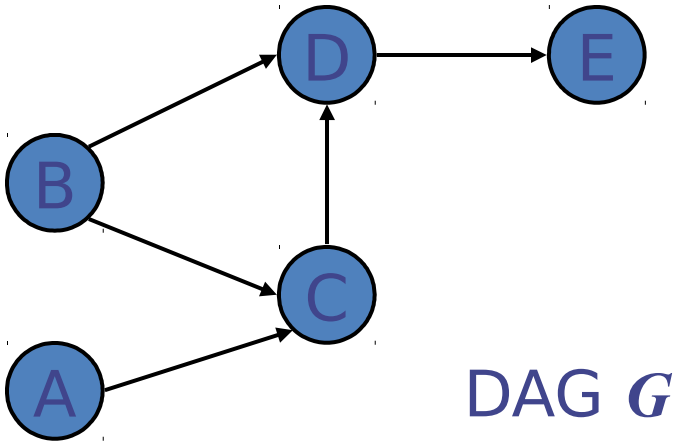
- Grafo diretto aciclico (DAG): digrafo privo di cicli diretti
- Ordinamento topologico: numerazione

  $v_1, \ldots, v_n$

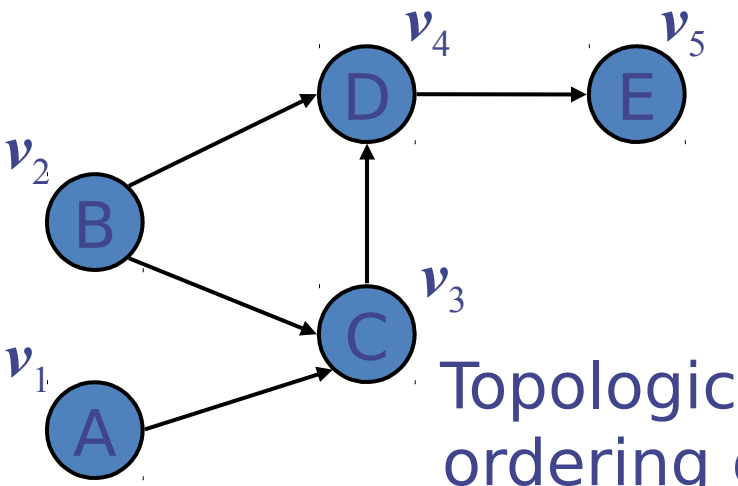
  dei vertici tale che per ogni arco(v_i, v_j), abbiamo $i < j$
- Esempio: se il grafo rappresenta le precedenze tra task, un ordinamento topologico è un sequenziamento dei task che soddisfa i vincoli di precedenza

Teorema

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



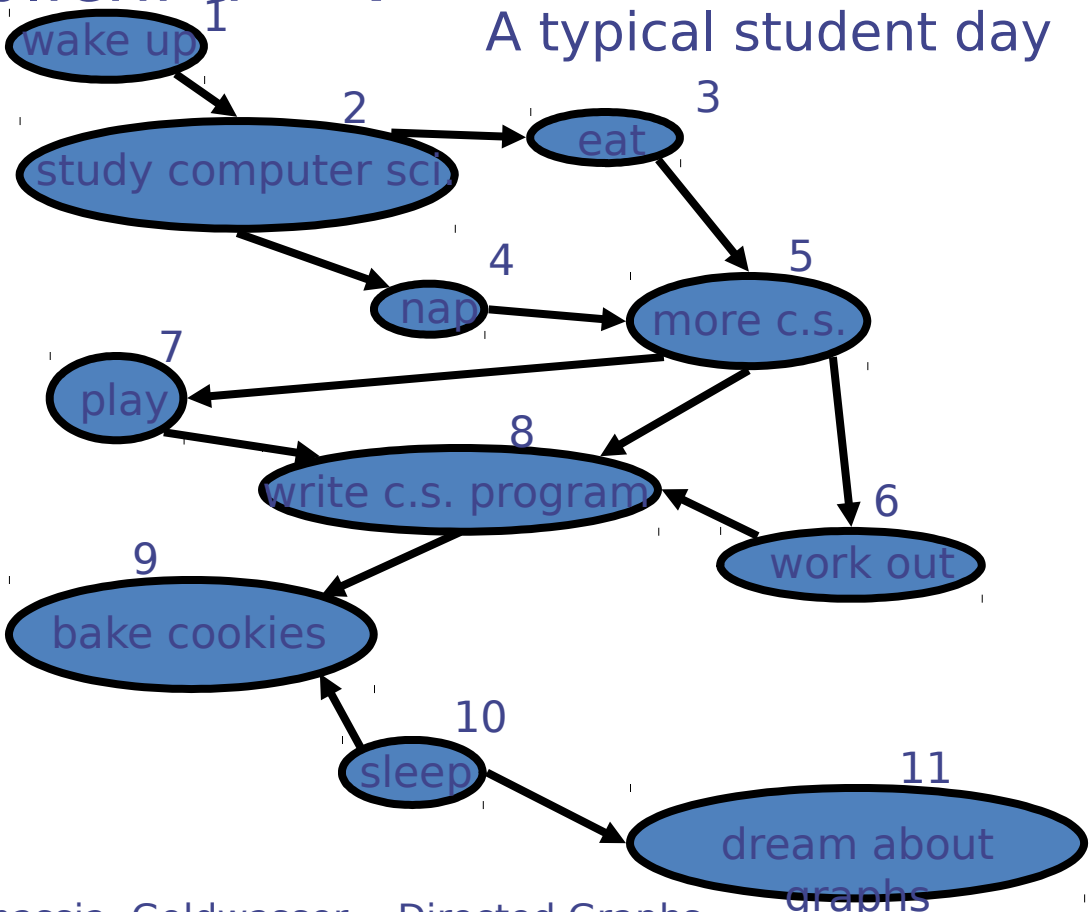
DAG G



Topological ordering of G

Ordinamento topologico

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



Algoritmo per l'ordinamento topologico

- Note: This algorithm is different than the one in the book

```
Algorithm TopologicalSort(G)
    H ← G   // Copia temporanea di G
    n ← G.numVertices()
    while H is not empty do
     Let v be a vertex with no outgoing edges
     Label v ← n
     n ← n − 1
     Remove v from H
```

- Complessità: O(n + m)

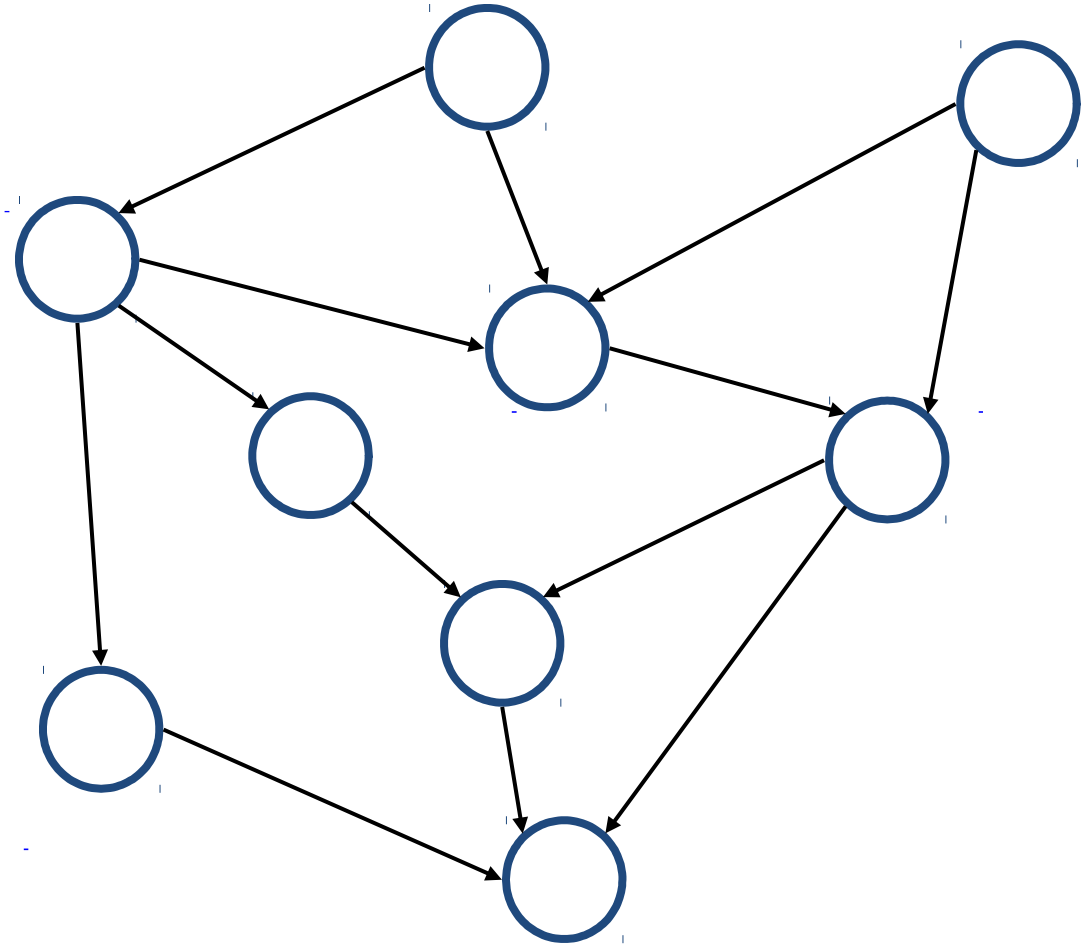
Implementazione con DFS

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

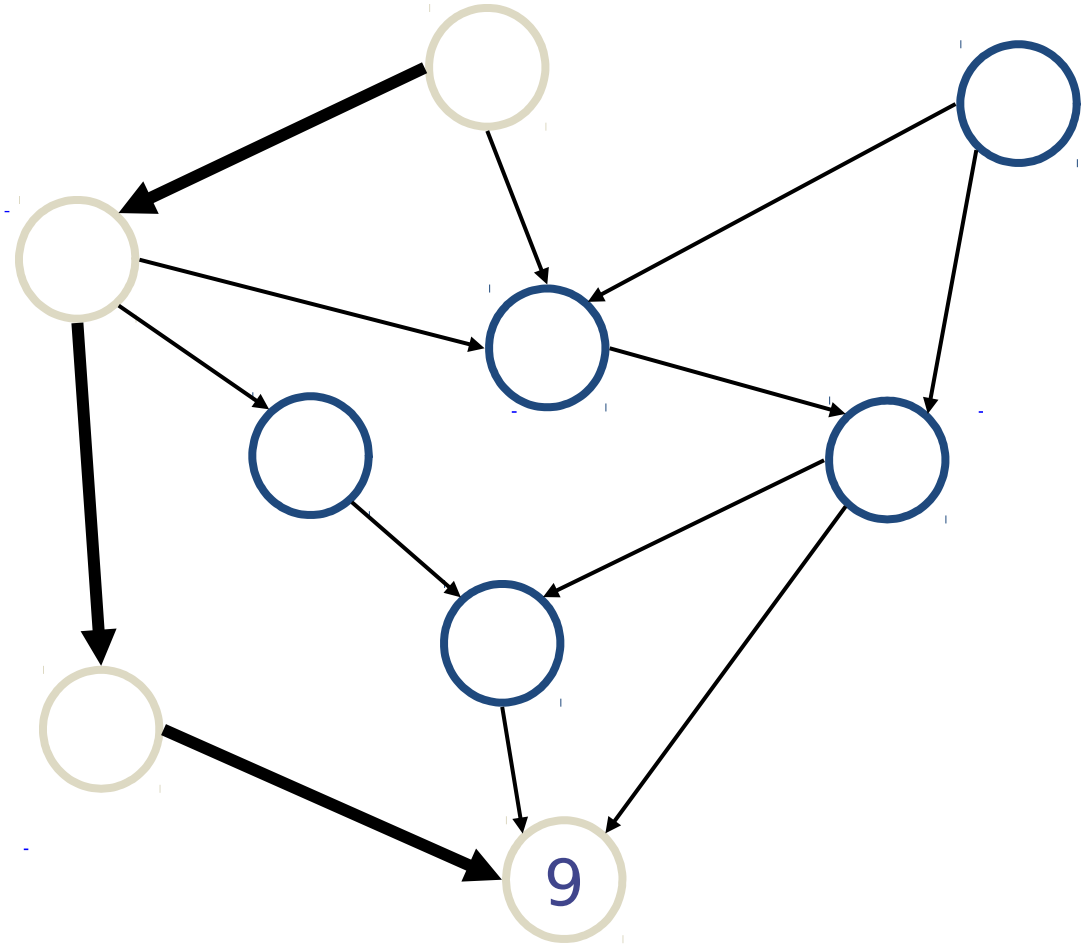
```
Algorithm topologicalDFS(G)
    Input dag G
    Output topological ordering of G
  n ← G.numVertices()
  for all u ∈ G.vertices()
        setLabel(u, UNEXPLORED)
  for all v ∈ G.vertices()
        if getLabel(v) = UNEXPLORED
              topologicalDFS(G, v)
```

```
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, VISITED)
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 is a forward or cross edge }
  Label v with topological number n
  n ← n - 1
```

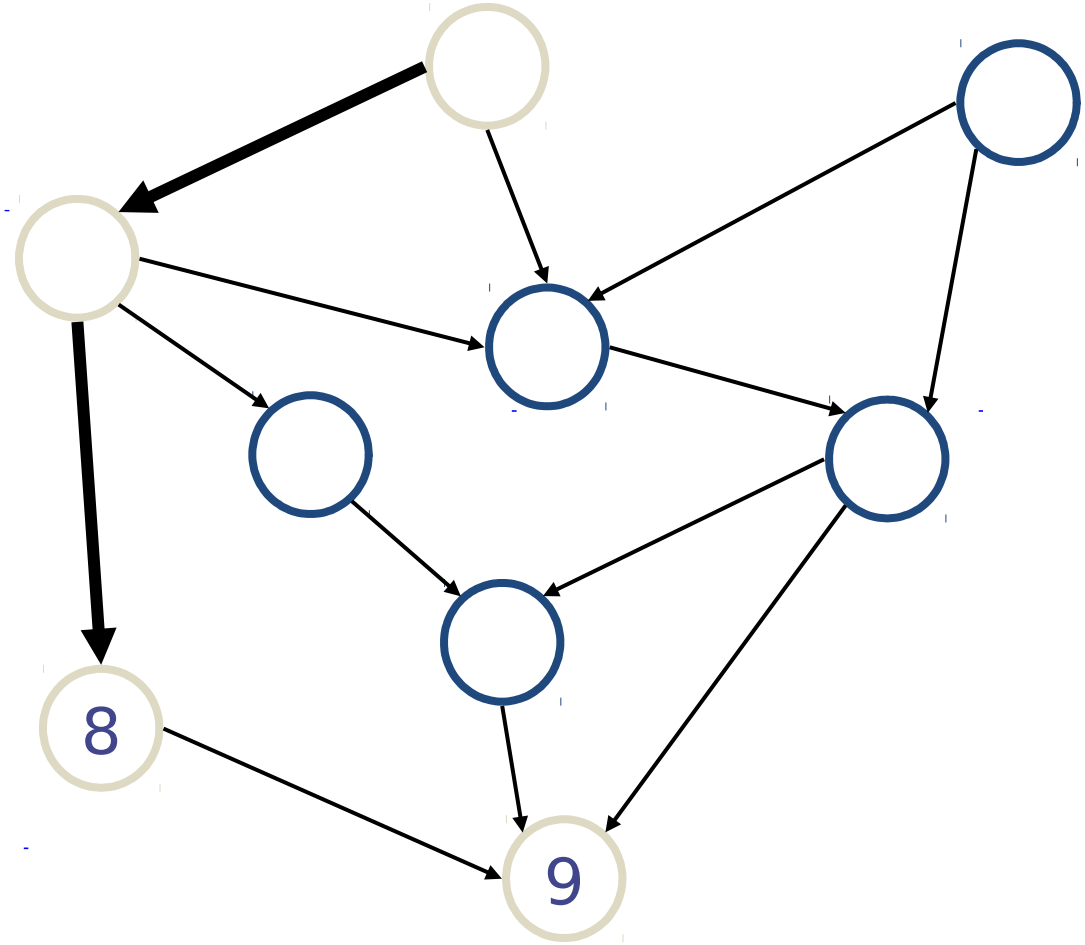
Topological Sorting Example



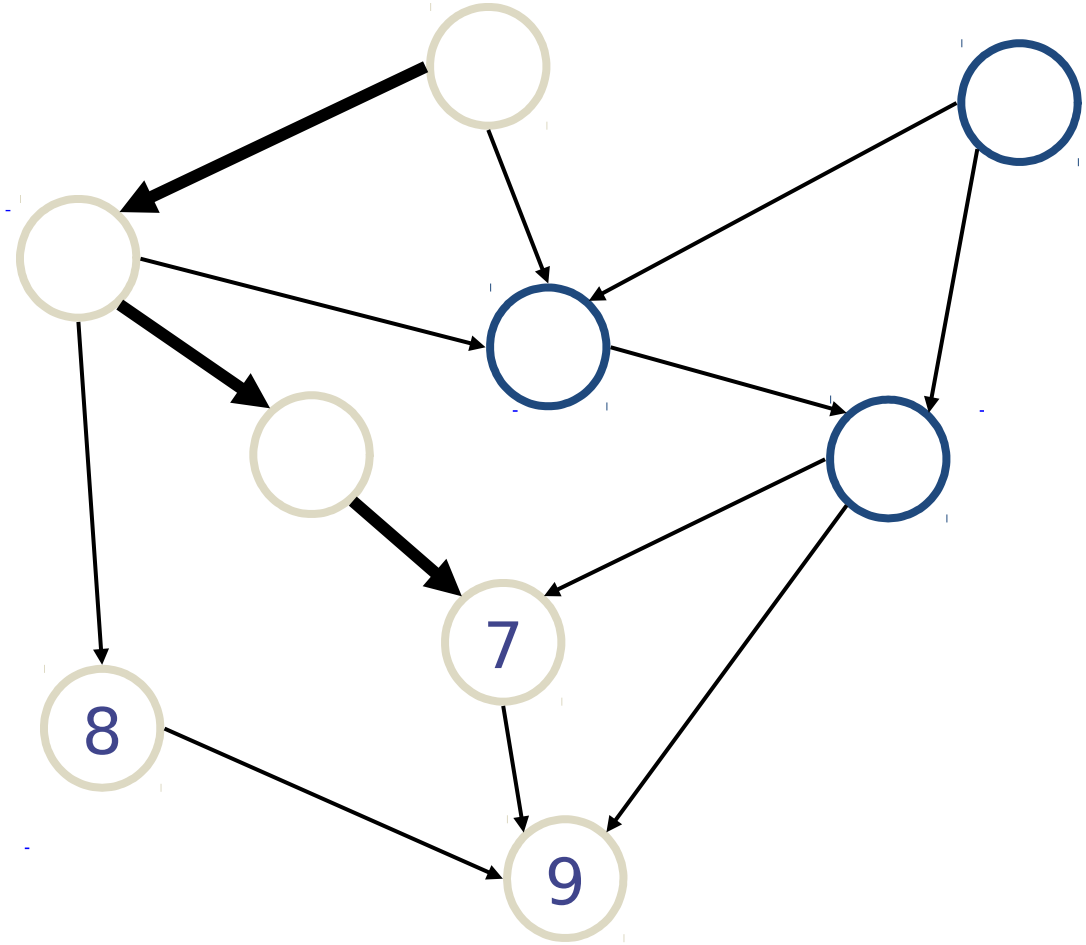
Topological Sorting Example



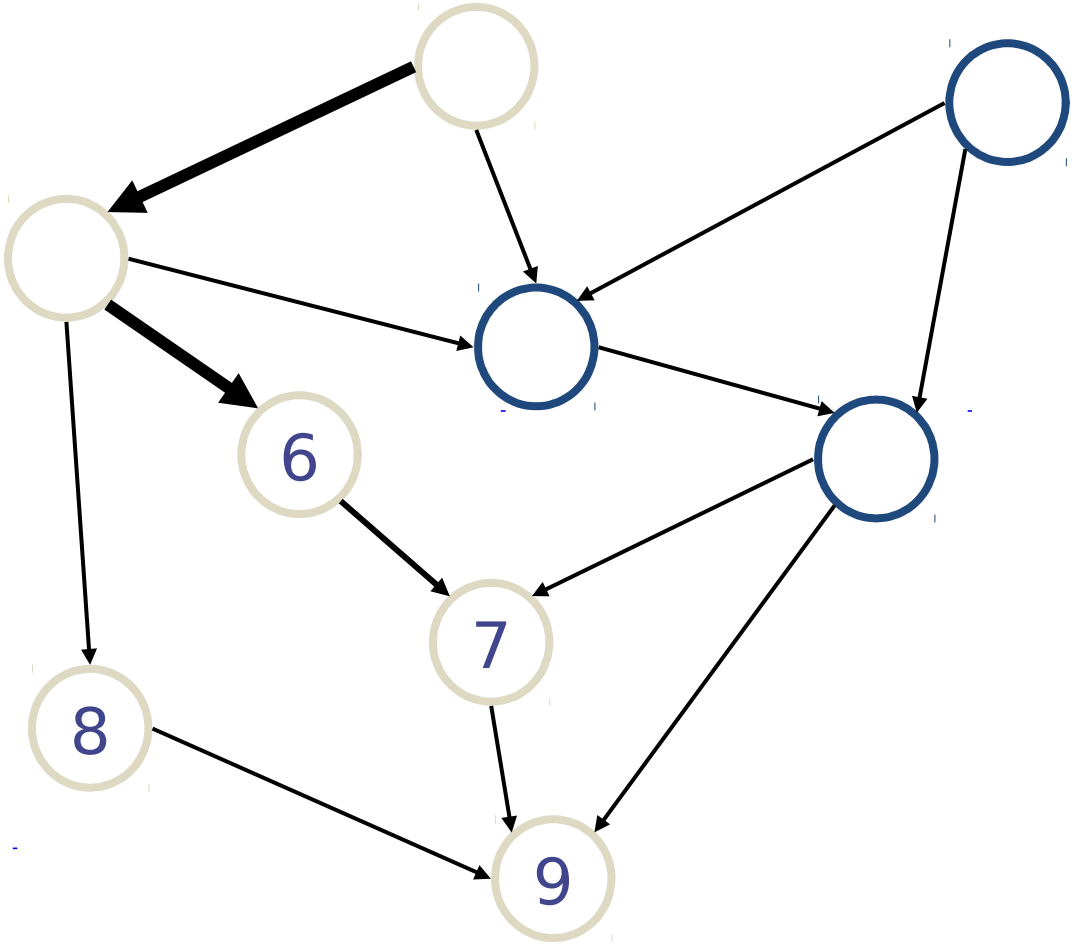
Topological Sorting Example



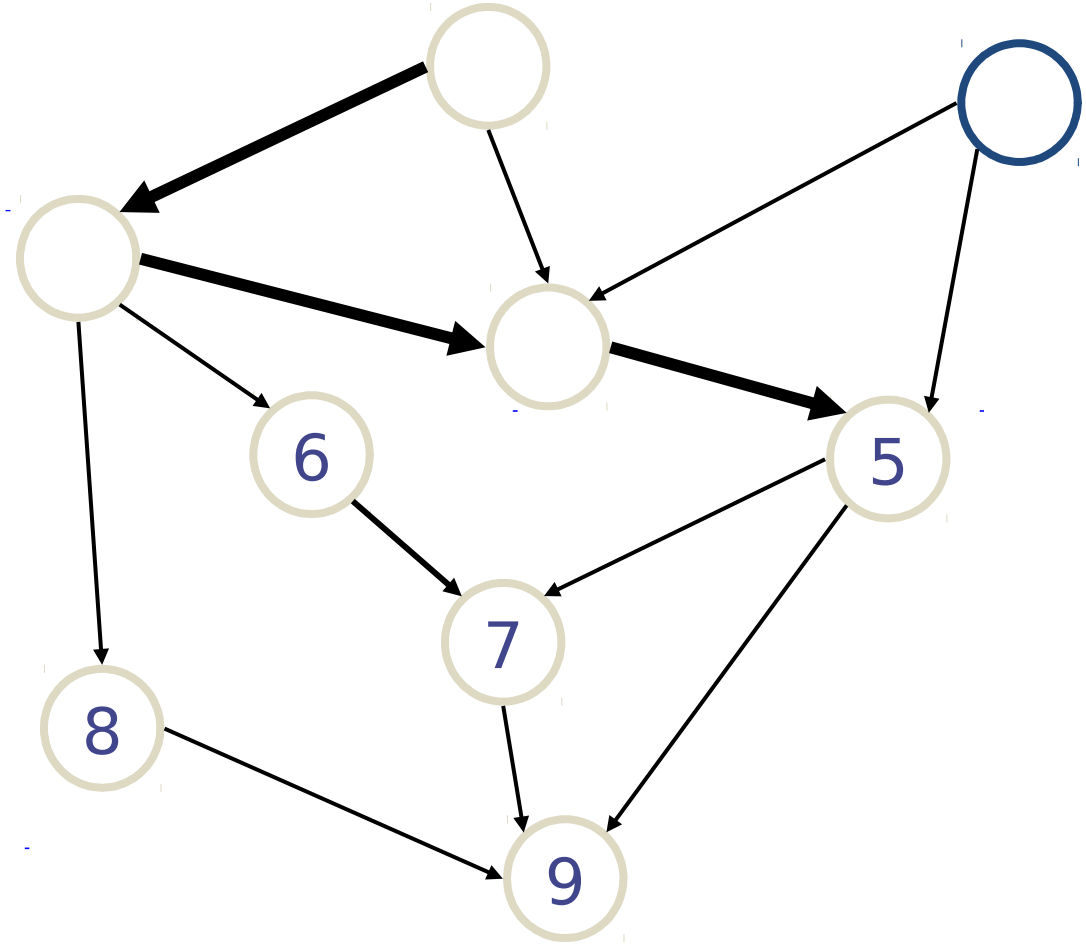
Topological Sorting Example



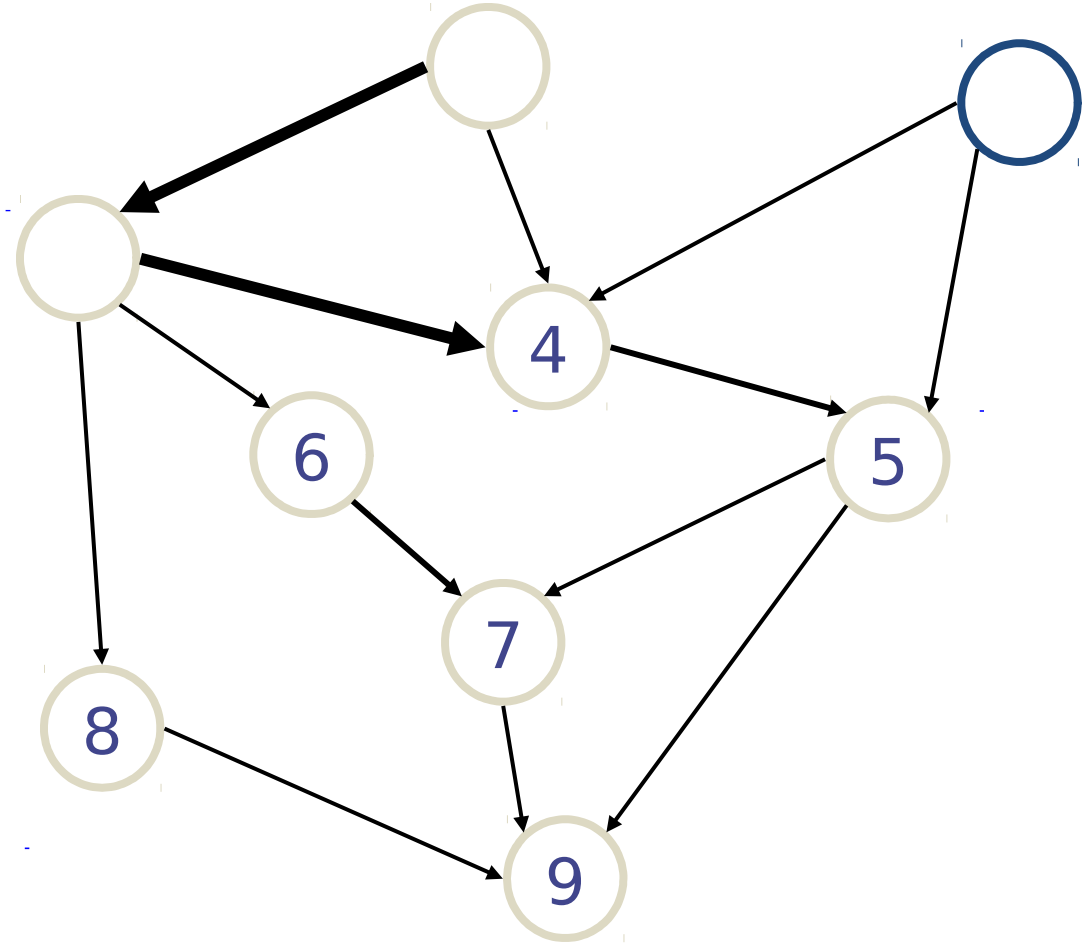
Topological Sorting Example



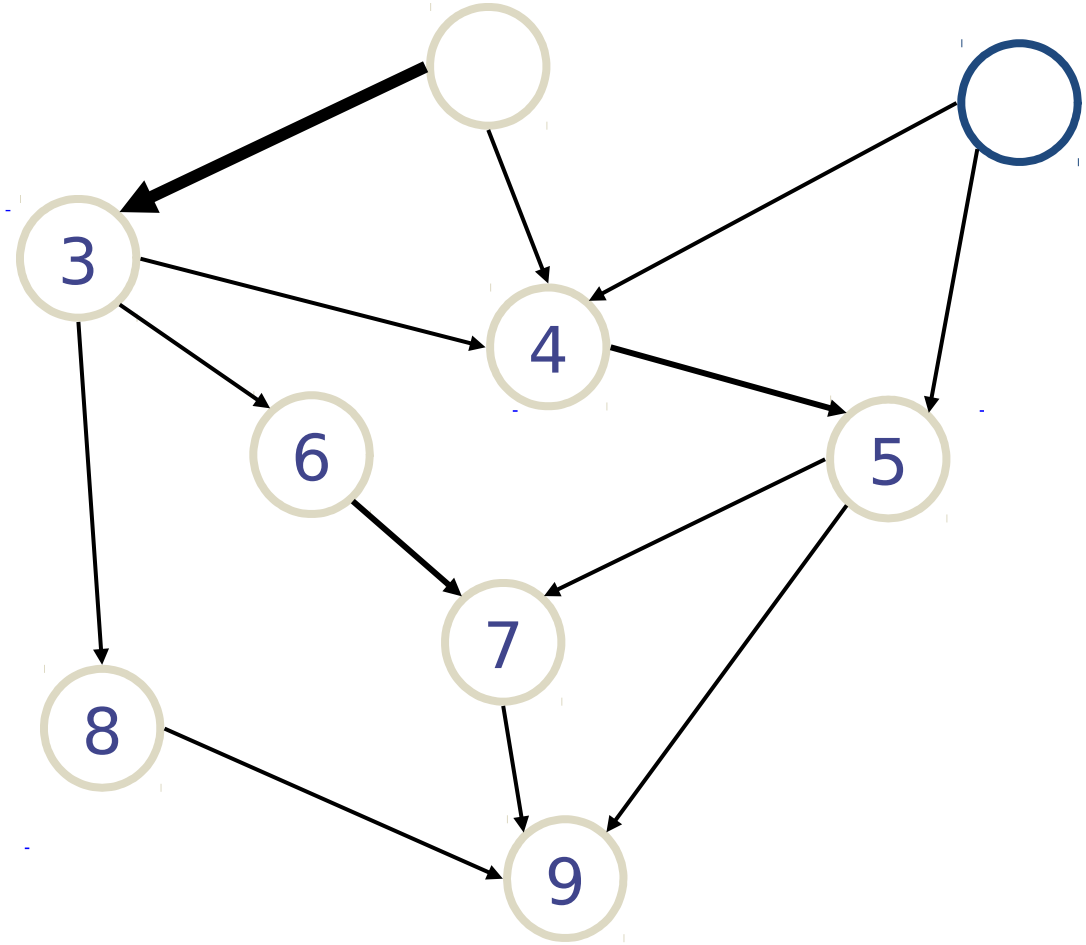
Topological Sorting Example



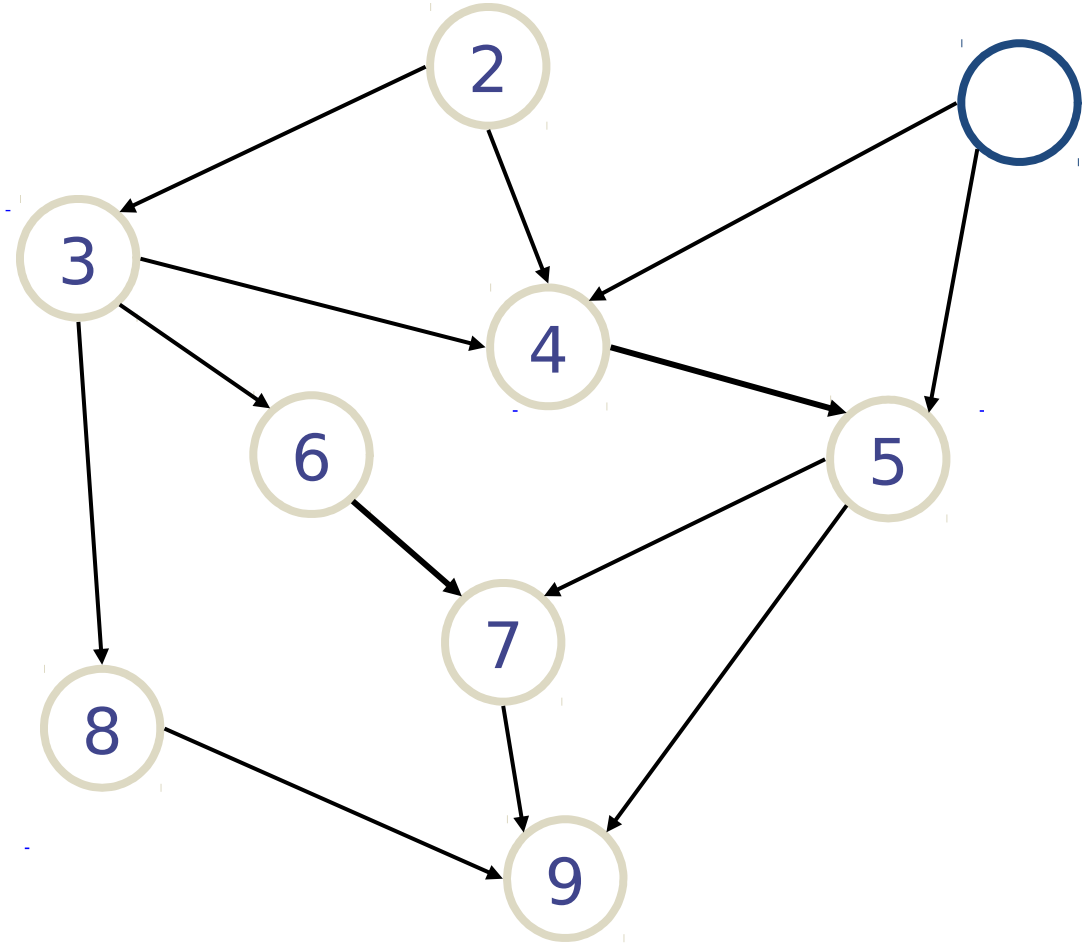
Topological Sorting Example



Topological Sorting Example



Topological Sorting Example



Topological Sorting Example