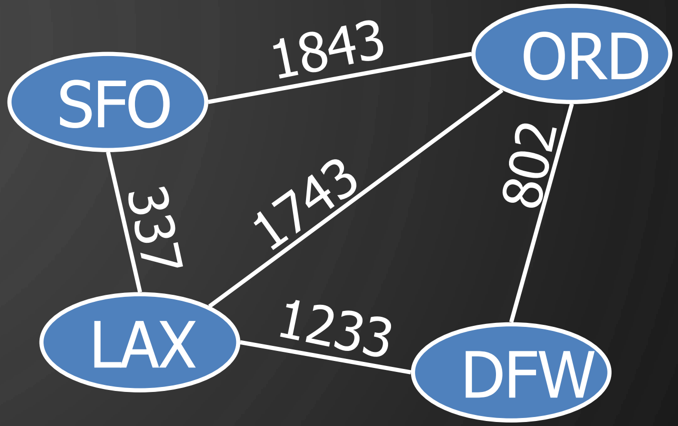
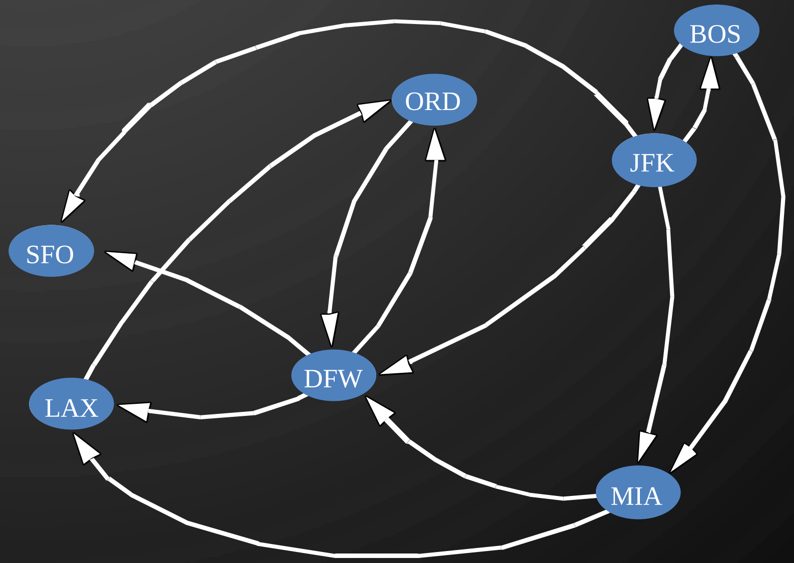
CHAPTER 13
GRAPH ALGORITHMS



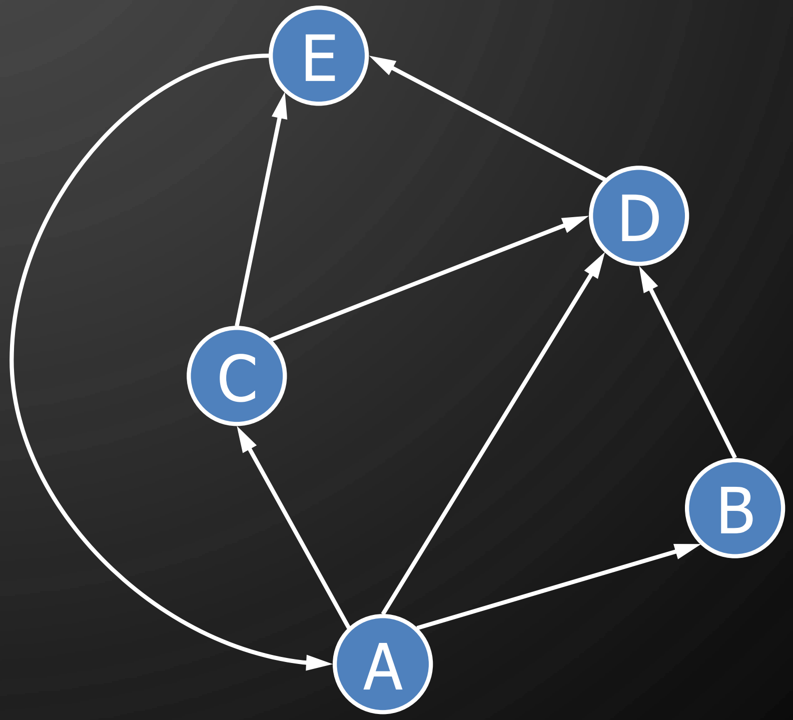
ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH DATA STRUCTURES AND ALGORITHMS IN C++, GOODRICH, TAMASSIA AND MOUNT (WILEY 2004) AND SLIDES FROM NANCY M. AMATO

DIRECTED GRAPHS



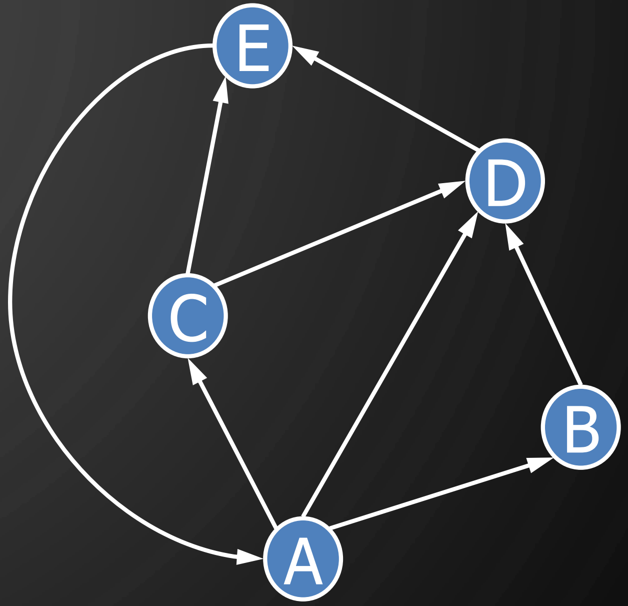
DIGRAPHS

- A digraph is a graph whose edges are all directed
  - Short for “directed graph”
- Applications
  - one-way streets
  - flights
  - task scheduling



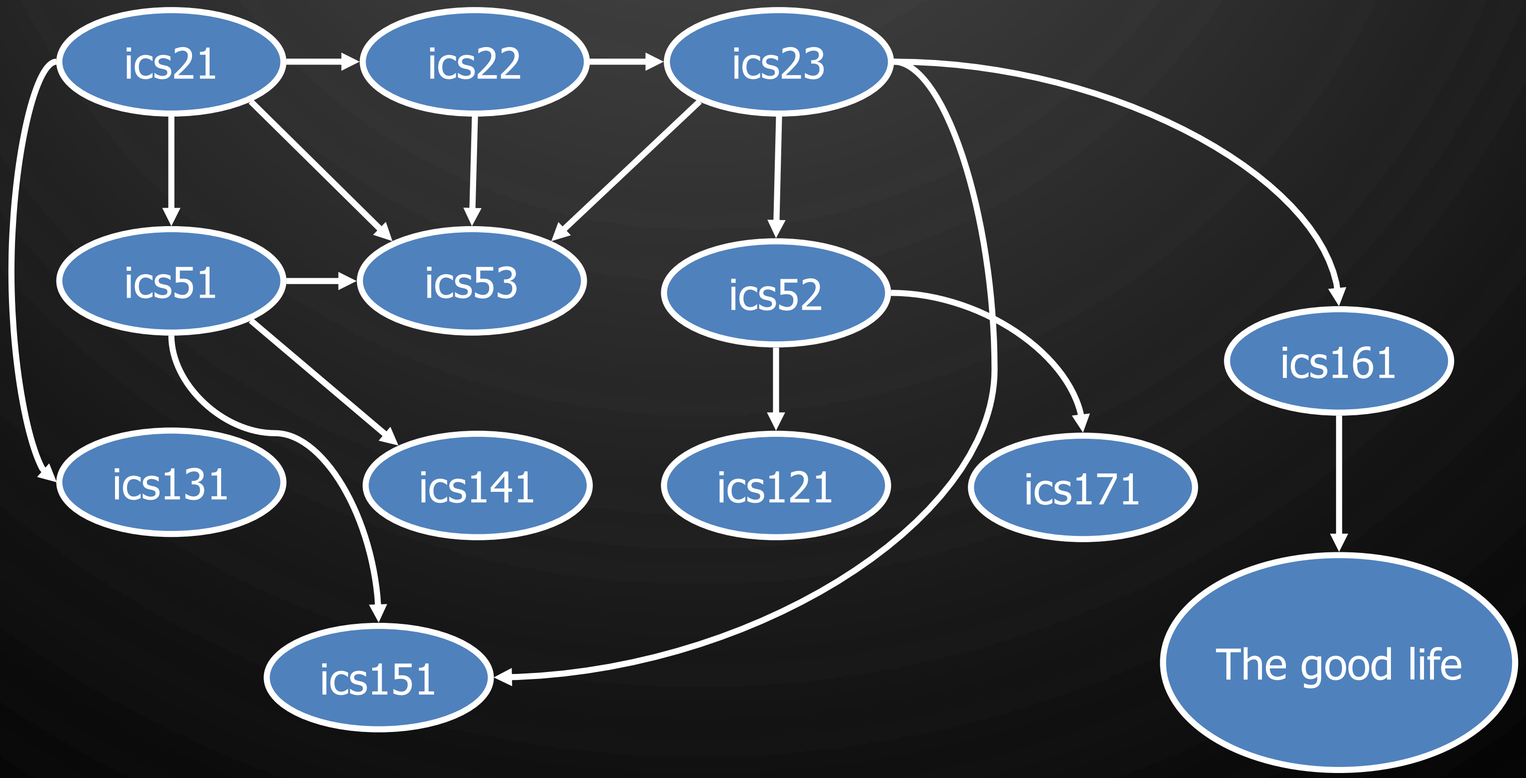
DIGRAPH PROPERTIES

- A graph $G = (V, E)$ such that
  - Each edge goes in one direction:
  - Edge (a, b) goes from a to b, but not b to a
- If G is simple, $m < n(n-1)$
- If we keep in-edges and out-edges in separate adjacency lists, we can perform listing of incoming edges and outgoing edges in time proportional to their size



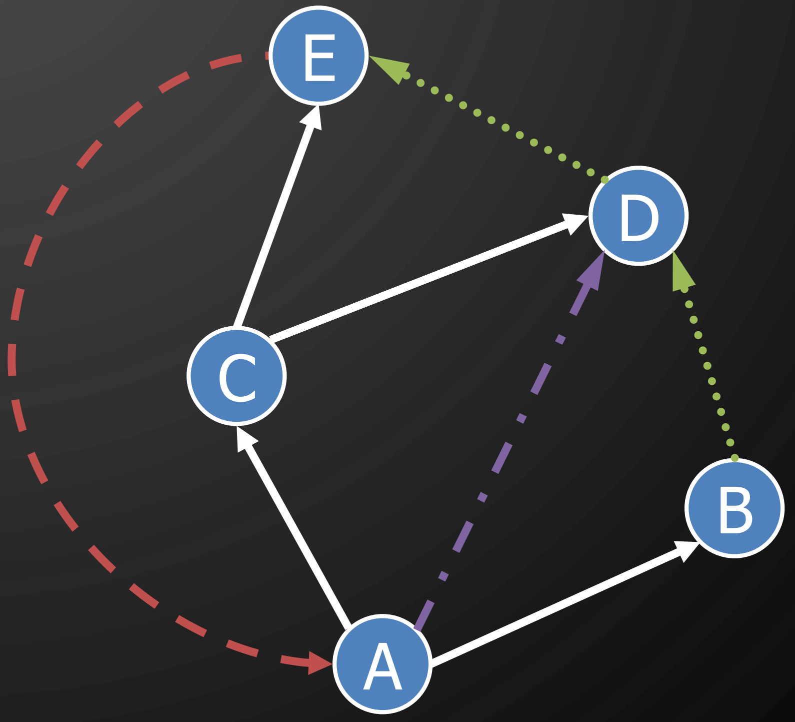
DIGRAPH APPLICATION

- Scheduling: edge (a, b) means task a must be completed before b can be started



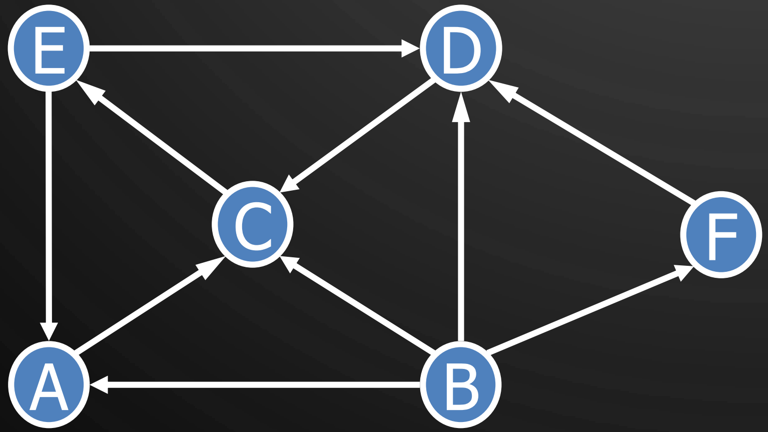
DIRECTED DFS

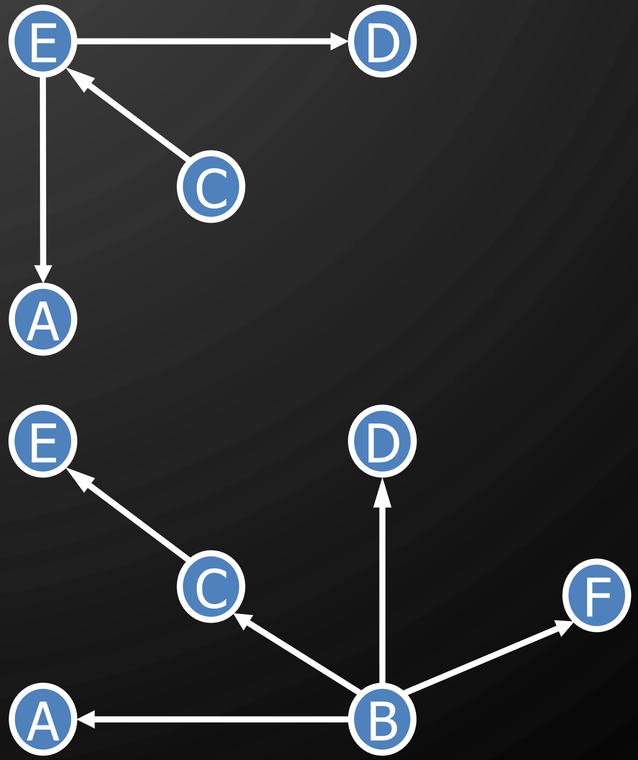
- We can specialize the traversal algorithms (DFS and BFS) to digraphs by traversing edges only along their direction
- In the directed DFS algorithm, we have four types of edges
  - discovery edges
  - back edges
  - forward edges
  - cross edges
- A directed DFS starting at a vertex s determines the vertices reachable from s



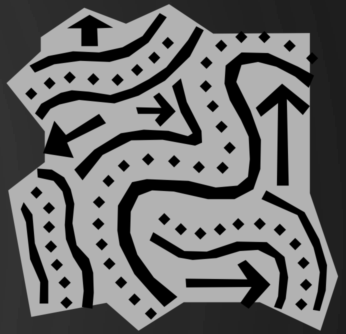
REACHABILITY

- DFS tree rooted at v: vertices reachable from v via directed paths

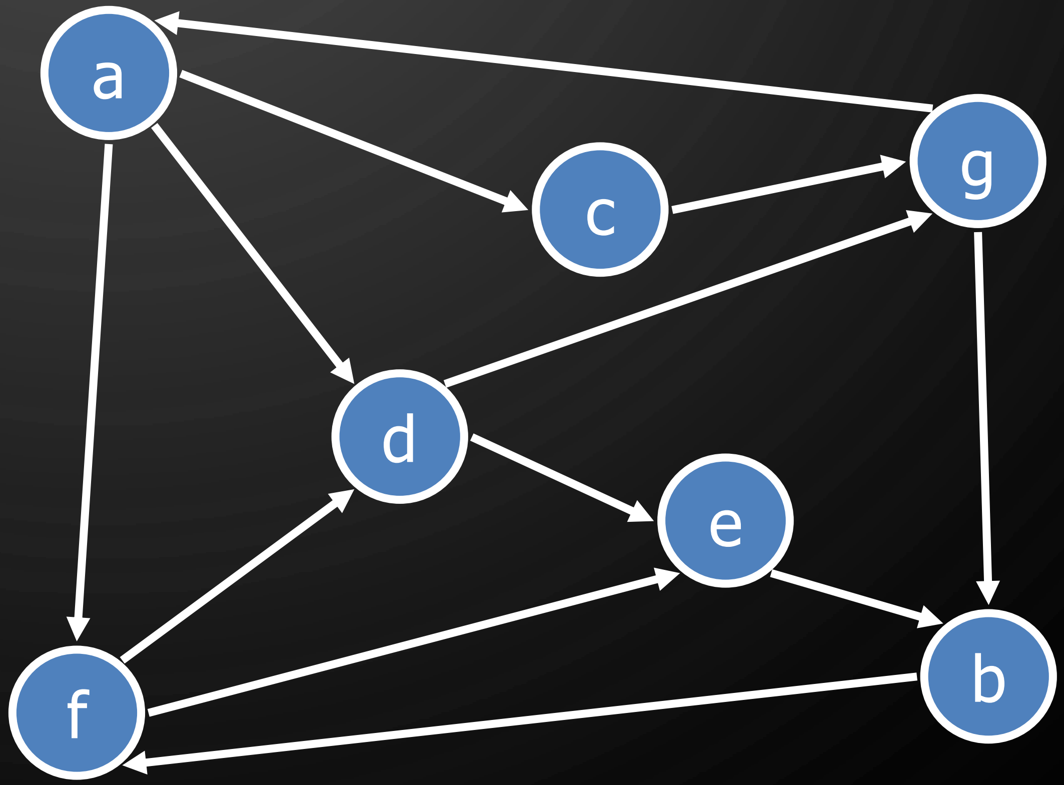




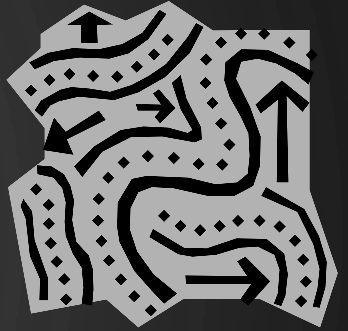
STRONG CONNECTIVITY



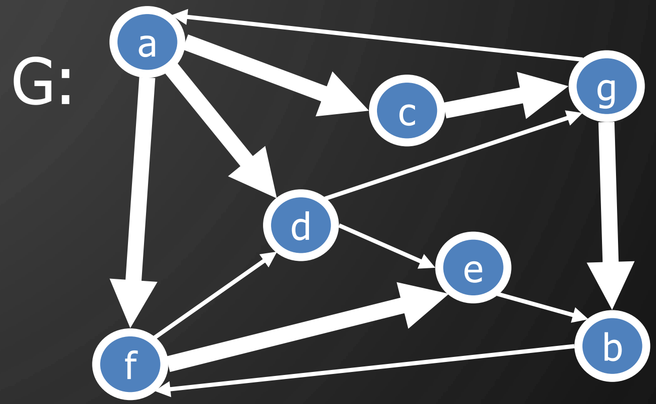
- Each vertex can reach all other vertices

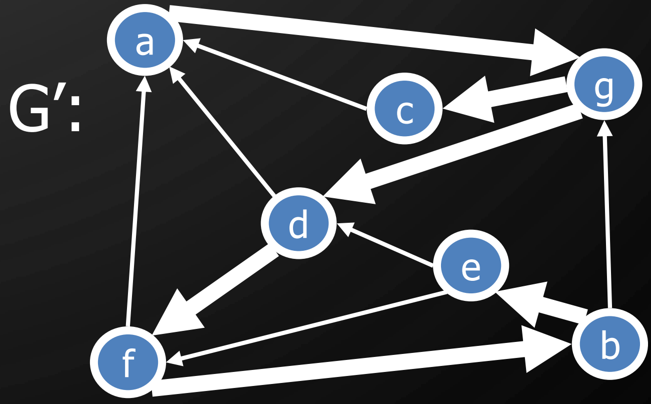


STRONG CONNECTIVITY ALGORITHM



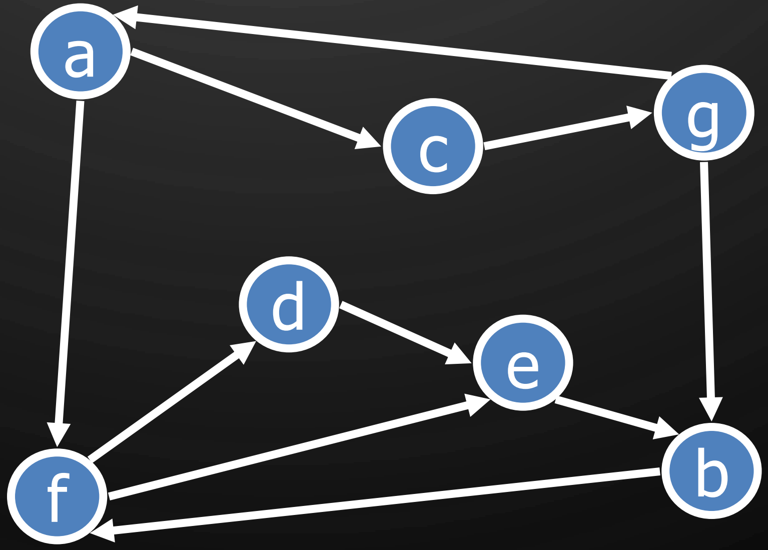
- Pick a vertex v in G
- Perform a DFS from v in G
  - If there's a w not visited, print "no"
- Let G' be G with edges reversed
- Perform a DFS from v in G'
  - If there's a w not visited, print "no"
  - Else, print "yes"
- Running time: $O(n + m)$





STRONGLY CONNECTED COMPONENTS

- Maximal subgraphs such that each vertex can reach all other vertices in the subgraph
- Can also be done in $O(n + m)$ time using DFS, but is more complicated (similar to biconnectivity).

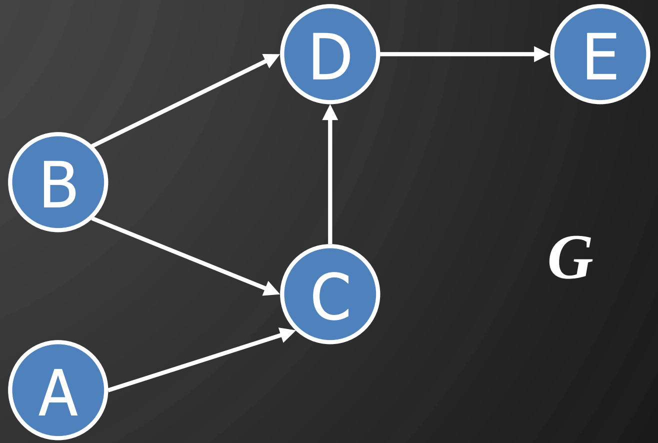


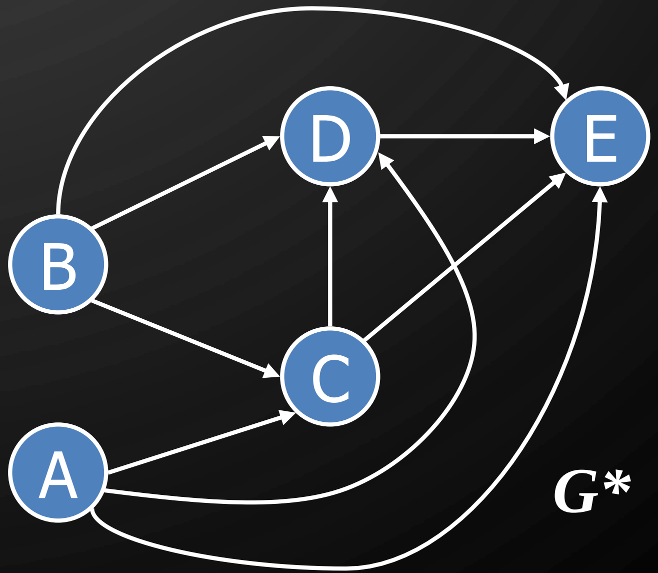
$\{a, c, g\}$

$\{f, d, e, b\}$

TRANSITIVE CLOSURE

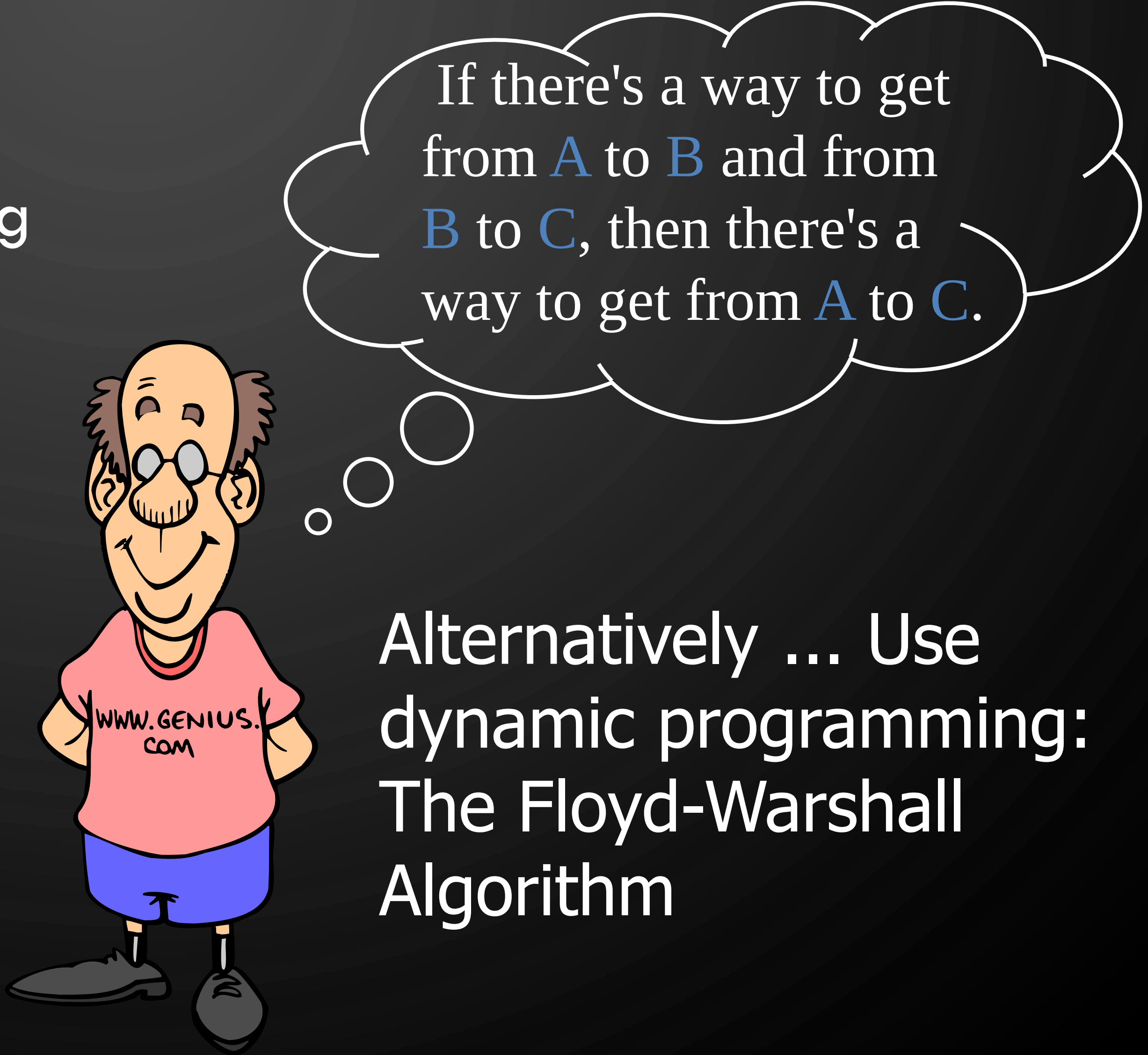
- Given a digraph G, the transitive closure of G is the digraph G^* such that
    - $G*$ has the same vertices as G
    - if G has a directed path from u to v $(u \quad v)$, G^* has a directed edge from u to v
- The transitive closure provides reachability information about a digraph





COMPUTING THE TRANSITIVE CLOSURE

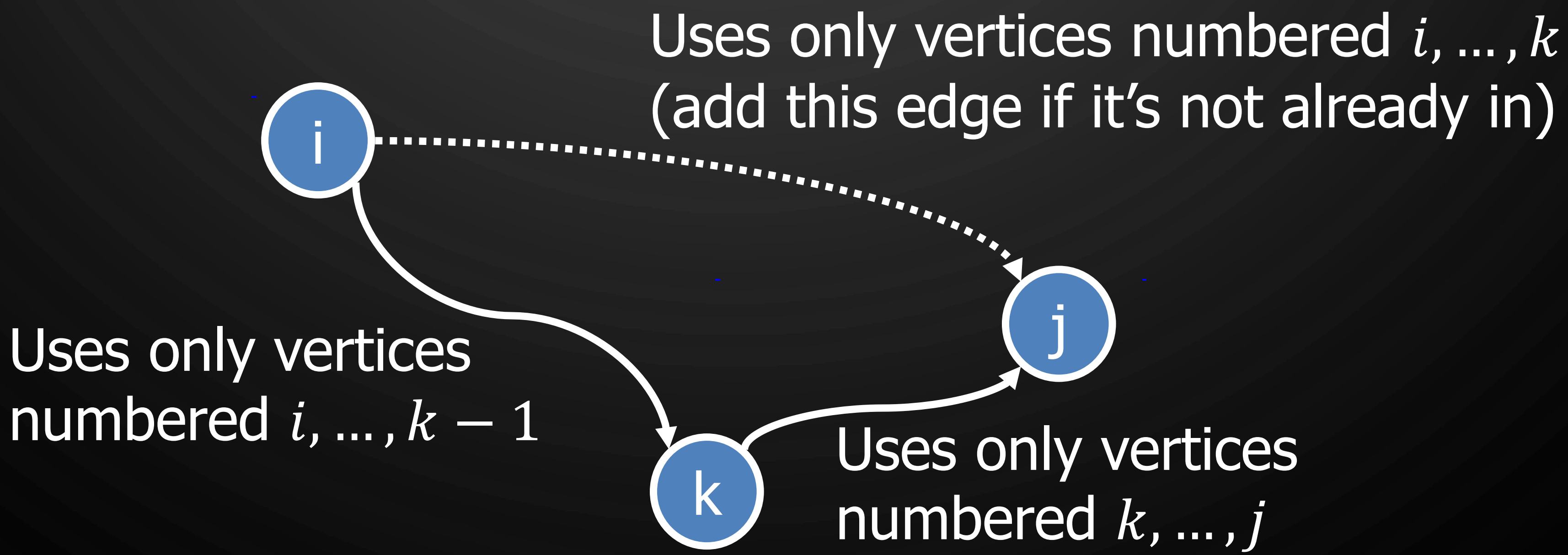
- We can perform DFS starting at each vertex
  - $O(n(n+m))$


If there's a way to get from A to B and from B to C, then there's a way to get from A to C.
WWW.GENIUS.COM

Alternatively ... Use dynamic programming: The Floyd-Warshall Algorithm

FLOYD-WARSHALL TRANSITIVE CLOSURE

- Idea #1: Number the vertices $1, 2, \ldots, n$.
- Idea #2: Consider paths that use only vertices numbered $1, 2, \ldots, k$, as intermediate vertices:


Uses only vertices numbered $i, \ldots, k$
(add this edge if it's not already in)
i
j
k
Uses only vertices numbered $i, \ldots, k-1$
Uses only vertices numbered $k, \ldots, j$

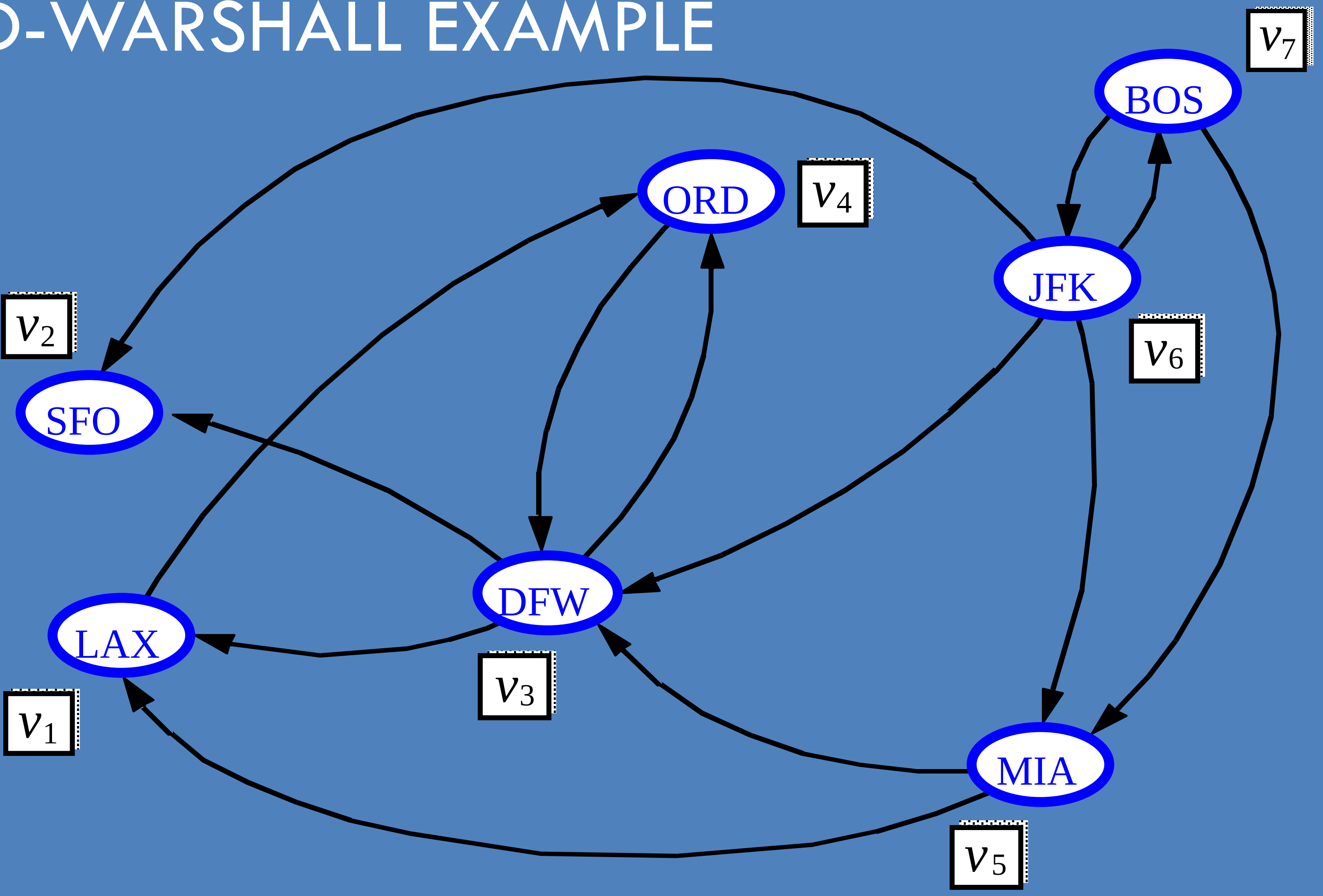
FLOYD-WARSHALL'S ALGORITHM

- Number vertices $v_1, \dots, v_n$
- Compute digraphs $G_0, \dots, G_n$
  - $G_0 \leftarrow G$
  - G_k has directed edge (v_i, v_j) if G has a directed path from v_i to v_j
- We have that $G_n = G^*$
- In phase k, digraph G_k is computed from G_{k-1}
- Running time: $O(n^3)$, assuming $G.\text{areAdjacent}(v_i, v_j)$ is $O(1)$ (e.g., adjacency matrix)

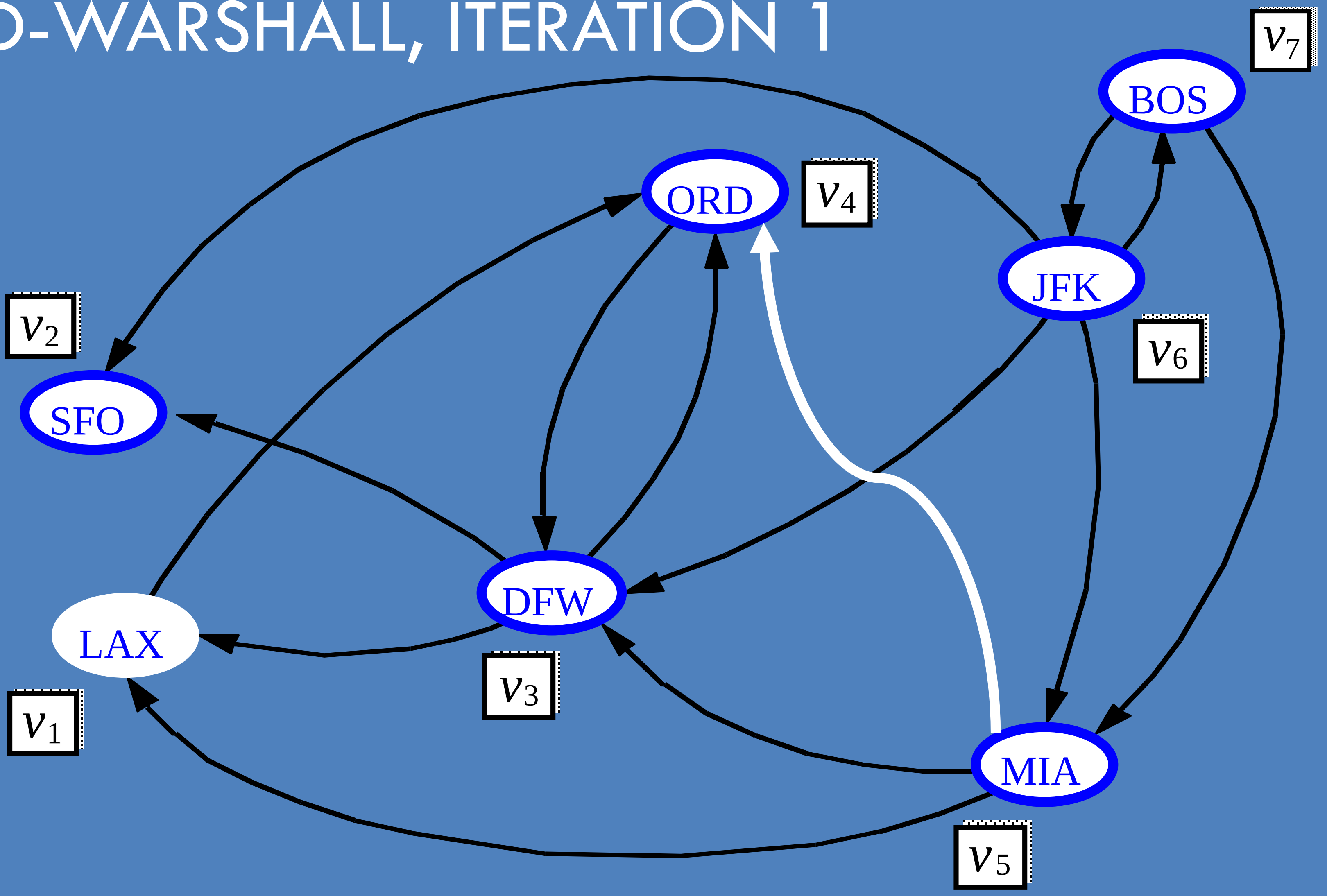
Algorithm FloydWarshall(G)
Input: Digraph G
Output: Transitive Closure G^* of G

```
1. Name each vertex $v \in G.\text{vertices}()$ with $i = 1 \dots n$
2. $G_0 \leftarrow G$
3. for $k \leftarrow 1 \dots n$ do
4.   $G_k \leftarrow G_{k-1}$
5.   for $i \leftarrow 1 \dots n \mid i \neq k$ do
6.     for $j \leftarrow 1 \dots n \mid j \neq i, k$ do
7.       if $G_{k-1}.\text{areAdjacent}(v_i, v_k) \wedge$
            $G_{k-1}.\text{areAdjacent}(v_k, v_j) \wedge$
            $\neg G_k.\text{areAdjacent}(v_i, v_j)$ then
8.           $G_k.\text{insertDirectedEdge}(v_i, v_j)$
9. return $G_n$
```

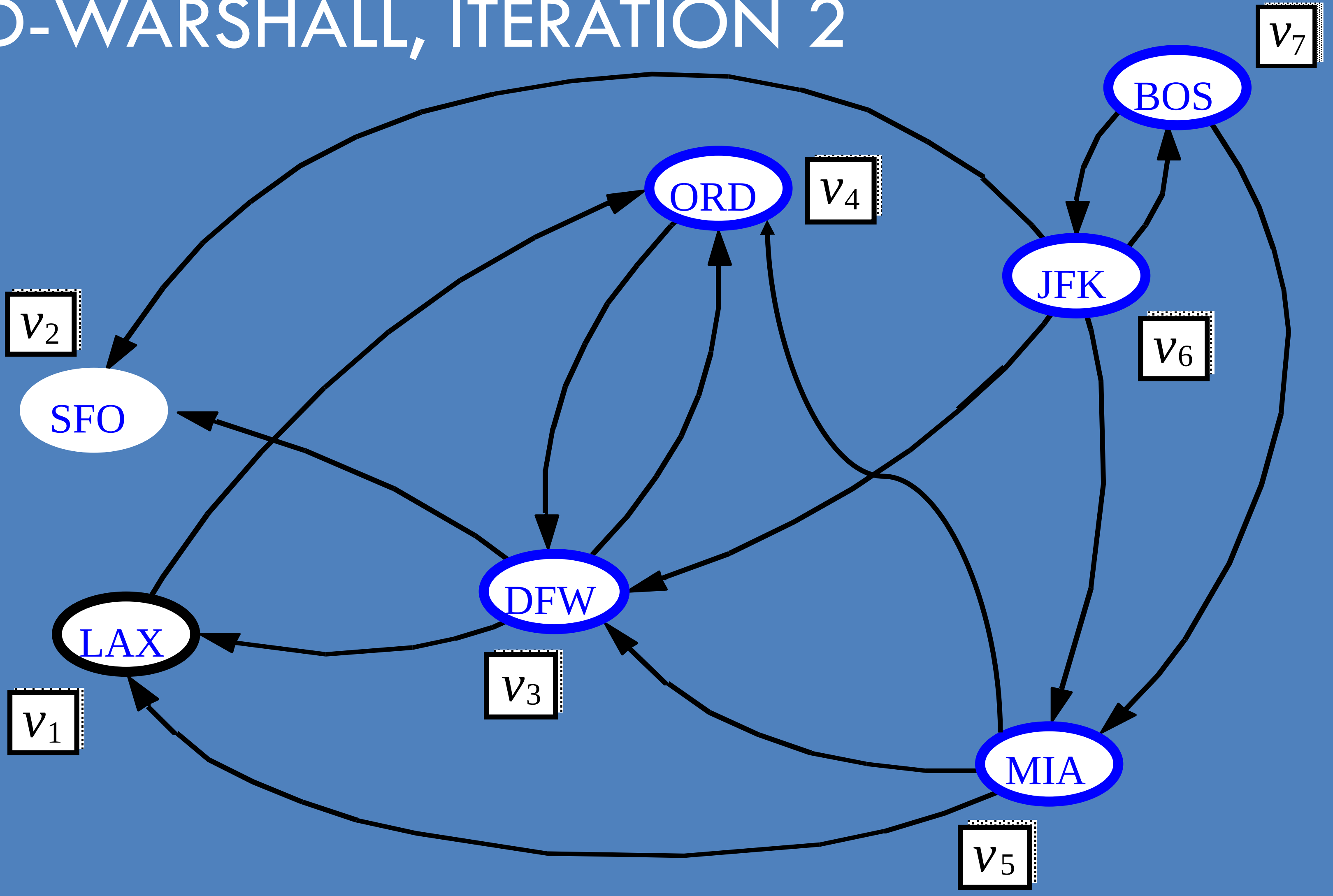
FLOYD-WARSHALL EXAMPLE



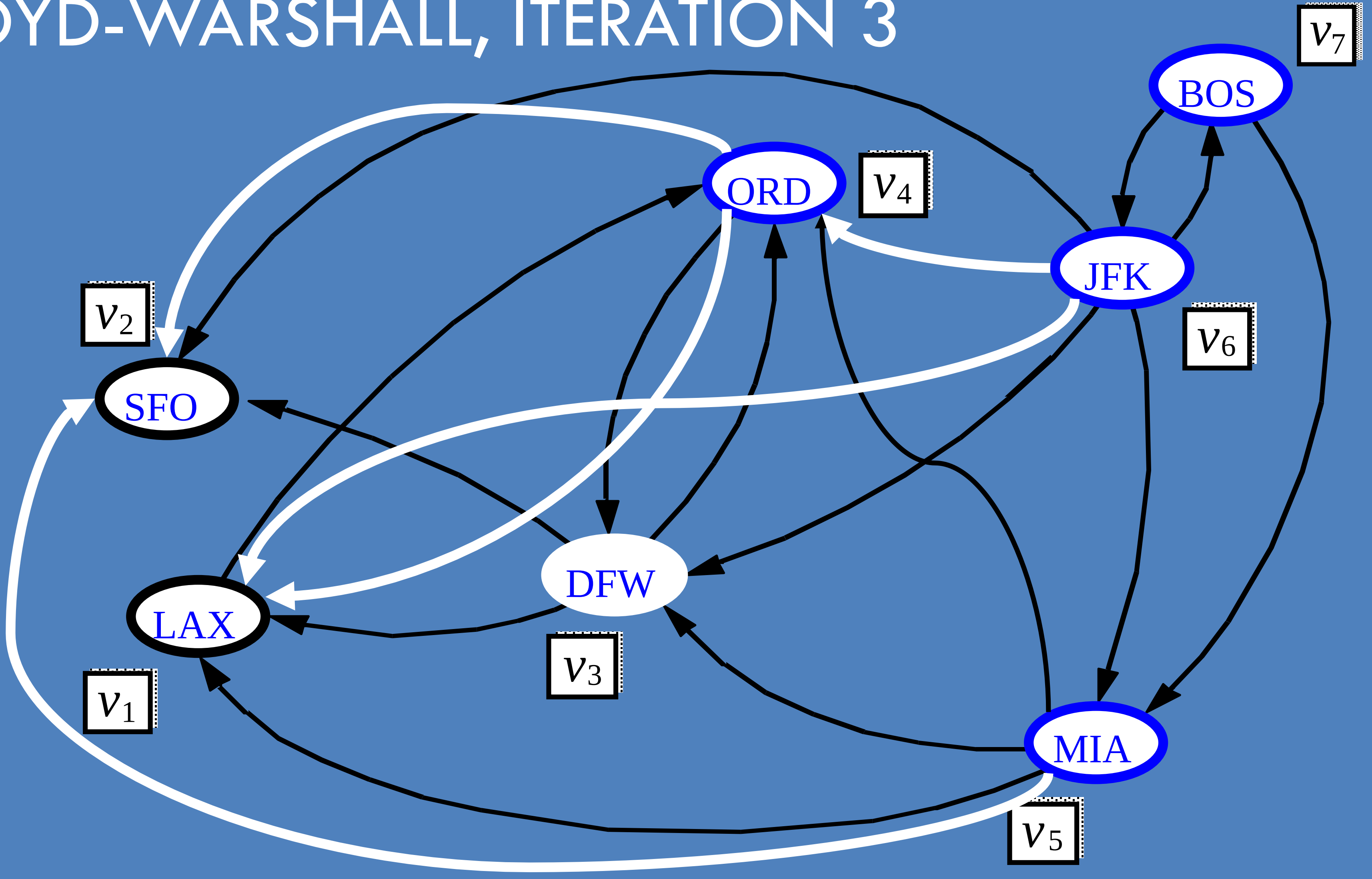
FLOYD-WARSHALL, ITERATION 1



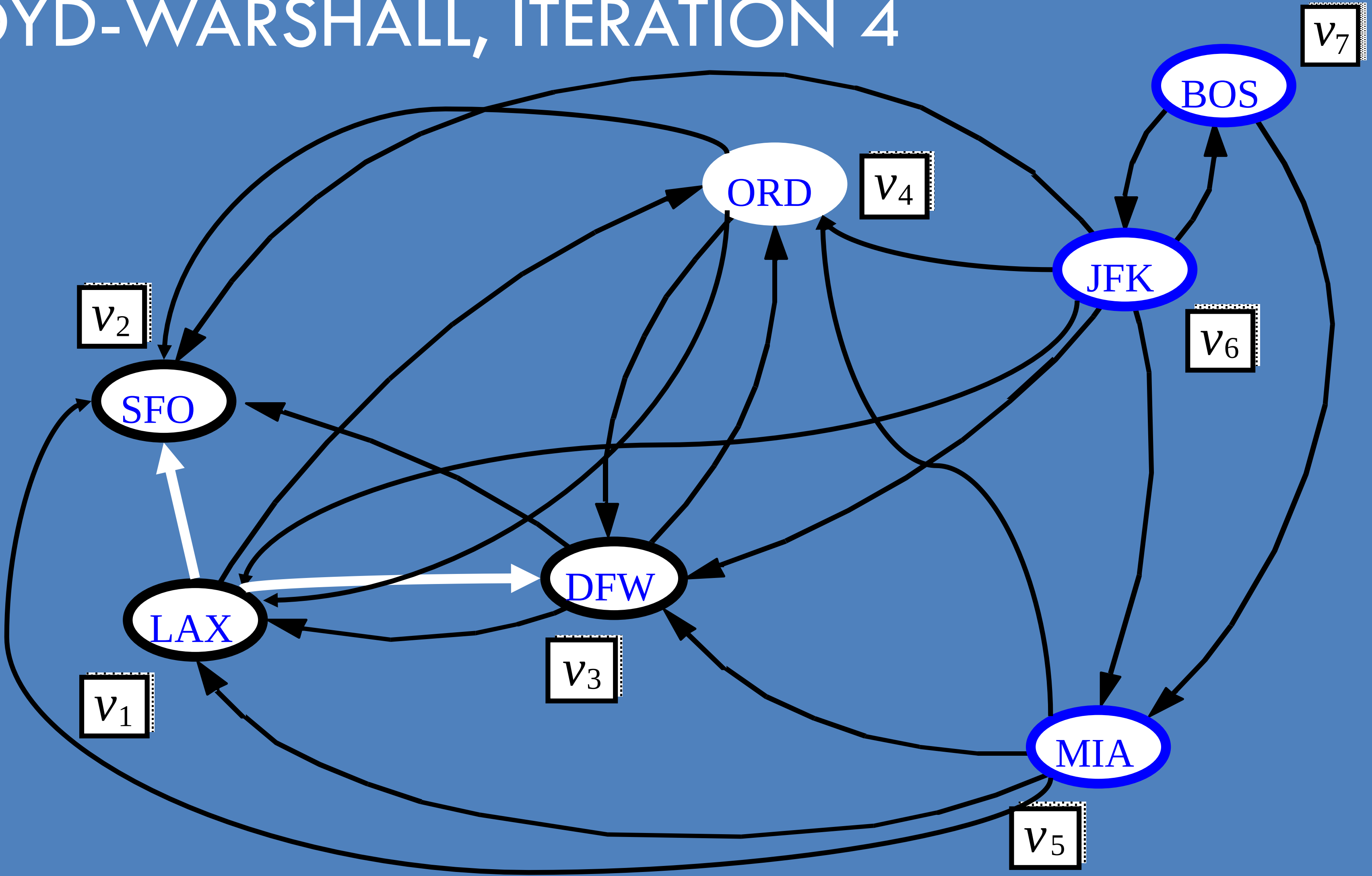
FLOYD-WARSHALL, ITERATION 2



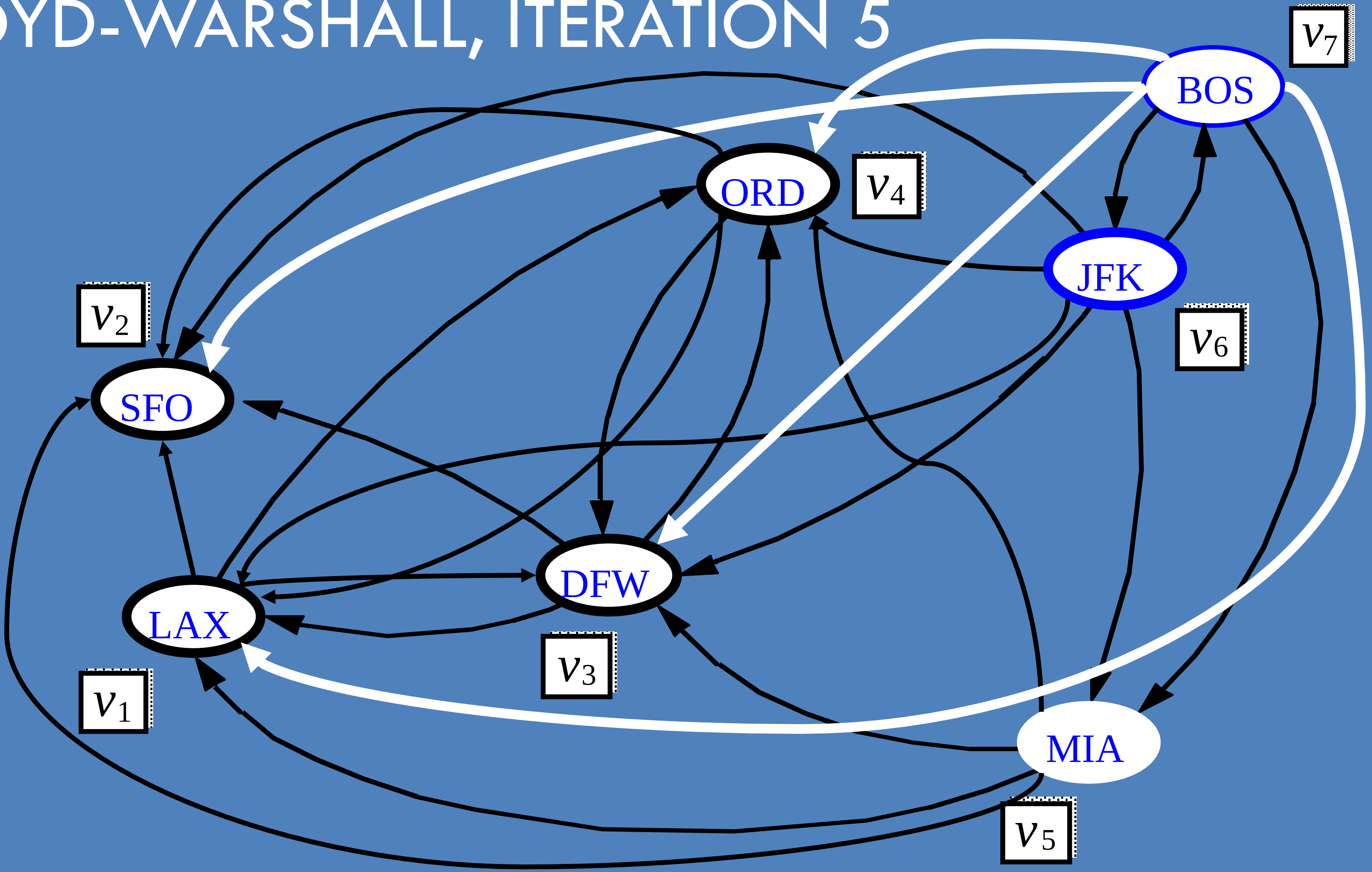
FLOYD-WARSHALL, ITERATION 3



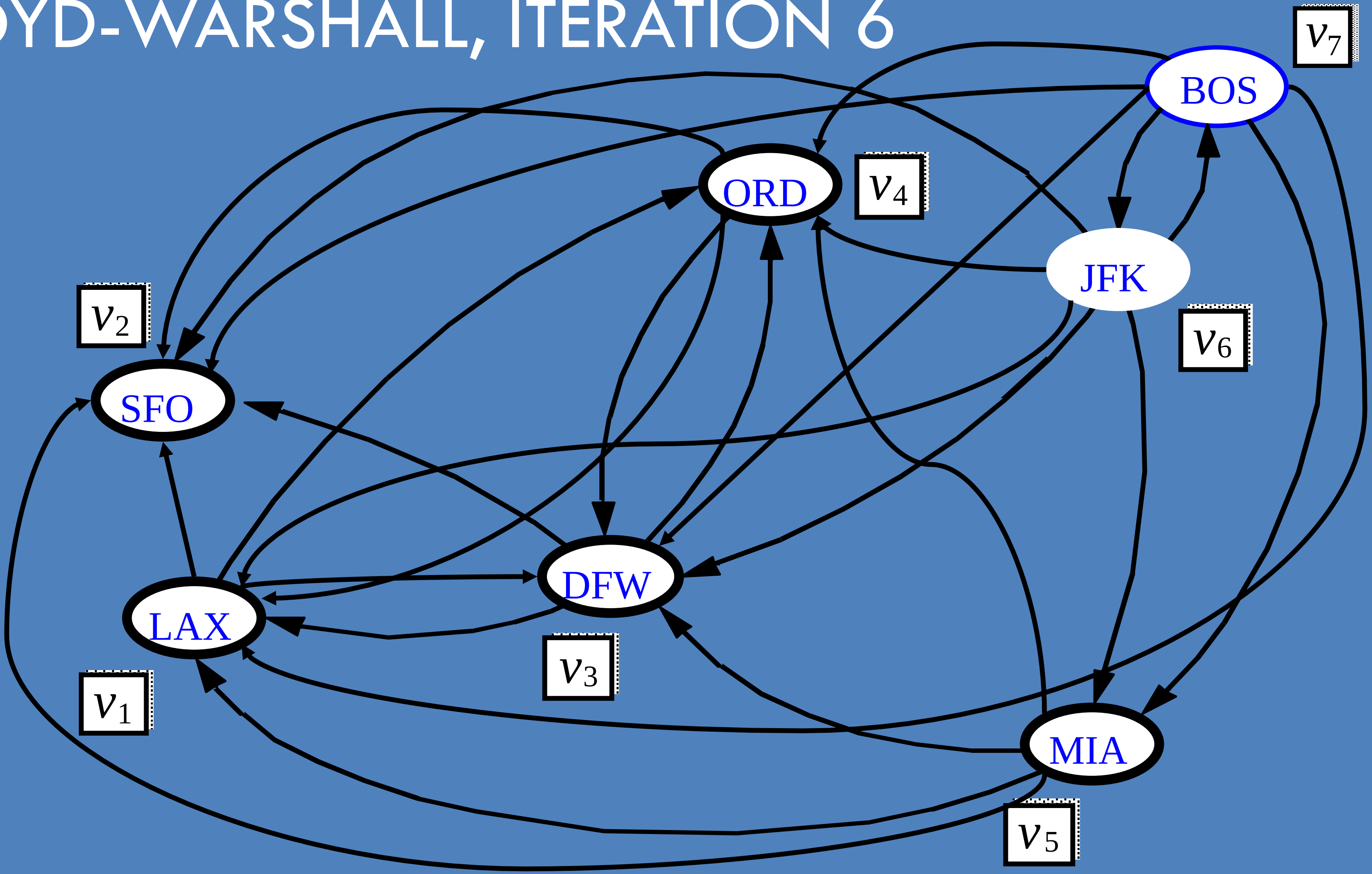
FLOYD-WARSHALL, ITERATION 4



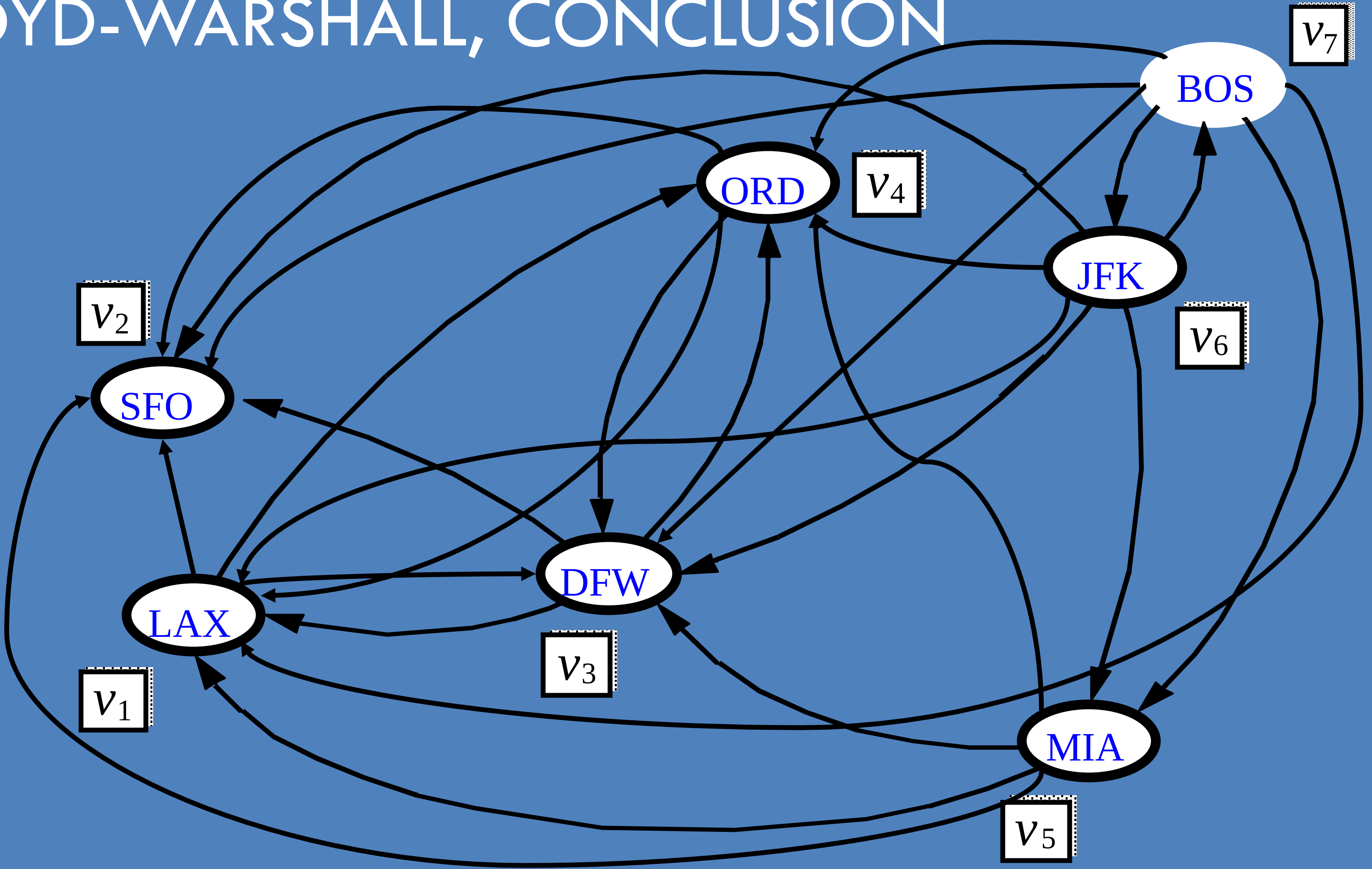
FLOYD-WARSHALL, ITERATION 5



FLOYD-WARSHALL, ITERATION 6

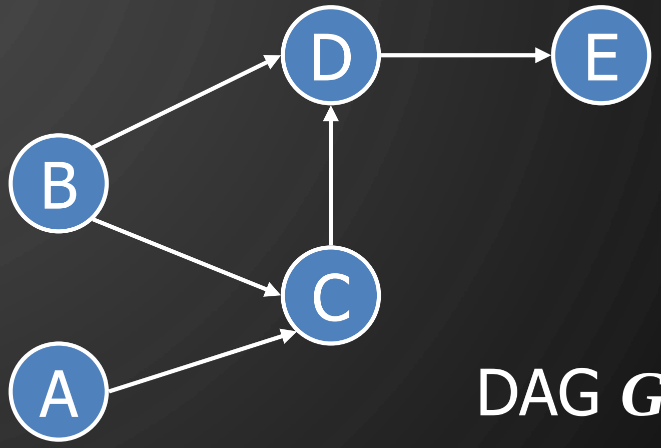


FLOYD-WARSHALL, CONCLUSION

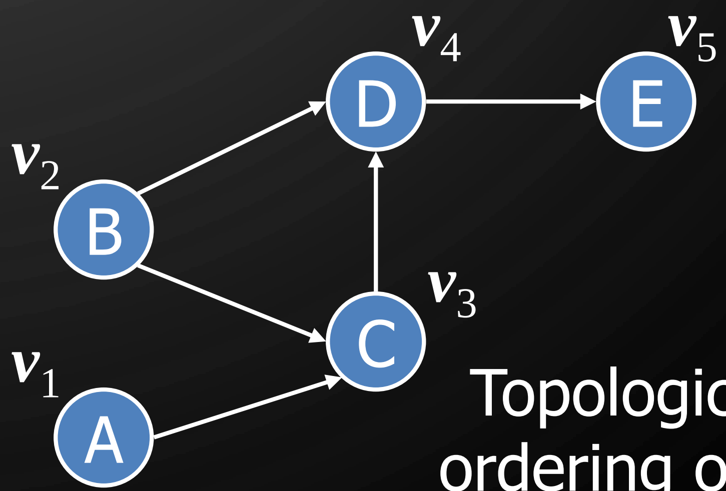


DAGS AND TOPOLOGICAL ORDERING

- A directed acyclic graph (DAG) is a digraph that has no directed cycles
- A topological ordering of a digraph is a numbering
  - $v_1, \ldots, v_n$
  - Of the vertices such that for every edge (v_i, v_j), we have $i < j$
- Example: in a task scheduling digraph, a topological ordering a task sequence that satisfies the precedence constraints
- Theorem - A digraph admits a topological ordering if and only if it is a DAG



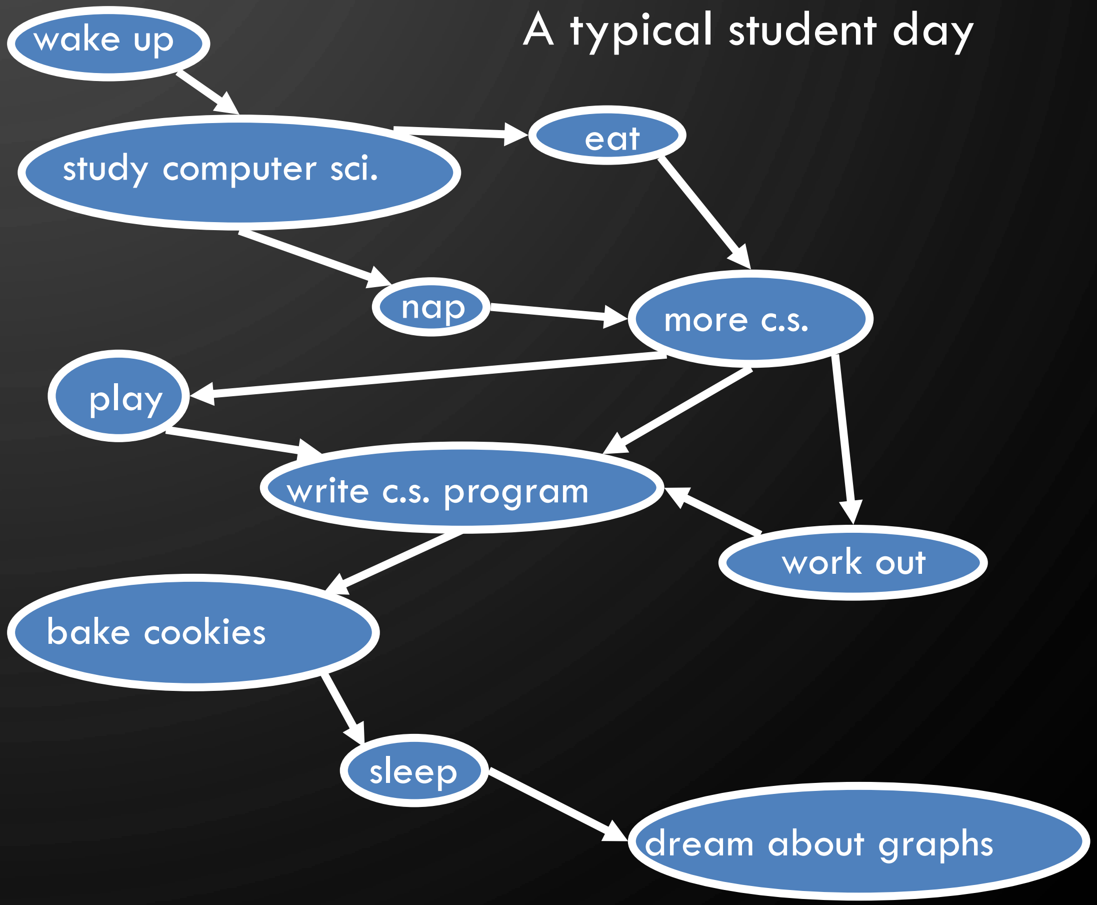
DAG G



Topological ordering of G

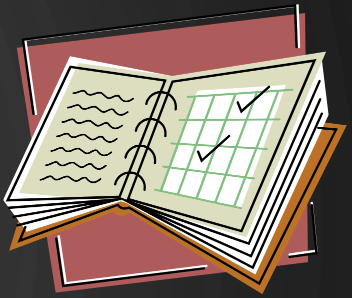
EXERCISE
TOPOLOGICAL SORTING

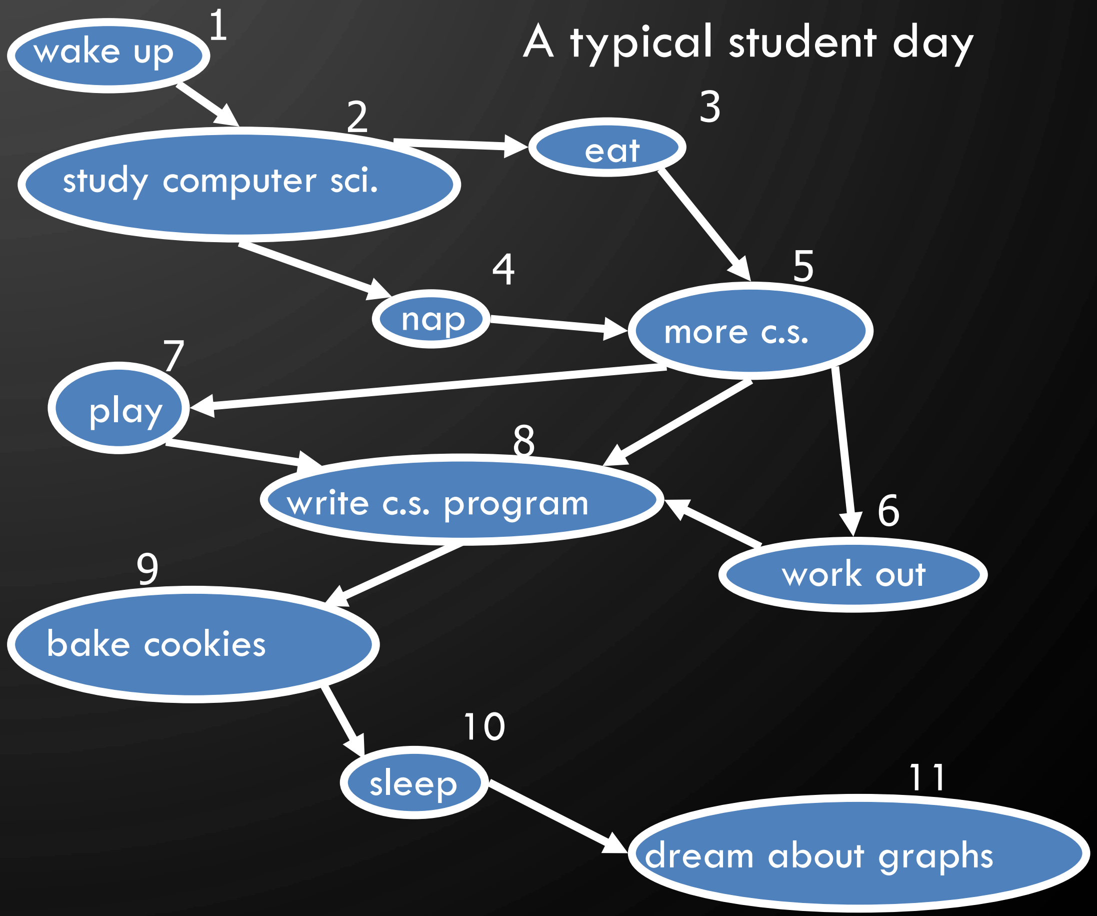
- Number vertices, so that (u, v) in E implies $u < v$



EXERCISE
TOPOLOGICAL SORTING

- Number vertices, so that (u, v) in E implies $u < v$





ALGORITHM FOR TOPOLOGICAL SORTING

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

```
Algorithm TopologicalSort($G$)
1. $H \leftarrow G$
2. $n \leftarrow G.\mathrm{numVertices}(\,)$
3. while $\neg H.\mathrm{empty}(\,)$ do
4.   Let $v$ be a vertex with no outgoing edges
5.   Label $v \leftarrow n$
6.   $n \leftarrow n - 1$
7.   $H.\mathrm{eraseVertex}(v)$
```

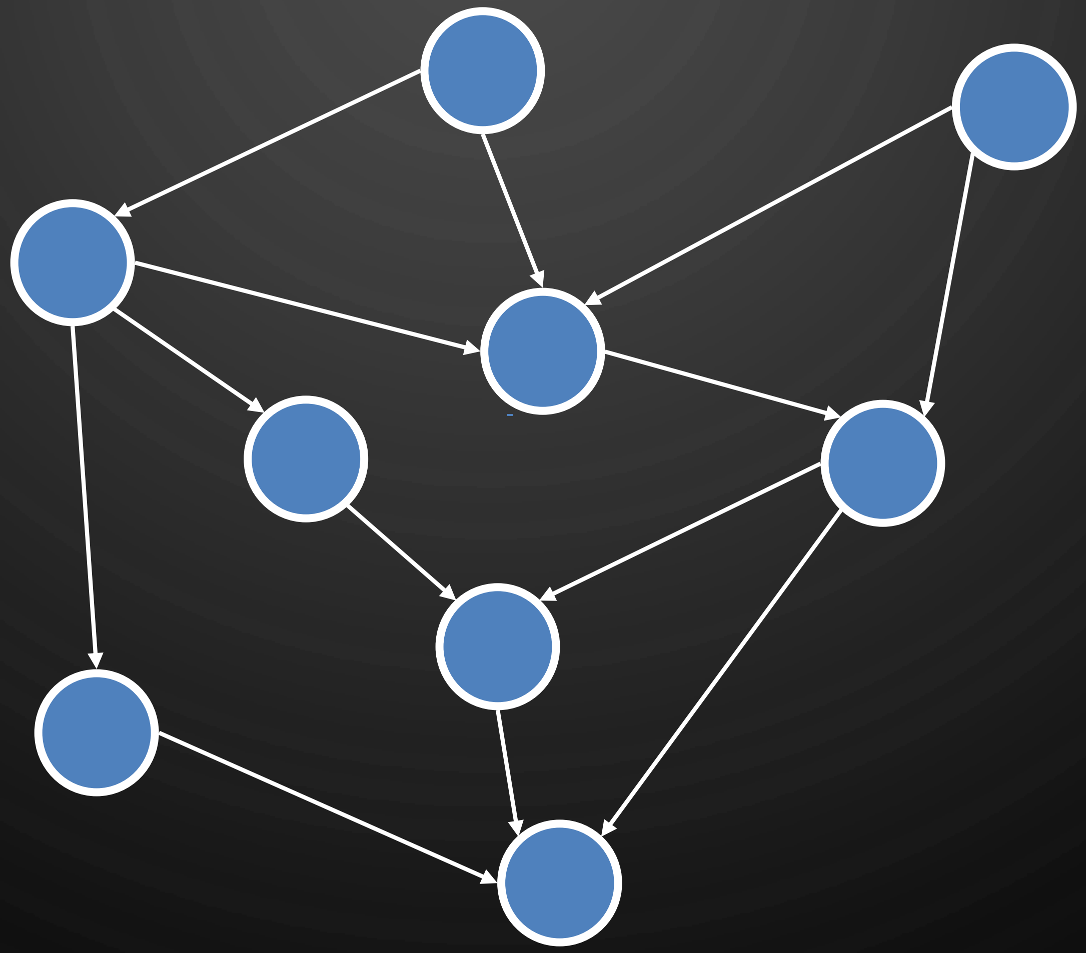
IMPLEMENTATION WITH DFS

- Simulate the algorithm by using depth-first search
- $O(n+m)$ time.

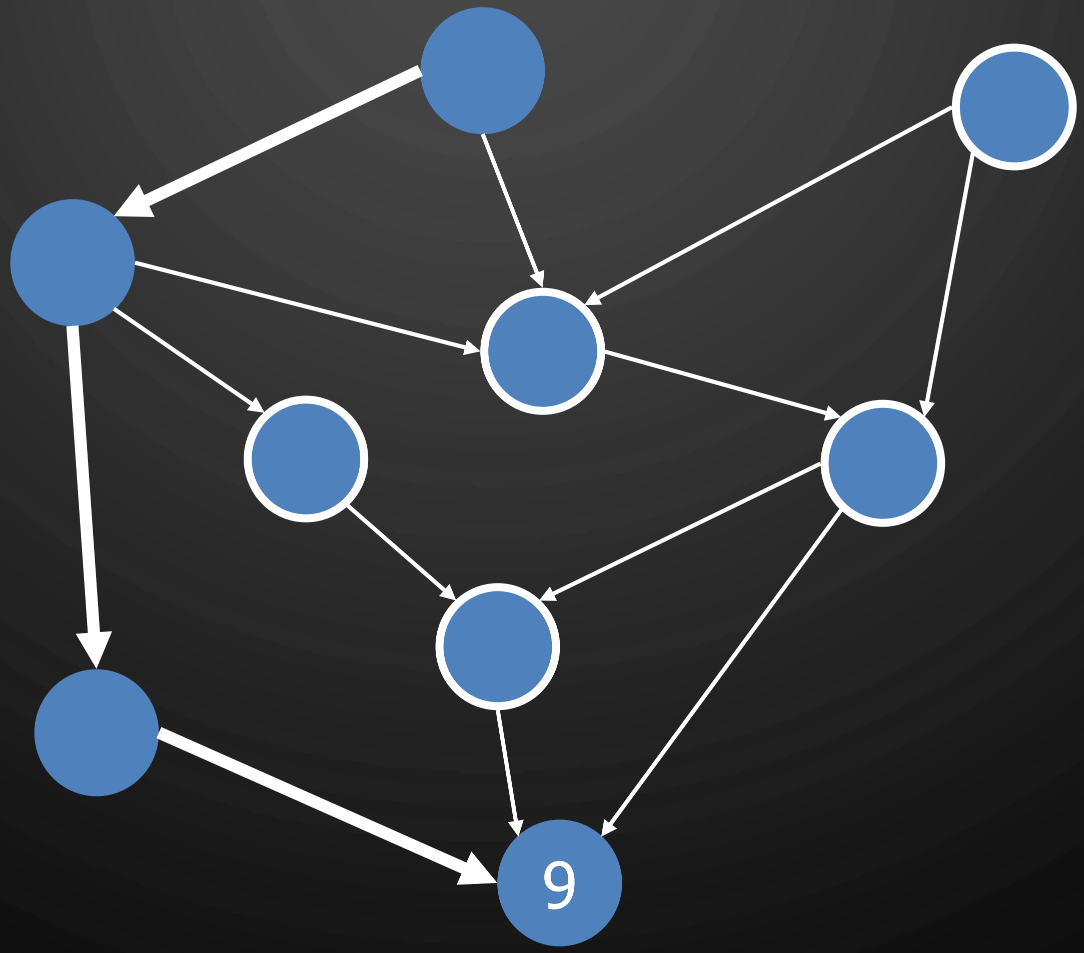
```
Algorithm topologicalDFS(G)
Input: DAG $G$
Output: Topological ordering of $g$
1.  $n \leftarrow G.\mathrm{numVertices}()$
2.  Initialize all vertices as $UNEXPLORED$
3.  for each vertex $v \in G.\mathrm{vertices}()$ do
4.    if $v.\mathrm{getLabel}() = UNEXPLORED$ then
5.      $\mathrm{topologicalDFS}(G, v)$
```

```
Algorithm topologicalDFS(G, v)
Input: DAG $G$, start vertex $v$
Output: Labeling of the vertices of $G$
        in the connected component of $v$
1.  $v.\mathrm{setLabel}(VISITED)$
2.  for each $e \in v.\mathrm{outEdges}()$ do
3.    $w \leftarrow e.\mathrm{dest}()$
4.    if $w.\mathrm{getLabel}() = UNEXPLORED$ then
5.      //$e$ is a discovery edge
6.      $\mathrm{topologicalDFS}(G, w)$
7.    else
8.      //$e$ is a forward, cross, or back edge
9.  Label $v$ with topological number $n$
10. $n \leftarrow n - 1$
```

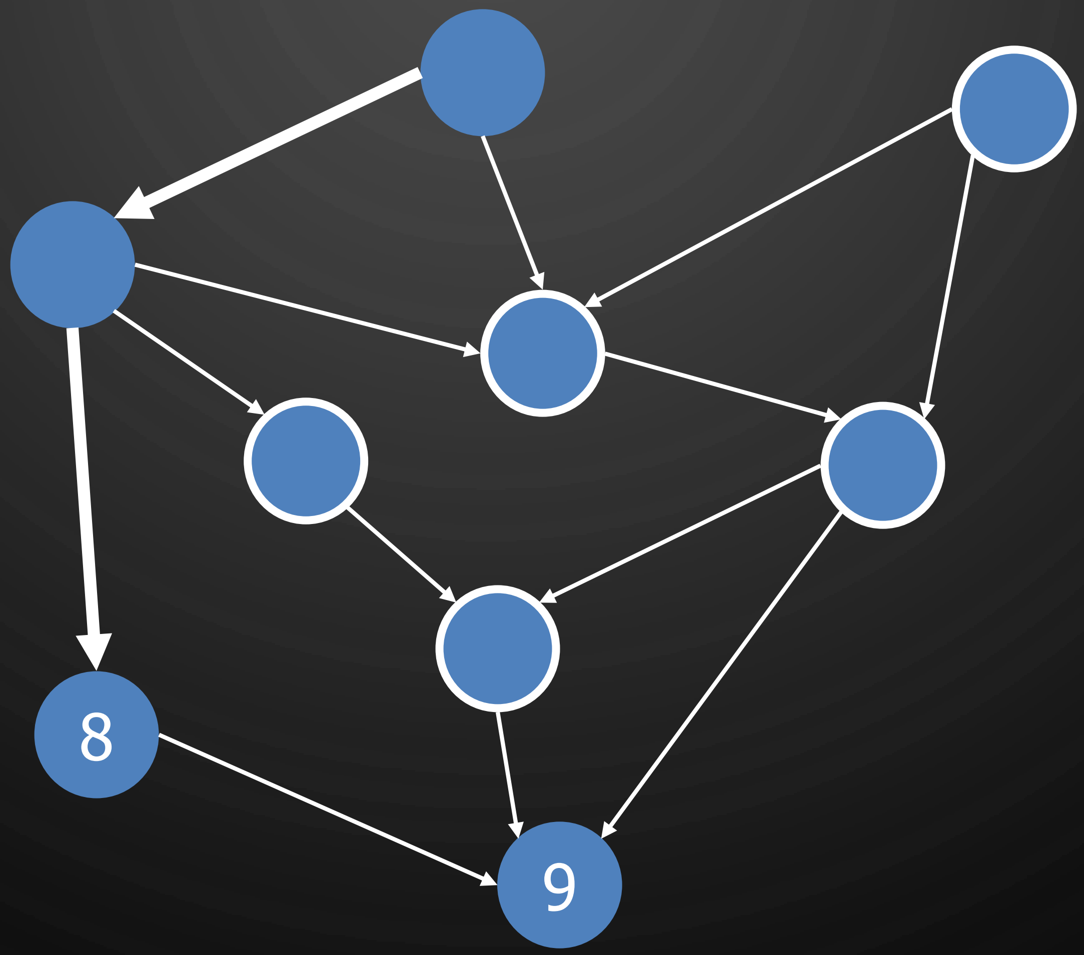
TOPOLOGICAL SORTING EXAMPLE



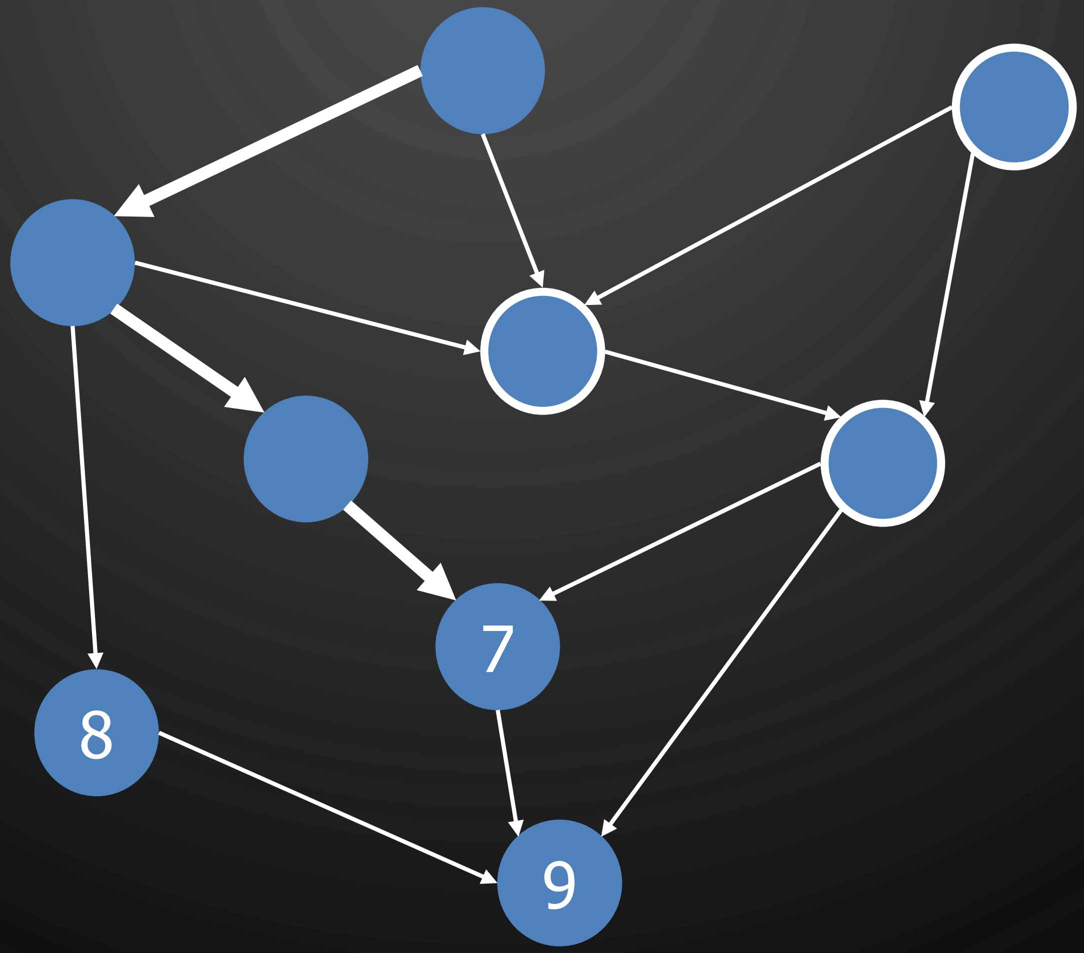
TOPOLOGICAL SORTING EXAMPLE



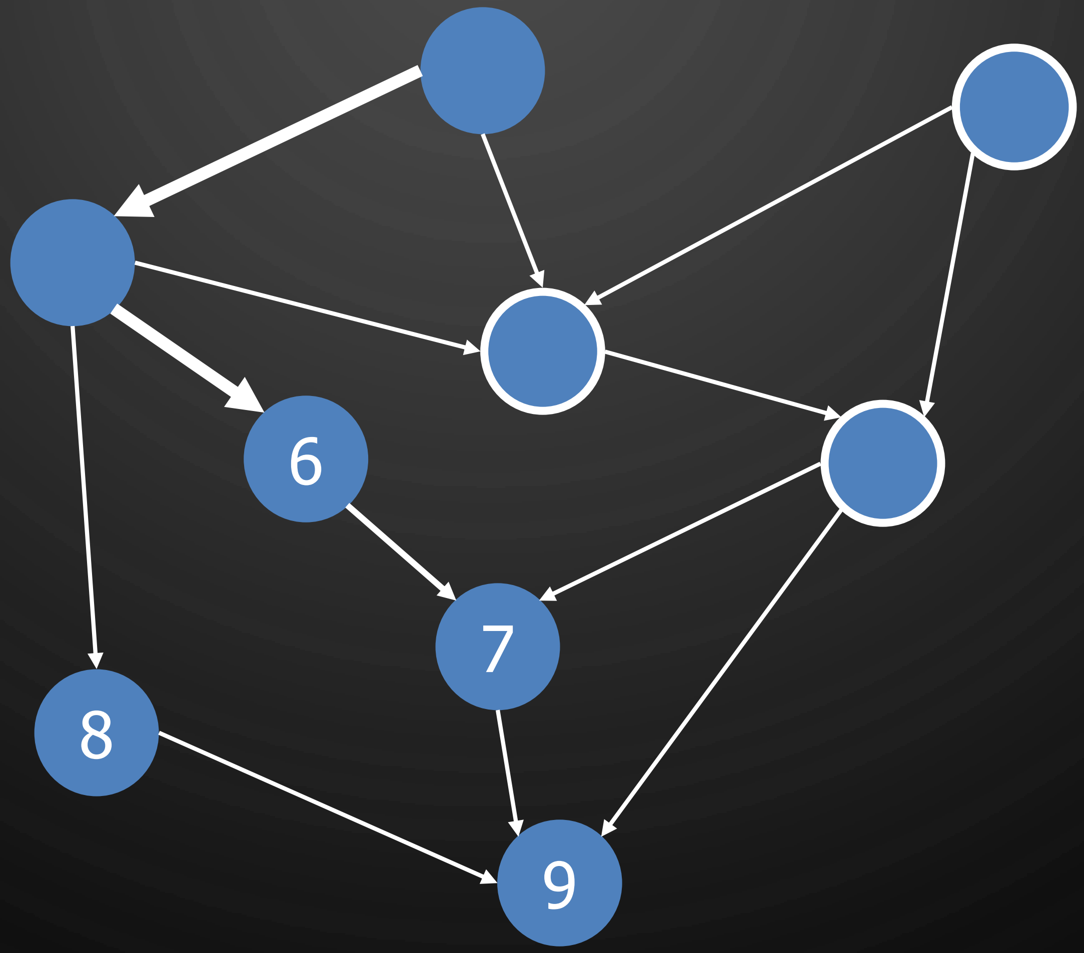
TOPOLOGICAL SORTING EXAMPLE



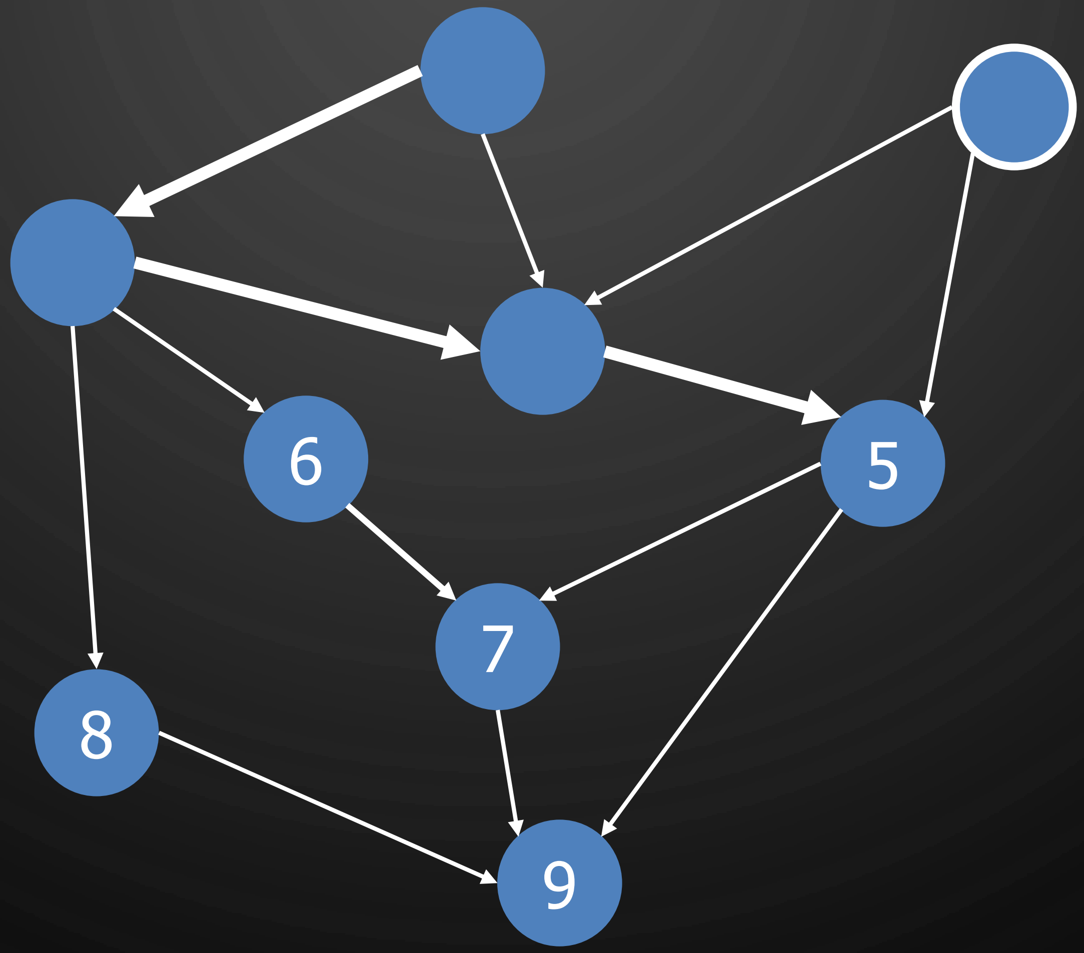
TOPOLOGICAL SORTING EXAMPLE



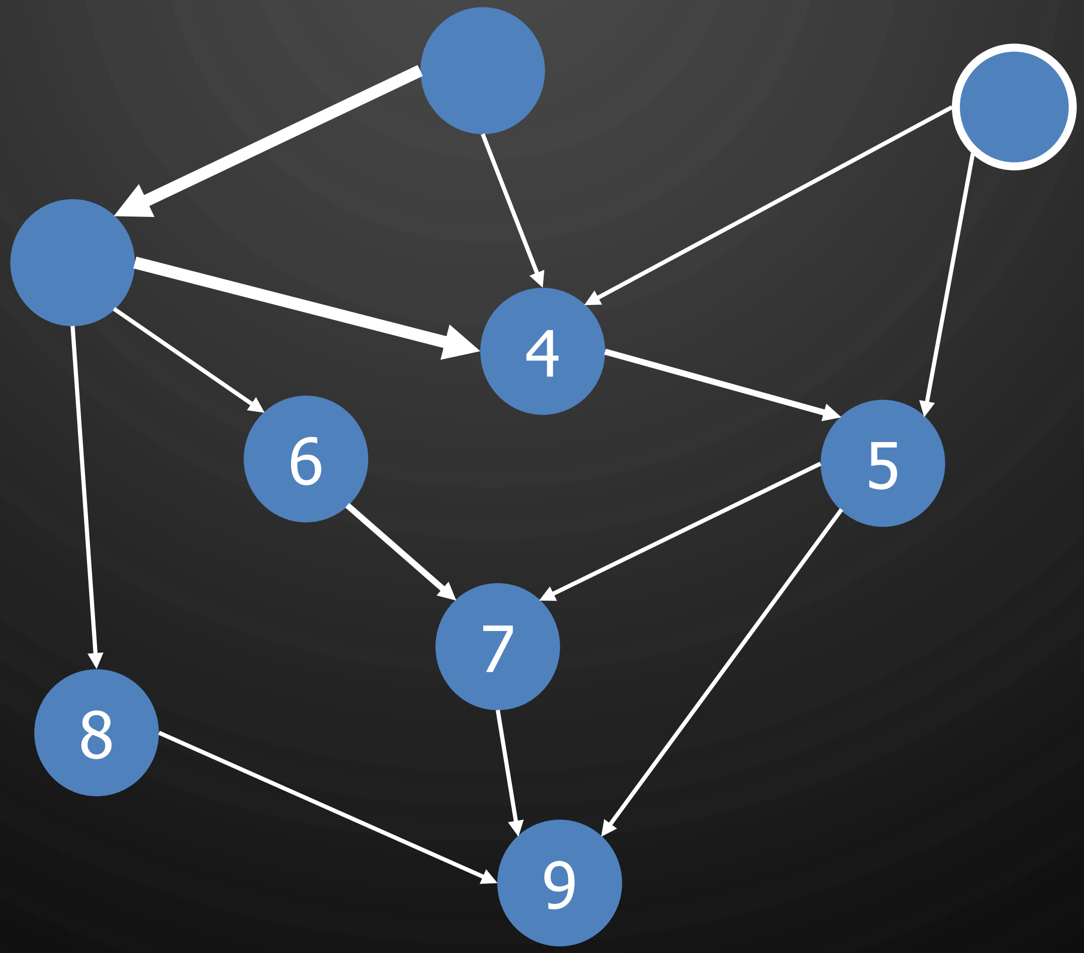
TOPOLOGICAL SORTING EXAMPLE



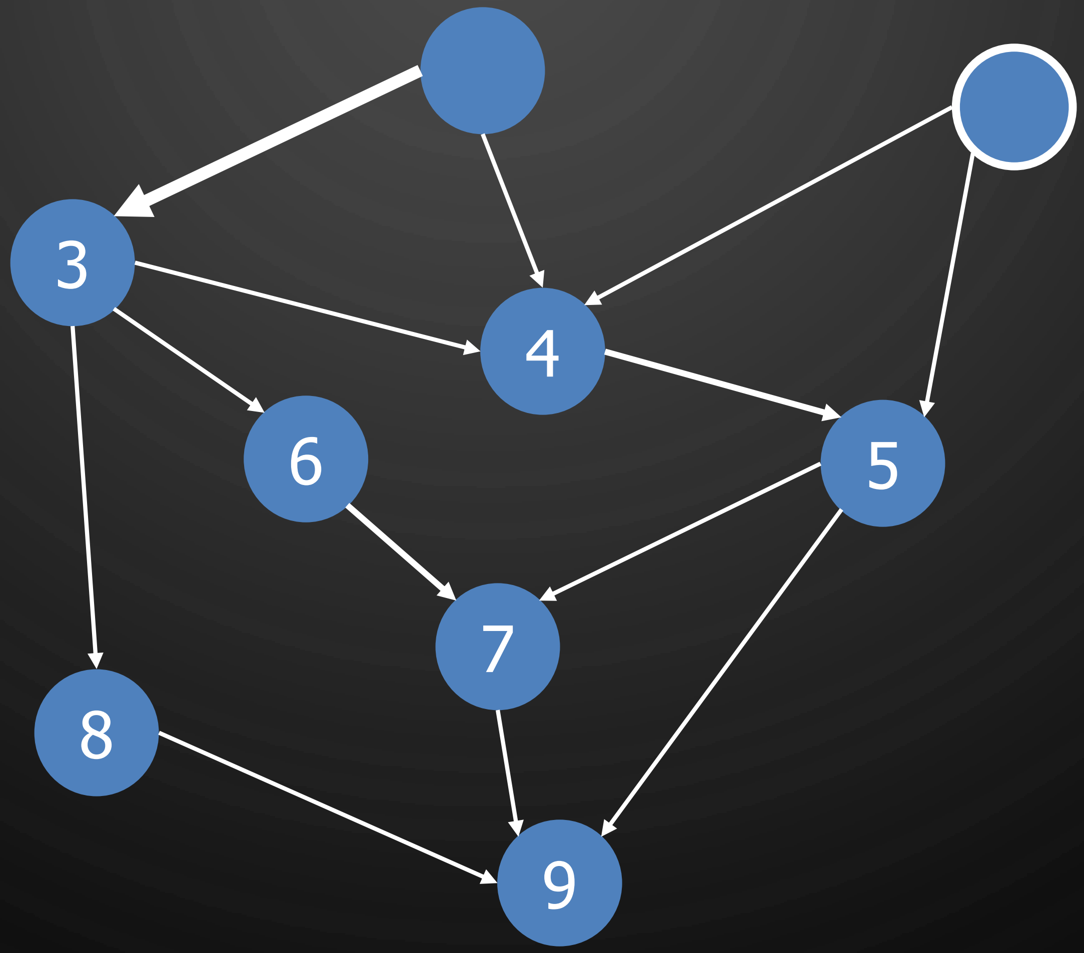
TOPOLOGICAL SORTING EXAMPLE



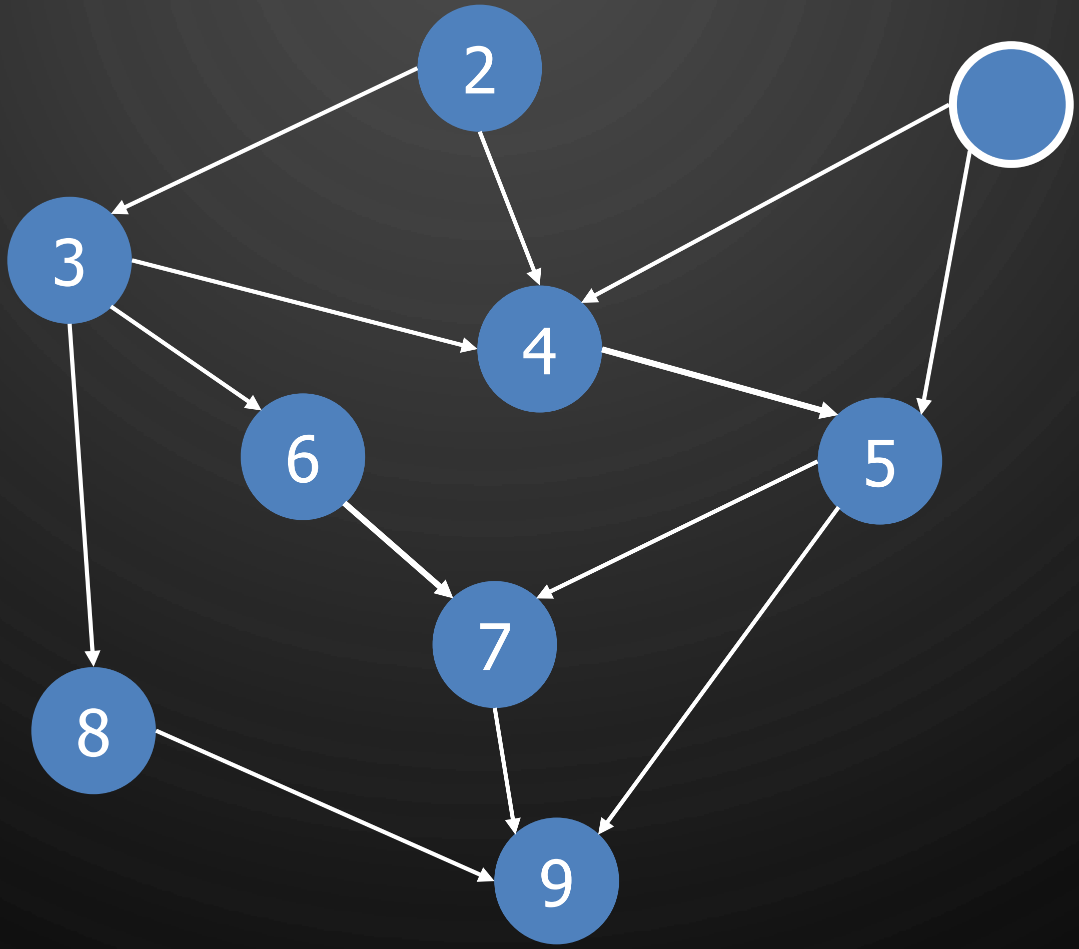
TOPOLOGICAL SORTING EXAMPLE



TOPOLOGICAL SORTING EXAMPLE



TOPOLOGICAL SORTING EXAMPLE



TOPOLOGICAL SORTING EXAMPLE