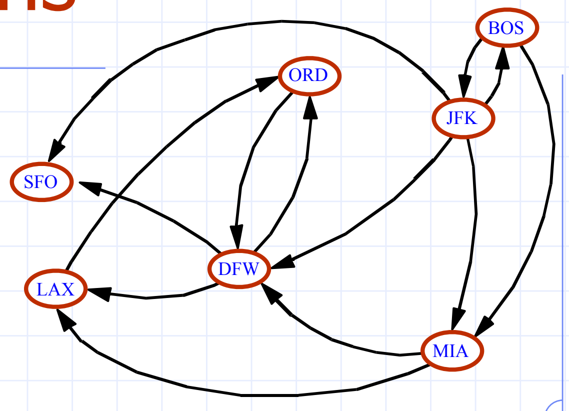
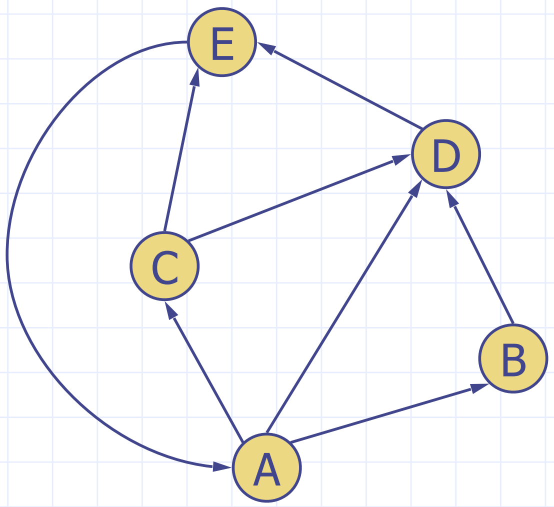
Directed Graphs

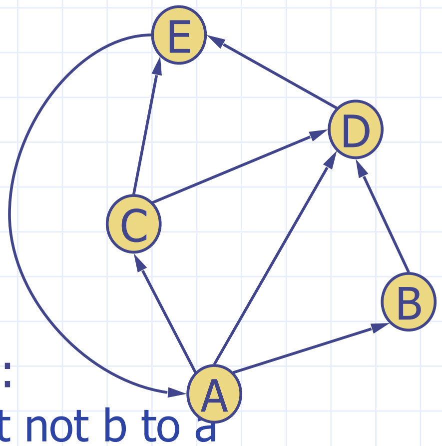


Digraphs

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



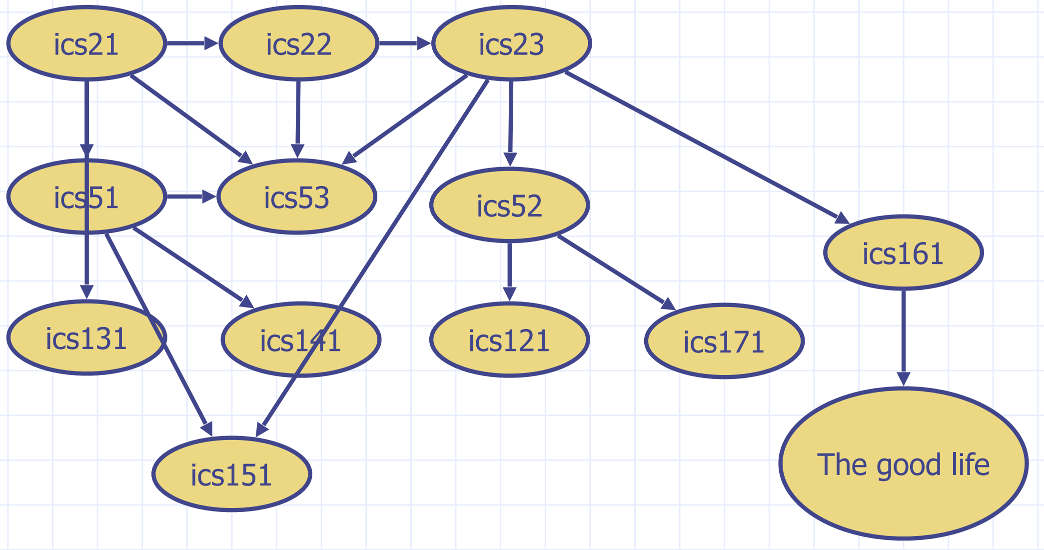
Digraph Properties



- 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, $m \leq n \cdot (n - 1)$
- 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

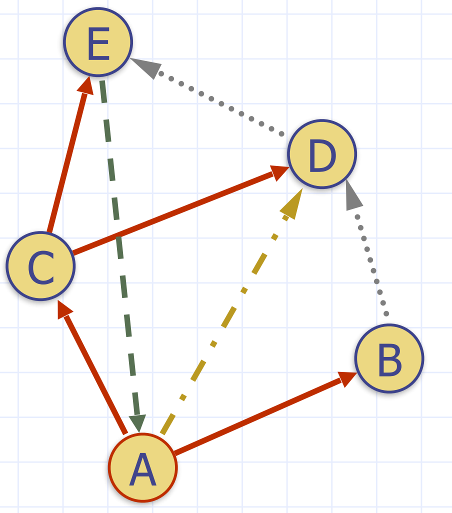
Digraph Application

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



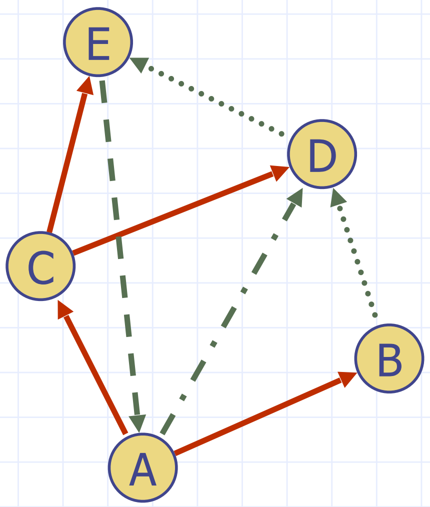
Directed DFS

- 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



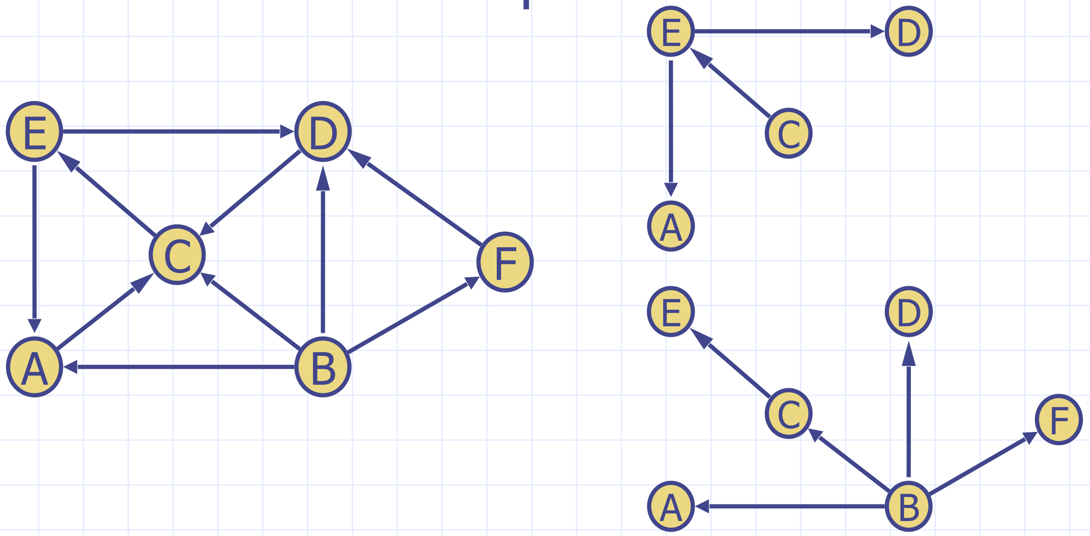
Directed DFS

- Back edge (**v,w**)
  - **w** is an ancestor of **v** in the tree of discovery edges
- Forward edge (**v,w**)
  - v is an ancestor (but not the parent) of w in the tree of discovery edges
- Cross edge (*v*,*w*)
  - *w* is in the same level as *v* or in the next level in the tree of discovery edges
- Discovery edge (**v,w**)
  - v is the parent of w in the tree of discovery edges



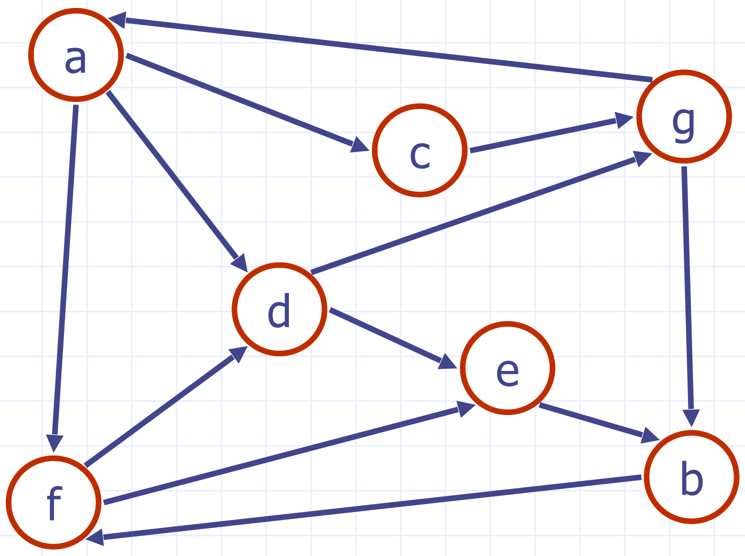
Reachability

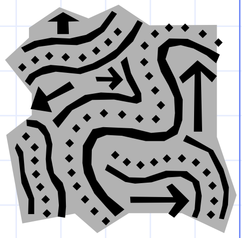
- DFS tree rooted at v: vertices reachable from v via directed paths



Strong Connectivity

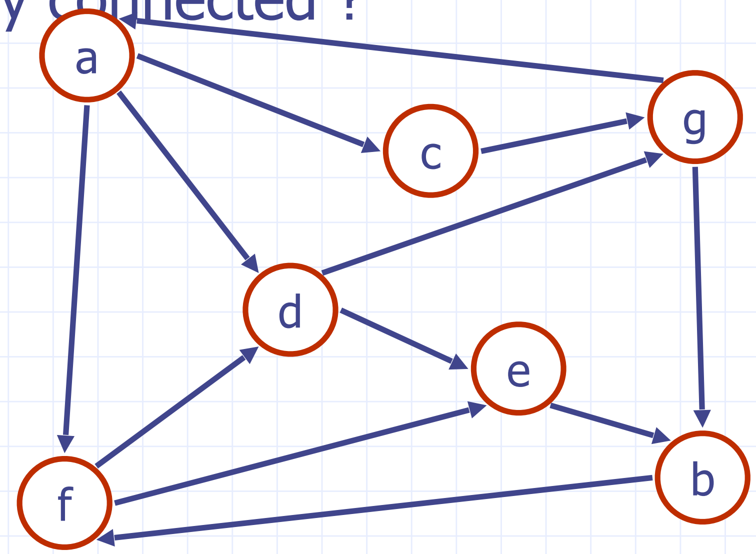
- Each vertex can reach all other vertices



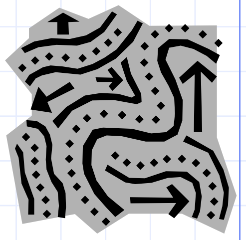


Strong Connectivity

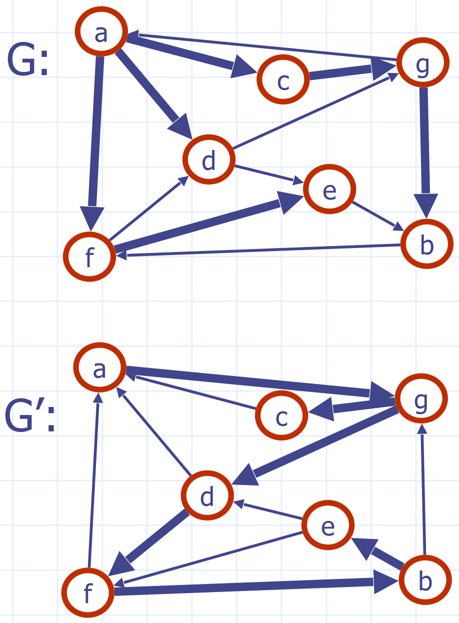
- How do you determine if a digraph is strongly connected ?



Strong Connectivity Algorithm

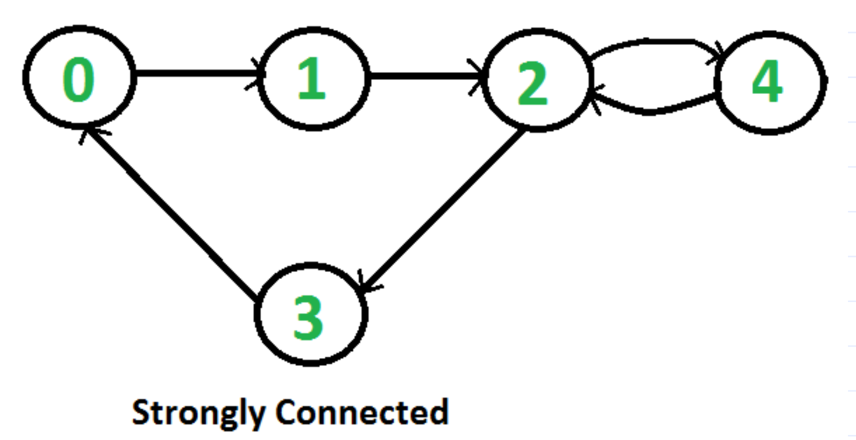


- Pick a vertex v in G
- Perform a DFS from v in G
  - If there's a w not visited, print "no"
- Let G' be G with edges reversed
- Perform a DFS from v in G'
  - If there's a w not visited, print "no"
  - Else, print "yes"
- Running time: O(n+m)
- Transpose of Graph G



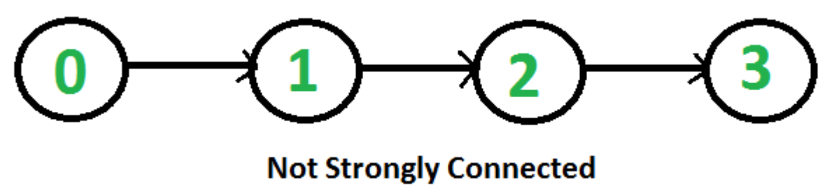
Graph connectivity

- Strongly

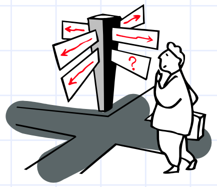


Graph connectivity

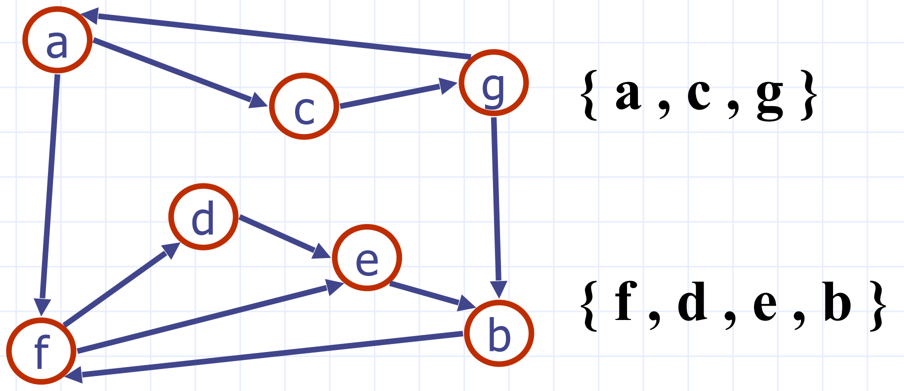
- Strongly



Strongly Connected Components

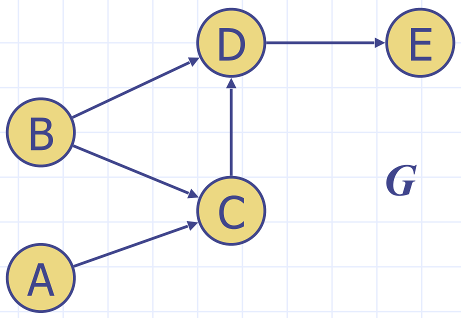


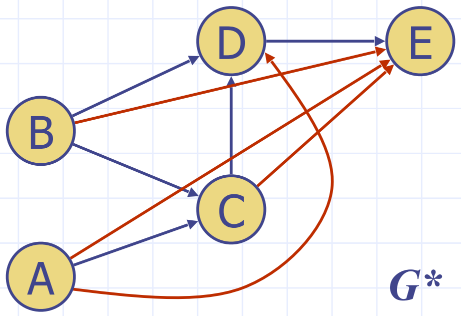
- Maximal subgraphs such that each vertex can reach all other vertices in the subgraph
- Can also be done in O(n+m) time using DFS, but is more complicated (similar to biconnectivity).



Transitive Closure

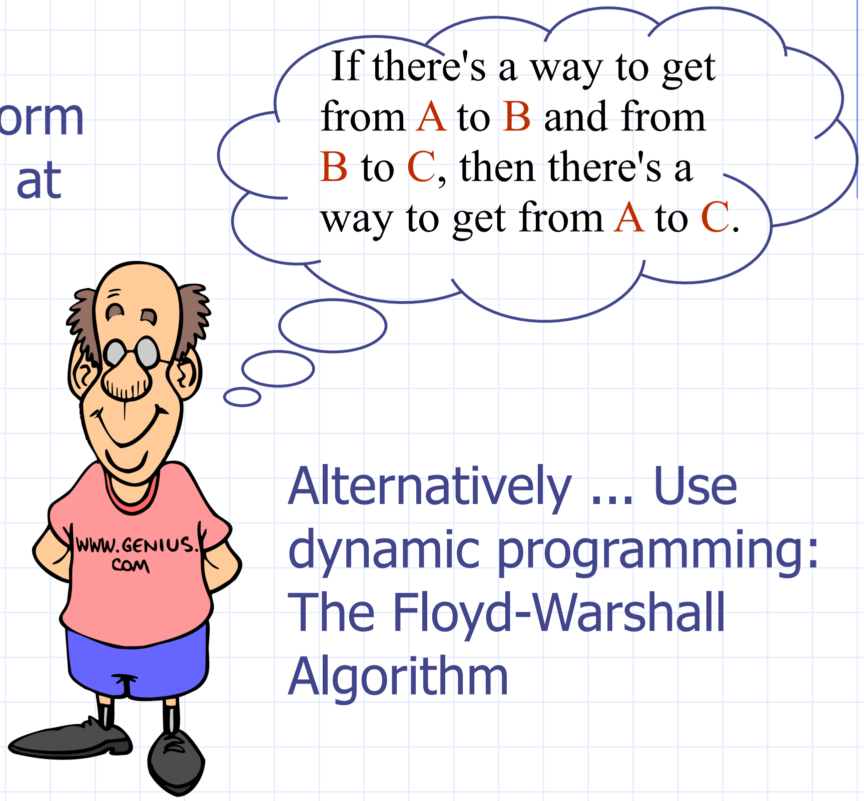
- Given a digraph G, the transitive closure of G is the digraph G^* such that
  - G^* has the same vertices as G
  - if G has a directed path from u to v ($u \neq v$), G^* has a directed edge from u to v
- The transitive closure provides reachability information about a digraph





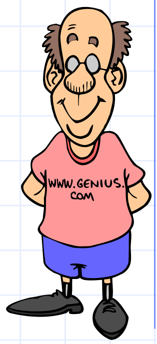
Computing the Transitive Closure

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

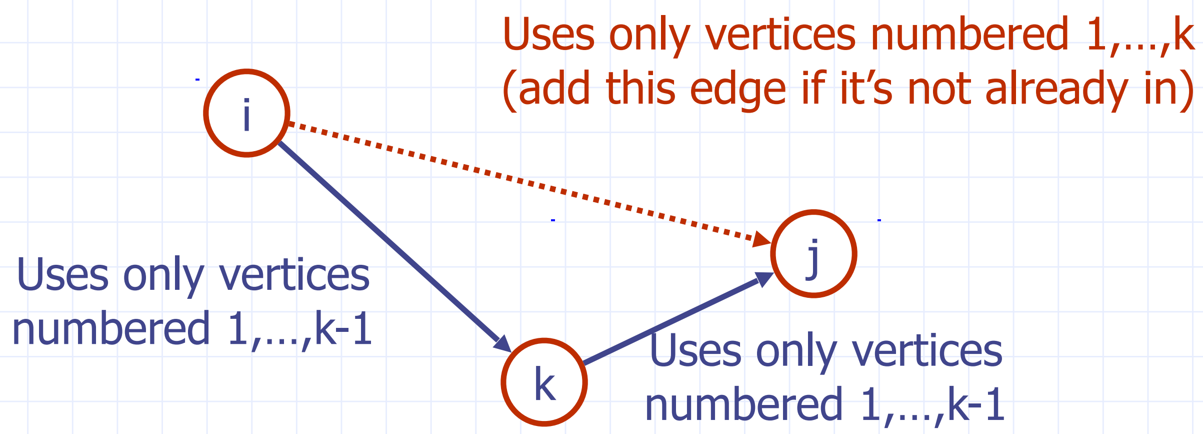


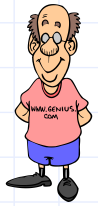
Alternatively ... Use dynamic programming: The Floyd-Warshall Algorithm

Floyd-Warshall Transitive Closure



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



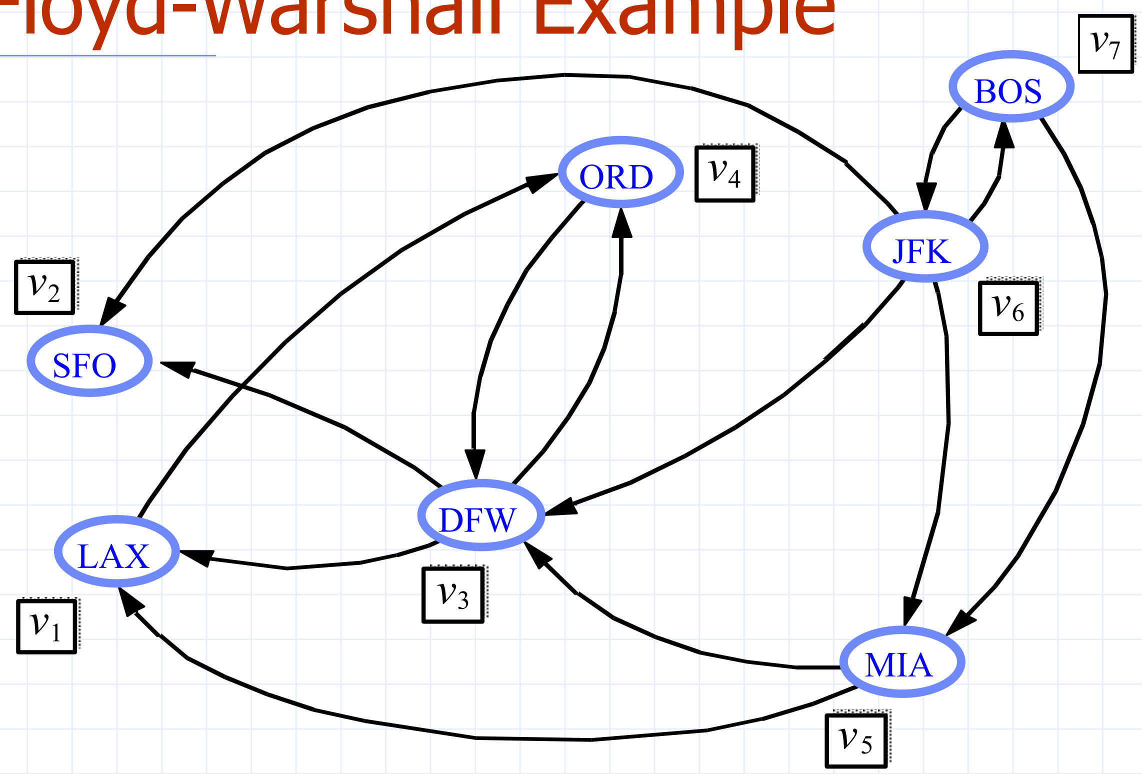


Floyd-Warshall's Algorithm

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

Algorithm *FloydWarshall*(G)
  Input digraph G
  Output transitive closure G^* of G
  $i \leftarrow 1$
  for all $v \in G.vertices()$
    denote v as 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

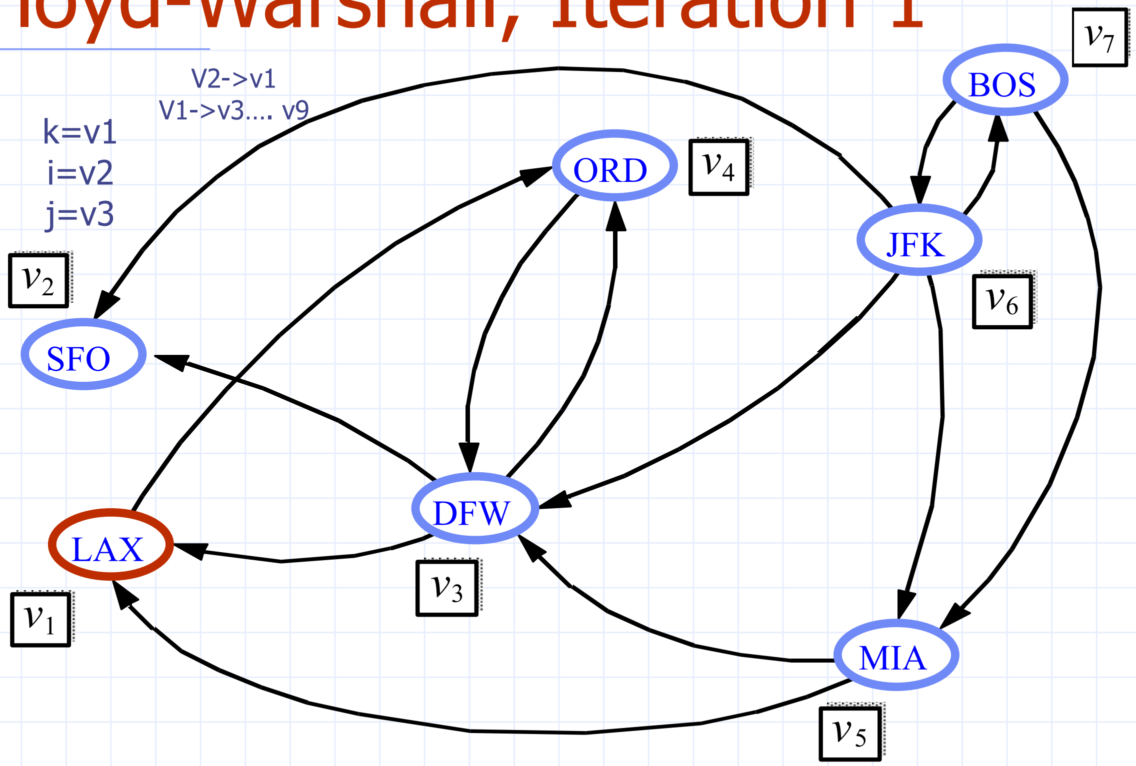
Floyd-Warshall Example



Floyd-Warshall, Iteration 1

k=v1
i=v2
j=v3

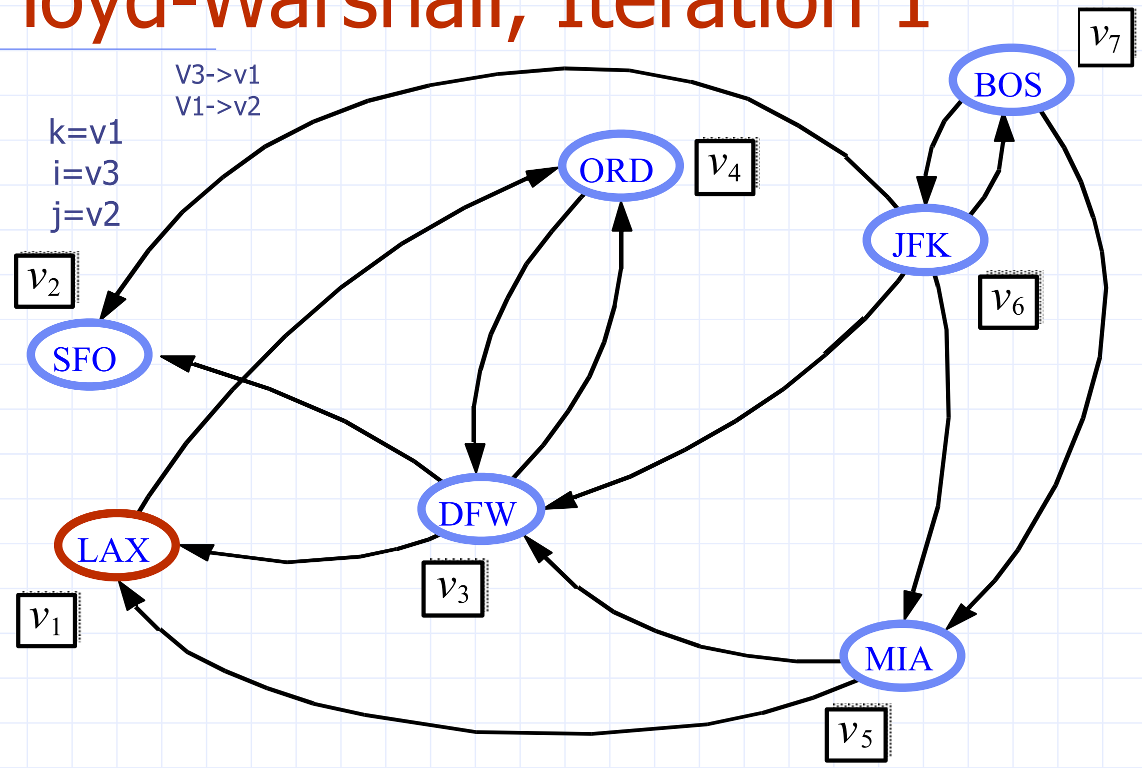
V2->v1
V1->v3.... v9



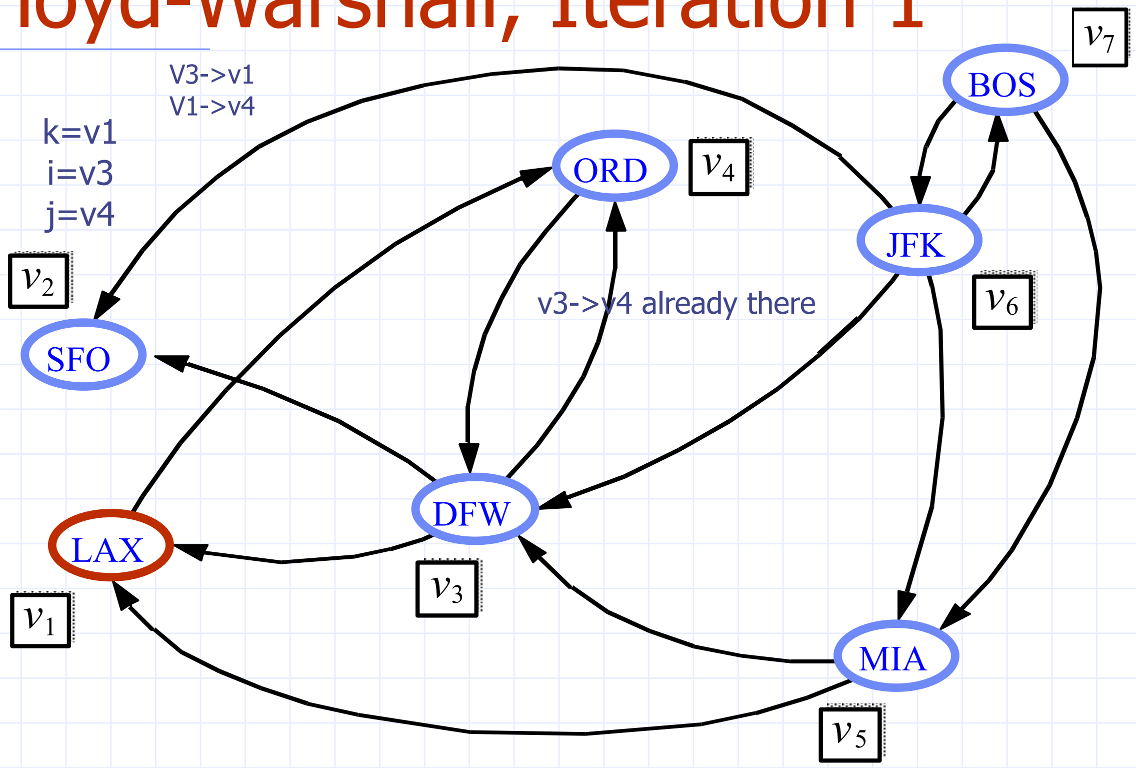
Floyd-Warshall, Iteration 1

k=v1
i=v3
j=v2

V3->v1
V1->v2



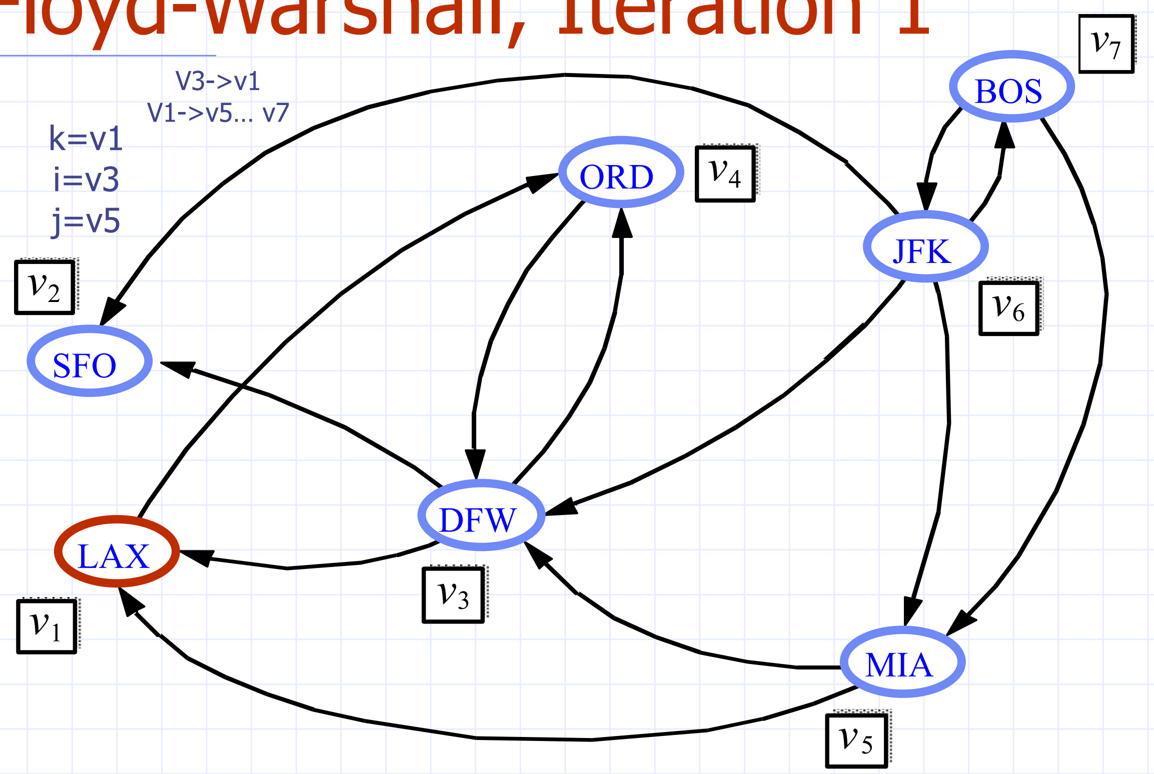
Floyd-Warshall, Iteration 1



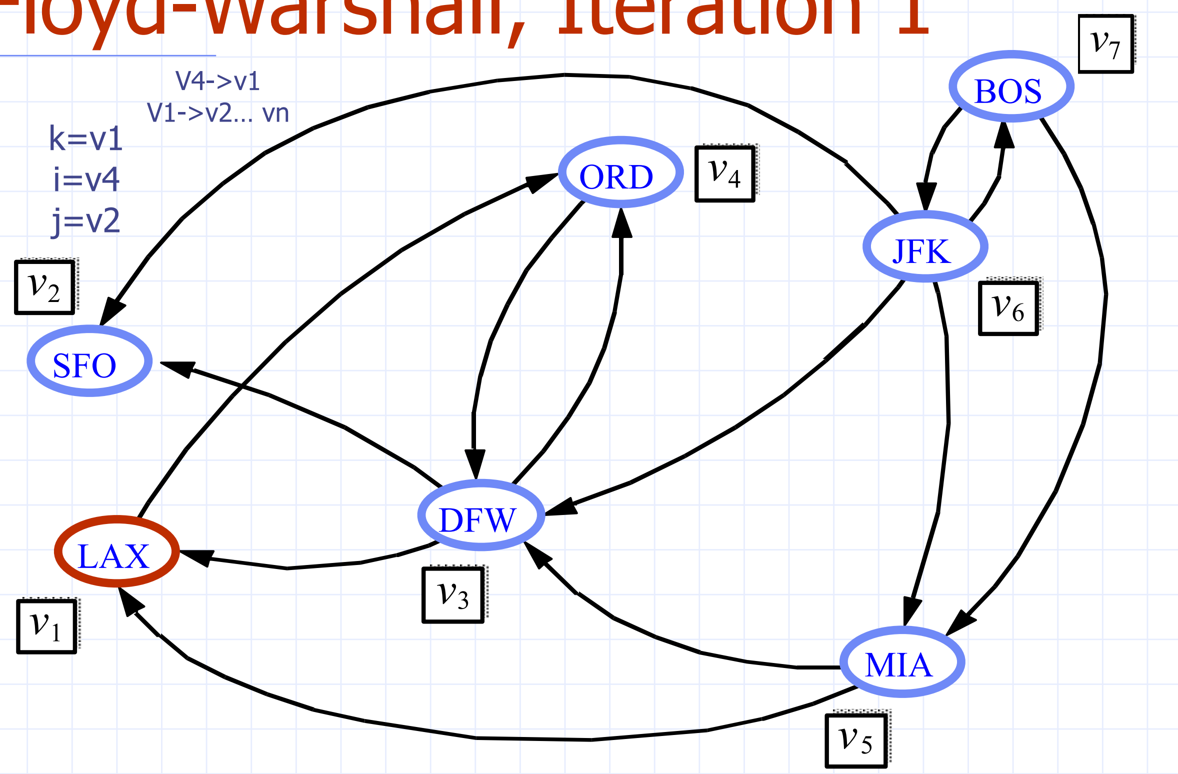
Floyd-Warshall, Iteration 1

V3->v1
V1->v5... v7

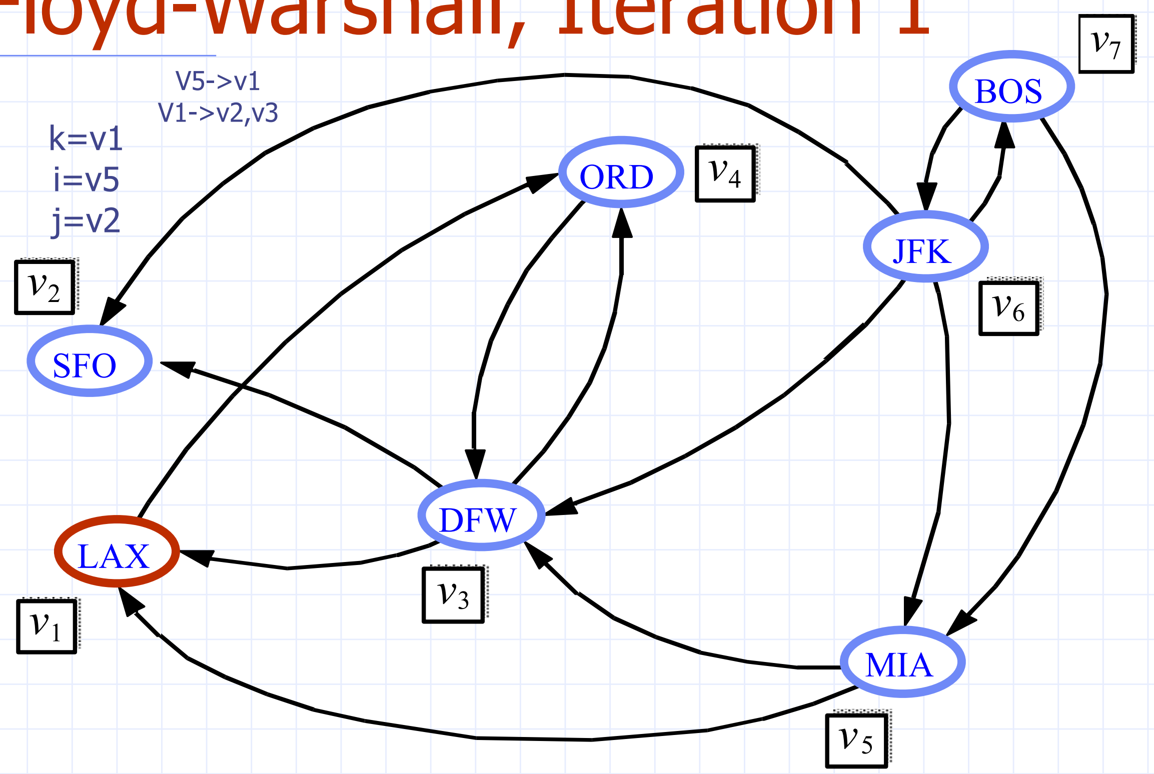
k=v1
i=v3
j=v5



Floyd-Warshall, Iteration 1

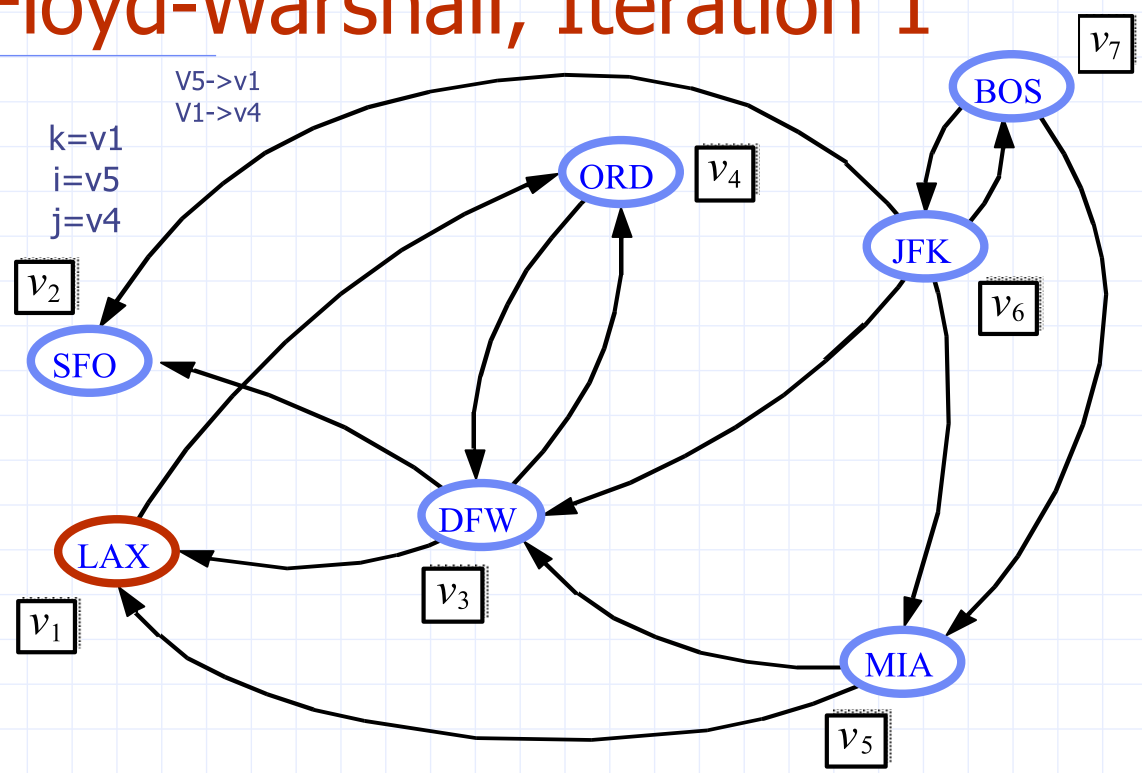


Floyd-Warshall, Iteration 1

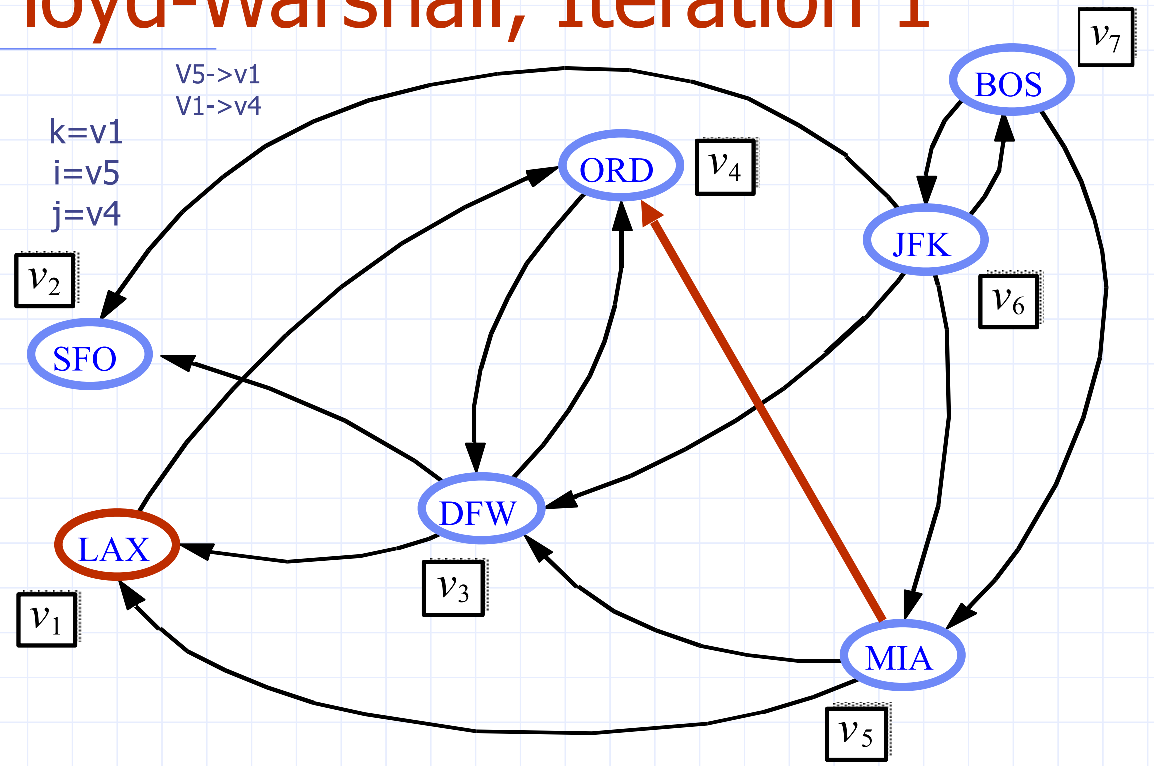


Floyd-Warshall, Iteration 1

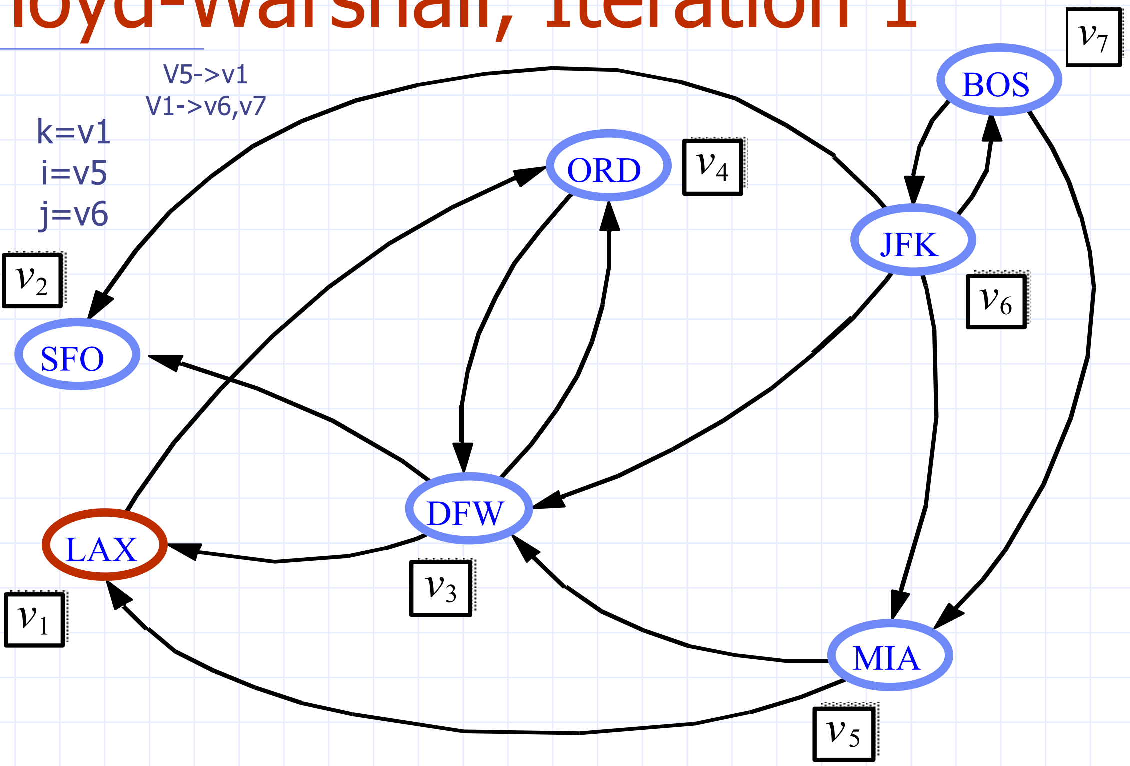
k=v1
i=v5
j=v4



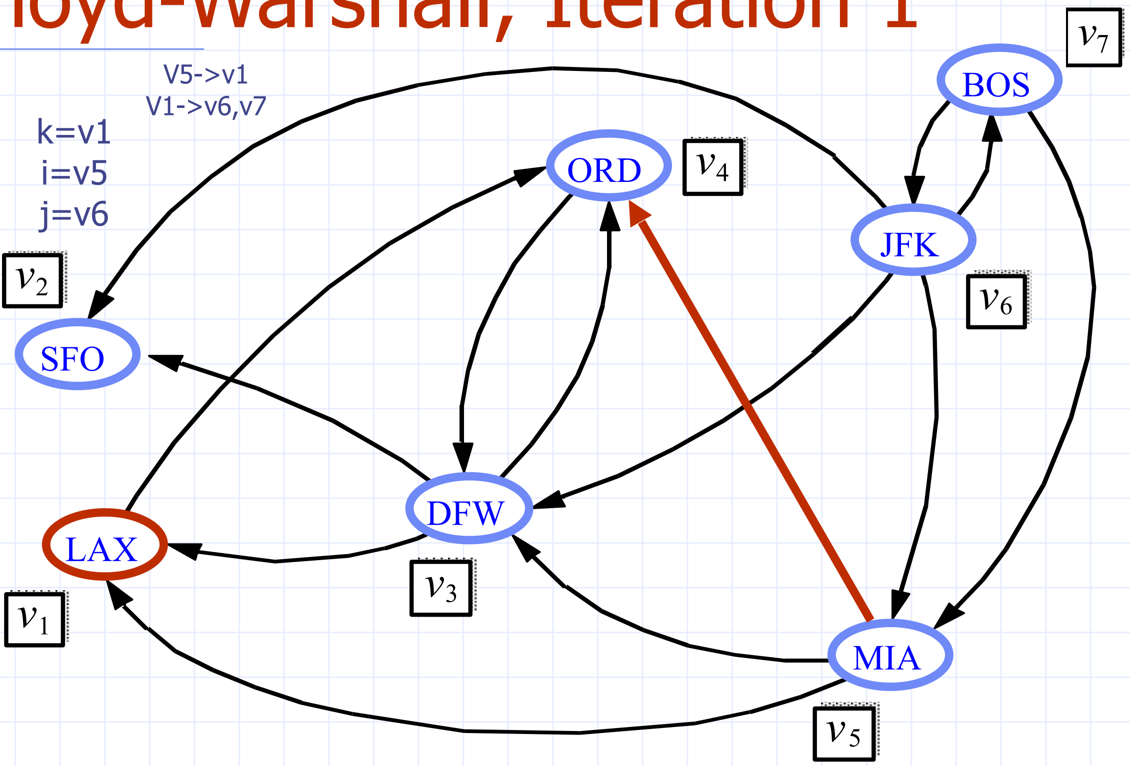
Floyd-Warshall, Iteration 1



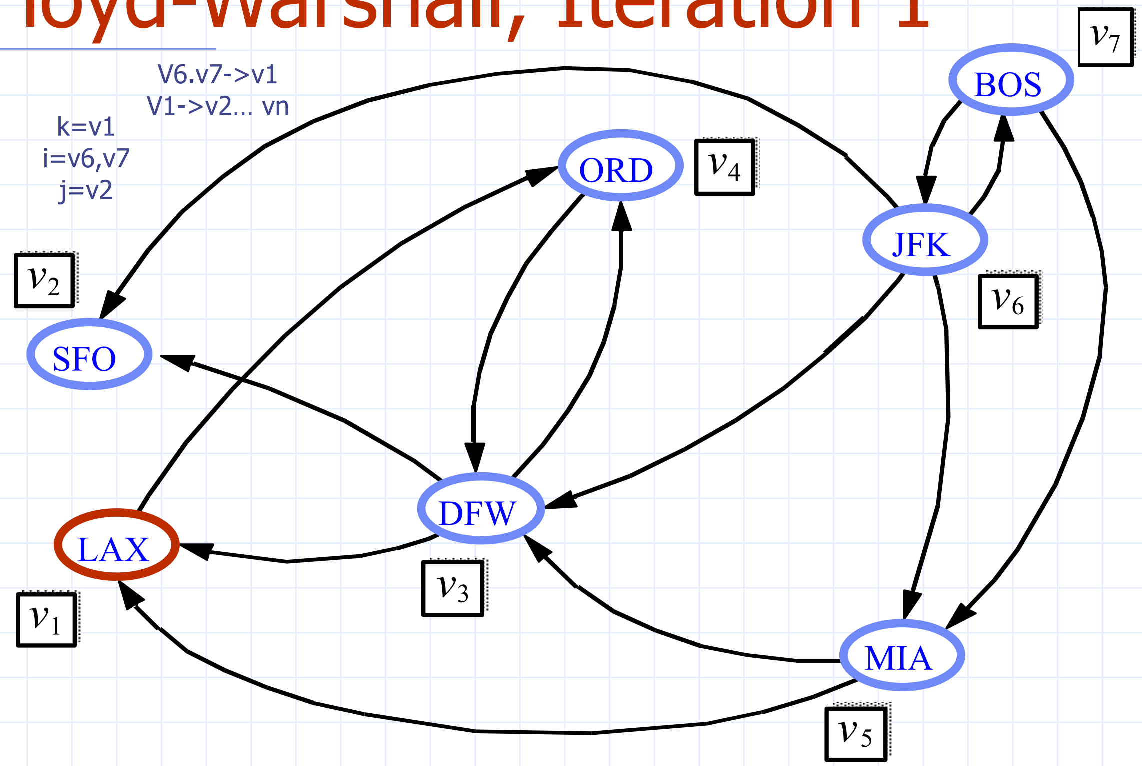
Floyd-Warshall, Iteration 1



Floyd-Warshall, Iteration 1



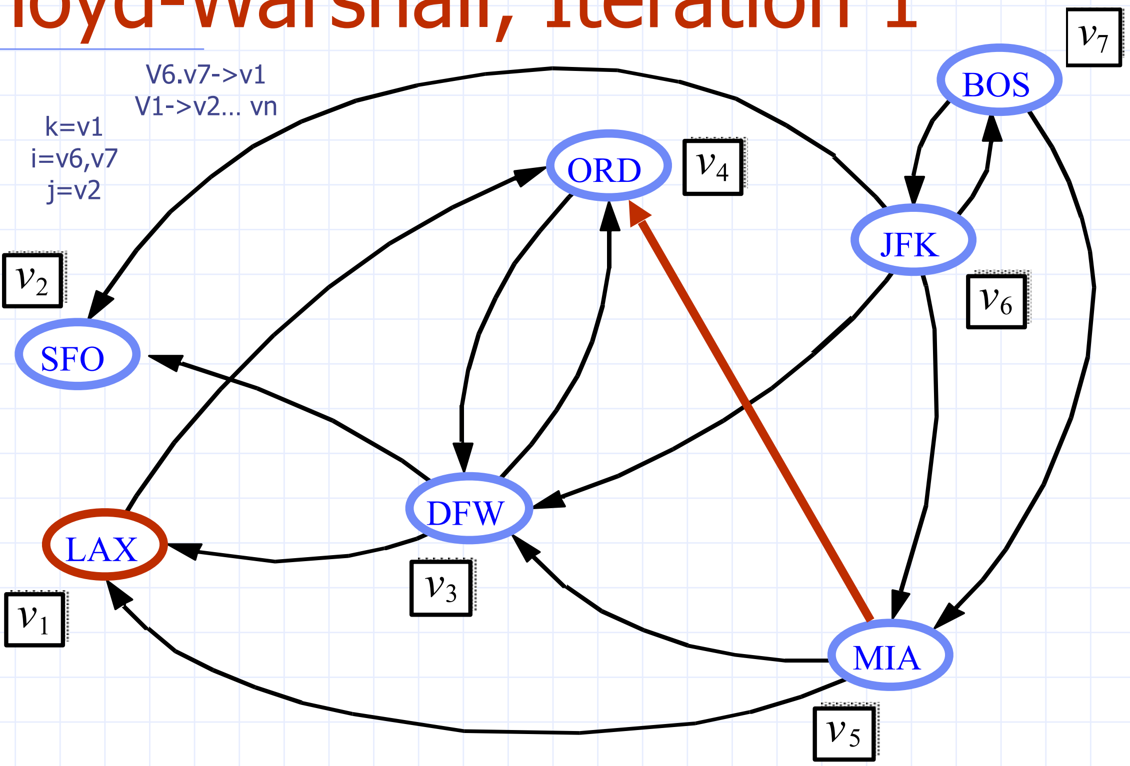
Floyd-Warshall, Iteration 1



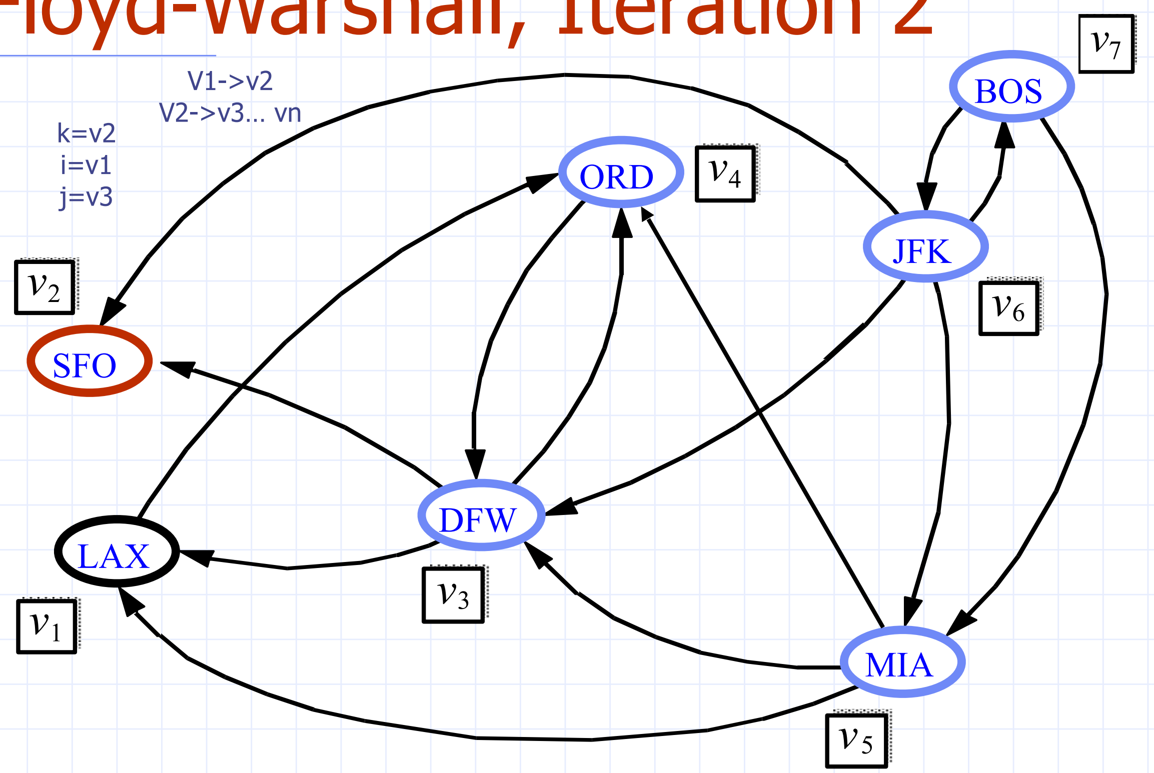
Floyd-Warshall, Iteration 1

V6.v7->v1
V1->v2... vn

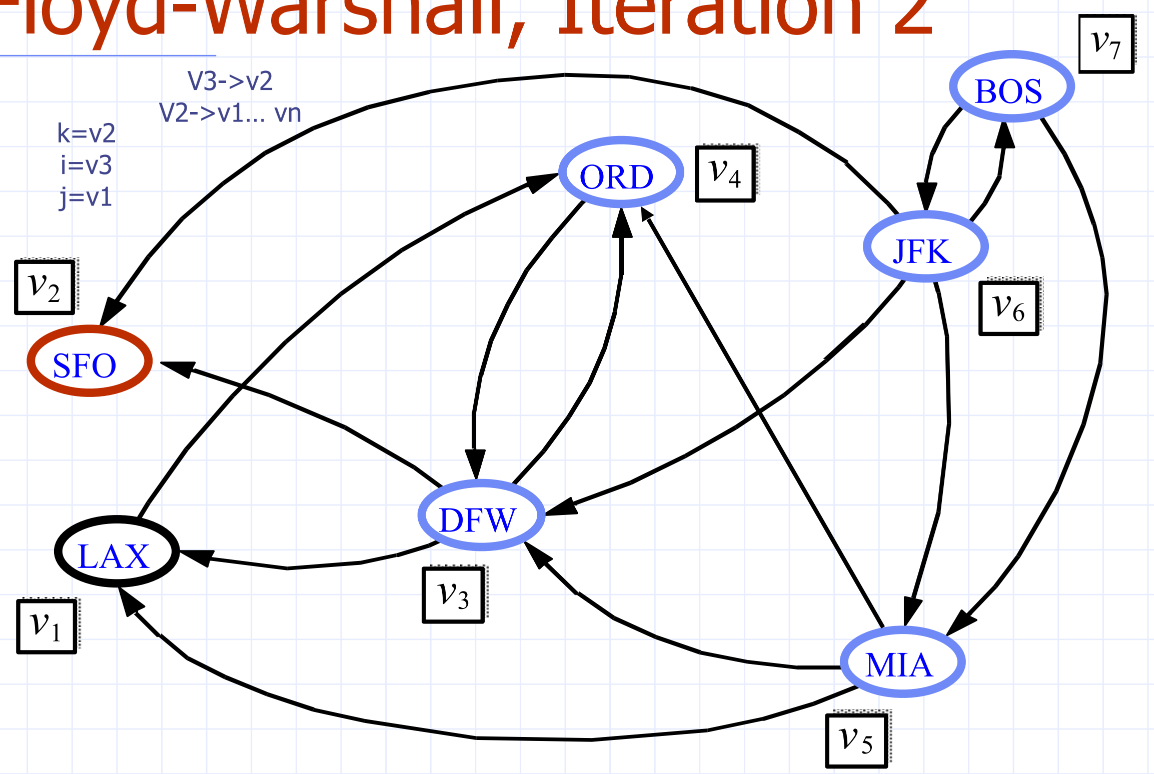
k=v1
i=v6,v7
j=v2



Floyd-Warshall, Iteration 2



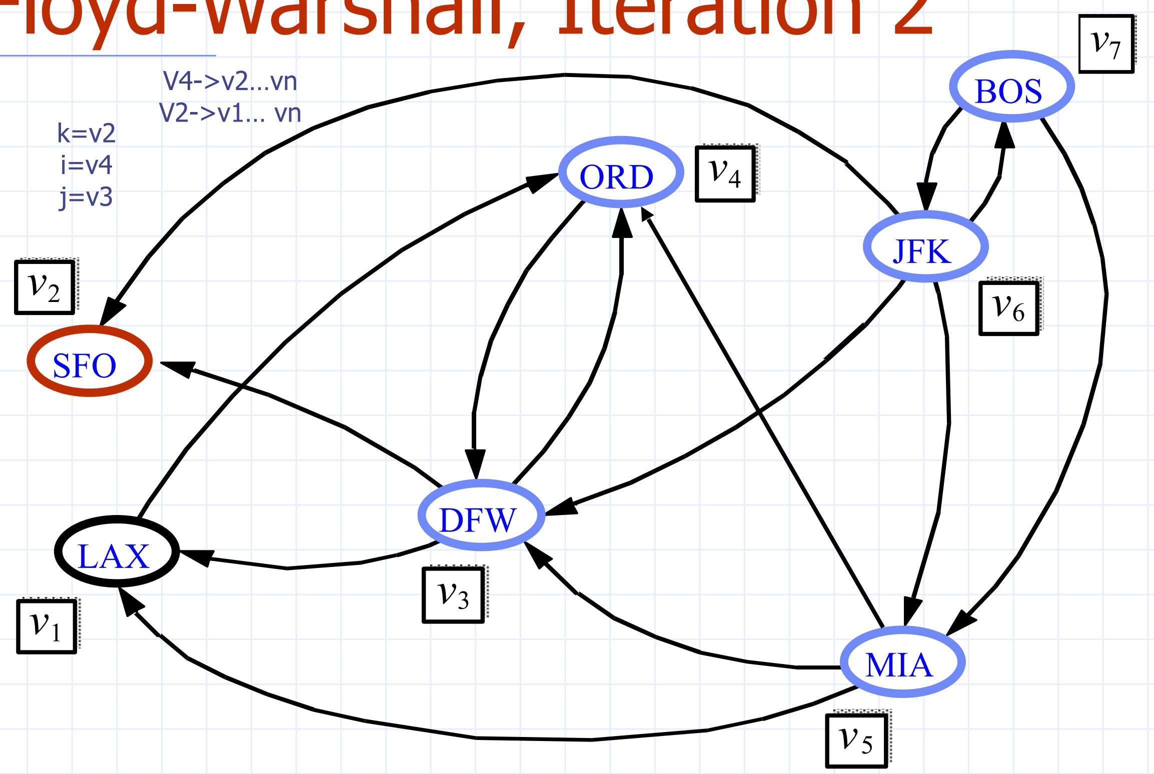
Floyd-Warshall, Iteration 2



Floyd-Warshall, Iteration 2

V4->v2…vn
V2->v1… vn

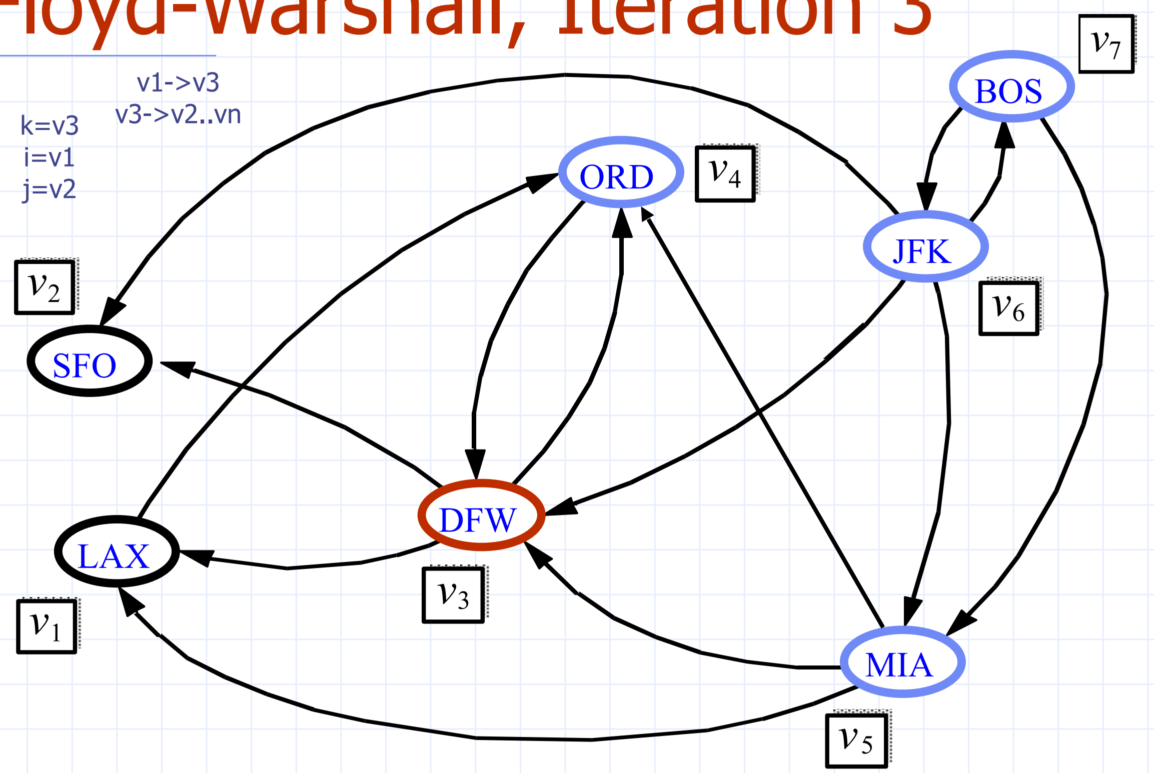
k=v2
i=v4
j=v3



Floyd-Warshall, Iteration 3

k=v3
i=v1
j=v2

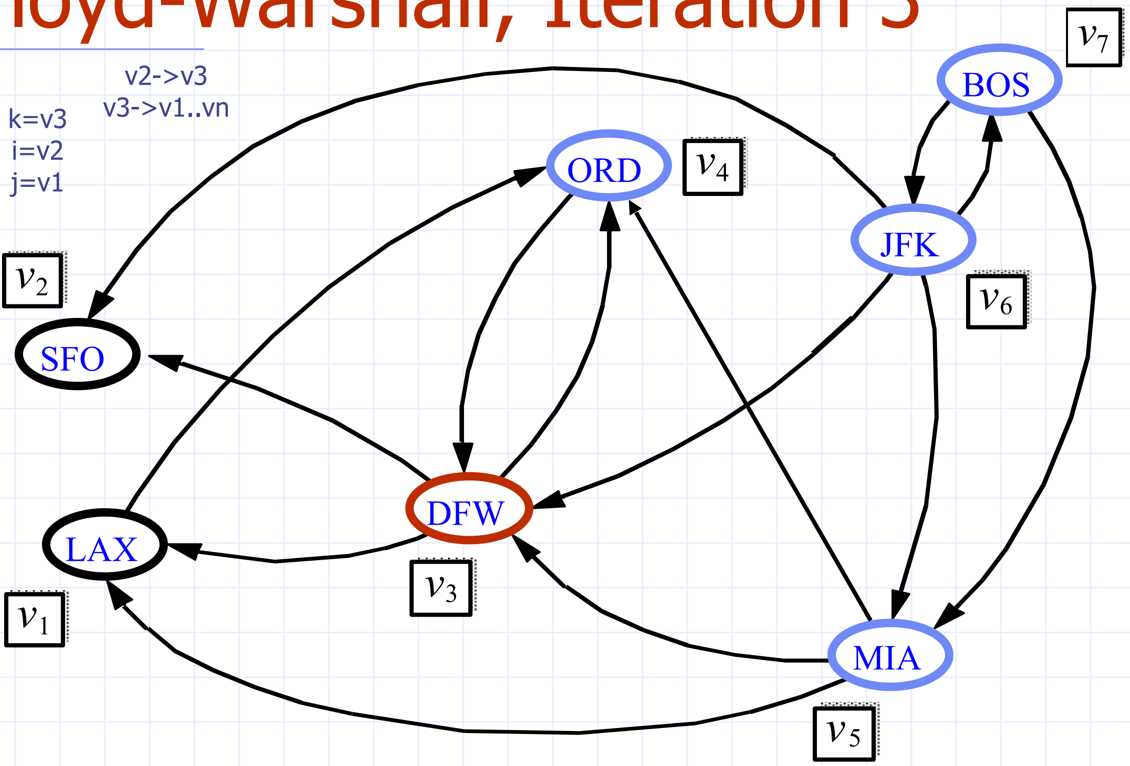
v1->v3
v3->v2..vn



Floyd-Warshall, Iteration 3

k=v3
i=v2
j=v1

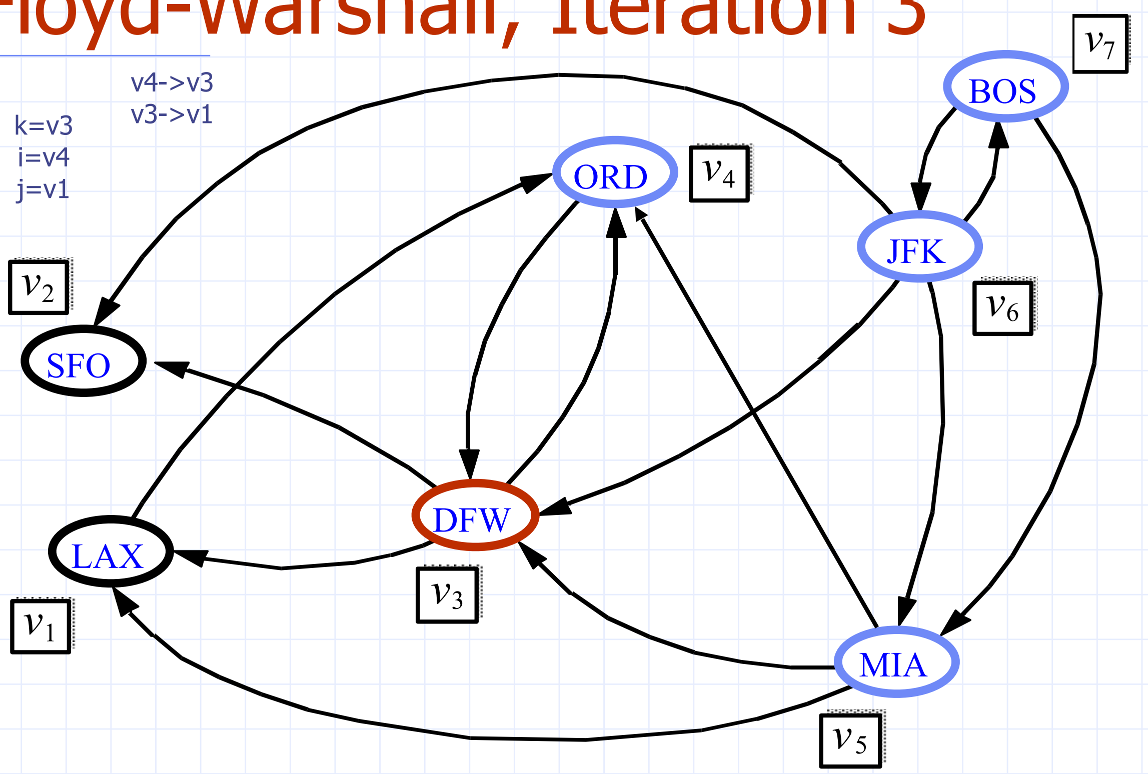
v2->v3
v3->v1..vn



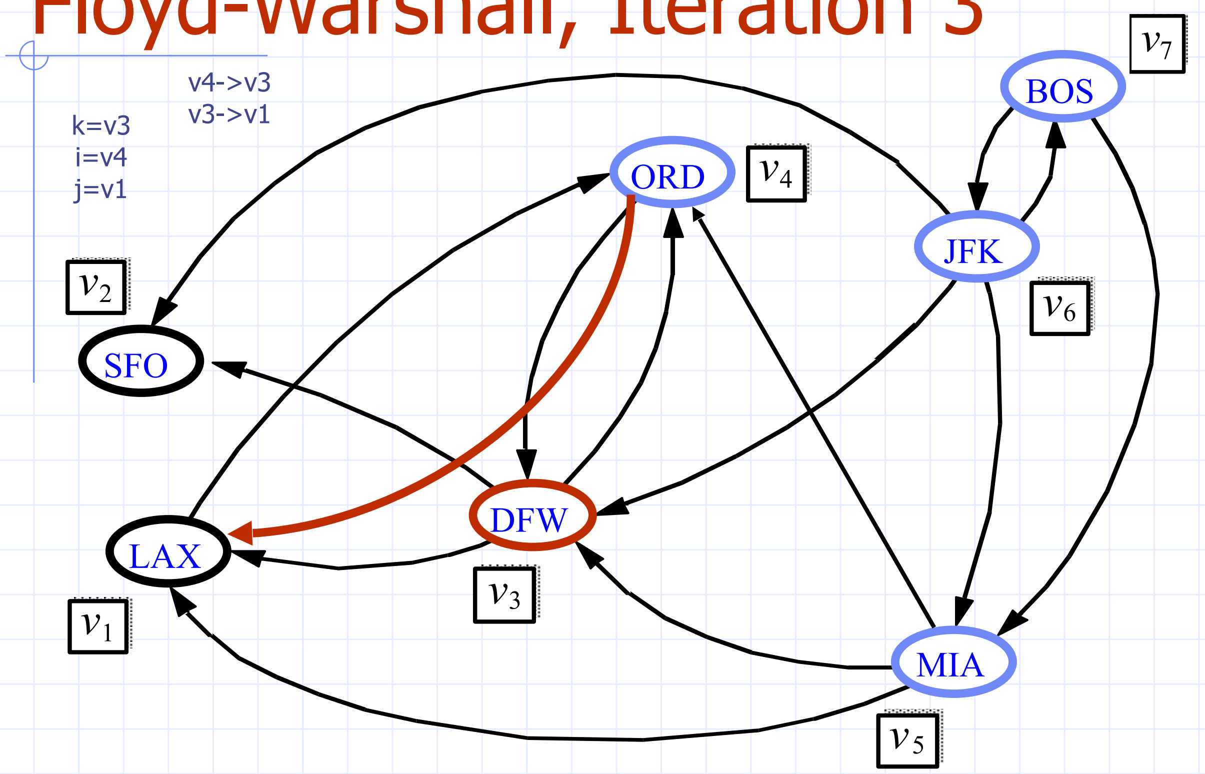
Floyd-Warshall, Iteration 3

k=v3
i=v4
j=v1

v4->v3
v3->v1



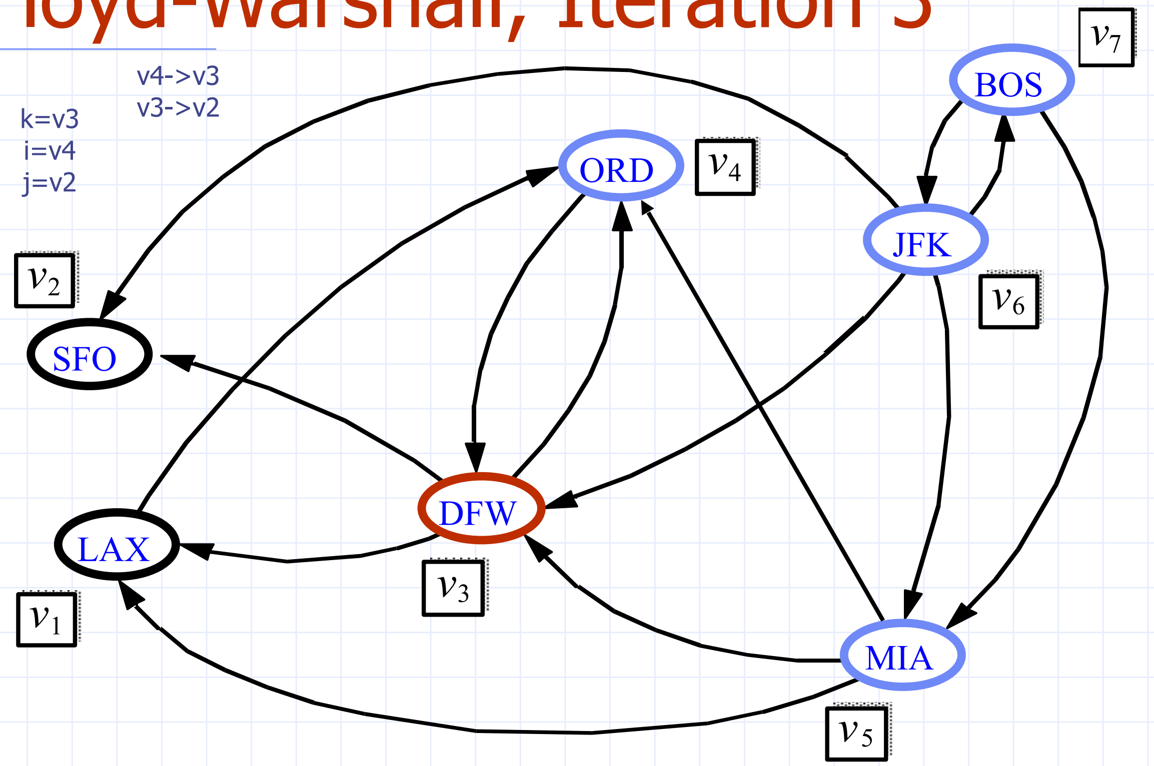
Floyd-Warshall, Iteration 3



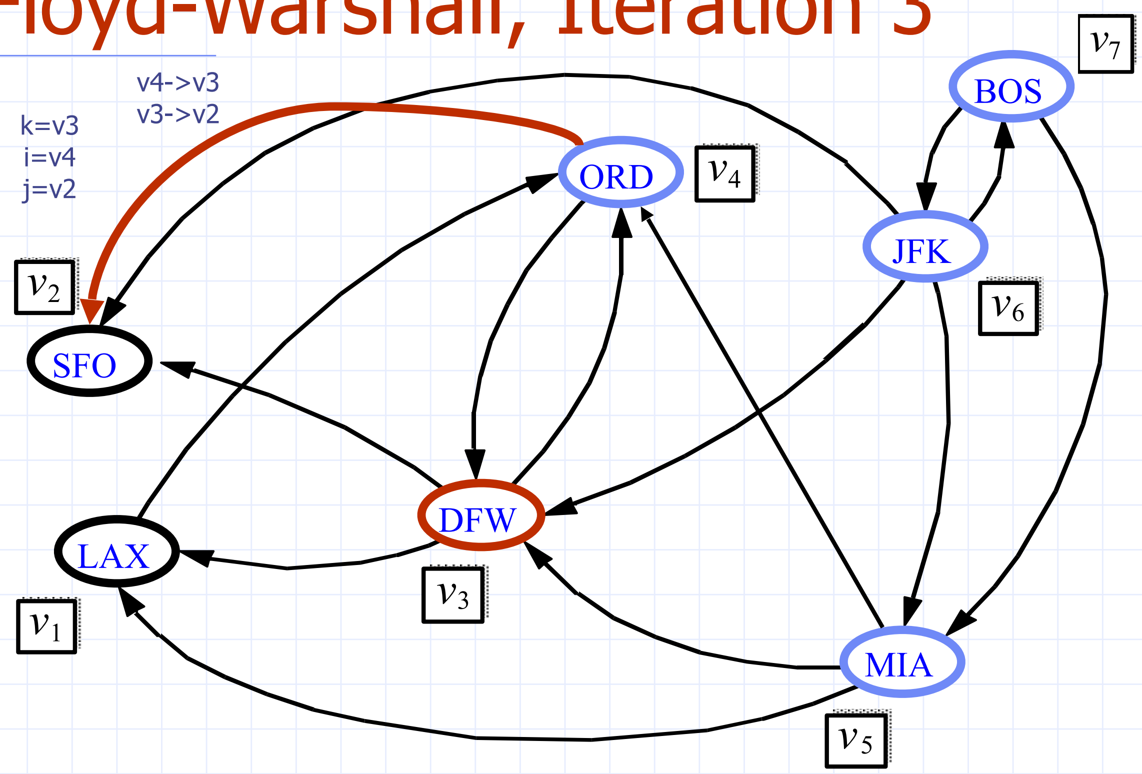
Floyd-Warshall, Iteration 3

k=v3
i=v4
j=v2

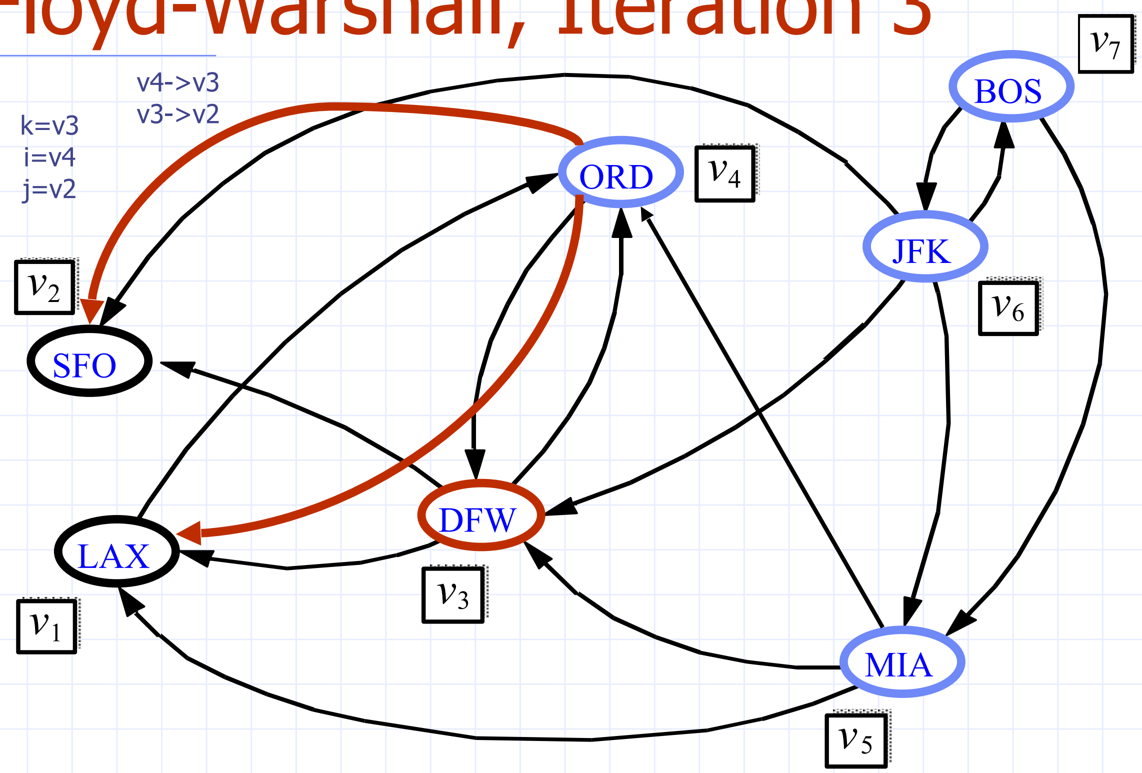
v4->v3
v3->v2



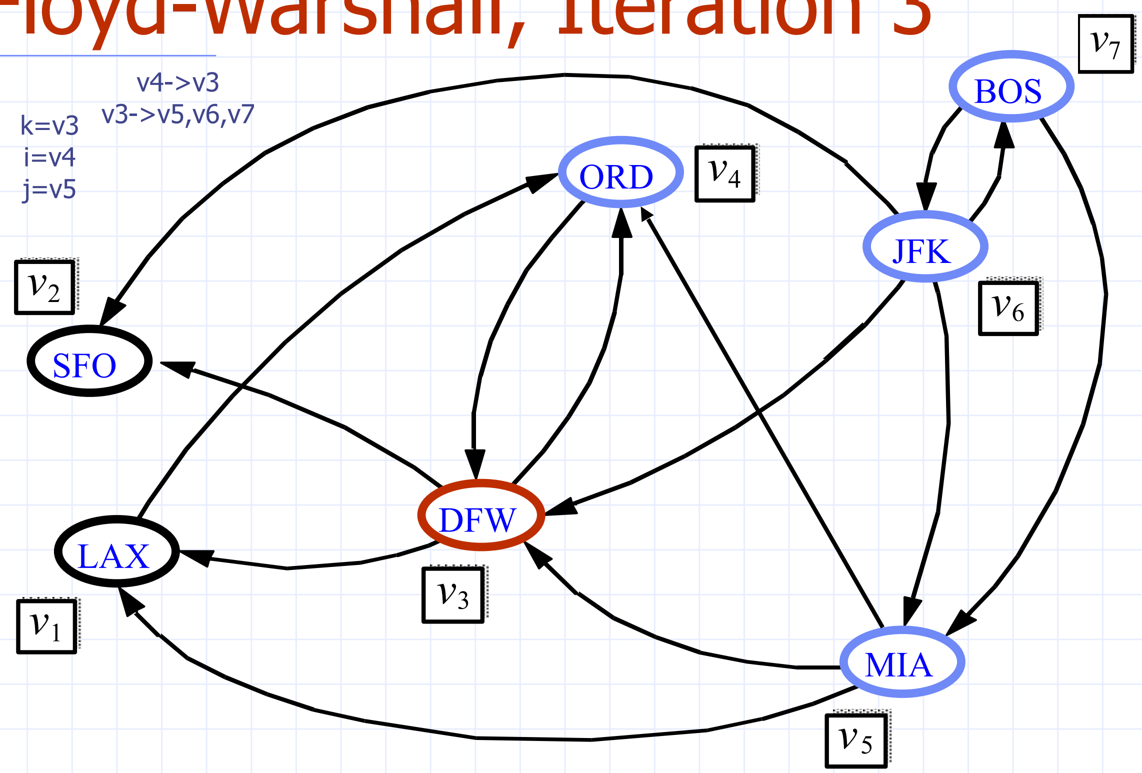
Floyd-Warshall, Iteration 3



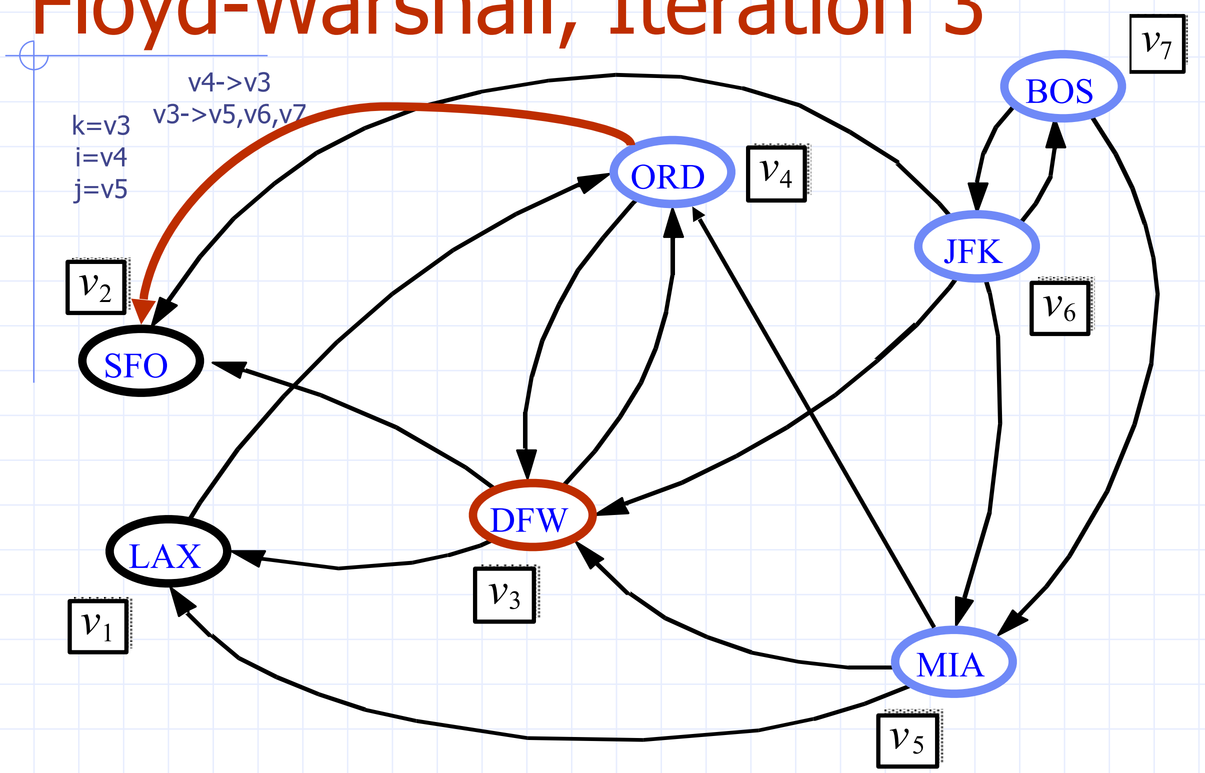
Floyd-Warshall, Iteration 3



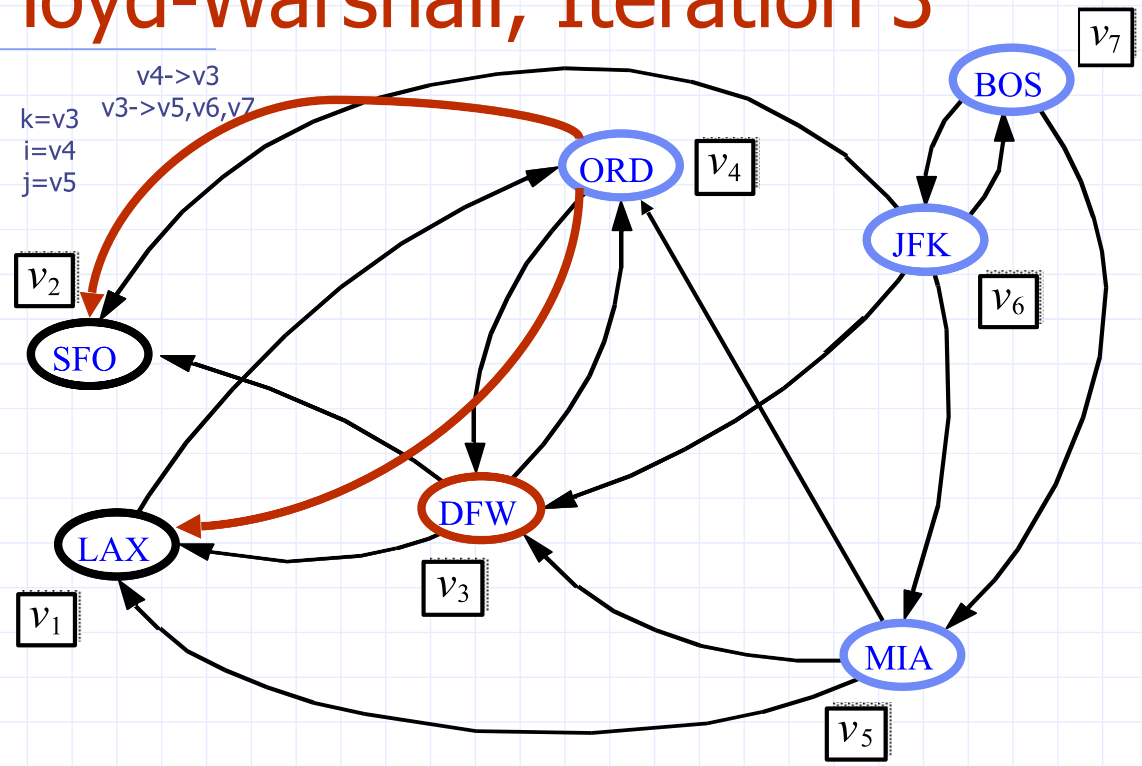
Floyd-Warshall, Iteration 3



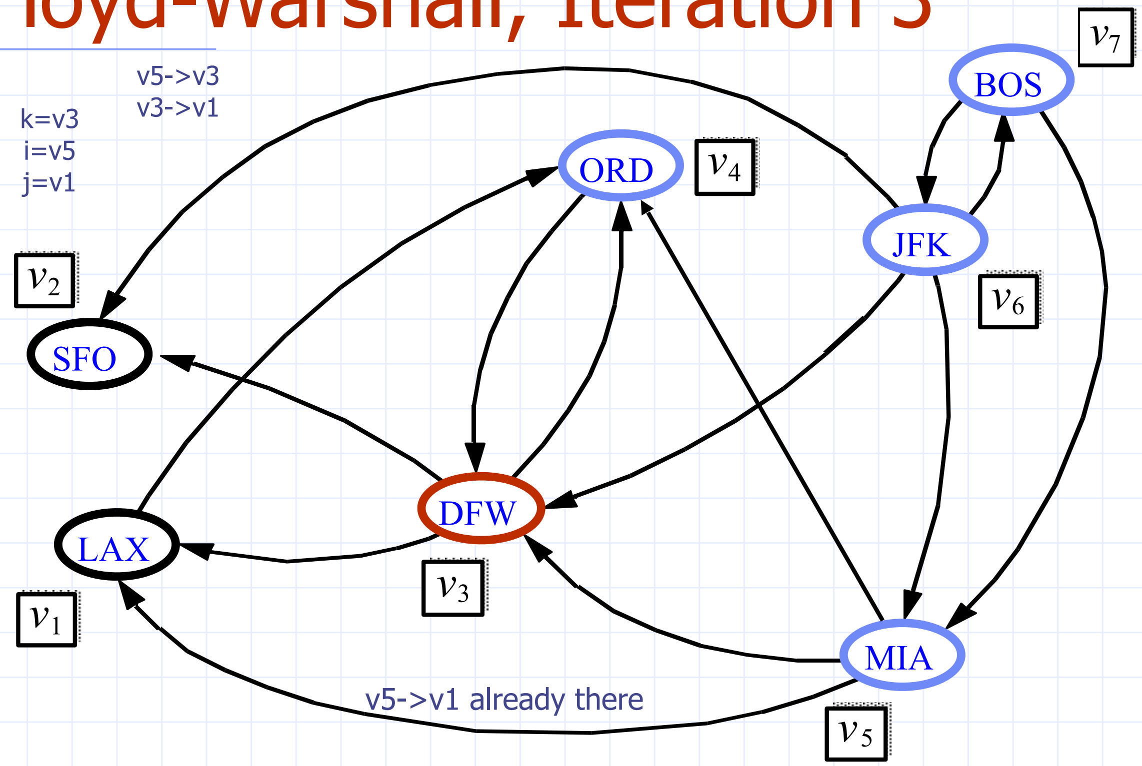
Floyd-Warshall, Iteration 3



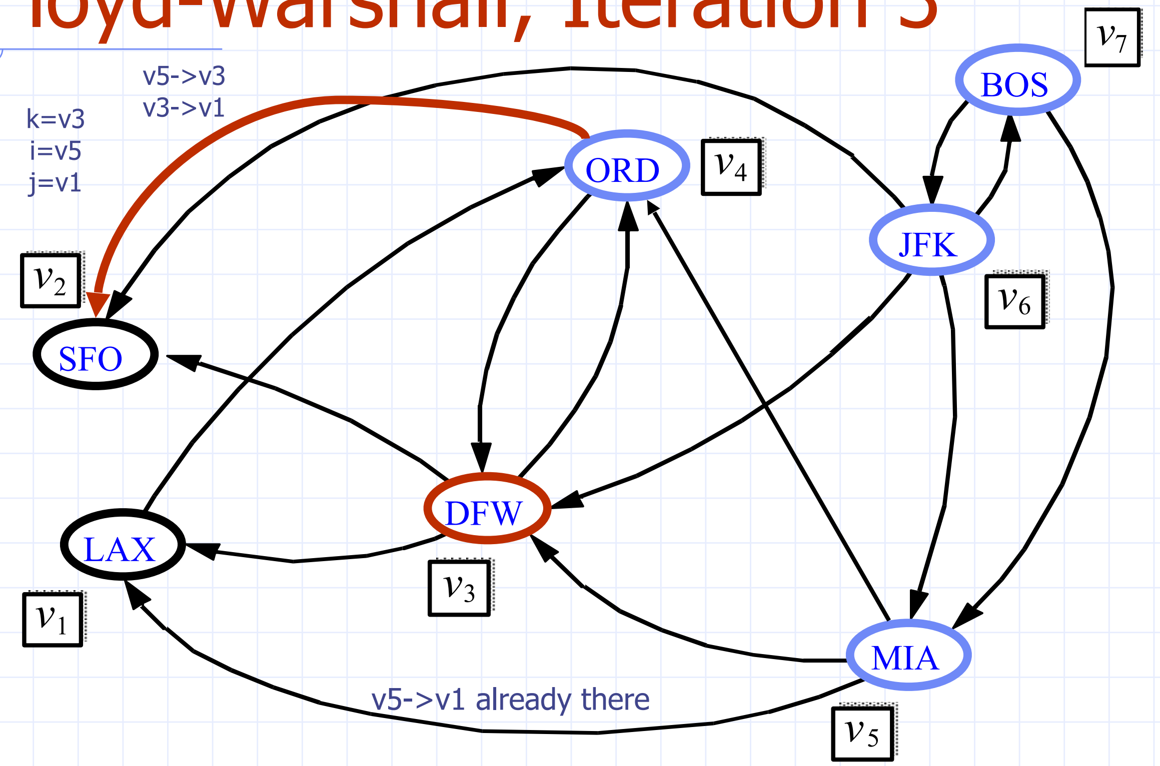
Floyd-Warshall, Iteration 3



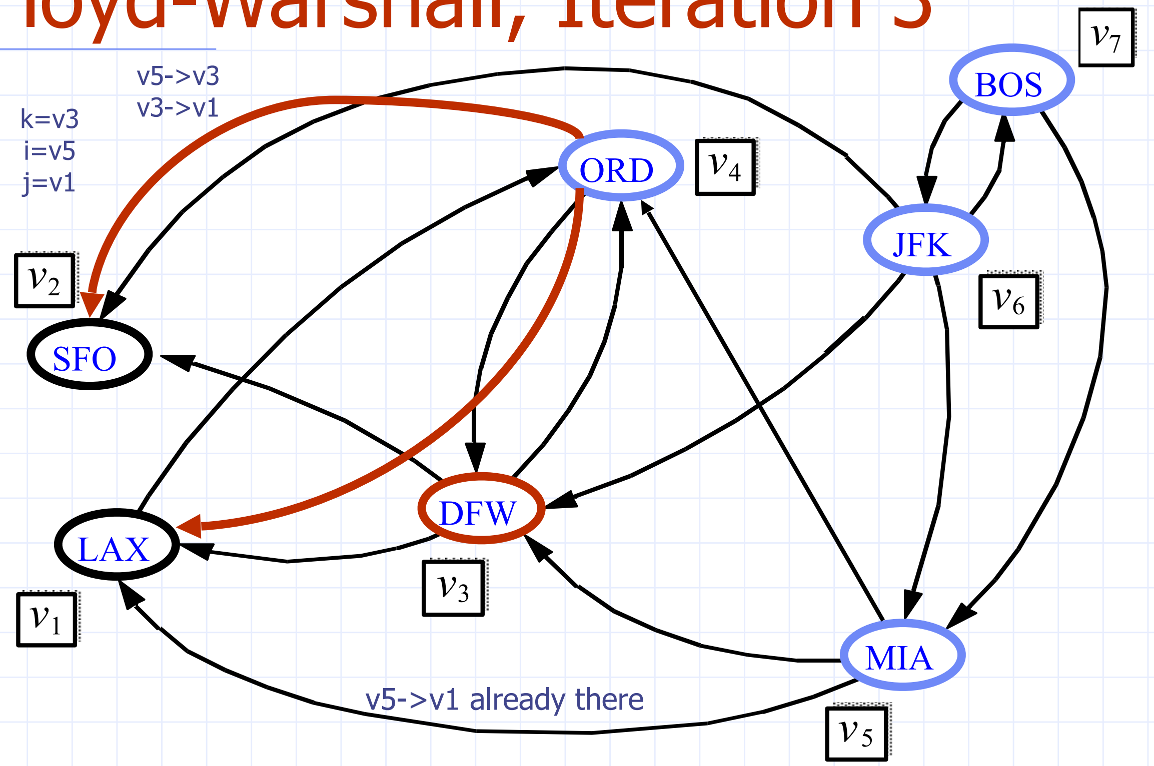
Floyd-Warshall, Iteration 3



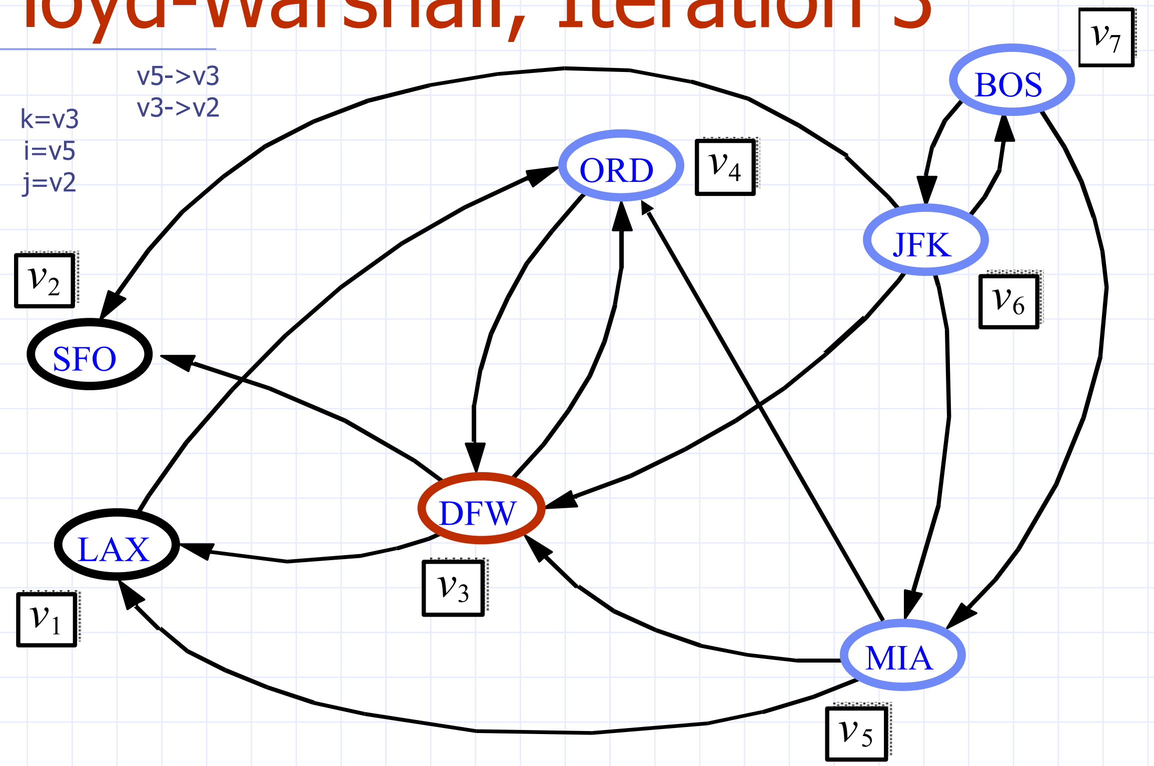
Floyd-Warshall, Iteration 3



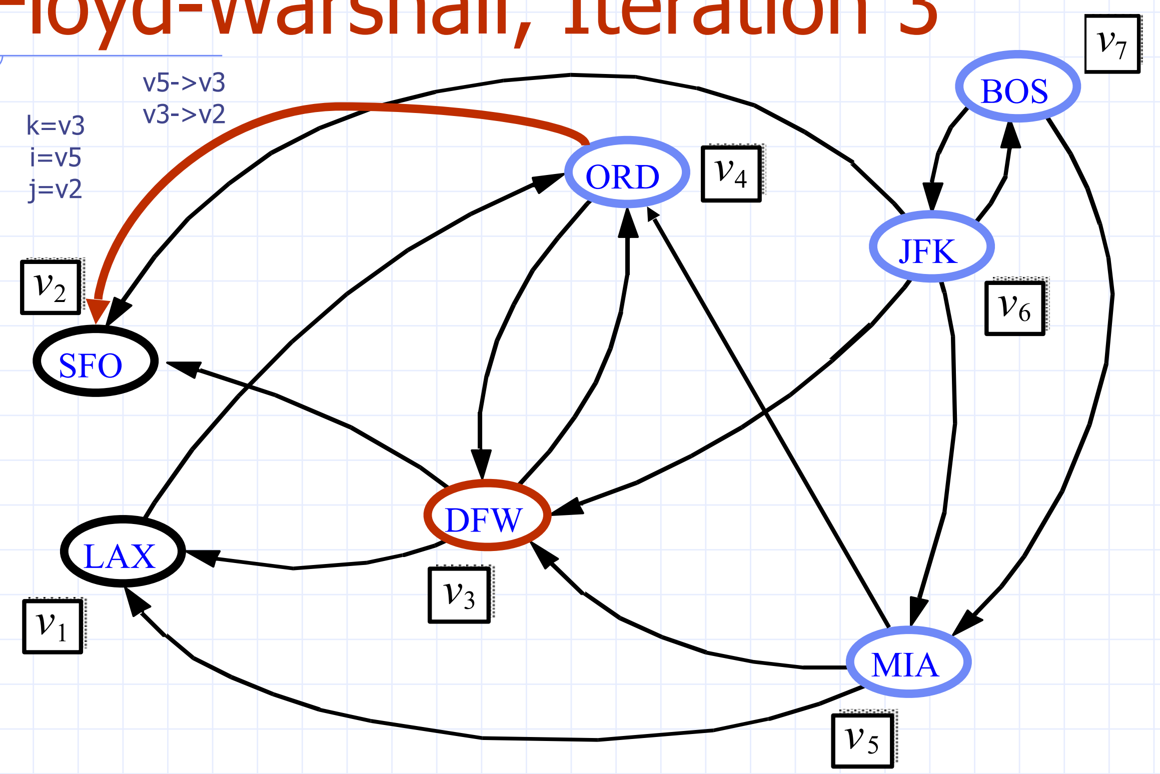
Floyd-Warshall, Iteration 3



Floyd-Warshall, Iteration 3



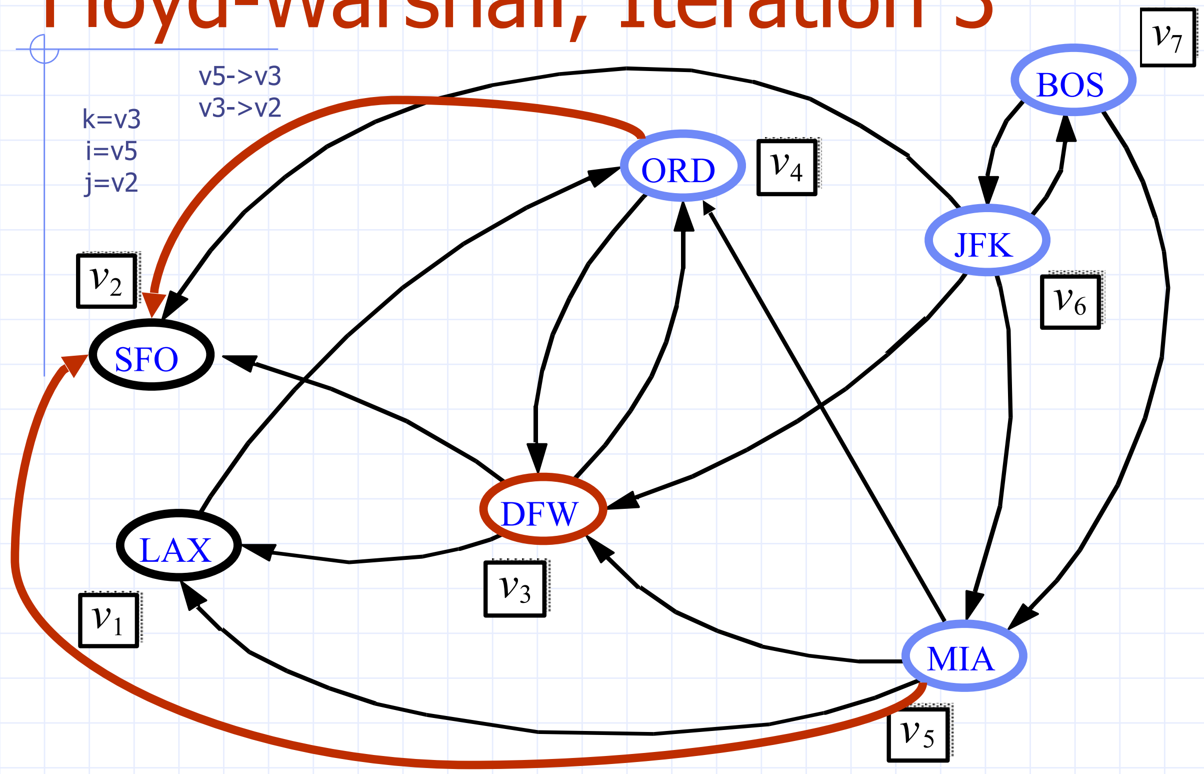
Floyd-Warshall, Iteration 3



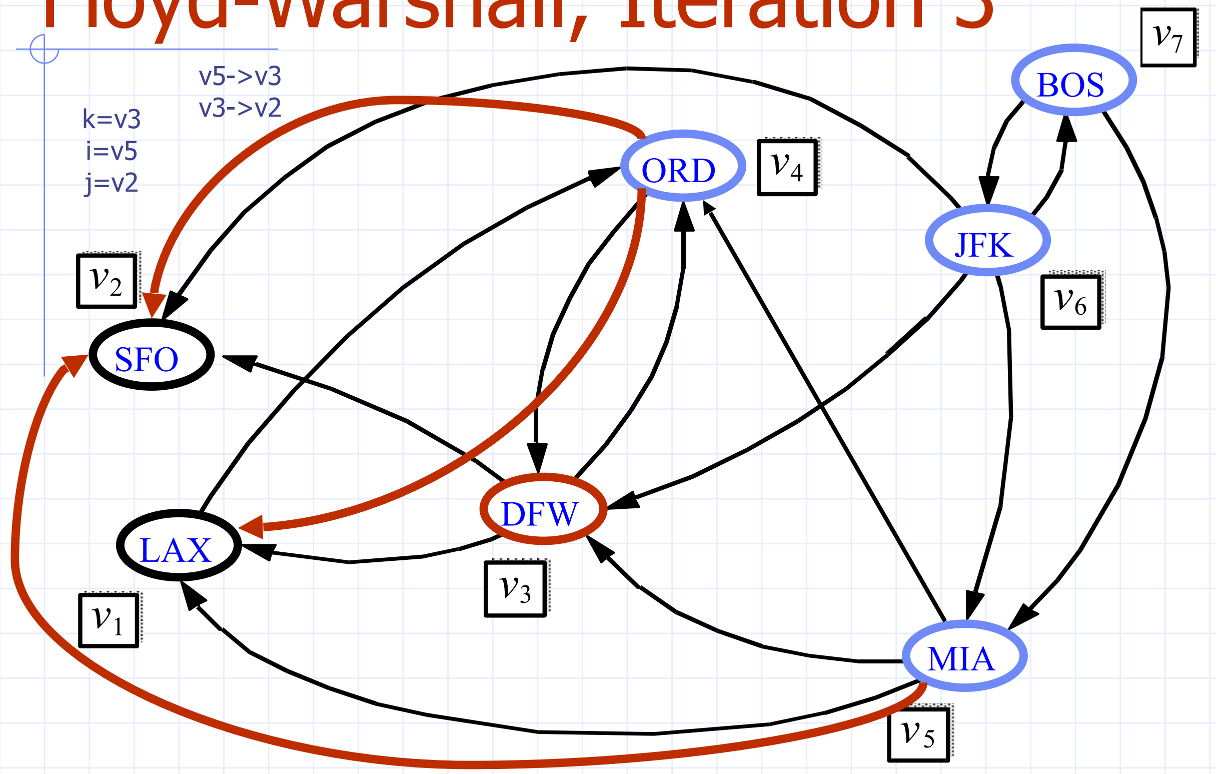
Floyd-Warshall, Iteration 3

k=v3
i=v5
j=v2

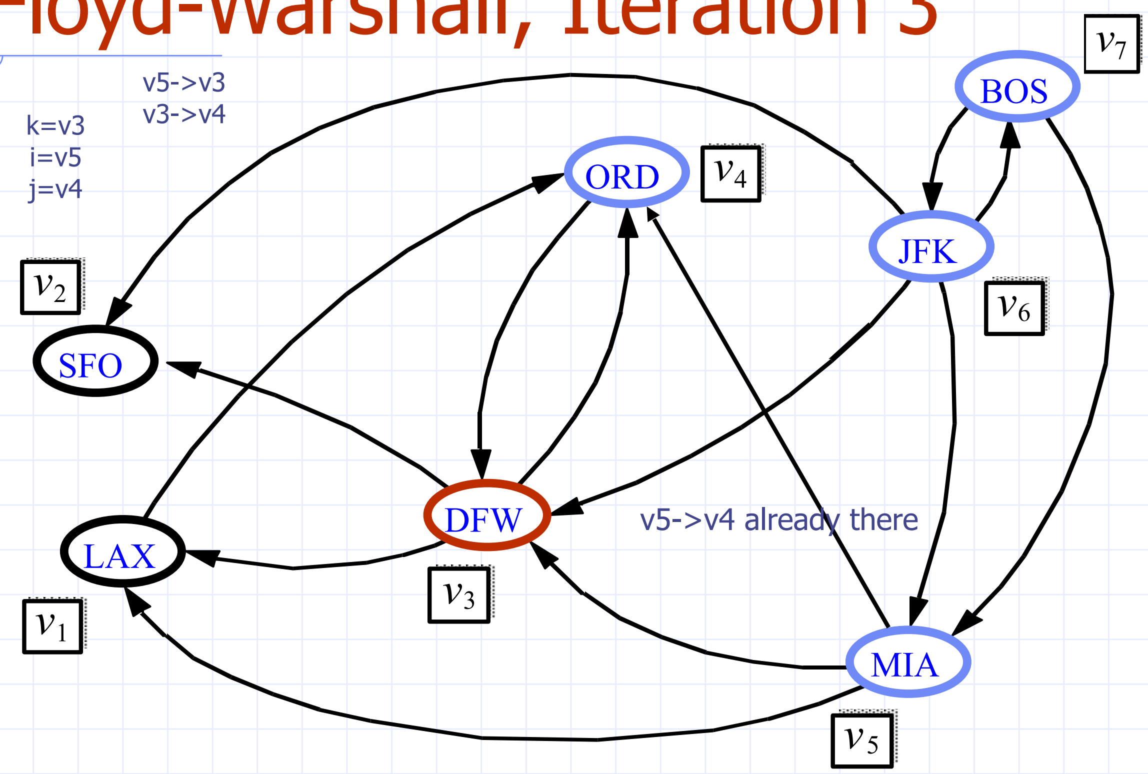
v5->v3
v3->v2



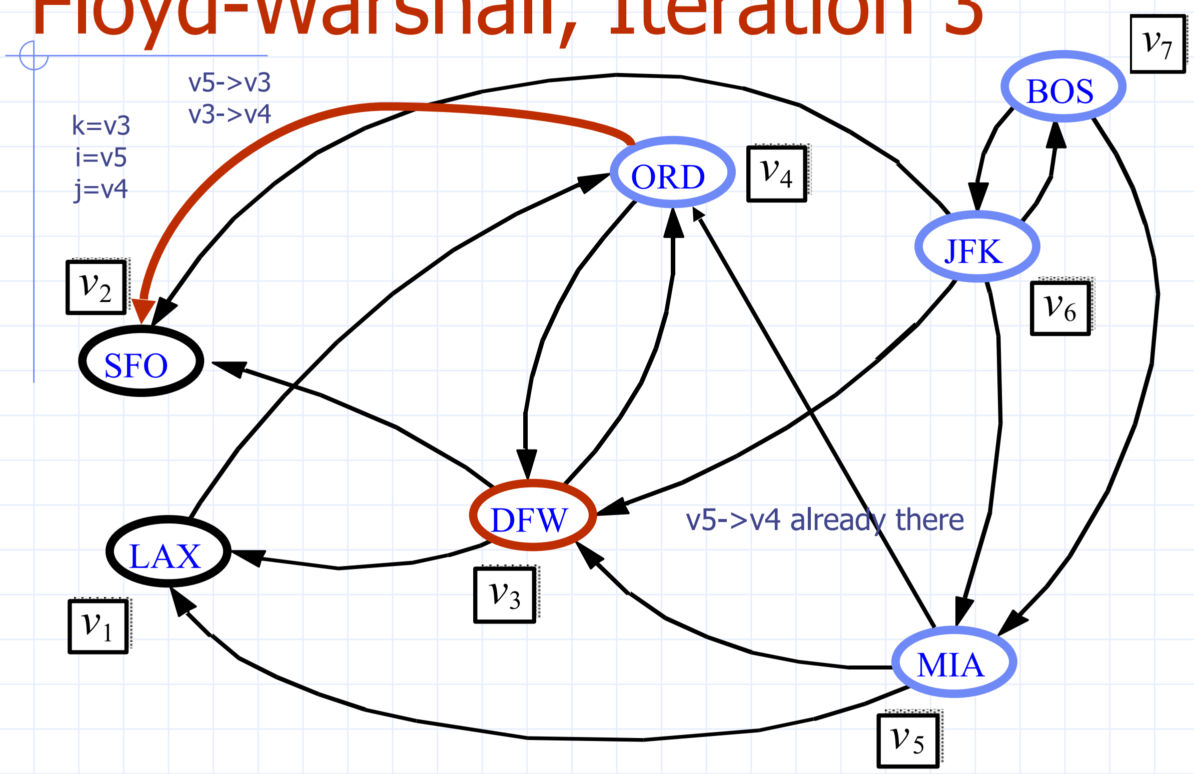
Floyd-Warshall, Iteration 3



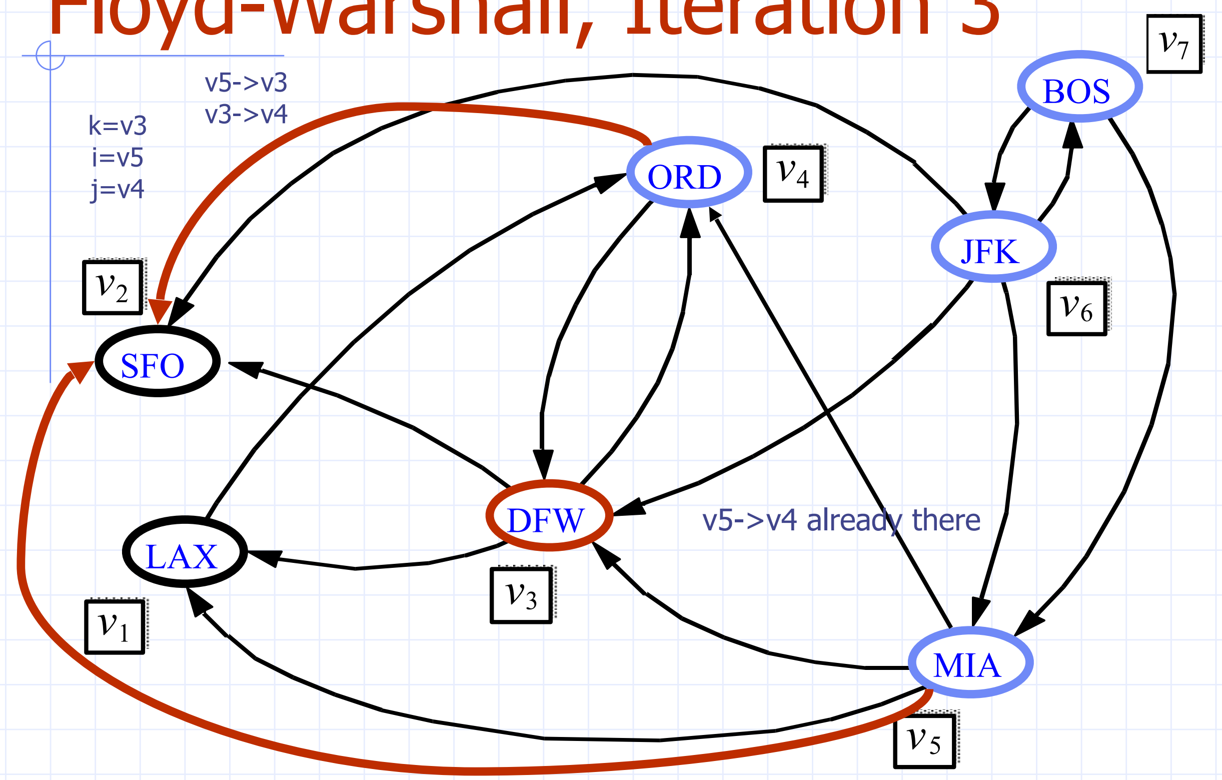
Floyd-Warshall, Iteration 3



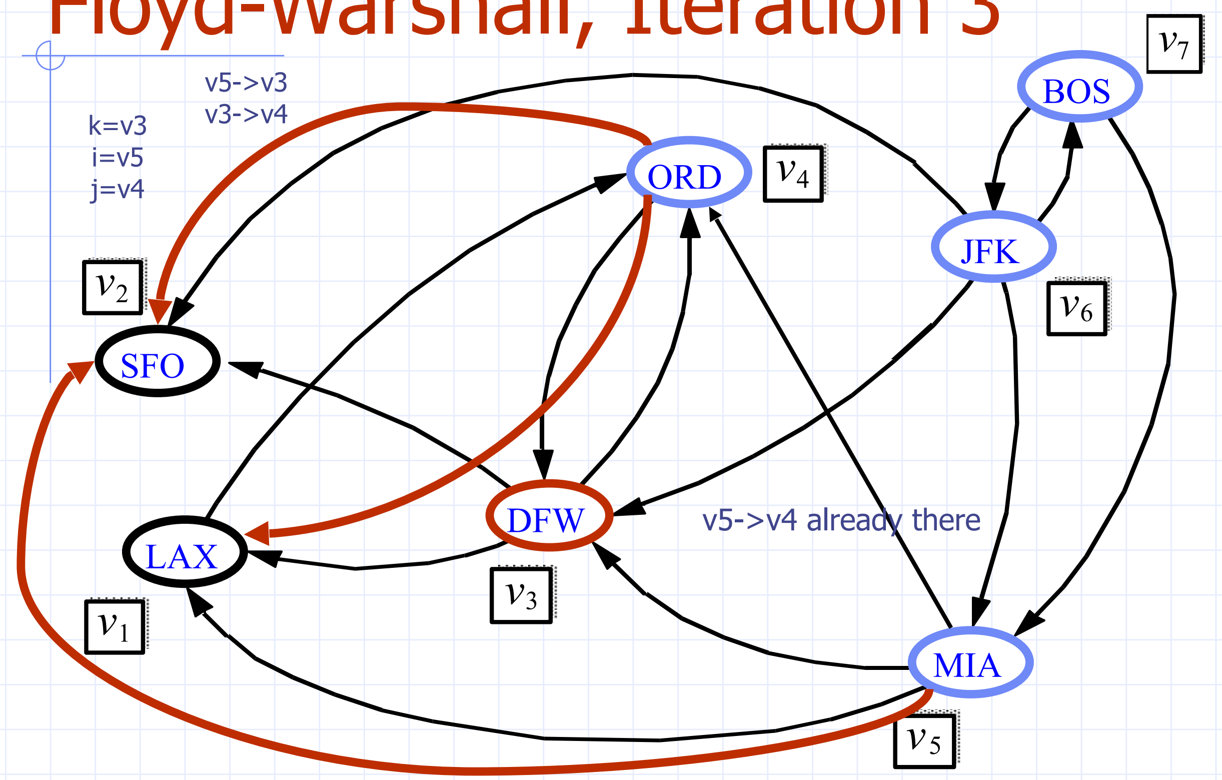
Floyd-Warshall, Iteration 3



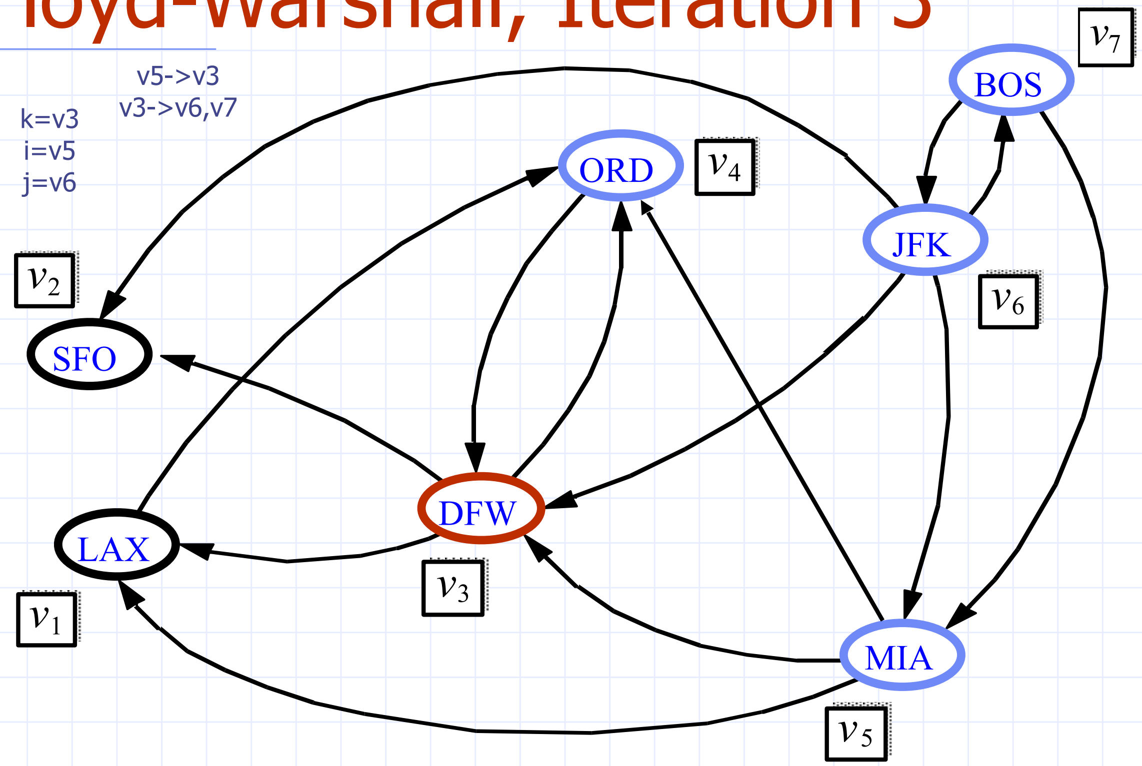
Floyd-Warshall, Iteration 3



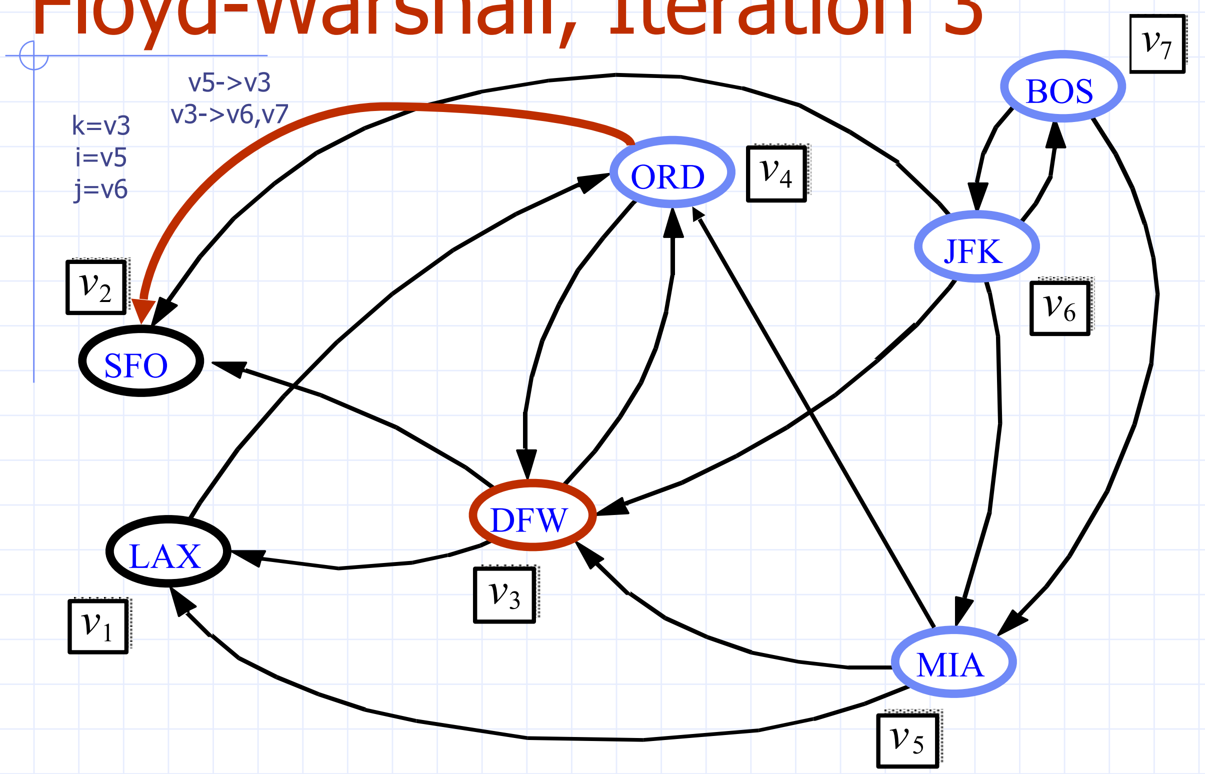
Floyd-Warshall, Iteration 3



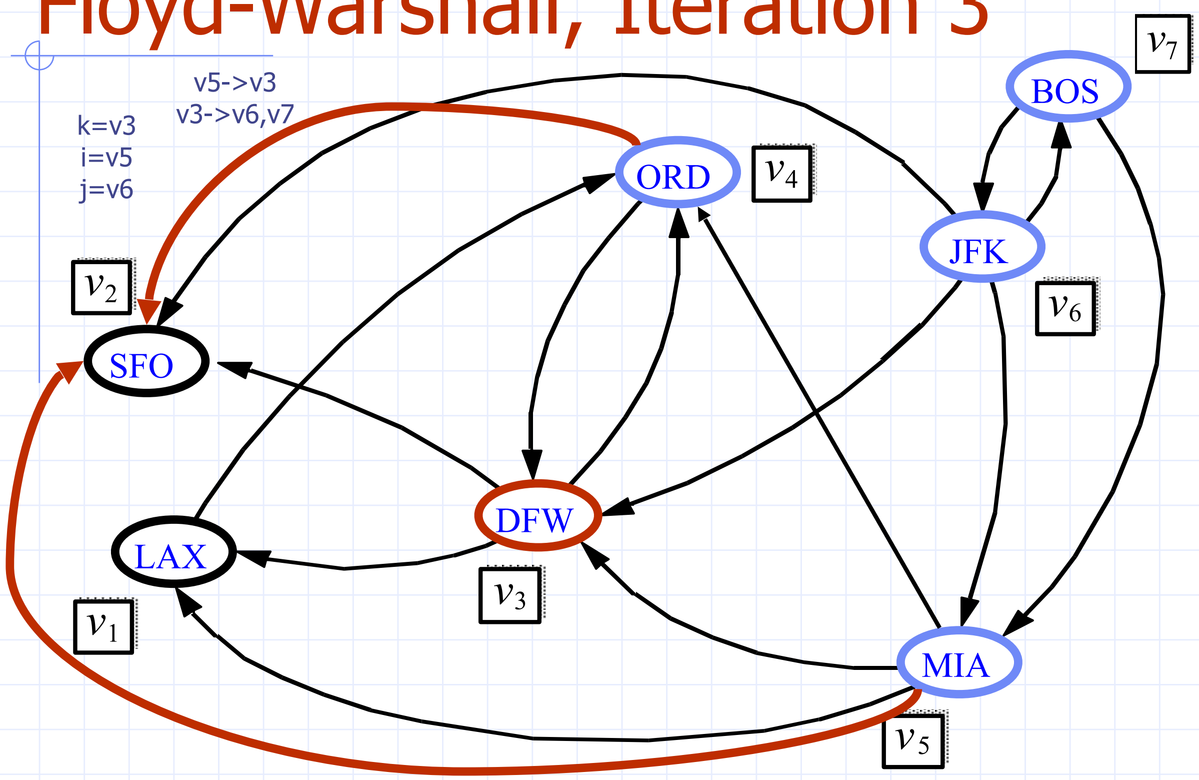
Floyd-Warshall, Iteration 3



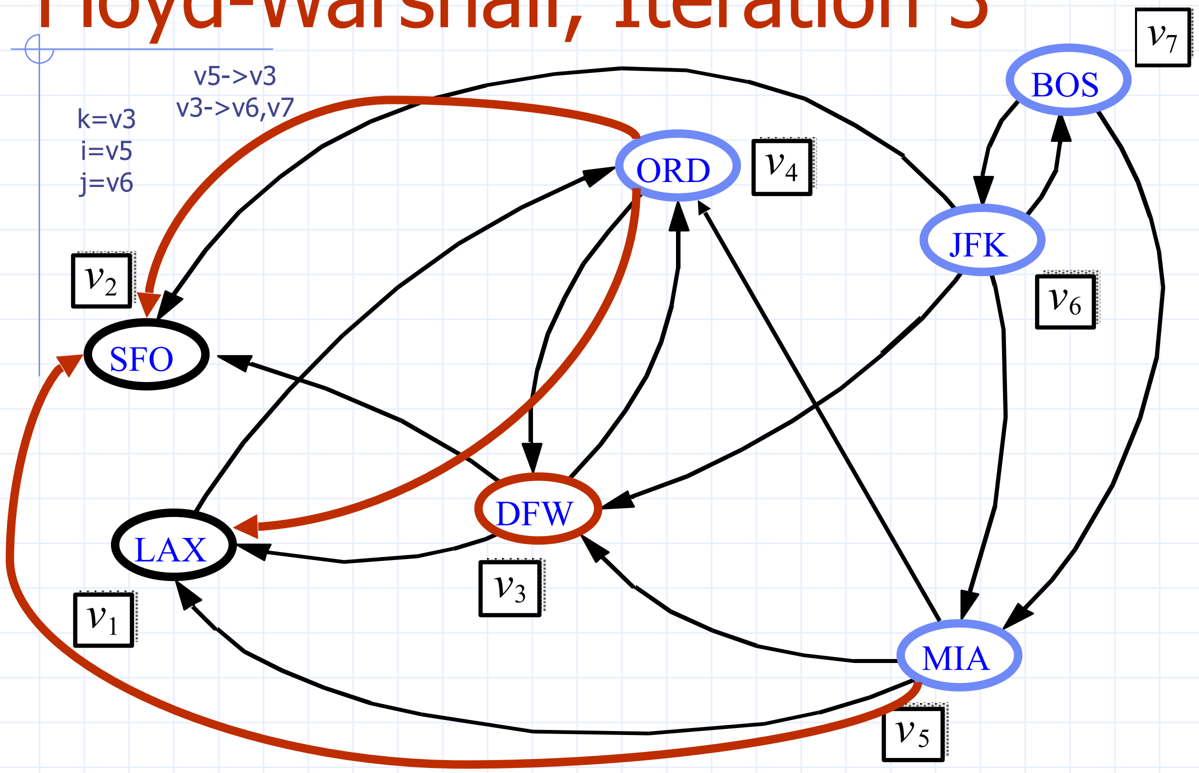
Floyd-Warshall, Iteration 3



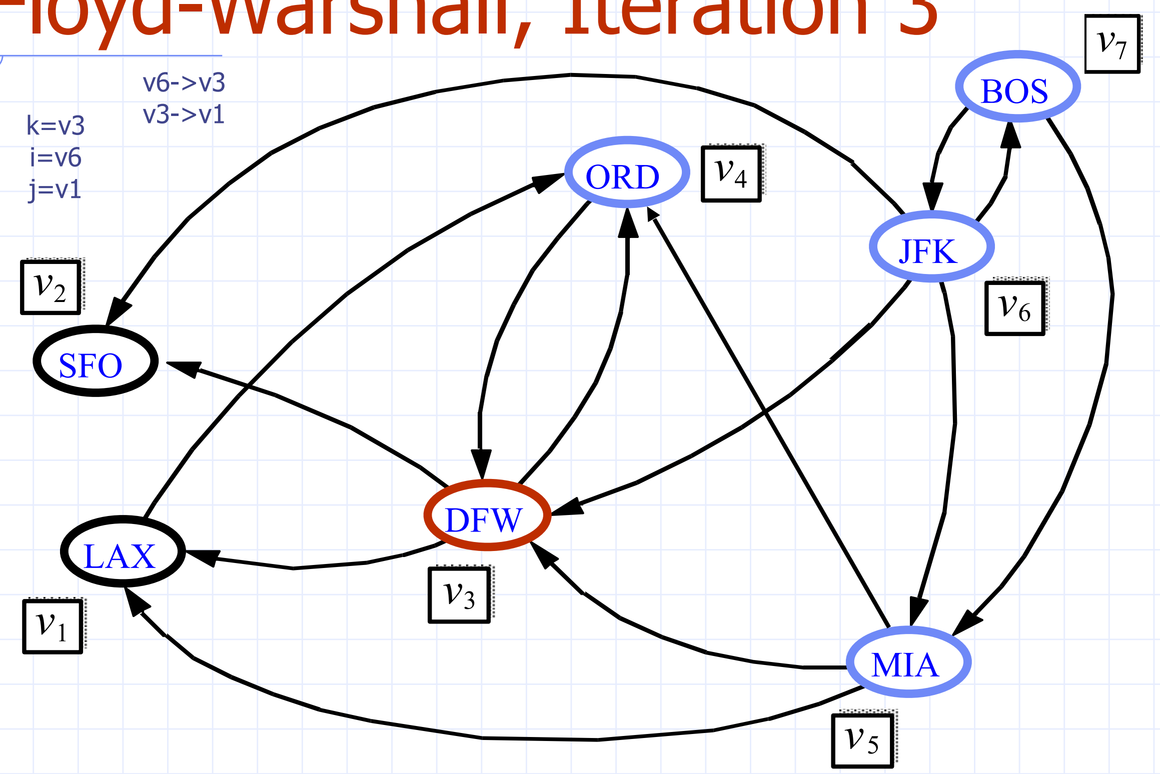
Floyd-Warshall, Iteration 3



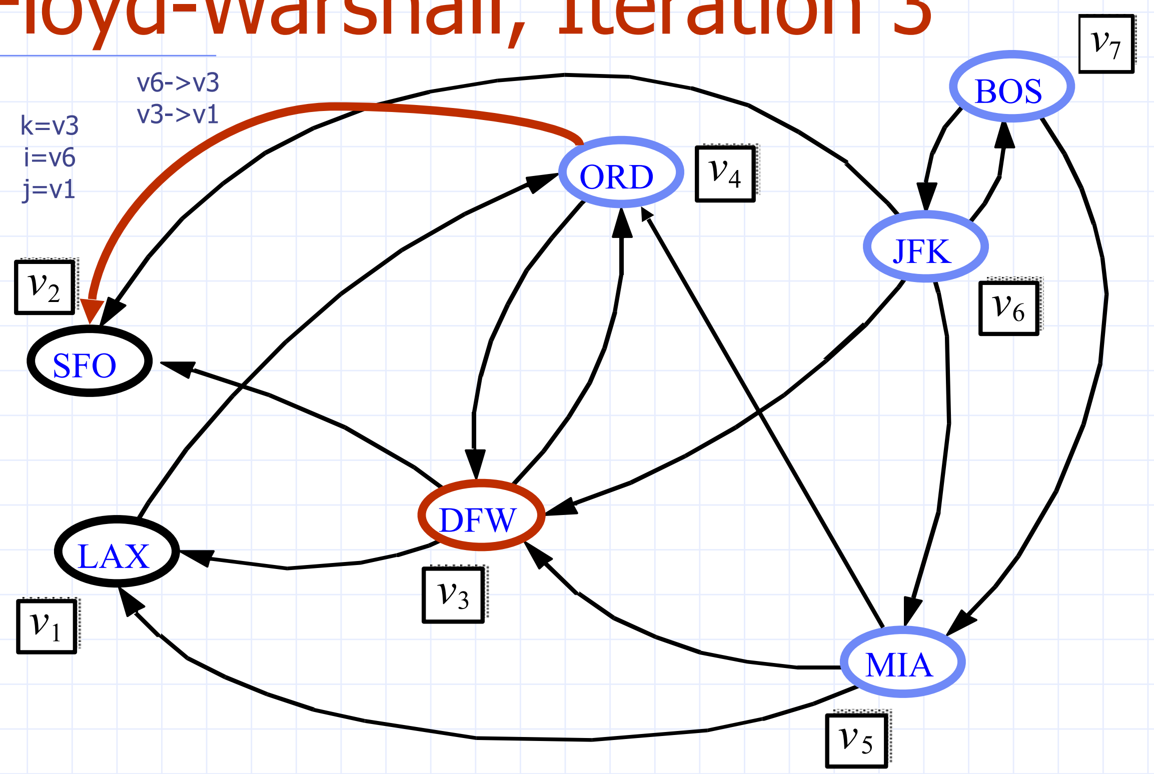
Floyd-Warshall, Iteration 3



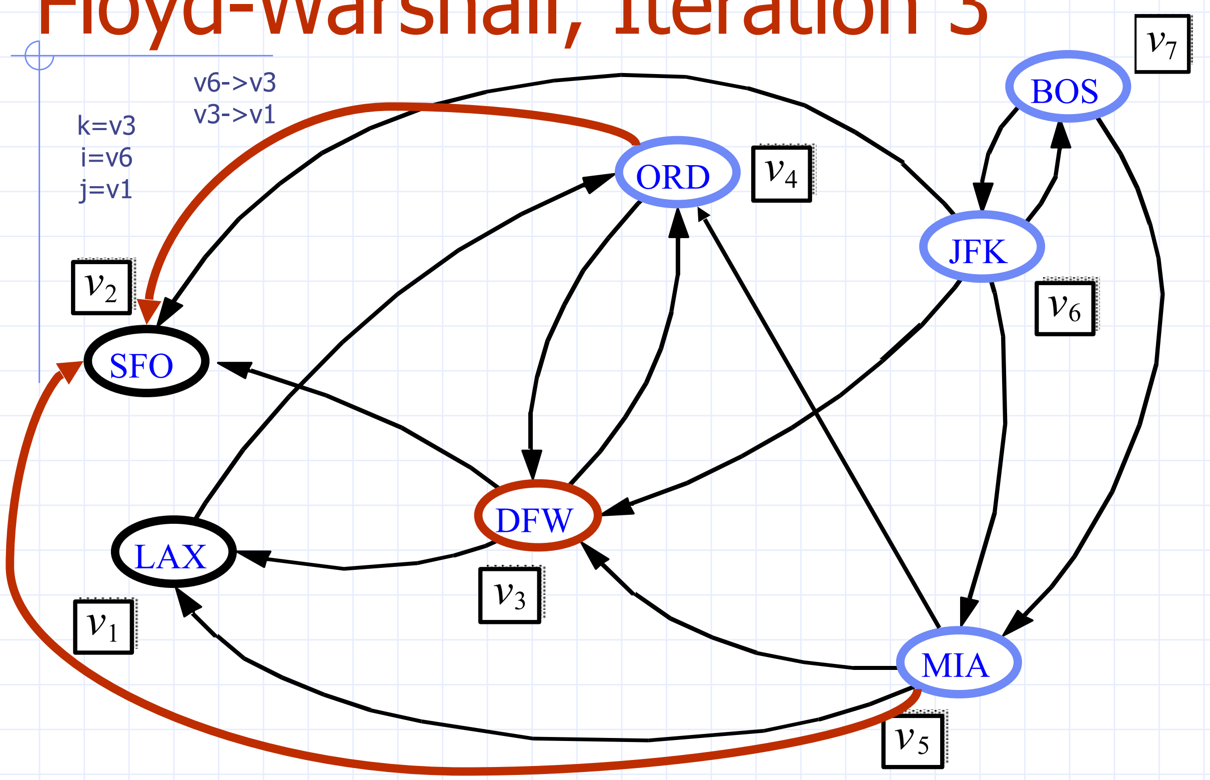
Floyd-Warshall, Iteration 3



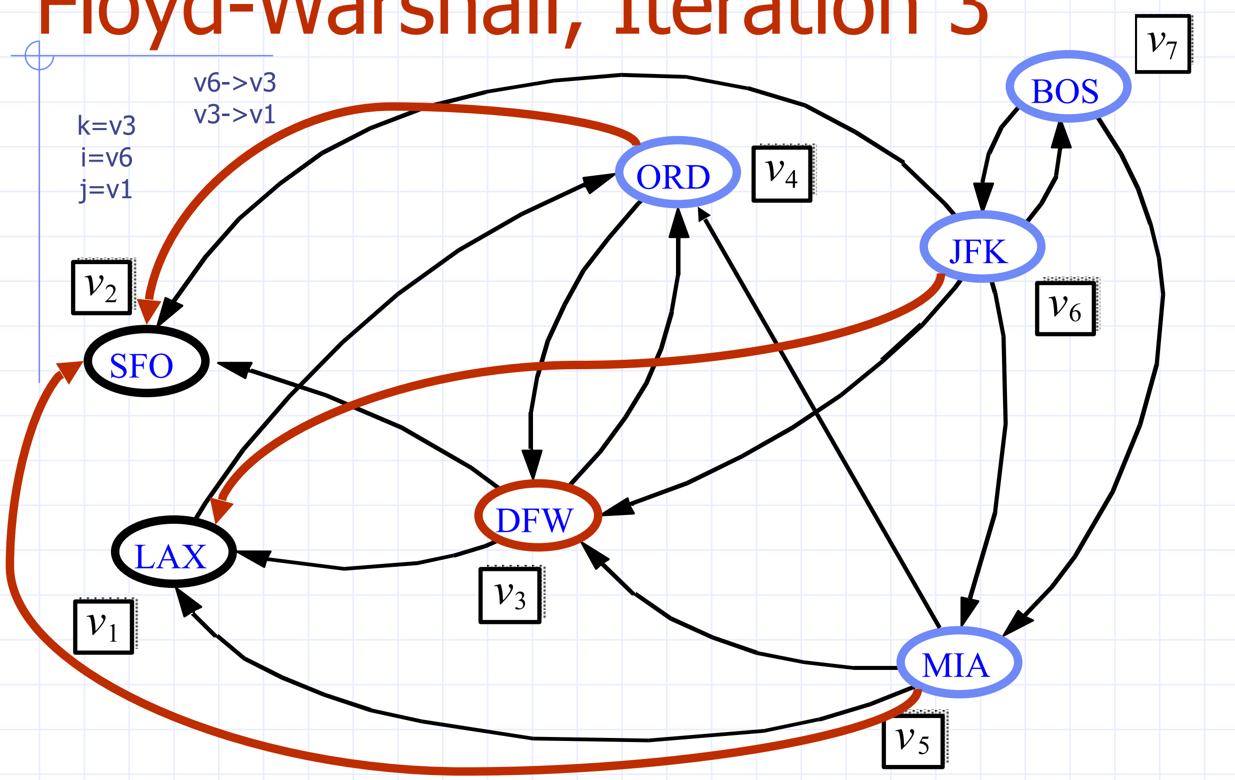
Floyd-Warshall, Iteration 3



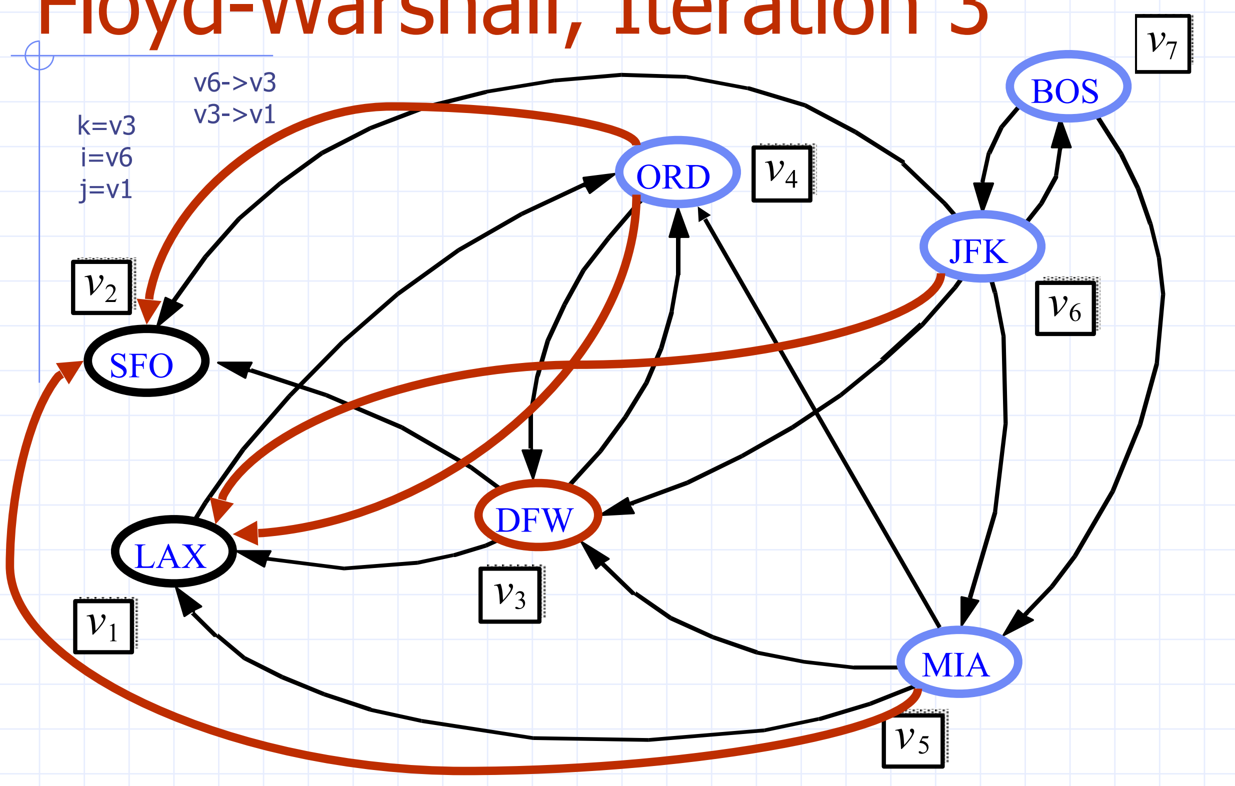
Floyd-Warshall, Iteration 3



Floyd-Warshall, Iteration 3



Floyd-Warshall, Iteration 3

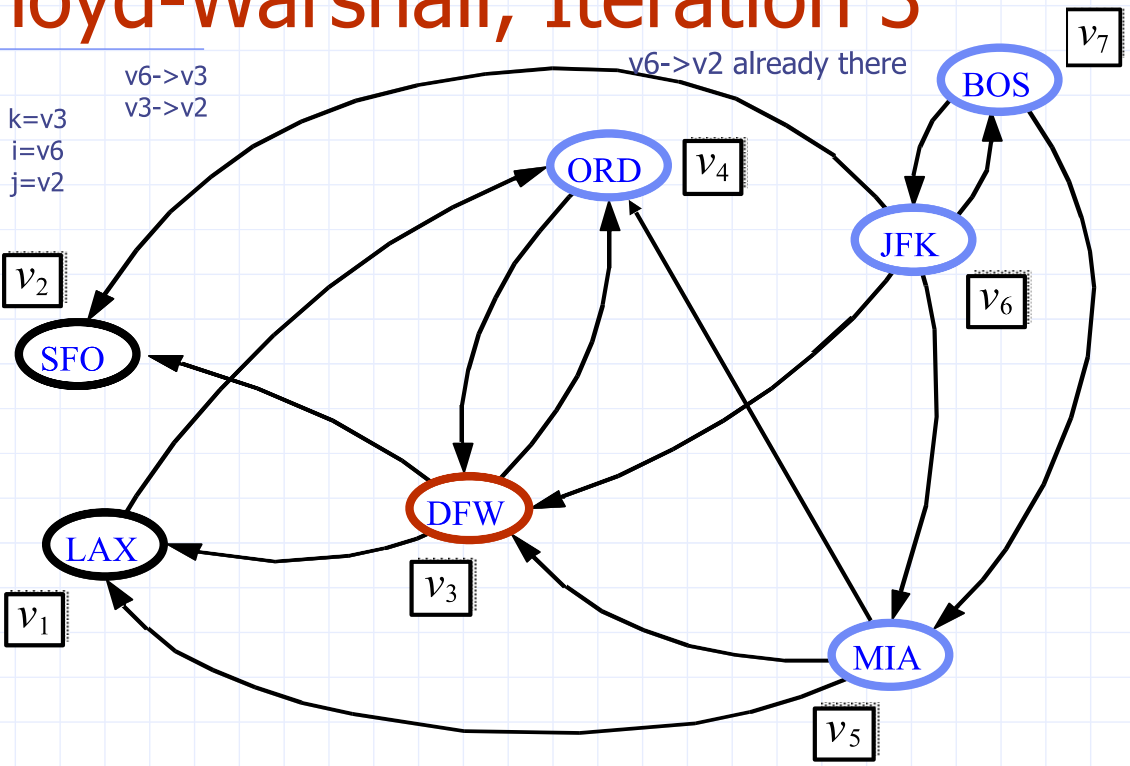


Floyd-Warshall, Iteration 3

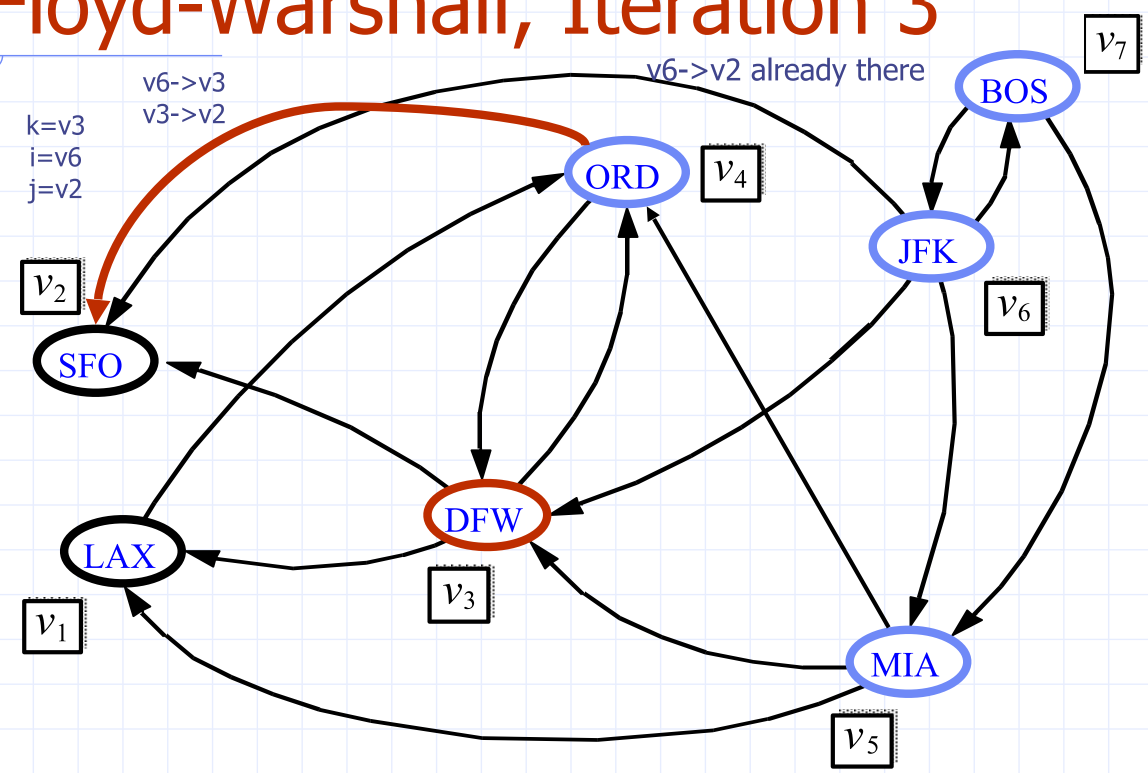
k=v3
i=v6
j=v2

v6->v3
v3->v2

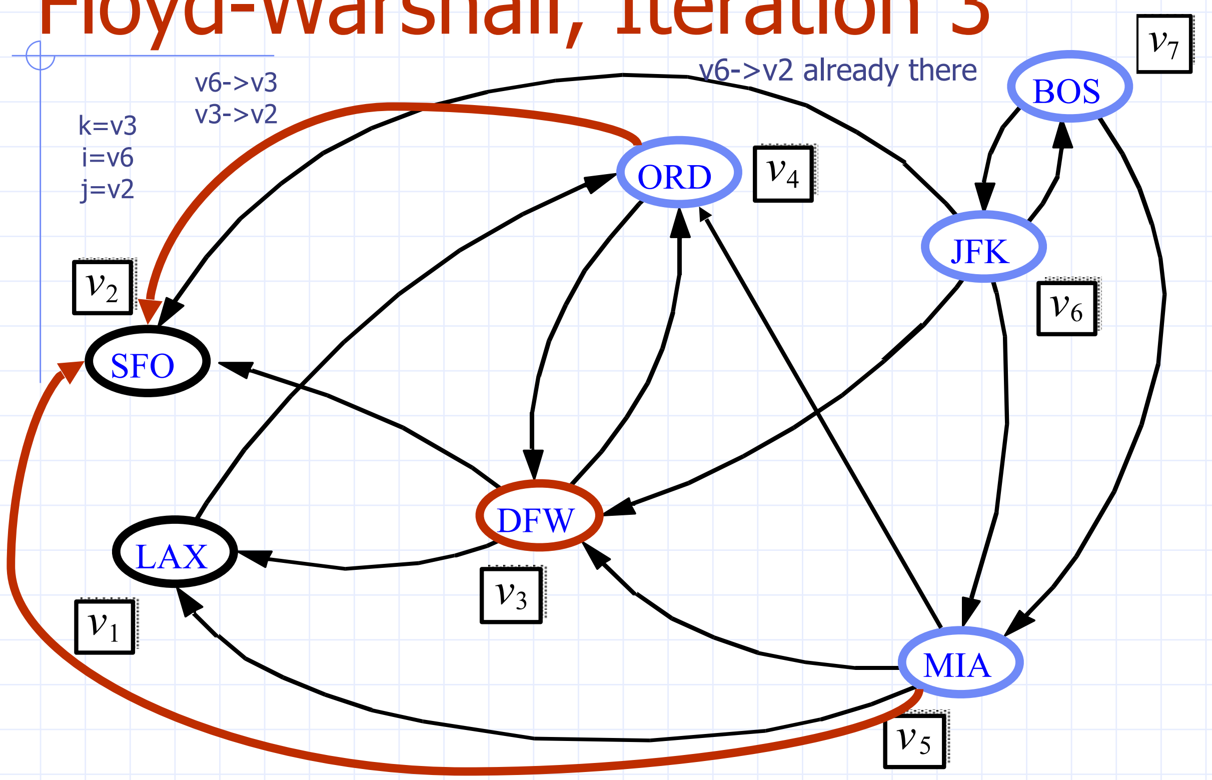
v6->v2 already there



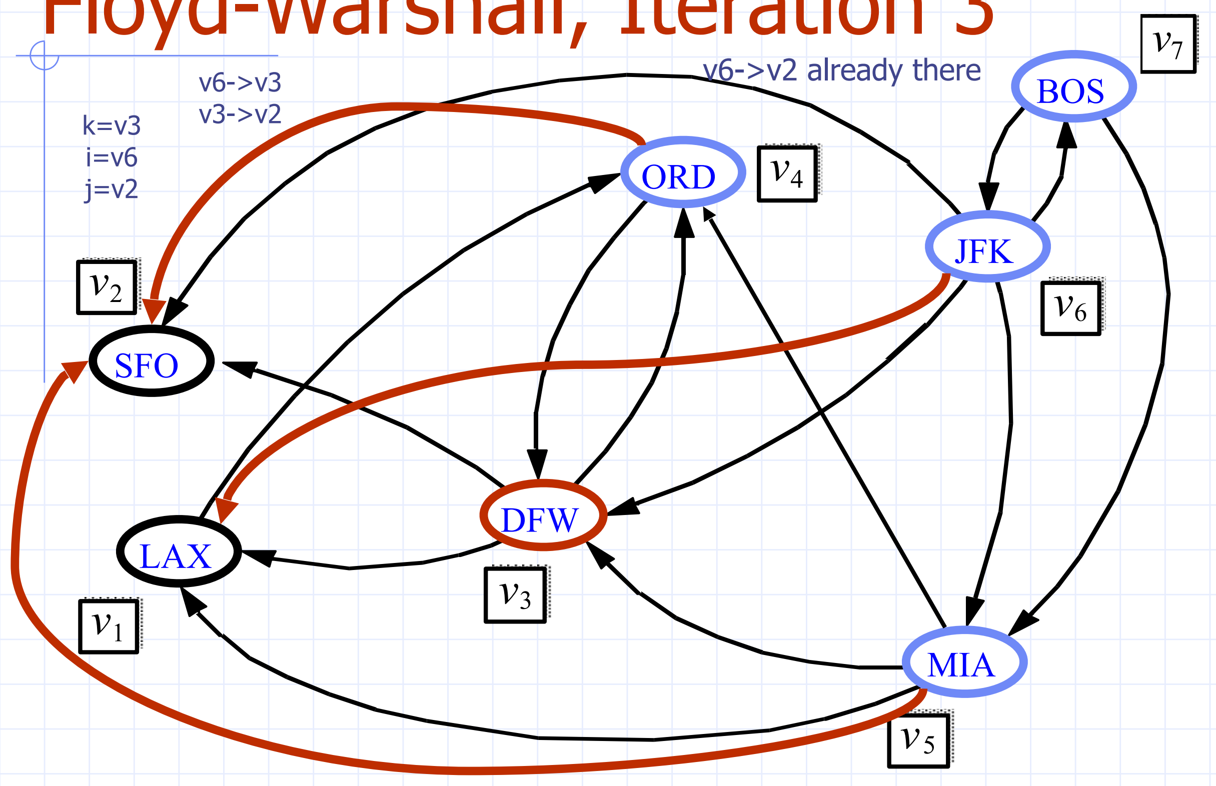
Floyd-Warshall, Iteration 3



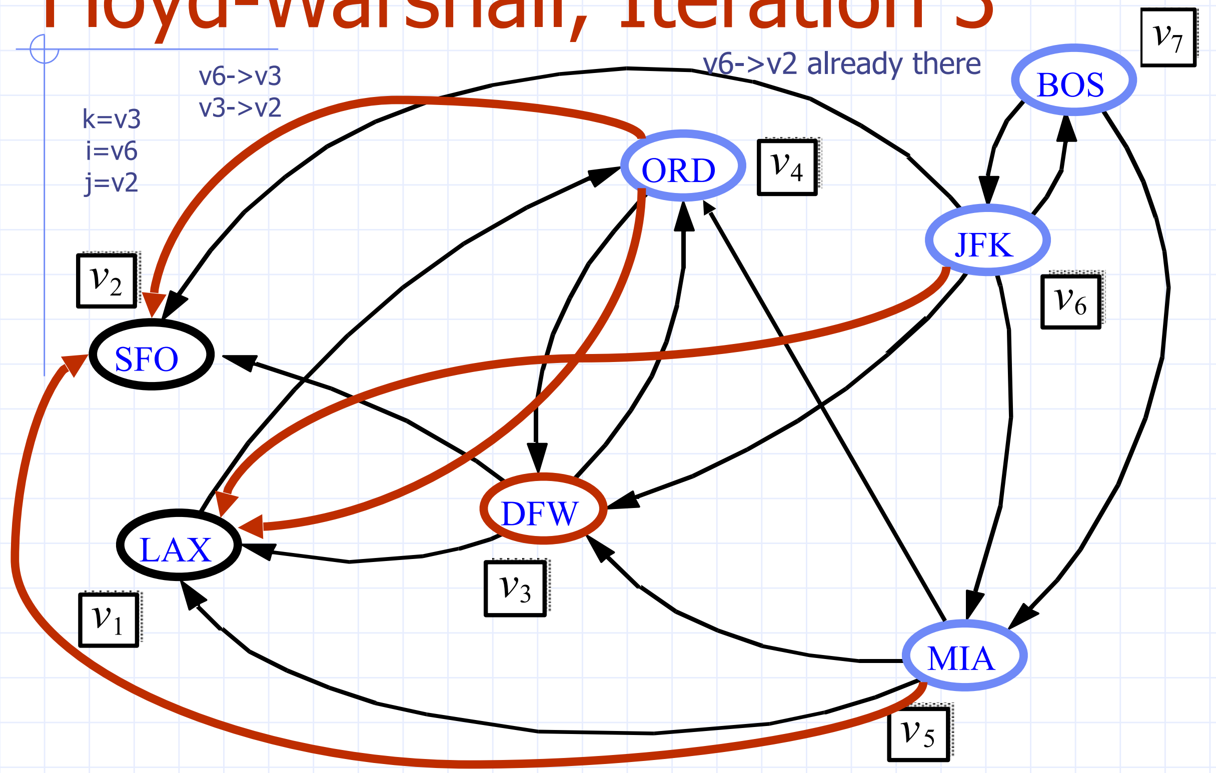
Floyd-Warshall, Iteration 3



Floyd-Warshall, Iteration 3



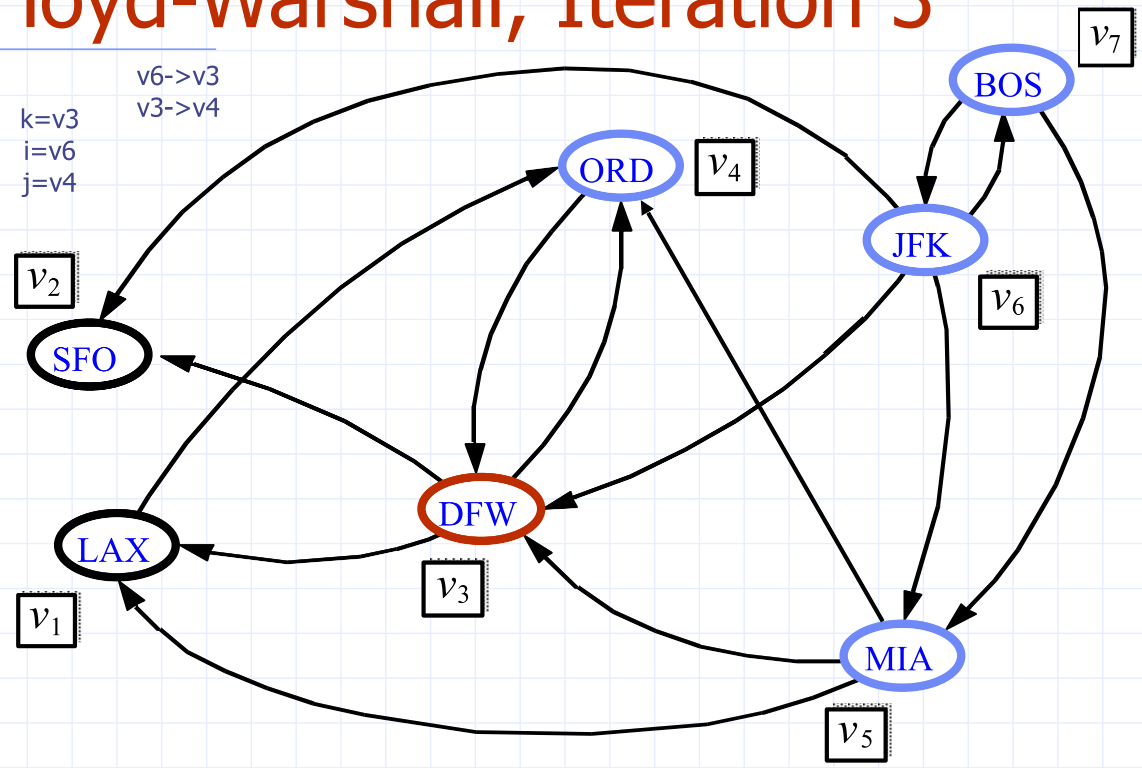
Floyd-Warshall, Iteration 3



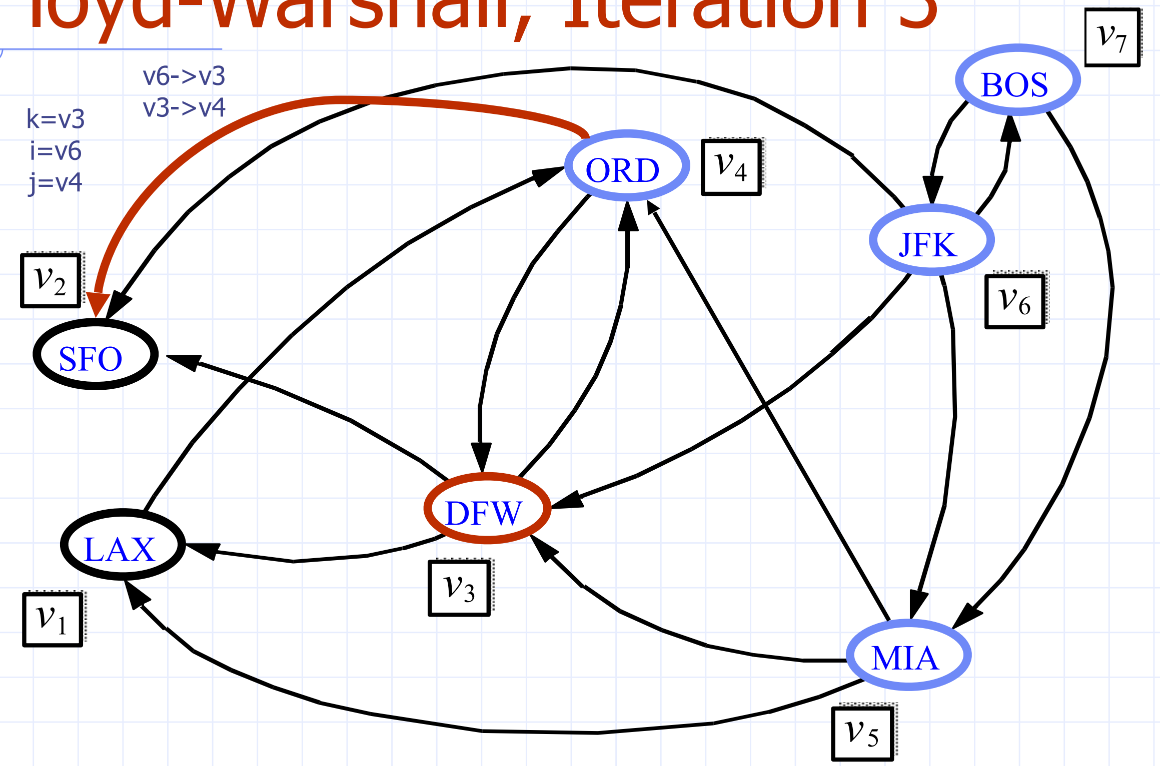
Floyd-Warshall, Iteration 3

k=v3
i=v6
j=v4

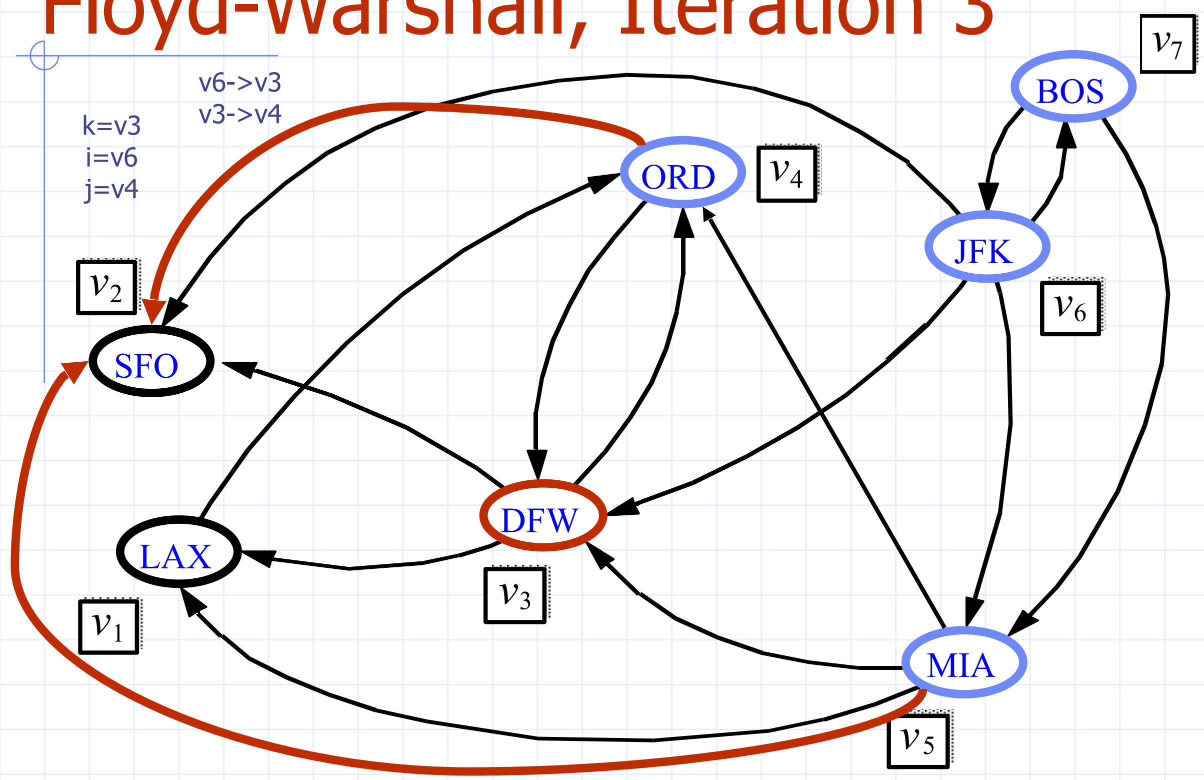
v6->v3
v3->v4



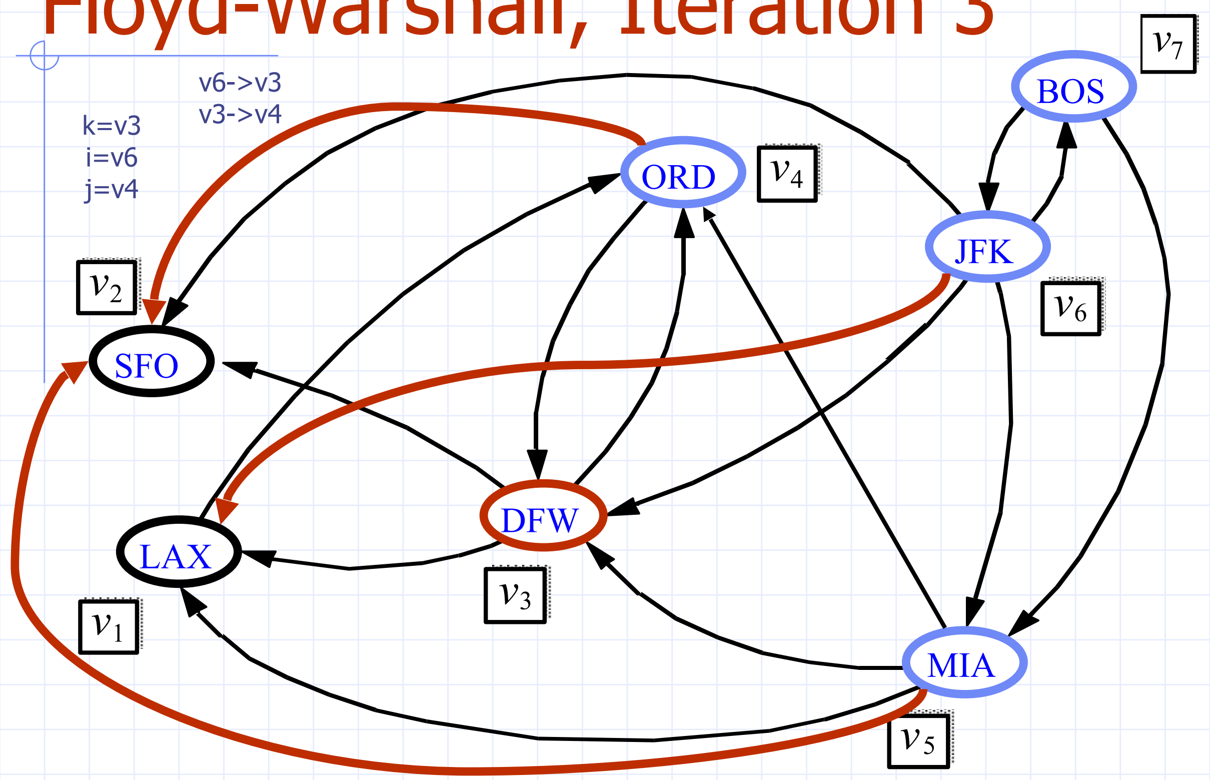
Floyd-Warshall, Iteration 3



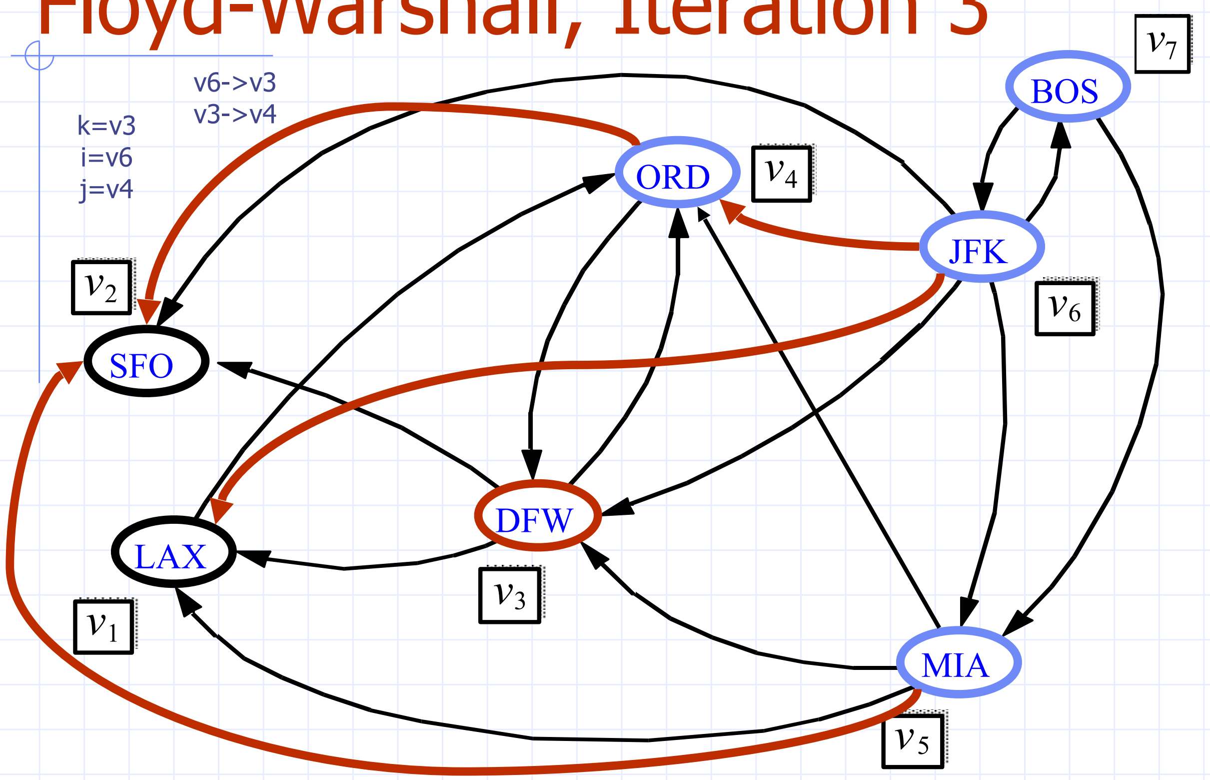
Floyd-Warshall, Iteration 3



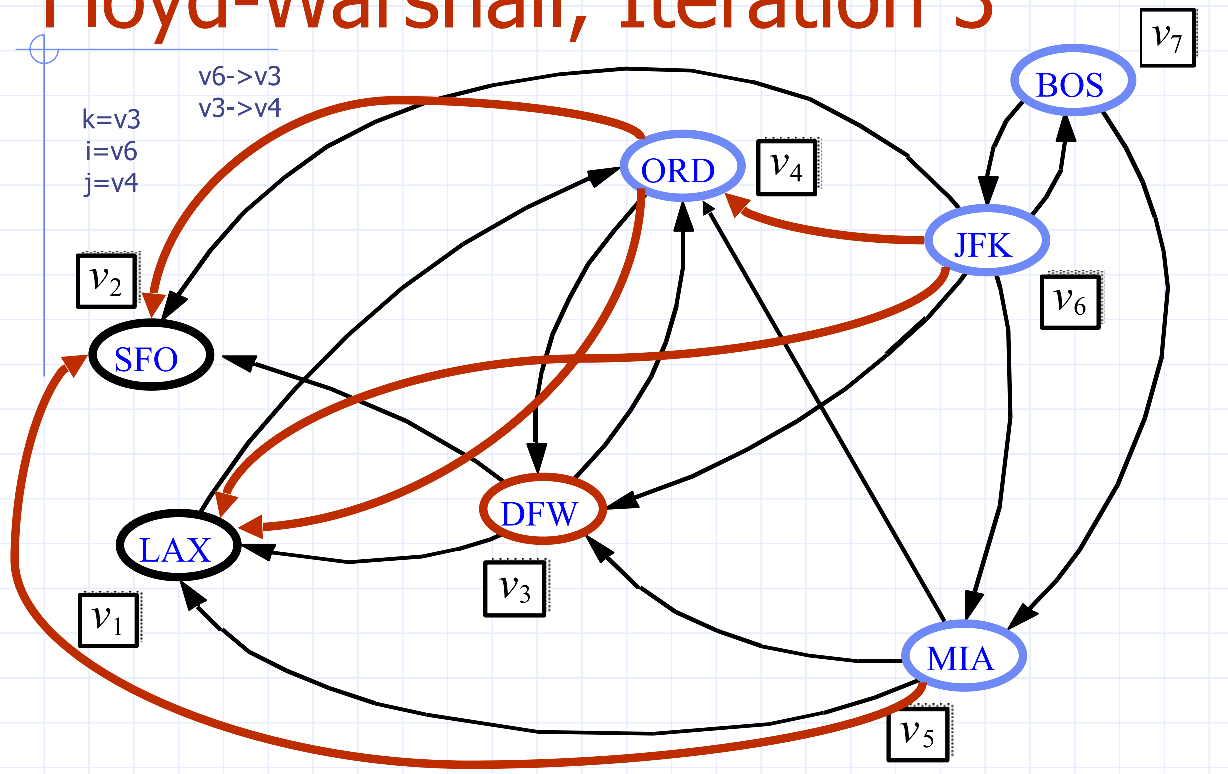
Floyd-Warshall, Iteration 3



Floyd-Warshall, Iteration 3



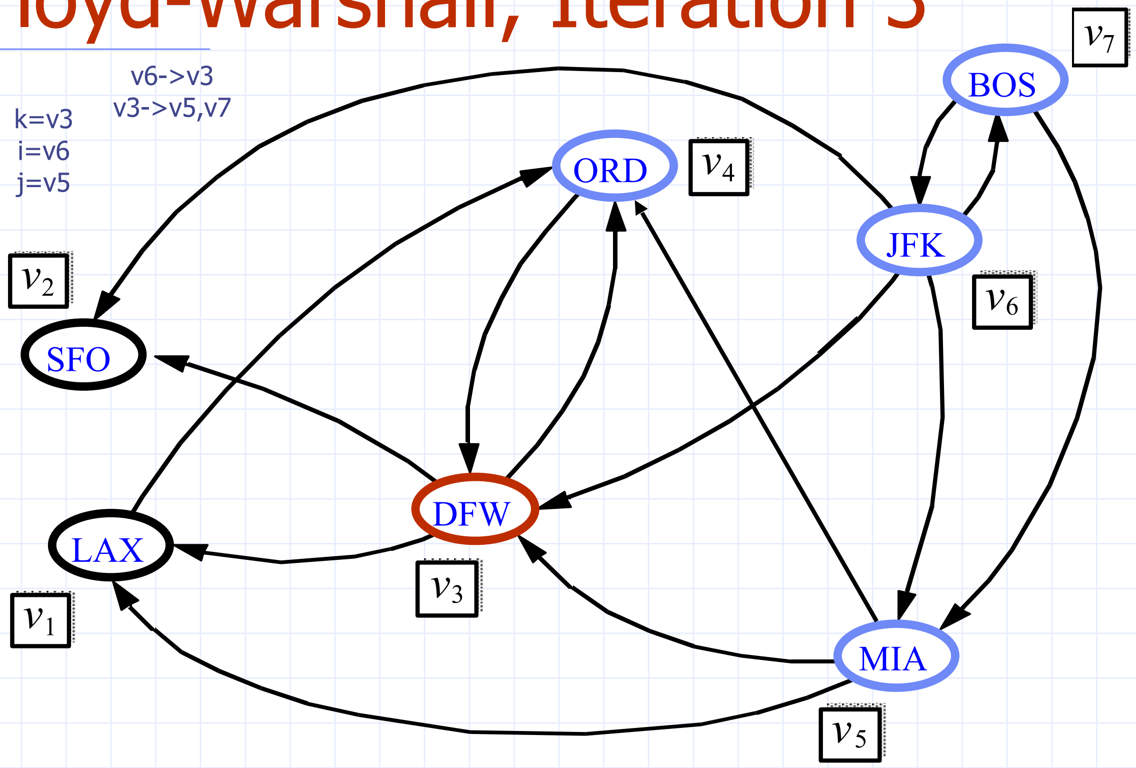
Floyd-Warshall, Iteration 3



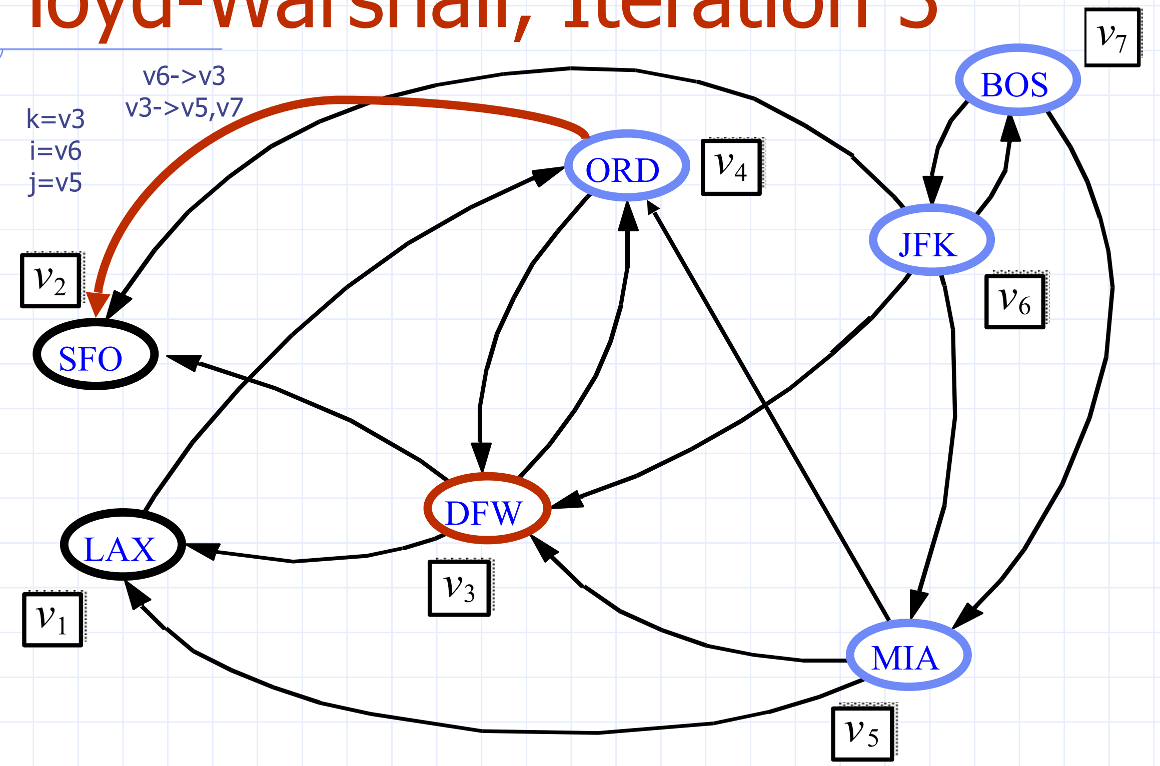
Floyd-Warshall, Iteration 3

k=v3
i=v6
j=v5

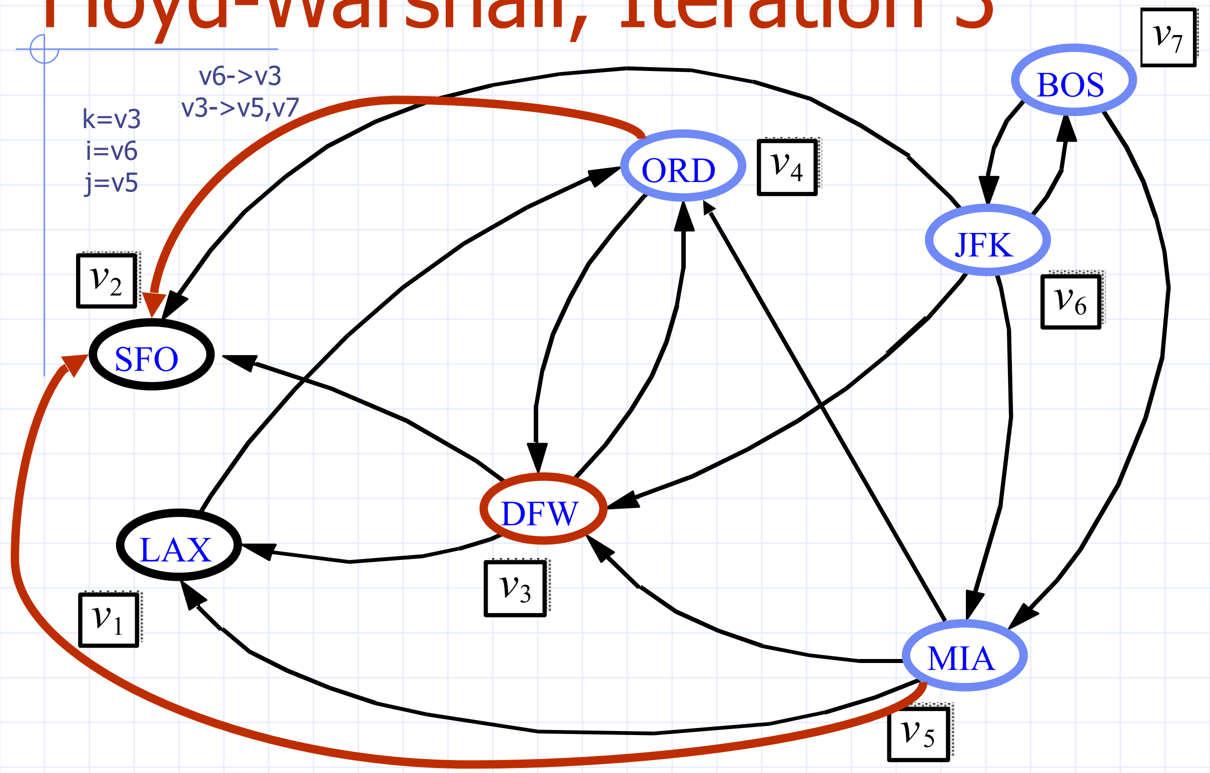
v6->v3
v3->v5,v7



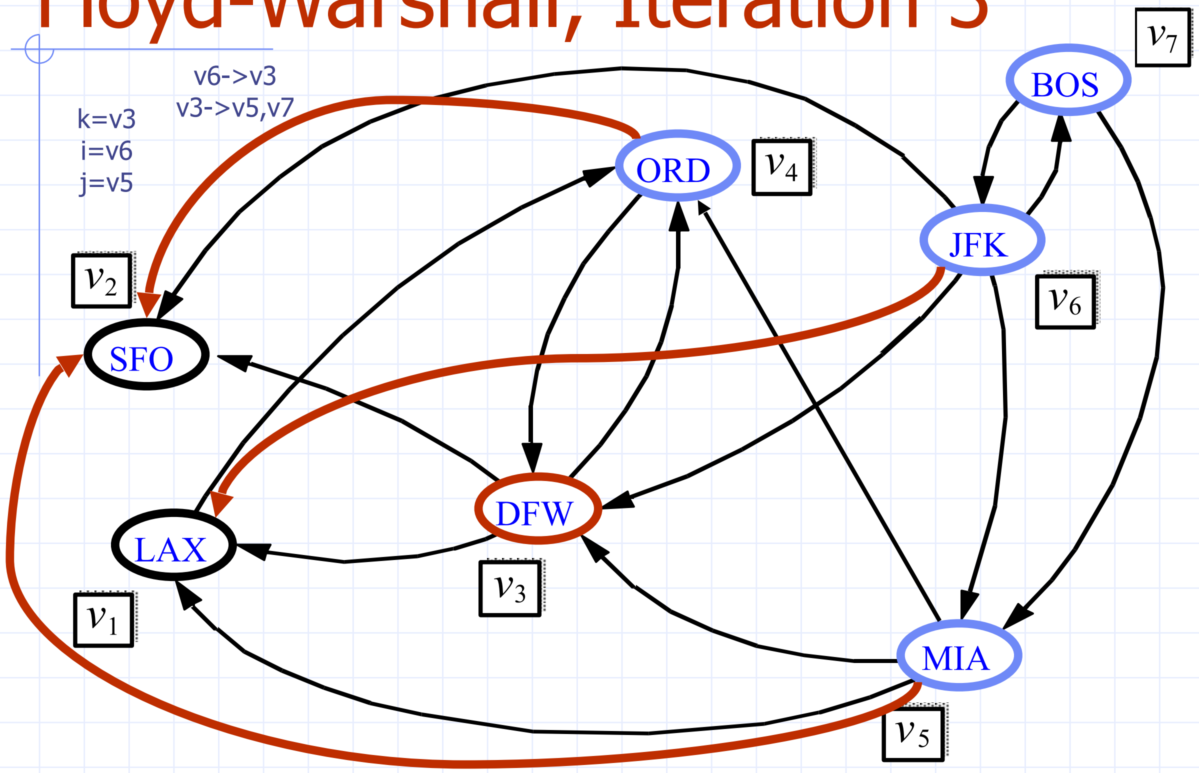
Floyd-Warshall, Iteration 3



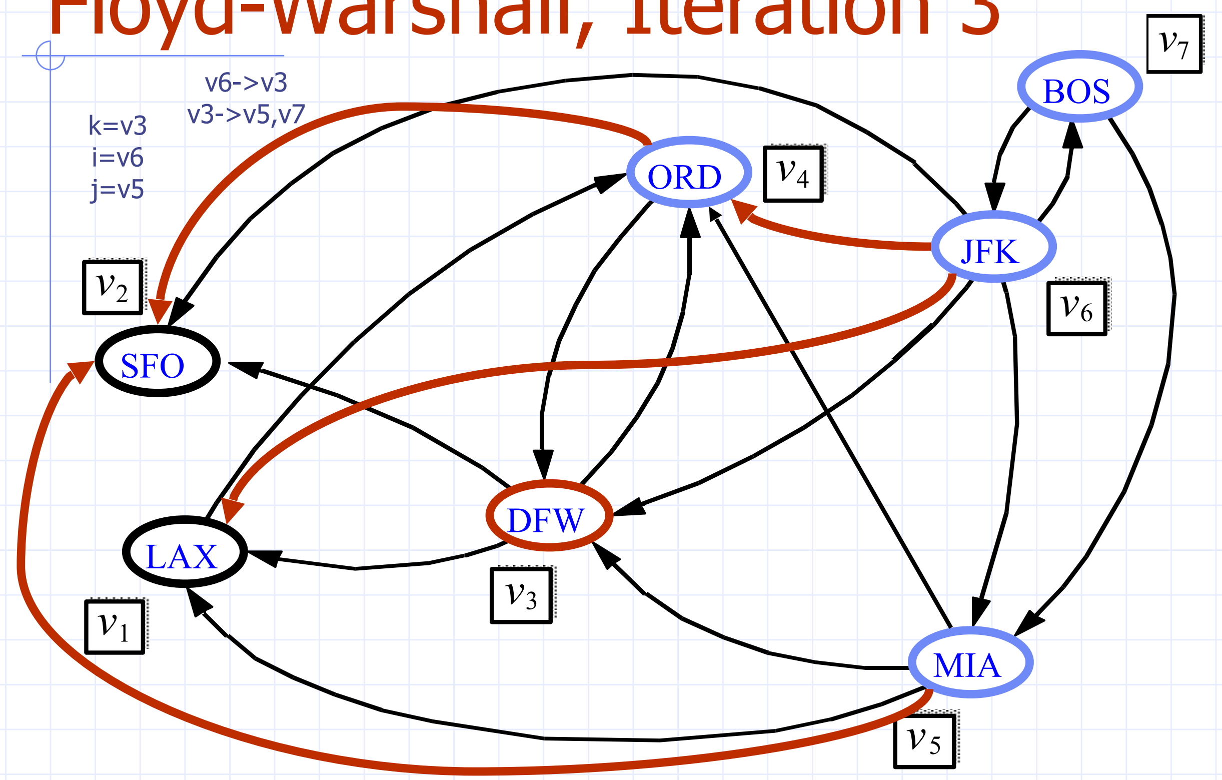
Floyd-Warshall, Iteration 3



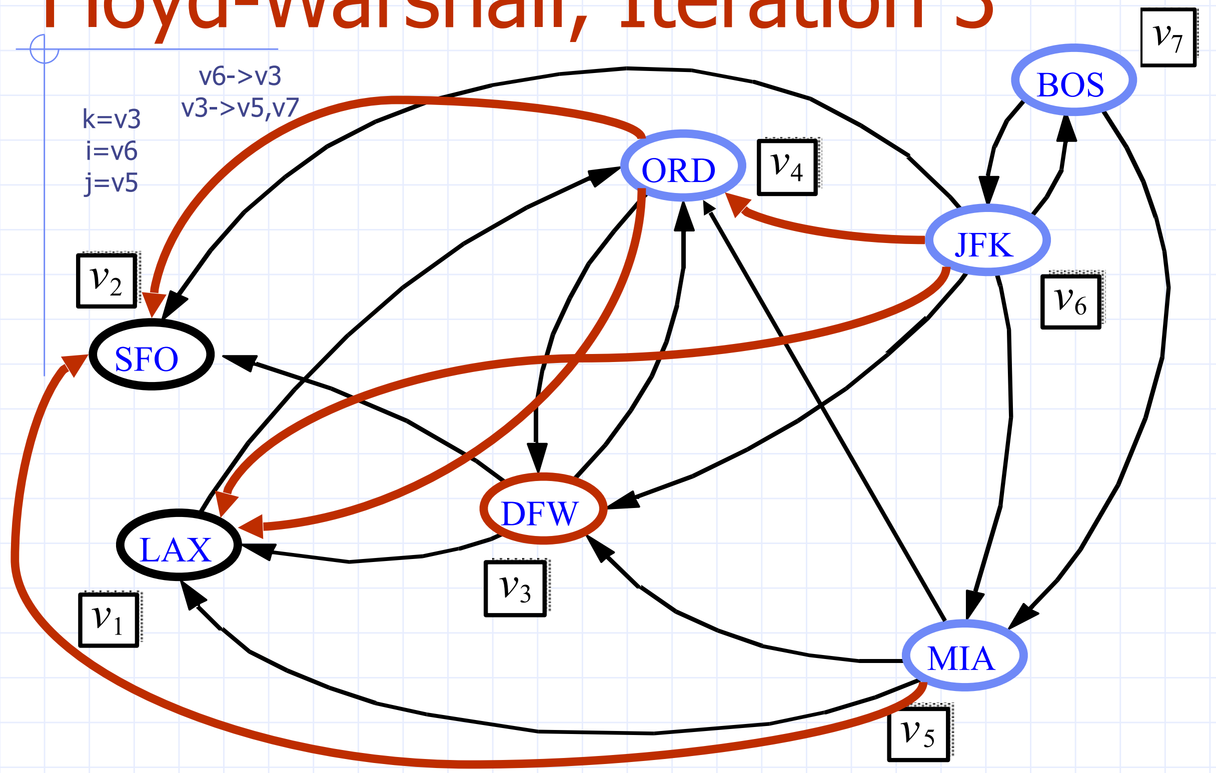
Floyd-Warshall, Iteration 3



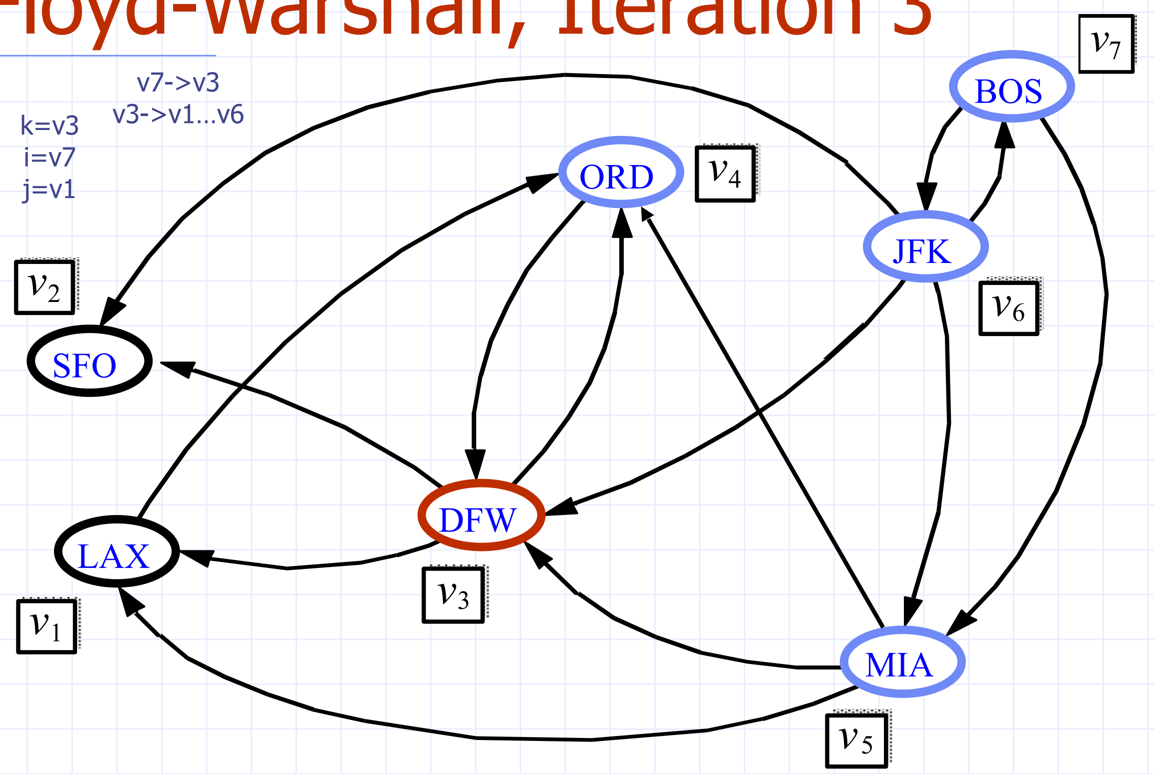
Floyd-Warshall, Iteration 3



Floyd-Warshall, Iteration 3



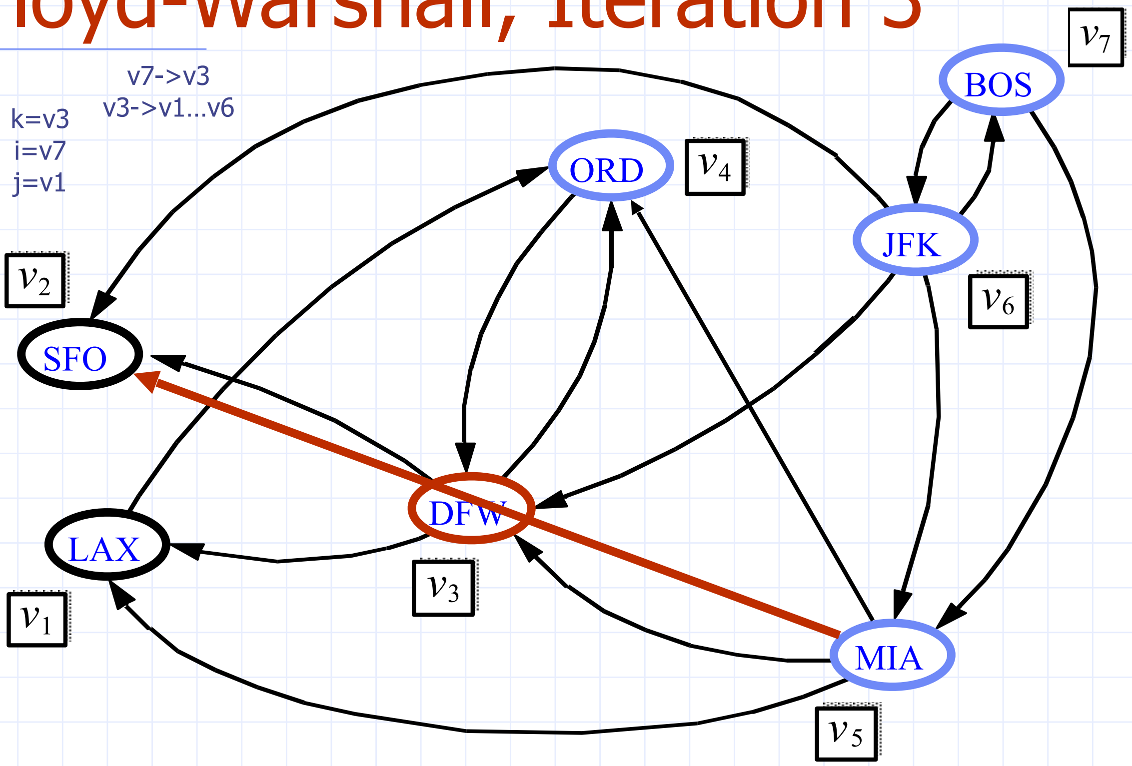
Floyd-Warshall, Iteration 3



Floyd-Warshall, Iteration 3

k=v3
i=v7
j=v1

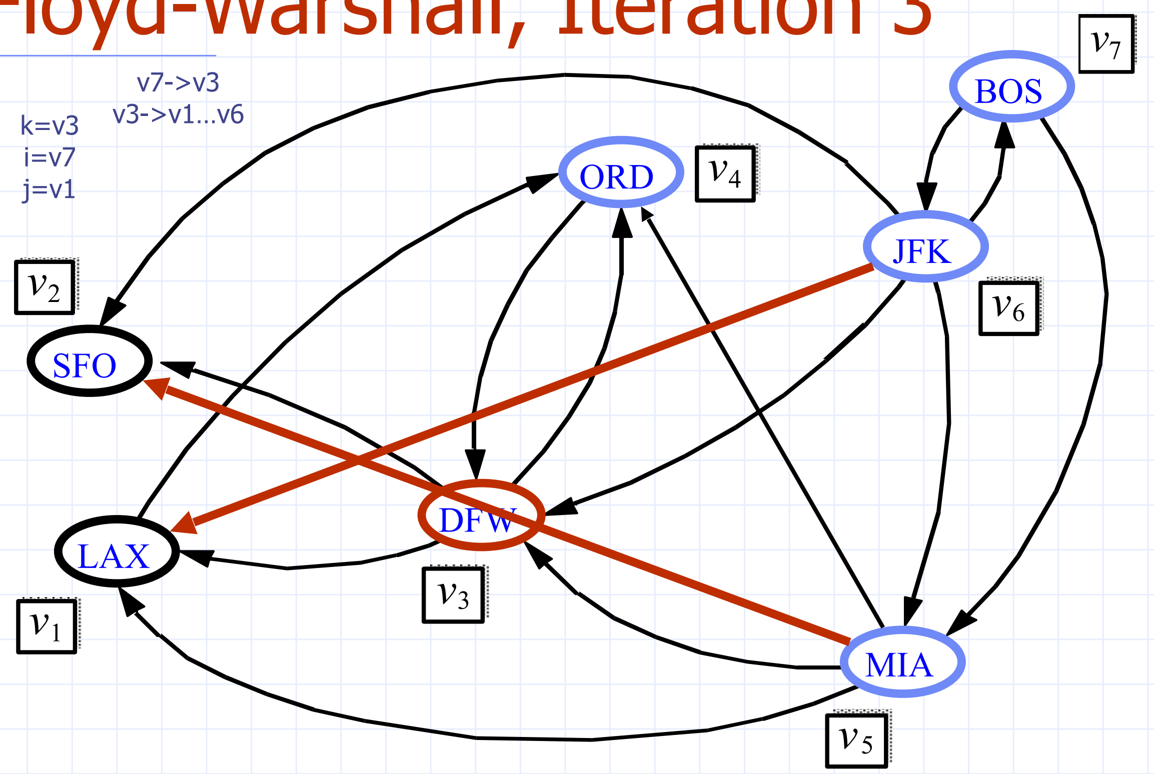
v7->v3
v3->v1…v6



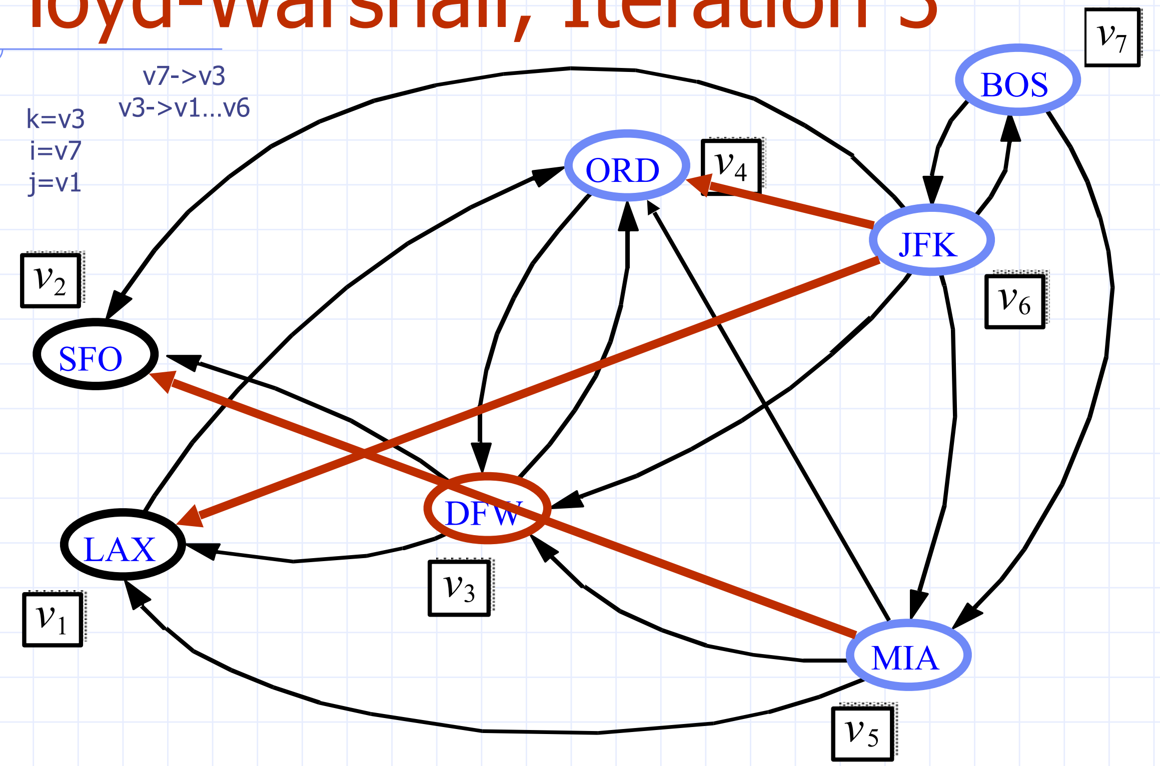
Floyd-Warshall, Iteration 3

k=v3
i=v7
j=v1

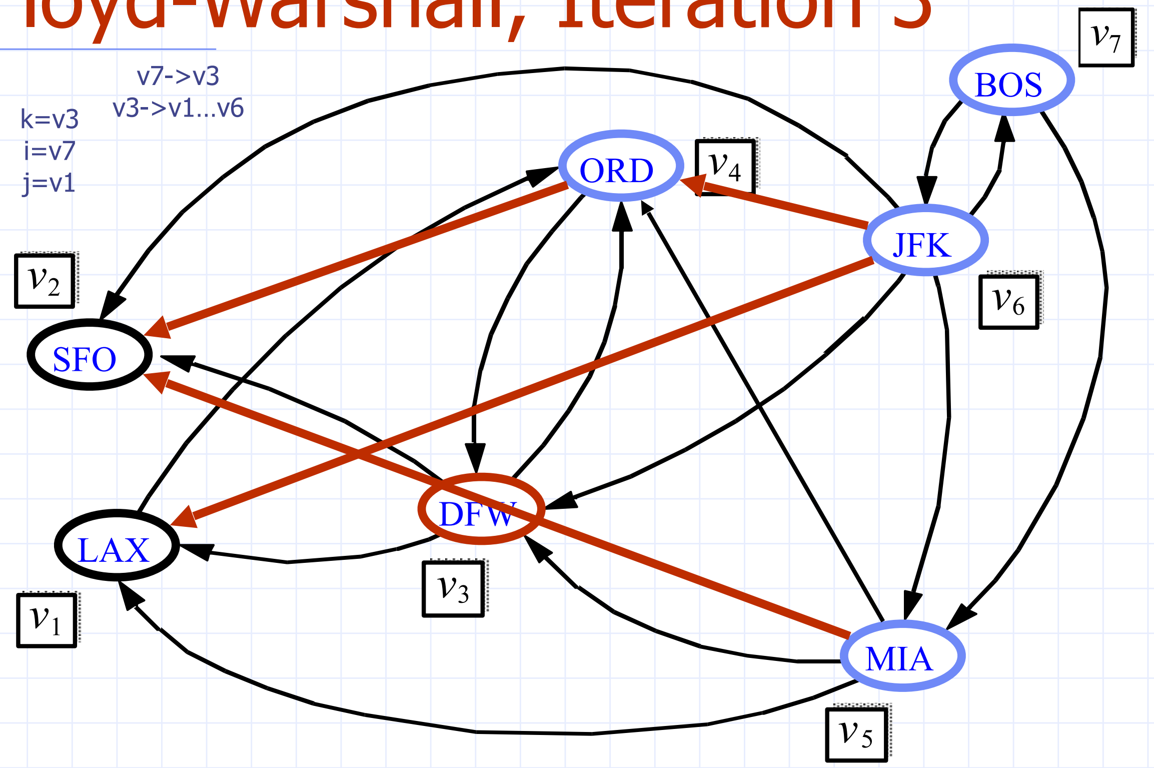
v7->v3
v3->v1...v6



Floyd-Warshall, Iteration 3



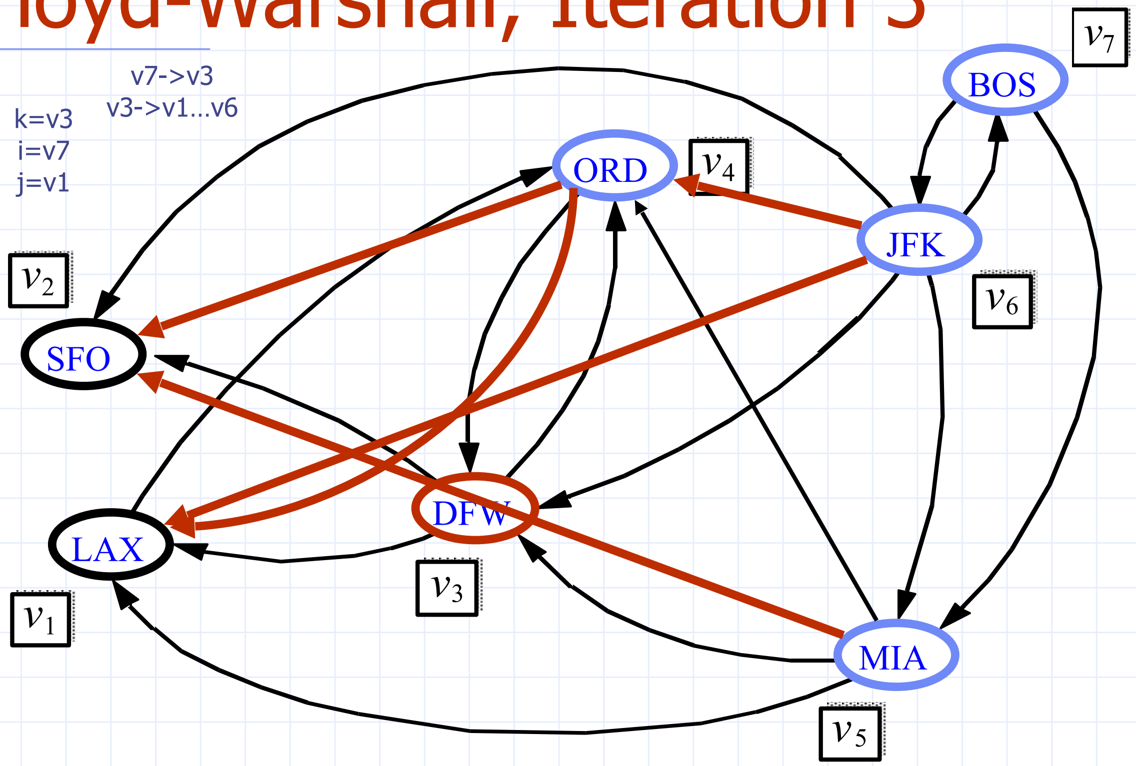
Floyd-Warshall, Iteration 3



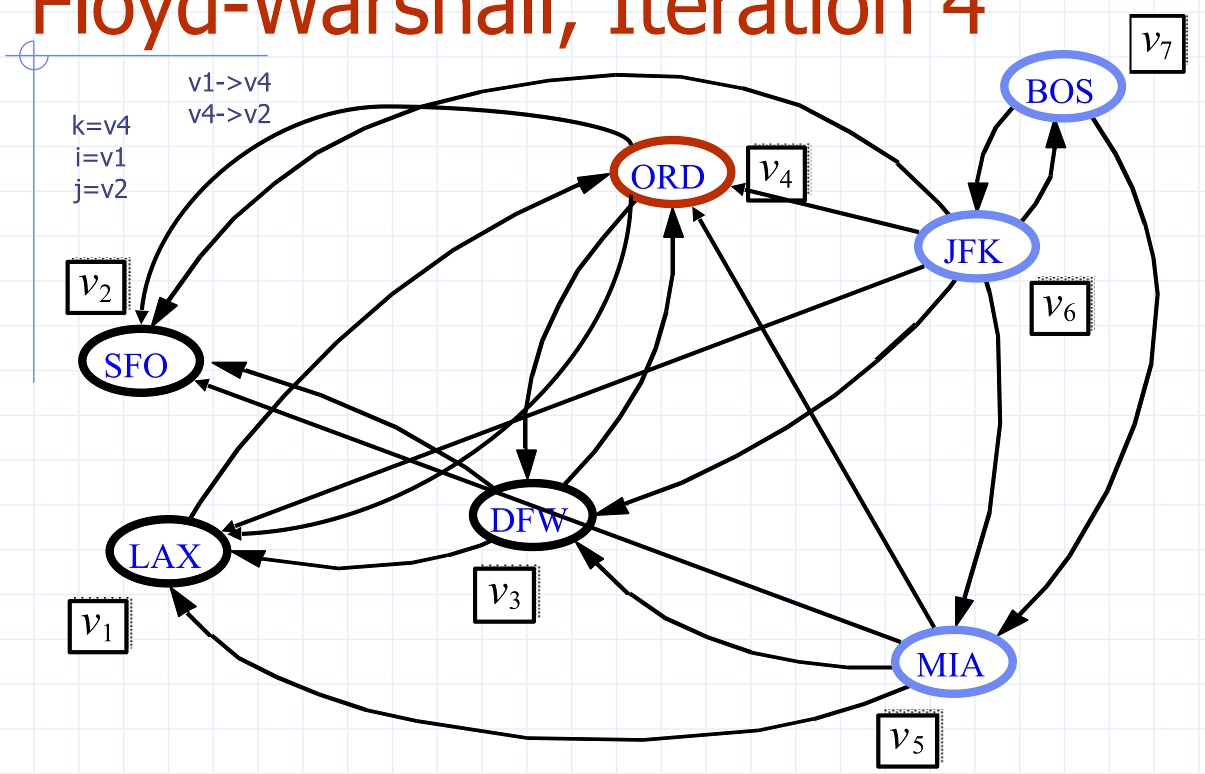
Floyd-Warshall, Iteration 3

k=v3
i=v7
j=v1

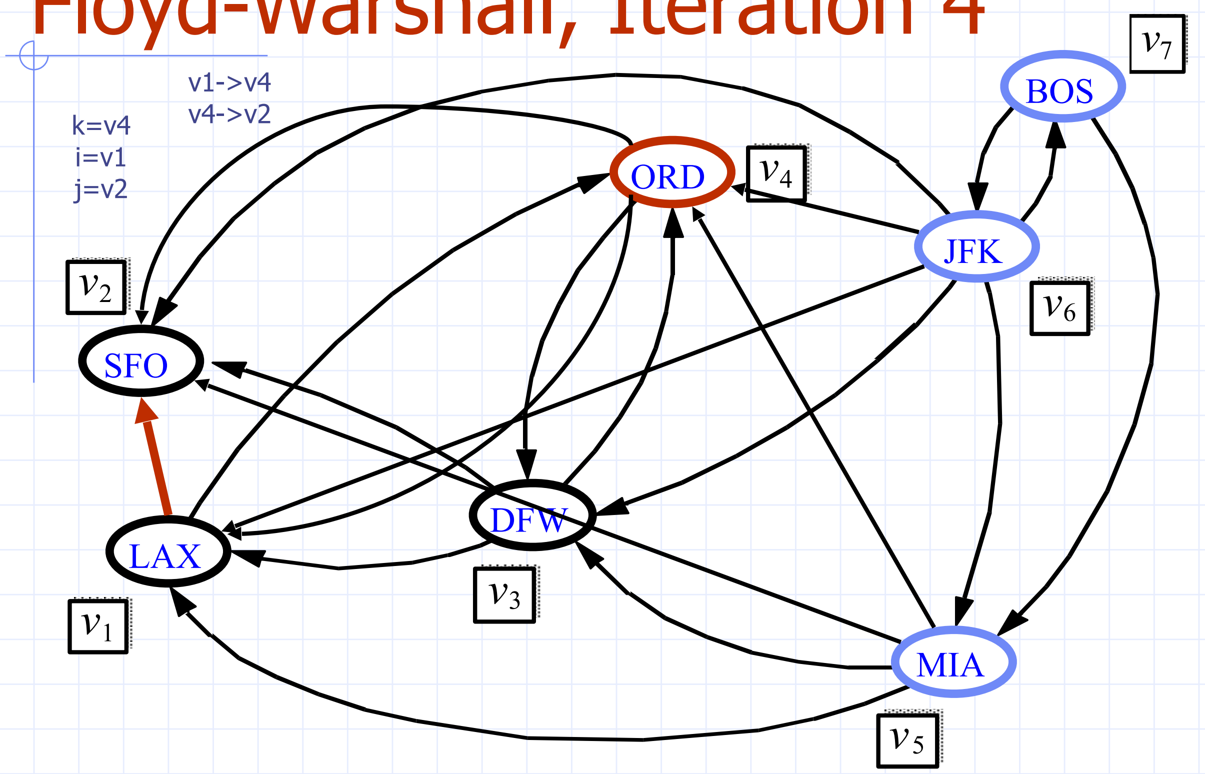
v7->v3
v3->v1…v6



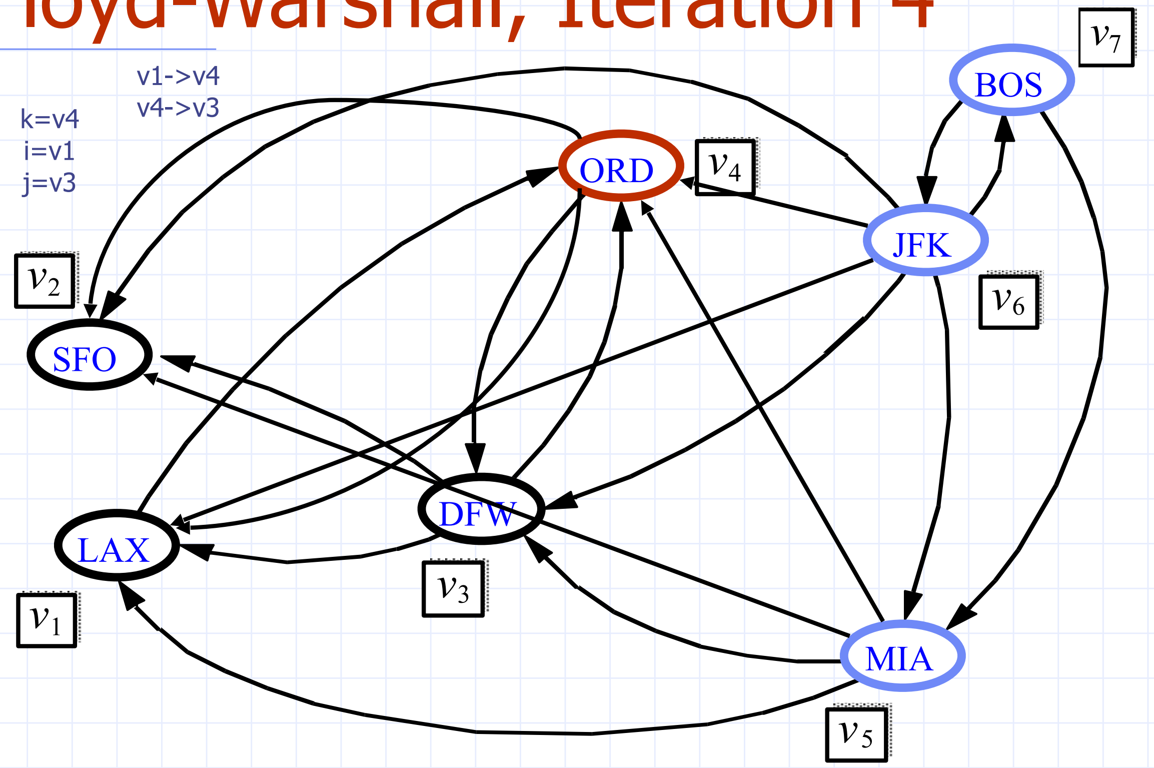
Floyd-Warshall, Iteration 4



Floyd-Warshall, Iteration 4



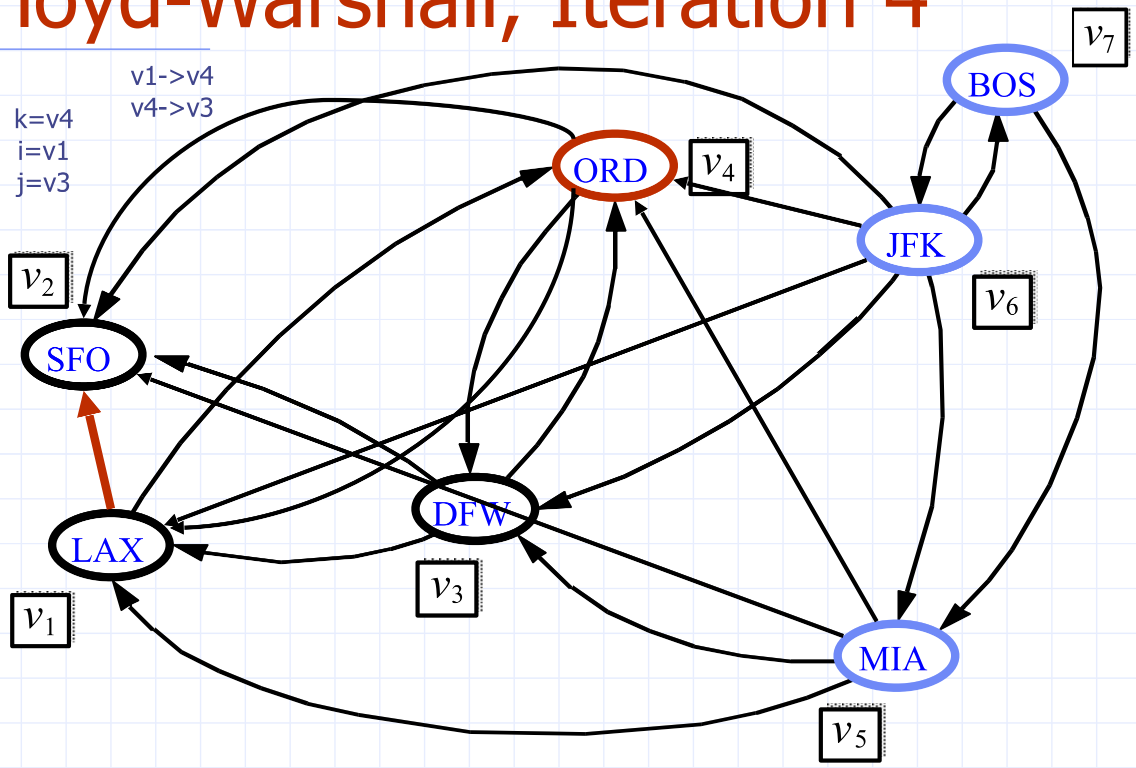
Floyd-Warshall, Iteration 4



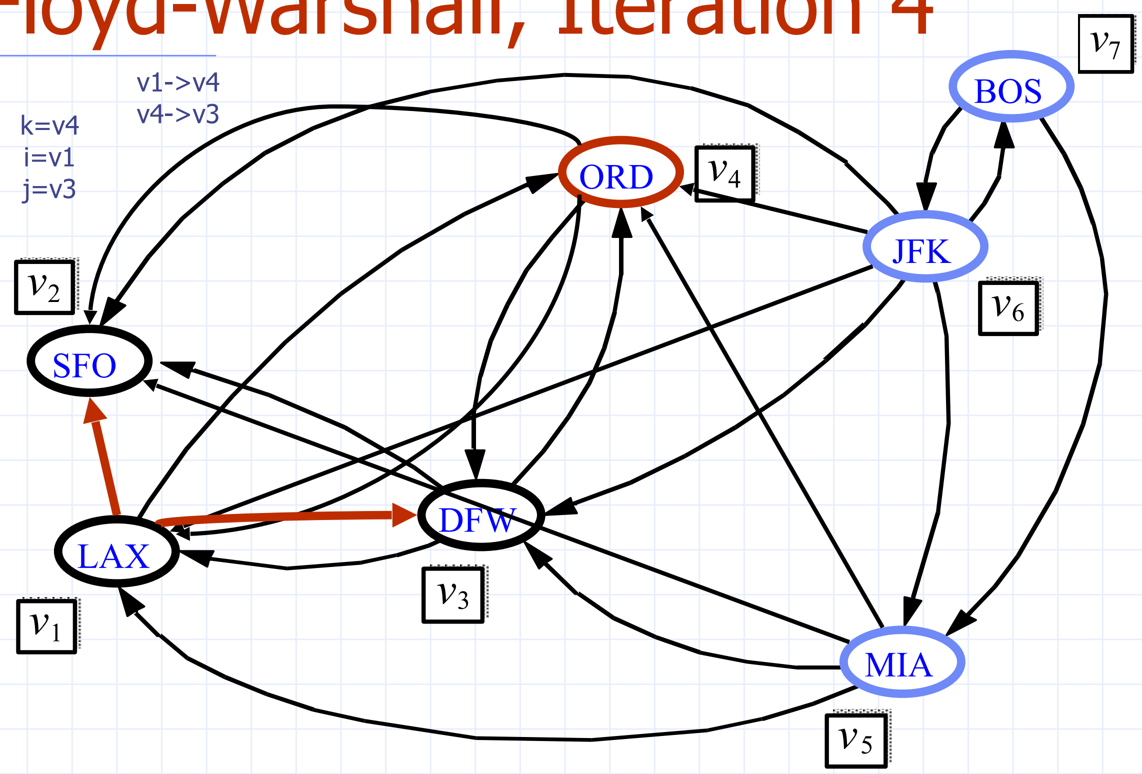
Floyd-Warshall, Iteration 4

k=v4
i=v1
j=v3

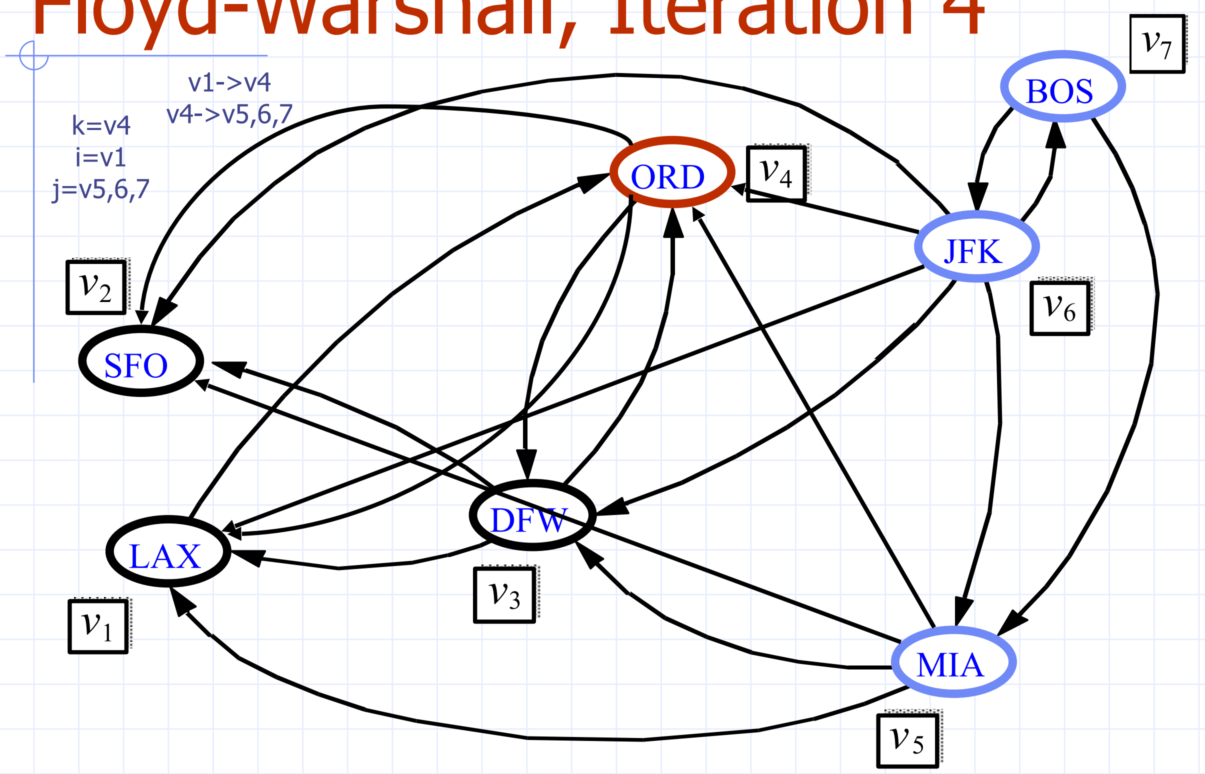
v1->v4
v4->v3



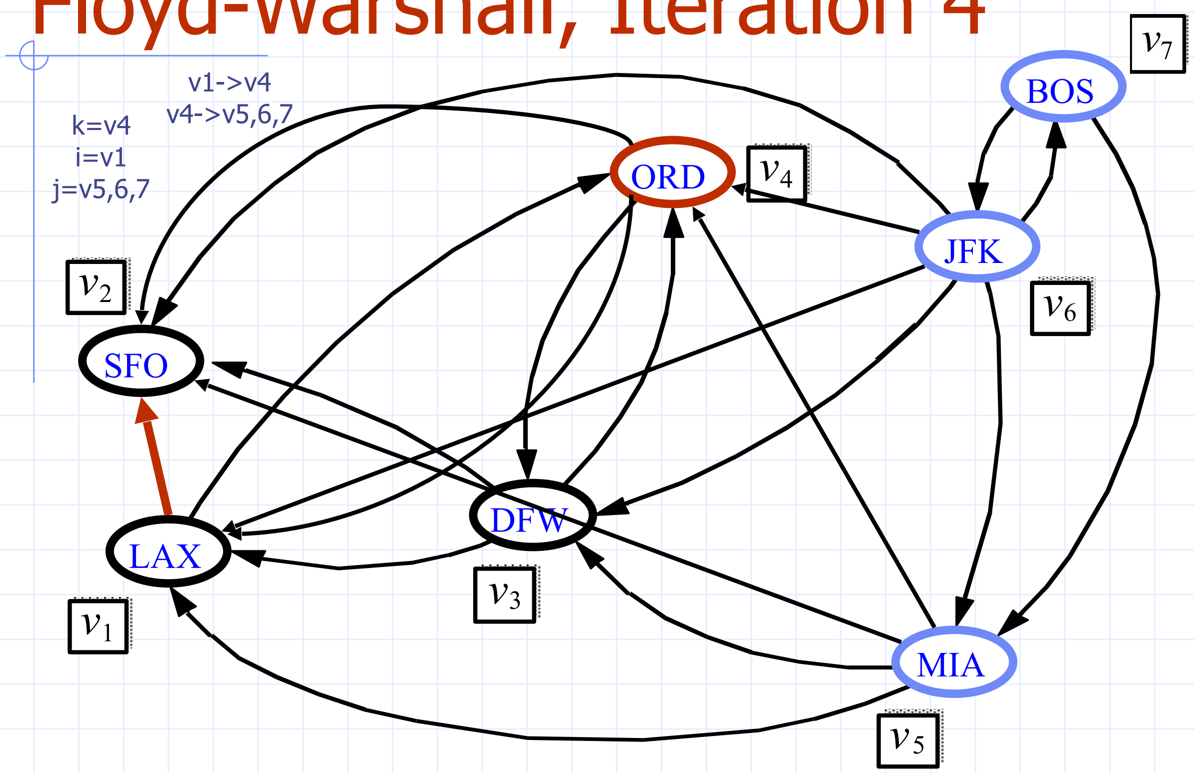
Floyd-Warshall, Iteration 4



Floyd-Warshall, Iteration 4



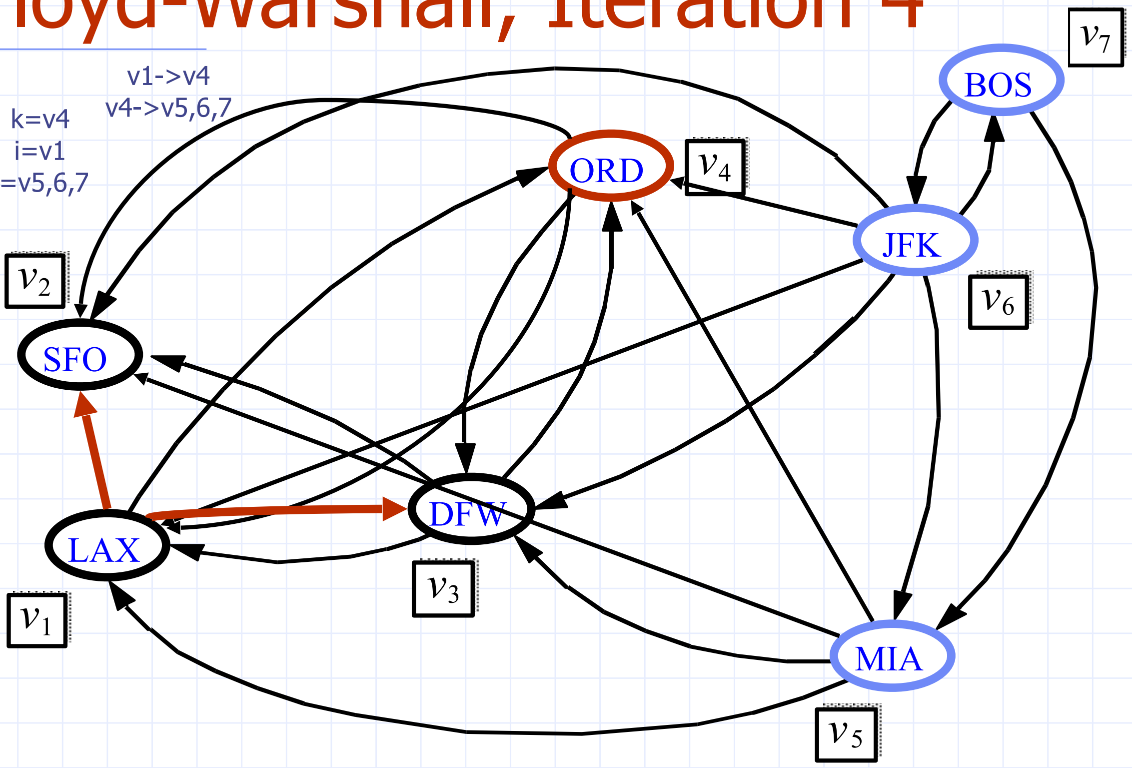
Floyd-Warshall, Iteration 4



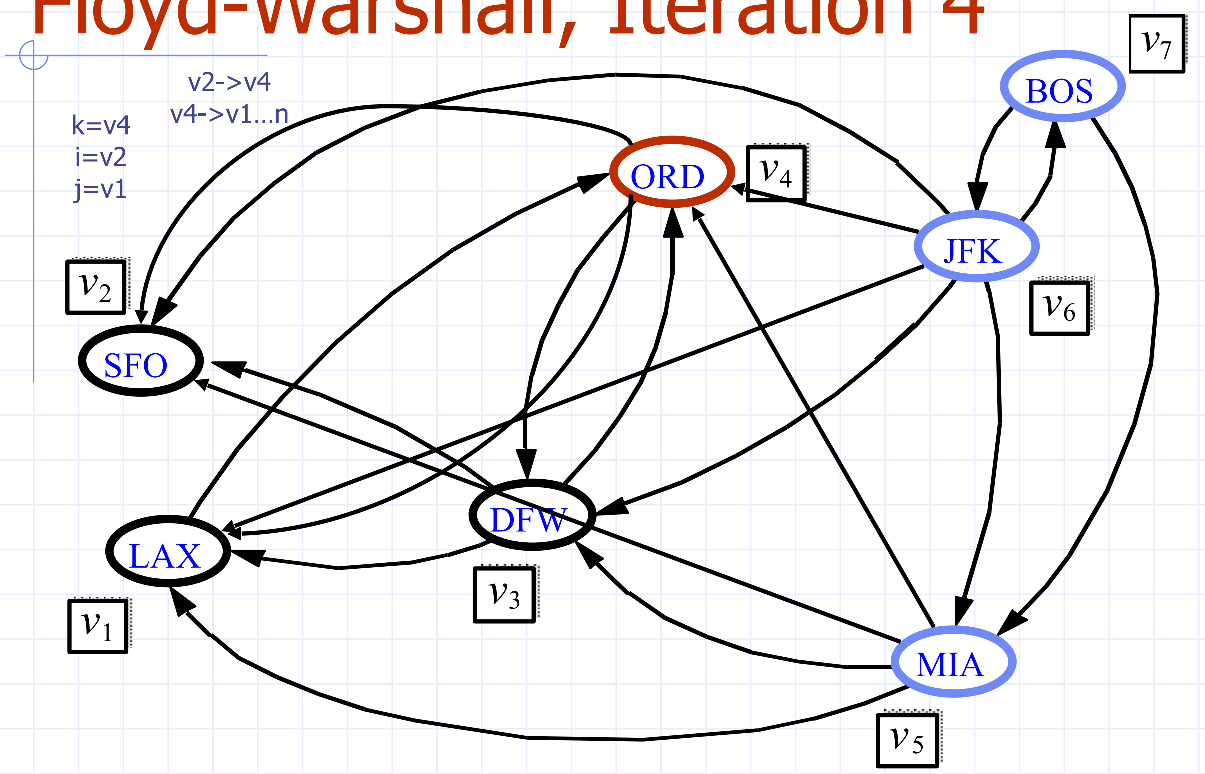
Floyd-Warshall, Iteration 4

k=v4
i=v1
j=v5,6,7

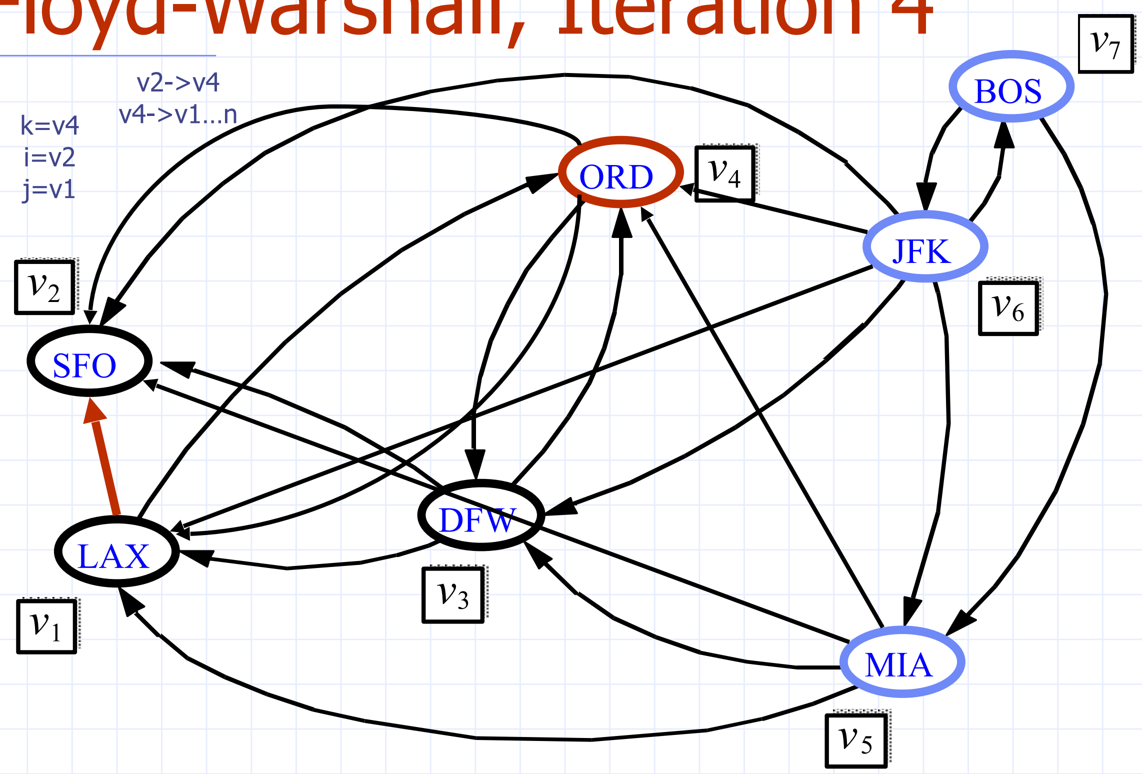
v1->v4
v4->v5,6,7



Floyd-Warshall, Iteration 4



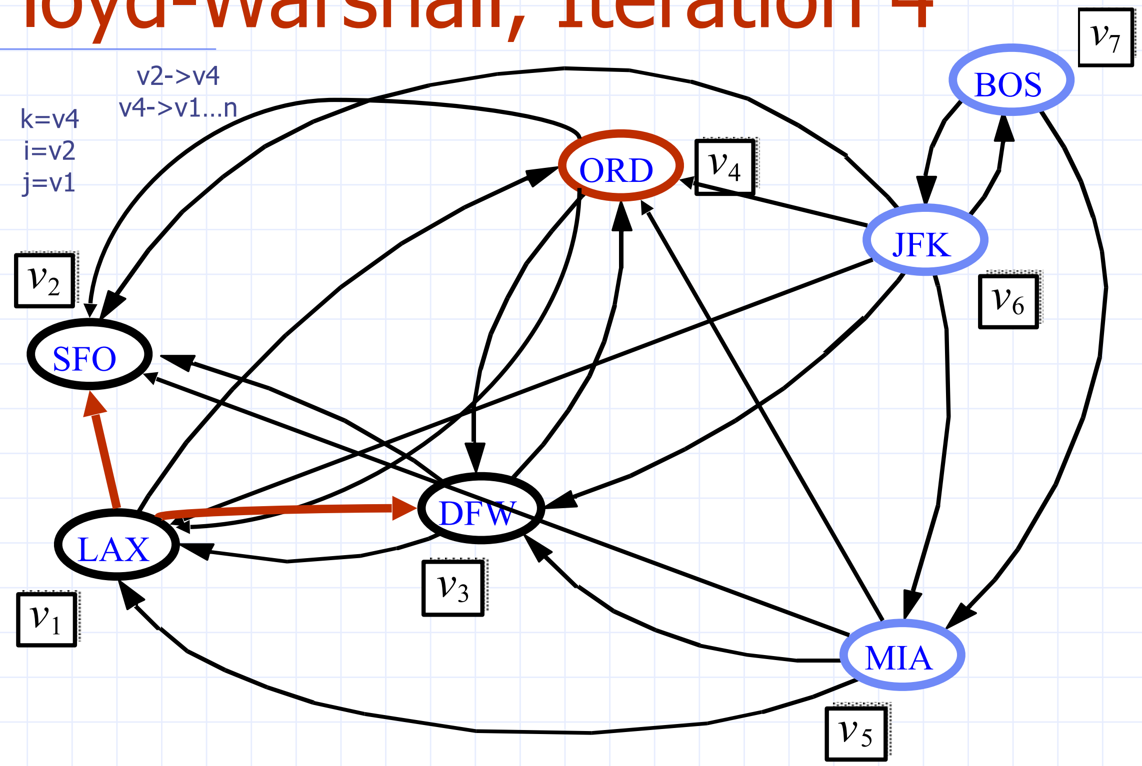
Floyd-Warshall, Iteration 4



Floyd-Warshall, Iteration 4

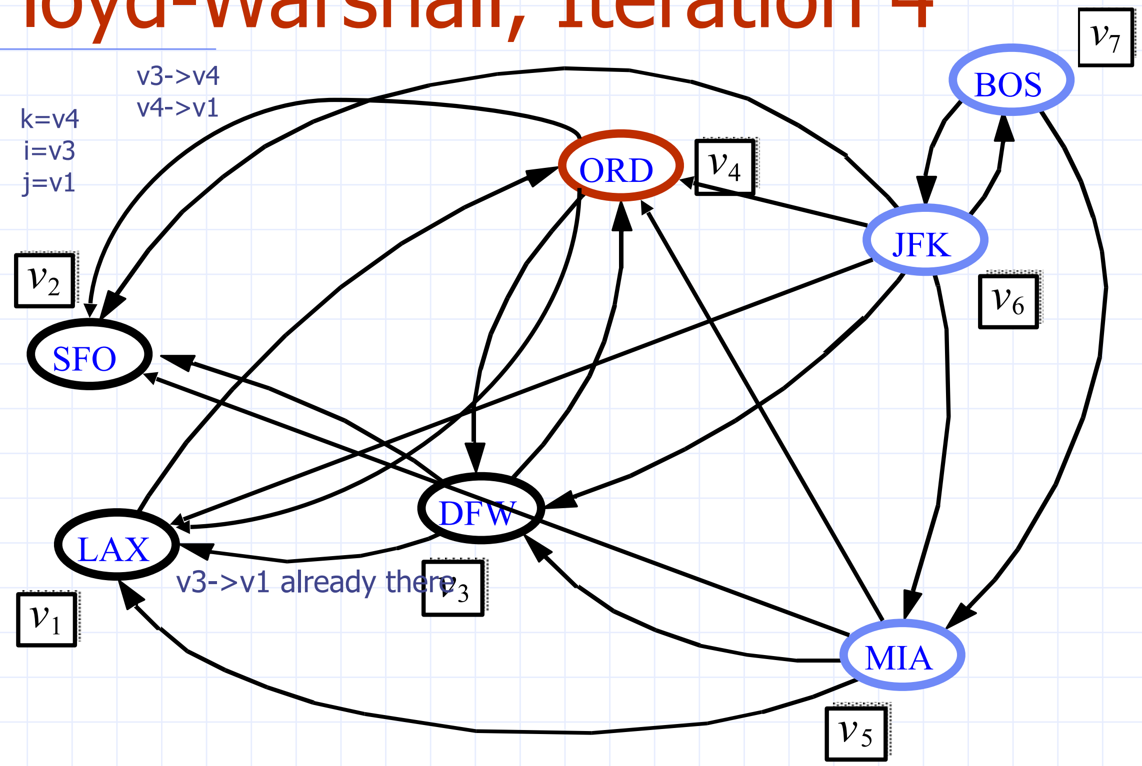
k=v4
i=v2
j=v1

v2->v4
v4->v1...n

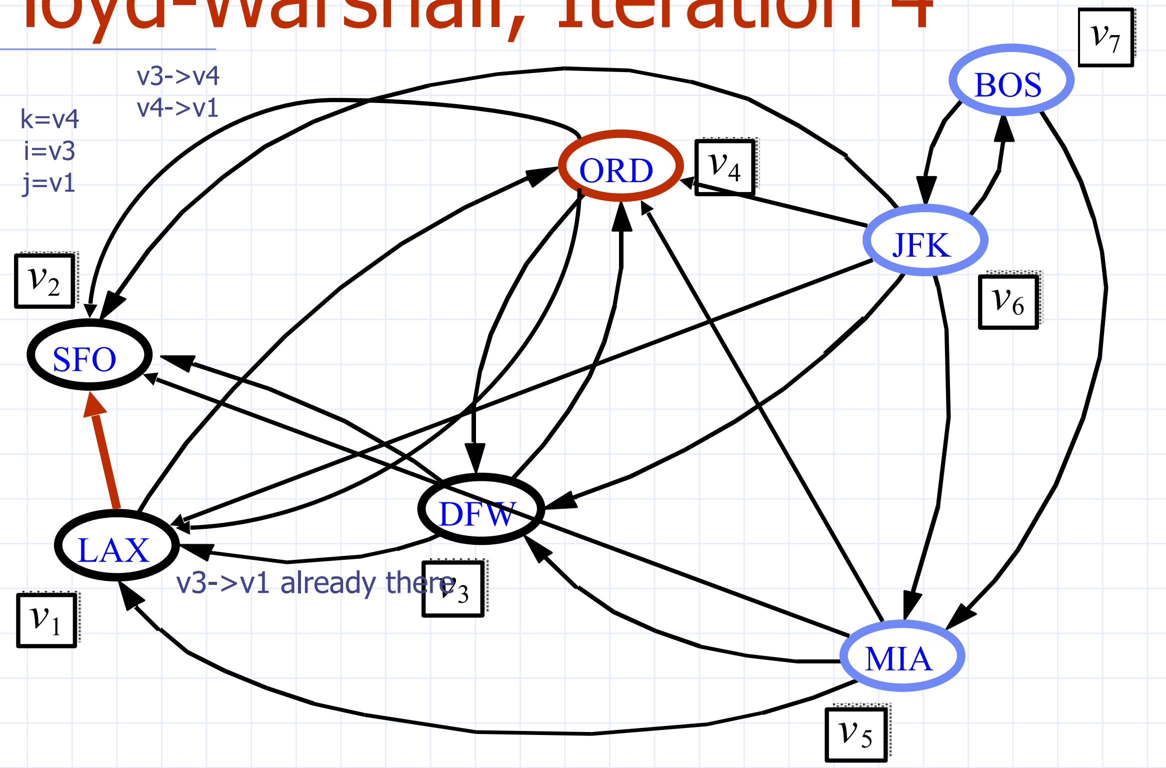


Floyd-Warshall, Iteration 4

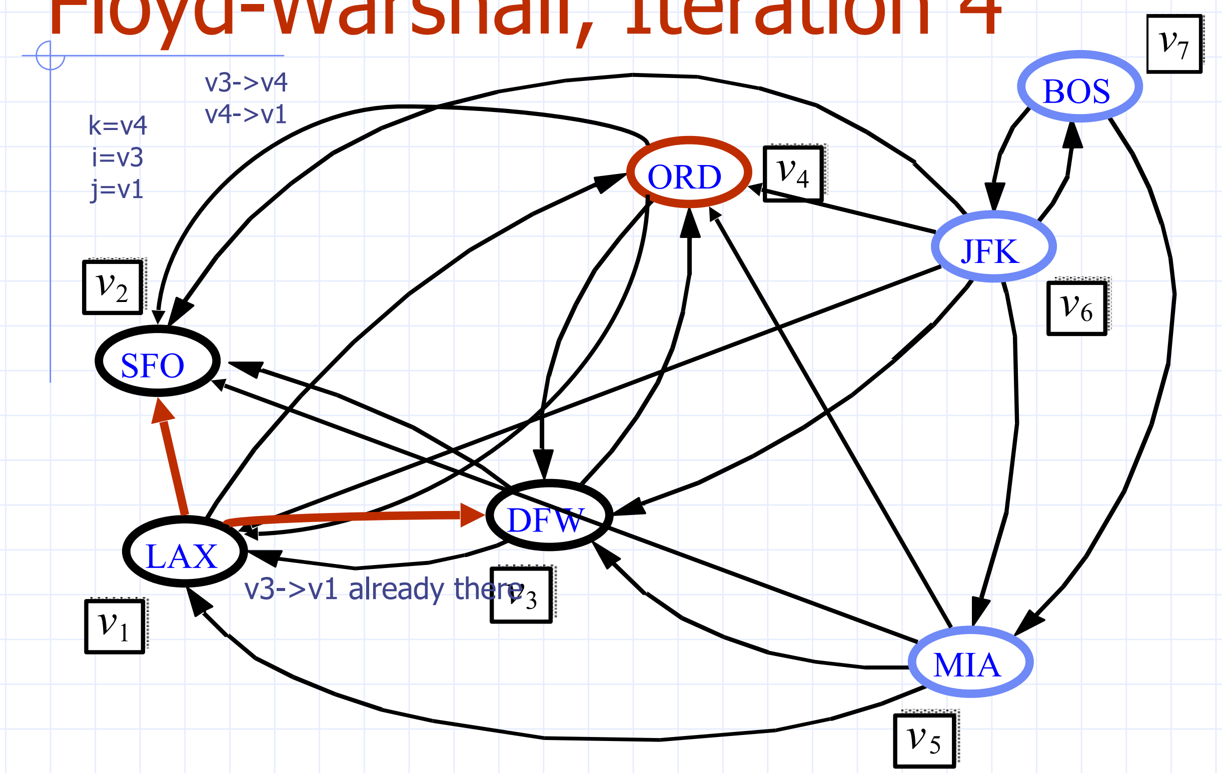
k=v4
i=v3
j=v1



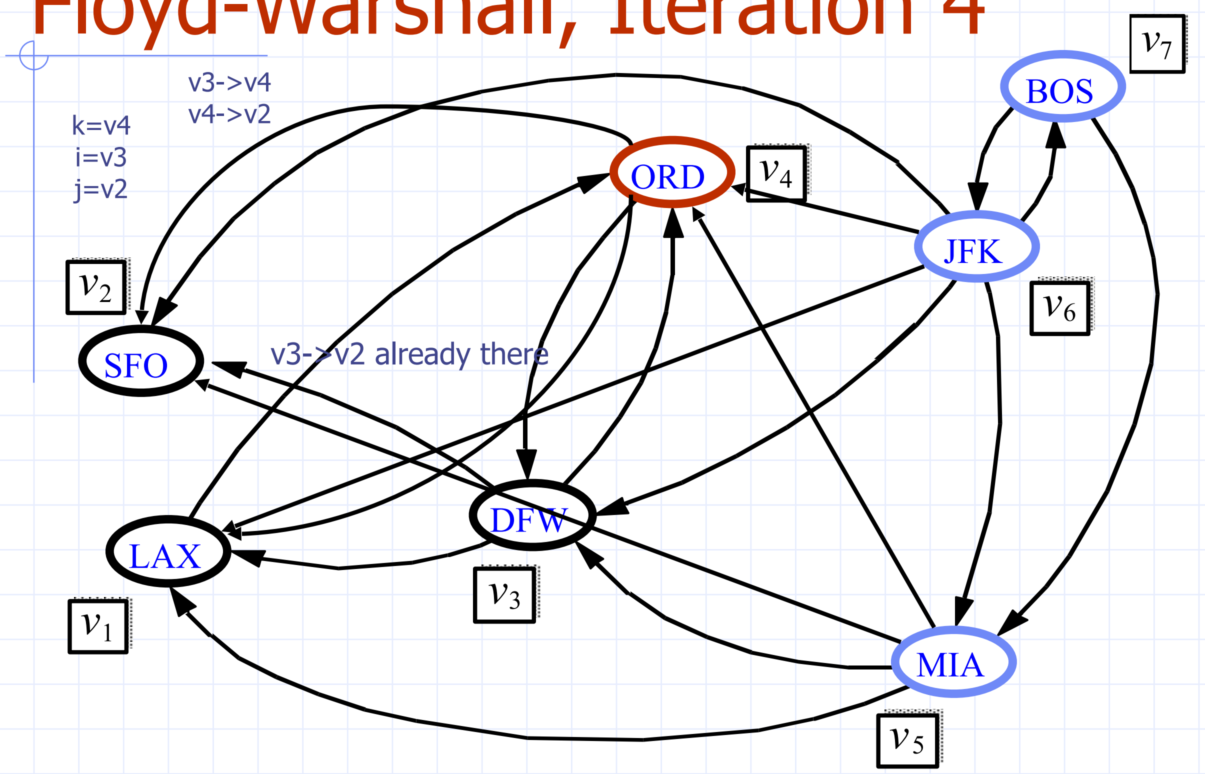
Floyd-Warshall, Iteration 4



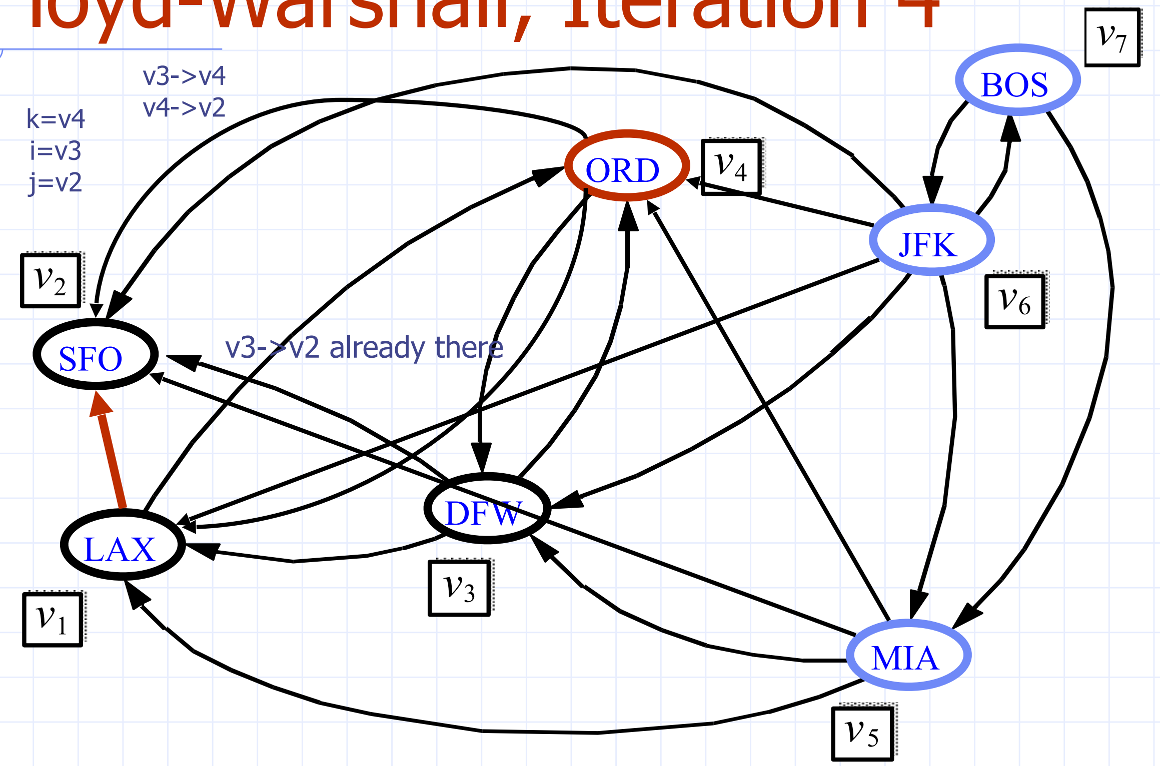
Floyd-Warshall, Iteration 4



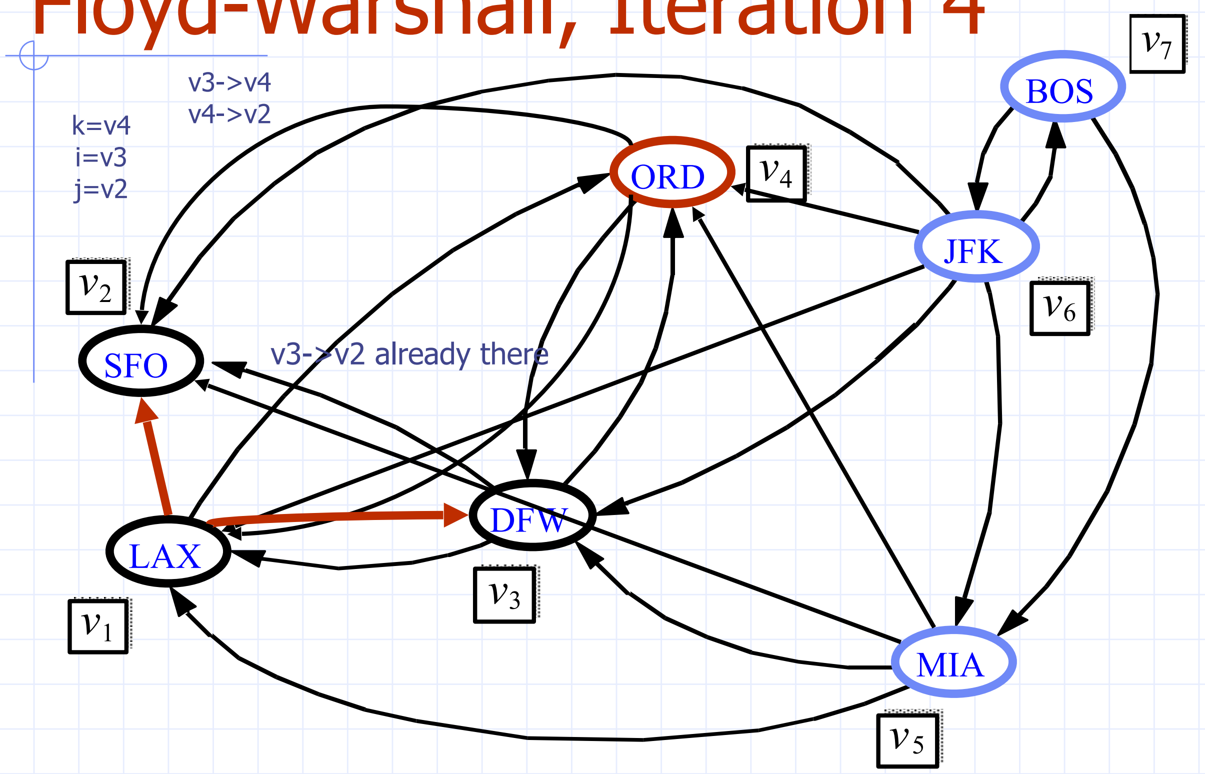
Floyd-Warshall, Iteration 4



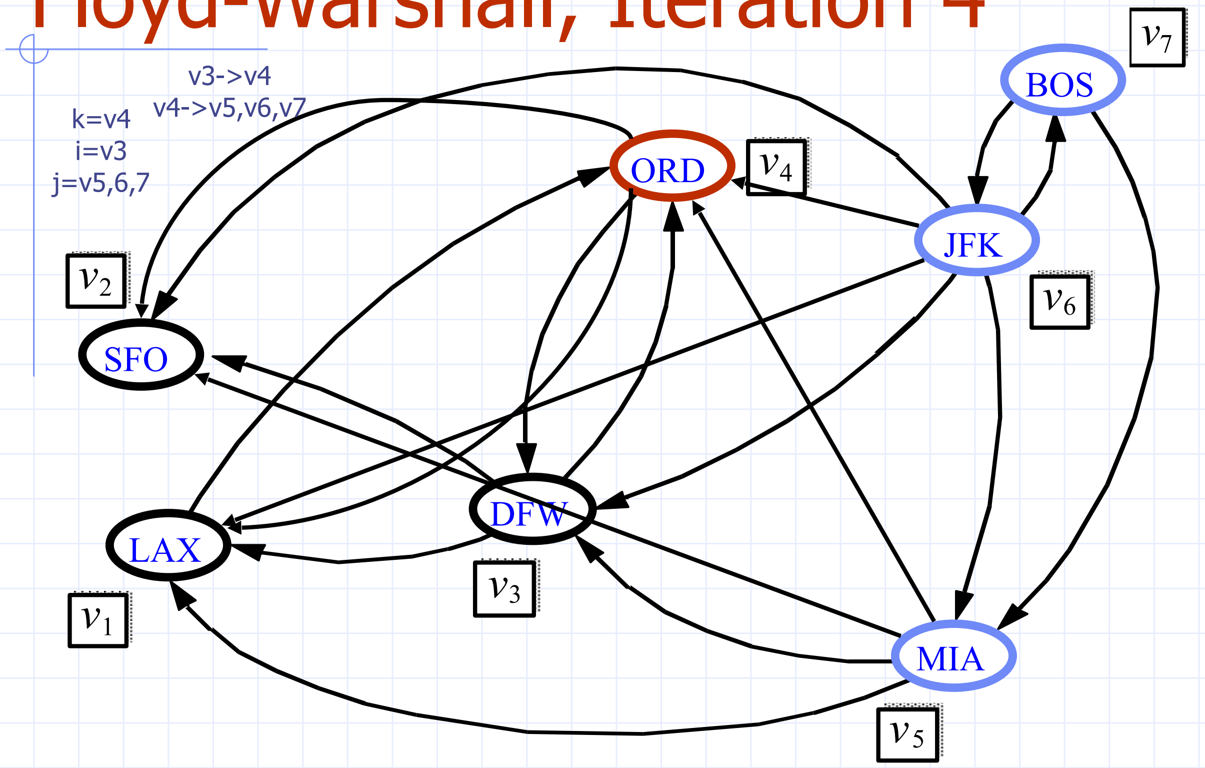
Floyd-Warshall, Iteration 4



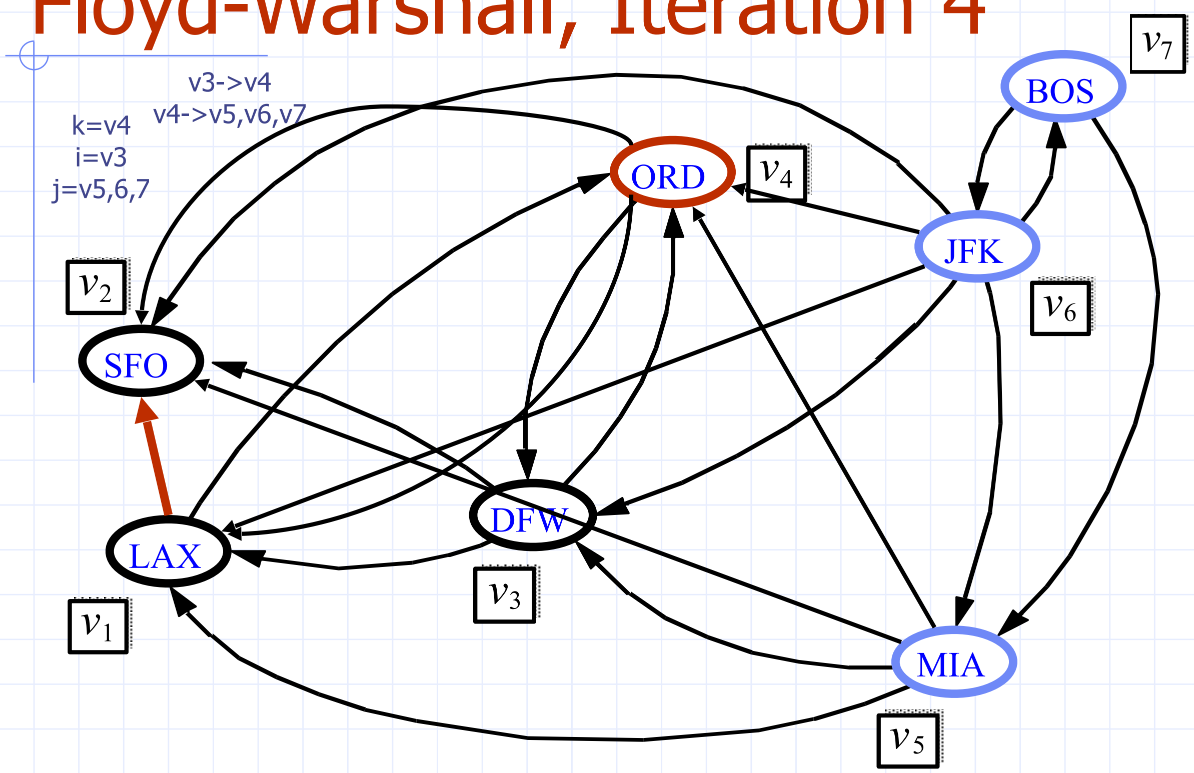
Floyd-Warshall, Iteration 4



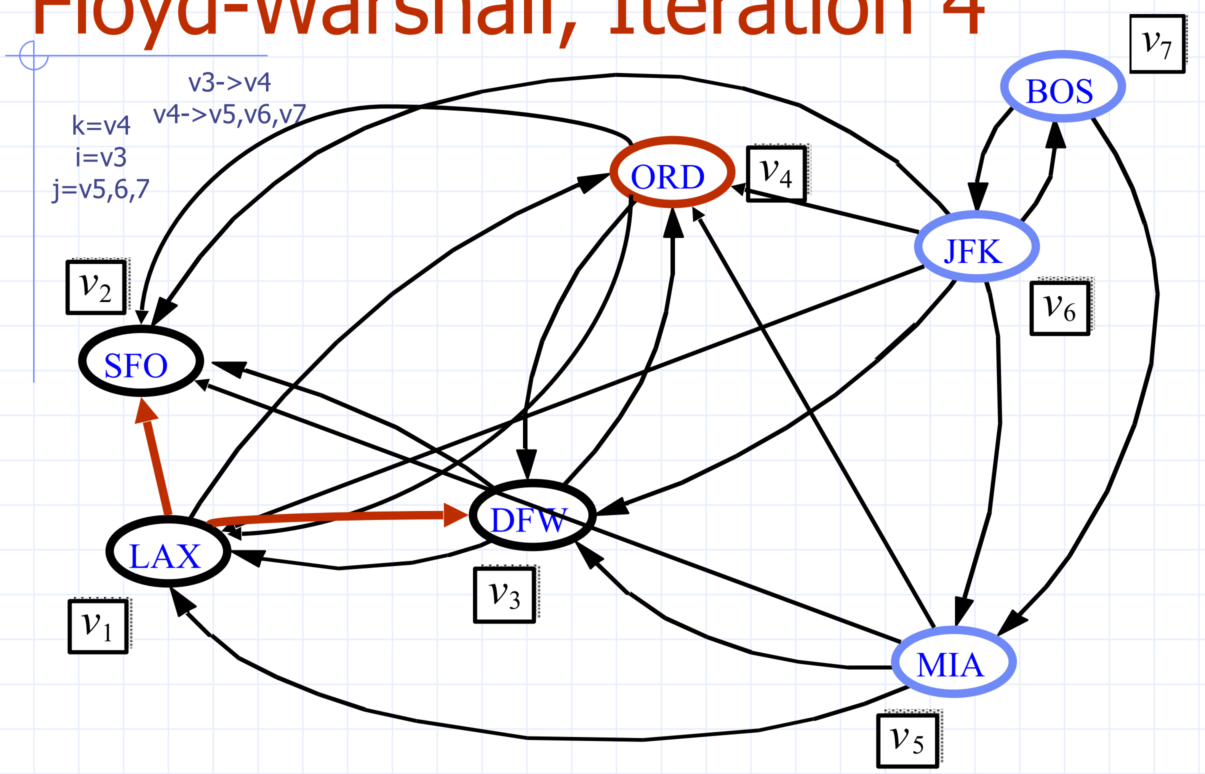
Floyd-Warshall, Iteration 4



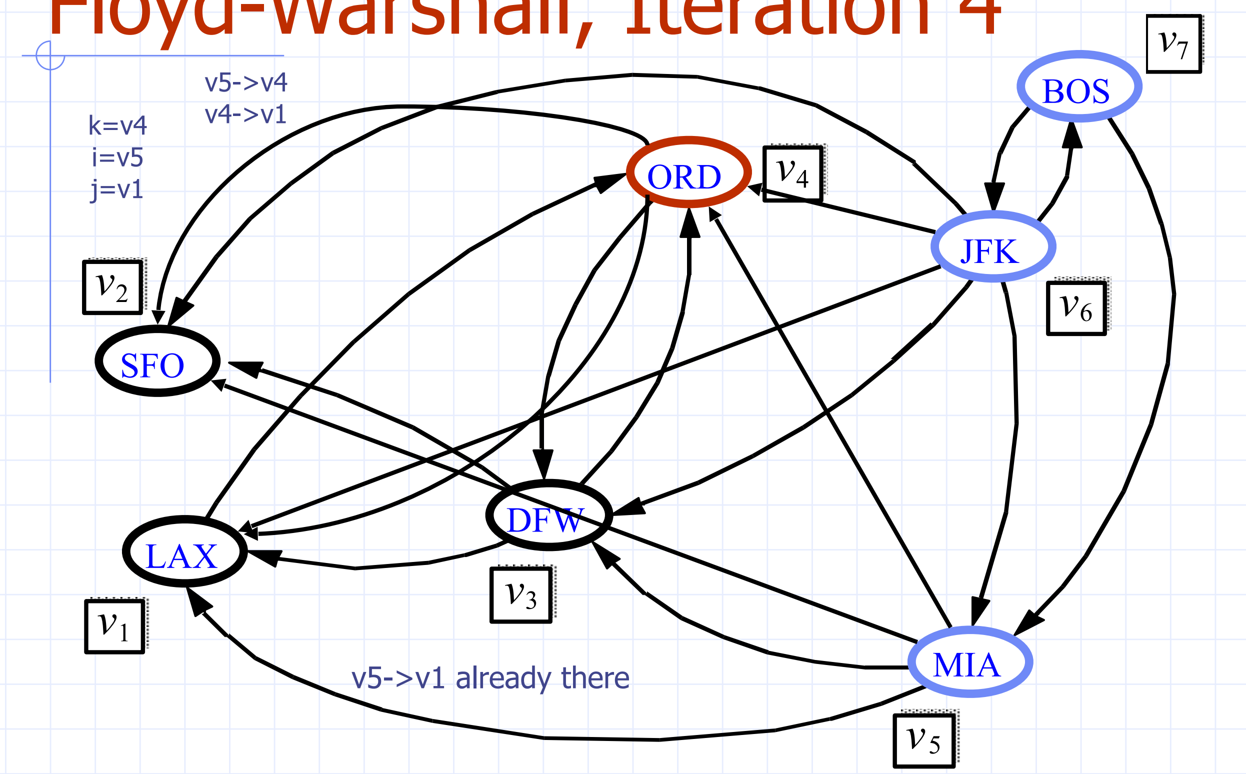
Floyd-Warshall, Iteration 4



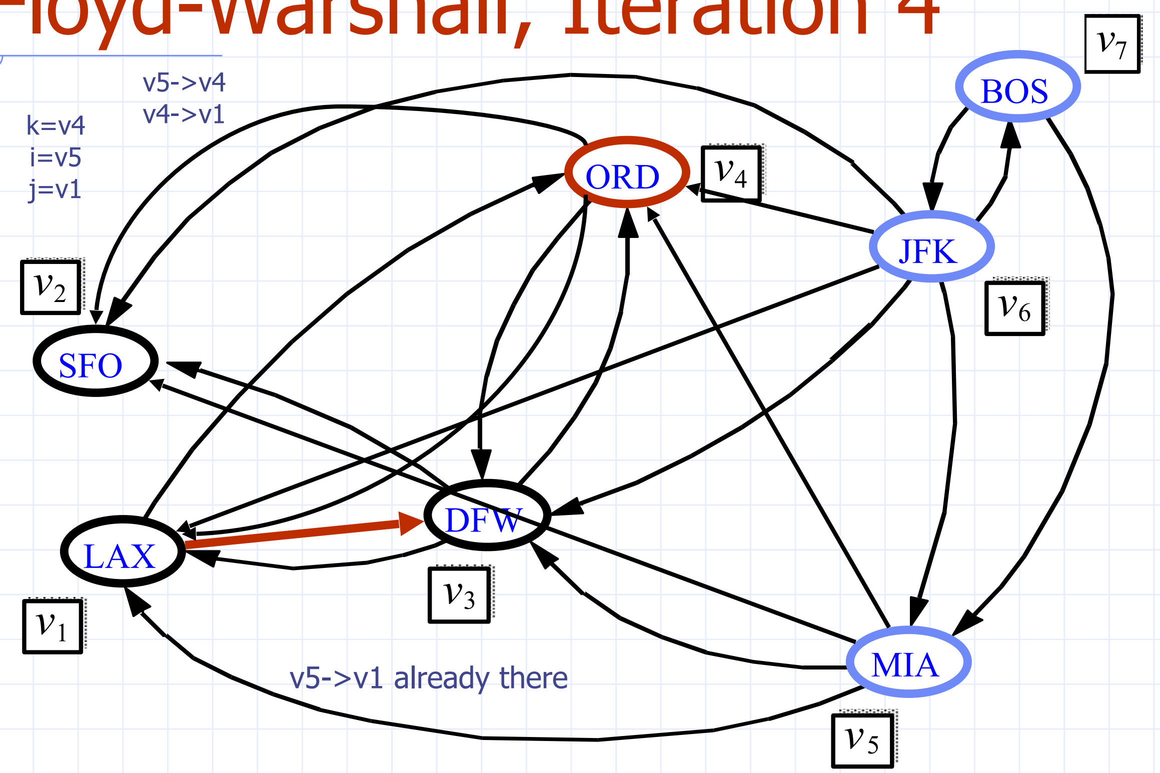
Floyd-Warshall, Iteration 4



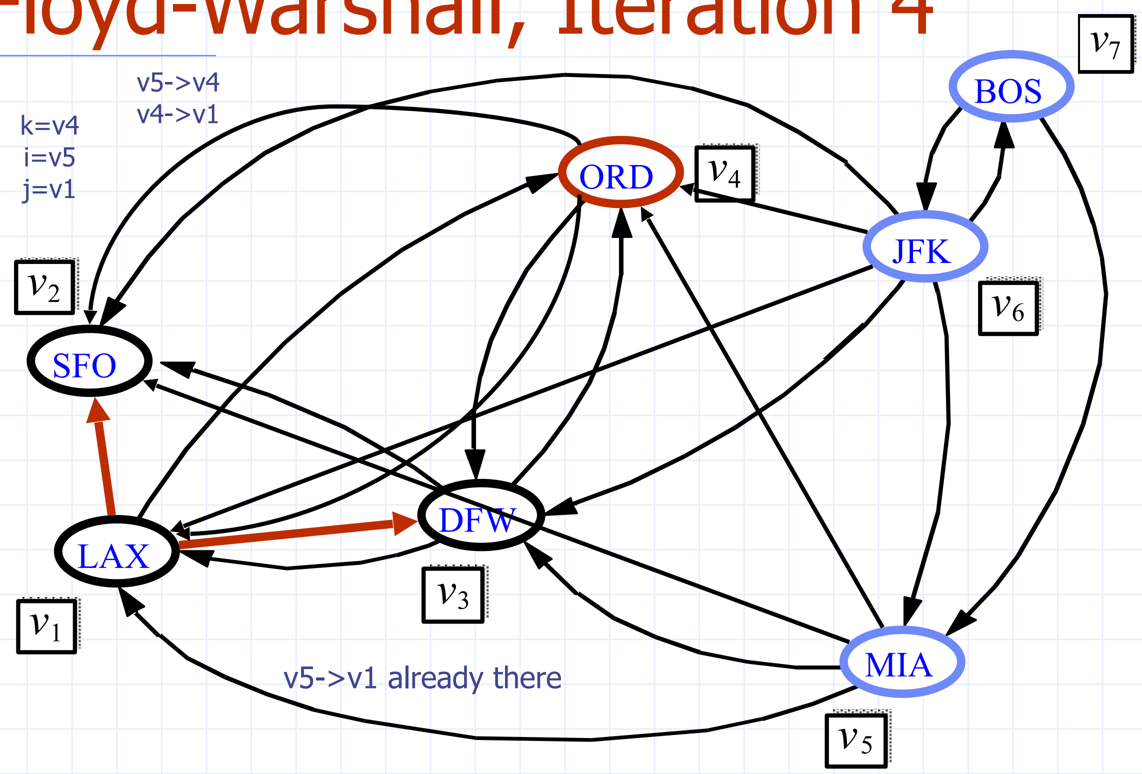
Floyd-Warshall, Iteration 4



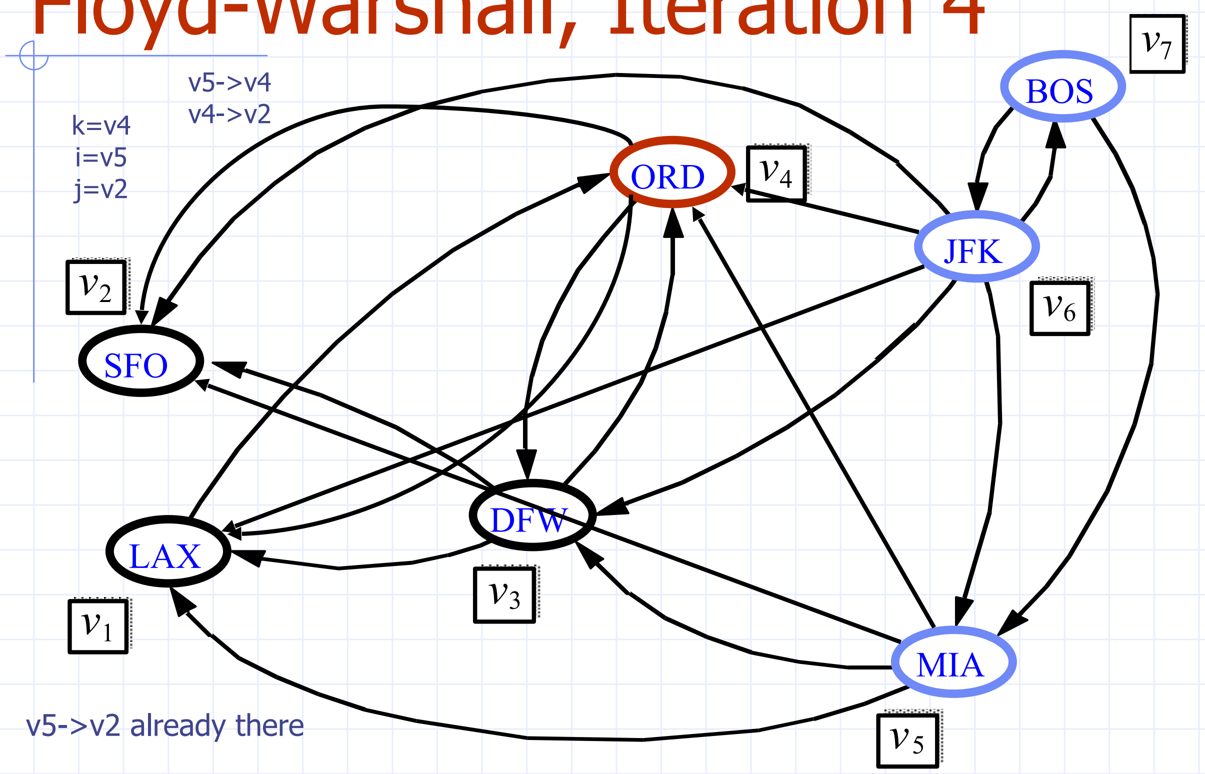
Floyd-Warshall, Iteration 4



Floyd-Warshall, Iteration 4



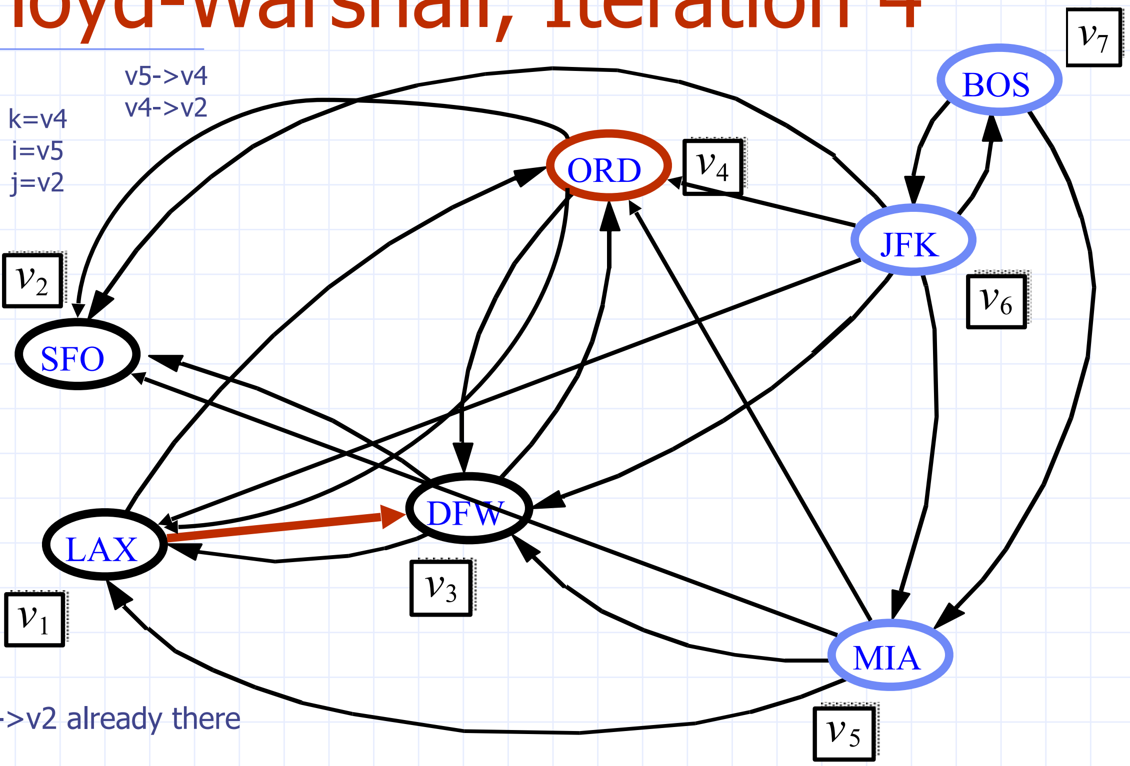
Floyd-Warshall, Iteration 4



Floyd-Warshall, Iteration 4

k=v4
i=v5
j=v2

v5->v4
v4->v2

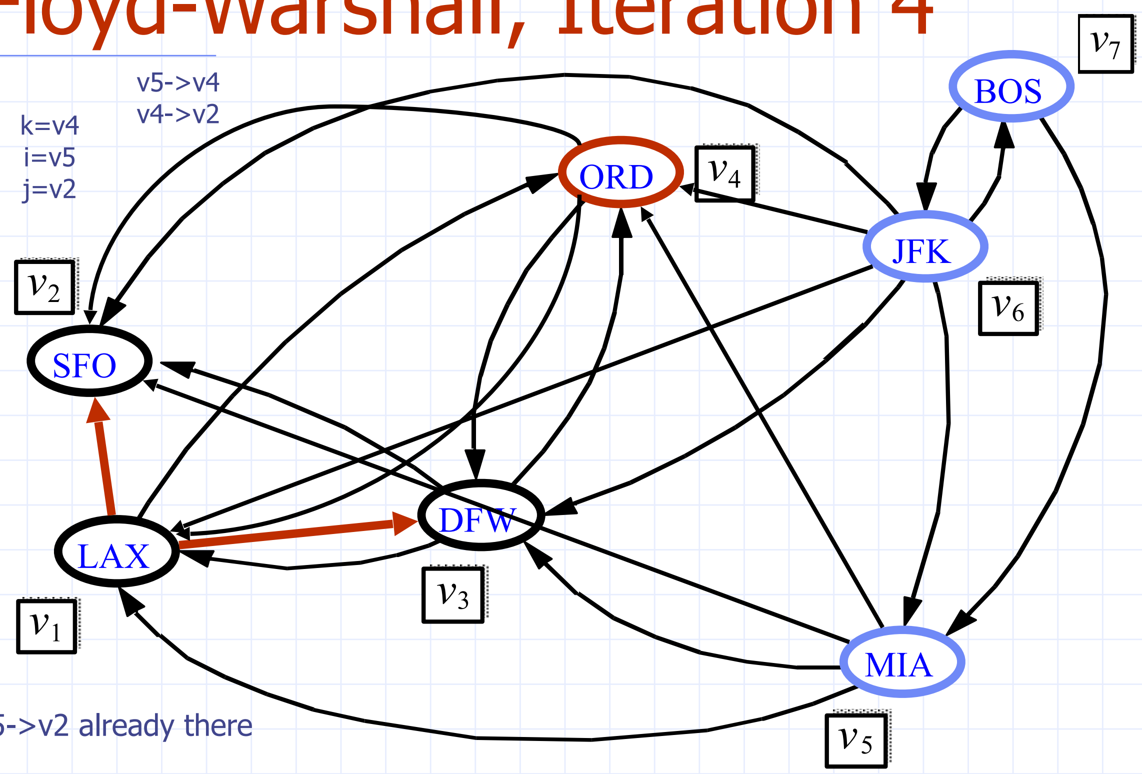


v5->v2 already there

Floyd-Warshall, Iteration 4

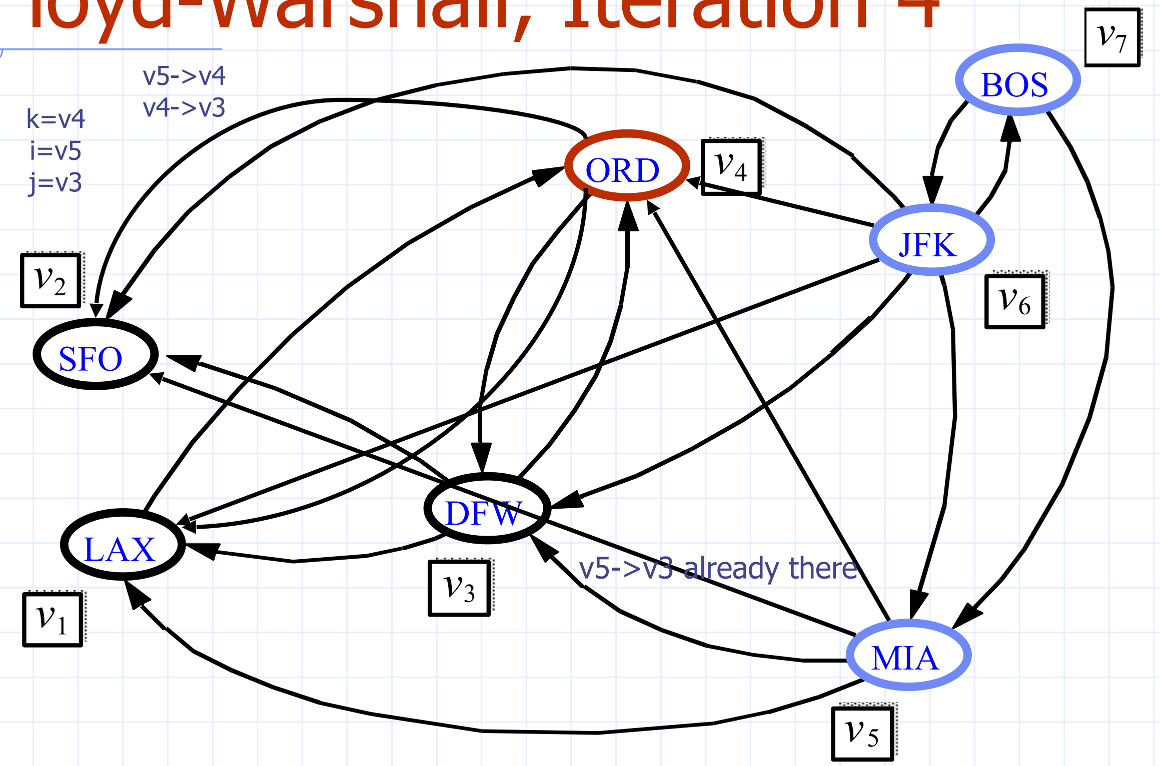
k=v4
i=v5
j=v2

v5->v4
v4->v2

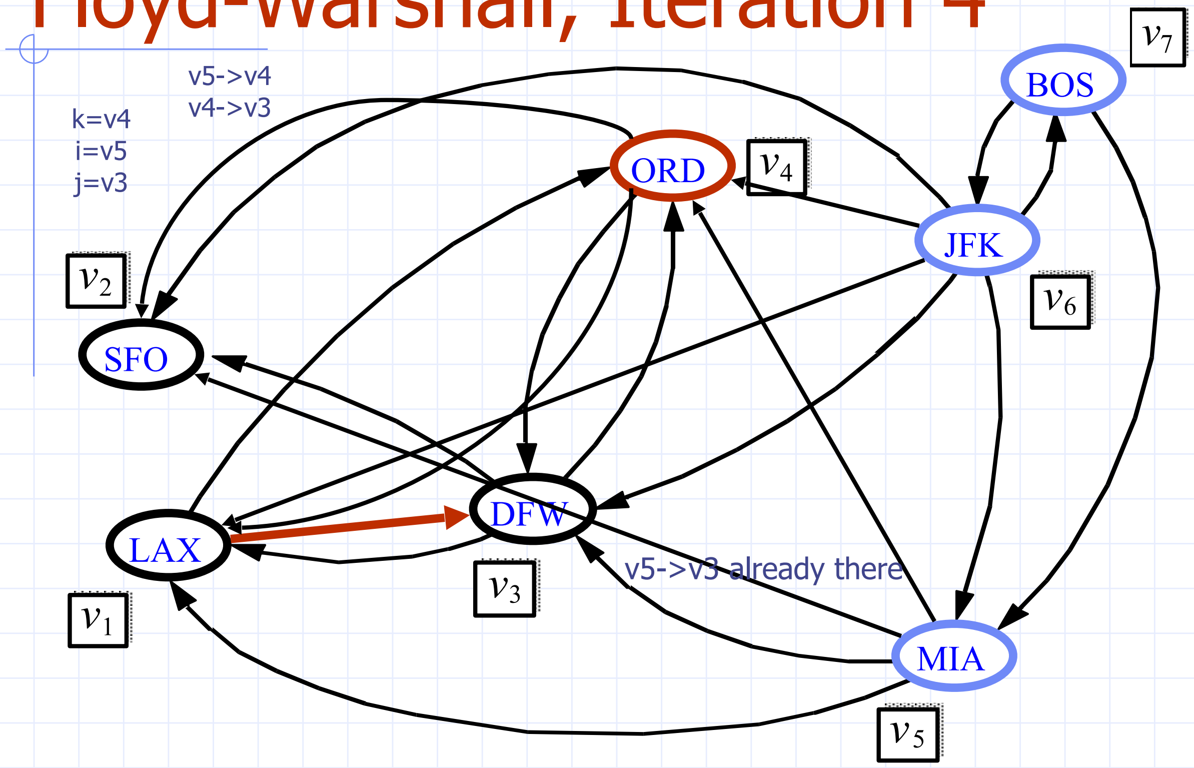


v5->v2 already there

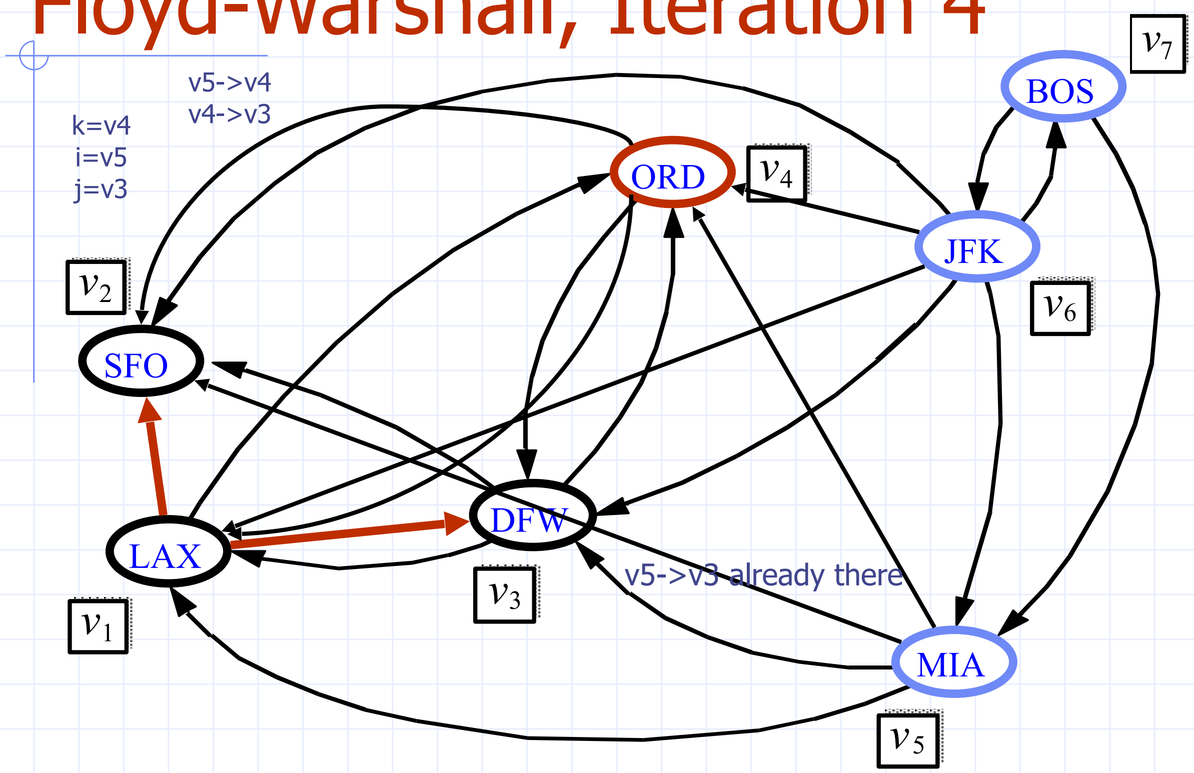
Floyd-Warshall, Iteration 4



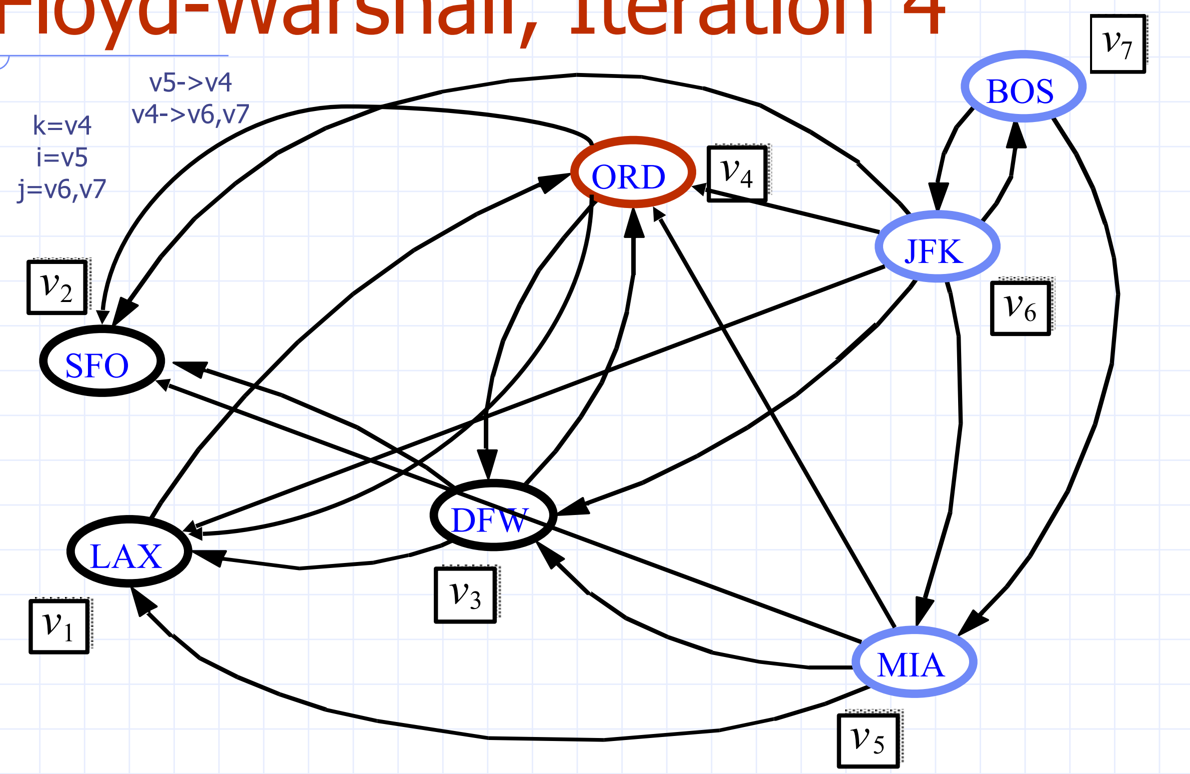
Floyd-Warshall, Iteration 4



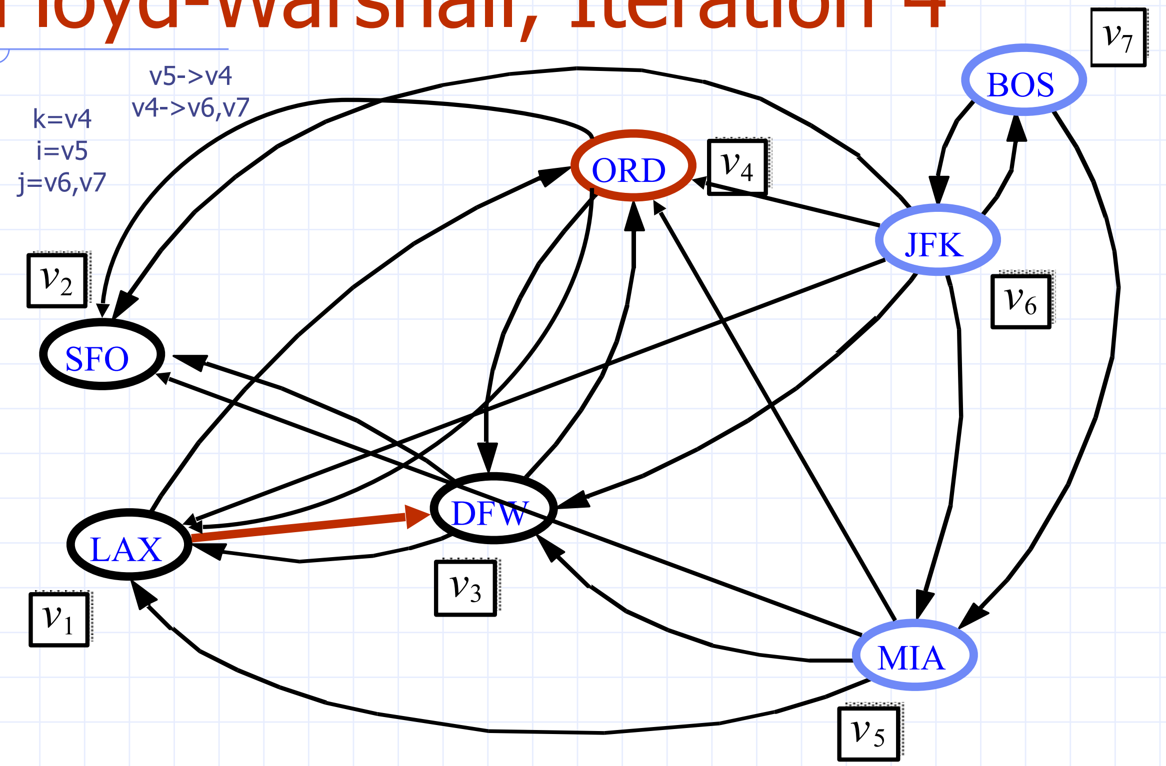
Floyd-Warshall, Iteration 4



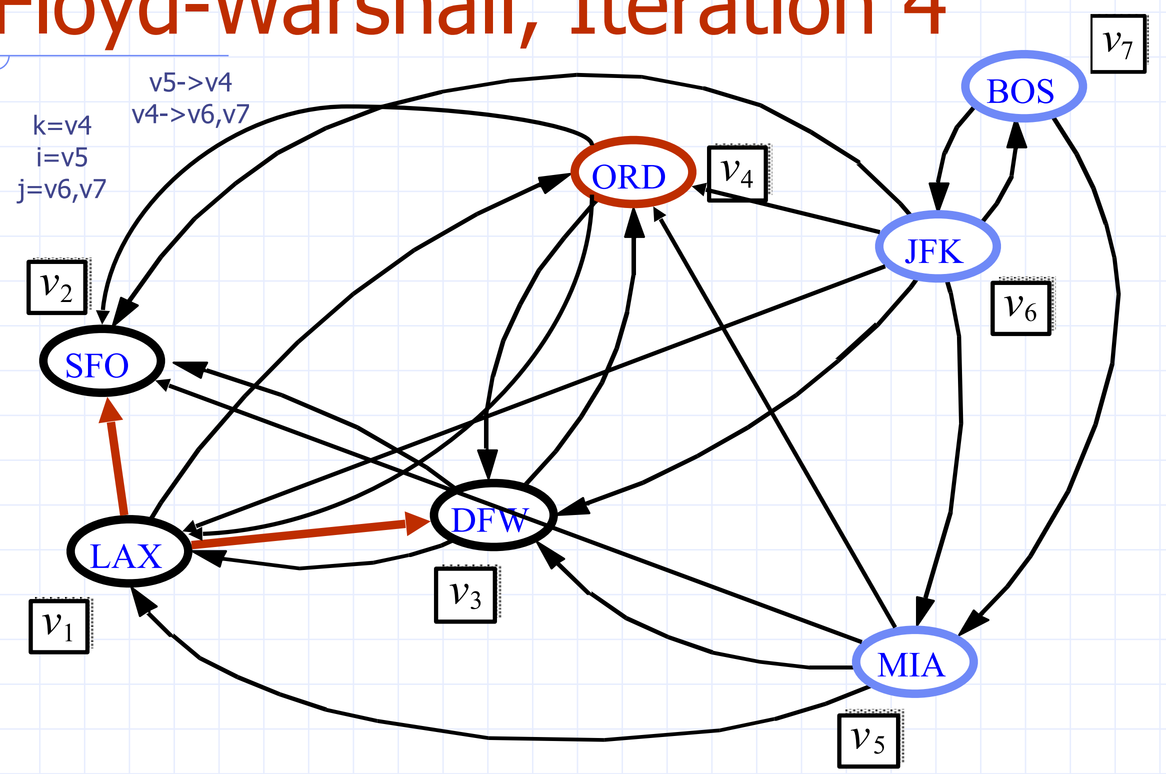
Floyd-Warshall, Iteration 4



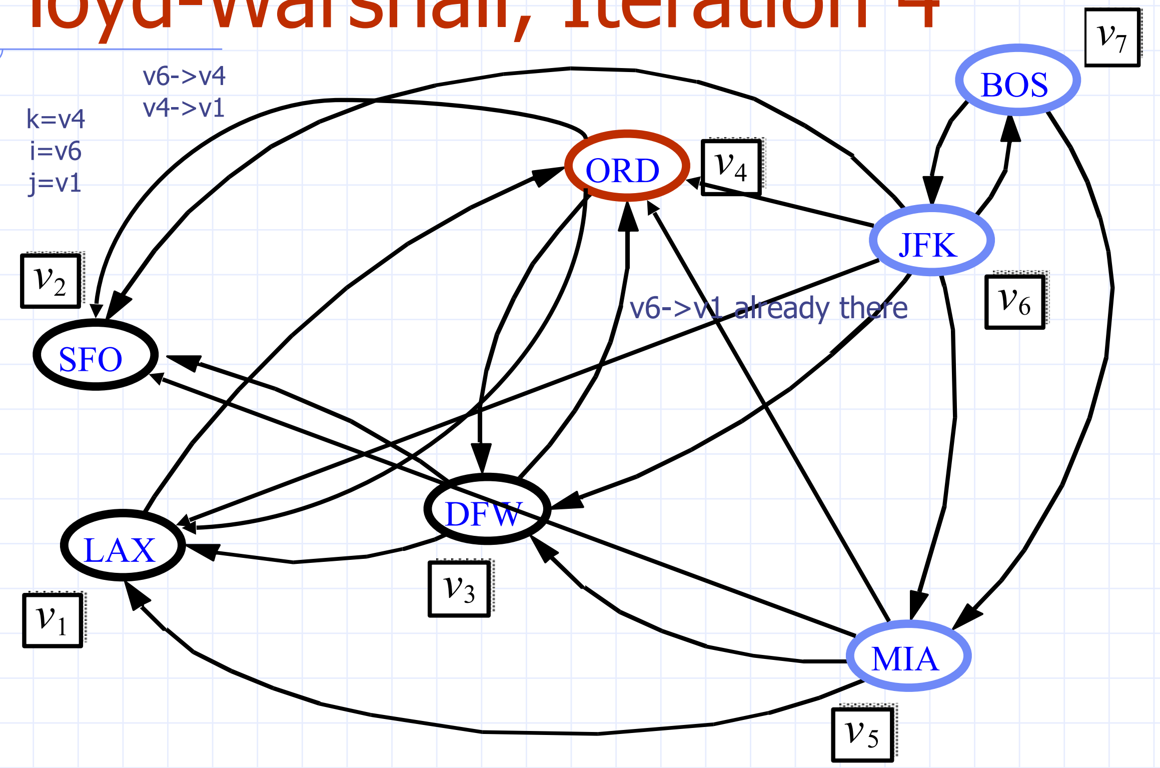
Floyd-Warshall, Iteration 4



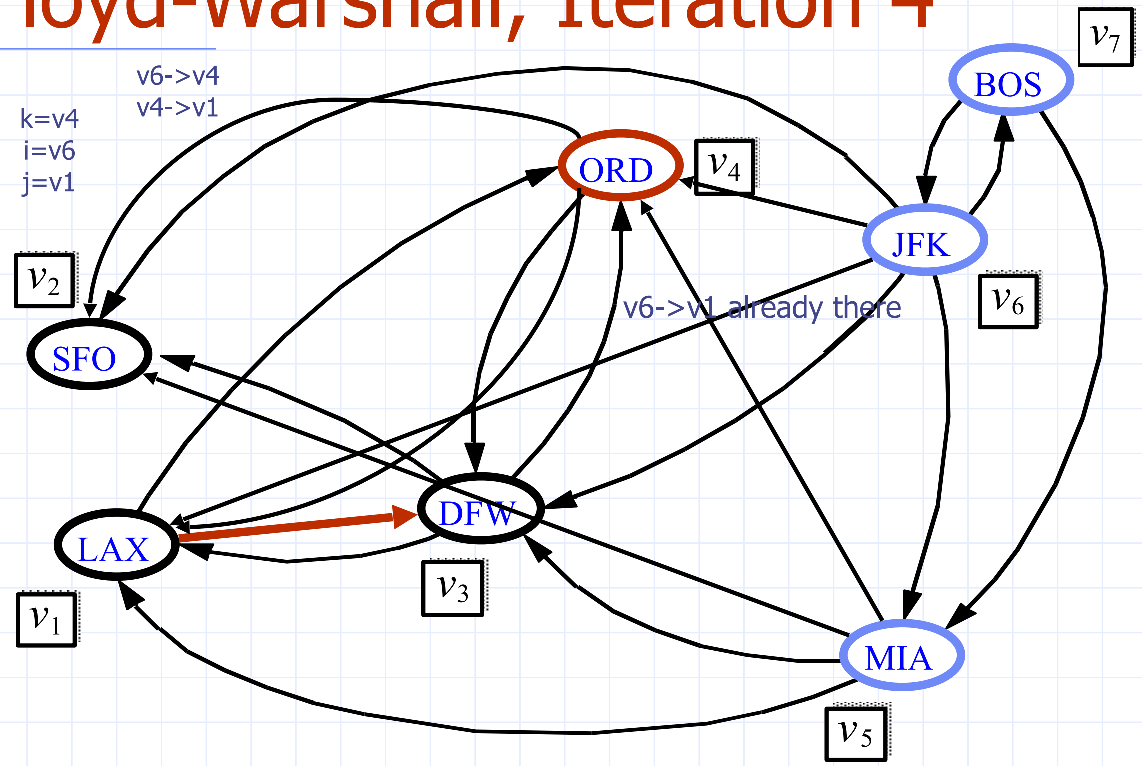
Floyd-Warshall, Iteration 4



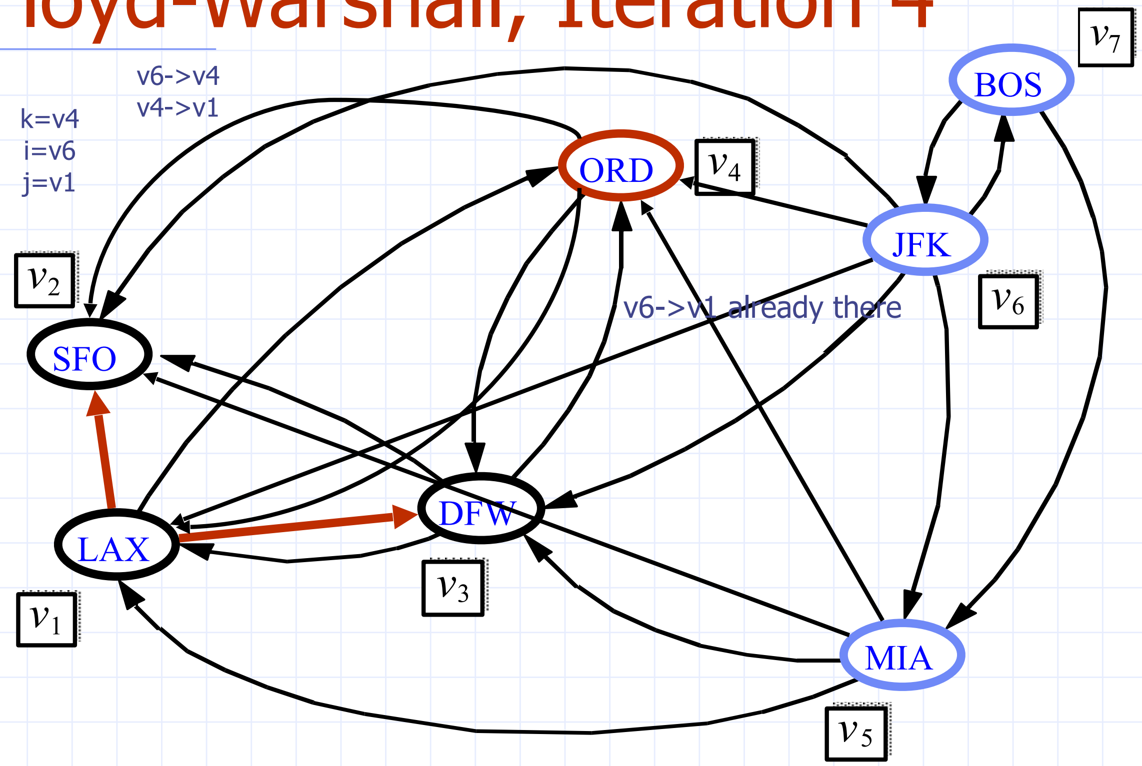
Floyd-Warshall, Iteration 4



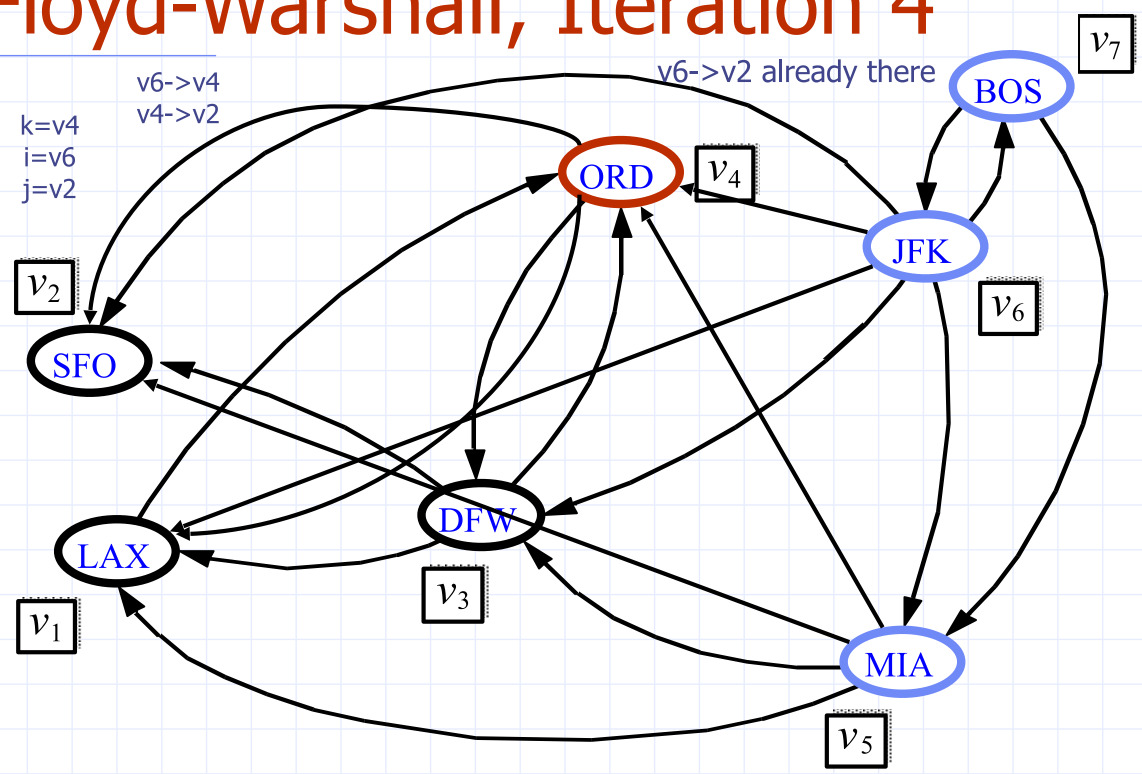
Floyd-Warshall, Iteration 4



Floyd-Warshall, Iteration 4



Floyd-Warshall, Iteration 4

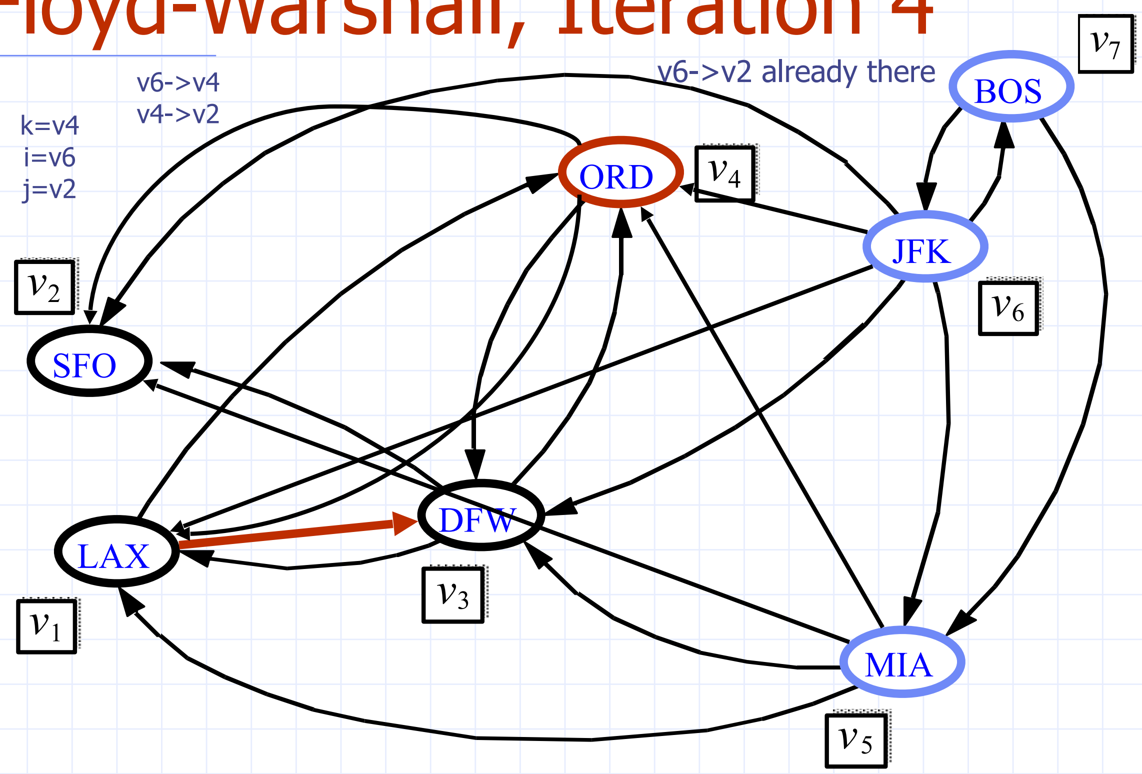


Floyd-Warshall, Iteration 4

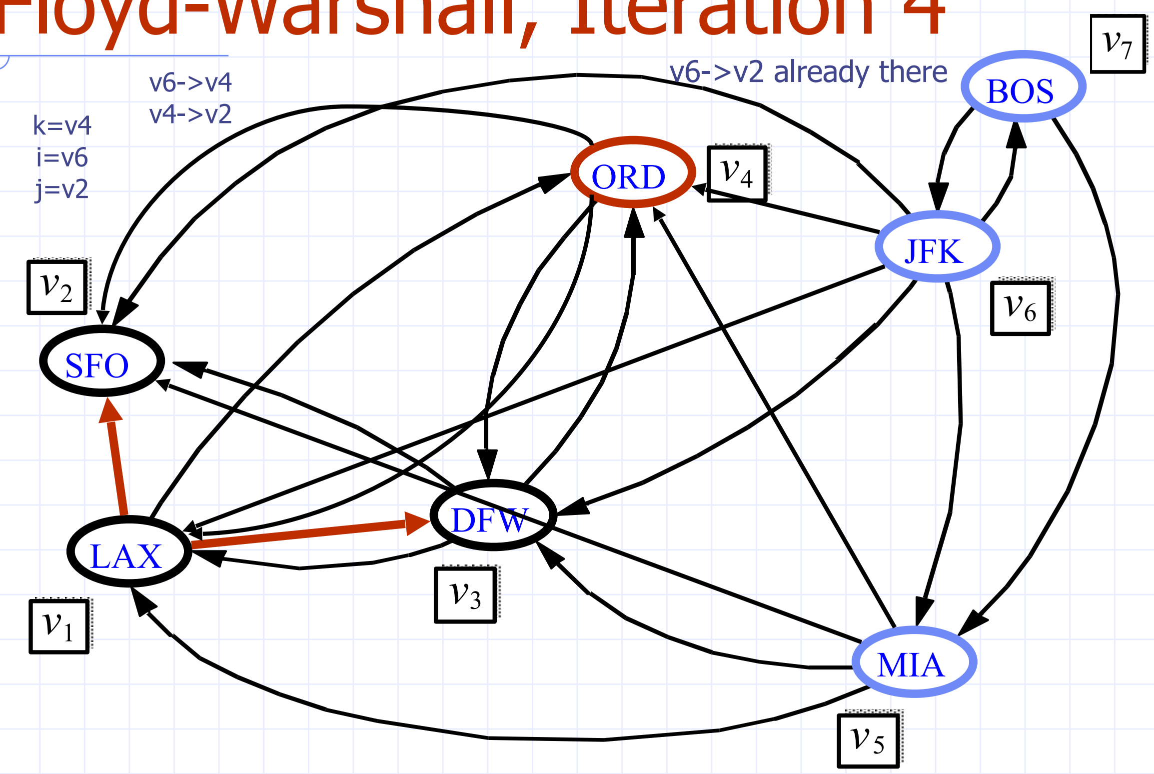
k=v4
i=v6
j=v2

v6->v4
v4->v2

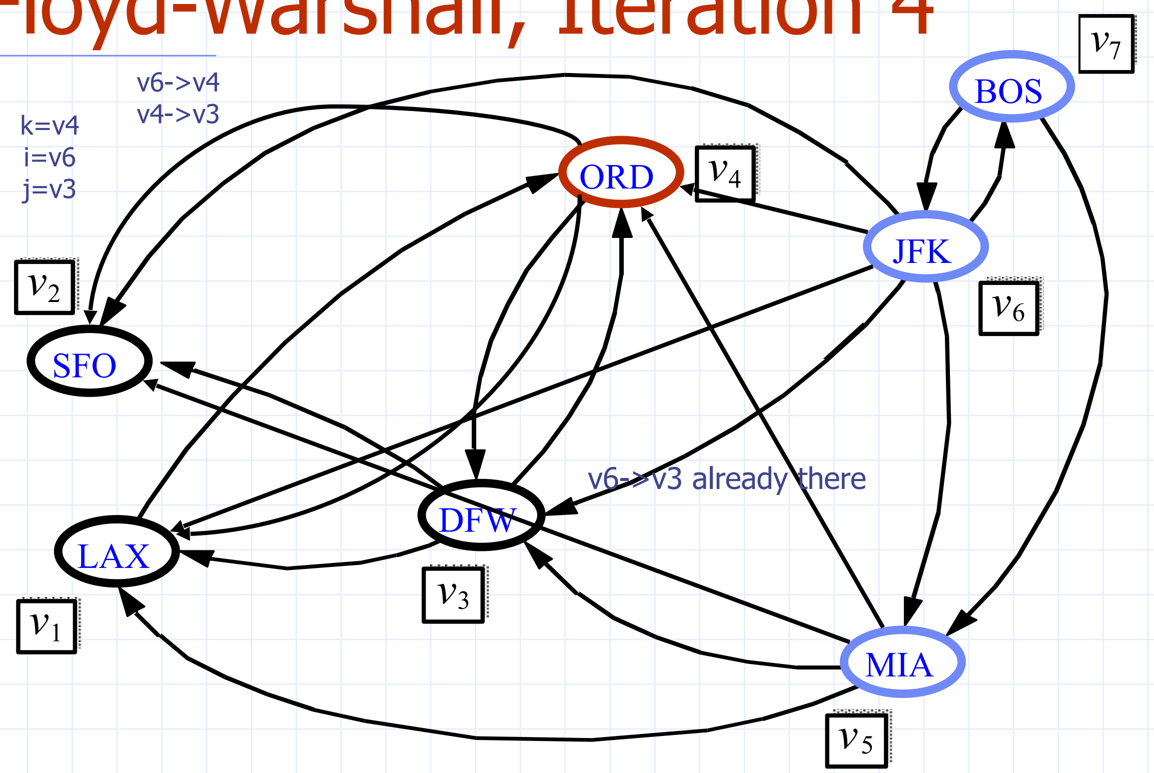
v6->v2 already there



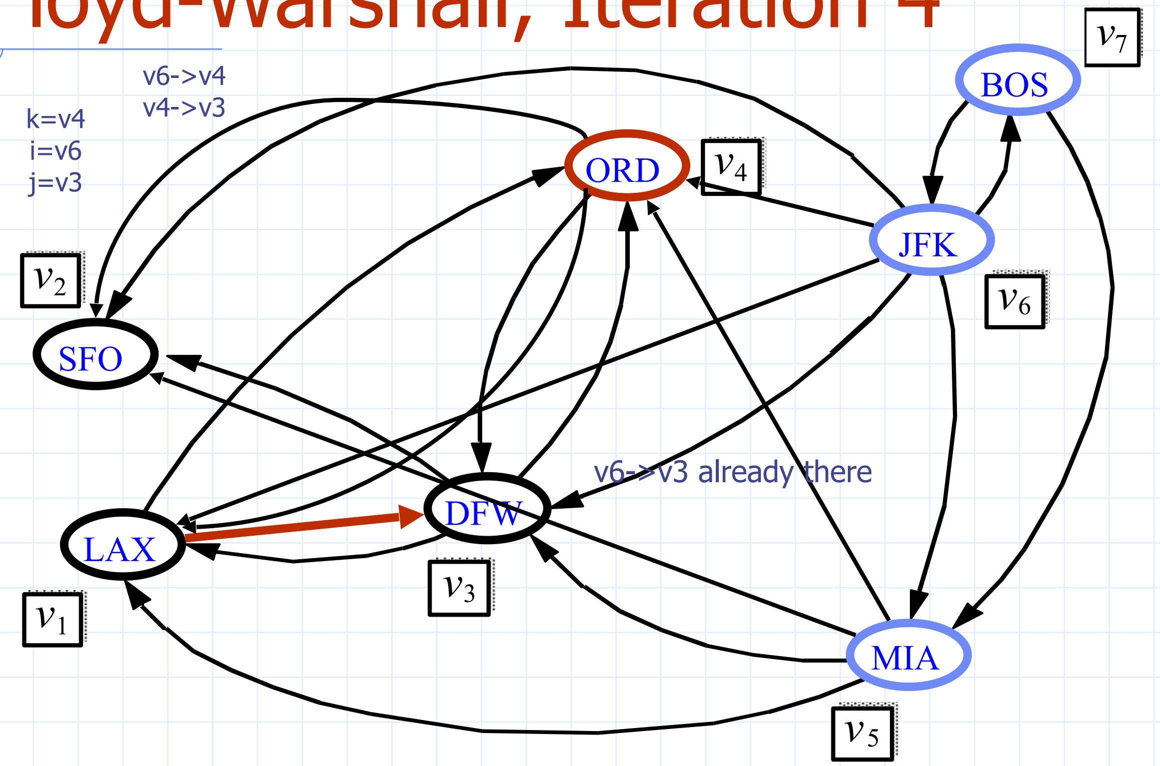
Floyd-Warshall, Iteration 4



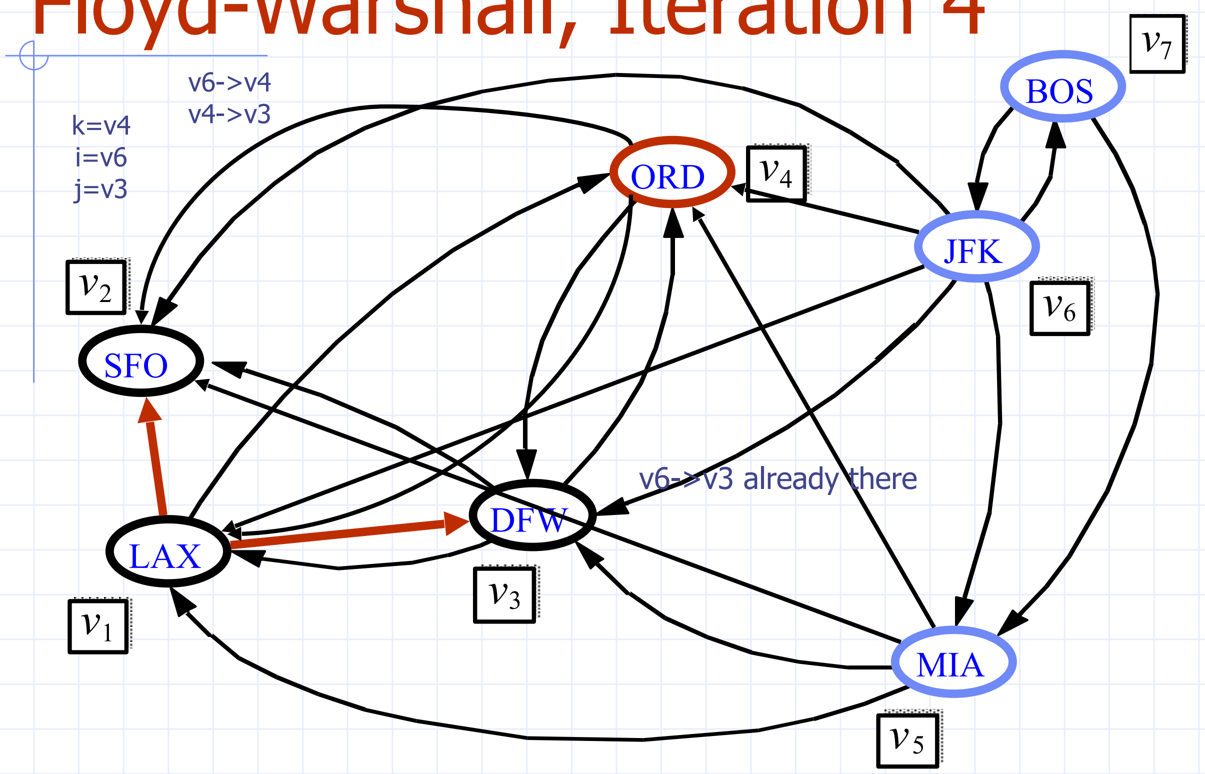
Floyd-Warshall, Iteration 4



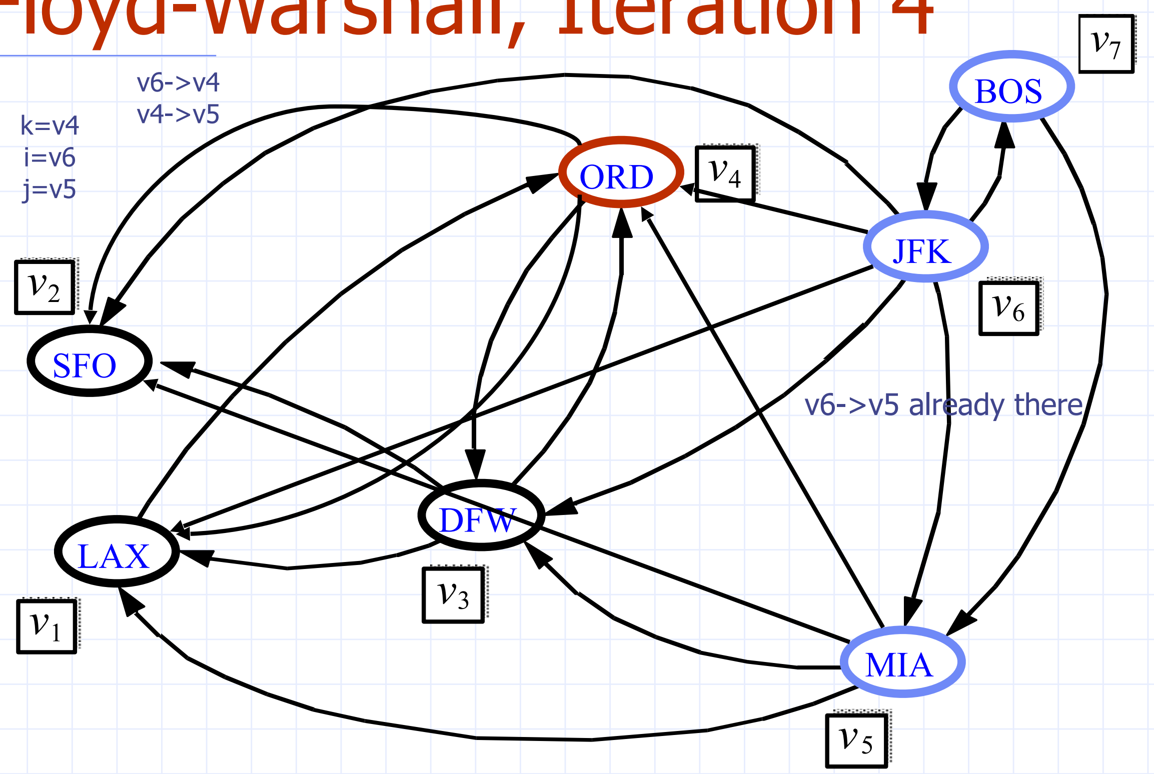
Floyd-Warshall, Iteration 4



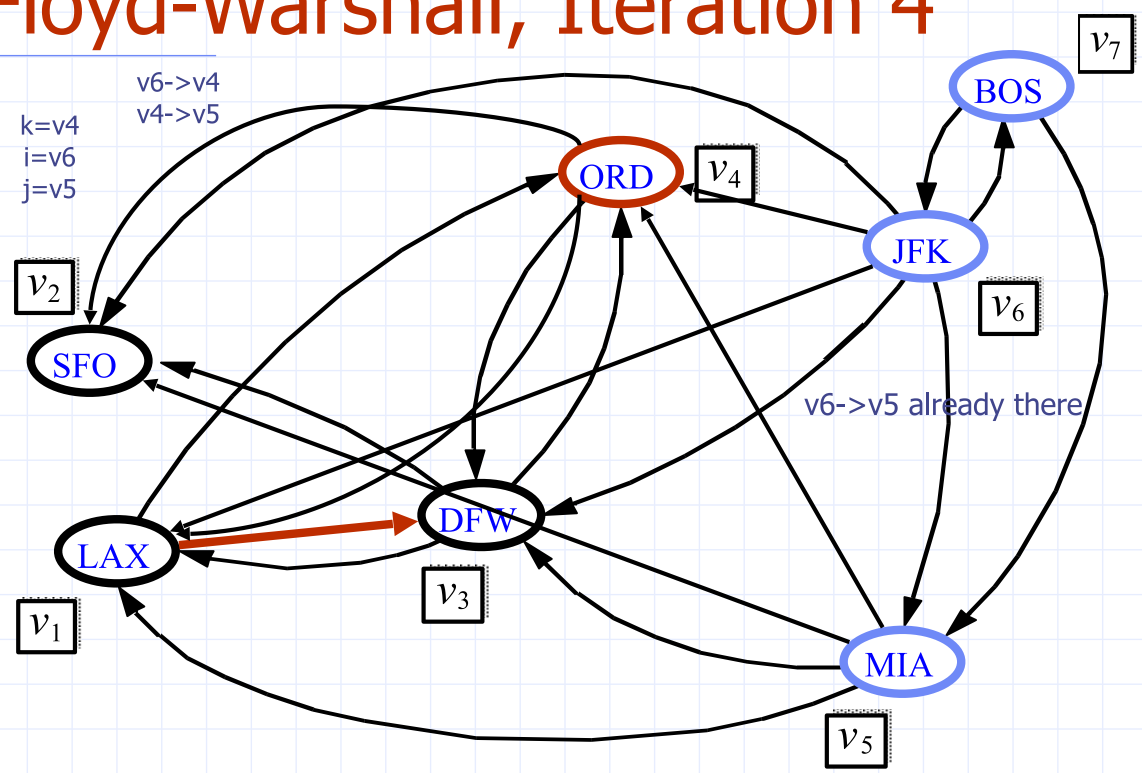
Floyd-Warshall, Iteration 4



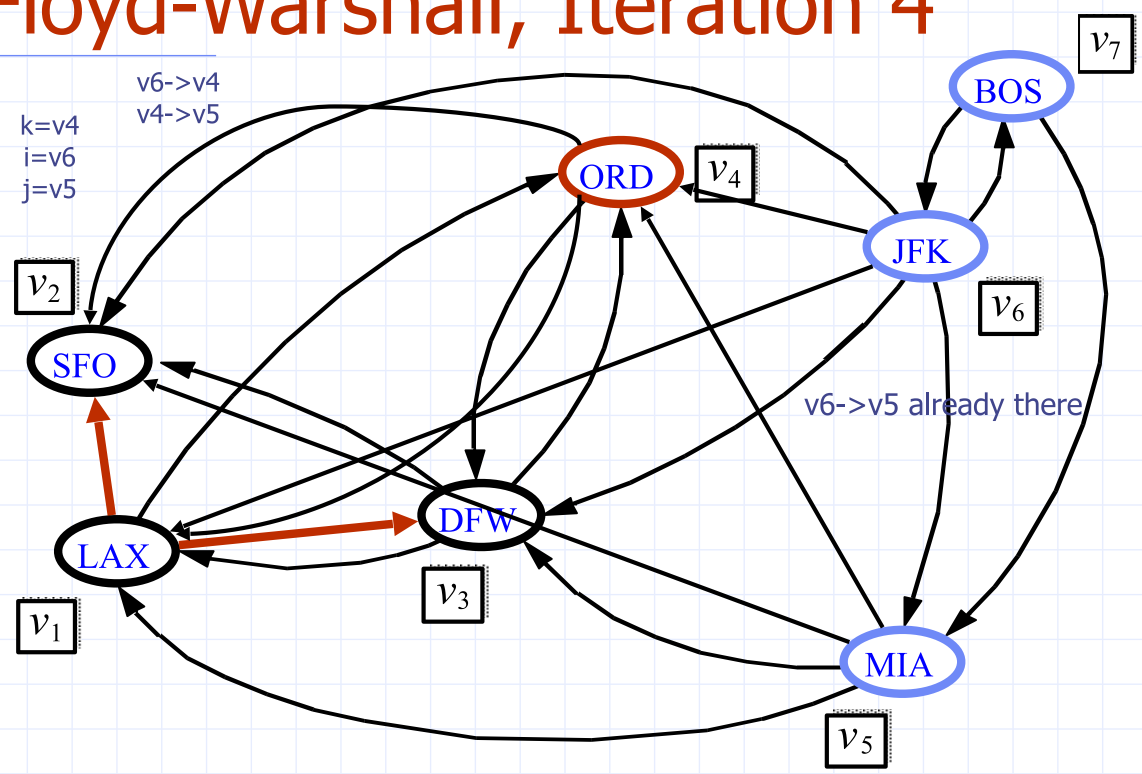
Floyd-Warshall, Iteration 4



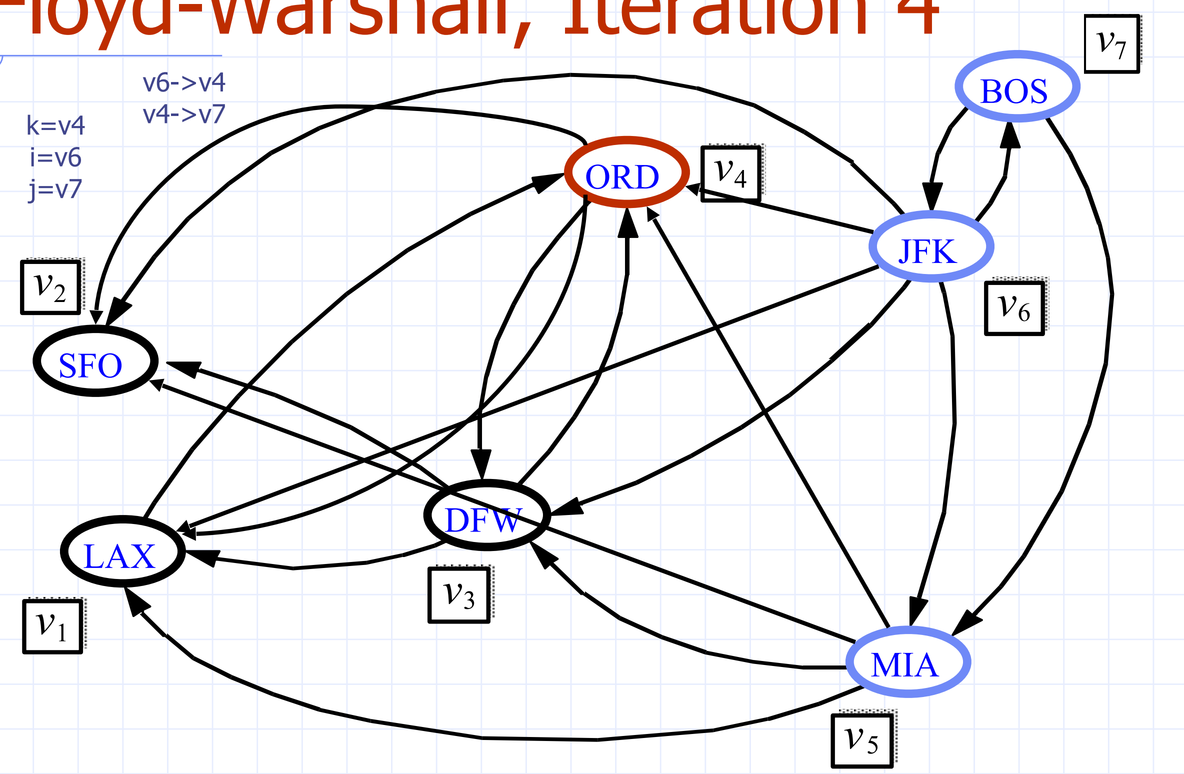
Floyd-Warshall, Iteration 4



Floyd-Warshall, Iteration 4



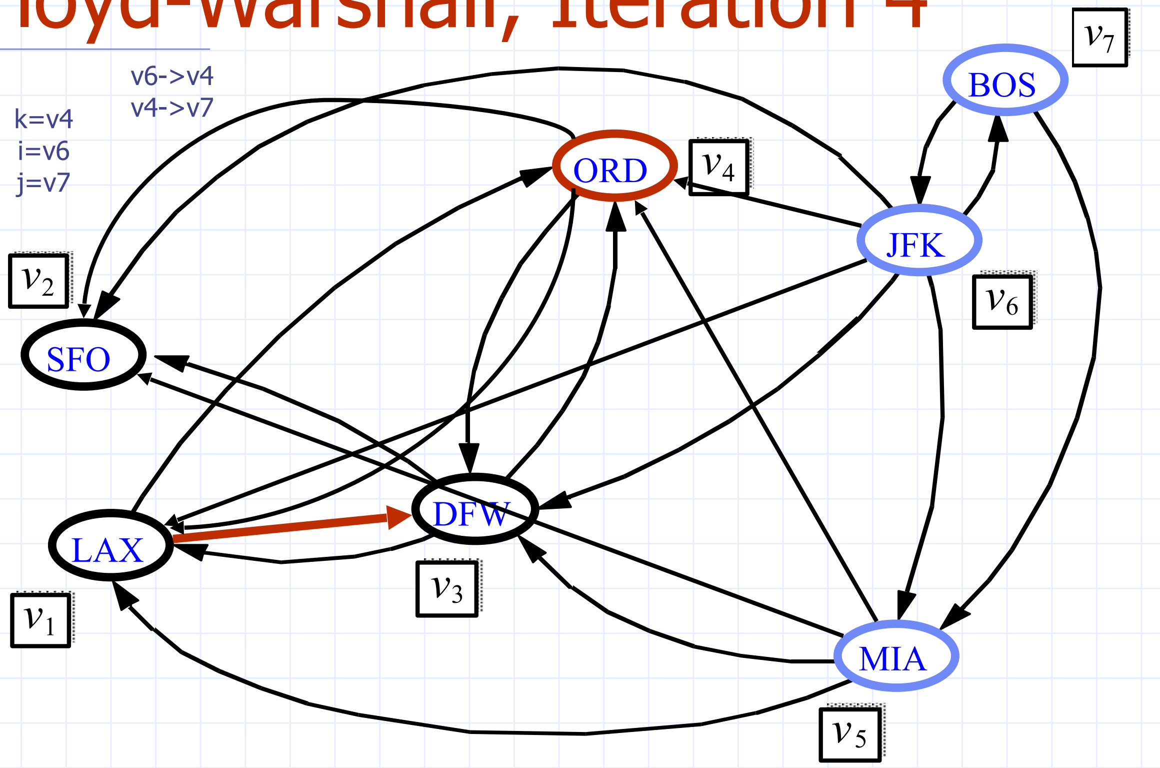
Floyd-Warshall, Iteration 4



Floyd-Warshall, Iteration 4

k=v4
i=v6
j=v7

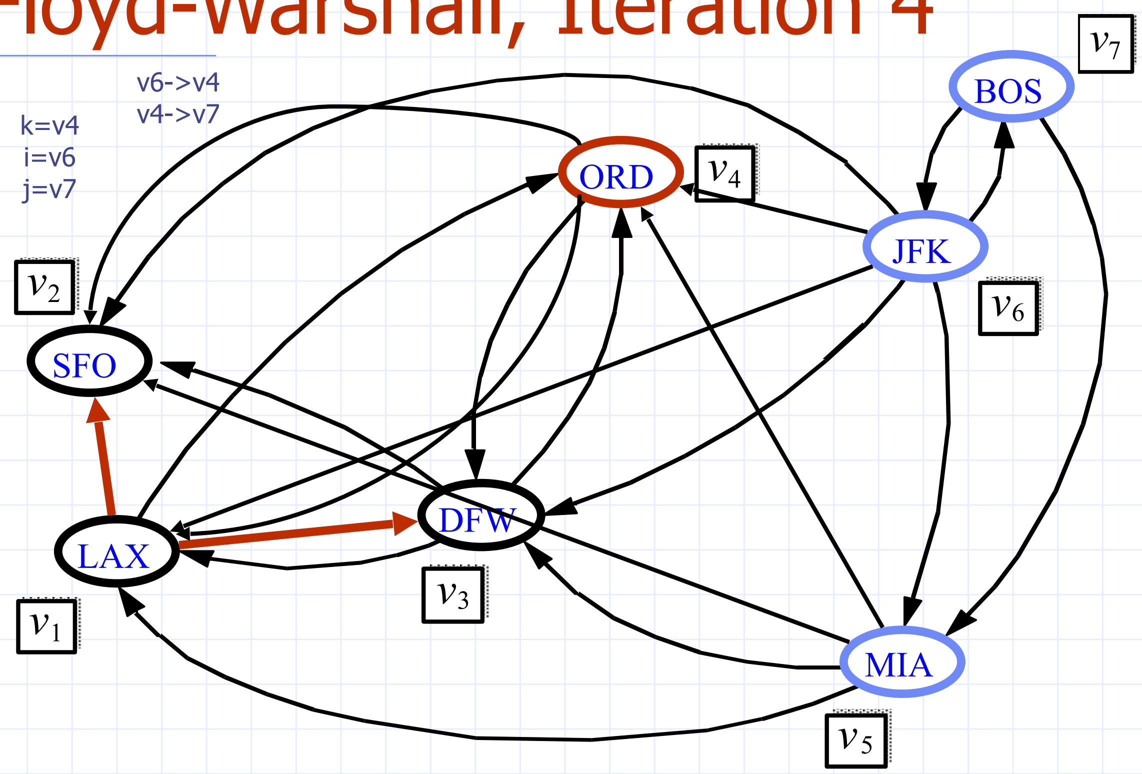
v6->v4
v4->v7



Floyd-Warshall, Iteration 4

k=v4
i=v6
j=v7

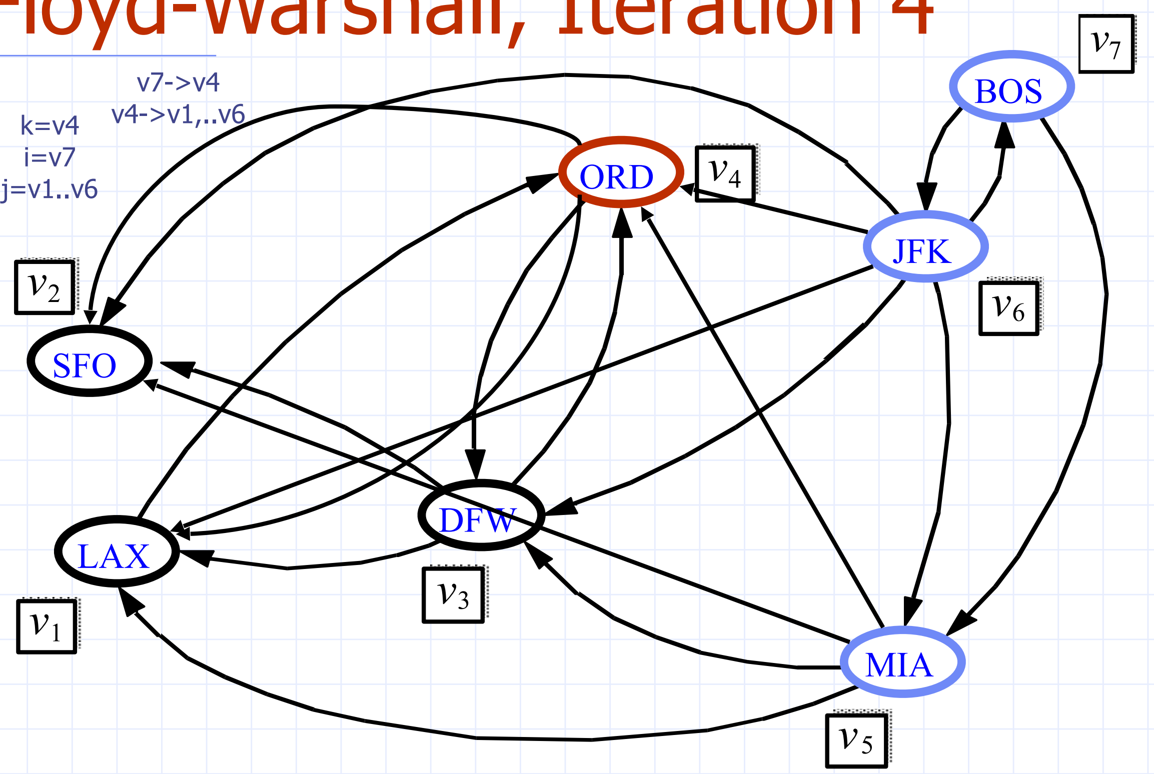
v6->v4
v4->v7



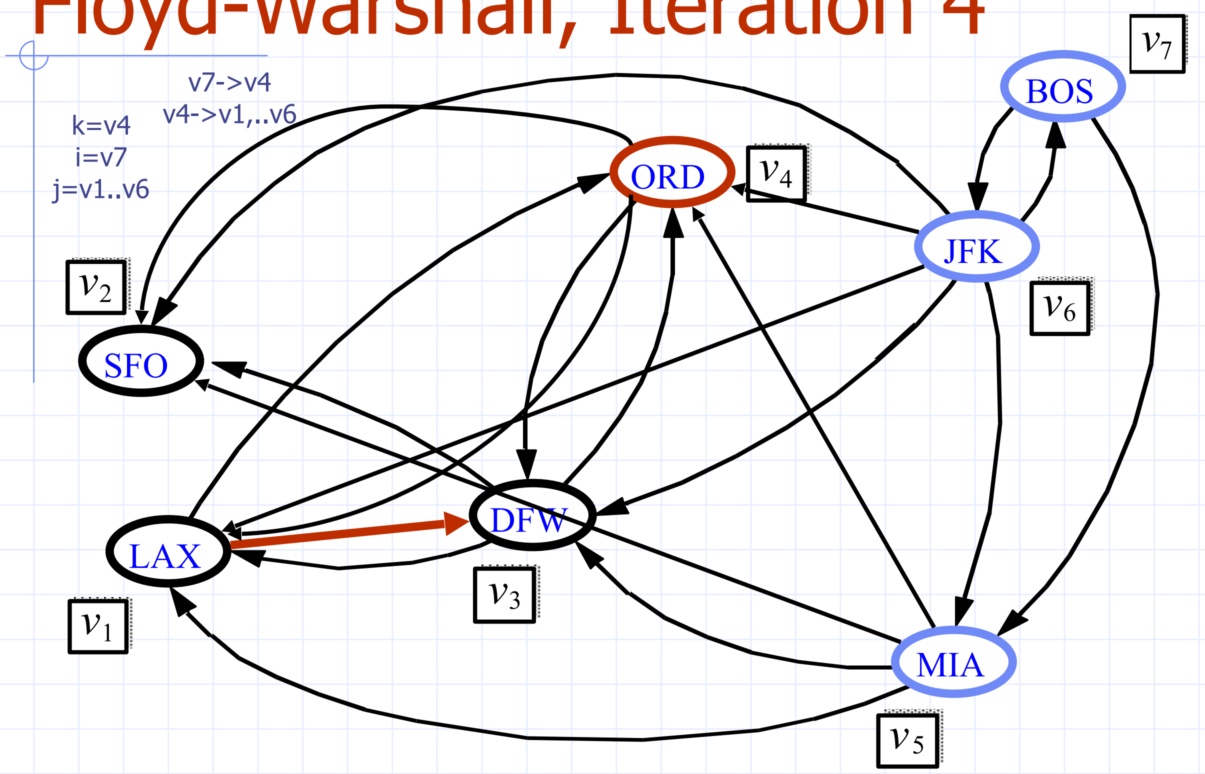
Floyd-Warshall, Iteration 4

k=v4
i=v7
j=v1..v6

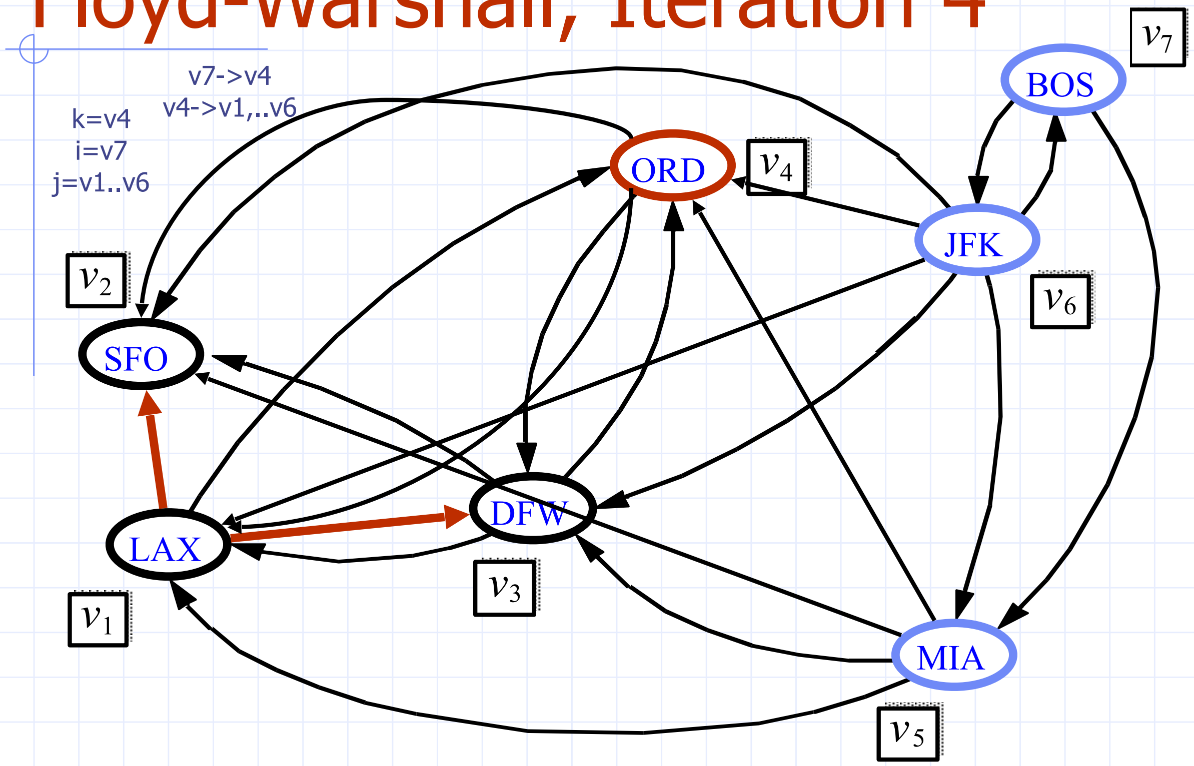
v7->v4
v4->v1,..v6



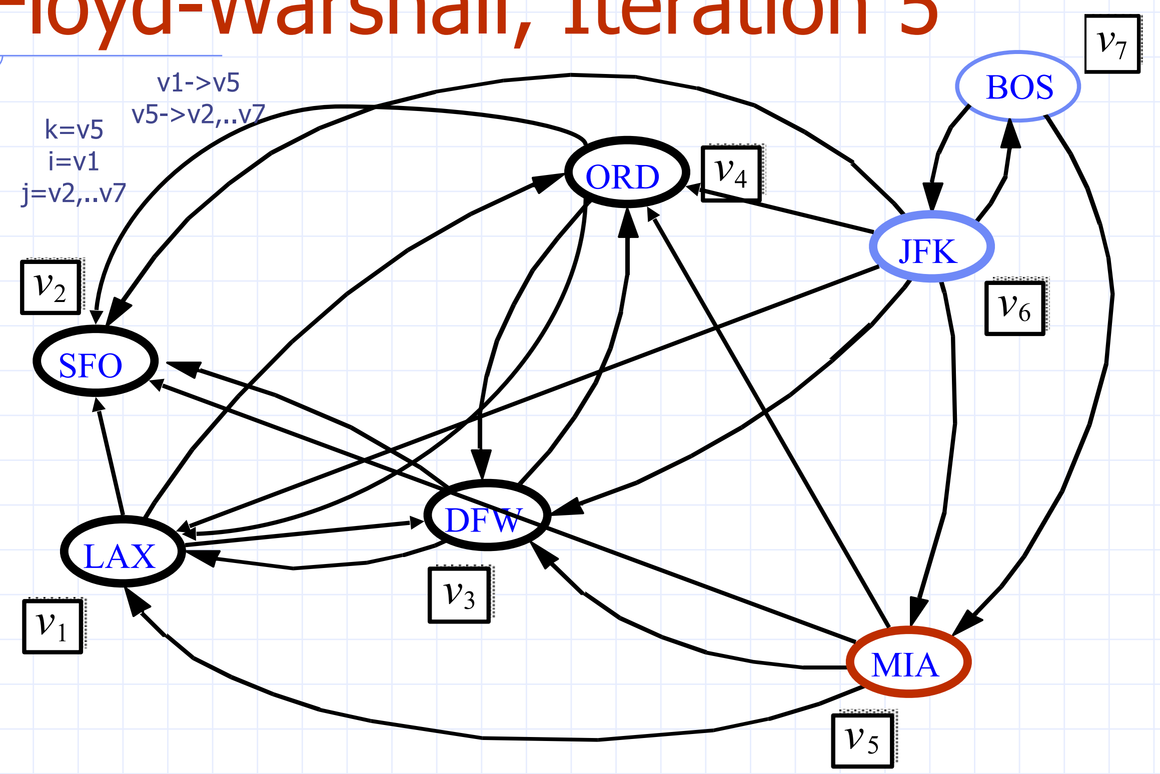
Floyd-Warshall, Iteration 4



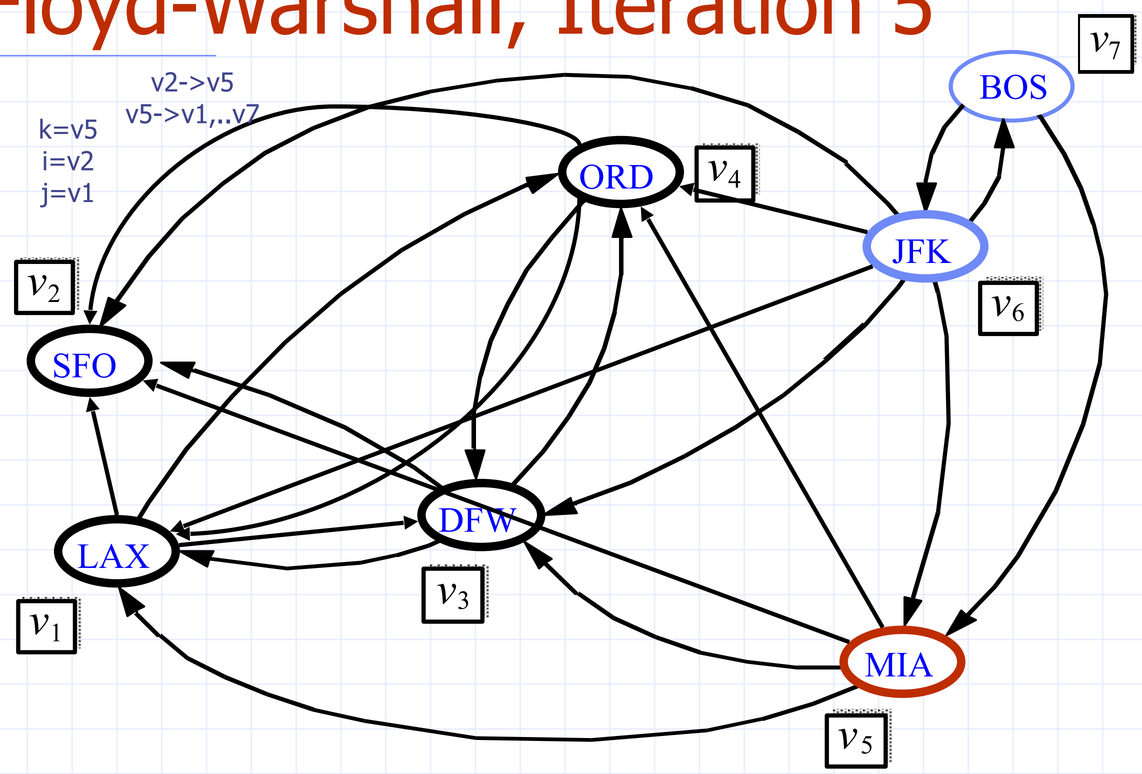
Floyd-Warshall, Iteration 4



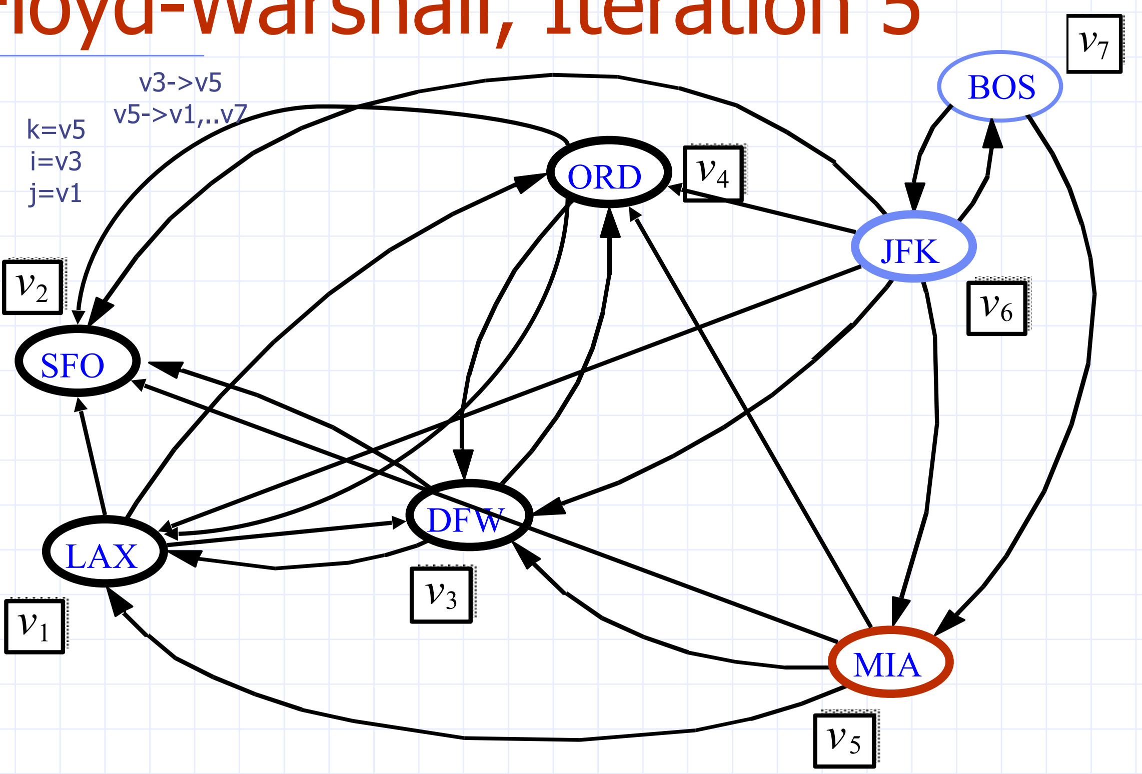
Floyd-Warshall, Iteration 5



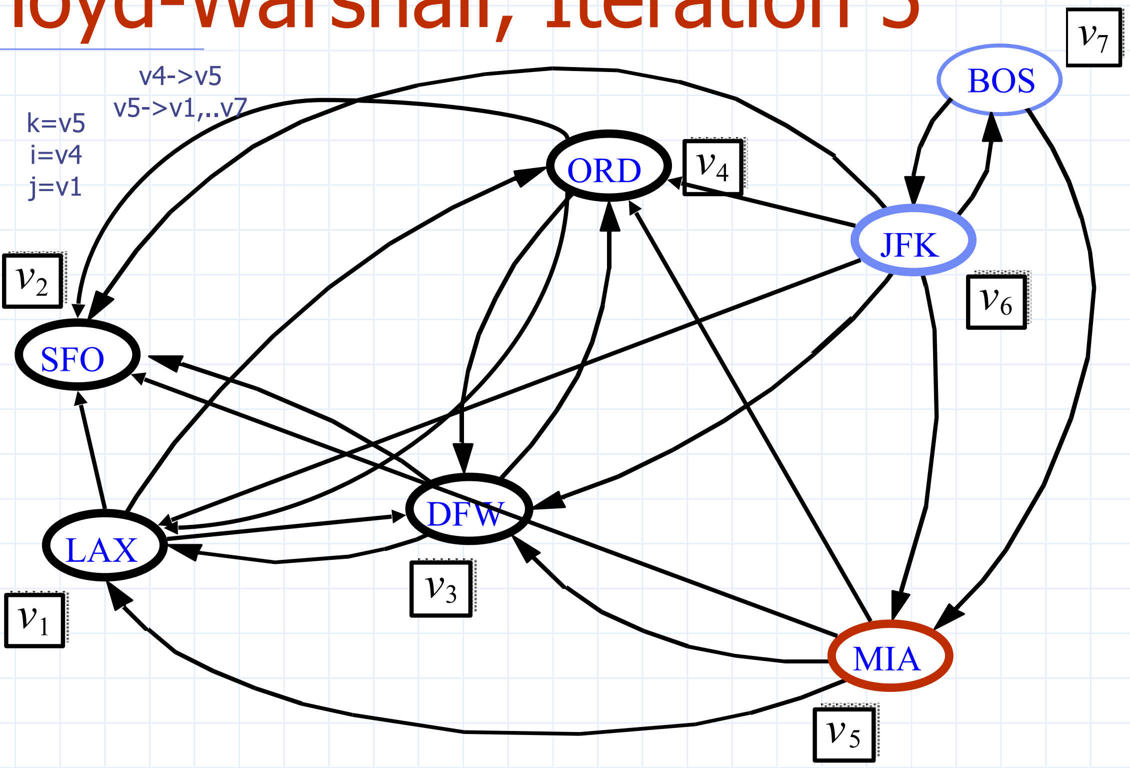
Floyd-Warshall, Iteration 5



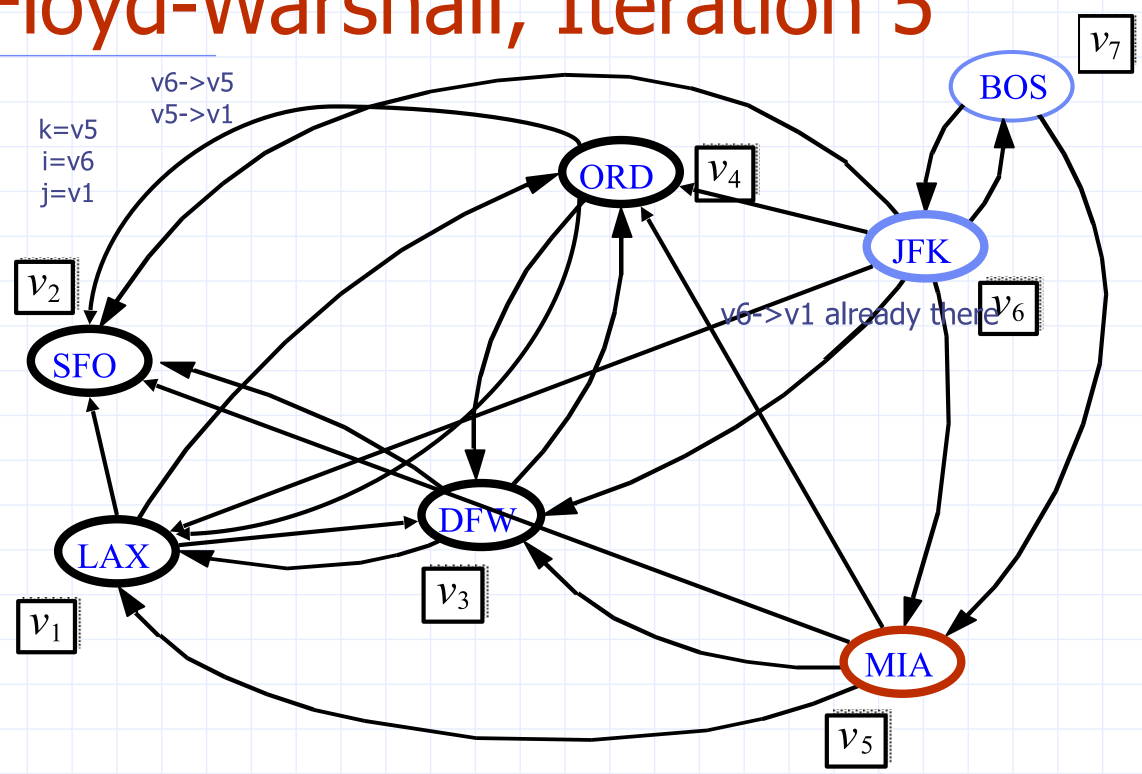
Floyd-Warshall, Iteration 5



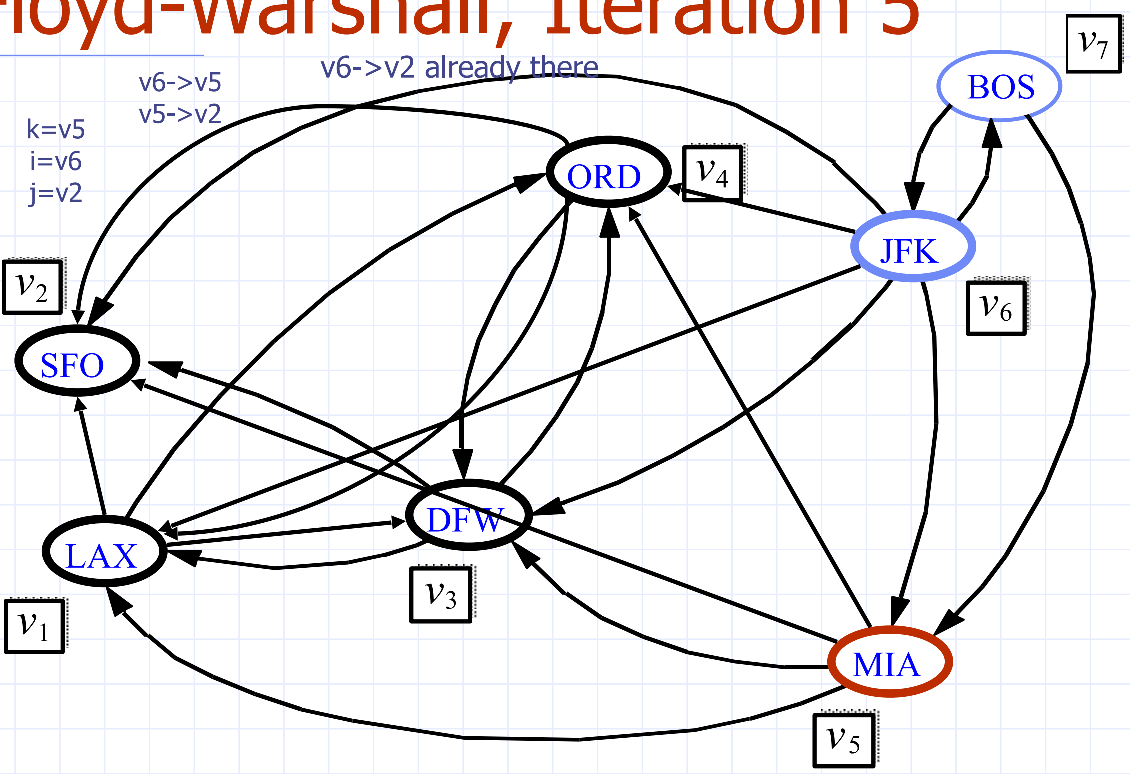
Floyd-Warshall, Iteration 5



Floyd-Warshall, Iteration 5



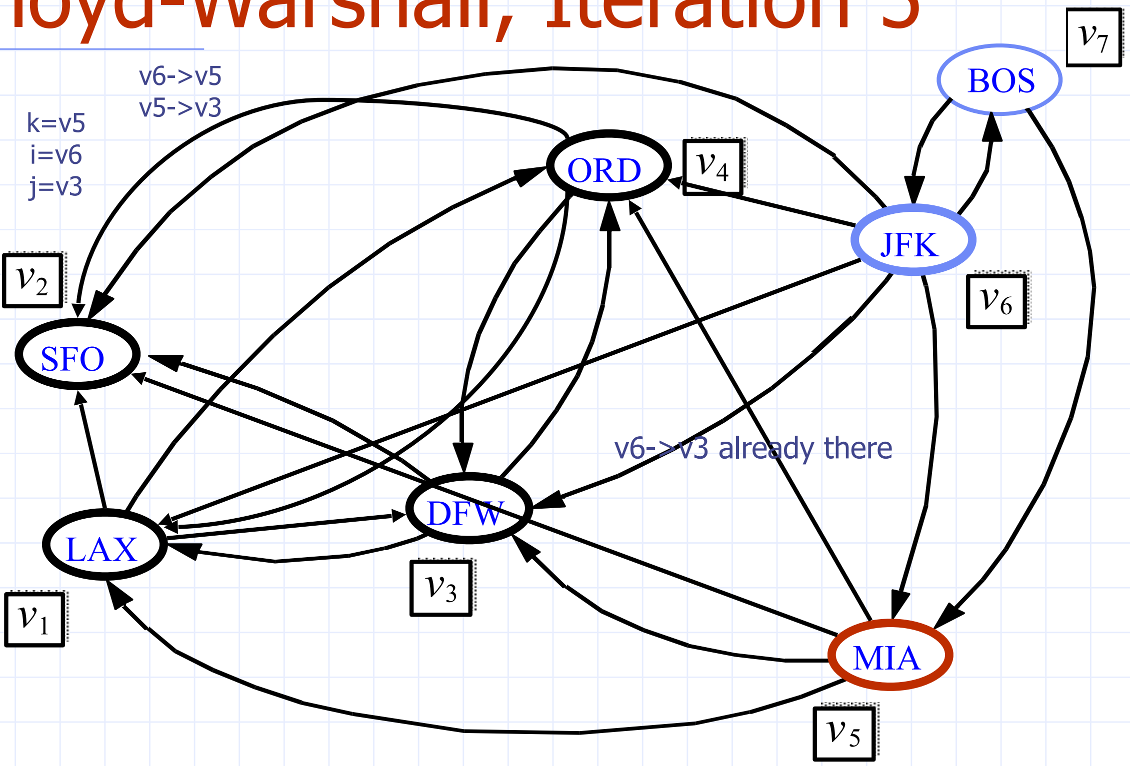
Floyd-Warshall, Iteration 5



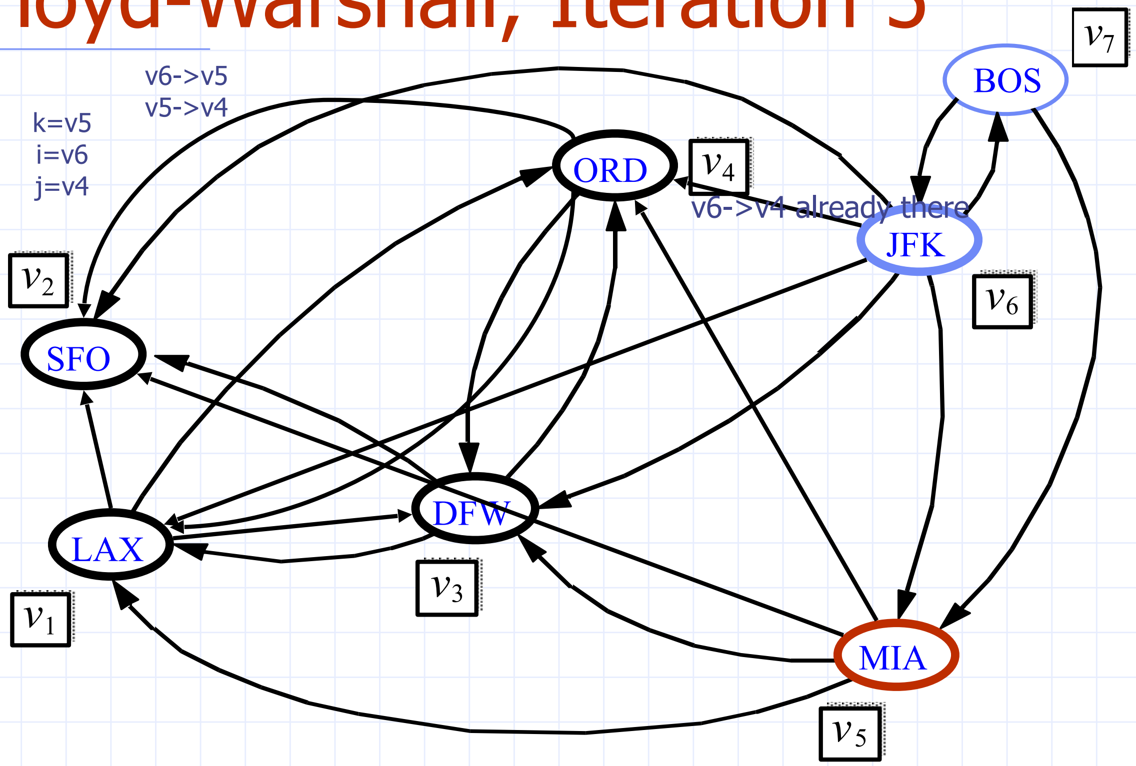
Floyd-Warshall, Iteration 5

k=v5
i=v6
j=v3

v6->v5
v5->v3



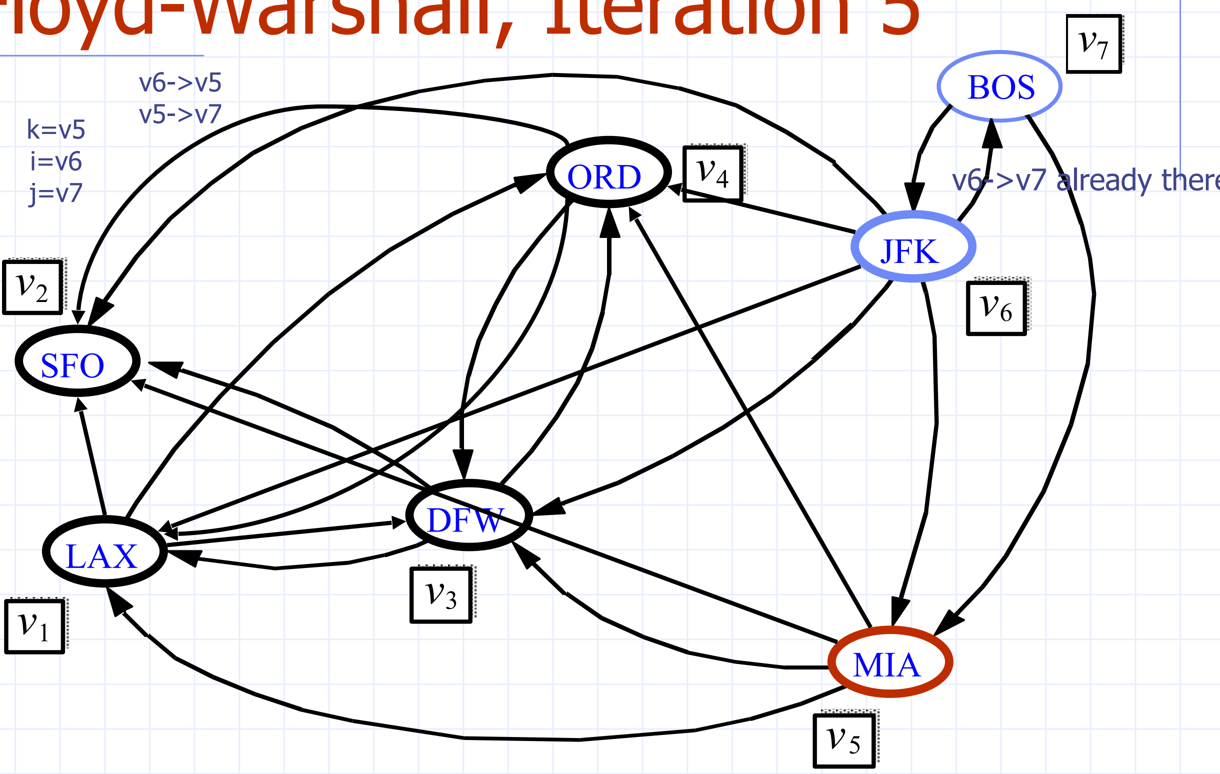
Floyd-Warshall, Iteration 5



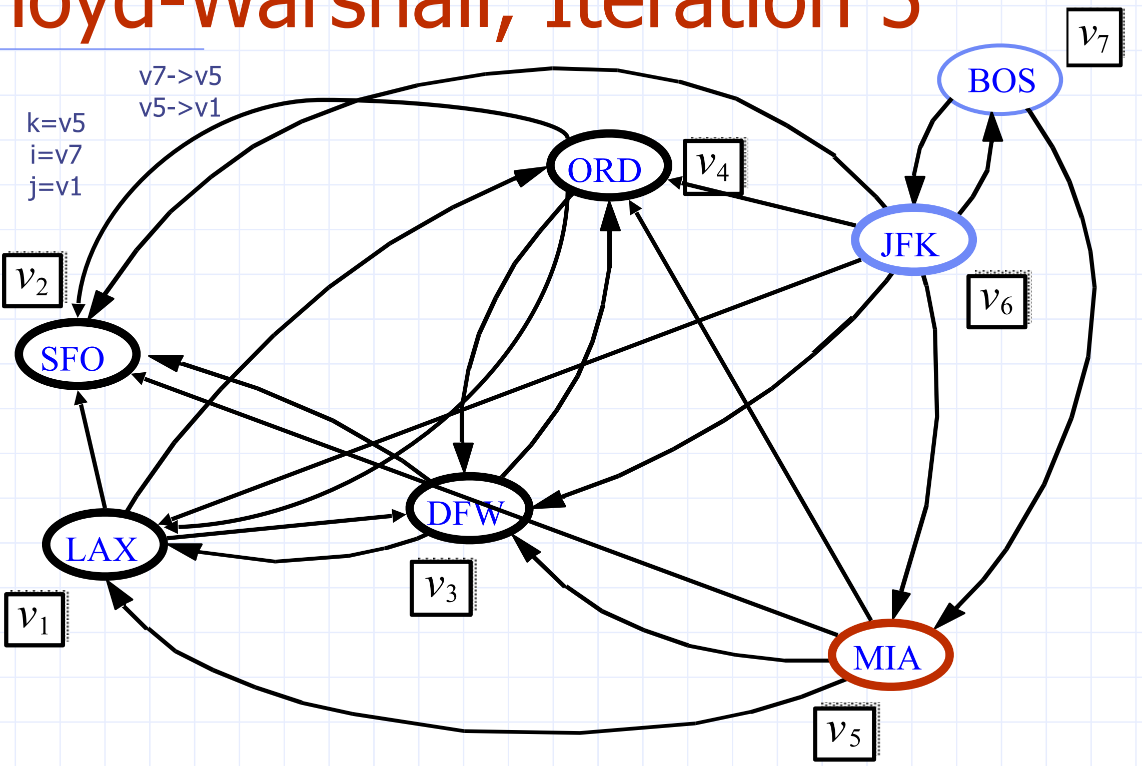
Floyd-Warshall, Iteration 5

k=v5
i=v6
j=v7

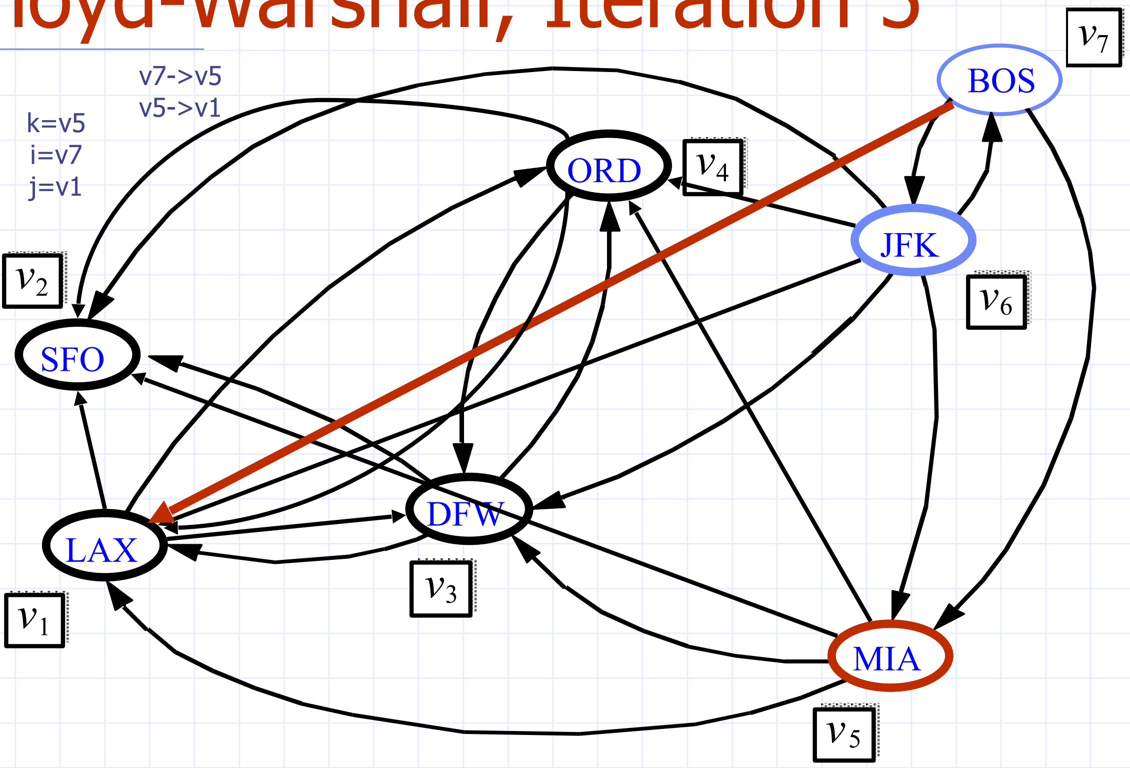
v6->v5
v5->v7



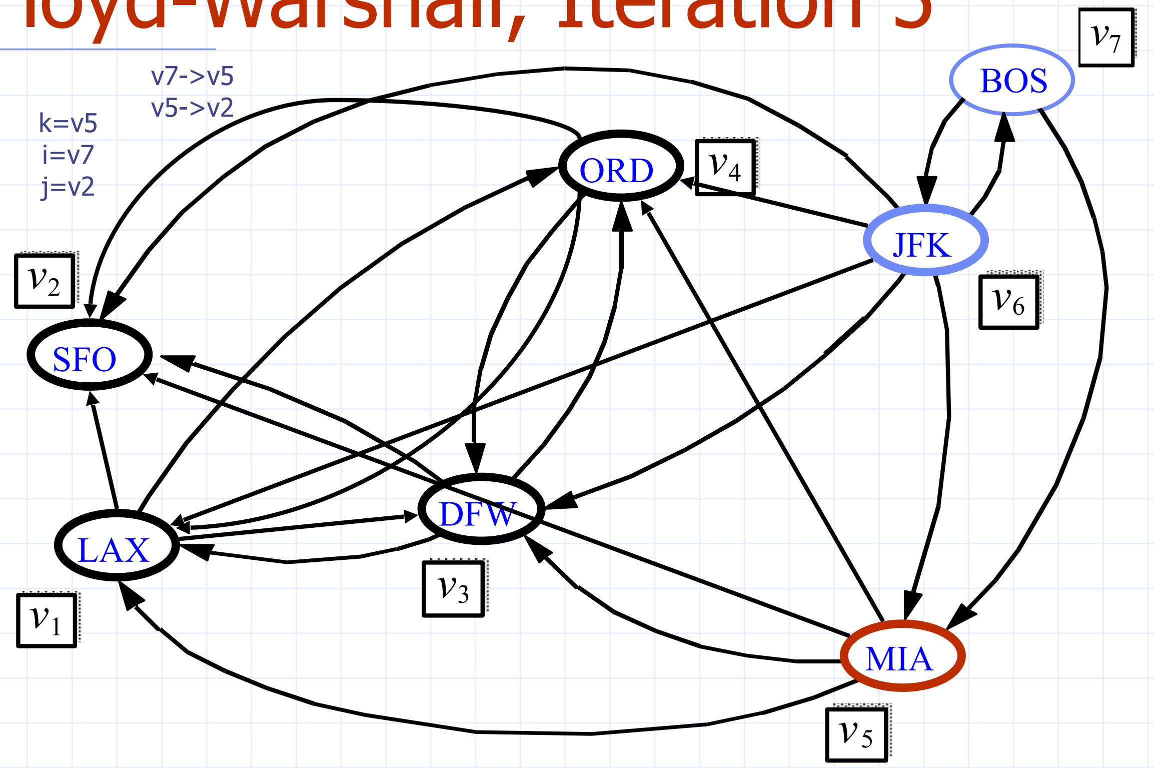
Floyd-Warshall, Iteration 5



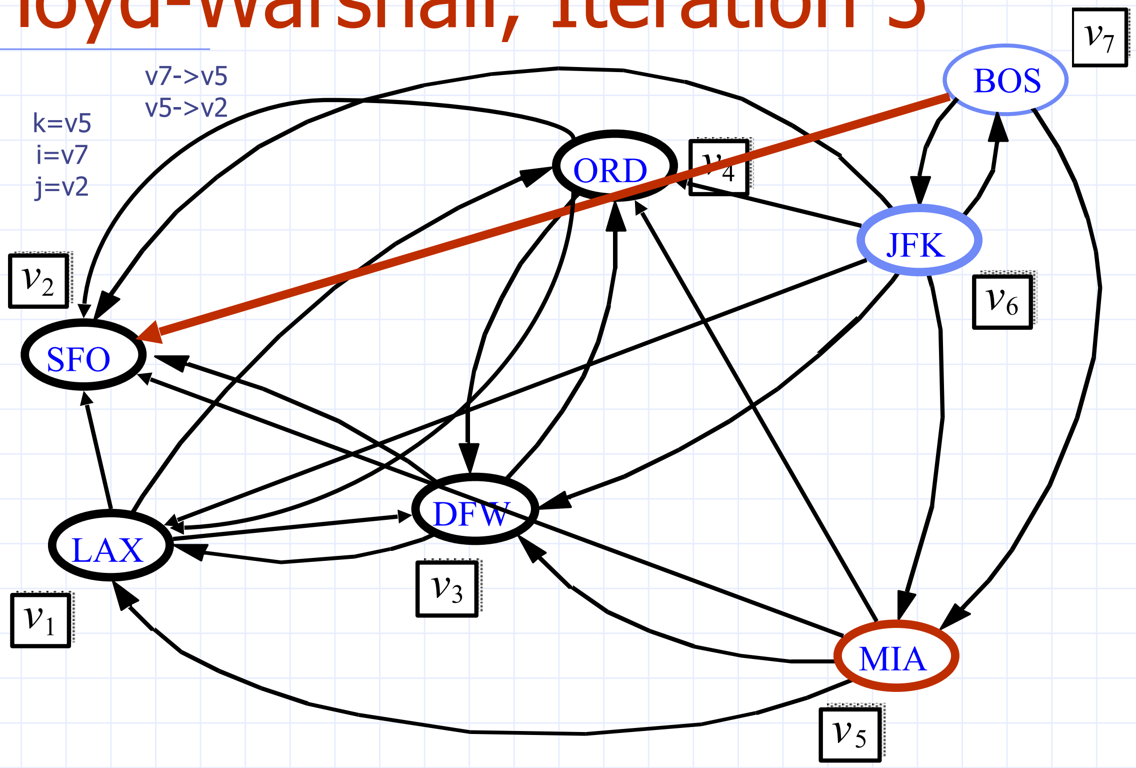
Floyd-Warshall, Iteration 5



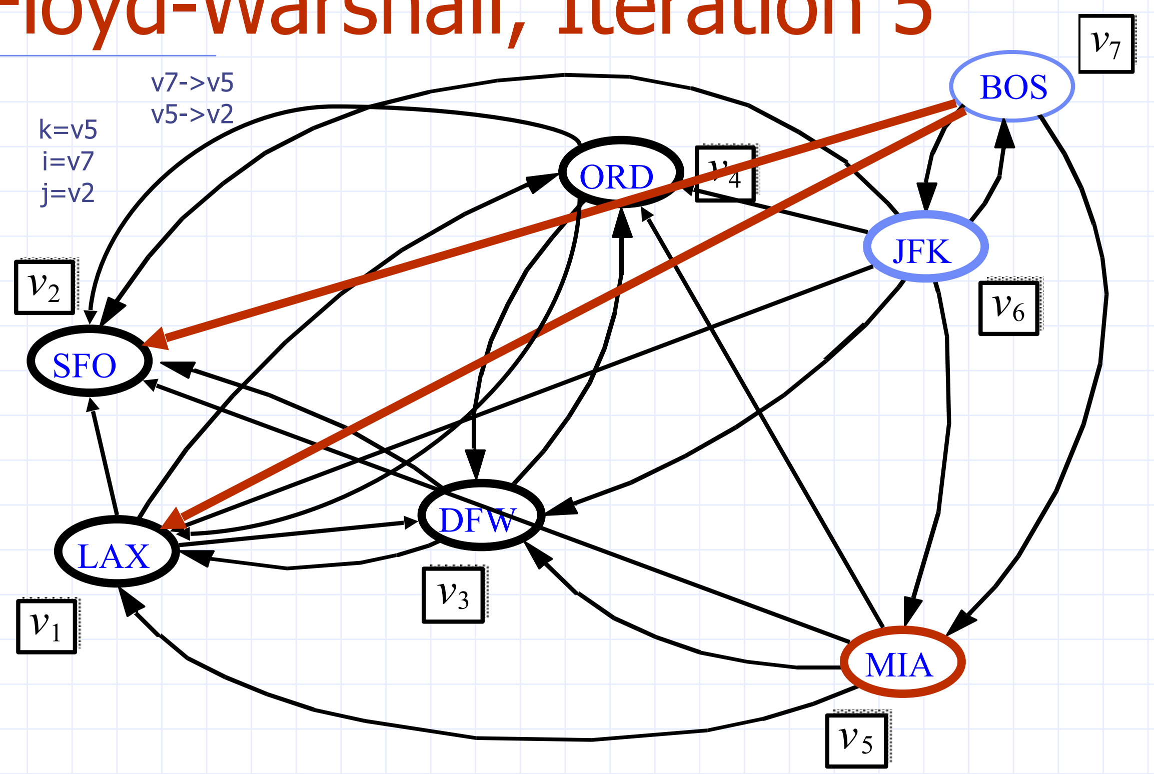
Floyd-Warshall, Iteration 5



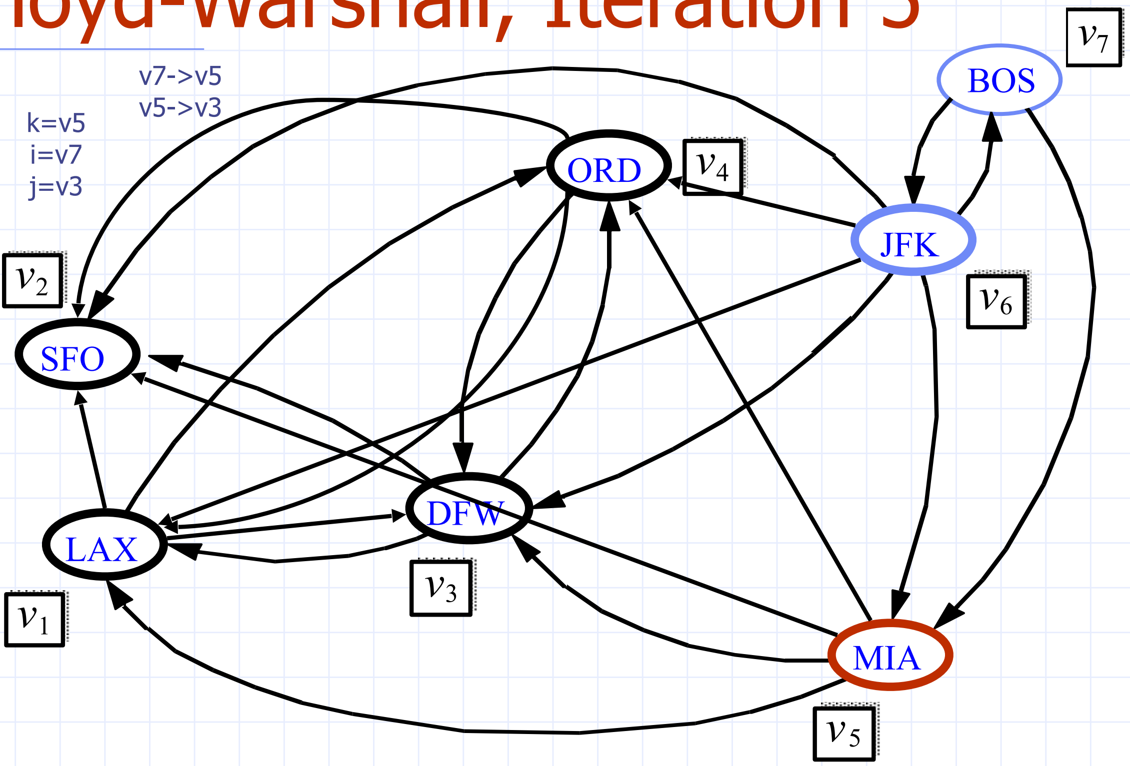
Floyd-Warshall, Iteration 5



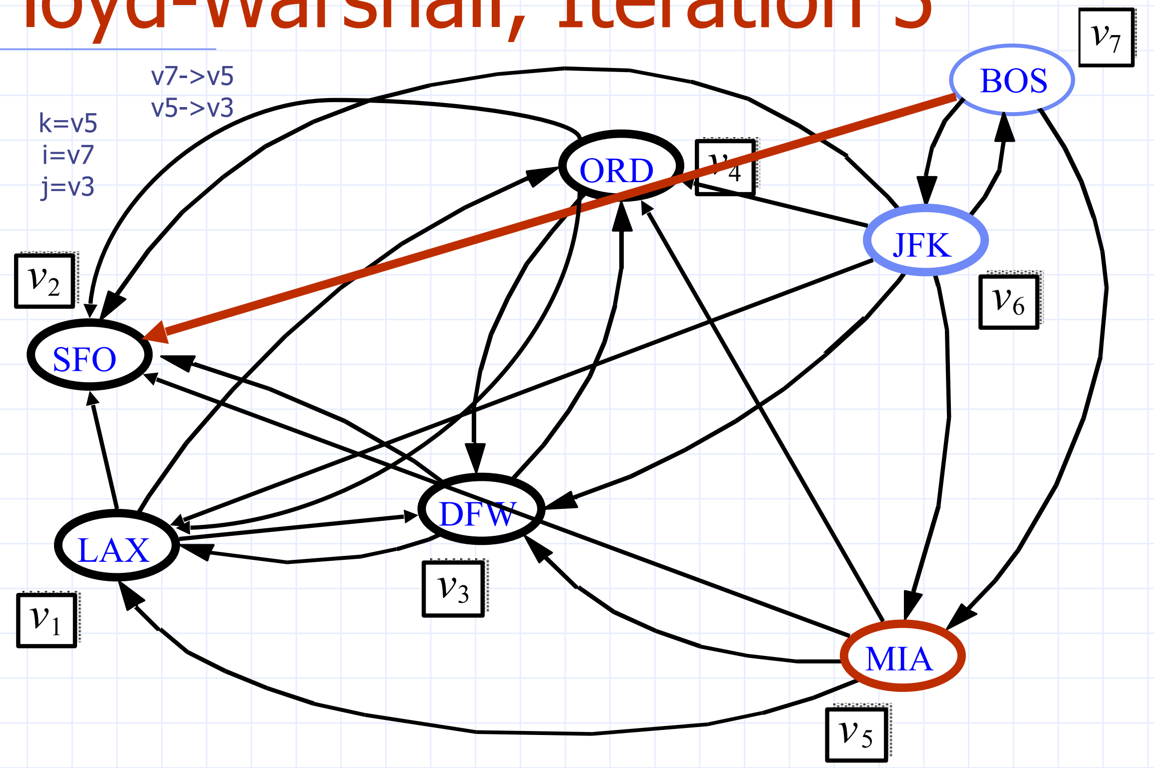
Floyd-Warshall, Iteration 5



Floyd-Warshall, Iteration 5



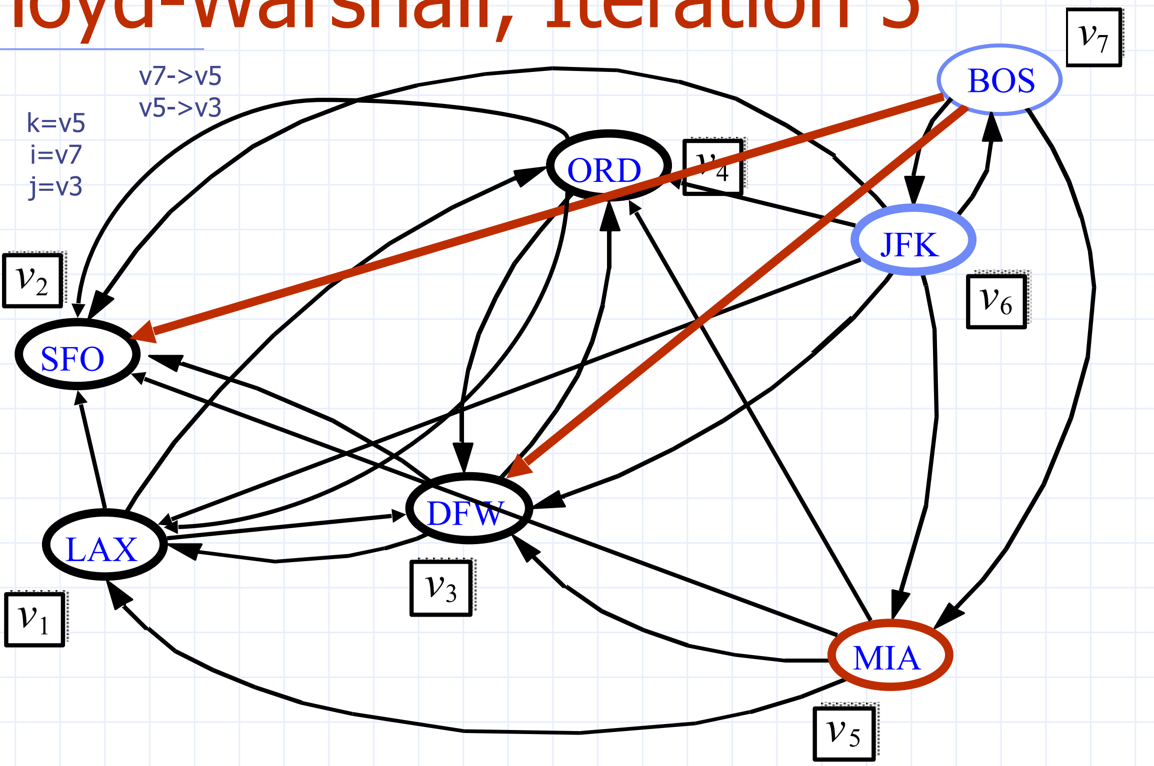
Floyd-Warshall, Iteration 5



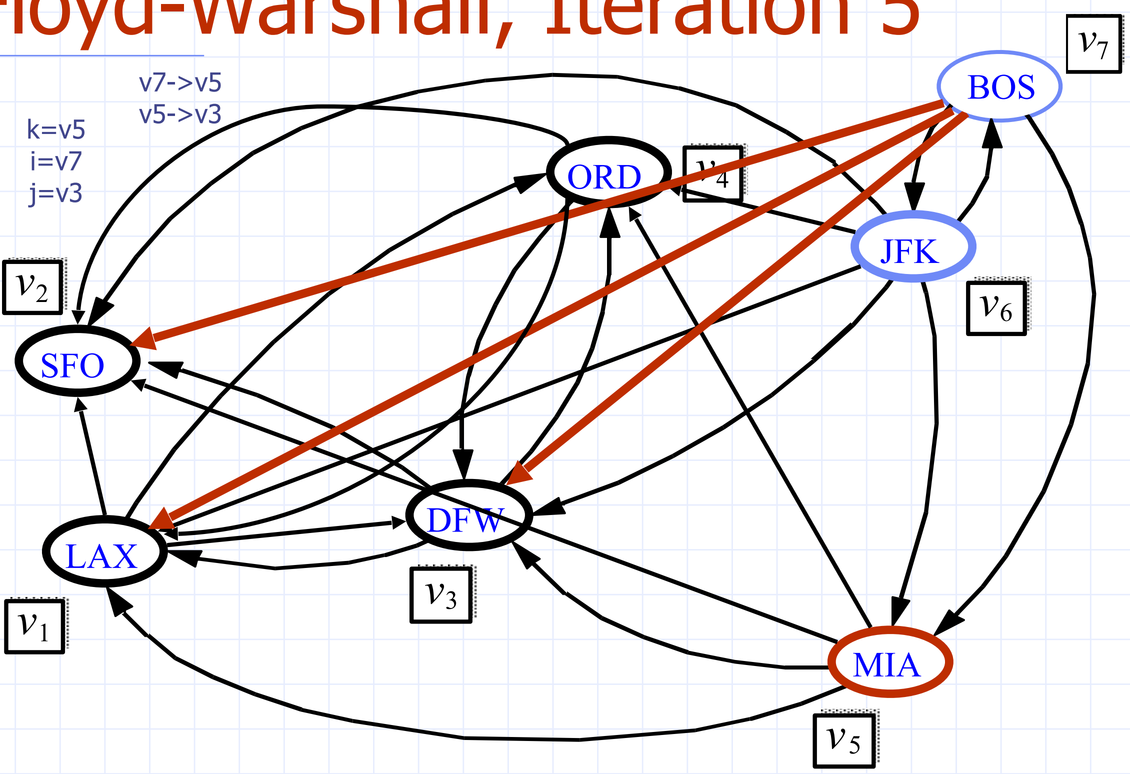
Floyd-Warshall, Iteration 5

k=v5
i=v7
j=v3

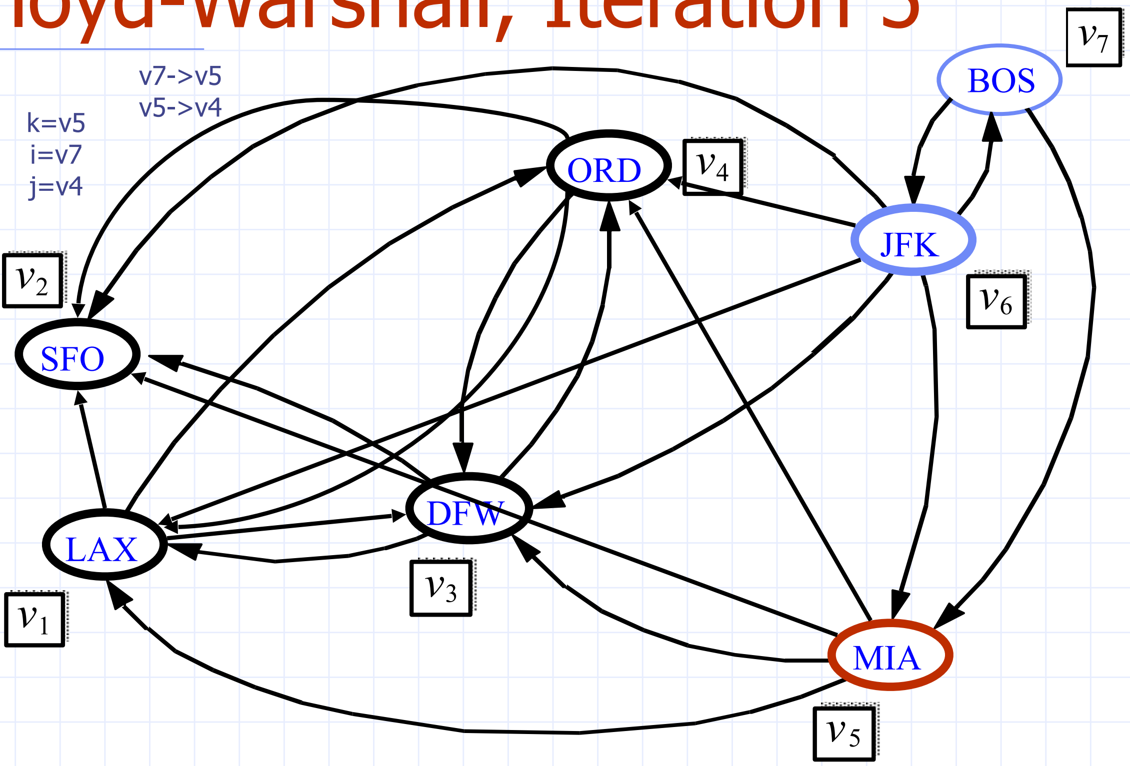
v7->v5
v5->v3



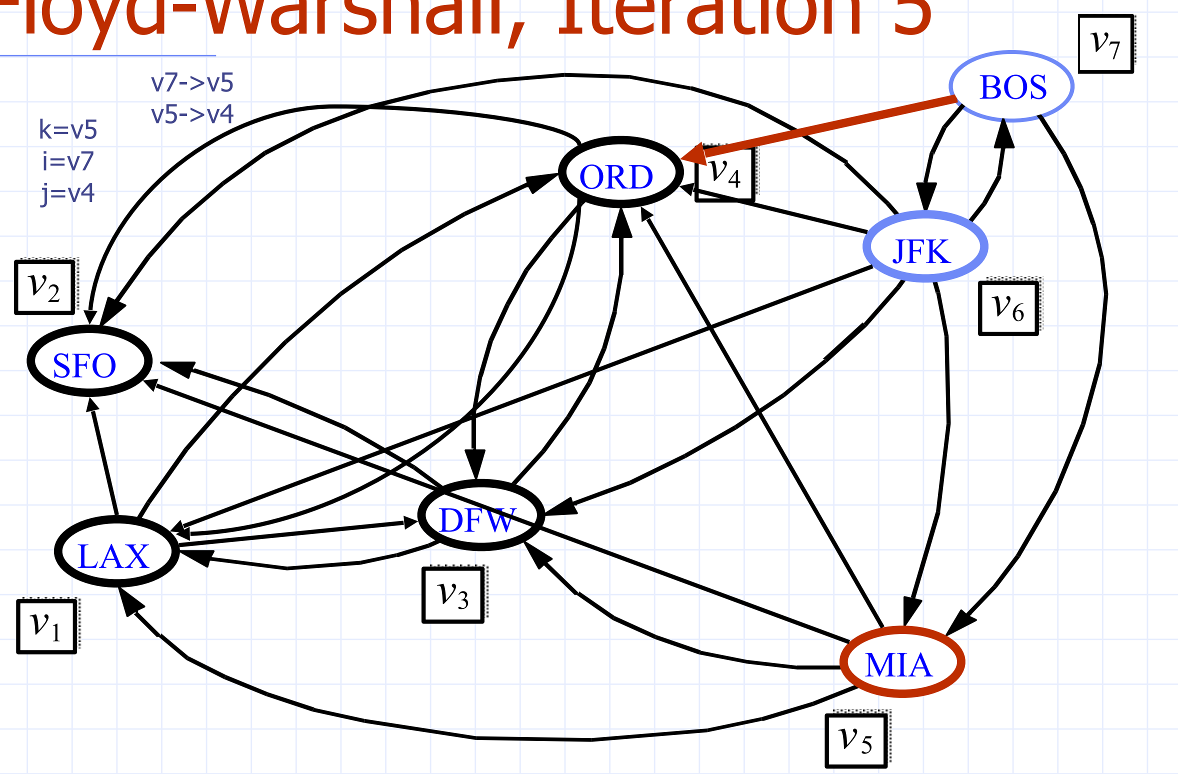
Floyd-Warshall, Iteration 5



Floyd-Warshall, Iteration 5



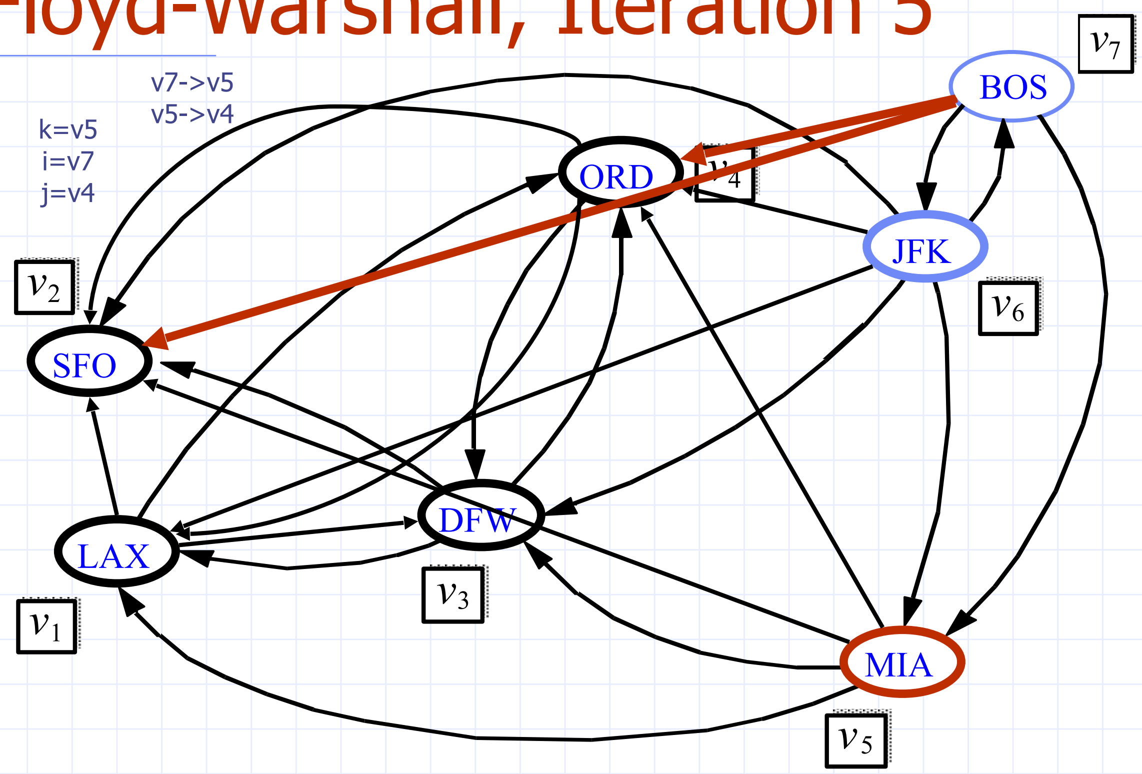
Floyd-Warshall, Iteration 5



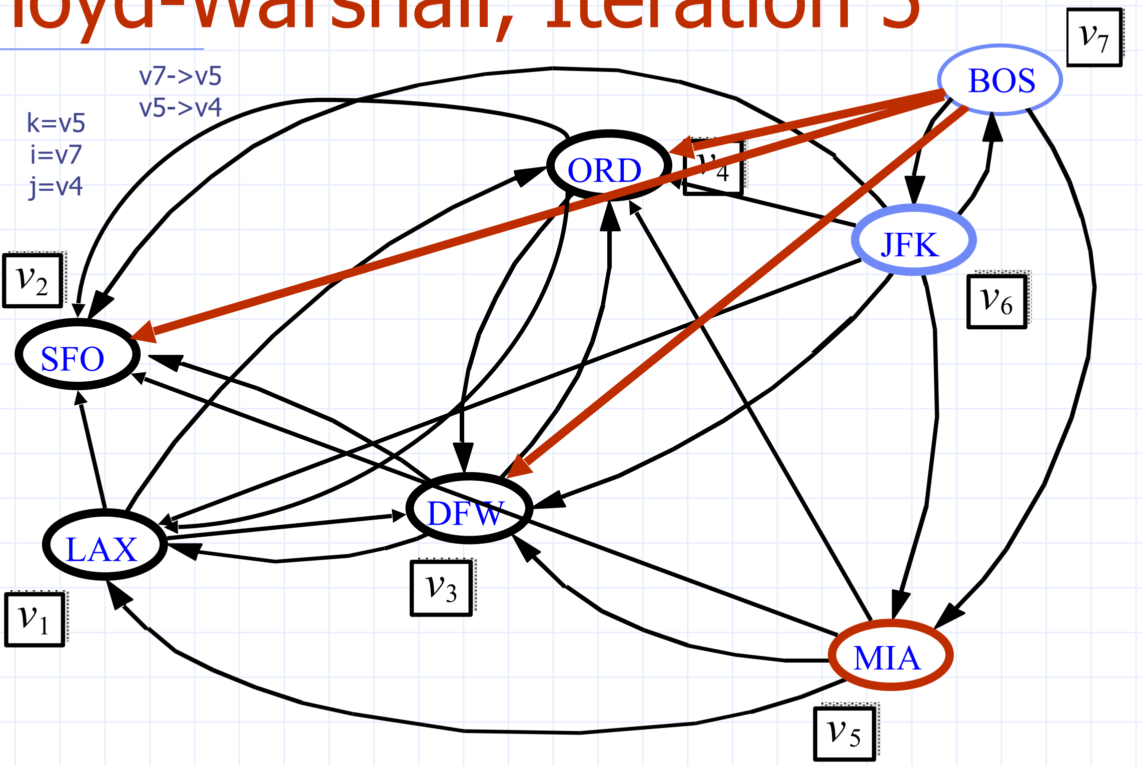
Floyd-Warshall, Iteration 5

k=v5
i=v7
j=v4

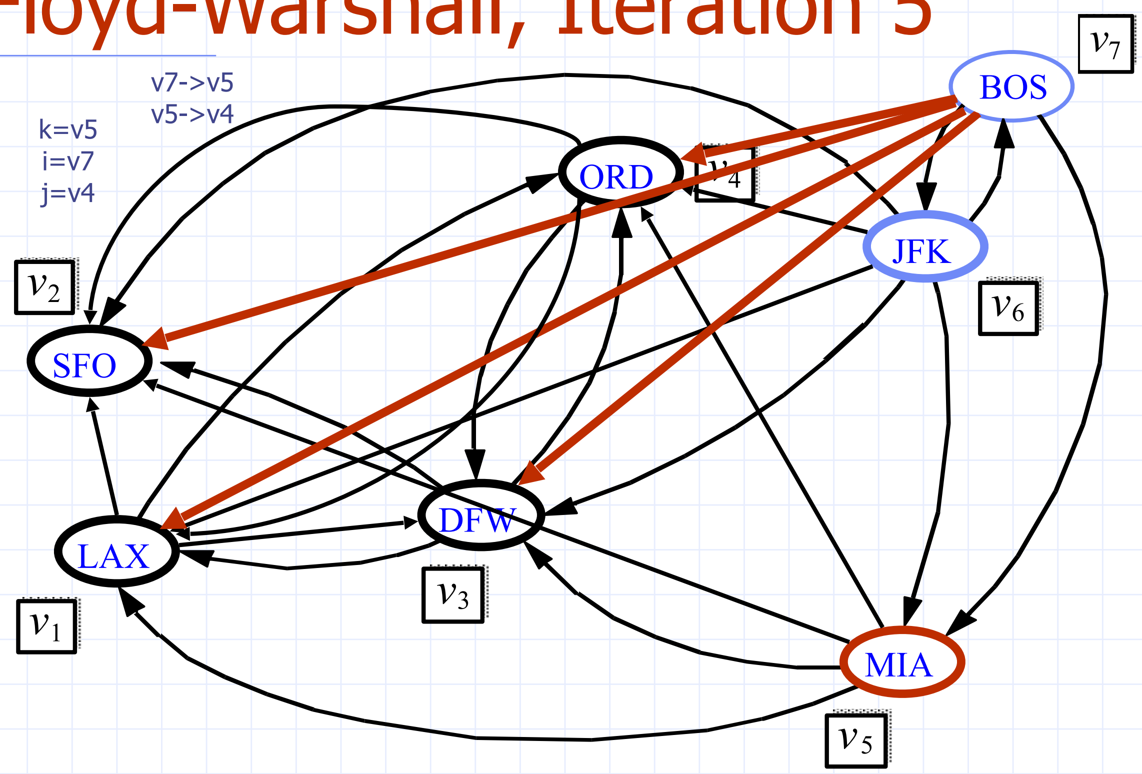
v7->v5
v5->v4



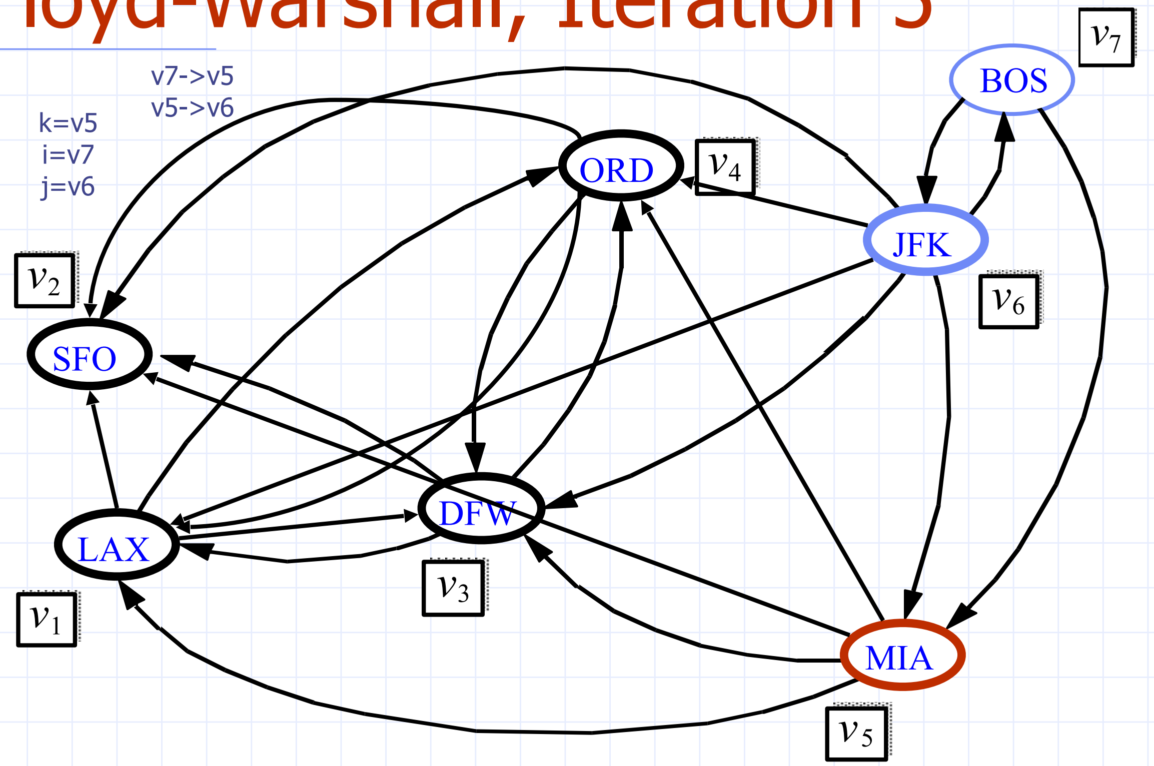
Floyd-Warshall, Iteration 5



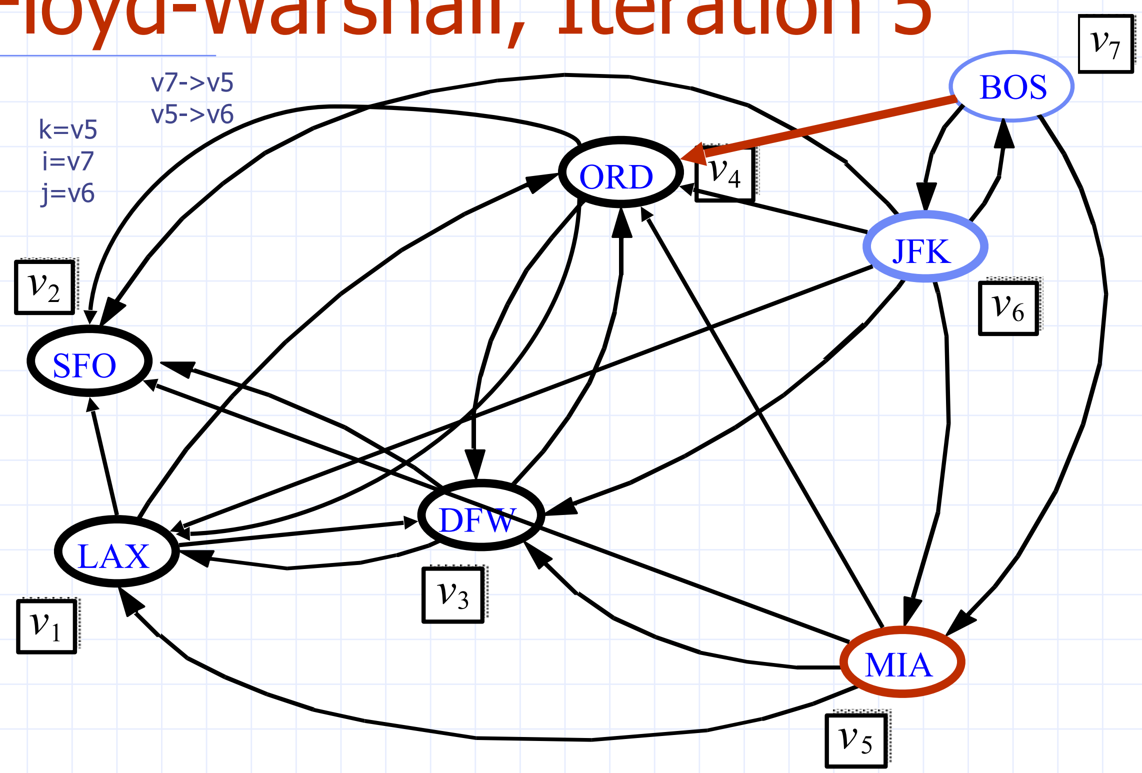
Floyd-Warshall, Iteration 5



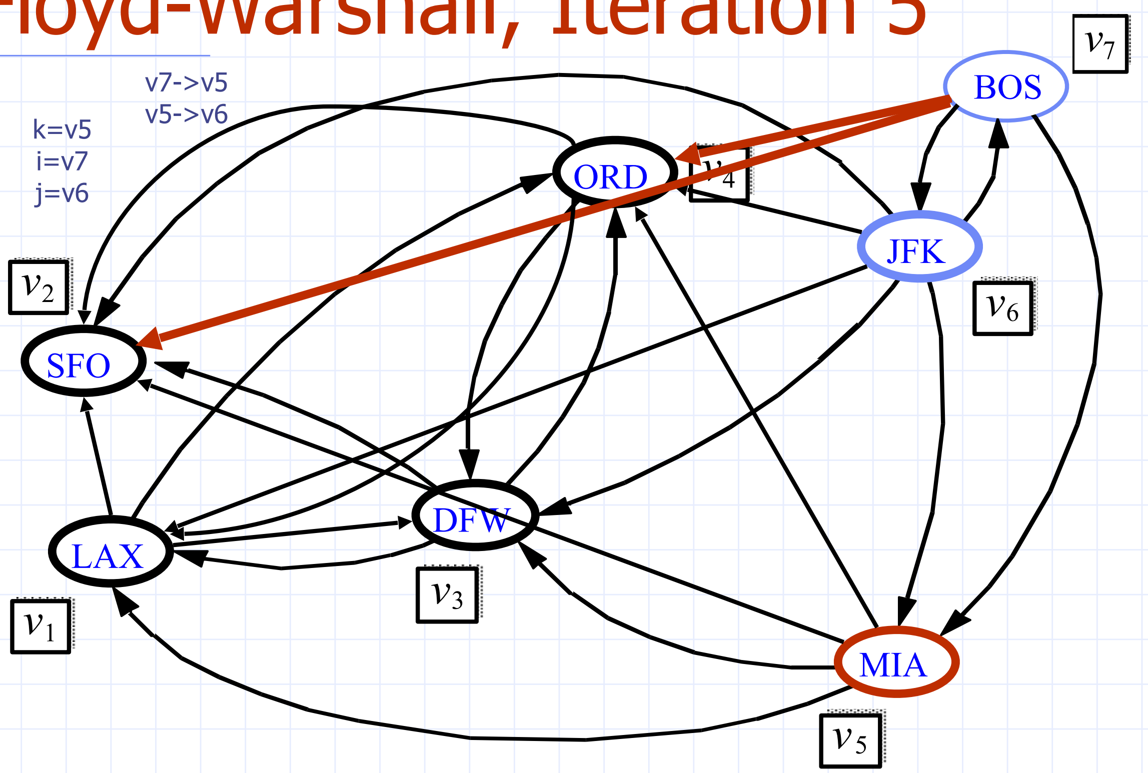
Floyd-Warshall, Iteration 5



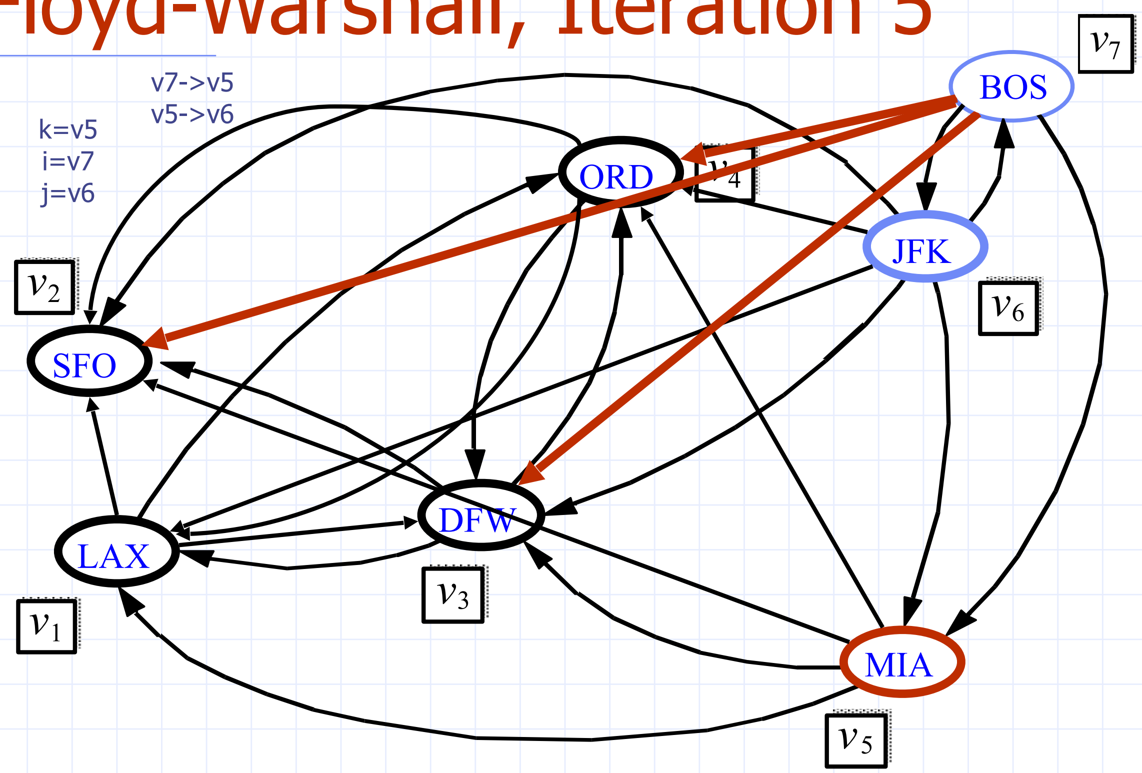
Floyd-Warshall, Iteration 5



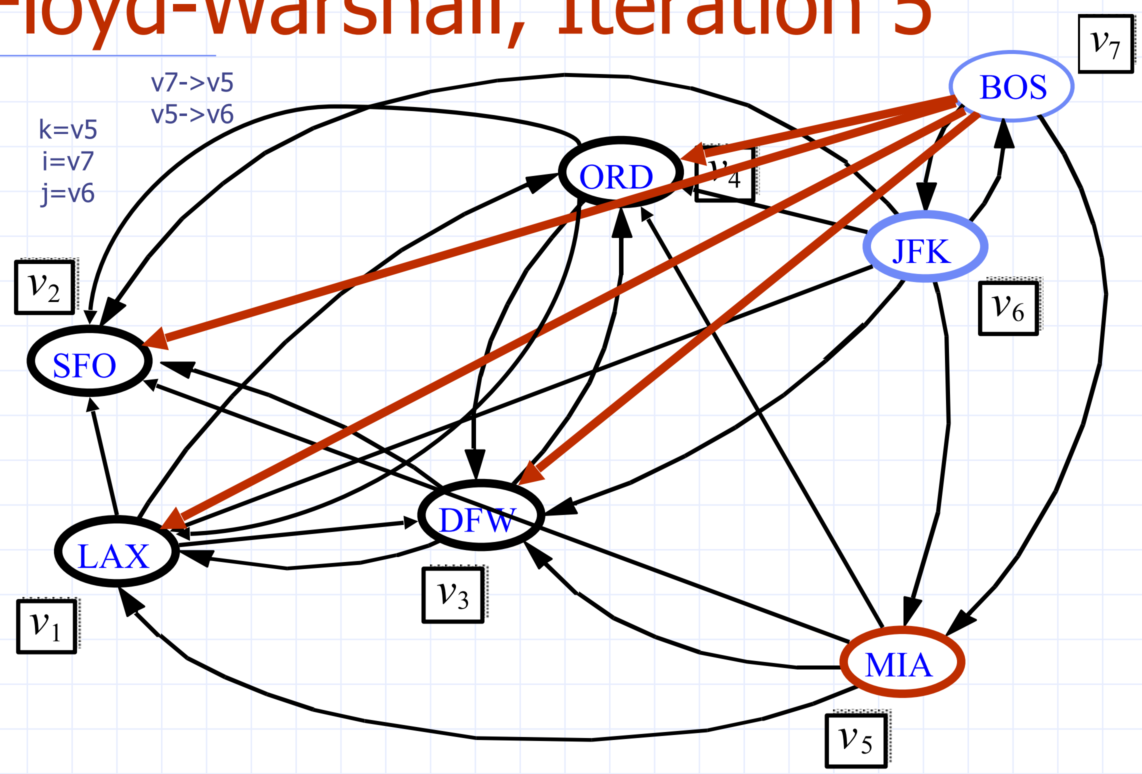
Floyd-Warshall, Iteration 5



Floyd-Warshall, Iteration 5



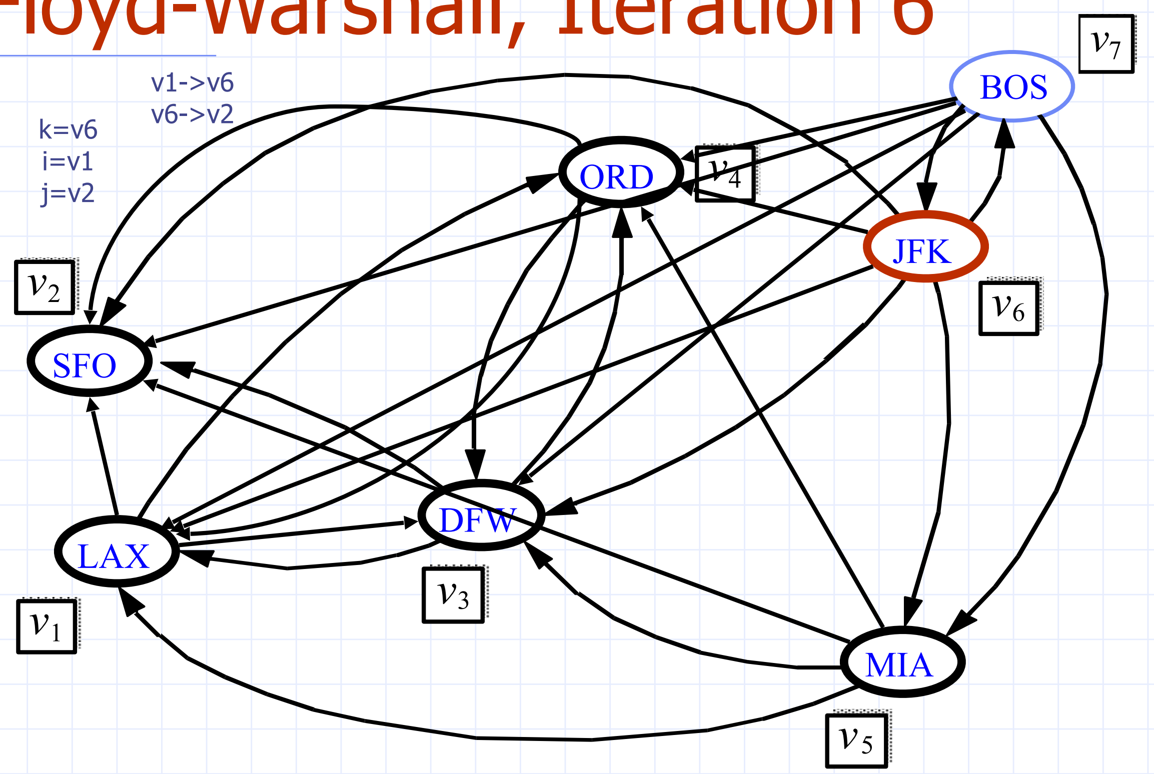
Floyd-Warshall, Iteration 5



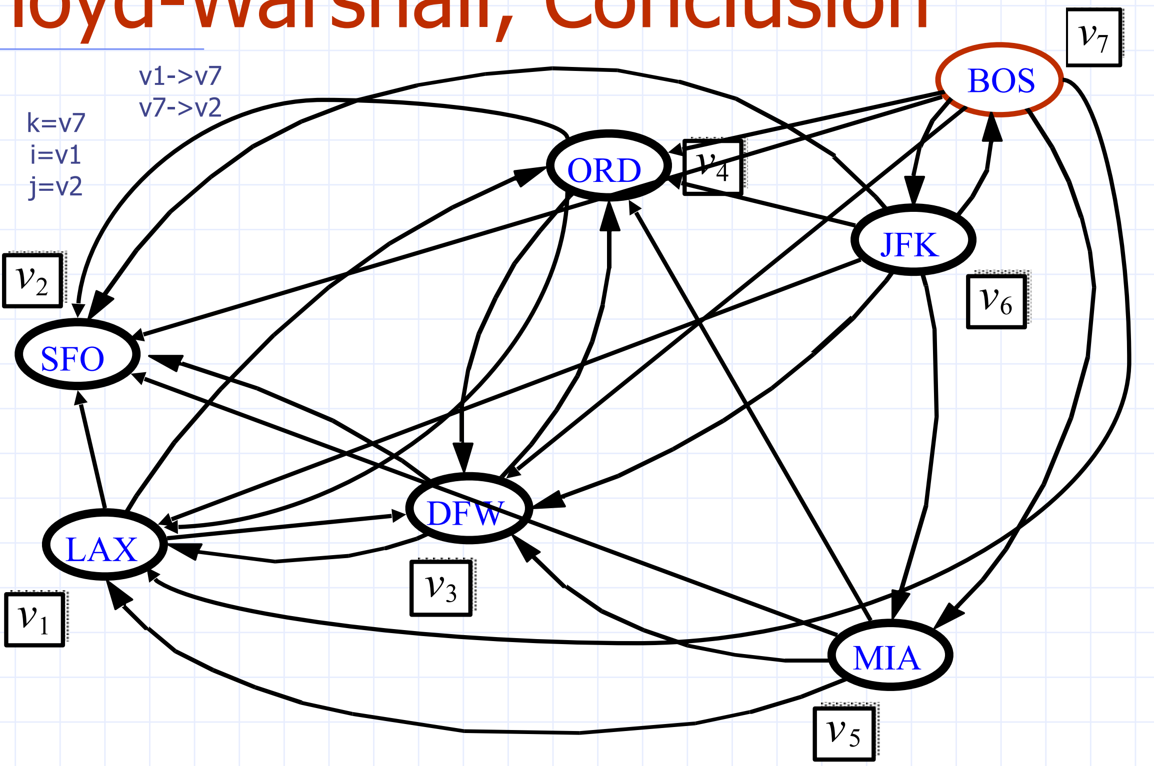
Floyd-Warshall, Iteration 6

k=v6
i=v1
j=v2

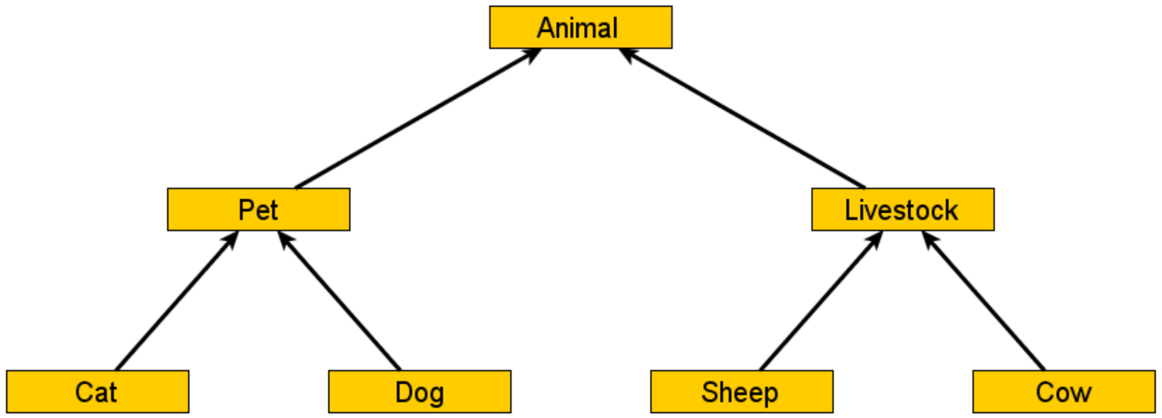
v1->v6
v6->v2



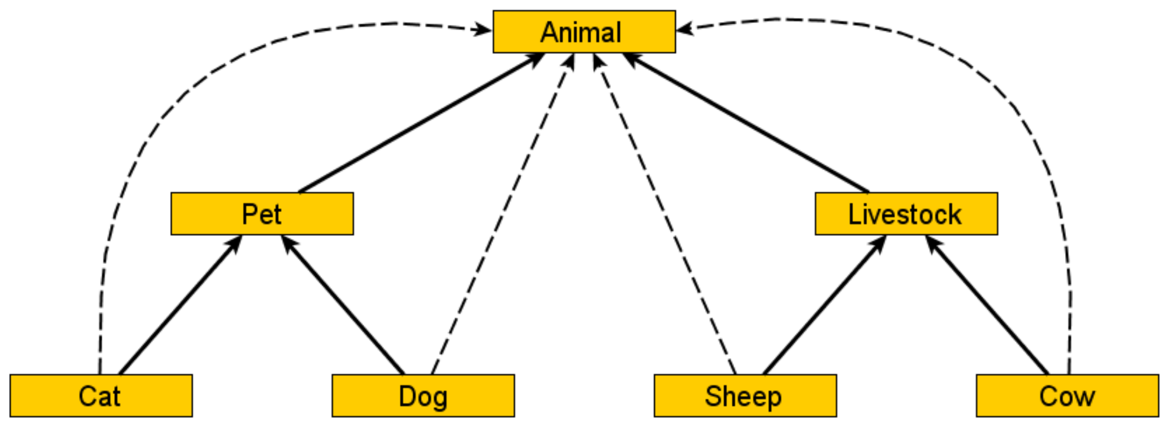
Floyd-Warshall, Conclusion



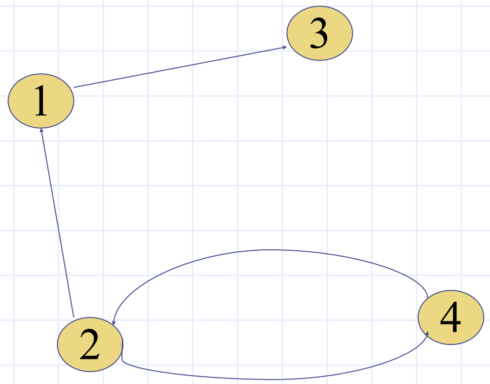
- Transitive closure



- Transitive closure



- Transitive closure



- Transitive closure