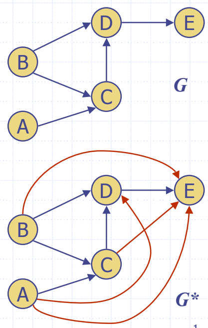
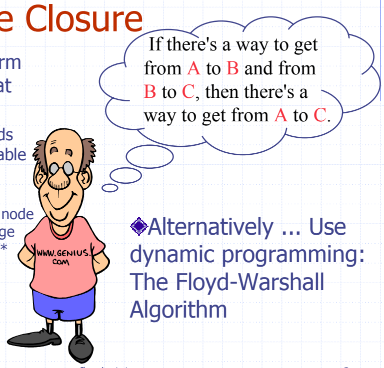
Transitive Closure

- 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



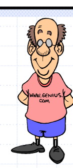
Computing the Transitive Closure

- We can perform DFS starting at each vertex
  - DFS(G,v) finds nodes reachable from v
    - for each reachable node w, add edge (v,w) to G*
  - O(n(n+m))

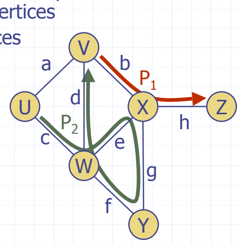


- Alternatively ... Use dynamic programming: The Floyd-Warshall Algorithm

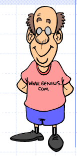
Floyd-Warshall Transitive Closure



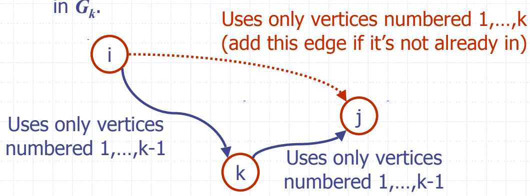
- Idea #1: Number the vertices 1, 2, …, n. Call them v_1, v_2, … v_n.
- Idea #2: Consider paths that use only vertices v_1, v_2, …, v_k, as intermediate vertices
- On path P_2, intermediate vertices are X,W, and Y.
- Subproblem definition: G_k is a graph where
  - directed edge (v_i, v_j) if G has a directed path from v_i to v_j with intermediate vertices in the set {v_1, v_2, …, v_k}



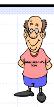
Floyd-Warshall Transitive Closure



- Constructing G_k from G_{k-1} :
- For each pair of vertices (v_i,v_j) in G_{k-1}
  - If (v_i,v_j) is in G_{k-1}, then it is also in G_k
  - If (v_i,v_k) and (v_k,v_j) are in G_{k-1}, then (v_i,v_j) is in G_k.



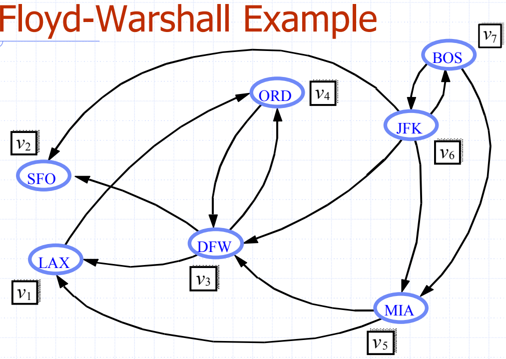
Floyd-Warshall's Algorithm



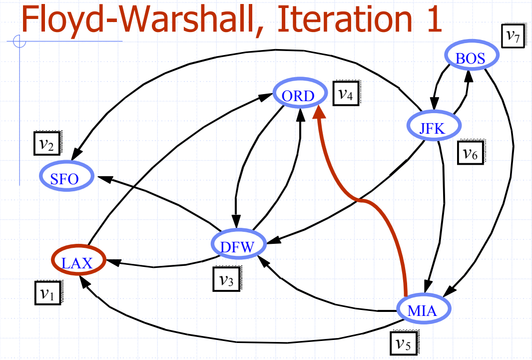
- Floyd-Warshall's algorithm numbers the vertices of G as $v_1, \ldots, v_n$ and computes a series of digraphs $G_0, \ldots, G_n$
  - $G_0 = G$
  - G_k has a directed edge (v_i, v_j) if G has a directed path from v_i to v_j with intermediate vertices in the set $\{v_1, \ldots, v_k\}$
- We have that $G_n = G^*$
- 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 ← 1
  for all v ∈ G.vertices()
    denote v as v_i
    i ← i + 1
  G_0 ← G
  for k ← 1 to n do
    G_k ← G_{k-1}
     for i ← 1 to n (i ≠ k) do
       for j ← 1 to n (j ≠ i, k) do
         if G_{k-1}.areAdjacent(v_i, v_k) ∧
            G_{k-1}.areAdjacent(v_k, v_j)
           if ¬G_k.areAdjacent(v_i, v_j)
              G_k.insertDirectedEdge(v_i, v_j, k)
    return G_n
```

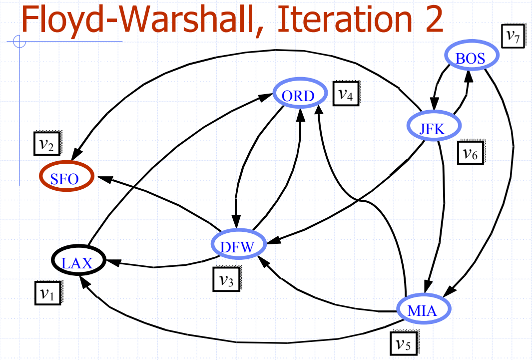

Floyd-Warshall Example



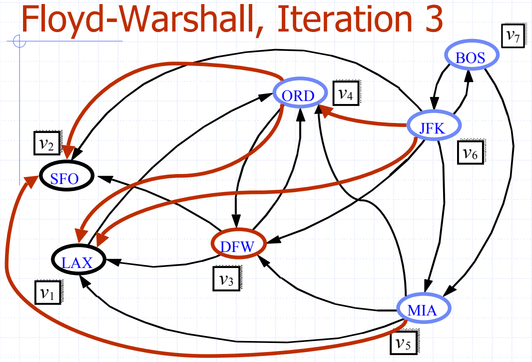
Floyd-Warshall, Iteration 1

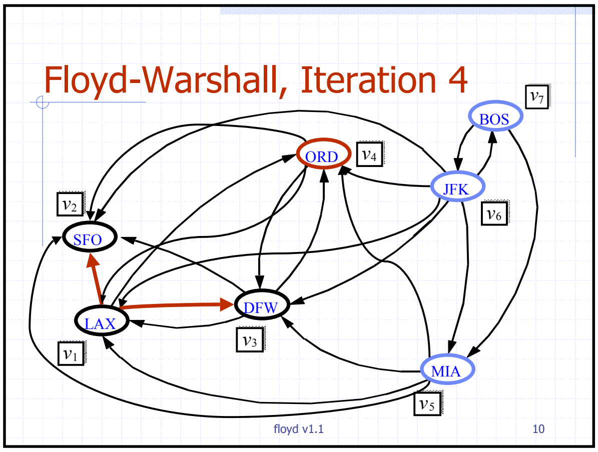


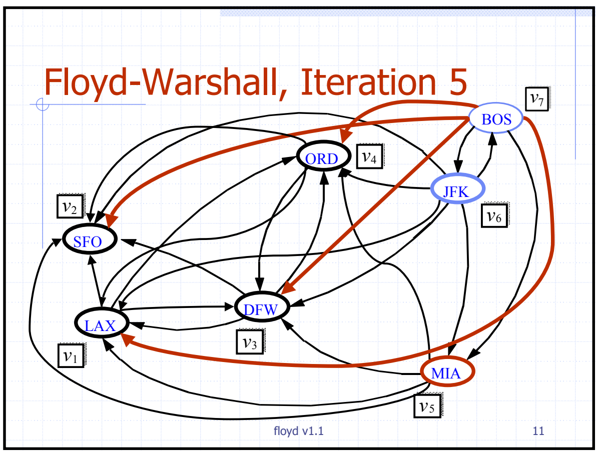
Floyd-Warshall, Iteration 2

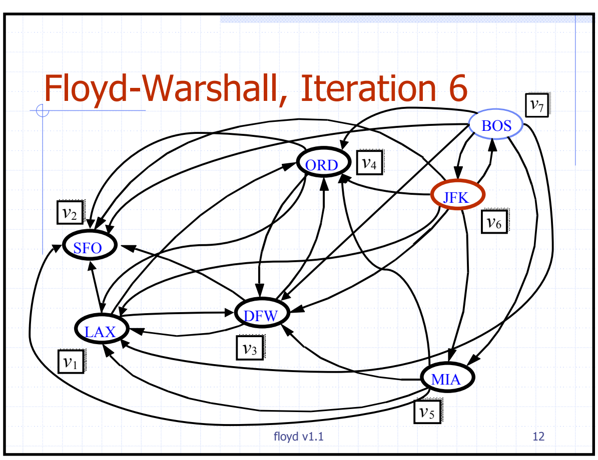


Floyd-Warshall, Iteration 3




Floyd-Warshall, Iteration 4
BOS
v7
ORD
v4
JFK
v6
v2
SFO
DFW
LAX
v3
v1
MIA
v5
floyd v1.1
10


Floyd-Warshall, Iteration 5
BOS
v7
ORD
v4
JFK
v6
v2
SFO
DFW
LAX
v3
v1
MIA
v5
floyd v1.1
11


Floyd-Warshall, Iteration 6
BOS
v7
ORD
v4
JFK
v6
v2
SFO
DFW
LAX
v3
v1
MIA
v5
floyd v1.1
12

Floyd-Warshall, Conclusion