

Directed Graphs

Directed Graphs 1

Outline and Reading (§6.4)

- ◆ Reachability (§6.4.1)
 - Directed DFS
 - Strong connectivity
- ◆ Transitive closure (§6.4.2)
 - The Floyd-Warshall Algorithm
- ◆ Directed Acyclic Graphs (DAG's) (§6.4.4)
 - Topological Sorting

Directed Graphs 2

Digraphs

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

Directed Graphs 3

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, $m \leq n(n-1)$.
- ◆ If we keep in-edges and out-edges in separate adjacency lists, we can perform listing of the sets of in-edges and out-edges in time proportional to their size.

Directed Graphs 4

Digraph Application

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


Directed Graphs 5

Directed DFS

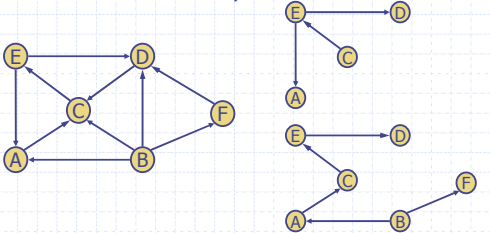
- ◆ We can specialize the traversal algorithms (DFS and BFS) to digraphs by traversing edges only along their direction
- ◆ In the directed DFS algorithm, we have four types of edges
 - discovery edges
 - back edges
 - forward edges
 - cross edges
- ◆ A directed DFS starting at a vertex s determines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

Directed Graphs 6

Reachability




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

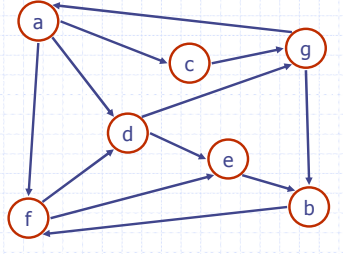


Directed Graphs 7

Strong Connectivity




- ◆ Each vertex can reach all other vertices



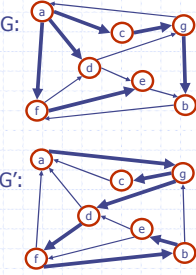
Directed Graphs 8

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: $O(n+m)$.

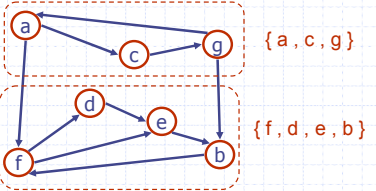


Directed Graphs 9

Strongly Connected Components



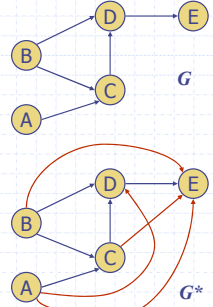
- ◆ Maximal subgraphs such that each vertex can reach all other vertices in the subgraph
- ◆ Can also be done in $O(n+m)$ time using DFS, but is more complicated (similar to biconnectivity).



Directed Graphs 10

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




Directed Graphs 11

Computing the Transitive Closure

- ◆ We can perform DFS starting at each vertex
 - $O(n(n+m))$
- ◆ Alternatively ... Use dynamic programming: the Floyd-Warshall Algorithm

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.



Directed Graphs 12

Floyd-Warshall Transitive Closure

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

Uses only vertices numbered $1, \dots, k-1$

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

Uses only vertices numbered $1, \dots, k-1$

Directed Graphs 13

Floyd-Warshall's Algorithm

Floyd-Warshall's algorithm numbers the vertices of G as v_1, \dots, v_n and computes a series of digraphs G_0, \dots, 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 $\{v_1, \dots, v_k\}$

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)
            then G_k.insertDirectedEdge(v_i, v_j, k)
return G_n
    
```

Directed Graphs 14

Floyd-Warshall Example

Directed Graphs 15

Floyd-Warshall, Iteration 1

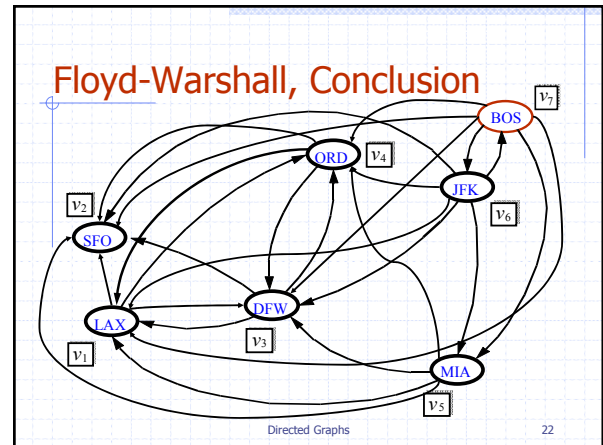
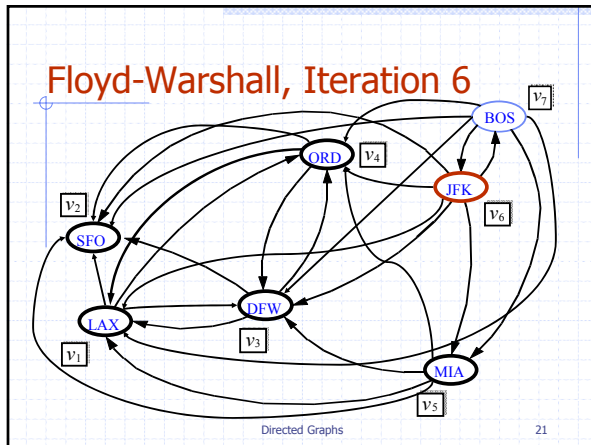
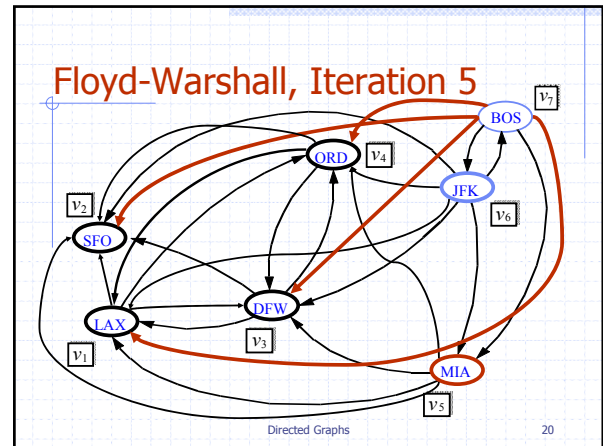
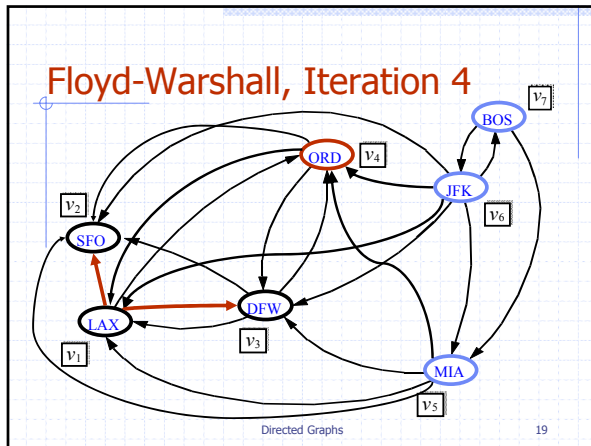
Directed Graphs 16

Floyd-Warshall, Iteration 2

Directed Graphs 17

Floyd-Warshall, Iteration 3

Directed Graphs 18



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, \dots, 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 is a task sequence that satisfies the precedence constraints

Theorem
A digraph admits a topological ordering if and only if it is a DAG

Directed Graphs 23

Topological Sorting

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

Directed Graphs 24

Algorithm for Topological Sorting

Note: This algorithm is different than the one in Goodrich-Tamassia

```

Method TopologicalSort(G)
  H ← G // Temporary copy of 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
    
```

Running time: $O(n + m)$. How...?

Directed Graphs 25

Topological Sorting Algorithm using DFS

Simulate the algorithm by using depth-first search

```

Algorithm topologicalDFS(G)
  Input dag G
  Output topological ordering of 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)
    
```

```

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.incidentEdges(v)
    if getLabel(e) = UNEXPLORED
      w ← opposite(v,e)
      if getLabel(w) = UNEXPLORED
        setLabel(e, DISCOVERY)
        topologicalDFS(G, w)
      else
        {e is a forward or cross edge}
  Label v with topological number n
  n ← n - 1
    
```

$O(n+m)$ time.

Directed Graphs 26

