

# For Monday

- Read 9.5-9.6
- Homework:
  - Chapter 9, exercises 5, 7a, 11

# Program 4

# Research Paper

- Any questions?

# Weighted Graphs

- Dijkstra's algorithm
- A **greedy** algorithm
- A form of best-first search

# Negative Weights Problem

# Network Flow

- We have a weighted directed graph with a **source** and a **sink**--a **network**.
- Note that cycles are possible.
- Each edge's weight represent the maximum possible **flow** through that edge.
- We want to determine the maximum possible flow through the network and a way to achieve that maximum.

# Ford-Fulkerson Method

- Start with 0 flow in the network
- Repeat
  - Select a path from source to sink with no full **forward** edges or a non-full **backward** edge.
  - Increase flow by maximum possible amount on that path
- Until no paths can be selected
- You now have maximal flow

# Selecting a Path to Augment

- If you select the longest path, problems can arise.
- If the shortest available path is used at every iteration: the number of paths used before maximum flow is found is less than  $VE$
- Can be improved by using the path with the best improvement to flow at each iteration.

# Critical Path

- The longest path between two nodes
- For what kind of graph is this meaningful?
- Why would we want to compute it?
- How could we compute it?

# Transitive Closure

- The problem is: for each node in the graph, what other nodes can be reached from that node?
- Not an issue in an undirected graph. Why?
- Can be done by doing a search from each node and discovering all of the nodes that can be found from each node.

# Transitive Closure (cont.)

- Depth First Search can be used to compute the transitive closure of a graph represented with an adjacency list in  $O(V(V+E))$  time.
- DFS can be used to compute the transitive closure of a graph represented with an adjacency matrix in  $O(V^3)$  time.

# Warshall's Algorithm

- Note that if there is a path from  $X$  to  $Y$  and there is a path from  $Y$  to  $Z$ , then there is a path from  $X$  to  $Z$
- for ( $y = 0; y < V; y++$ )  
  for ( $x = 0; x < V; x++$ )  
    if ( $a[x][y]$ )  
      for ( $j = 0; j < V; j++$ )  
        if ( $a[y][j]$ )  
           $a[x][j] = 1;$

# All Shortest Paths

- In sparse matrix, just run Dykstra's algorithm for each vertex
- In dense graphs, use a matrix and use an algorithm similar to Warshall's algorithm
- Computes all shortest paths in  $O(V^3)$  time
- To determine actual path, need an additional matrix.

# Floyd's Algorithm

- for ( $y = 0; y < V; y++$ )  
  for ( $x = 0; x < V; x++$ )  
    if ( $a[x][y]$ )  
      for ( $j = 0; j < V; j++$ )  
        if ( $a[y][j] > 0$ )  
          if ( $!a[x][j] \parallel$   
              ( $a[x][y] + a[y][j] < a[x][j]$ ))  
             $a[x][j] = a[x][y] + a[y][j];$