CS W3134: Data Structures in Java

Lecture #23: Graphs III
12/2/04
Janak J Parekh

Administrivia

- HW#6 out
  - Test emails will go up today

Agenda

- Graphs cont'd.

Topological sort

- Come up with a legitimate ordering of processing the nodes
  - Often useful for *partial ordering* problems, such as aforementioned course prerequisites
  - Result: a order where no vertex y comes before a vertex x where x→y
  - There can be multiple correct answers!

Topological sort (II)

- Find a vertex that has no successors, i.e., arrows that point to *it*
  - Look at columns of the adjacency matrix
- Delete that vertex and print it out
- Repeat
- What kinds of graphs doesn't this work for?
  - Cycles – what happens?
  - "Catch-22" in real life
  - In other words, works on generalized trees (multiple roots, etc.) – *DAG*

Topological sort (III)

- Complexity again $O(V+E)/O(V^2)$
- How to find node with no successors?
- How do you delete a node?

Connectivity in directed graphs

- Can't just do an arbitrary BFS or DFS
  - Connectivity *depends* on starting node, i.e., "what can you reach from node X?"
  - Do DFS from every vertex!
- Alternative: develop *connectivity matrix* from adjacency matrix
  - *Transitive closure* of adjacency matrix
  - If L → M and M → N, L → N

Warshall's Algorithm

- For all rows *y*,
  - For all columns *x* in row *y*,
    - If any value (x,y) is 1, then for all rows *z* in column *y*,
      - If (y,z) is 1, then (x,z) should be 1
- i.e., "transitive closure"

Warshall's Algorithm (II)

- That's it!
  - Remember array references are "backwards" [y][x]
- Yes, this actually works in one pass – all the holes are filled
- What's the complexity of *this* algorithm?

Weighted graphs

- How to represent? Not just 0s and 1s in the adjacency matrix; weight instead
- Example
  - Roadmap!
- Can be directed or undirected

MSTs with weights

- Many possible STs; how do we figure out the minimum?
- Simple idea: grow the tree from one node
  - Pick smallest edge from vertices that we know to nodes not in tree
  - Add edge and corresponding destination vertex to tree
  - Add edges from new vertex to unknown nodes into priority queue
- Picking smallest edges: priority queue
- Applications
  - Minimizing wiring given multiple choices
  - In general, *undirected* graphs

However…

- If an edge to a destination vertex already exists in PQ, and we find a shorter path, need to *replace* the existing entry with shorter path
  - Simplest way: scan through PQ, see if any such edges exist, remove them, and insert the new one
  - Slicker ways of doing it include backpointers from vertices
- By the way, this is called "Prim"

Shortest-path problem

- Given a graph with weighted edges, and a starting vertex, find shortest path to a target
- Dijkstra's algorithm most canonical way of doing it
- So turns out you get shortest paths to all remote vertices from that starting vertex
- Can handle both directed and undirected graphs
  - Produces a directed tree
- *Cannot* handle negative weights

Dijkstra's Algorithm: Basic idea

- Initialize an array of distances from starting node to each vertex – if there doesn't exist a direct edge to a vertex, consider it at "infinite" distance
- Add the closest node not already in the shortest-path tree
- Update weights based on edges from newest node plus distance from starting to new – and keep track of the node we used to get to that target
- Repeat
- To find a path to a node, go backwards through the parent nodes

Next time

- Continue weighted graphs
- We're almost there. ☺