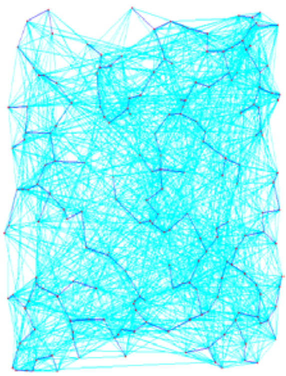


Finish Graphs

Lecture 25
CS211 – Fall 2006

Announcements

- Paul Chew's office hours for today (Tuesday, Nov 21) are cancelled
- No Section-meetings this week due to Thanksgiving Break

Recall: Greedy Algorithms

- Dijkstra's Algorithm is an example of a Greedy Algorithm
- The Greedy Strategy is an algorithm design technique
  - Like Divide & Conquer
- The greedy algorithms are used to solve optimization problems
  - The goal is to find the *best* solution
- Works when the problem has the greedy-choice property
  - A global optimum can be reached by making locally optimum choices
- Problem: Given an amount of money, find the smallest number of coins to make that amount
- Solution: Use a Greedy Algorithm
  - Give as many large coins as you can
- This greedy strategy produces the optimum number of coins for the US coin system
- Different money system ⇒ greedy strategy may fail
  - Example: suppose the US introduces a 4¢ coin

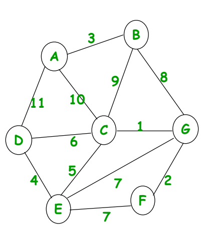
Minimum Spanning Trees

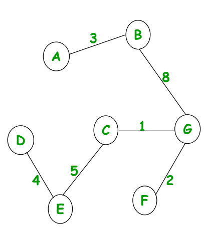
<u>Definition</u>
A *spanning tree* of an undirected graph *G* is a *tree* whose nodes are the vertices of *G* and whose edges are a subset of the edges of *G*

<u>Definition</u>
A *Minimum Spanning Tree* (*MST*) for a weighted graph *G* is the spanning tree of least cost (sum of edge-weights)

- Alternately, an MST can be defined as the least-cost set of edges so that all the vertices are connected
  - This has to be a tree… Why?
- A greedy strategy works for this problem
  - Add vertices one at a time
  - Always add the one that is closest to the current tree
  - This is called *Prim's Algorithm*

An Example Graph and Its MST





Prim's Algorithm

- s is the start vertex
- c(i,j) is the cost from i to j
- Initially, vertices are unmarked
- dist[v] is length of smallest tree-to-v edge
- Initially, dist[v] = ∞, for all v

```
prim(s):
    dist[s] = 0;
    while (some vertices are unmarked) {
        v = unmarked vertex with
            smallest dist;
        Mark v;
        for (each w adj to v) {
            dist[w] = min[ dist[w], c(v,w) ];
        }
    }
```

- Runtime analysis
  - $O(n^2)$ for adj matrix
    - While-loop is executed n times
    - O(n) time for for-loop
  - O(m + n log n) for adj list
    - <u>Total</u> time in for-loop is O(m)
    - Use a PQ to find smallest dist
      - Regular PQ produces time O(m log m)
      - Can improve to O(m + n log n) by using fancier heap

Similar Code Structures

```
while (some vertices are unmarked) {
   v = best of unmarked vertices;
   Mark v;
   for (each w adj to v)
         Update w;
}
```

- bfsDistance
  - best: next in queue
  - update: dist[w] = dist[v]+1
- dijkstra
  - best: next in PQ
  - update:dist[w] =min [dist[w],dist[v]+cost(v,w)]
- prim
  - best: next in PQ
  - update: dist[w] = min [dist[w],cost(v,w)]

Remembering Your Choices

- How can you remember which choices were made?
  - Whenever dist[w] is updated we can remember the current v by using parent[w] = v;
  - Can use the parent info to construct the *bfs tree*, the *shortest path tree*, or the *minimum spanning tree*

```
while (some vertices are unmarked) {
   v = best of unmarked vertices;
   Mark v;
   for (each w adj to v)
         Update w;
         if (w changed) parent[w] = v;
}
```

Two MST Algorithms (Both Greedy)

Kruskal's Algorithm

- Choose the shortest edge e such that
  - e is not yet processed
  - e does not make a cycle

Prim's Algorithm

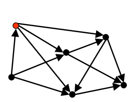
- Choose the shortest edge e such that
  - e touches the tree
  - e touches a vertex not in the tree

Depth-First Search

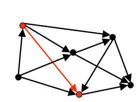
- Follow edges depth-first starting from an arbitrary vertex s, using a *Stack* to remember where you came from
- When you encounter a vertex previously visited, or there are no outgoing edges, retreat and try another path
- Eventually visit all vertices reachable from s
- If there are still unvisited vertices, repeat

Easy to see this takes O(m) time

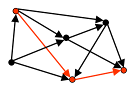
Depth-First Search



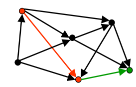
Depth-First Search



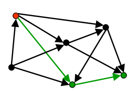
Depth-First Search



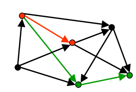
Depth-First Search



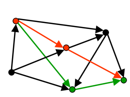
Depth-First Search



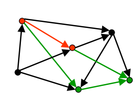
Depth-First Search



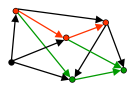
Depth-First Search



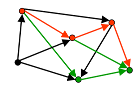
Depth-First Search



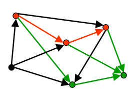
Depth-First Search



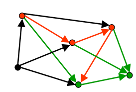
Depth-First Search



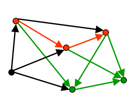
Depth-First Search



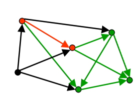
Depth-First Search



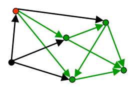
Depth-First Search



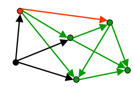
Depth-First Search



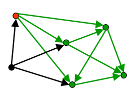
Depth-First Search



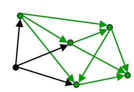
Depth-First Search



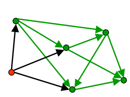
Depth-First Search



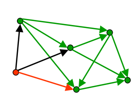
Depth-First Search



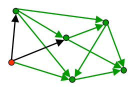
Depth-First Search



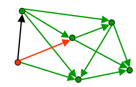
Depth-First Search



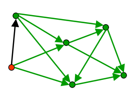
Depth-First Search



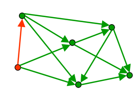
Depth-First Search



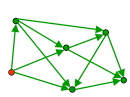
Depth-First Search



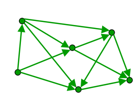
Depth-First Search



Depth-First Search



Depth-First Search



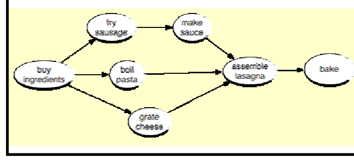
DFS Notes

- Same as BFS, except we use a Stack instead of a Queue to determine which edge to explore next
- Can also implement DFS recursively
  - The Stack is represented *implicitly* in the Stack Frames created by the recursive calls

Topological Sort using Recursive DFS

- Recall topological sort: find an ordering for the nodes of a dag so that all edges are "forward" edges
- Can use recursive DFS to do topological sort
  - Call DFS_visit on each *unmarked* vertex

DFS_visit (Vertex v):
  Mark v as active
  for each successor w of v:
    if w not yet marked:
        DFS_visit(v)
    if w is active: Exception
  Mark v as done
  Add v to *head* of topSort list



Graph Overview

- Graph Definitions
  - Directed graph (digraph)
  - Undirected graph
  - Directed acyclic graph (dag)
  - Paths & cycles
- Graph Properties
  - Graph coloring
  - Planarity
  - Bipartite graphs
- Graph Implementations
  - Adjacency matrix
  - Adjacency lists
- Graph Searching
  - Breadth First Search (BFS)
  - Depth First Search (DFS)
- Other Graph Algorithms
  - Topological Sort
    - Using in-degree-0 nodes
    - Using DFS
  - Single-source shortest paths
    - Dijkstra's Algorithm
  - Minimum spanning tree (MST)
    - Prim's Algorithm
    - (Kruskal's Algorithm)