Announcements

• Reading
  – 4.4, 4.5, 4.7
• Midterm
  – Monday, October 30
  – In class, closed book
  – Material through 4.7
  – Old midterm questions available
    • Note – some listed questions are out of scope

Dijkstra’s Algorithm

Assume all edges have non-negative cost

Correctness Proof

• Elements in S have the correct label
• Induction: when v is added to S, it has the correct distance label
  – Dist(s, v) = d[v] when v added to S

Dijkstra Implementation

O(n^2) Implementation for Dense Graphs
Future stuff for shortest paths

- Bellman-Ford Algorithm
  - $O(nm)$ time
  - Handles negative cost edges
    - Identifies negative cost cycle if present
  - Dynamic programming algorithm
  - Very easy to implement

Bottleneck Shortest Path

- Define the bottleneck distance for a path to be the maximum cost edge along the path

Compute the bottleneck shortest paths

How do you adapt Dijkstra’s algorithm to handle bottleneck distances

- Does the correctness proof still apply?

Dijkstra’s Algorithm for Bottleneck Shortest Paths

Minimum Spanning Tree

- Introduce Problem
- Demonstrate three different greedy algorithms
- Provide proofs that the algorithms work
Minimum Spanning Tree Definitions

- \( G = (V, E) \) is an UNDIRECTED graph
- Weights associated with the edges
- Find a spanning tree of minimum weight
  - If not connected, complains

Greedy Algorithms for Minimum Spanning Tree

- Extend a tree by including the cheapest outgoing edge
- Add the cheapest edge that joins disjoint components
- Delete the most expensive edge that does not disconnect the graph

Greedy Algorithm 1
Prim’s Algorithm

- Extend a tree by including the cheapest outgoing edge

Greedy Algorithm 2
Kruskal’s Algorithm

- Add the cheapest edge that joins disjoint components

Greedy Algorithm 3
Reverse-Delete Algorithm

- Delete the most expensive edge that does not disconnect the graph
Dijkstra's Algorithm for Minimum Spanning Trees

1. \( S = {} \), \( d[s] = 0 \), \( d[v] = \infty \) for \( v \neq s \)

2. While \( S \neq V \):
   a. Choose \( v \) in \( V - S \) with minimum \( d[v] \)
   b. Add \( v \) to \( S \)
   c. For each \( w \) in the neighborhood of \( v \), \( d[w] = \min(d[w], c(v, w)) \)

Greedy Algorithms for Minimum Spanning Tree

- **[Prim]** Extend a tree by including the cheapest outgoing edge
- **[Kruskal]** Add the cheapest edge that joins disjoint components
- **[ReverseDelete]** Delete the most expensive edge that does not disconnect the graph

Why do the greedy algorithms work?

- For simplicity, assume all edge costs are distinct

Edge inclusion lemma

- Let \( S \) be a subset of \( V \), and suppose \( e = (u, v) \) is the minimum cost edge of \( E \), with \( u \) in \( S \) and \( v \) in \( V - S \)
- \( e \) is in every minimum spanning tree of \( G \)
  - Or equivalently, if \( e \) is not in \( T \), then \( T \) is not a minimum spanning tree

Proof

- Suppose \( T \) is a spanning tree that does not contain \( e \)
- Add \( e \) to \( T \), this creates a cycle
- The cycle must have some edge \( e_i = (u_i, v_i) \) with \( u_i \) in \( S \) and \( v_i \) in \( V - S \)
- \( T_i = T - \{e_i\} + \{e\} \) is a spanning tree with lower cost
  - Hence, \( T \) is not a minimum spanning tree