Who was Dijkstra?

• What were his major contributions?

http://www.cs.utexas.edu/users/EWD/

• Edsger Wybe Dijkstra was one of the most influential members of computing science’s founding generation. Among the domains in which his scientific contributions are fundamental are
  – algorithm design
  – programming languages
  – program design
  – operating systems
  – distributed processing
  – formal specification and verification
  – design of mathematical arguments

Shortest Paths

• Negative Cost Edges
  – Dijkstra’s algorithm assumes positive cost edges
  – For some applications, negative cost edges make sense
  – Shortest path not well defined if a graph has a negative cost cycle

Negative Cost Edge Preview

• Topological Sort can be used for solving the shortest path problem in directed acyclic graphs
• Bellman-Ford algorithm finds shortest paths in a graph with negative cost edges (or reports the existence of a negative cost cycle).

Dijkstra’s Algorithm

Implementation and Runtime

S = \{\}; d[s] = 0; d[v] = \infty for v \neq s
While S \neq V
  Choose v in V-S with minimum d[v]
  Add v to S
  For each w in the neighborhood of v
    d[w] = \min(d[w], d[v] + c(v, w))

Edge costs are assumed to be non-negative
**Bottleneck Shortest Path**

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

**Compute the bottleneck shortest paths**

**Dijkstra's Algorithm for Bottleneck Shortest Paths**

\[ S = \{\}; \quad d[s] = \text{negative infinity}; \quad d[v] = \text{infinity for } v \neq s \]

While \( S \neq V \)

1. Choose \( v \) in \( V - S \) with minimum \( d[v] \)
2. Add \( v \) to \( S \)
3. For each \( w \) in the neighborhood of \( v \)
   \[ d[w] = \min(d[w], \max(d[v], c(v, w))) \]

**Minimum Spanning Tree**

- Introduce Problem
- Demonstrate three different greedy algorithms
- Provide proofs that the algorithms work

**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

Why do the greedy algorithms work?
- For simplicity, assume all edge costs are distinct
- 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

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

Optimality Proofs
- Prim’s Algorithm computes a MST
- Kruskal’s Algorithm computes a MST
Reverse-Delete Algorithm

- Lemma: The most expensive edge on a cycle is never in a minimum spanning tree

Dealing with the assumption of no equal weight edges

- Force the edge weights to be distinct
  - Add small quantities to the weights
  - Give a tie breaking rule for equal weight edges

Dijkstra's Algorithm for Minimum Spanning Trees

\[
S = \{\}; \quad d[s] = 0; \quad d[v] = \text{infinity for } v \neq s
\]

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