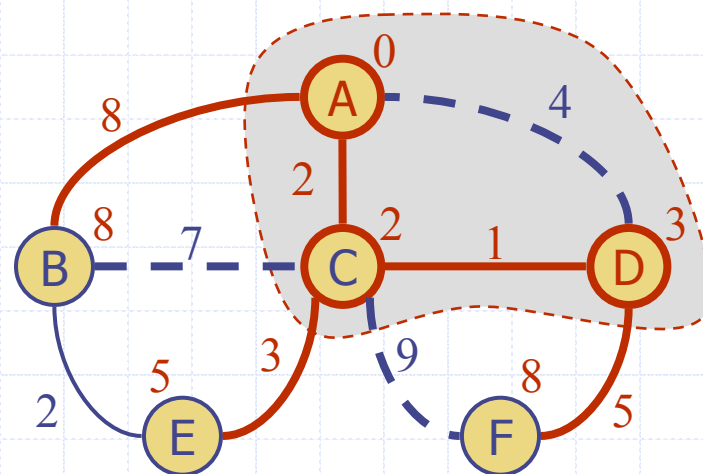


Shortest Path

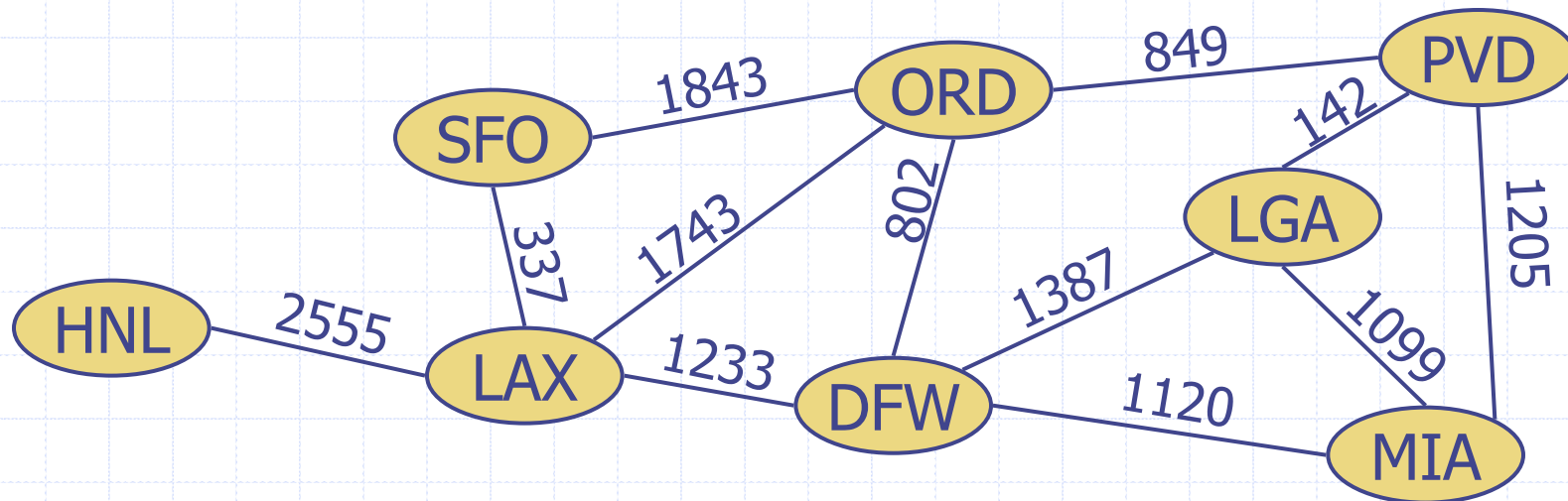


Outline and Reading

- ◆ Shortest path (§13.6)
 - Weighted graph (§13.5)
 - Shortest path problem
 - Shortest path properties
- ◆ Dijkstra's algorithm (§13.6.1)
 - Algorithm
 - Edge relaxation
 - Example
 - Analysis

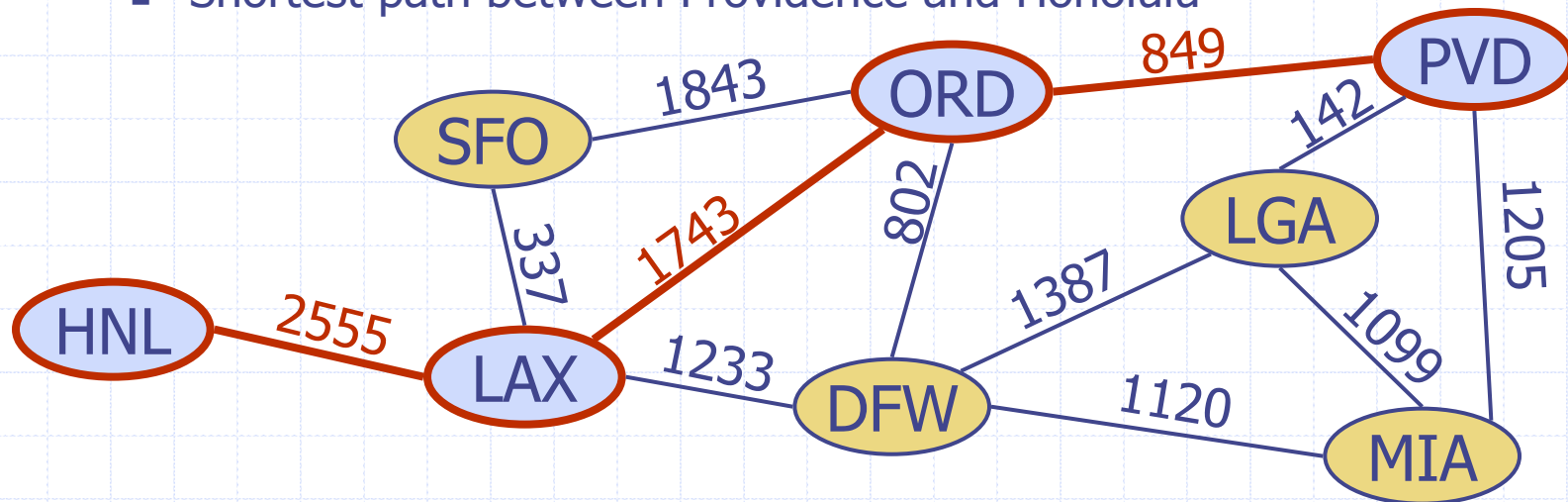
Weighted Graph

- ◆ In a weighted graph, each edge has an associated numerical value, called the weight of the edge
- ◆ Edge weights may represent, distances, costs, etc.
- ◆ Example:
 - In a flight route graph, the weight of an edge represents the distance in miles between the endpoint airports



Shortest Path Problem

- ◆ Given a weighted graph and two vertices u and v , we want to find a path of minimum total weight between u and v
- ◆ Applications
 - Flight reservations
 - Driving directions
 - Internet packet routing
- ◆ Example:
 - Shortest path between Providence and Honolulu



Shortest Path Properties

Property 1:

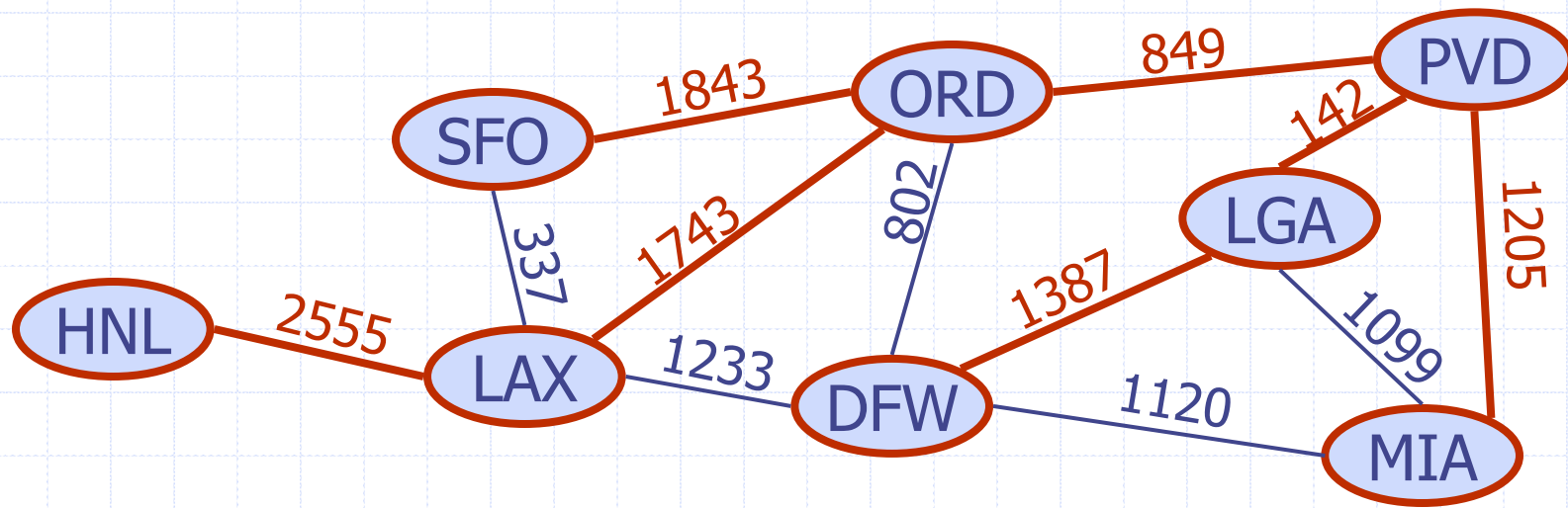
A subpath of a shortest path is itself a shortest path

Property 2:

There is a tree of shortest paths from a start vertex to all the other vertices

Example:

Tree of shortest paths from Providence



Dijkstra's Algorithm

- ◆ The distance of a vertex v from a vertex s is the length of a shortest path between s and v
- ◆ Dijkstra's algorithm computes the distances of all the vertices from a given start vertex s
- ◆ Assumptions:
 - the graph is connected
 - the edges are undirected
 - the edge weights are nonnegative
- ◆ We grow a "cloud" of vertices, beginning with s and eventually covering all the vertices
- ◆ We store with each vertex v a label $d(v)$ representing the distance of v from s in the subgraph consisting of the cloud and its adjacent vertices
- ◆ At each step
 - We add to the cloud the vertex u outside the cloud with the smallest distance label
 - We update the labels of the vertices adjacent to u

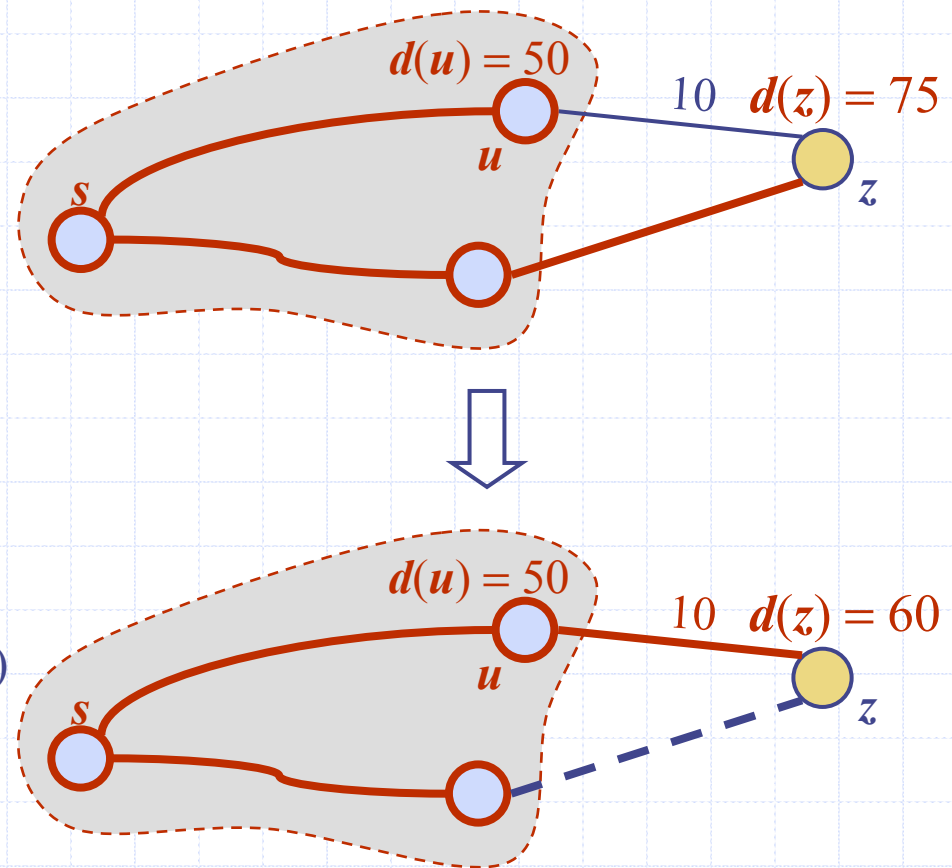
Edge Relaxation

◆ Consider an edge $e = (u, z)$ such that

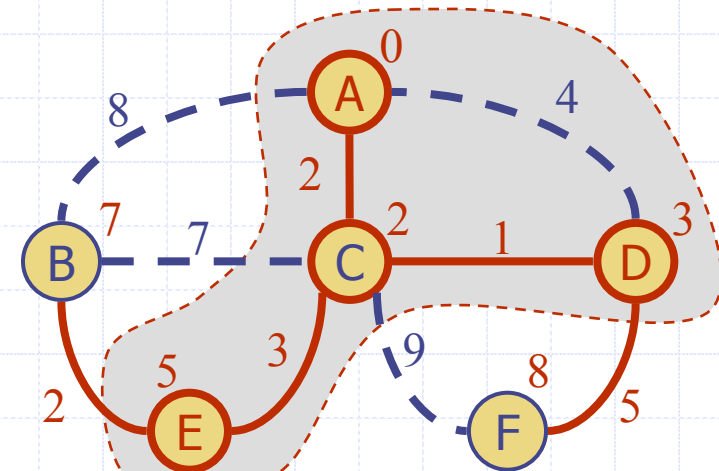
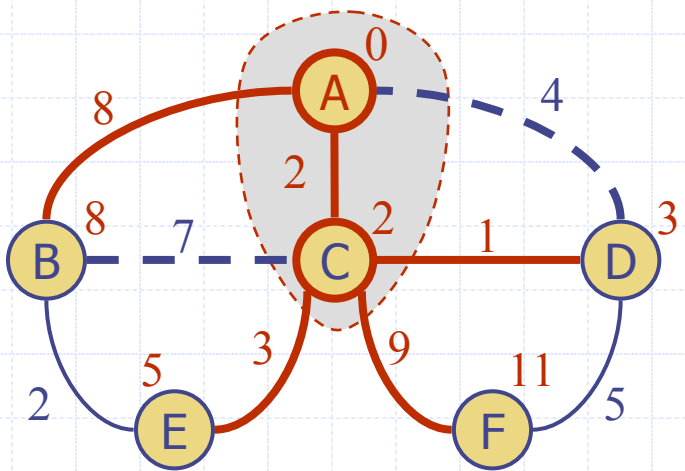
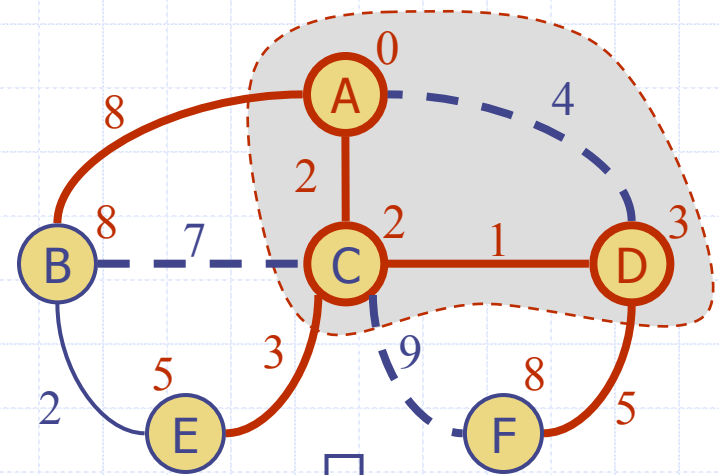
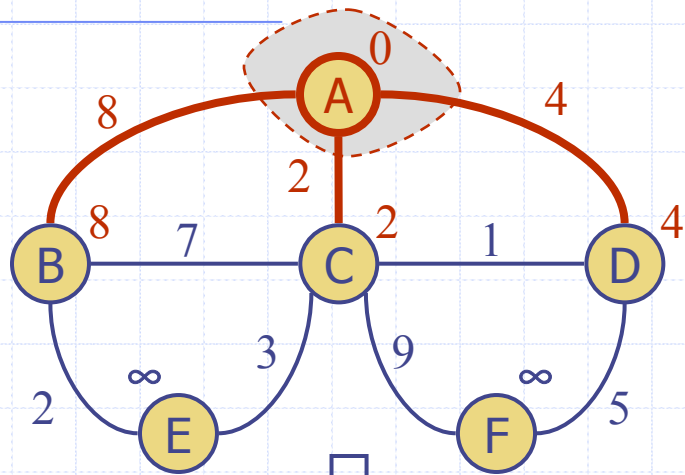
- u is the vertex most recently added to the cloud
- z is not in the cloud

◆ The relaxation of edge e updates distance $d(z)$ as follows

$$d(z) \leftarrow \min(d(z), d(u) + \text{weight}(e))$$



Example

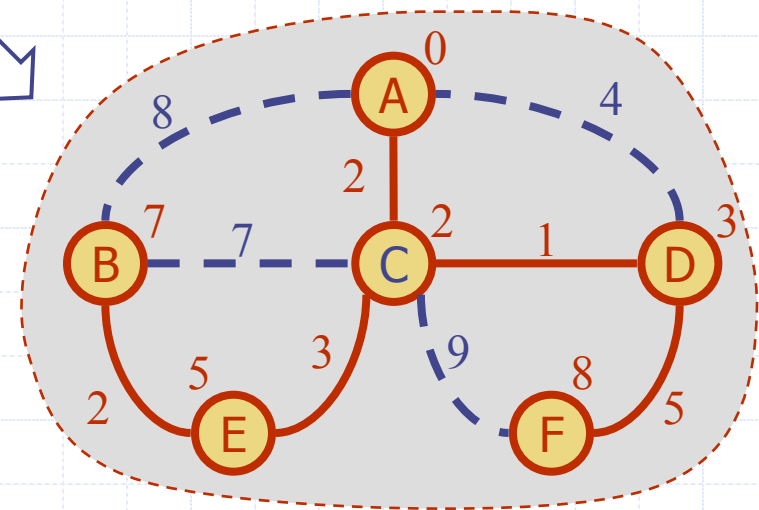
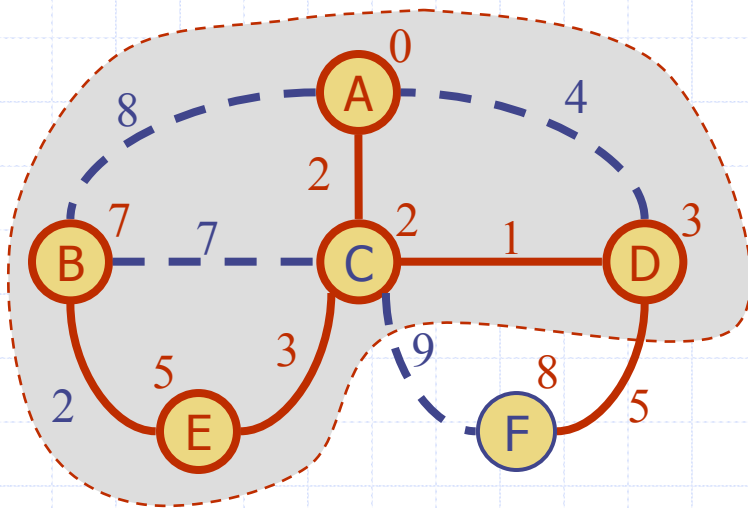


4/3/2006

Shortest Path

8

Example (cont.)



Dijkstra's Algorithm

- ◆ A priority queue stores the vertices outside the cloud
 - Key: distance
 - Element: vertex
- ◆ Adaptable priority queue methods
 - *insert(k,o)* returns an entry
 - *replaceKey(l,k)* changes the key of entry *l*
- ◆ We store two labels with each vertex:
 - Distance (*d(v)* label)
 - Entry in priority queue
- ◆ The distance labels give the output on termination

```
Algorithm DijkstraDistances(G, s)  
  Q ← new heap-based adaptable  
    priority queue  
  for all v ∈ G.vertices()  
    if v = s  
      setDistance(v, 0)  
    else  
      setDistance(v, ∞)  
      l ← Q.insert(getDistance(v), v)  
      setEntry(v,l)  
  while ¬Q.isEmpty()  
    u ← Q.removeMin().getValue()  
    for all e ∈ G.incidentEdges(u)  
      { relax edge e }  
      z ← G.opposite(u,e)  
      r ← getDistance(u) + weight(e)  
      if r < getDistance(z)  
        setDistance(z,r)  
        Q.replaceKey(getEntry(z),r)
```

Analysis

- ◆ Graph operations
 - Method `incidentEdges` is called once for each vertex
- ◆ Label operations
 - We set/get the distance and entry labels of vertex z $O(\deg(z))$ times
 - Setting/getting a label takes $O(1)$ time
- ◆ Priority queue operations
 - Each vertex is inserted once into and removed once from the priority queue, where each insertion or removal takes $O(\log n)$ time
 - The key of a vertex in the priority queue is replaced at most $\deg(w)$ times, where each key replacement takes $O(\log n)$ time
- ◆ Dijkstra's algorithm runs in $O((n + m) \log n)$ time provided the graph is represented by the adjacency list structure
 - Recall that $\sum_v \deg(v) = 2m$
- ◆ The running time can also be expressed as $O(m \log n)$ since the graph is connected

Extension

- ◆ Using the template method pattern, we can extend Dijkstra's algorithm to return a tree of shortest paths from the start vertex to all other vertices
- ◆ We store with each vertex a third label:
 - parent edge in the shortest path tree
- ◆ In the edge relaxation step, we update the parent label

Algorithm *DijkstraShortestPathsTree*(G, s)

```
...  
for all  $v \in G.vertices()$   
...  
  setParent( $v, \emptyset$ )  
...  
while  $\neg Q.isEmpty()$   
   $u \leftarrow Q.removeMin().getValue()$   
  for all  $e \in G.incidentEdges(u)$   
    { relax edge  $e$  }  
     $z \leftarrow G.opposite(u, e)$   
     $r \leftarrow getDistance(u) + weight(e)$   
    if  $r < getDistance(z)$   
      setDistance( $z, r$ )  
      setParent( $z, e$ )  
      Q.replaceKey(getEntry( $z$ ),  $r$ )
```