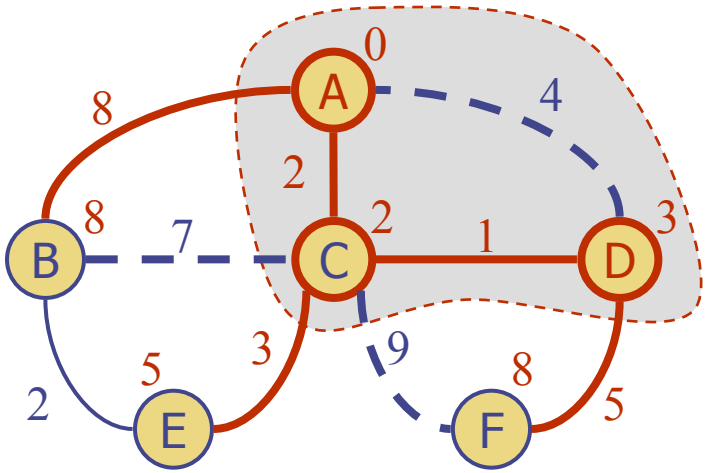
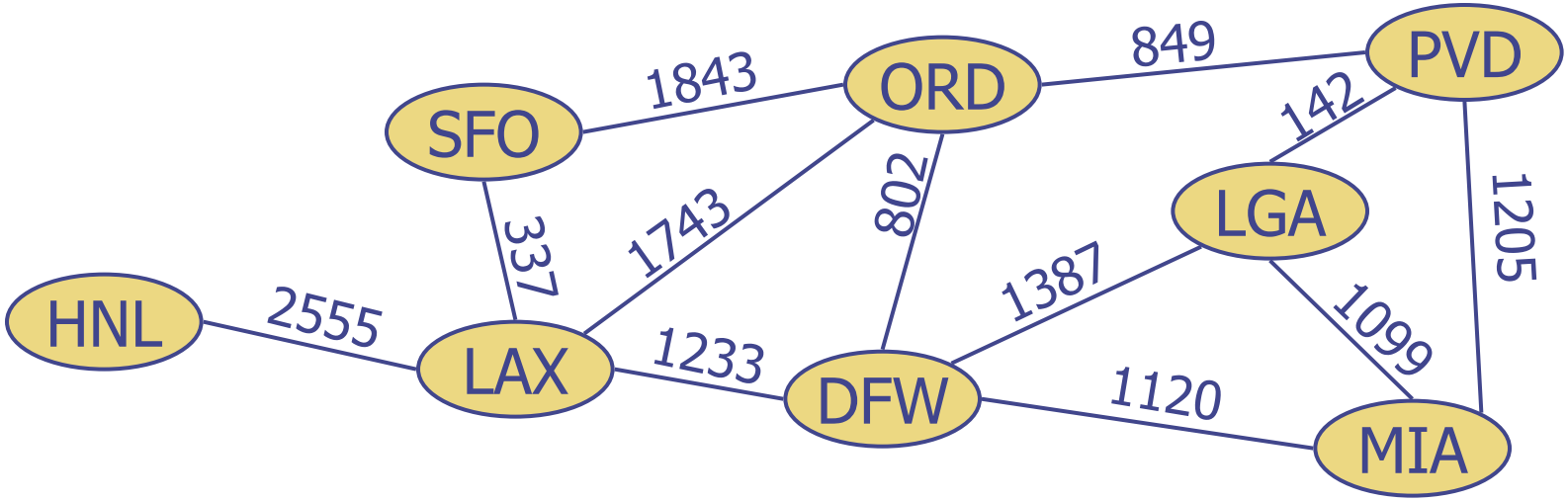
Shortest Paths



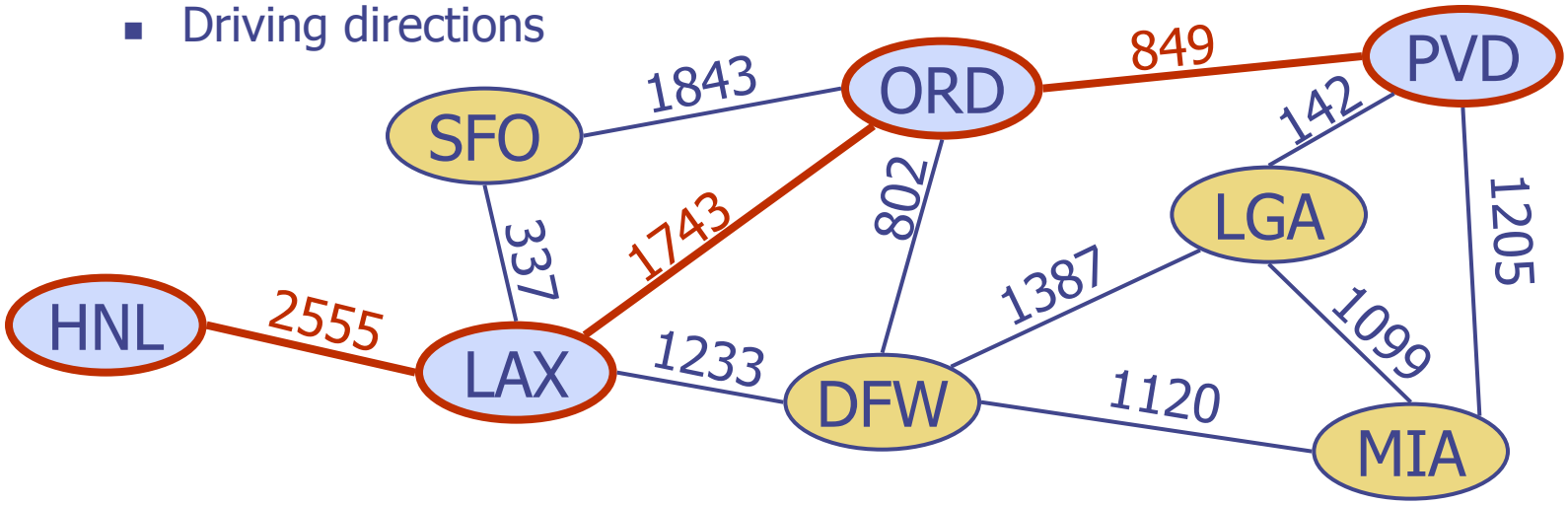
Weighted Graphs (§ 14.5)

- 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 Paths (§ 12.6)

- Given a weighted graph and two vertices u and v, we want to find a path of minimum total weight between u and v.
  - Length of a path is the sum of the weights of its edges.
- Example:
  - Shortest path between Providence and Honolulu
- Applications
  - Internet packet routing
  - Flight reservations
  - Driving directions



Shortest Path Properties

Property 1:

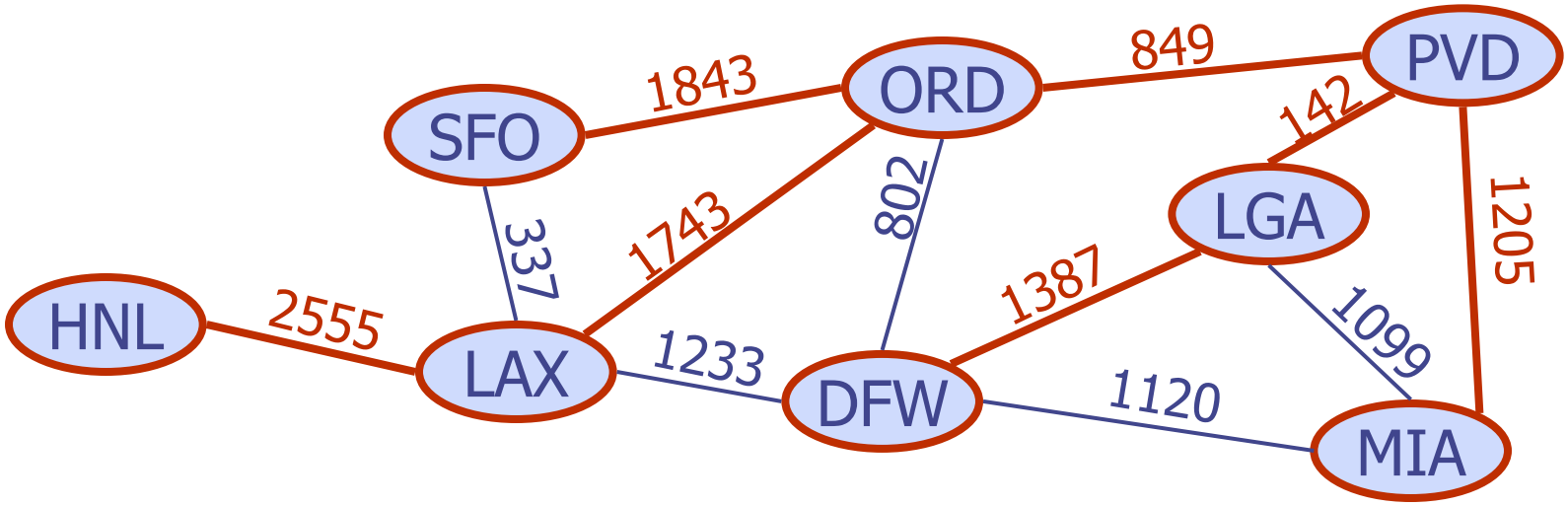
A subpath of a shortest path is itself a shortest path

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



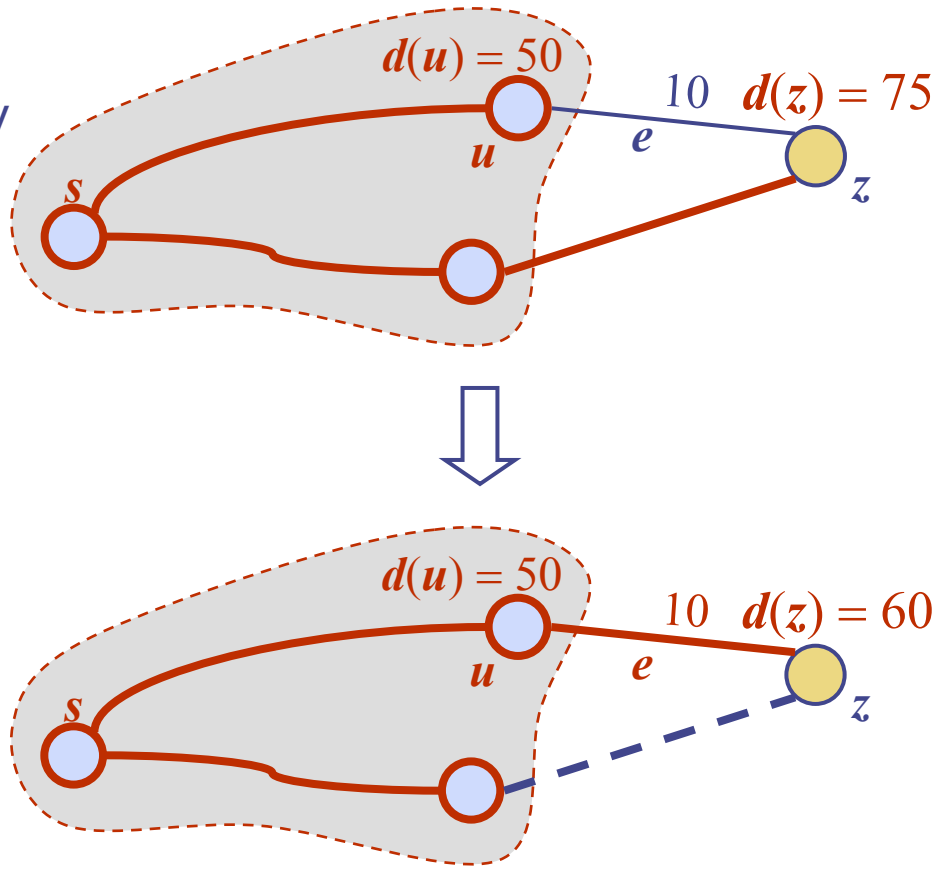
Dijkstra's Algorithm (§ 14.6.1)

- 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, $d(u)$
  - We update the labels of the vertices adjacent to u

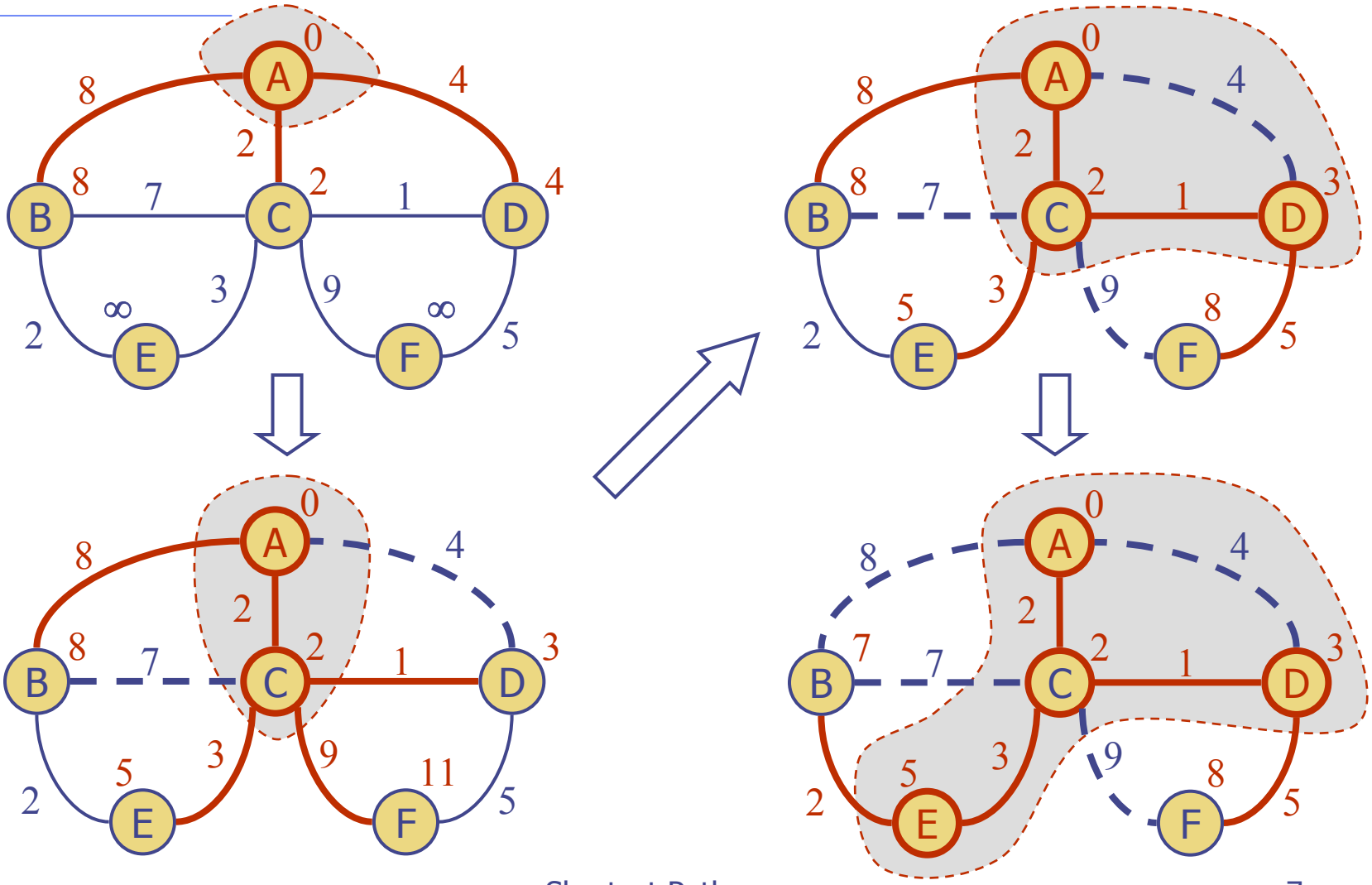
Edge Relaxation

- 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:

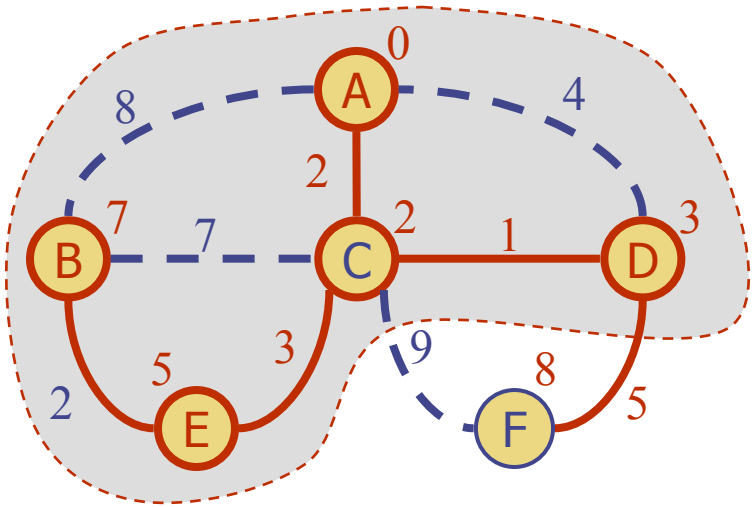
$$d(z) \leftarrow \min\{d(z), d(u) + weight(e)\}$$

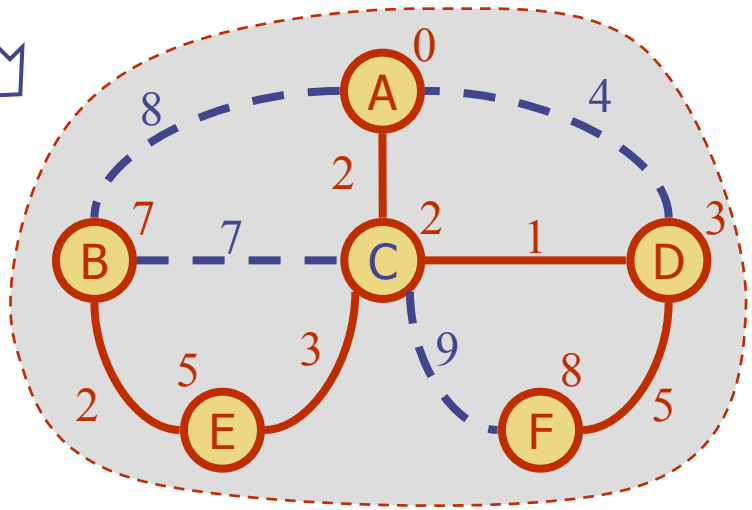


Example


A
B
C
D
E
F
0
8
4
2
8
7
2
1
4
2
∞
3
9
∞
5
A
B
C
D
E
F
0
8
4
2
8
7
2
1
3
2
5
3
9
8
5
A
B
C
D
E
F
0
8
4
2
8
7
2
1
3
2
5
3
9
11
5
A
B
C
D
E
F
0
8
4
2
7
7
2
1
3
2
5
3
9
8
5

Example (cont.)





Dijkstra's Algorithm

- A set S stores the vertices outside the cloud
- We store a label with each vertex:
  - Distance (d(v) label)

```
Algorithm DijkstraDistances(G, s)
    S ← {}
    for all v ∈ G.vertices()
        if v = s
            d(v) ← 0
        else d(v) ← ∞
        Add v to S
    while ¬S.isEmpty()
        Remove from S the vertex u
        with minimum value d(u)
        for all e ∈ G.incidentEdges(u)
            { relax edge e }
            z ← G.opposite(u,e)
            r ← d(u) + weight(e)
            if r < d(z)
                d(z) ← r
```

Analysis of Dijkstra's Algorithm

- Graph operations
  - Method incidentEdges is called once for each vertex
- Label operations
  - We set/get the distance label of vertex z $O(\deg(z))$ times
  - Setting/getting a label takes $O(1)$ time
- Vertex operations
  - Each vertex is selected one in the **while** loop; this takes $O(\log n)$ time if the vertices u are stored in a balanced tree using $d(u)$ as key
  - The key of a vertex in the balanced tree is modified at most $\deg(w)$ times, where each key change 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

Shortest Paths Tree

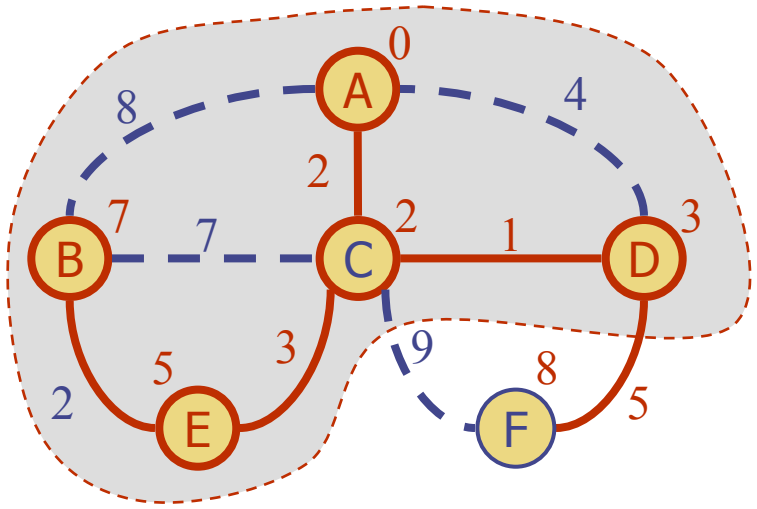
- We can extend Dijkstra's algorithm to return a tree of shortest paths from the start vertex to all other vertices
- We store with each vertex a second label:
  - parent edge in the shortest path tree
- In the edge relaxation step, we update the parent label

Algorithm *DijkstraShortestPathsTree*(*G*, *s*)

```
…
for all v ∈ G.vertices()
  …
  setParent(v, ∅)
  …
  for all e ∈ G.incidentEdges(u)
      { relax edge e }
      z ← G.opposite(u,e)
      r ← d(u) + weight(e)
      if r < d(z)
          d(z) ← r
          setParent(z,u)
```

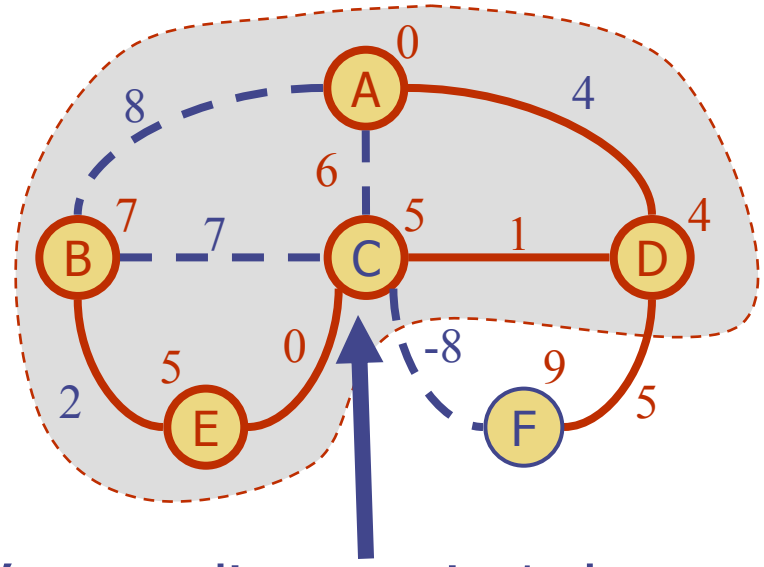
Why Dijkstra's Algorithm Works

- Dijkstra's algorithm is based on the greedy method. It adds vertices by increasing distance.
  - Suppose it didn't find all shortest distances. Let F be the first wrong vertex the algorithm processed.
  - When the previous node, D, on the true shortest path was considered, its distance was correct.
  - But the edge (D,F) was **relaxed** at that time!
  - Thus, so long as d(F)≥d(D), F's distance cannot be wrong. That is, there is no wrong vertex.



Why It Doesn't Work for Negative-Weight Edges

- Dijkstra's algorithm is based on the greedy method. It adds vertices by increasing distance.
  - If a node with a negative incident edge were to be added late to the cloud, it could mess up distances for vertices already in the cloud.



C's true distance is 1, but it is already in the cloud with d(C)=5!