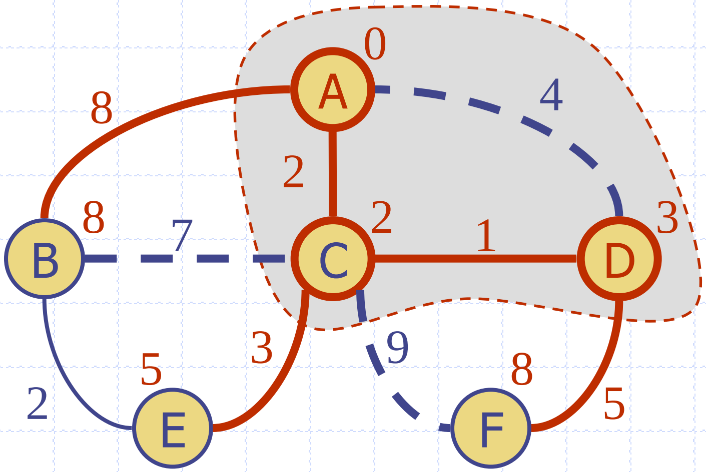
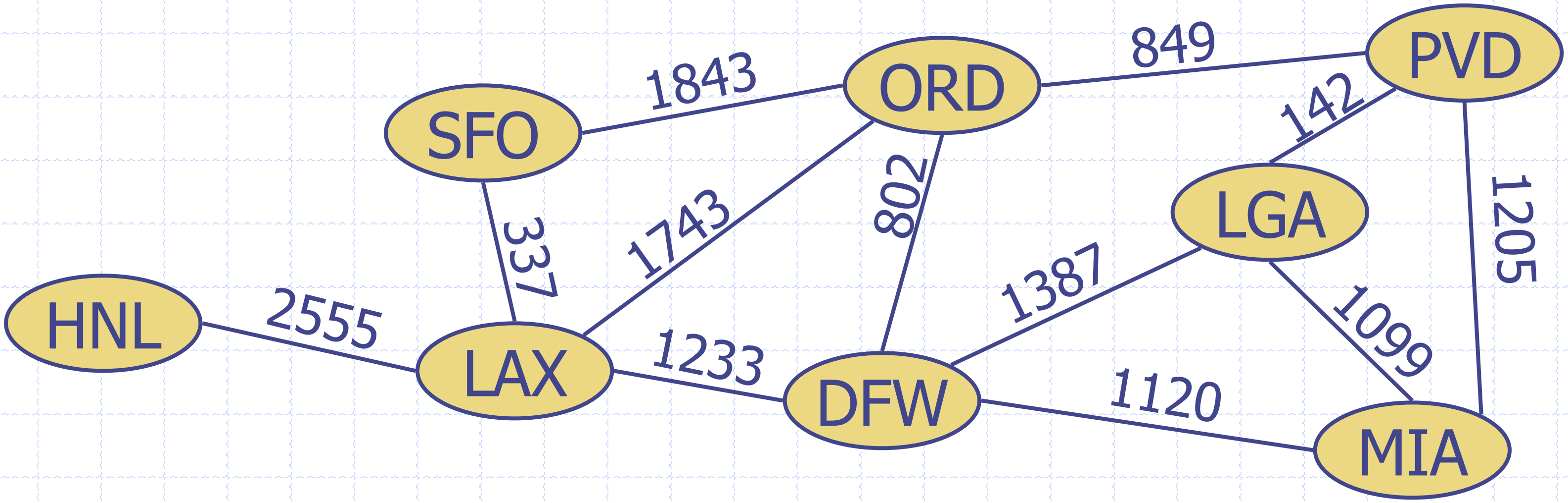
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Shortest Paths



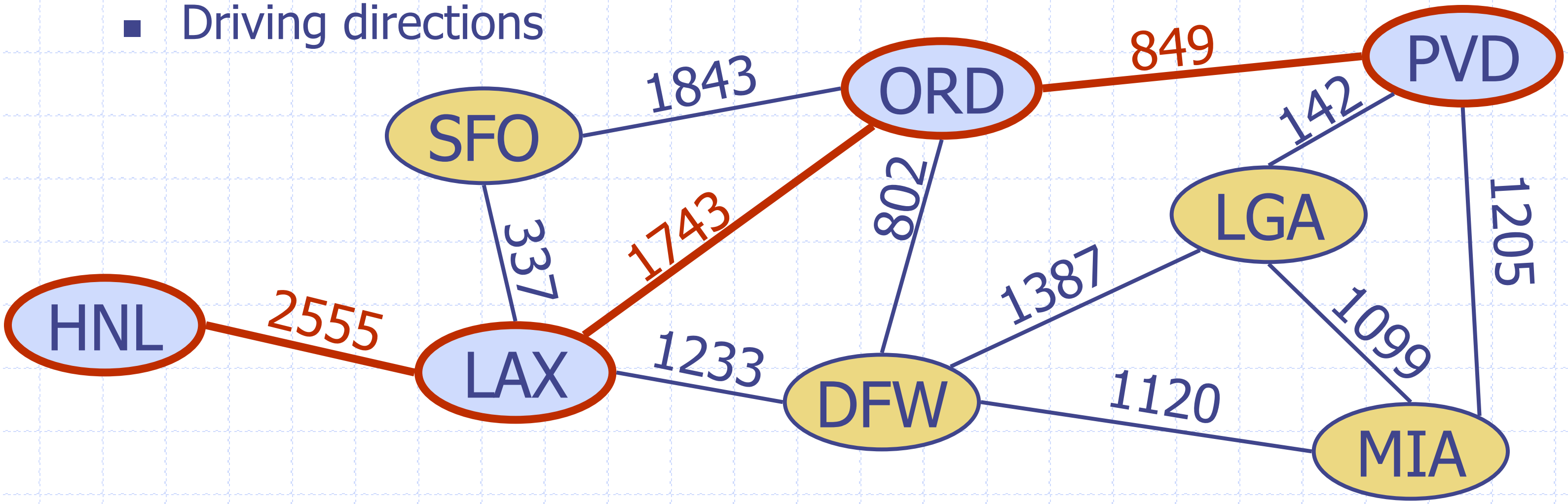
Weighted Graphs

- 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

- 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:

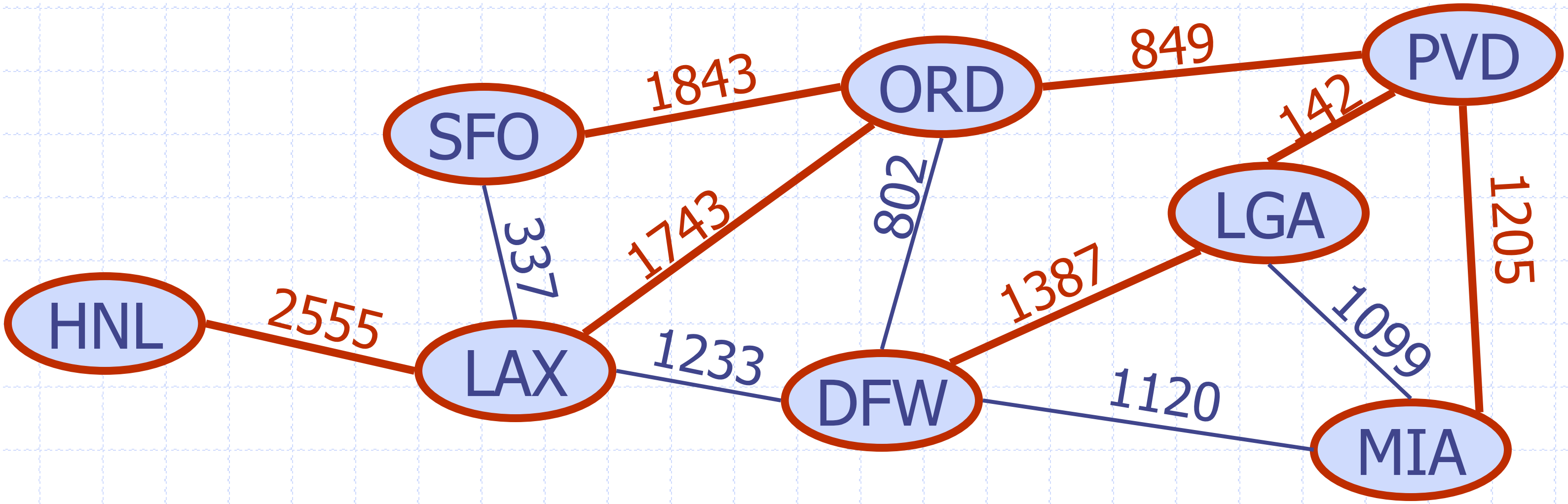
A subpath of a shortest path is itself a shortest path

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



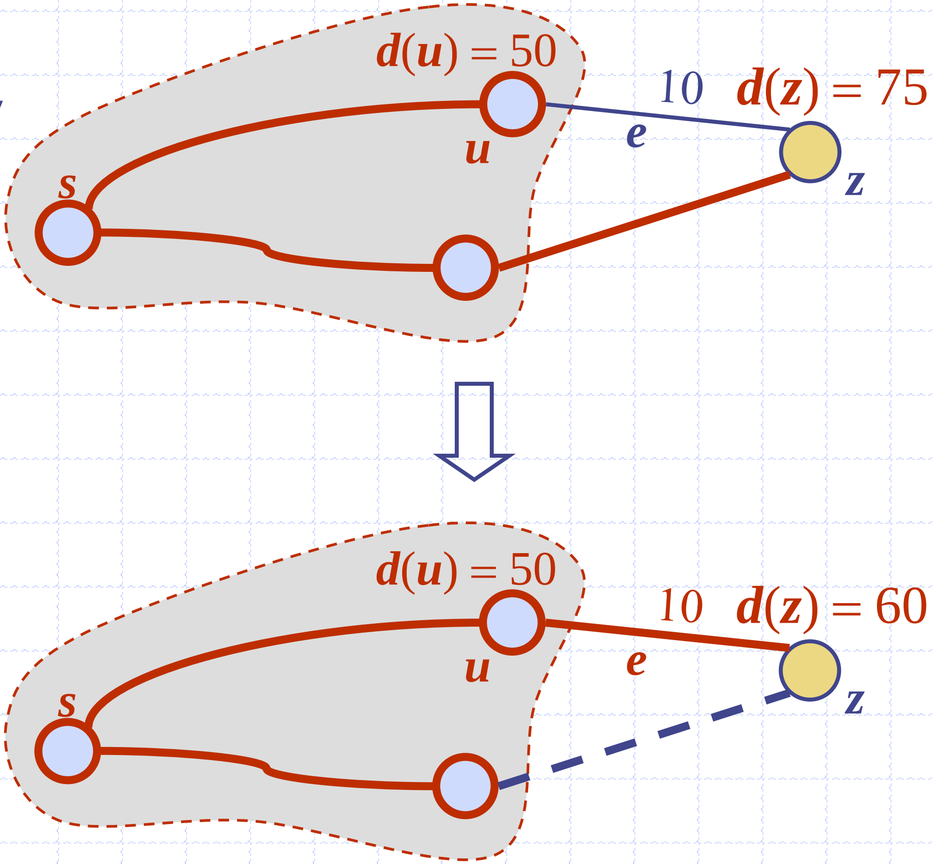
Dijkstra's Algorithm

- 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

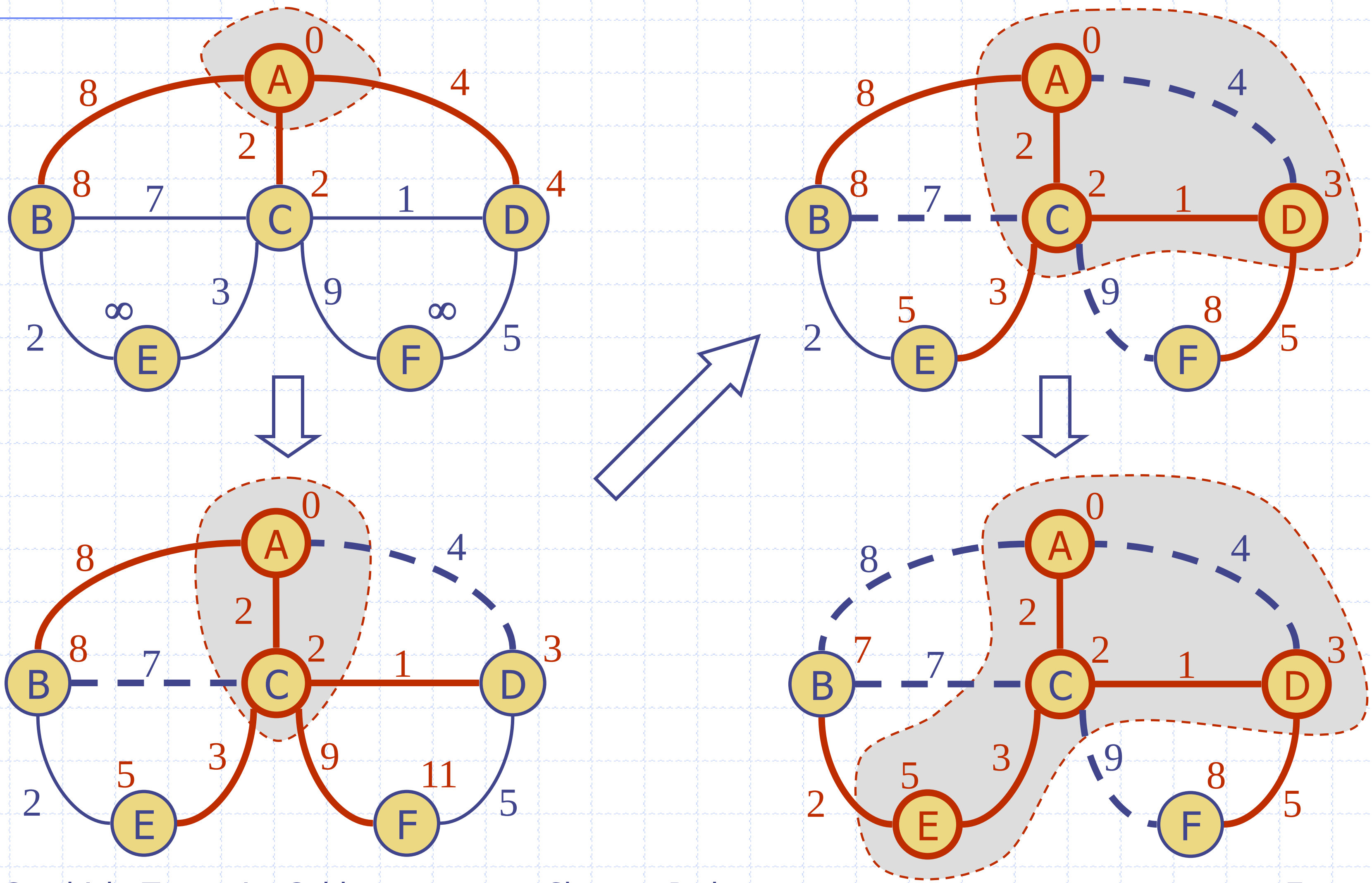
Edge Relaxation

- 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:

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

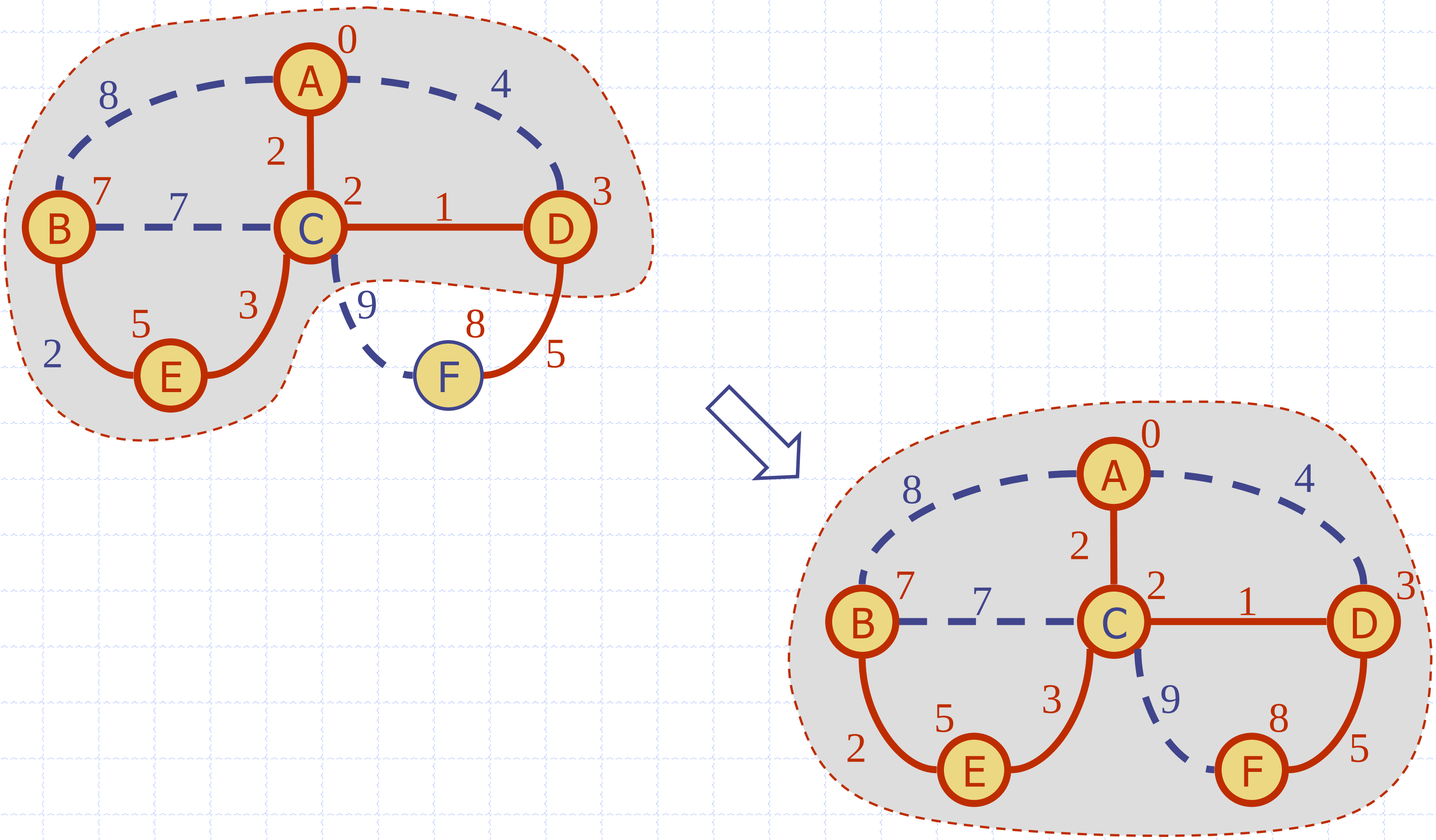


Example



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

Example (cont.)



Dijkstra's Algorithm

```
Algorithm Dijkstra (G,s)
In: weighted connected graph G and vertex s
Out: {compute shortest paths from s to the other vertices}
for each vertex u of G do {
    u.d ← ∞            // distance from vertex s to vertex u
    u.p ← null         // predecessor or parent of vertex u in a shortest paths tree
    u.marked ← false
}
s.d ← 0                // Distance from s to itself is 0
for i ← 0 to n-1 do {
    min ← ∞            // Find unmarked vertex u with minimum distance to s
    for each vertex v of G do
        if (v.marked = false) and (v.d < min) then {
            min ← v.d
            u ← v
        }
    u.marked ← true    // Relax all edges incident on vetex u
    for each edge (u,v) incident on u do
        if u.d + length(u,v) < v.d then {
            v.d ← u.d + length(u,v)
            v.p ← u
        }
}
```

c_1

$c_1 n$

c_2

$c_2 n$

c_4

$\sum_{u \in G} (c_4 + c_2 n + c_3 n)$ (adj. matrix)

c_3

$c_3 n$ (adj. matrix)

$c_3 \deg(u)$ (adj. list)

$\sum_{u \in G} (c_4 + c_2 n + c_3 \deg(u))$ (adj. list)

Analysis of Dijkstra's Algorithm

Using an adjacency matrix:

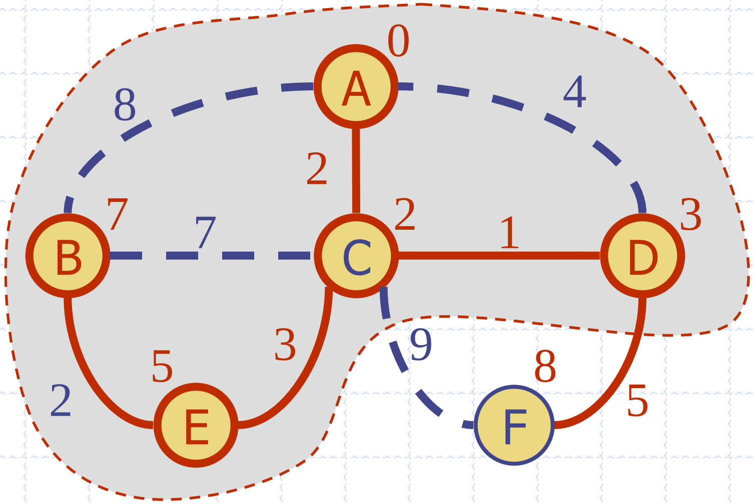
$$f(n,m) = c_1n + \sum_{u \,\varepsilon\, G}(c_4+c_2n+c_3n) = c_1n + c_4n + c_2n^2 + c_3n^2 \text{ is } O(n^2)$$

Using an adjacency list:

$$f(n,m) = c_1n + \sum_{u \,\varepsilon\, G}(c_4+c_2n+c_3deg(u)) = c_1n + c_4n + c_2n^2 + 2c_3m \text{ is } O(n^2)$$

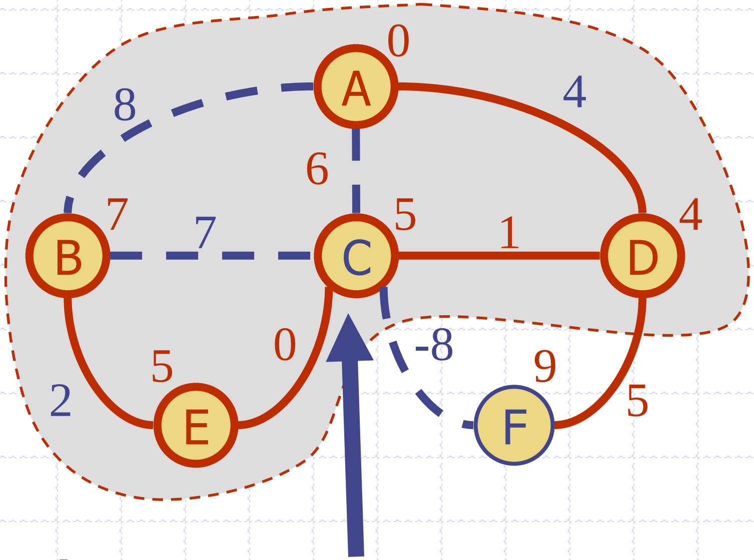
Why Dijkstra's Algorithm Works

- 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!