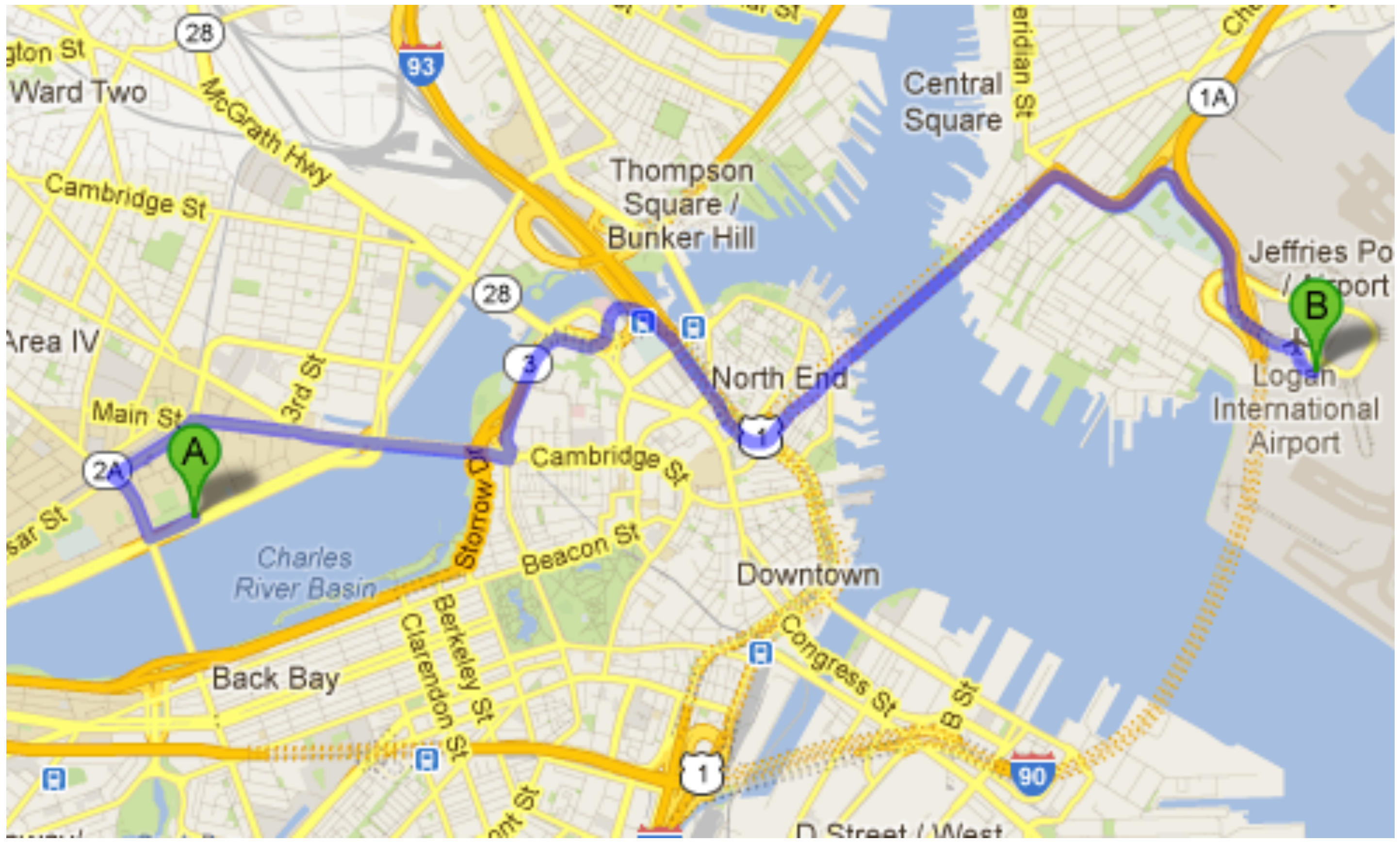
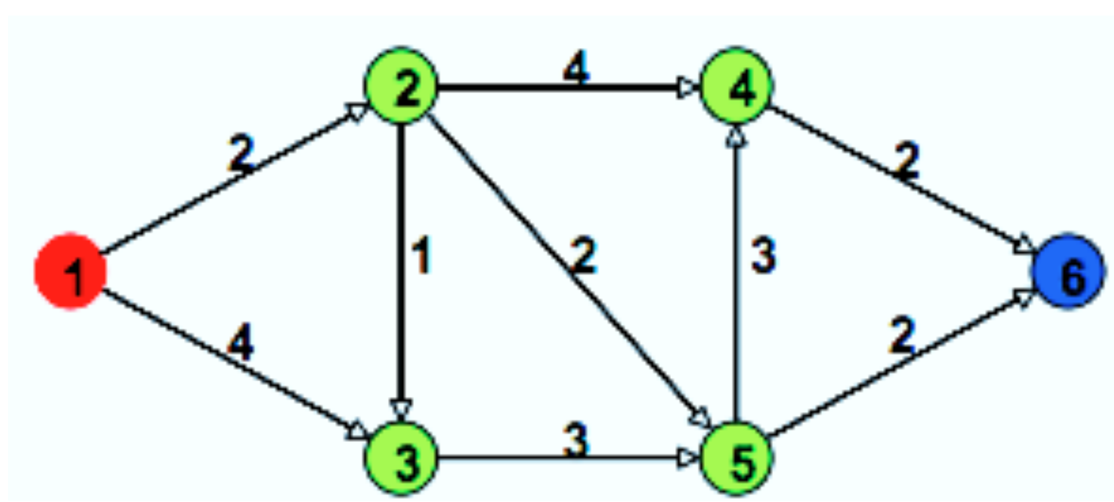
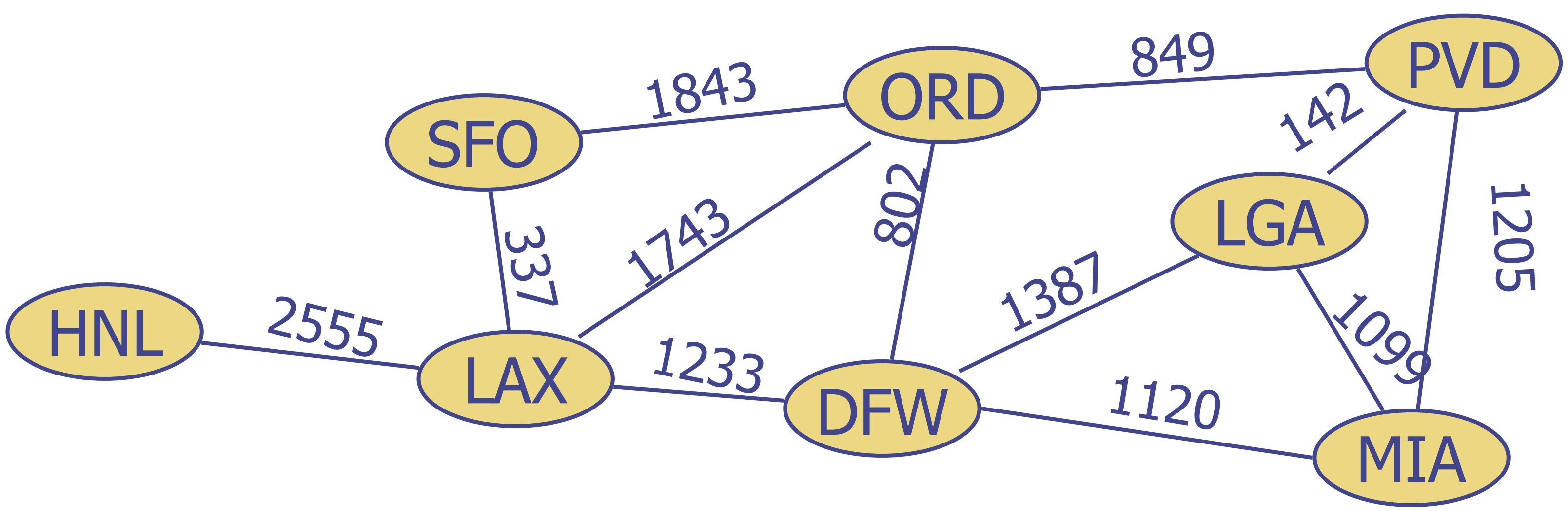
How to get to Logan airport?





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

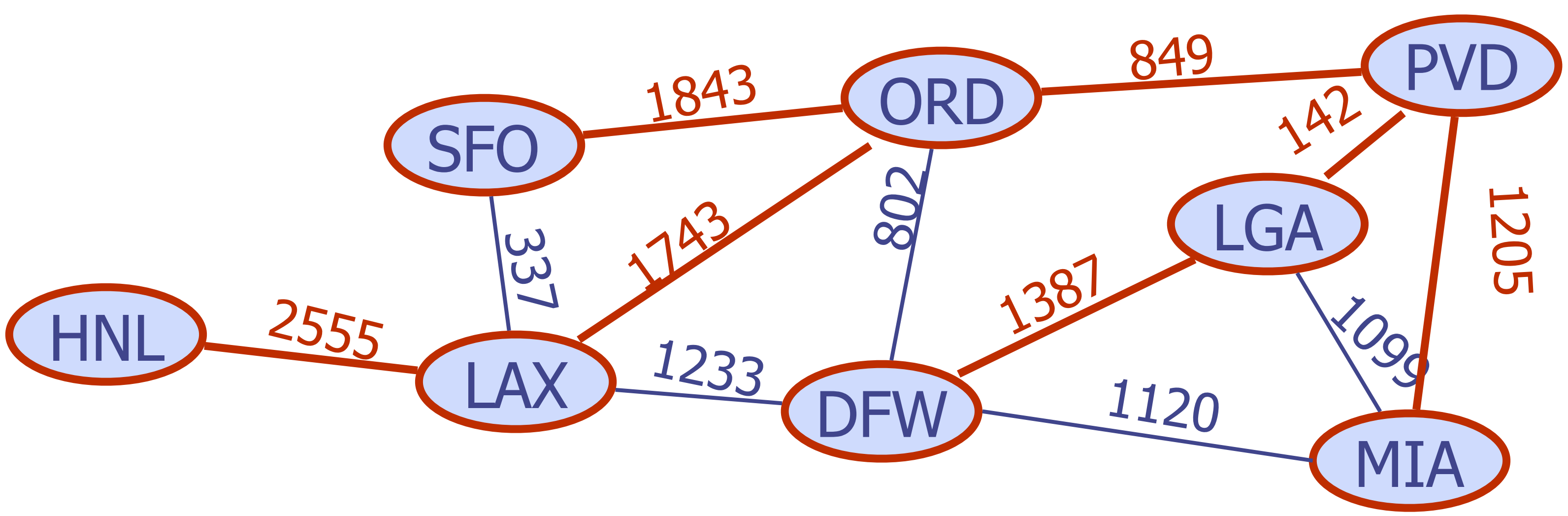


Shortest Path Properties

Property 1:
A subpath of a shortest path is itself a shortest path

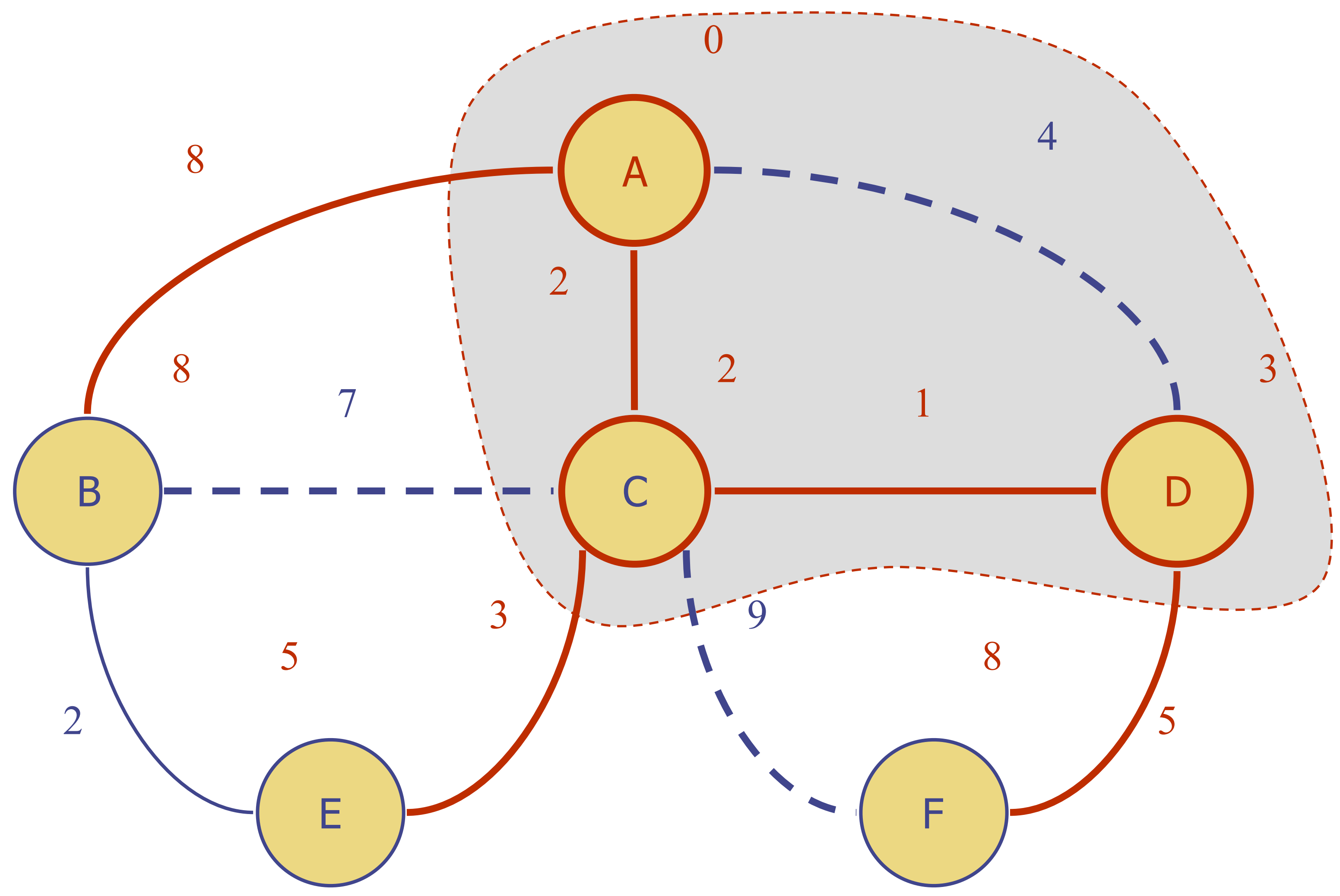
Property 2:
There is a **tree of shortest paths** from a start vertex to all the other vertices

Example:
Tree of shortest paths from Providence



Providence airport

Shortest Paths



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***

Dijkstra’s Algorithm

1 Assign to every node a tentative distance value: set it to **zero** for our initial node and to **infinity** for all other nodes.

2 Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the ***unvisited set***.

3 For the current node, **consider all of its unvisited neighbors and calculate their *tentative* distances**. **Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one.** For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbor *B* has length 2, then the distance to *B* (through *A*) will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.

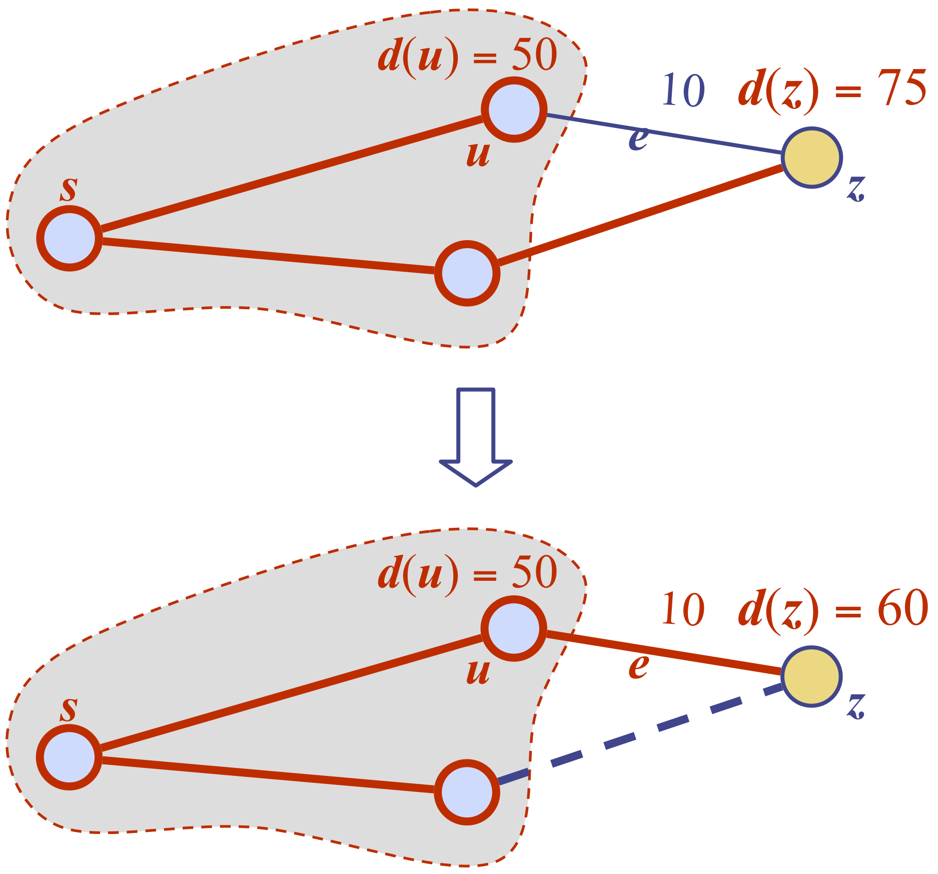
4 **When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.**

5 If the **destination node has been marked visited** (when planning a route between two specific nodes) **or if the smallest tentative distance among the nodes in the *unvisited set* is infinity** (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), **then stop.** The algorithm has finished.

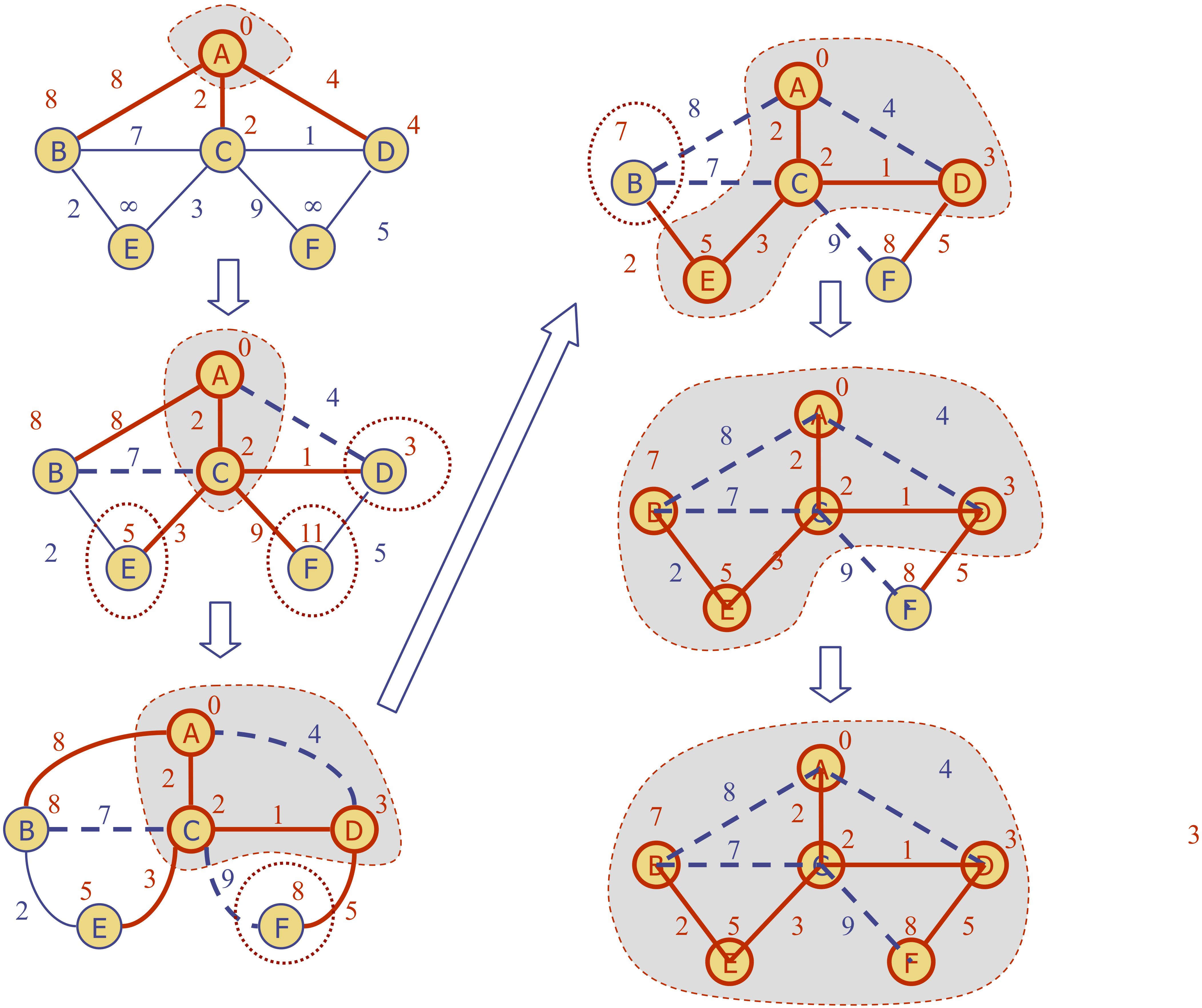
6 Otherwise, **select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.**

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) ← min{d(z),d(u) + weight(e)}



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

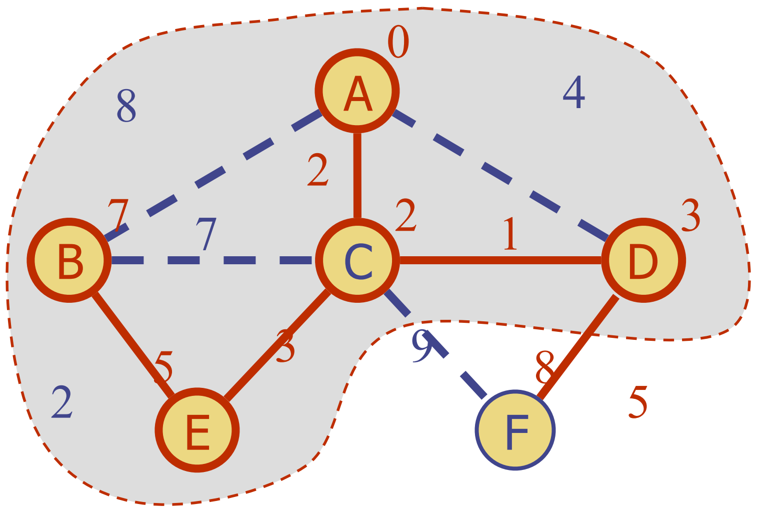
Analysis

- Graph operations
  - Method incidentEdges is called once for each vertex
- Label operations
  - We set/get the distance and locator 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 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 $\Sigma_v \deg(v) = 2m$
- The running time can also be expressed as $O(m \log n)$ since the graph is connected

Why Dijkstra's Algorithm Works

Dijkstra's algorithm is based on the greedy method. It adds vertices by increasing distance.

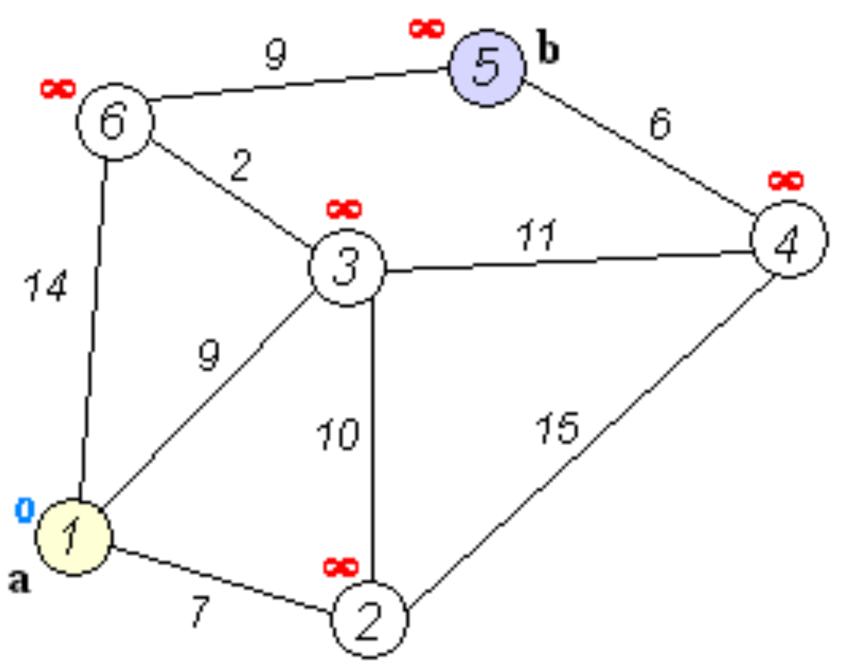
- Suppose it didn't find all shortest distances. Let F be the first wrong vertex the algorithm processed.
- When the previous node, D, on the true shortest path was considered, its distance was correct.
- But the edge (D,F) was **relaxed** at that time!
- Thus, so long as d(F)≥d(D), F's distance cannot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



Dijkstra's algorithm: Correctness proof by induction

- **G** is the input graph,
- **s** is the source vertex,
- **ℓ(u,v)** is the length of an edge from **u** to **v**
- **ℓ(Q)** is the length of a path **Q**
- **V** is the set of vertices
- **R** is the set of visited nodes
- **d(v)** is the current distance from the source, updated along the algorithm
- **δ(v)** be the shortest path distance from s-to-v (true distance)

$\textsc{Dijkstra}(G, s)$
  for all $u \in V \setminus \{s\}$, $d(u) = \infty$
  $d(s) = 0$
  $R = \{\}$
  while $R \neq V$
    pick $u \notin R$ with smallest $d(u)$
    $R = R \cup \{u\}$
    for all vertices v adjacent to u
      if $d(v) > d(u) + \ell(u, v)$
        $d(v) = d(u) + \ell(u, v)$



Let **d(v)** be the label found by the algorithm and let **δ(v)** be the shortest path distance from s-to-v.

We want to show that **d(v) = δ(v)** for every vertex **v** at the end of the algorithm, showing that the algorithm correctly computes the distances.

We prove this by induction on the cardinality of **R** (|R|) (**R** is the set of visited nodes and its size (cardinality) increases by one at each step of the algorithm)

Lemma: For each **x** ∈ **R**, **d(x) = δ(x)**.

Proof by Induction:

1) ***Base case*** (|R| = 1): Since **R** only grows in size, the only time |R| = 1 is when **R = {s}** and **d(s) = 0 = δ(s)**, which is correct.

2) ***Inductive hypothesis (IH) :*** Let **u** be the last vertex added to **R**. Let **R′ = R**∪**{u}**. Our I.H. is: for each **x** ∈ **R′**, **d(x) = δ(x)**.

Idea: By the inductive hypothesis, for every vertex in $\mathbf{R}'$ that isn't **u**, we have the correct distance label. We need only show that $\mathbf{d(u) = \delta(u)}$ to complete the proof (induction on the size of R)

We show that this is true by contradiction.

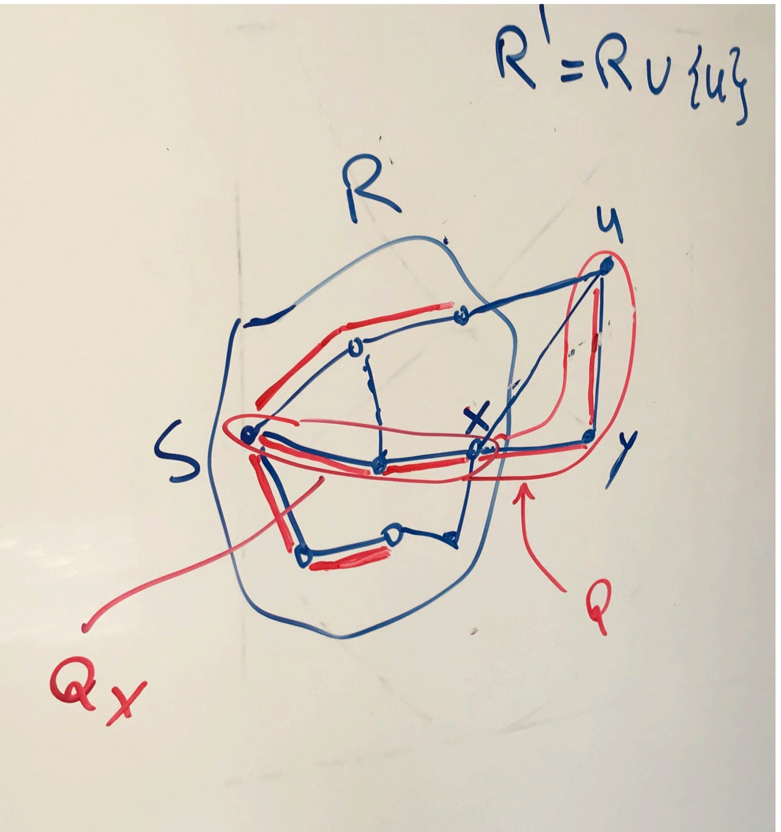
Suppose for a contradiction that the shortest path from s-to-u is **Q** and has length

$$\boldsymbol{\ell(Q) < d(u)}$$

Q starts in $\mathbf{R}'$ and at some vertex leaves $\mathbf{R}'$ (to get to **u**). Let **(x,y)** be the first edge along **Q** that leaves $\mathbf{R}'$.

Let $\mathbf{Q_x}$ be the s-to-x subpath of **Q**. Clearly:

$$\boldsymbol{\ell(Q_x) + \ell(x,y) \leq \ell(Q).}$$



Since $\mathbf{d(x)}$ is the length of the shortest s-to-x path by the I.H., $\mathbf{d(x) = \ell(Q_x)}$, giving us

$$\mathbf{d(x) + \ell(x,y) \le \ell(Q).}$$

Since $\mathbf{y}$ is adjacent to $\mathbf{x}$, $\mathbf{d(y)}$ must have been updated by the algorithm, so

$$\mathbf{d(y) \le d(x) + \ell(x,y).}$$

Finally, since $\mathbf{u}$ was picked by the algorithm, $\mathbf{u}$ must have the smallest distance label:

$$\mathbf{d(u) \le d(y).}$$

Combining these inequalities in reverse order gives us the contradiction that $\mathbf{d(x) < d(x)}$.

Therefore, no such shorter path $\mathbf{Q}$ can exist and so $\mathbf{d(u) = \delta(u)}$.

This lemma shows the algorithm is correct by "applying" the lemma for R = V .

Steps to find the contradiction:

$\mathbf{d(u) \leq d(y)}$

using $d(y) \leq d(x) + \ell(x,y)$ we get

$\mathbf{d(u) \leq d(x) + \ell(x,y)}$

using $d(u) < \ell(Q)$ we get

$\mathbf{\ell(Q) < d(x) + \ell(x,y)}$

using $\ell(Q_x) + \ell(x,y) \leq \ell(Q)$ we get

$\mathbf{\ell(Q_x) + \ell(x,y) < d(x) + \ell(x,y)}$

$\mathbf{\ell(Q_x) < d(x)}$

by the I.H. we have $\ell(Q_x)=d(x)$, hence

$\mathbf{d(x) < d(x)}$ contradiction!