Shortest Path Algorithms

- Shortest path problem
  - Dijkastra Algorithm
  - Belman-Ford algorithm

Shortest Paths Problem

Problem:

Find the minimum-weight path from a given source vertex s to another vertex v in a given weighted directed graph G

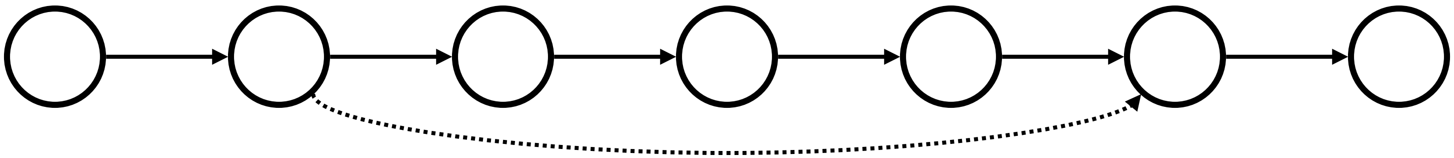
- "Shortest-path" = minimum weight
- Weight of path is sum of edges

Eg. :A railway map

- Variants:
  - **Single source shortest paths problem** → shortest path from s to each vertex x
  - Single destination shortest problems → shortest path to a given destination t from each vertex v
  - Single pair shortest path problem → shortest path from u to v for given vertex u and v
  - All pairs shortest paths problem→ shortets path from u to v for every pair of vertices u and v

Shortest Path Properties

- *optimal substructure*: the shortest path consists of shortest subpaths:

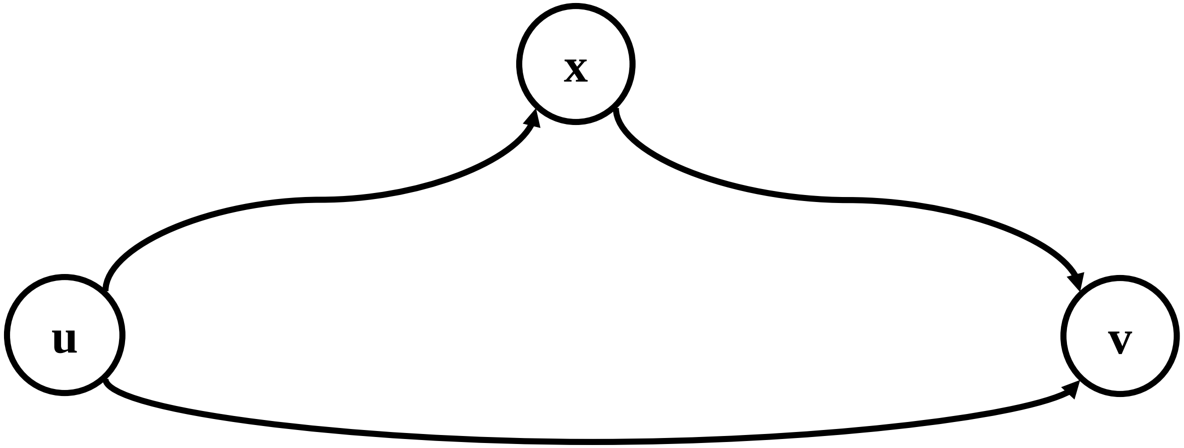


suppose some subpath is not a shortest path

- then there must exist a shorter subpath
- Could substitute the shorter subpath for a shorter path
- But then overall path is not shortest path. Contradiction

Shortest Path Properties

- Let $\delta(u,v) \rightarrow$ the weight of the shortest path from u to v
- Shortest paths satisfy the *triangle inequality*: $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$



This path is no longer than any other path

Initialization

```
INITIALIZE-SINGLE-SOURCE(G, s)
  for each vertex v ∈ V[G]
      do d[v] ← ∞
         π[v] ← NIL
  d[s] ← 0
```

-- Distance from source to node is ∞

-- Predecessor is NIL

Relaxation

- A key technique in shortest path algorithms
  - It lowers the weight upper-bound to a vertex if new edge is lower than current estimate
  - Current estimate *d[v]* is shortest path explored so far from source s to *v*
  - Specifically, ***for all v, maintain upper bound d[v] on*** $\boldsymbol{\delta}$***(s,v)***

$\text{Relax}(u, v, w)$

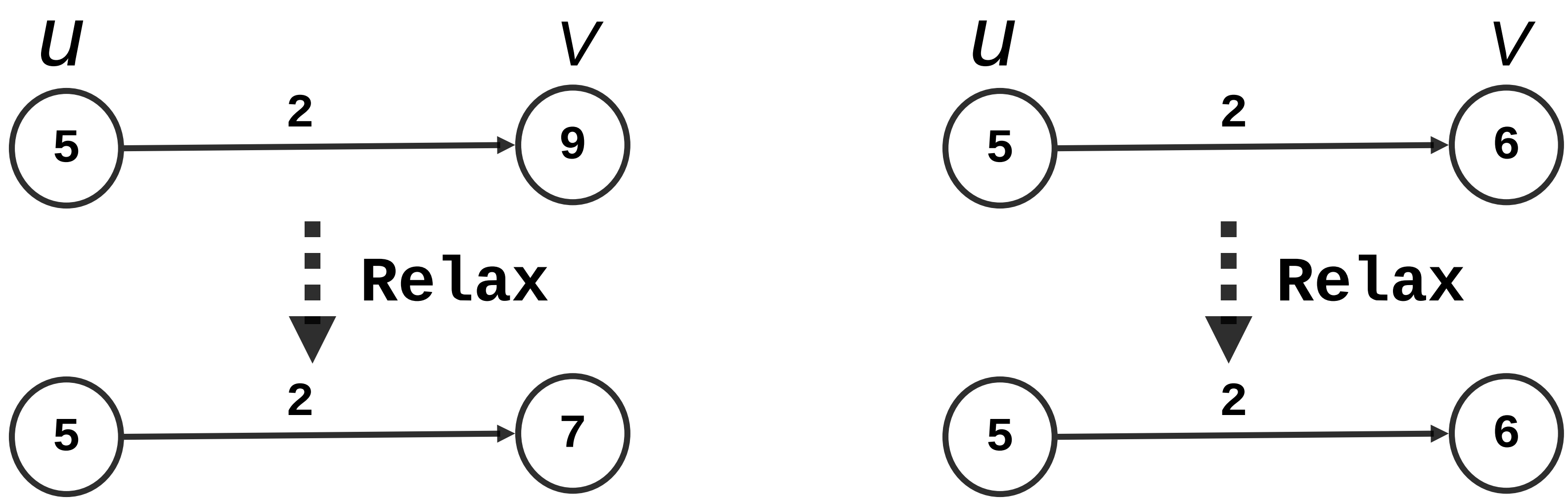
$\textbf{if } d[v] > d[u] + w(u, v)$

$\quad \textbf{then } d[v] \leftarrow d[u] + w(u, v)$

$\qquad \pi[v] \leftarrow u$

Relaxation

- Testing whether we can improve shortest path to v found so far by going through u ,If so update *d[v]* and *π[v]*
- Relaxation may decrease value of shortest path estimate and update ***v'***s predecessor field



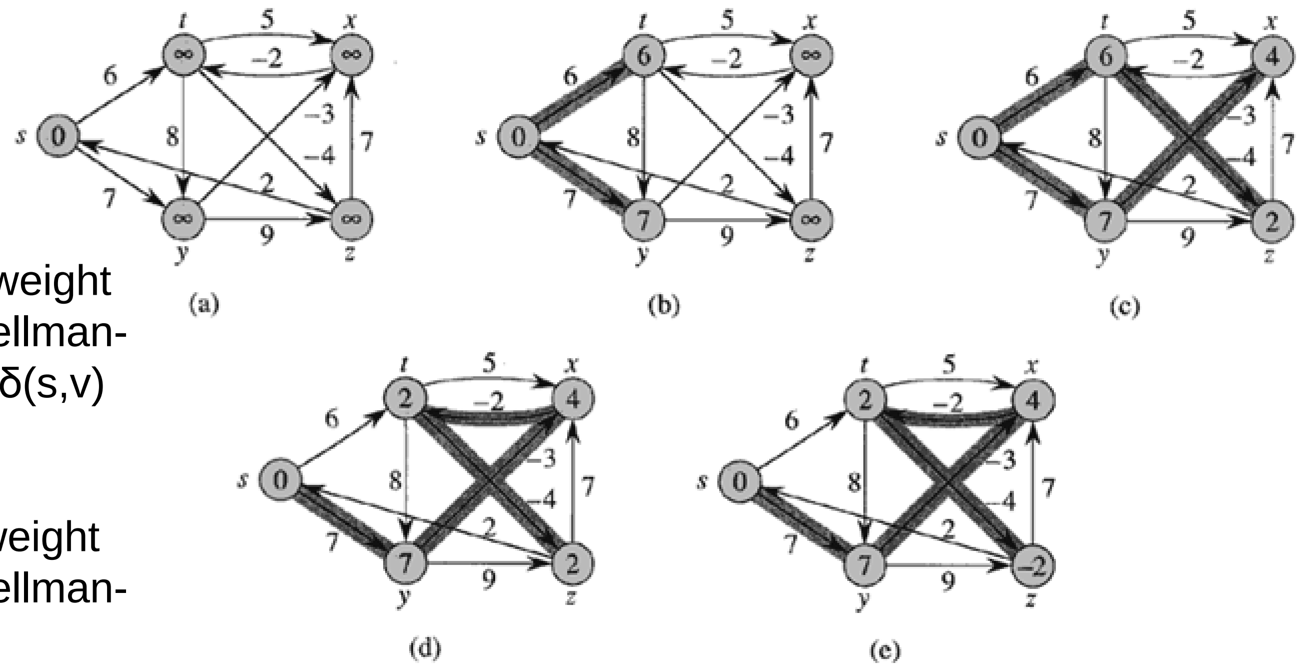
Bellman-Ford Algorithm

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i ← 1 to |V[G]| − 1
      do for each edge (u, v) ∈ E[G]
            do RELAX(u, v, w)
for each edge (u, v) ∈ E[G]
      do if d[v] > d[u] + w(u, v)
            then return FALSE
return TRUE
```

1.If there is no negative edge weight cycle reachable from s then Bellman-Ford returns TRUE and d[v] = δ(s,v) for every vertex v.

2. If there is a negative edge weight cycle reachable from s then Bellman-Ford returns FALSE.

Running time: Θ(|V||E|)



The execution of the Bellman-Ford algorithm. The source is vertex s. The d values are shown within the vertices, and shaded edges indicate predecessor values: if edge (u, v) is shaded, then $\pi[v] = u$. In this particular example, each pass relaxes the edges in the order $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$. **(a)** The situation just before the first pass over the edges. **(b)–(e)** The situation after each successive pass over the edges. The d and π values in part (e) are the final values. The Bellman-Ford algorithm returns TRUE in this example.

Dijkstra's Algorithm

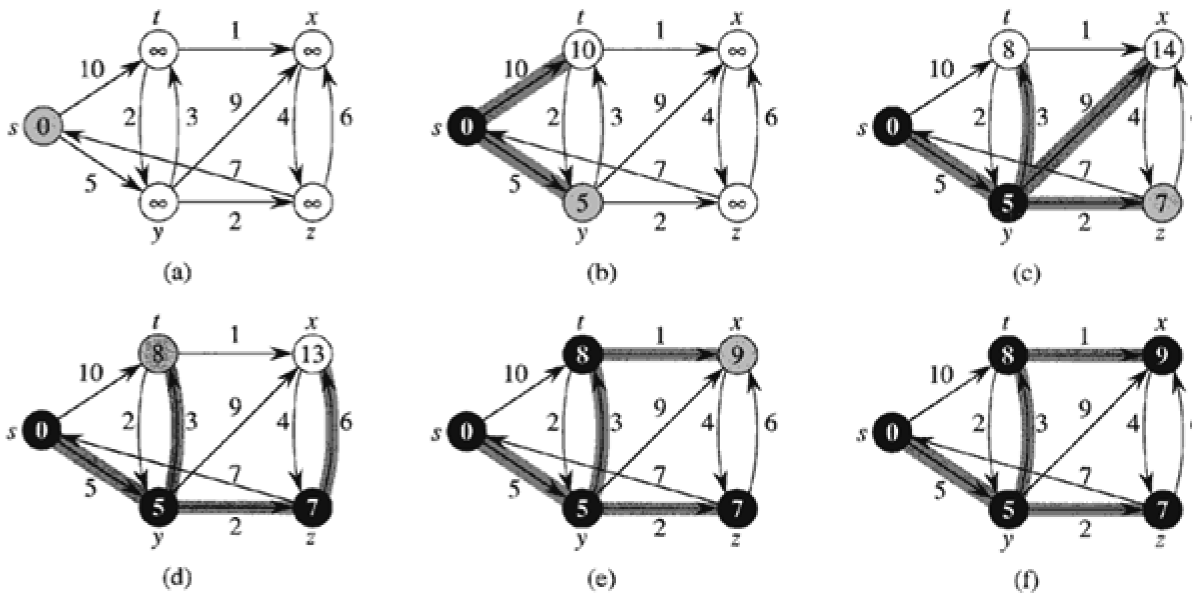
- If no negative edge weights, we can beat BF
- Similar to breadth-first search
  - Grow a tree gradually, advancing from vertices taken from a queue
- Also similar to Prim's algorithm for MST
  - Use a priority queue keyed on d[v]

Dijkstra's Algorithm

// all edge weights are assumed non-negative

DIJKSTRA(G, w, s)

1 INITIALIZE-SINGLE-SOURCE(G, s)
2 $S \leftarrow \emptyset$ /// contains vertices of final shortest-path weights from s
3 $Q \leftarrow V[G]$ // Initialize priority queue Q
4 **while** $Q \neq \emptyset$
5 **do** $u \leftarrow$ EXTRACT-MIN(Q) // Extract new vertex
6 $S \leftarrow S \cup \{u\}$
7 **for** each vertex $v \in Adj[u]$ ///Perform relaxation for each vertex *v* adjacent to u
8 **do** RELAX(u, v, w)



The execution of Dijkstra's algorithm. The source s is the leftmost vertex. The shortest-path estimates are shown within the vertices, and shaded edges indicate predecessor values. Black vertices are in the set S, and white vertices are in the min-priority queue $Q = V - S$. **(a)** The situation just before the first iteration of the **while** loop of lines 4–8. The shaded vertex has the minimum d value and is chosen as vertex u in line 5. **(b)–(f)** The situation after each successive iteration of the **while** loop. The shaded vertex in each part is chosen as vertex u in line 5 of the next iteration. The d and π values shown in part (f) are the final values.