Introduction to Algorithms

Chapter 24 : Single-Source Shortest Paths

Xiang-Yang Li and Haisheng Tan

School of Computer Science and Technology
University of Science and Technology of China (USTC)

Fall Semester 2025

Outline of Topics

shortest-paths problem

In a **shortest-paths problem**, we are given a weighted, directed graph $G = (V, E)$, with weight function $w : E \to \mathbb{R}$ mapping edges to real-valued weights.

The **weight** $w(p)$ of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

We define the **shortest-path weight** $\delta(u, v)$ from u to v by

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there is a path from } u \text{ to } v, \\ \infty & \text{otherwise.} \end{cases}$$

A **shortest path** from vertex u to vertex v is then defined as any path p with weight $w(p) = \delta(u, v)$.

Variants

In this chapter, we shall focus on the **single-source shortest-paths problem**: given a graph $G = (V, E)$, we want to find a shortest path from a given source vertex $s \in V$ to each vertex $v \in V$. The algorithm for the single-source problem can solve many other problems, including the following variants:

- **Single-destination shortest-paths problem**: Find a shortest path to a given destination vertex t from each vertex v.
- **Single-pair shortest-path problem**: Find a shortest path from u to v for given vertices u and v.
- **All-pairs shortest-paths problem**: Find a shortest path from u to v for every pair of vertices u and v. Although we can solve this problem by running a single-source algorithm once from each vertex, we usually can solve it faster.

Optimal substructure of a shortest path

Lemma 24.1 (Subpaths of shortest paths are shortest paths)
Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of p from vertex v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.

Relaxation on an edge (u, v)

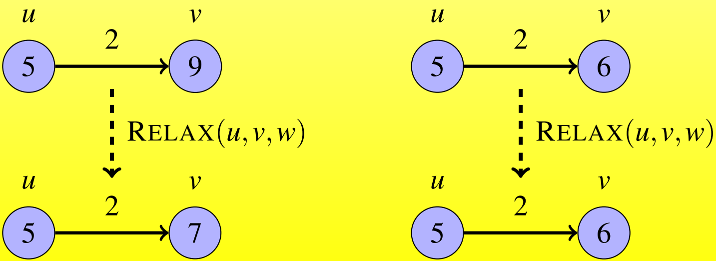
$v.d$: a shortest path (distance) estimation from the source s.
Initially set $v.d = +\infty$ except $s.d = 0$, and $v.\pi = nil$.

RELAX(u, v, w)

if $v.d > u.d + w(u, v)$ **then**

    $v.d = u.d + w(u, v)$

    $v.\pi = u$ // update the predecessor



Properties of shortest paths and relaxation

- **Triangle inequality** (Lemma 24.10) For any edge $(u,v) \in E$, we have $\delta(s,v) \leq \delta(s,u) + w(u,v)$
- **Upper-bound property** (Lemma 24.11) We always have $v.d \geq \delta(s,v)$ for all vertices $v \in V$, and once $v.d$ achieves the value $\delta(s,v)$, it never changes.
- **No-path property** (Corollary 24.12) If there is no path from s to v, then we always have $v.d = \delta(s,v) = \infty$
- **Convergence property** (Lemma 24.14) If $s \rightsquigarrow u \rightarrow v$ is a shortest path in G for some $u, v \in V$, and if $u.d = \delta(s,u)$ at any time prior to relaxing edge (u,v), then $v.d = \delta(s,v)$ at all times afterward.

Properties of shortest paths and relaxation

- **Path-relaxation property** (Lemma 24.15) If $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$. This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of p.
- **Predecessor-subgraph property** (Lemma 24.17) Once $v.d = \delta(s, v)$ for all $v \in V$, the predecessor subgraph is a shortest-paths tree rooted at s.

The Bellman-Ford Algorithm

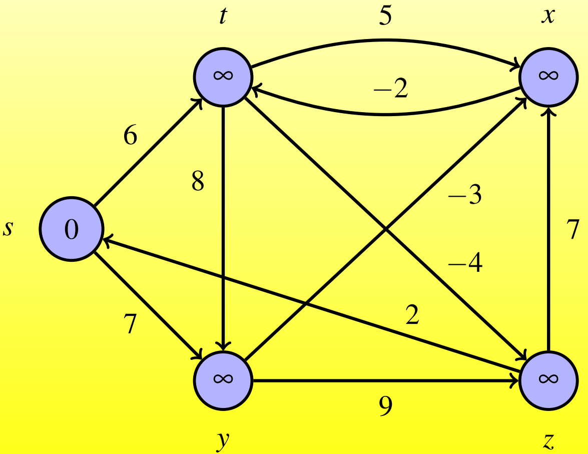
The **Bellman-Ford algorithm** solves the single-source shortest-paths problem in the general case in which edge weights **may be negative**. Given a weighted, directed graph $G = (V, E)$ with source s and weight function $w : E \to \mathbb{R}$, the Bellman-Ford algorithm returns a boolean value indicating **whether or not there is a negative-weight cycle that is reachable from the source**. If there is such a cycle, the algorithm indicates that no solution exists. If there is no such cycle, the algorithm produces the shortest paths and their weights.

BELLMAN-FORD

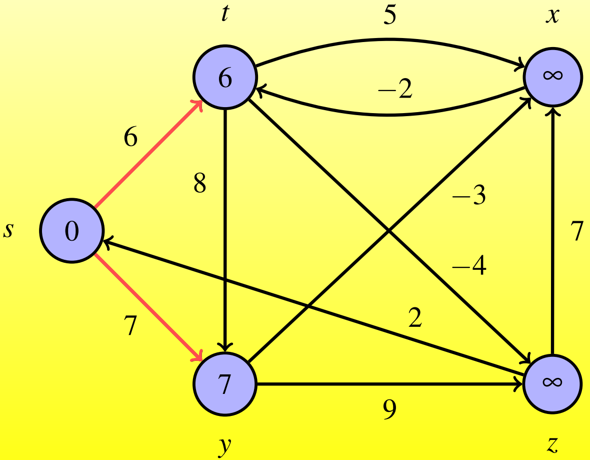
BELLMAN-FORD(G, w, s)

```
for each v ∈ V do
    v.d = ∞;        v.π = nil
s.d = 0
for i = 1 to |G.V| − 1 do
    for each edge (u, v) ∈ G.E do
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E do
    if v.d > u.d + w(u, v) then
        return FALSE
return TRUE
```

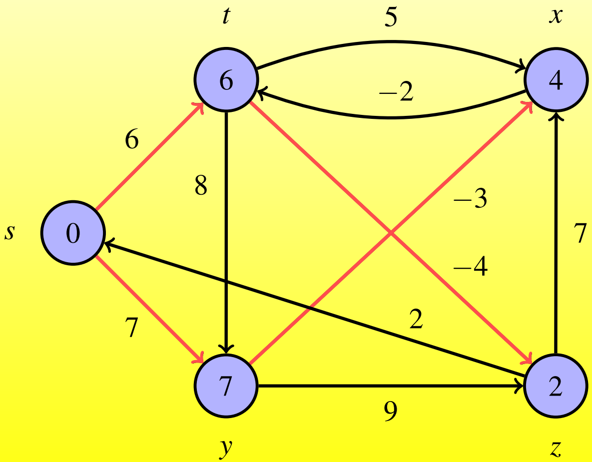
Example



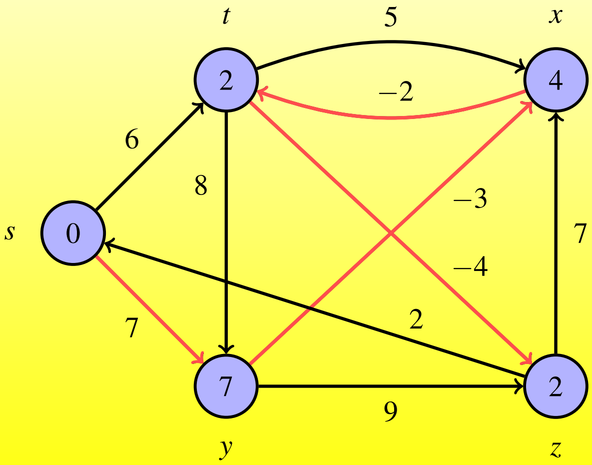
Example



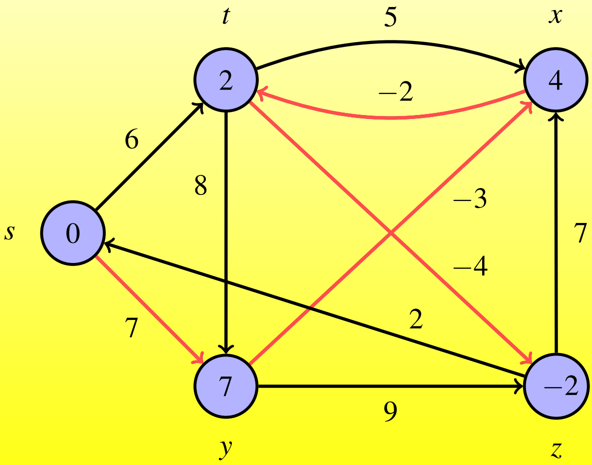
Example



Example



Example



BELLMAN-FORD : Analysis

Correctness? Time Complexity=$O(VE)$

BELLMAN-FORD(G, w, s)

```
for each v ∈ V do        // initialization
    v.d = ∞;    v.π = nil
s.d = 0
for i = 1 to |G.V| − 1 do        // Process each edge |V| − 1 times
    for each edge (u, v) ∈ G.E do        // relax each edge once
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E do // check for a negative-weight cycle
    if v.d > u.d + w(u, v) then
        return FALSE
return TRUE
```

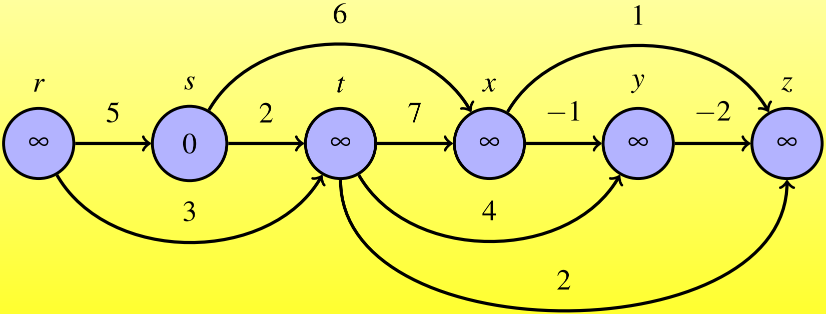
Single-source Shortest Paths in DAGs

By relaxing the edges of a weighted DAG (directed acyclic graph) $G = (V, E)$ according to a topological sort of its vertices, we can compute shortest paths from a single source in $\Theta(V + E)$ time. Shortest paths are always well defined in a DAG, since even if there are negative-weight edges, no negative-weight cycles can exist.

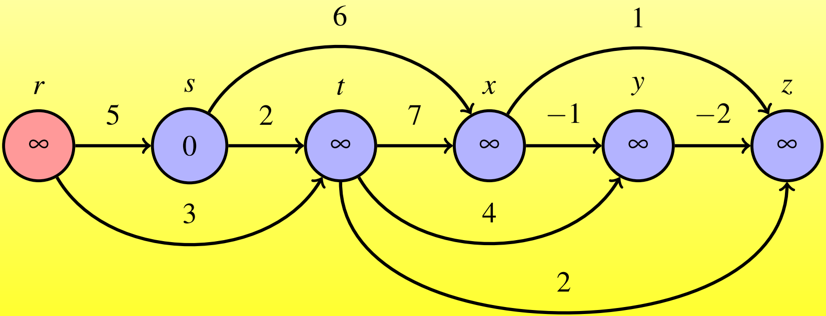
DAG-SHORTEST-PATHS(G, w, s)

```
topologically sort the vertices of G
INITIAL-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order do
    for each vertex v ∈ G.Adj[u] do
        RELAX(u, v, w)
```

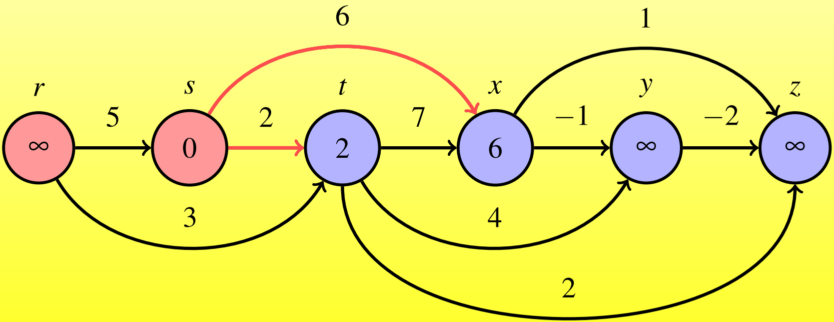
Example



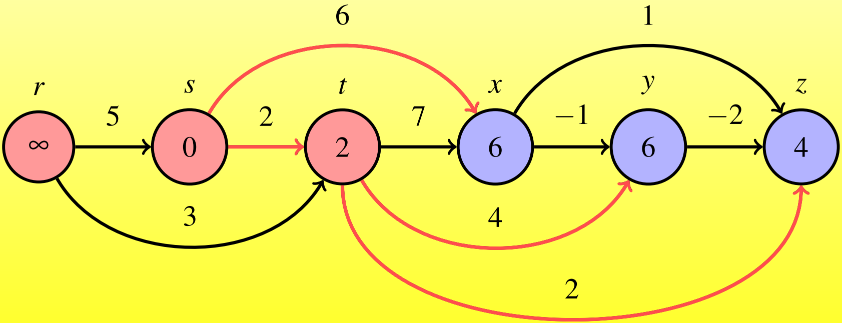
Example



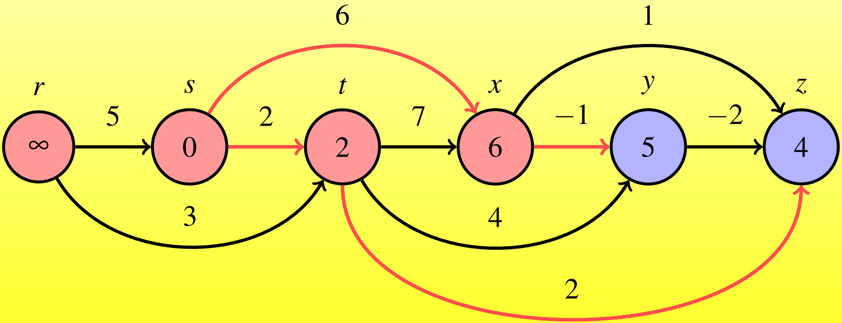
Example



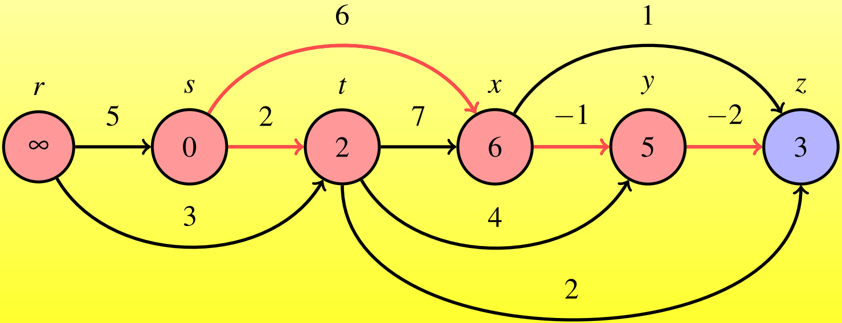
Example



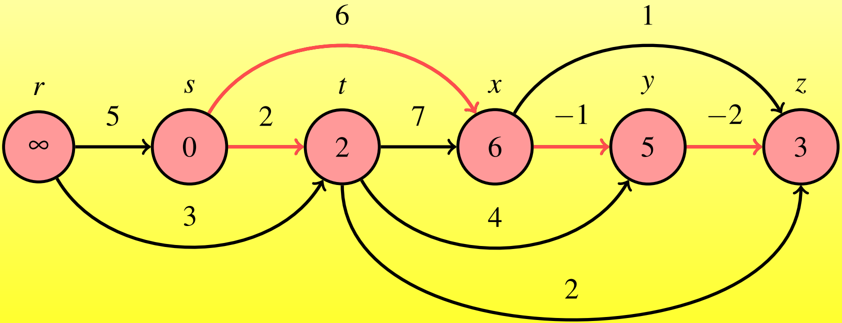
Example



Example



Example



Single-source Shortest Paths in DAGs: Analysis

Correctness?

Time Complexity=$O(V+E)$

DAG-SHORTEST-PATHS(G, w, s)

```
topologically sort the vertices of G
INITIAL-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order do
    for each vertex v ∈ G.Adj[u] do
        RELAX(u, v, w)
```

Dijkstra's Algorithm

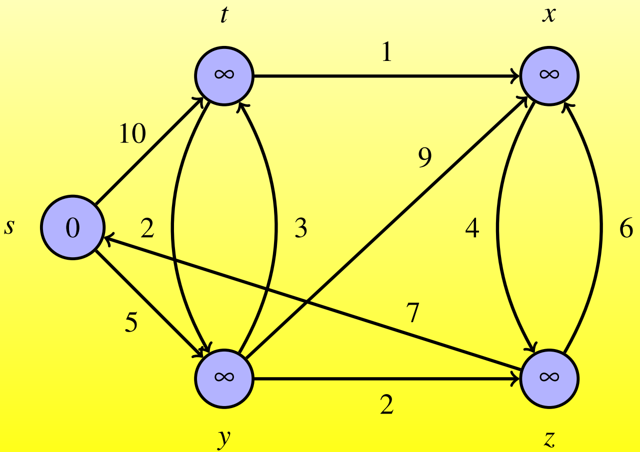
- If **no negative edge weights**, we can beat BF
- Similar to breadth-first search
  - Grow a tree gradually, advancing from vertices taken from a queue
- Also similar to Prim's algorithm for MST
  - Use a priority queue keyed on $d[v]$

Dijkstra's Algorithm

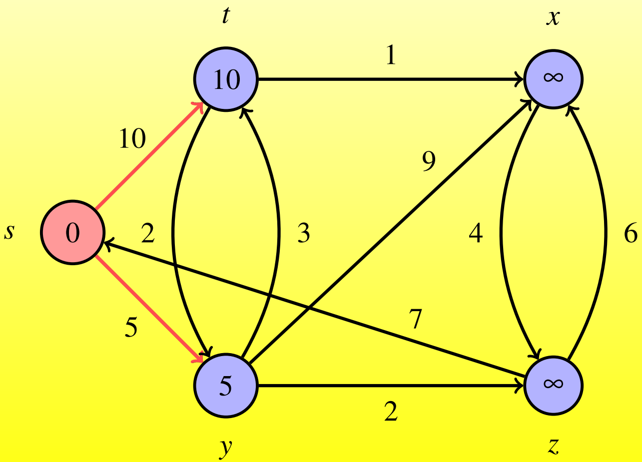
DIJKSTRA(G, w, s)

INITIAL-SINGLE-SOURCE(G, s)
$S = \varnothing$ // nodes with the shortest distance computed
$Q = G.V$
while $Q \neq \varnothing$ **do**
  $u =$ EXTRACT-MIN(Q)
  $S = S \cup \{u\}$
  for each vertex $v \in G.Adj[u]$ **do**
    RELAX(u, v, w)

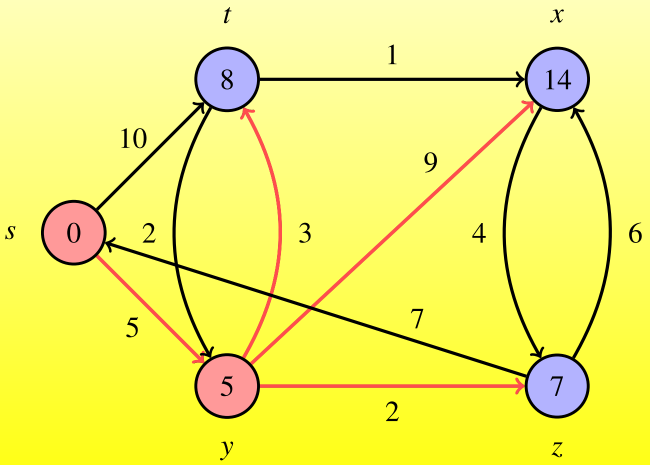
Example



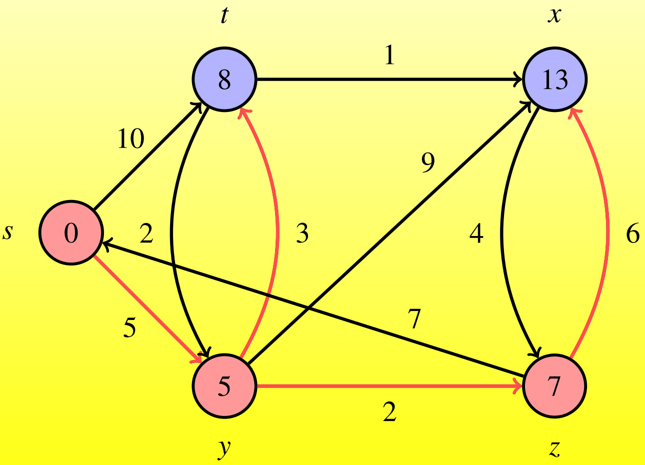
Example



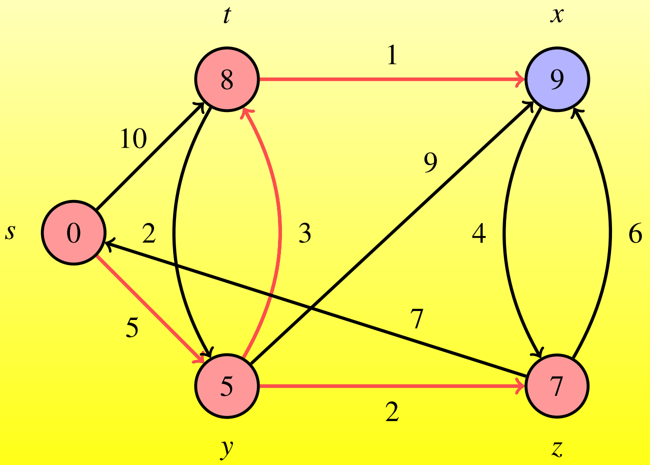
Example



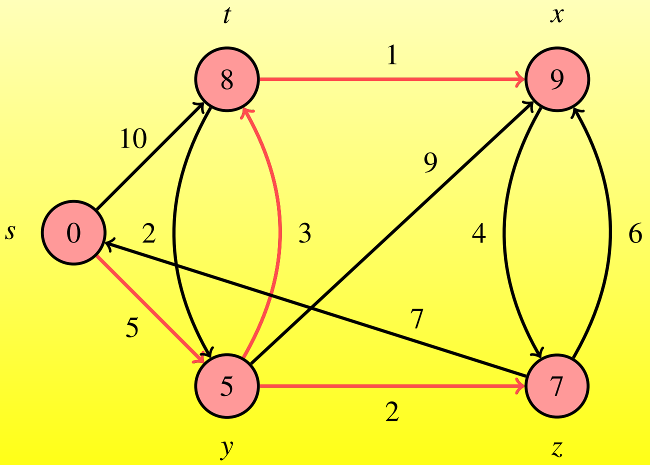
Example



Example



Example



Correctness of Dijkstra's algorithm

Theorem 24.6 (Correctness of Dijkstra's algorithm) Dijkstra's algorithm, run on a weighted, directed graph $G = (V, E)$ with non-negative weight function w and source s, terminates with $u.d = \delta(s, u)$ for all vertices $u \in V$.

Corollary 24.7 If we run Dijkstra's algorithm on a weighted, directed graph $G = (V, E)$ with non-negative weight function w and source s, then at termination, the predecessor subgraph G_π is a shortest-paths tree rooted at s.

Dijkstra's Algorithm - Time Complexity

Time: $O(E + V \log V)$, by implementing the min-priority queue with a Fibonacci heap.

DIJKSTRA(G, w, s)

1: INITIAL-SINGLE-SOURCE(G, s)
2: $S = \varnothing$
3: $Q = G.V$ // $|V|$ INSERT (Q)
4: **while** $Q \neq \varnothing$ **do**
5: $u =$ EXTRACT-MIN(Q) // $|V|$ EXTRACT-MIN(Q)
6: $S = S \cup \{u\}$
7: **for** each vertex $v \in G.Adj[u]$ **do**
8: RELAX(u, v, w) // $|E|$ DECREASE-KEY(Q)