

Shortest Paths

- Input: weighted, directed graph $G = (V, E)$, with weight function $w : E \rightarrow \mathbb{R}$.
- The **weight** of path $p = \langle v_0, v_1, \dots, v_k \rangle$ is the sum of the weights of its constituent edges:

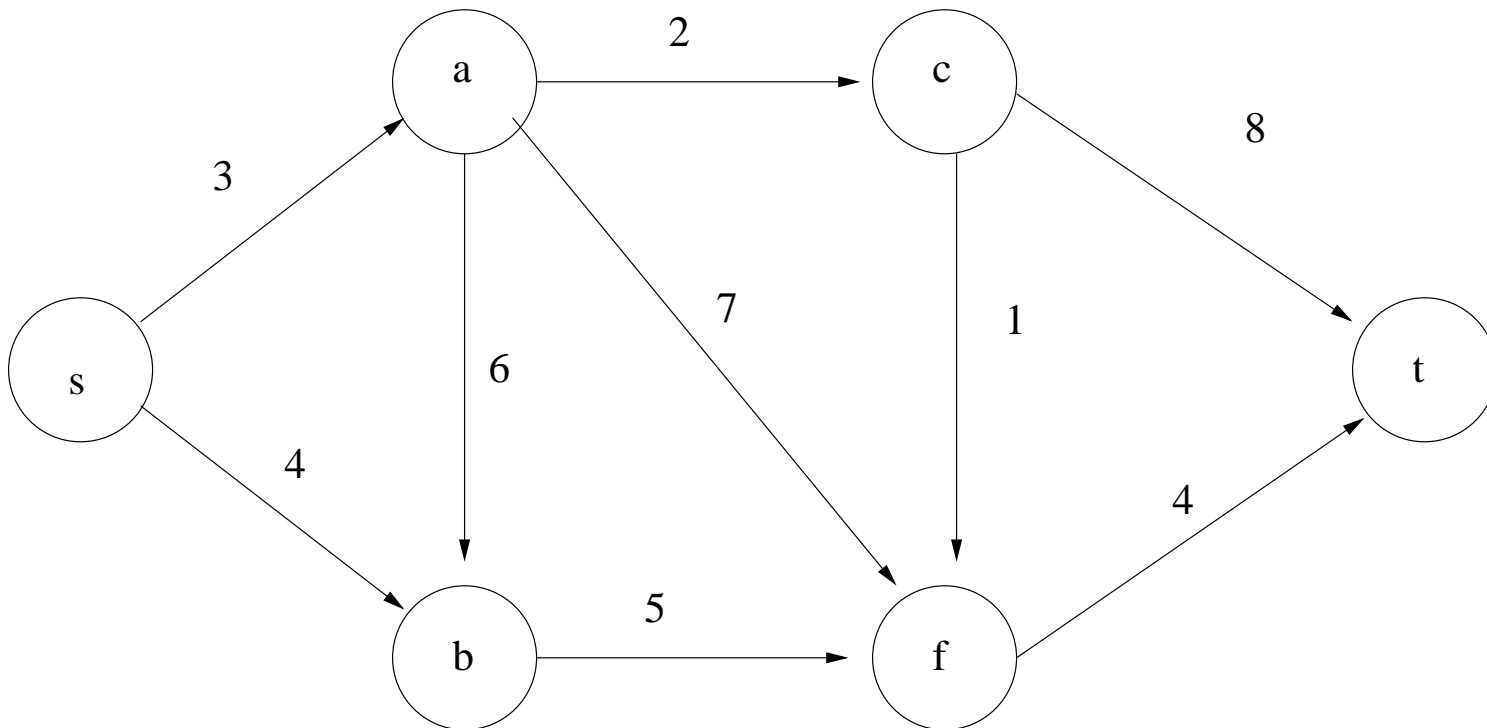
$$w(p) = \sum_{i=1}^k w(v_{i-1}, v_i) .$$

- The **shortest-path weight** from u to v is

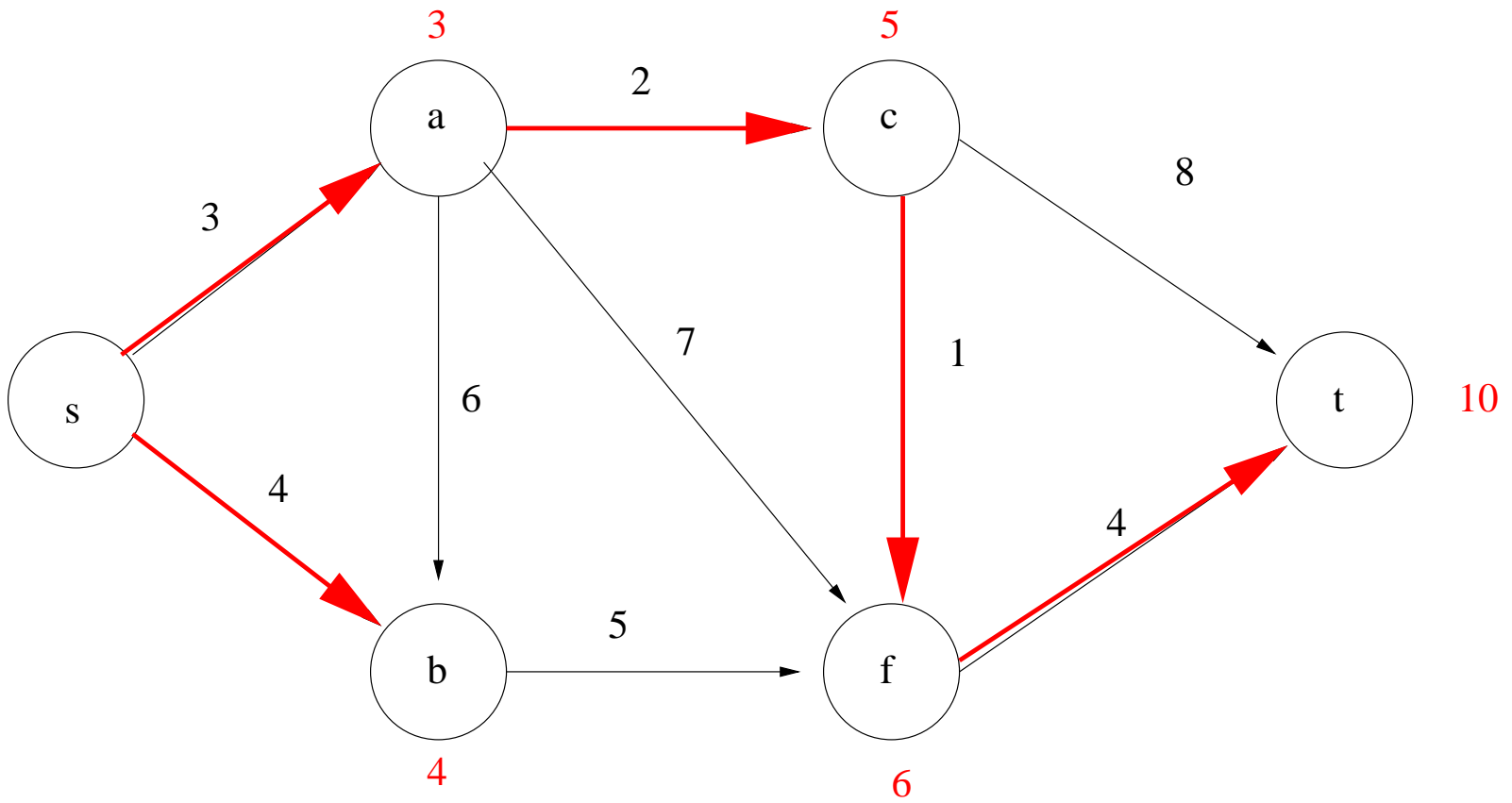
$$\delta(u, v) = \begin{cases} \min\{w(p)\} & \text{if there is a path } p \text{ from } u \text{ to } v , \\ \infty & \text{otherwise .} \end{cases}$$

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

Example



Solution



Shortest Paths

Shortest Path Variants

- Single Source-Single Sink
- Single Source (all destinations from a source s)
- All Pairs

Defs:

- Let $\delta(v)$ be the real shortest path distance from s to v
- Let $d(v)$ be a value computed by an algorithm

Edge Weights

- All non-negative
- Arbitrary

Note: Must have no negative cost cycles

Single Source Shortest Paths

Key Property: Subpaths of shortest paths are shortest paths Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \rightarrow \mathbb{R}$, let $p = \langle v_1, v_2, \dots, v_k \rangle$ be a shortest path from vertex v_1 to vertex v_k and, for any i and j such that $1 \leq i \leq j \leq k$, let $p_{ij} = \langle v_i, v_{i+1}, \dots, v_j \rangle$ 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 .

Note: this is optimal substructure

Corollary 1 For all edges $(u, v) \in E$,

$$\delta(v) \leq \delta(u) + w(u, v)$$

Corollary 2 Shortest paths follow a tree of edges for which

$$\delta(v) = \delta(u) + w(u, v)$$

More precisely, any edge in a shortest path must satisfy

$$\delta(v) = \delta(u) + w(u, v)$$

Relaxation

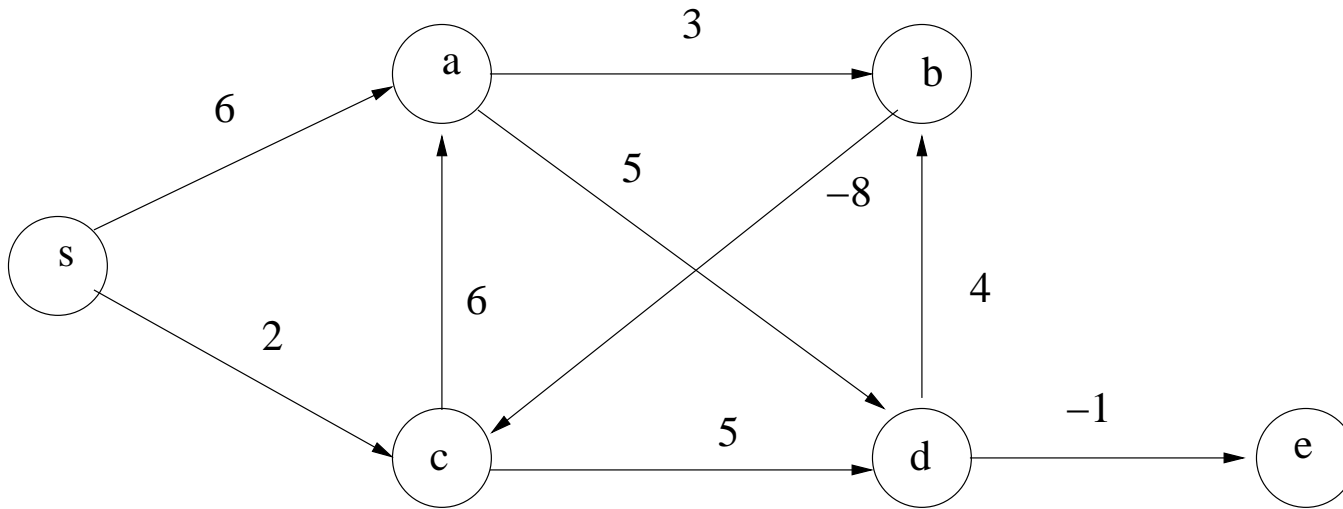
Relax(u, v, w)

- 1 **if** $d[v] > d[u] + w(u, v)$
- 2 **then** $d[v] \leftarrow d[u] + w(u, v)$
- 3 $\pi[v] \leftarrow u$ (**keep track of actual path**)

Lemma: Assume that we initialize all $d(v)$ to ∞ , $d(s) = 0$ and execute a series of Relax operations. Then for all v , $d(v) \geq \delta(v)$.

Lemma: Let $P = e_1, \dots, e_k$ be a shortest path from s to v . After initialization, suppose that we relax the edges of P in order (but not necessarily consecutively). Then $d(v) = \delta(v)$.

Example



Algorithms

Goal of an algorithm: Relax the edges in a shortest path in order (but not necessarily consecutively).

Algorithms

Goal of an algorithm: Relax the edges in a shortest path in order (but not necessarily consecutively).

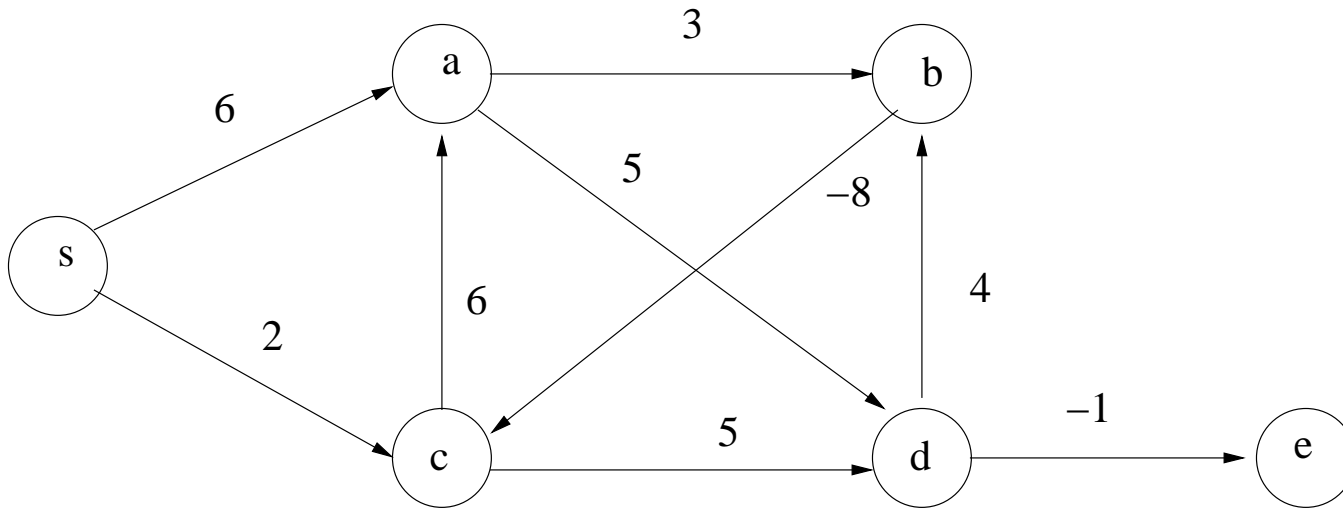
Bellman-Ford(G, w, s)

```
1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2  for  $i \leftarrow 1$  to  $|V[G]| - 1$ 
3      do for each edge  $(u, v) \in E[G]$ 
4          do RELAX( $u, v, w$ )
5  for each edge  $(u, v) \in E[G]$ 
6      do if  $d[v] > d[u] + w(u, v)$ 
7          then return FALSE
8  return TRUE
```

Initialize – Single – Source(G, s)

```
1  for each vertex  $v \in V[G]$ 
2      do  $d[v] \leftarrow \infty$ 
3      do  $\pi[v] \leftarrow \text{NIL}$ 
4   $d[s] \leftarrow 0$ 
```

Example



Correctness of Bellman Ford

- Every shortest path must be relaxed in order
- If there are negative weight cycles, the algorithm will return false

Running Time $O(VE)$

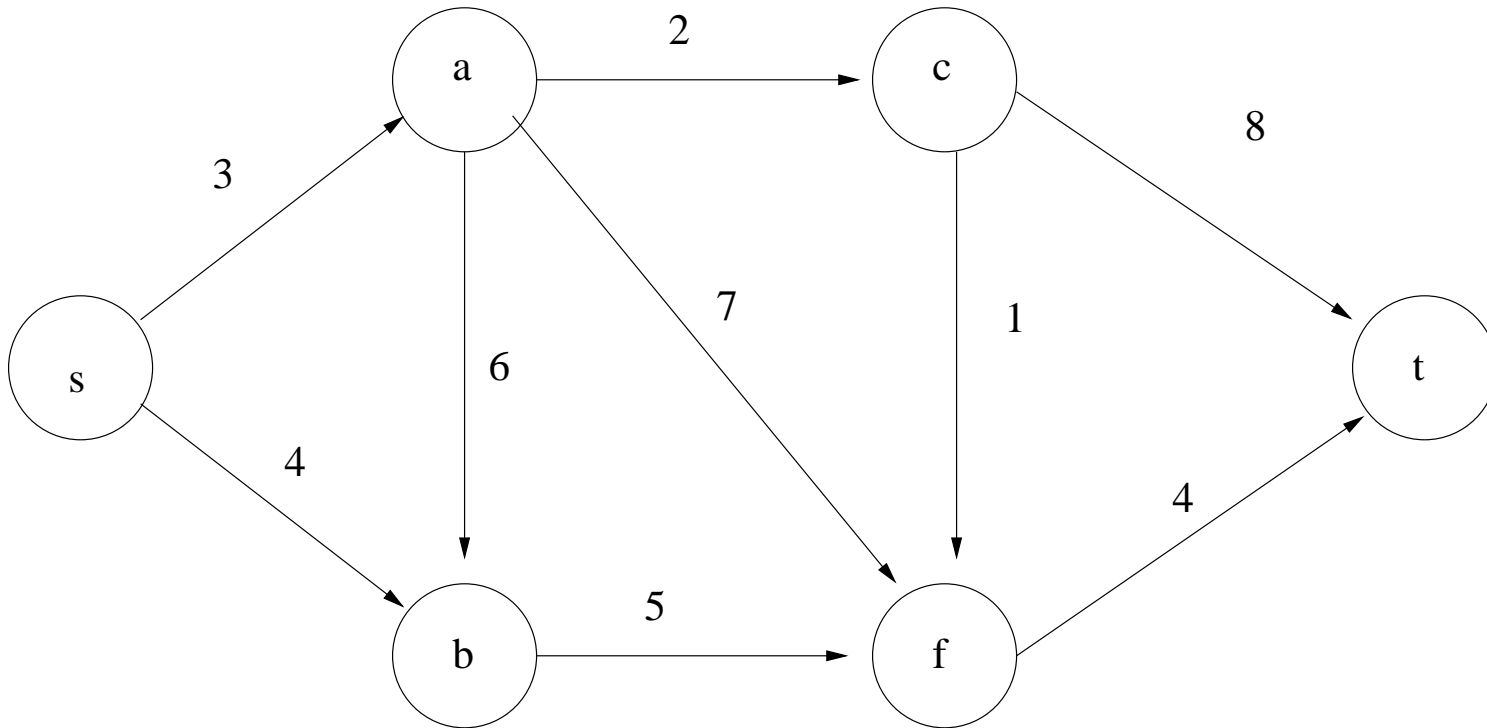
All edges non-negative

- Dijkstra's algorithm, a greedy algorithm
- Similar in spirit to Prim's algorithm
- Idea: Run a discrete event simulation of breadth-first-search. Figure out how to implement it efficiently
- Can relax edges out of each vertex exactly once.

Dijkstra(G, w, s)

```
1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2   $S \leftarrow \emptyset$ 
3   $Q \leftarrow V[G] \triangleright$  This line does  $V$  INSERTs
4  while  $Q \neq \emptyset$ 
5      do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6           $S \leftarrow S \cup \{u\}$ 
7          for each vertex  $v \in \text{Adj}[u]$ 
8              do RELAX( $u, v, w$ )  $\triangleright$  This line does a DECREASE-KEY
```

Example



Correctness

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

Claim to Prove: When v is put in S , $d(v) = \delta(v)$.

Proof

Claim to Prove: When v is put in S , $d(v) = \delta(v)$.

Proof

- $d(v) \geq \delta(v)$ because any algorithm that does a sequence of Relax calls has this property.
- Assume f.p.o.c. that $d(v) > \delta(v)$, and that v is the first such vertex that is permanently labelled that has this property.
- Consider the state of the world just before v is put in S
 - shortest path from s to v goes through an edge (x, y) where $x \in S$ and $y \notin S$ (it is possible that $y = v$ and/or $x = s$).
 - $d(x) = \delta(x)$, because $x \in S$
 - $d(y) = \delta(y)$ because (x, y) was relaxed when x was put in S .
- Putting these together with $\delta(y) \leq \delta(v)$ because y is before v on a shortest path, we have

$$d(y) = \delta(y) \leq \delta(v) < d(v)$$

.

- $d(y) < d(v)$, so the algorithm would have chosen to permanently label y and not v , which is a contradiction.

Running Time

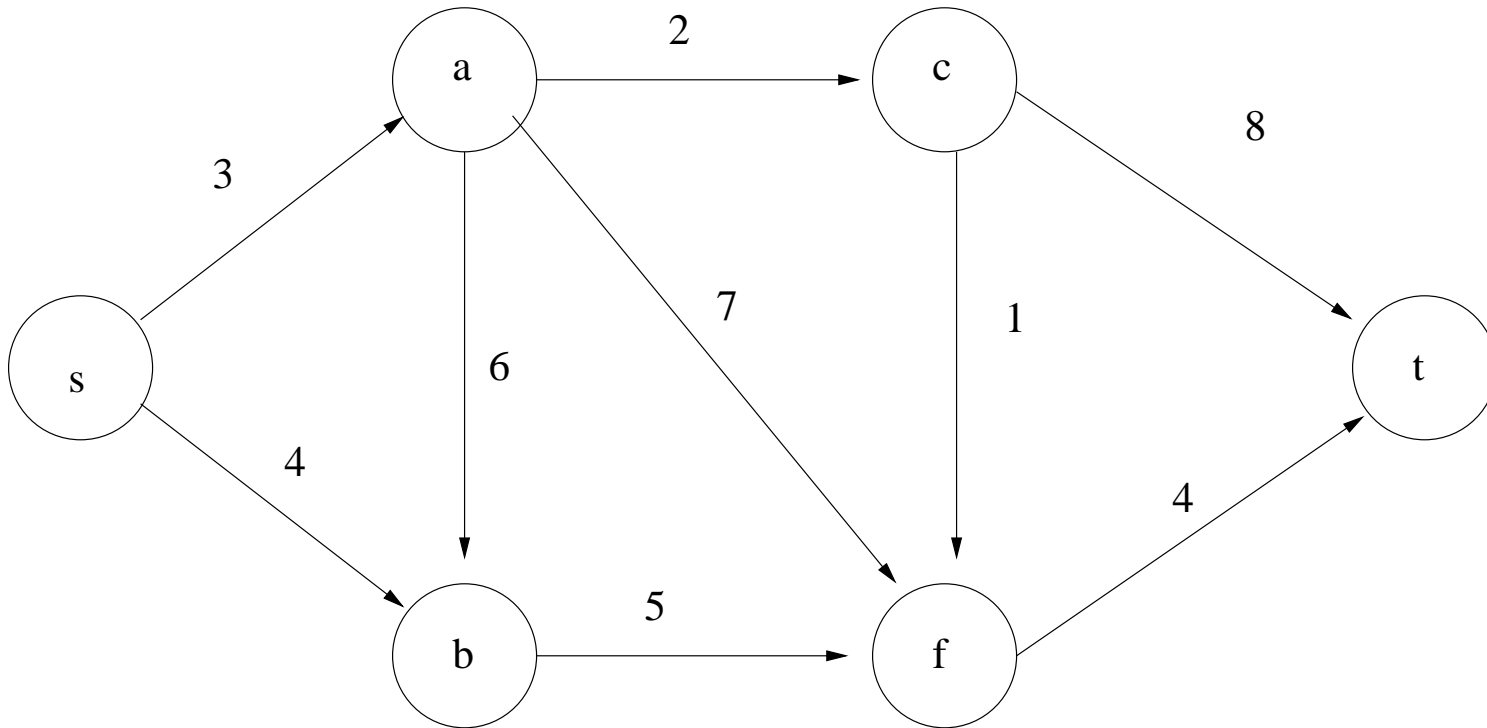
- E decrease keys and V delete-min's
- $O(E \log V)$ using a heap
- $O(E + V \log V)$ using a Fibonacci heap

Shortest Path in a DAG

Dag-Shortest-Paths(G, w, s)

- 1 topologically sort the vertices of G
- 2 INITIALIZE-SINGLE-SOURCE'(G, s)
- 3 for each u taken in topological order
- 4 do for each $v \in Adj[u]$
- 5 do RELAX(u, v, w)

Example



Correctness and Running Time

Correctness If a weighted, directed graph $G = (V, E)$ has source vertex s and no cycles, then at the termination of the DAG-SHORTEST-PATHS procedure, $d[v] = \delta(s, v)$ for all vertices $v \in V$, and the predecessor subgraph G_π is a shortest-paths tree.

Running Time

- Topological sort is linear time
- Each edge is relaxed once
- No additional data structure overhead

$O(V + E)$ time.