Analysis of Algorithms

Shortest Path Algorithms

Single-Source Shortest Paths

- Given a weighted directed graph $G = (V, E)$ with a weight function w: $E \to \mathbf{R}$, we define the shortest-path weight from u to v by
- $\delta(u,v) = \min\{w(p) : u \to v\}$ if there is a path from u to v
- $\delta(u,v) = \infty$ otherwise
- The weight $w(p)$ of a path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges.

Variants

- *Single-destination shortest-paths problem*: Find a shortest path to a given destination vertex t from every vertex $v \in V$. By reversing the direction of each edge in the graph, we can reduce this problem to single source problem.
- *Single-pair shortest-path problem*: Find a shortest path from u to v for given vertices u and v. If we solve the single-source shortest paths problem, we also solve this problem.
- *All-pairs shortest-paths problem*: Find a shortest path from u to v for every pair of vertices u and v. This can be solved by the single-source problem run for each vertex.

Well Definedness

- If there is a negative-weight cycle reachable from s, the shortest path weights are not well defined: no path from s to the vertex on the cycle can be a shortest path (a lesser-weight path can always be found).
- If there is a negative-weight cycle on some path from s to v, we define
- $\delta(s,v) = -\infty$

Example

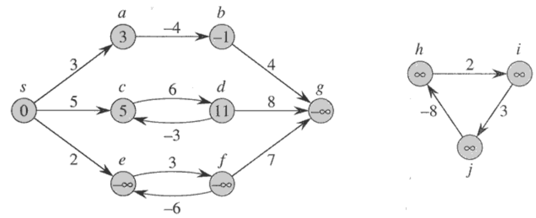


Figure 24.1 Negative edge weights in a directed graph. Shown within each vertex is its shortest-path weight from source s. Because vertices e and f form a negative-weight cycle reachable from s, they have shortest-path weights of $-\infty$. Because vertex g is reachable from a vertex whose shortest-path weight is $-\infty$, it, too, has a shortest-path weight of $-\infty$. Vertices such as h, i, and j are not reachable from s, and so their shortest-path weights are ∞, even though they lie on a negative-weight cycle.

More Examples

- Shortest paths are not necessarily unique

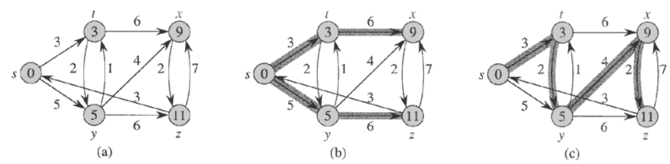
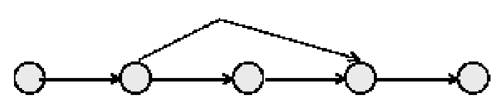


Figure 24.2 **(a)** A weighted, directed graph with shortest-path weights from source s. **(b)** The shaded edges form a shortest-paths tree rooted at the source s. **(c)** Another shortest-paths tree with the same root.

Optimal Substructure

Lemma 24.1: Let p = ‹v1, v2, …, vk› be a SP from v_1 to v_k. Then, $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ is a SP from v_i to v_j, where $1 \le i \le j \le k$.



Relaxation

- Technique used by shortest path algorithms
- For each vertex $v \in V$, we maintain an attribute $d[v]$, which is an **upper-bound** on the weight of a shortest path from source s to v. Attribute $d[v]$ is a **shortest-path estimate**.

```
Initialize(G, s)
  for each v ∈ V[G]
     d[v] ← ∞
     π[v] ← NIL
  d[s] := 0
```

Relaxation

Relax(u,v,w)

if $d[v] > d[u] + w(u,v)$

then $d[v] \leftarrow d[u] + w(u,v)$

$\pi(v) \leftarrow u$

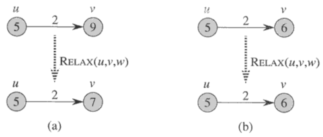


Figure 24.3 Relaxation of an edge (u, v) with weight $w(u, v) = 2$. The shortest-path estimate of each vertex is shown within the vertex. **(a)** Because $d[v] > d[u] + w(u, v)$ prior to relaxation, the value of $d[v]$ decreases. **(b)** Here, $d[v] \leq d[u] + w(u, v)$ before the relaxation step, and so $d[v]$ is unchanged by relaxation.

The Bellman-Ford Algorithm

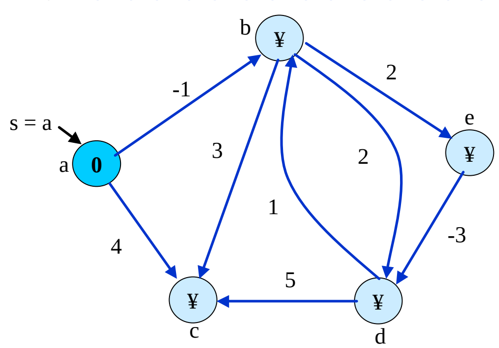
- Solves the single-source shortest-paths problem in the more general case, in which the edges can be negative.
  - Returns a boolean value indicating whether a negative-weight cycle is reachable from the source.
  - If there is no such cycle, the algorithm produces the shortest paths and their weights

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
```


Example

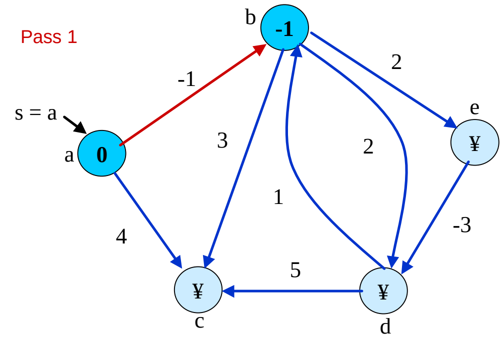
Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

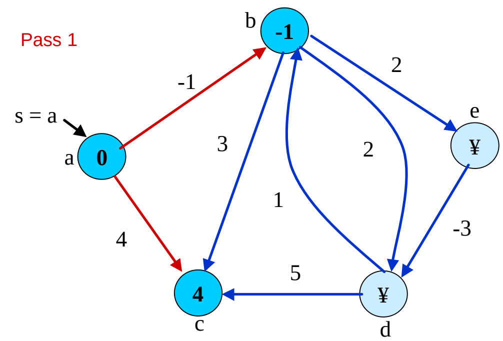
Pass 1



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

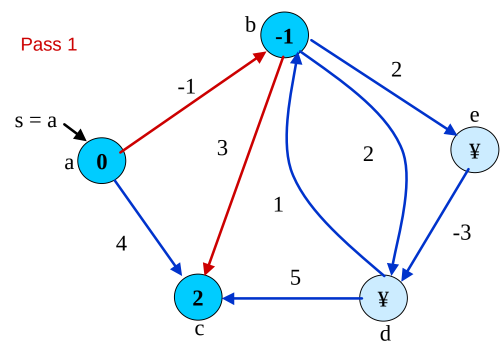
Pass 1



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

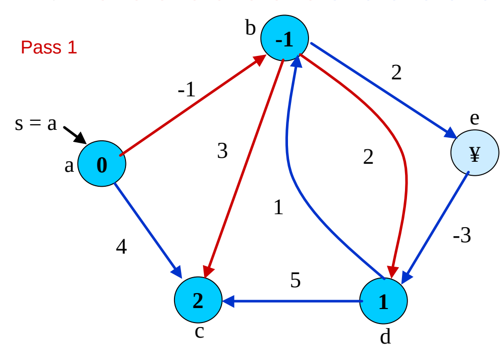
Pass 1



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

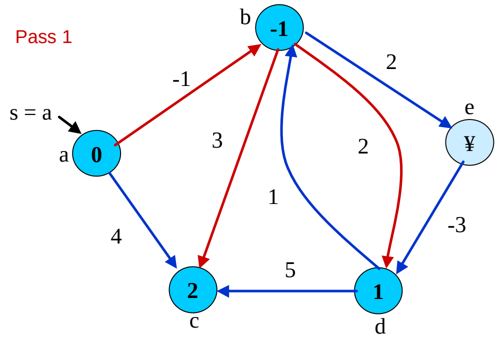
Pass 1



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

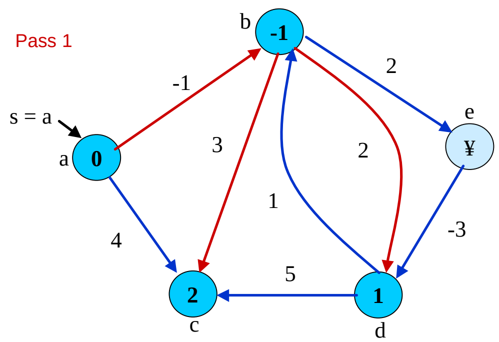
Pass 1



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

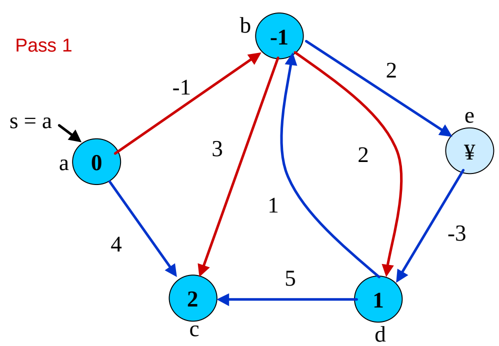
Pass 1



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

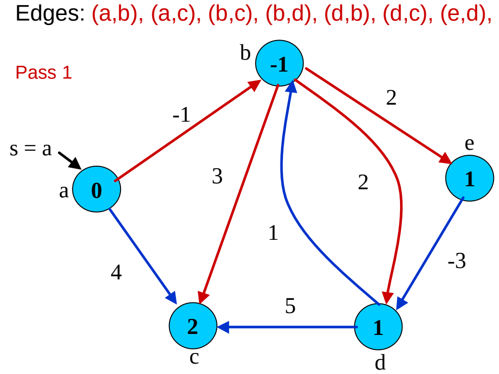
Pass 1



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

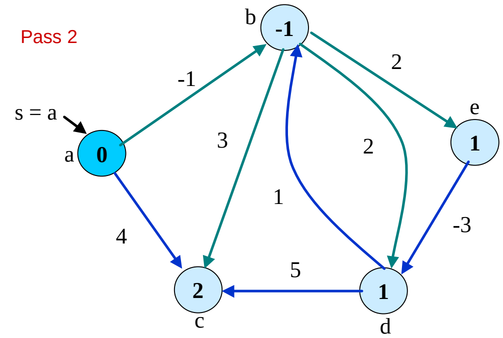
Pass 1



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

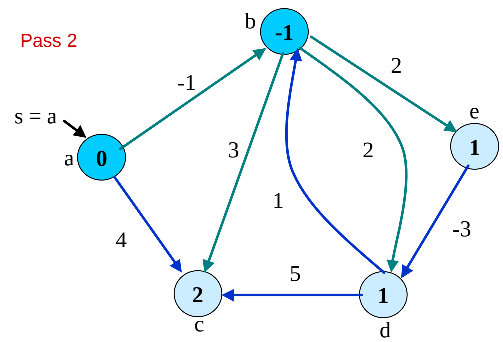
Pass 2



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

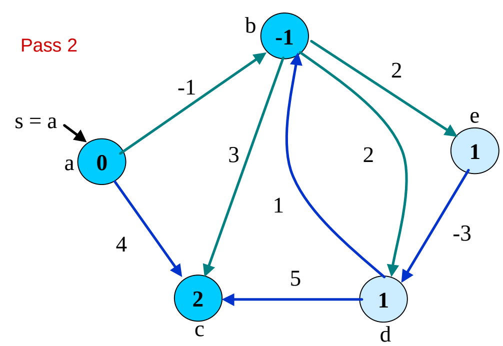
Pass 2



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

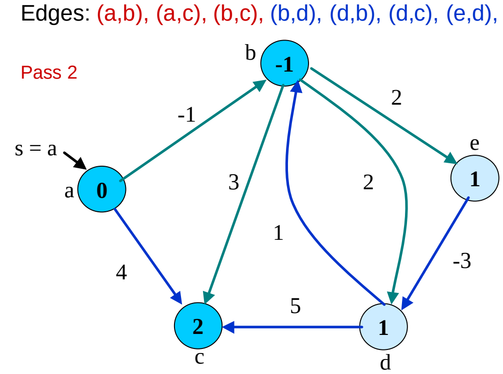
Pass 2



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

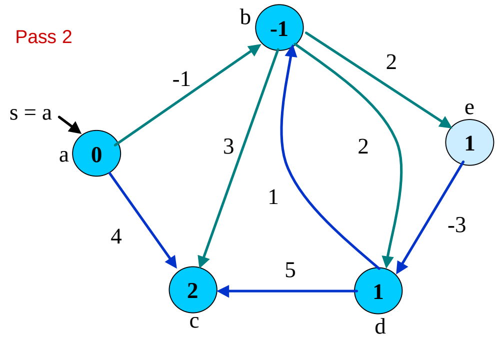
Pass 2



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

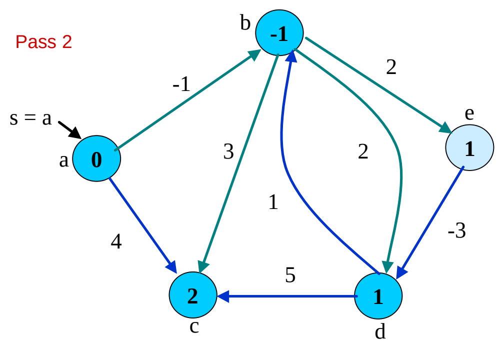
Pass 2



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

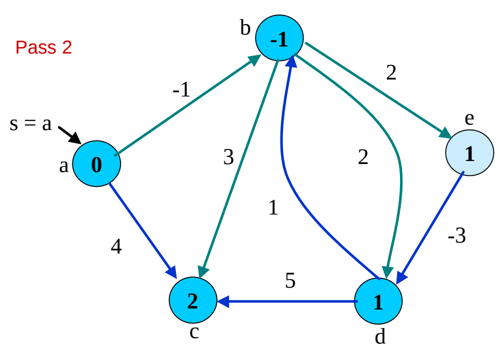
Pass 2



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

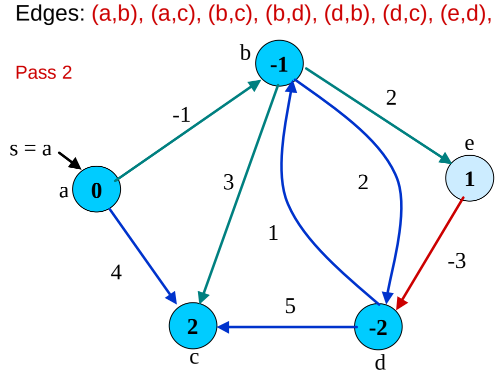
Pass 2



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

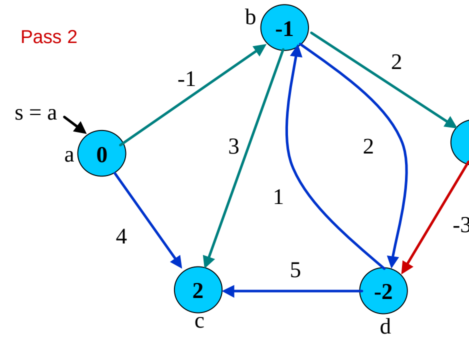
Pass 2



Example

Edges: (a,b), (a,c), (b,c), (b,d), (d,b), (d,c), (e,d), (b,e)

Pass 2



Time Complexity is O(VE).

Example

- Although this example will iterate for 4 passes (|V|-1passes), but the algorithm has converged in just two passes, therefore no change in 3rd and 4th pass.
- Iterations (relaxation steps) can be safely concluded in less |V|-1passes whenever no path (to a node) is improved during one pass.
- The algorithm terminates in |*V*|-1 passes because
  - if *G* has no negative weight cycles, every shortest path is **simple** (no cycles),
  - and the longest simple path can have at most |*V*|-1 edges.

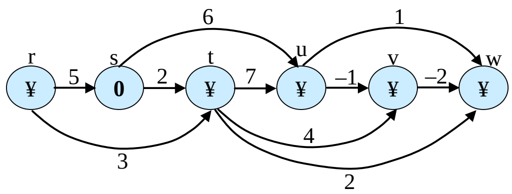
Single-source shortest paths in DAGs

- Shortest paths are always well defined in a dag, since even if there are negative weight edges, no negative weight cycles can exist.
- If there is a path from vertex u to v, then u precedes v in the topological order.
- We make just one pass over the vertices in the topologically sorted order. As we proceed each vertex, we relax each edge that leaves the vertex.

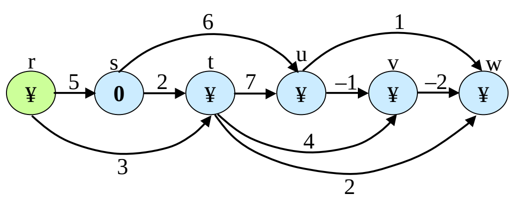
Single-source shortest paths in DAGs

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

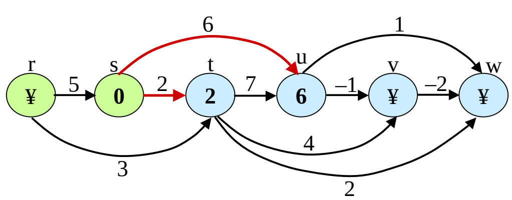
Example



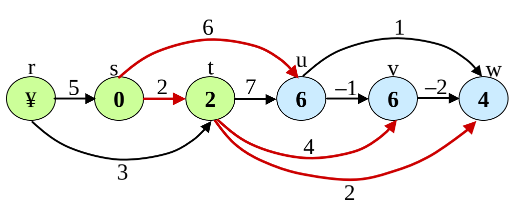
Example



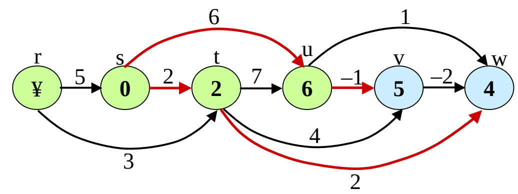
Example



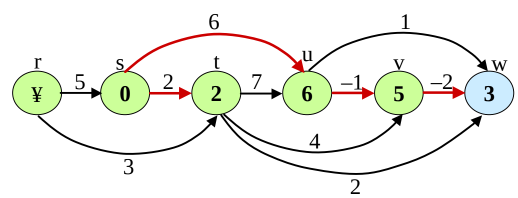
Example



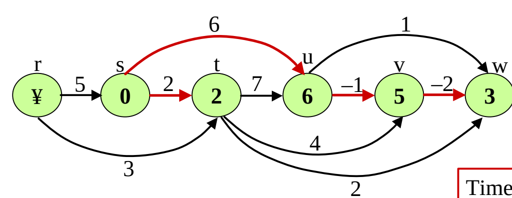
Example



Example



Example



Time Complexity is O(V+E).

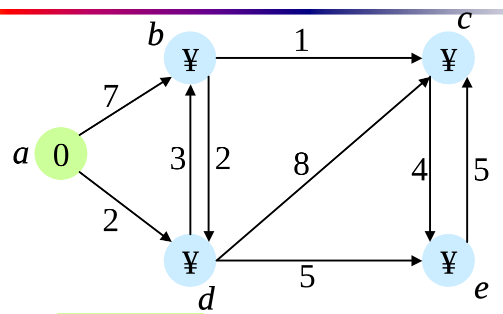
Dijkstra's Algorithm

- Maintains a subset of vertices, $S \subseteq V$, for which we know their true distance $d[u]=d(s,u)$. Initially $S = \emptyset$ and we set $d[s]=0$ and all others to ∞. One by one we select vertices from $V - S$ to add to S.
- For each vertex $u \in V\text{-}S$, we have computed a distance estimate $d[u]$. The greedy approach is to take the vertex for which $d[u]$ is minimum, i.e. take unprocessed vertex that is closest to s.
- We store the vertices of $V - S$ in a priority queue (heap), where the key value of each vertex u is $d[u]$. All operations can be done in O(lg n) time.

Dijkstra(*G*, *w*, *s*)

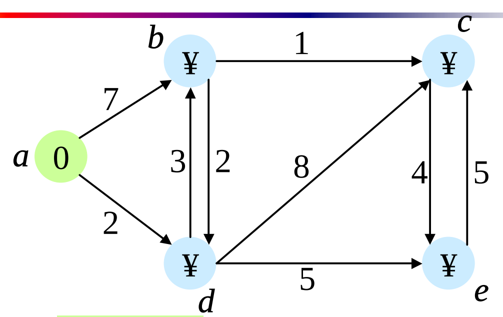
```
1.  Q ← V[G] and S ← ∅
2.  for each vertex u ∈ Q                        // initialization: O(V) time
3.       do d[u] ← ∞ and p[u] ← NIL
4.  d[s] ← 0                                     // start at the source
5.  p[s] ← NIL                                   // set parent of s to be NIL
6.  while Q ≠ ∅                                  // till all vertices processed
7.       do u ← Extract-Min(Q)                   // select closest to s
              S ← S ∪ {u}
8.          for each v ∈ adj[u]
9.               do if v ∈ Q and (d[u] + w(u,v) < d[v])
10.                      then p[v] ← u
11.                           d[v] ← d[u] + w(u,v)   //  Relax (u,v)
12.                           decrease_Key(Q, v, d[v])
```


Example: Dijkstra's Algorithm



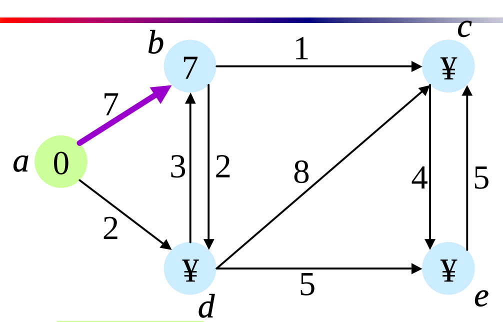
Green: in *S*

Example: Dijkstra's Algorithm



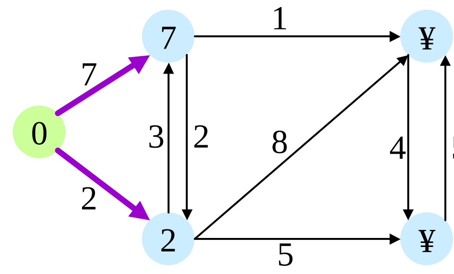
Green: in *S*

Example: Dijkstra's Algorithm



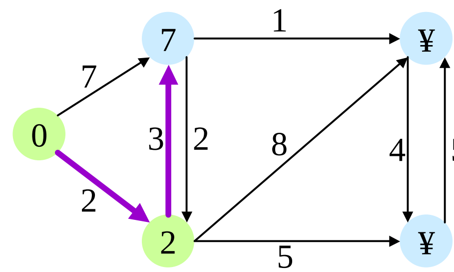
Green: in *S*

Example: Dijkstra's Algorithm



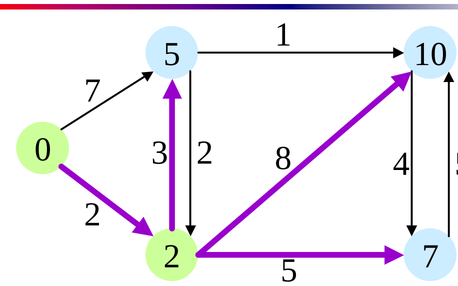
Green: in *S*

Example: Dijkstra's Algorithm



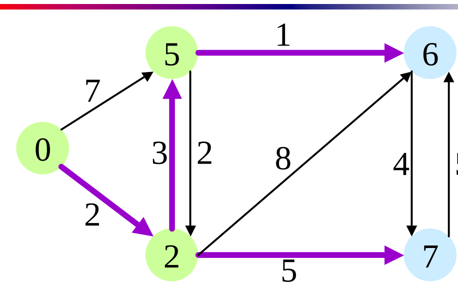
Green: in *S*

Example: Dijkstra's Algorithm



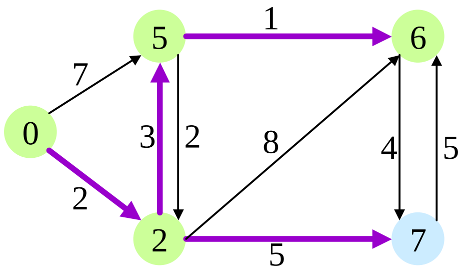
Green: in *S*

Example: Dijkstra's Algorithm



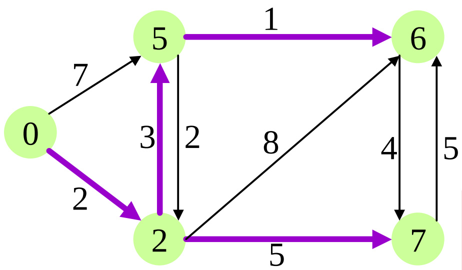
Green: in *S*

Example: Dijkstra's Algorithm



Green: in *S*

Example: Dijkstra's Algorithm



Time Complexity is O(E log V).

Green: in *S*