Shortest Path Algorithm

Alexandre David
B2-206

Menu

- Introduction to graphs, definitions.
- Finding a path.
- Shortest-path problem & algorithm.
  - Bellman-Ford.
  - Dijkstra.

Graphs – Definition

- A graph is a pair (V,E)
  - V finite set of vertices.
  - E finite set of edges.
    $e \in E$ is a pair (u,v) of vertices.
    Ordered pair $\rightarrow$ directed graph.
    Unordered pair $\rightarrow$ undirected graph.

V=
E=

V=
E=

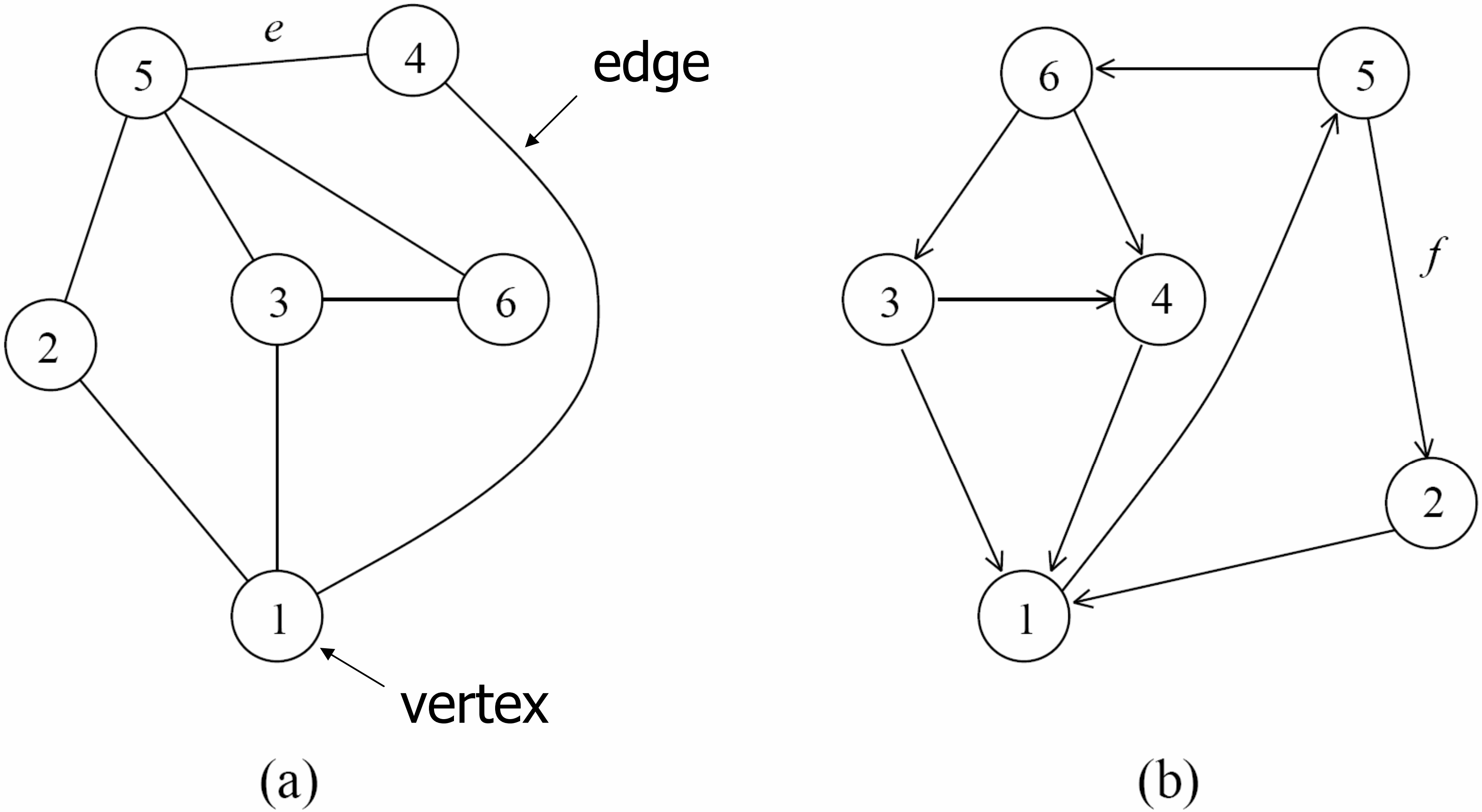


Figure 10.1 (a) An undirected graph and (b) a directed graph.

Graphs – Edges

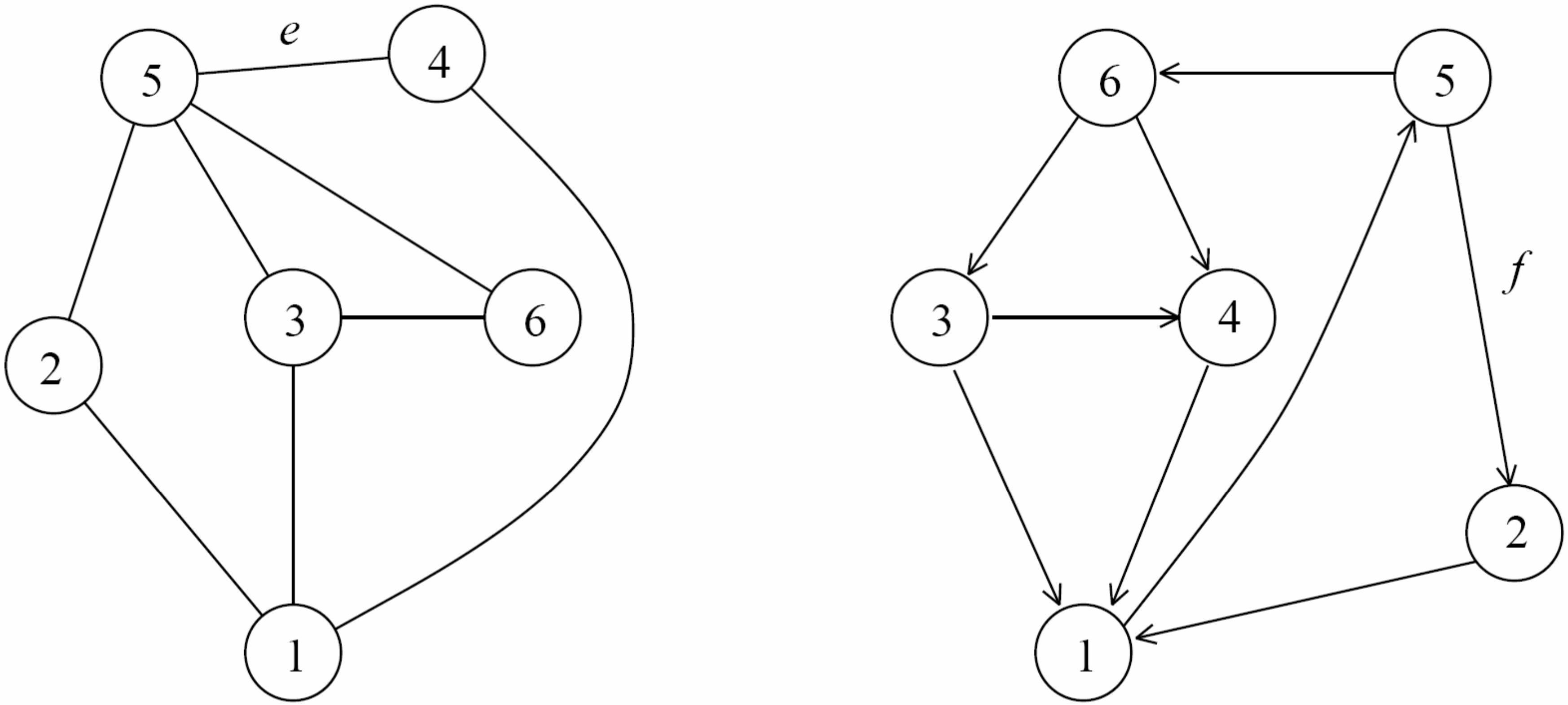
- Directed graph:
  - $(u,v) \in E$ is incident from u and incident to v.
  - $(u,v) \in E$: vertex v is adjacent to u.
- Undirected graph:
  - $(u,v) \in E$ is incident on u and v.
  - $(u,v) \in E$: vertices u and v are adjacent to each other.

Graphs – Paths

- A path is a sequence of adjacent vertices.
  - Length of a path = number of edges.
  - Path from v to $u \Rightarrow u$ is reachable from v.
  - Simple path: All vertices are distinct.
  - A path is a cycle if its starting and ending vertices are the same.
  - Simple cycle: All intermediate vertices are distinct.

Simple path:
Simple cycle:
Non simple cycle:

Simple path:
Simple cycle:
Non simple cycle:



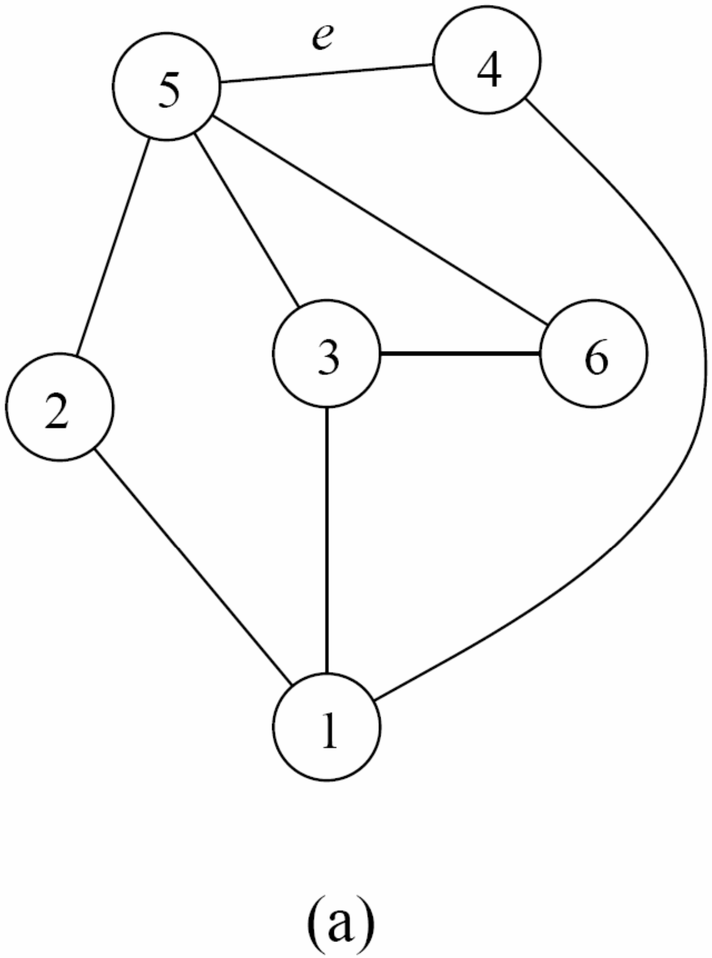
(a) (b)

Figure 10.1 (a) An undirected graph and (b) a directed graph.

Graphs

- Connected graph: ∃ path between any pair.
- $G'=(V',E')$ sub-graph of $G=(V,E)$ if $V'\subseteq V$ and $E'\subseteq E$.
- Sub-graph of G induced by V': Take all edges of E connecting vertices of $V'\subseteq V$.
- Complete graph: Each pair of vertices adjacent.
- Tree: connected acyclic graph.

Sub-graph:
Induced sub-graph:



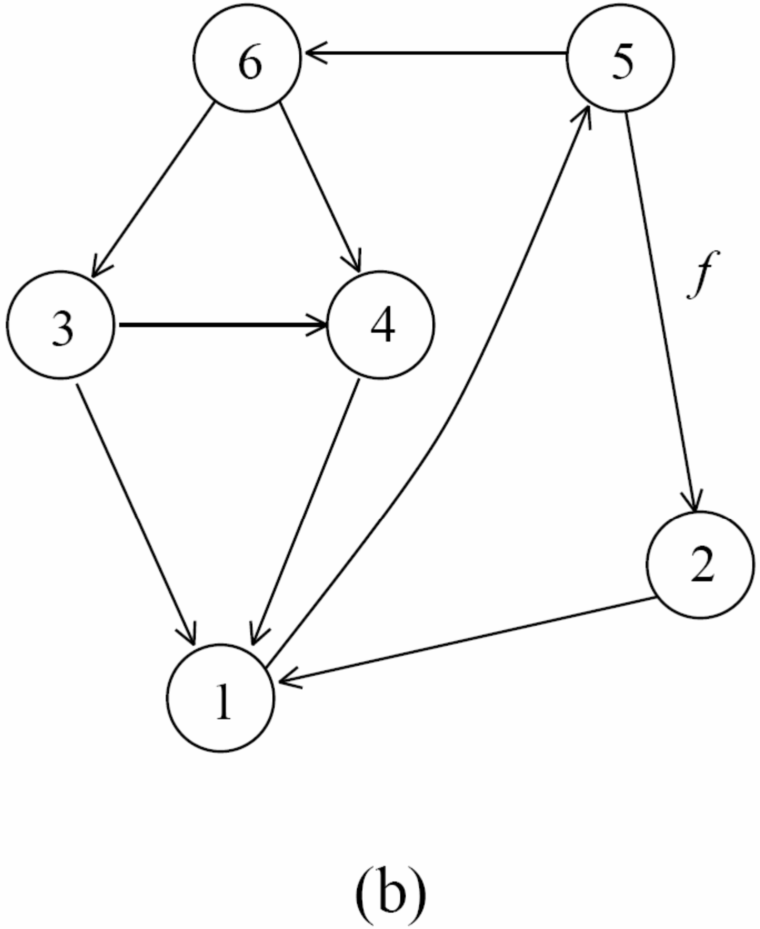
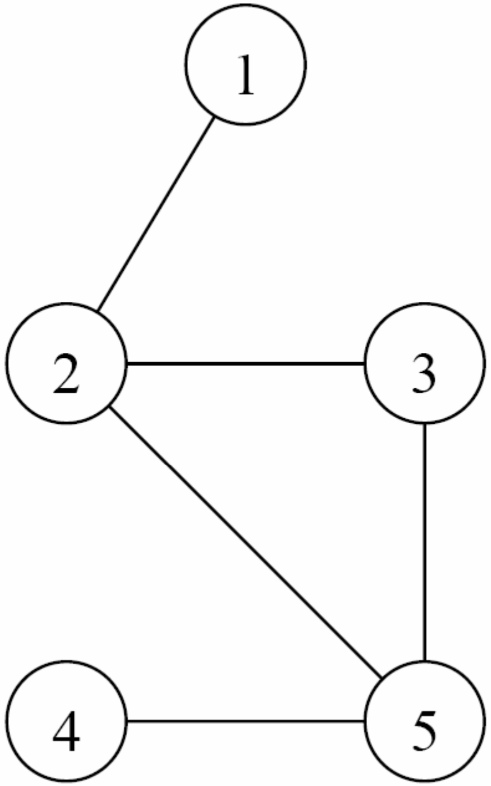


Figure 10.1 (a) An undirected graph and (b) a directed graph.

Graph Representation

- Sparse graph ($|E|$ much smaller than $|V|^2$):
  - Adjacency list representation.
- Dense graph:
  - Adjacency matrix.
- For weighted graphs (V,E,w): weighted adjacency list/matrix.

$$a_{i,j} = \begin{cases} 1 & if\ (v_i, v_j) \in E \\ 0 & otherwise \end{cases}$$



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

|V|

$|V|^2$ entries

Figure 10.2 An undirected graph and its adjacency matrix representation.

Undirected graph ⇒ symmetric adjacency matrix.

|V|+|E| entries

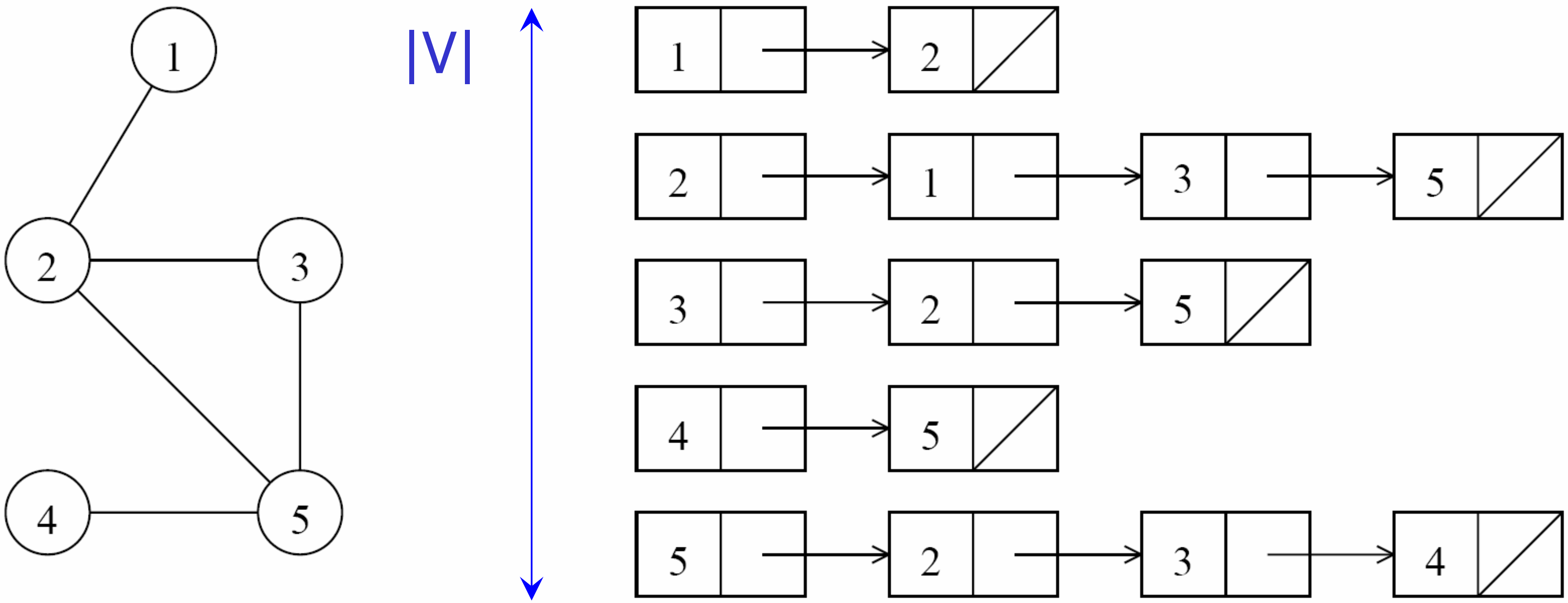


Figure 10.3 An undirected graph and its adjacency list representation.

Finding a Path

- Straight-forward DFS or BFS algorithm.
- Specialized DFS version. (Call with $S=\emptyset$).

```
DFS_find(G,s,t,S):
if s ∈ S then
   return false
fi
push(S,s)
if s = t then
   return true
fi
forall s → s' do
   if DFS_find(G,s',t,S) then
      return true
   fi
done
pop(s)
return false
```

Shortest Path Problem

- Given a weighted directed graph G=(V,E) with weight function w : E→**R**, the weight of a path p=$\langle v_0 \ldots v_k \rangle$ is defined by

$$w(p) = \sum_{i=0}^{k} w(v_{i-1}, v_i)$$

- The shortest-path weight from u to v is defined by δ(u,v)=min{w(p):there is a path from u to v}, ∞ otherwise.
- A shortest path from vextex u to vextex v is then defined by any path with weight w(p)=δ(u,v).

Variants

- Single-source shortest-paths: from a source to every vertex.
- Single-destination shortest-paths: from every vertex to a destination.
- Single-pair shortest path: between a pair of vertices u and v.
- All-pairs shortest-paths: for all pairs of vertices.

Optimal Sub-structure of Shortest Paths

- Shortest-paths algorithm rely on the property that a shortest path between 2 vertices *contains other shortest paths within it*.
- Lemma: Let $p=\langle v_0 \ldots v_k \rangle$ be a shortest path from v_0 to v_k. For any i,j $0 \le i \le j \le k$, $p_{ij}=\langle v_i \ldots v_j \rangle$ is a shortest path from v_i to v_j.
  - Proof technique: Suppose it is not the case and obtain a contradiction with the hypothesis.

Negative Weight Cycles

- Pose problems to define shortest-paths.
- We suppose we do not have negative weight cycles otherwise the shortest-path is not well-defined.
  - Stay in the cycle and get -∞ as the sum.
- Other cycles can be removed without loss of generality (if weight=0, otherwise not shortest).

Representation

- We want to compute shortest-path weights and the vertices on the path. For a graph $G=(V,E)$
  - $\pi(v)$ is a predecessor of $v \in V$, or NIL.
  - π values induce the predecessor sub-graph $G_\pi=(V_\pi,E_\pi)$.
    $V_\pi=\{v \in V : \pi(v) \neq \text{NIL}\}\cup\{s\}$. (+ source s)
    $E_\pi=\{(\pi(v),v) \in E : v \in V_\pi-\{x\}\}$.
  - The shortest-path algorithm computes π and the result is a ”shortest-path tree”.

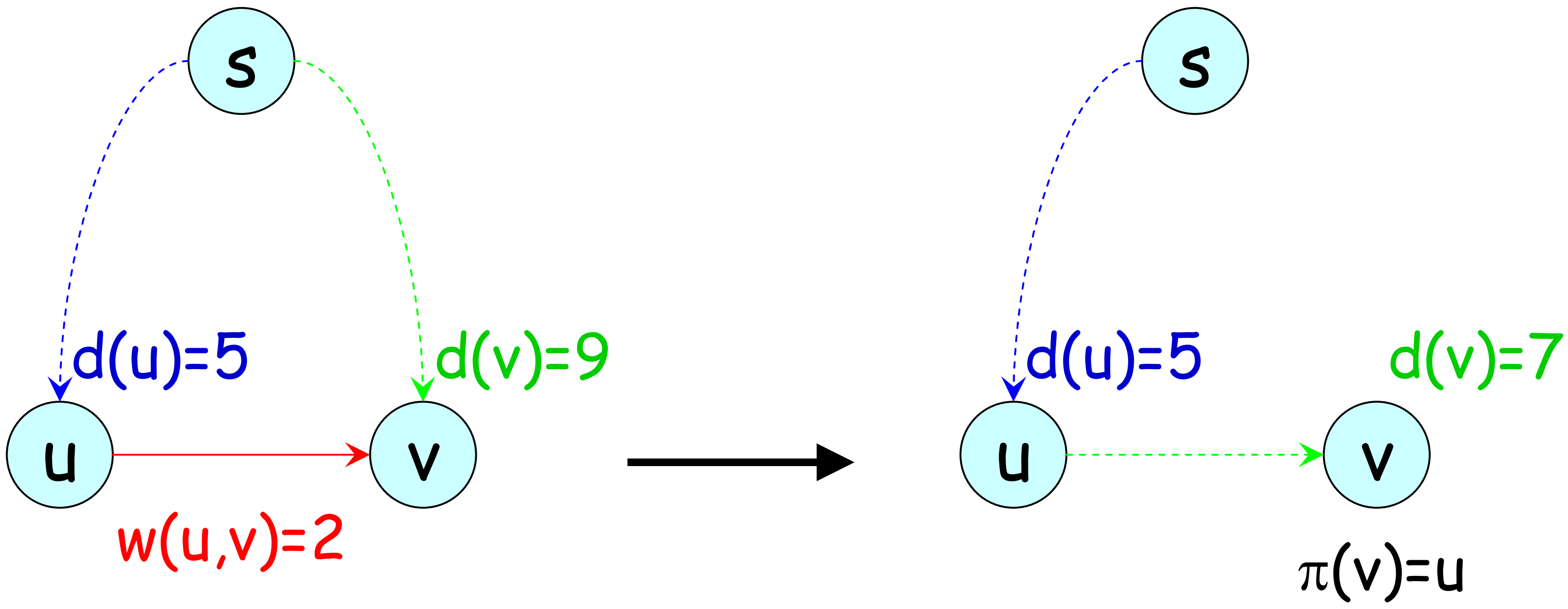
Tightening – Relaxation (Historical Reasons)

- Attribute d(v) is the current known shortest path weight to v, i.e., it's an upper-bound on **the** shortest path weight.

```
Initialize_single_source(G,s):
forall v ∈ V(G) do
    d(v) = ∞
    π(v) = NIL
done
d(s) = 0
```

```
Relax(u,v,w):
if d(v) > d(u)+w(u,v) then
    d(v) = d(u)+w(u,v)
    π(v) = u
fi
```

Tightening



Shorter to v via u:
$d(u)+w(u,v) < d(v)$.

Single-Source Shortest-Paths Algorithms

- Bellman-Ford.
  - General case with negative weights.
  - Detect if negative weight cycles are reachable.
- Dijkstra.
  - Requires positive weights.

Bellman-Ford

Repeat →

Relax all pairs of edges. →

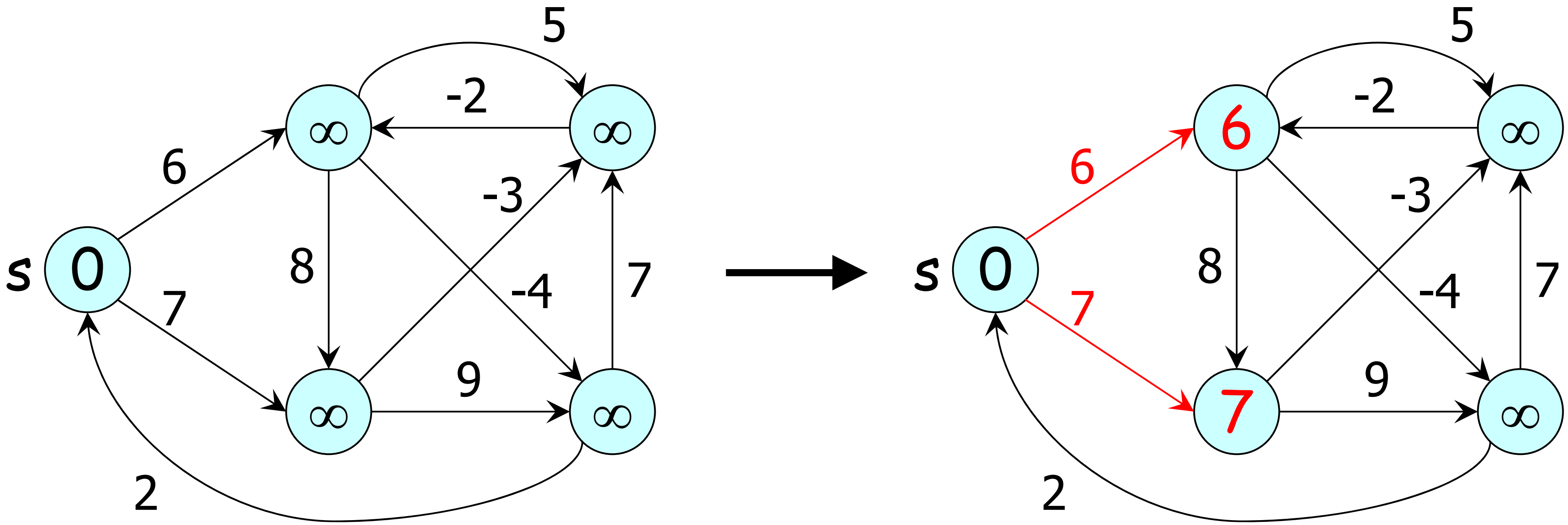
O(|V|*|E|)

Upon termination, either

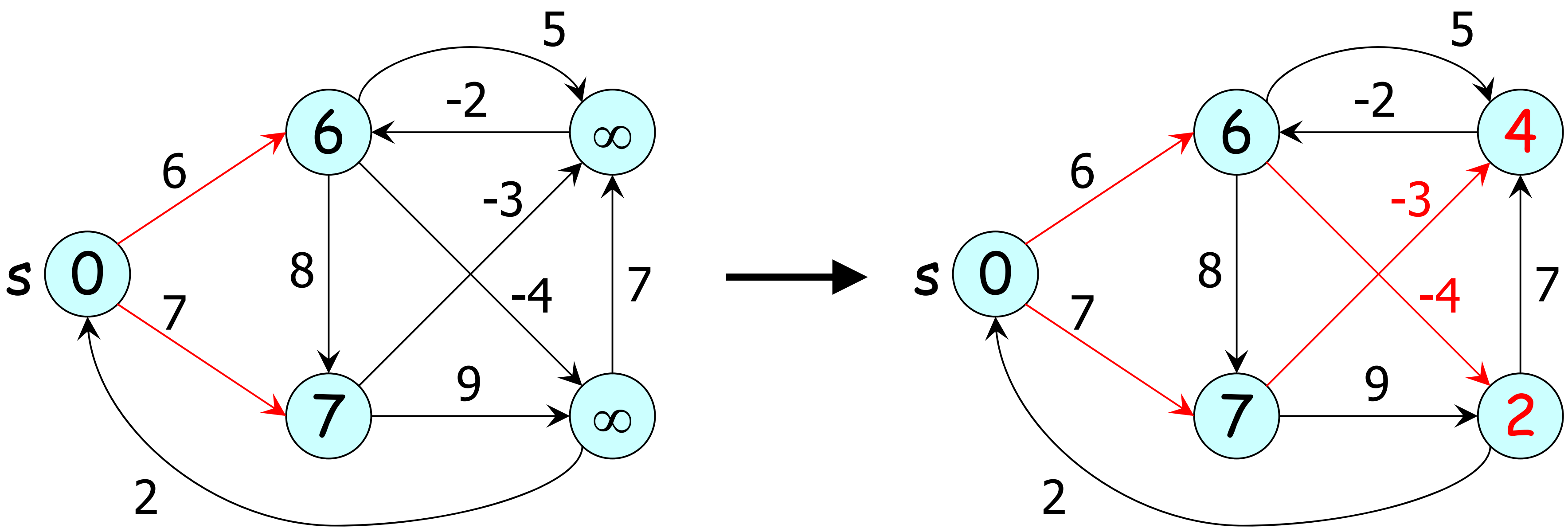
- the algorithm converged to the shortest path,
- or there is a negative cycle and it didn't converge.

```
Bellman_Ford(G,w,s):
Initialize_single_source(G,s)
for i = 1 to |V(G)|-1 do
    forall (u,v) ∈ E(G) do
        Relax(u,v,w)
    done
done
forall (u,v) ∈ E(G) do
    if d(v) > d(u)+w(u,v) then
        return false
    fi
done
return true
```

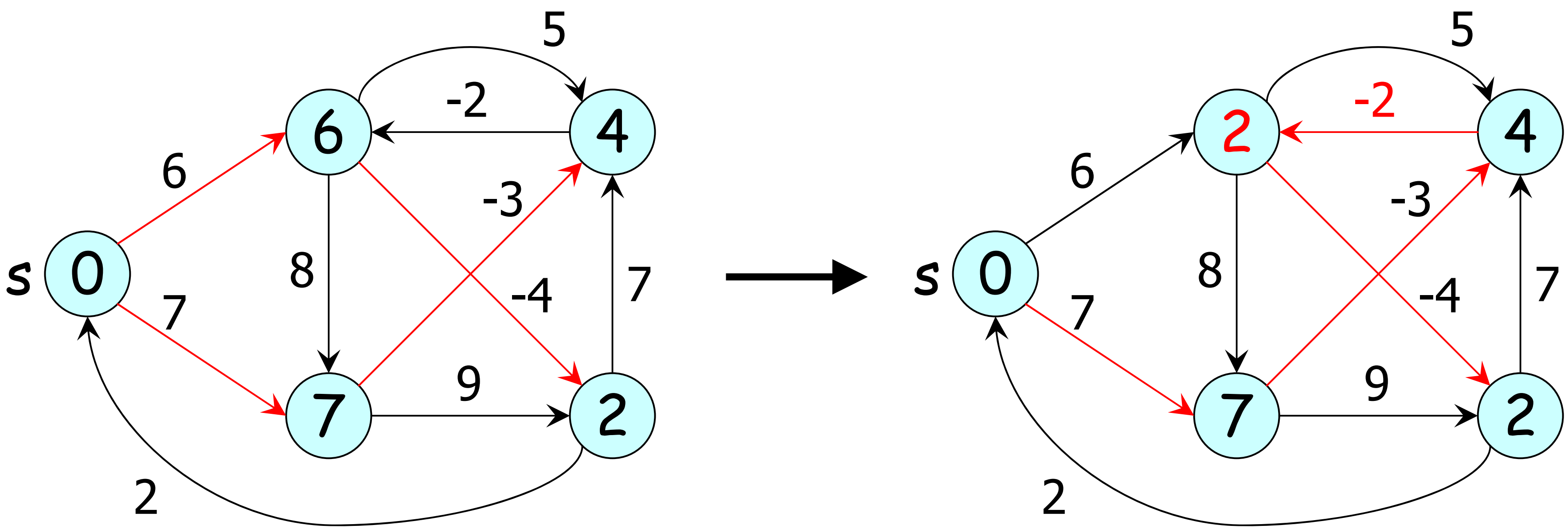
Example



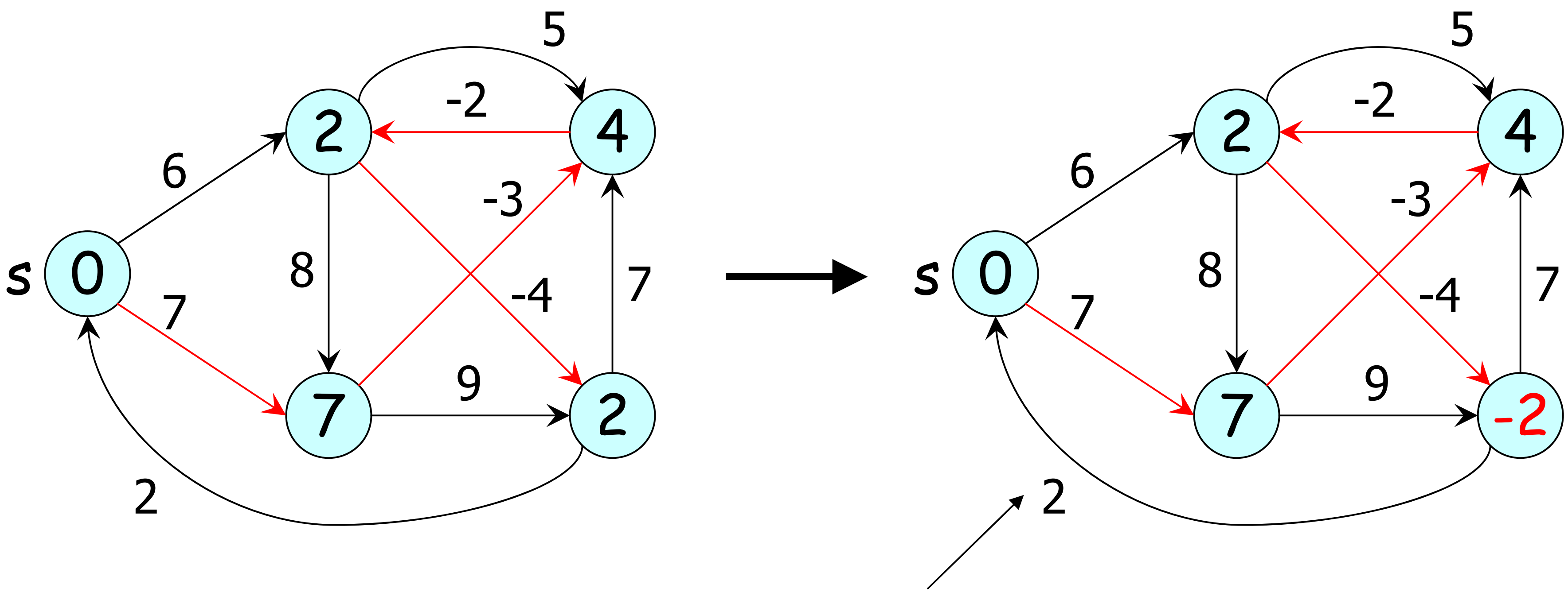
Example



Example



Example



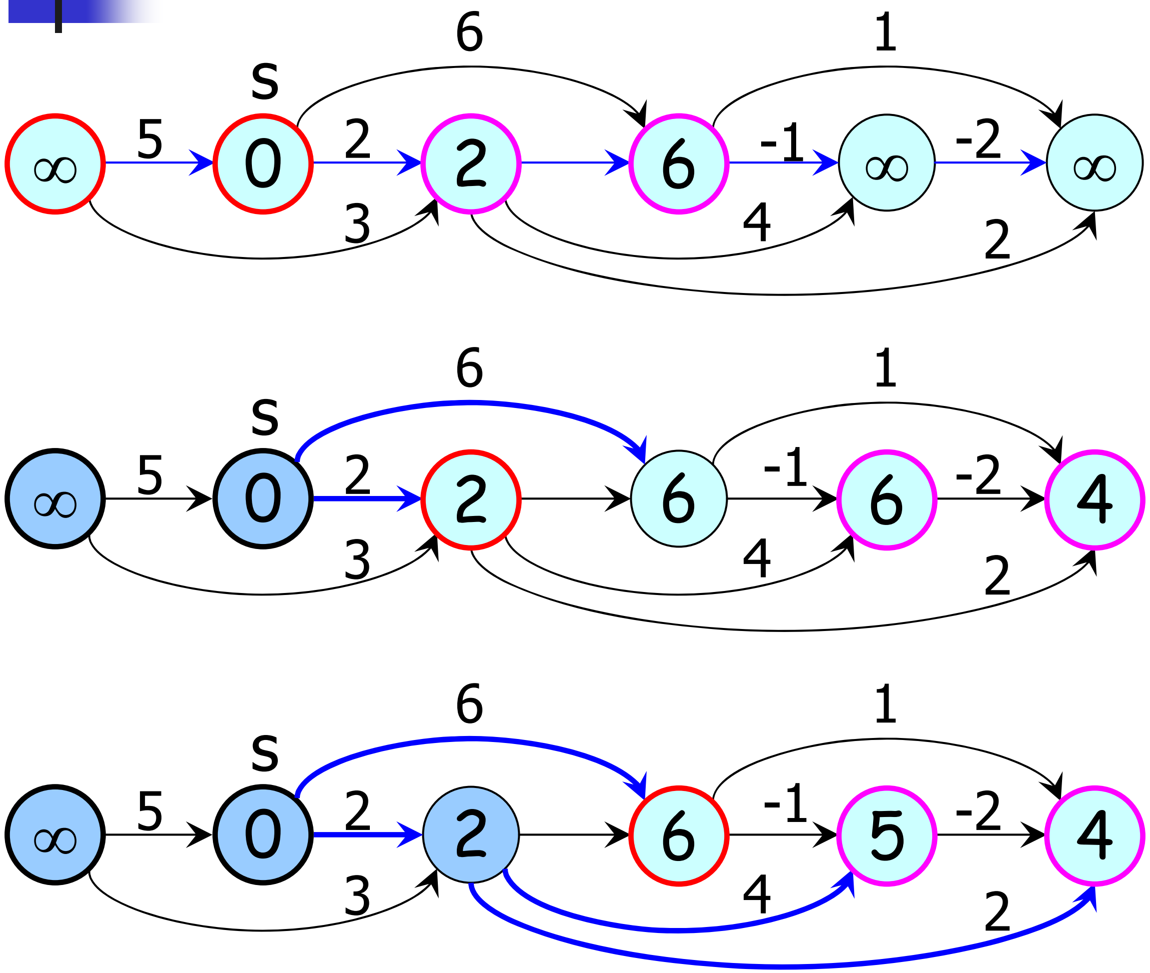
If 1 here, then
we have a negative cycle!

Special Case: Directed Acyclic Graphs

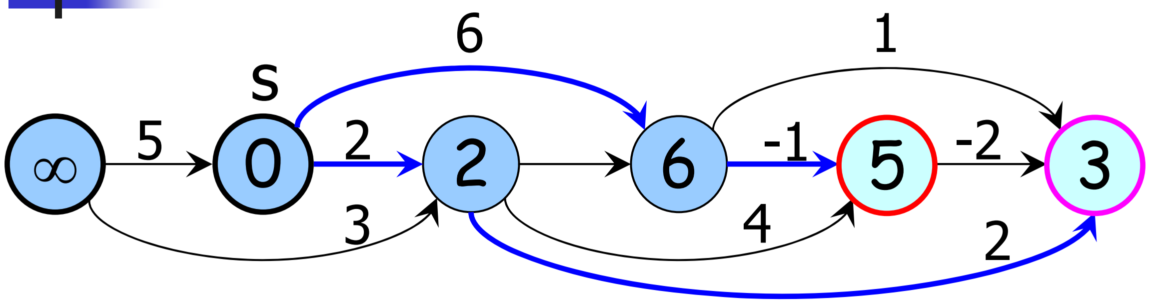
- Specialized algorithm: One pass over vertices in topologically sorted order.

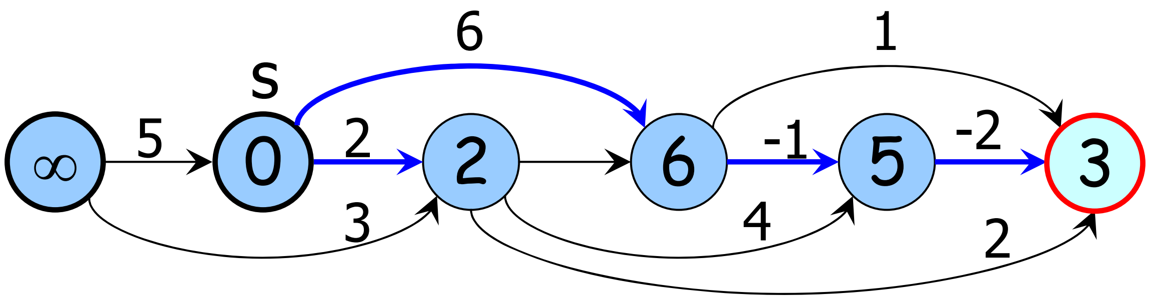
```
DAG_shortest_path(G,w,s):
Initialize_single_source(G,s)
forall u ∈ V in topological order do
    forall v adjacent to u do
        Relax(u,v,w)
    done
done
```

Example



Example





Dijkstra's Algorithm

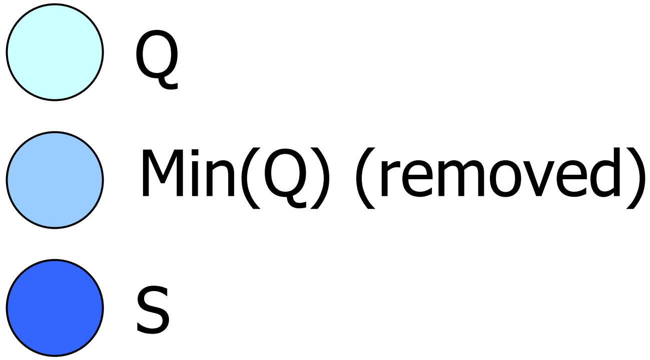
For non-negative weights

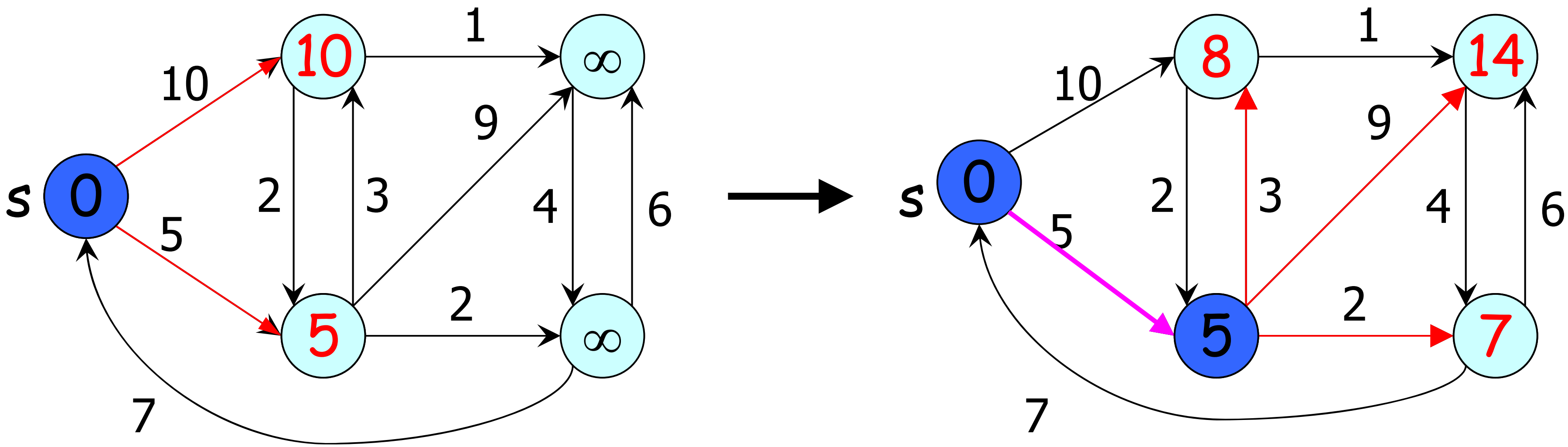
- Maintains a set S of vertices with known shortest paths.
  - Select $u \in V\text{-}S$ with minimum estimate.
  - Add u to S.
  - Relax edges leaving u.

Dijkstra's Algorithm

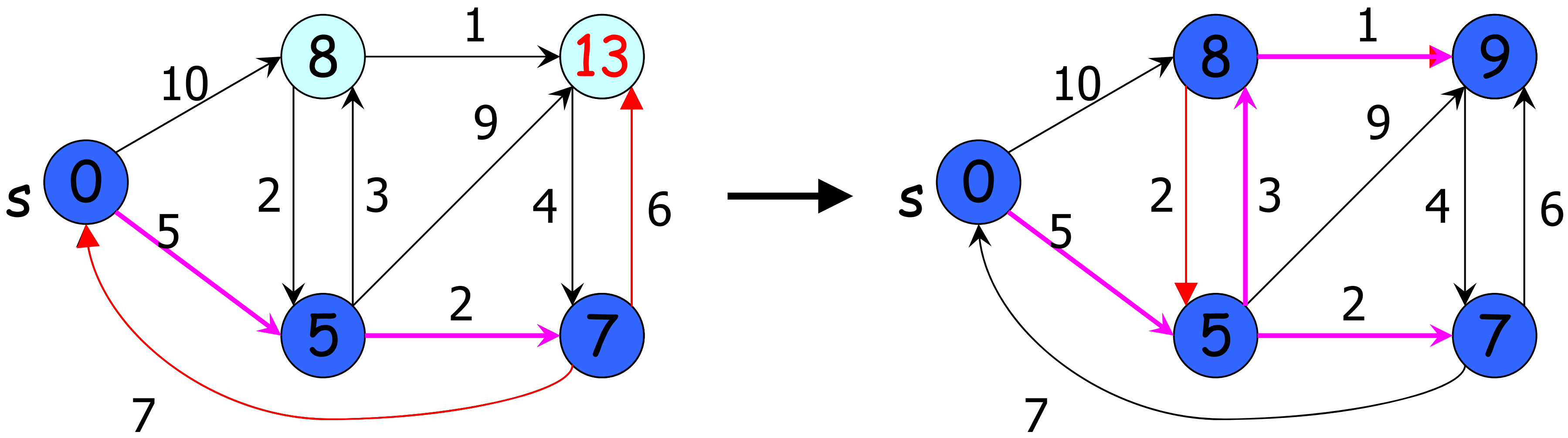
```
Dijkstra(G,w,s):
Initialize_single_source(G,s)
S = ∅
Q = V(G) priority queue keyed by d
while Q ≠ ∅ do
    u = get_min(Q)
    S = S∪{u}
    forall v adajacent to u do
        Relax(u,v,w)
    done
done
```

Example





Example



Dijkstra's Algorithm

- Efficiency: Depends on the priority queue. Can be $O((V+E) \lg V)$.
- Implementation:
  - Array d[] for "distance" from the source.
  - Array l[] for "last" vertex.
  - The priority queue.