Single-Source Shortest Paths

Heejin Park
Division of Computer Science and Engineering
Hanyang University

Contents

- Definition
- The Bellman-Ford algorithm
- Single-source shortest paths in directed acyclic graphs
- Dijkstra's algorithm

Definition

- ***Edge weight***

- ***Path weight***
  - The sum of all edge weights in the path.

- ***A Shortest path*** from u to v.
  - A path from u to v whose weight is the smallest.
  - Vertex u is the ***source*** and v is the ***destination***.

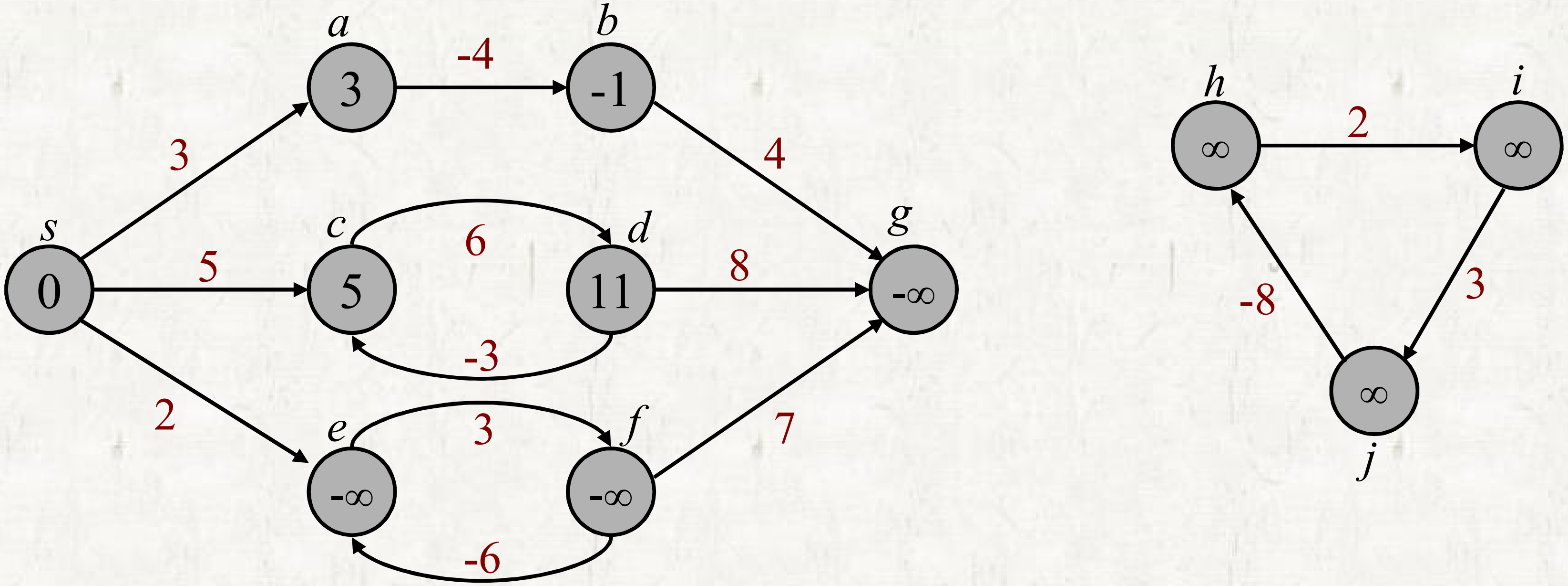
- ***The Shortest-path weight*** from u and v.
  - The weight of a shortest-path from u and v
  - $\delta(u,v)$

Definition

- **Shortest-path problems**
  - Single-source & single-destination
  - Single-source (& all destinations)
  - Single-destination (& all sources)
  - All pairs

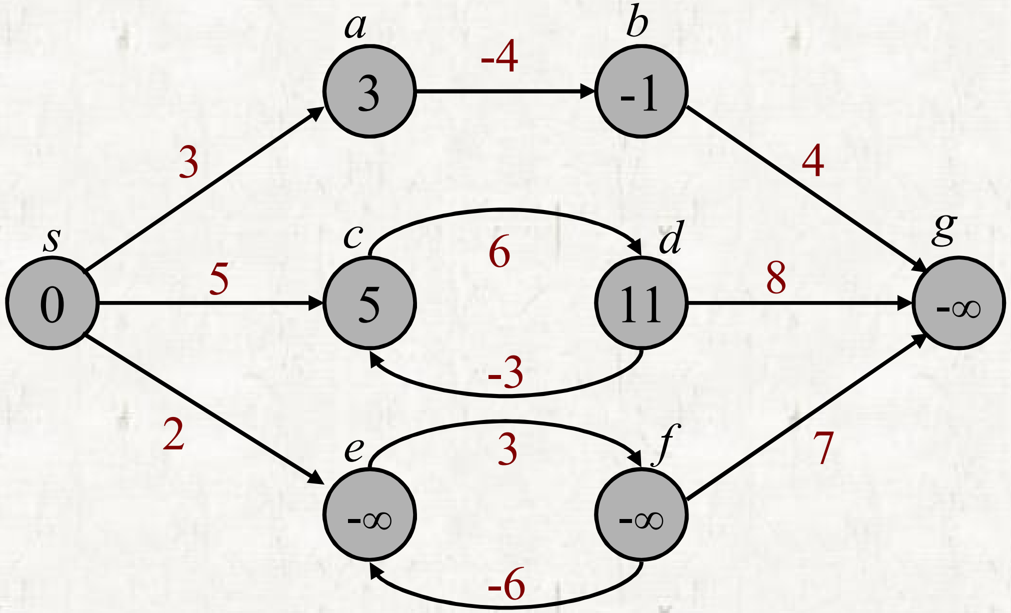
Negative-weight edges

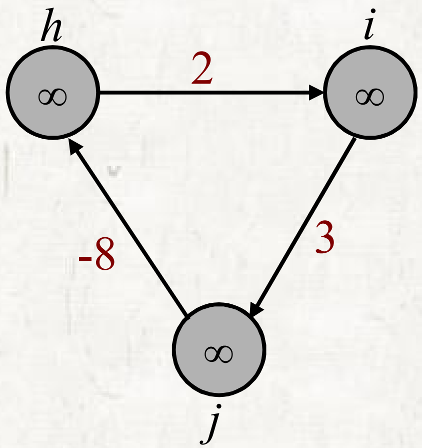
- What is a shortest path from *s* to *g*?



Negative-weight edges

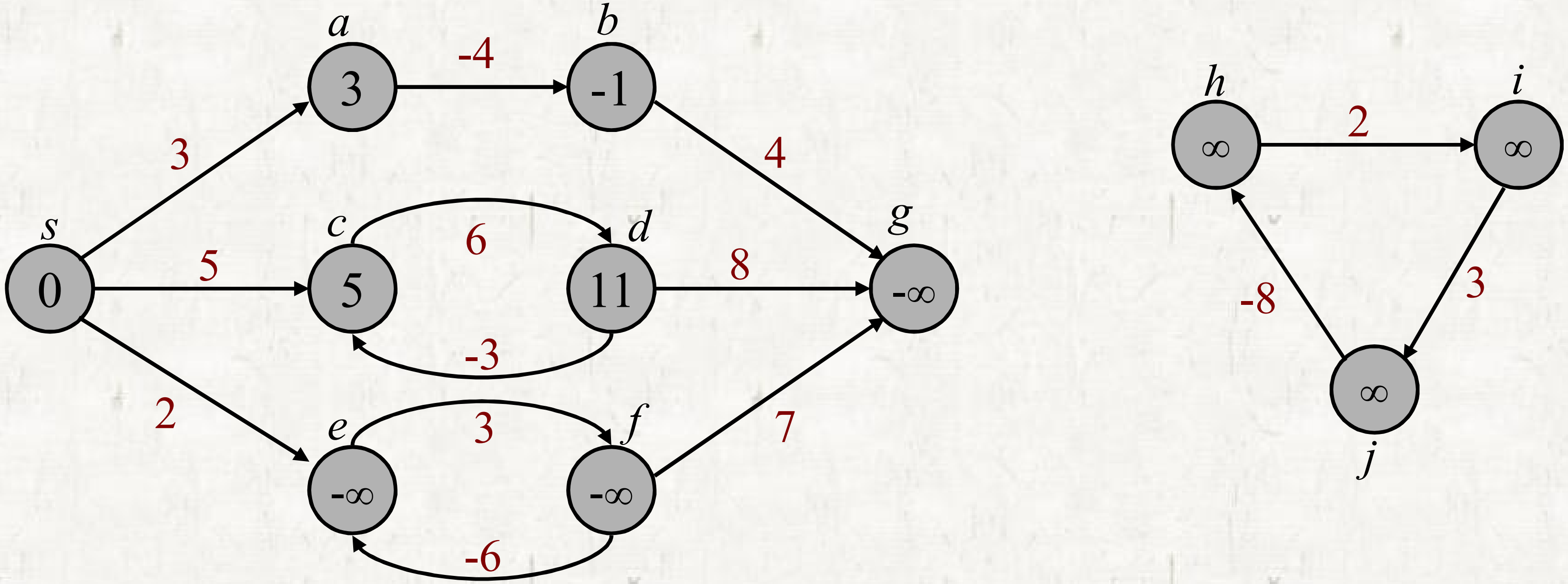
- Do all negative-weight edges cause a problem?
- Do all negative-weight cycles cause a problem?





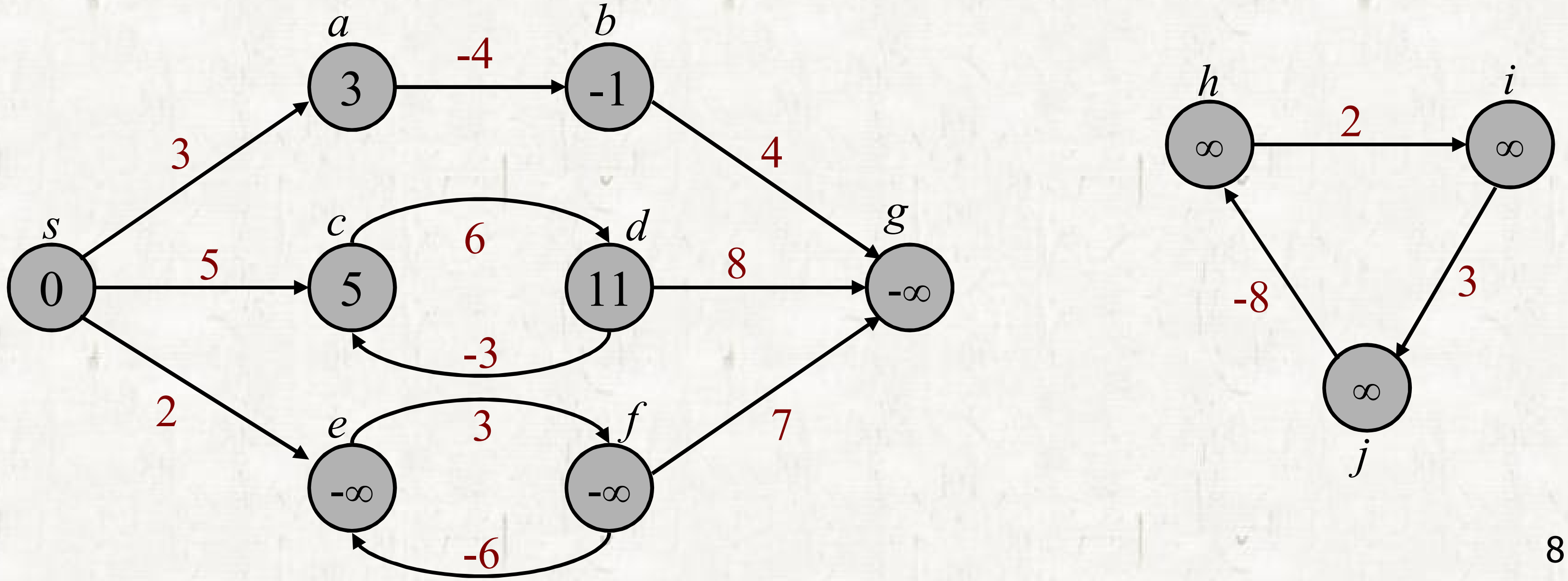
Negative-weight edges

- Do all negative-weight cycles reachable from the source cause a problem?



Negative-weight edges

- Single-source shortest paths can be defined if there are not any ***negative-weight cycles reachable from the source.***

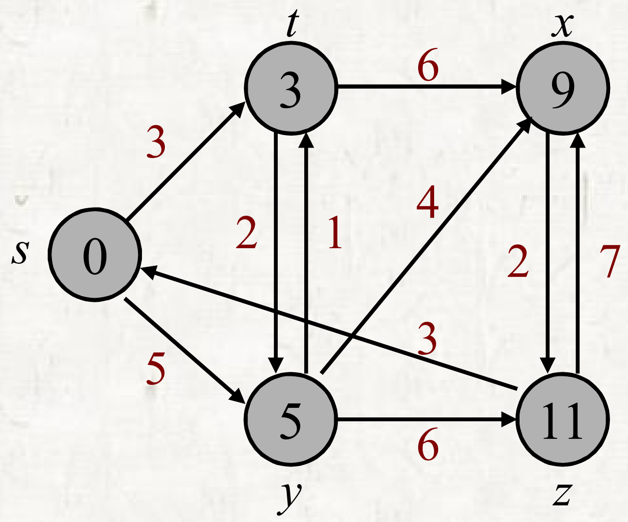


Cycles

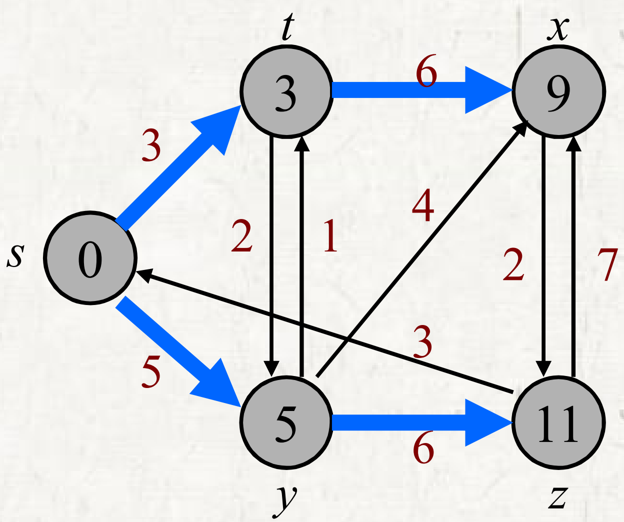
- **Cycles**
  - There is a shortest path that does not include cycles.
  - A shortest-path length is at most $|V|$-1.

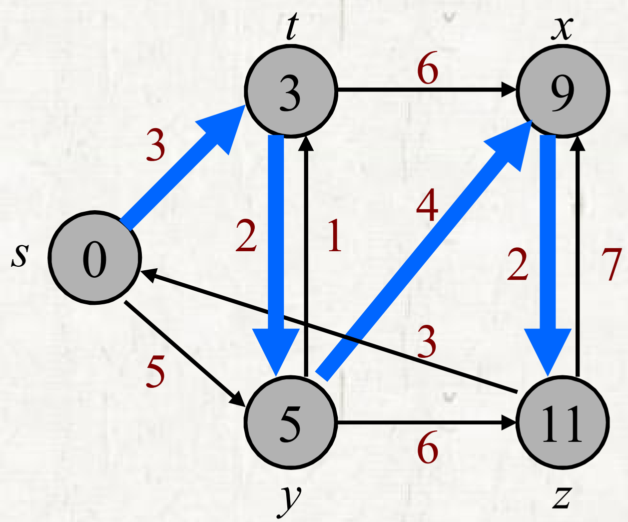
Predecessor subgraph

- **Predecessor subgraph**
  - Shortest-path tree (stores all SSSPs compactly.)
  - Optimal substructure



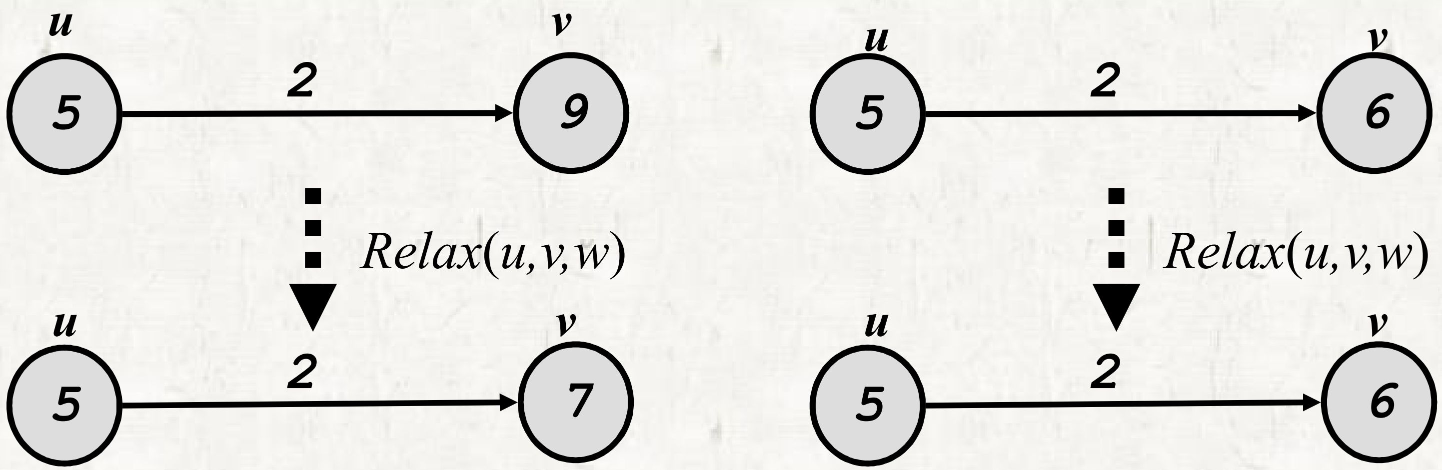
Predecessor subgraph





Relaxation

- **Relaxation**



Dijkstra's algorithm

- **Dijkstra's algorithm**
  - It works properly when all edge weights are ***nonnegative***.

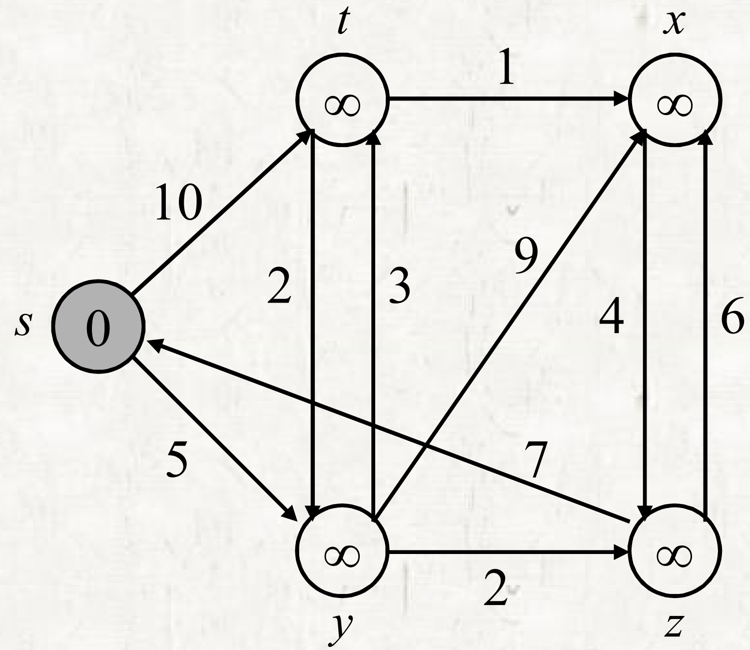
Dijkstra's algorithm

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = Ø
Q = G.V
while Q ≠ Ø
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Dijkstra's Algorithm

Q

s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞

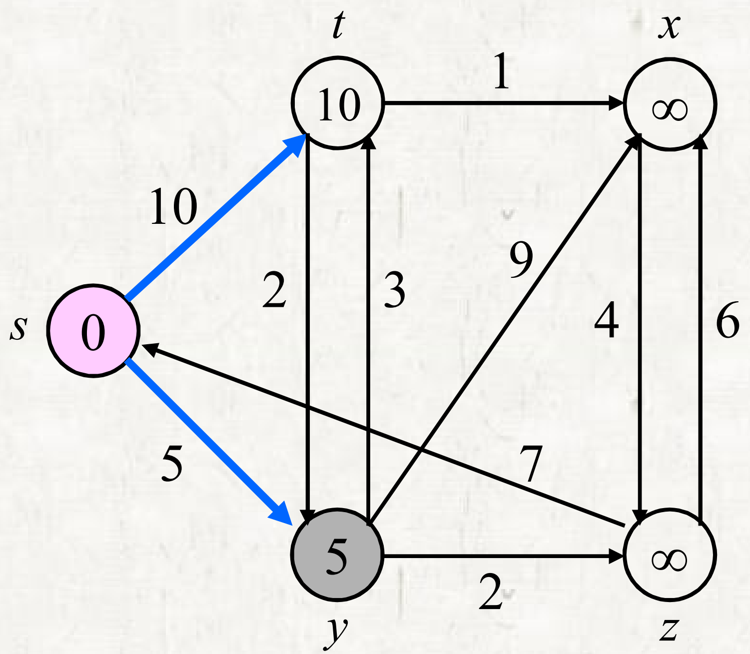


S

Dijkstra's Algorithm

Q

s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	5	-	-

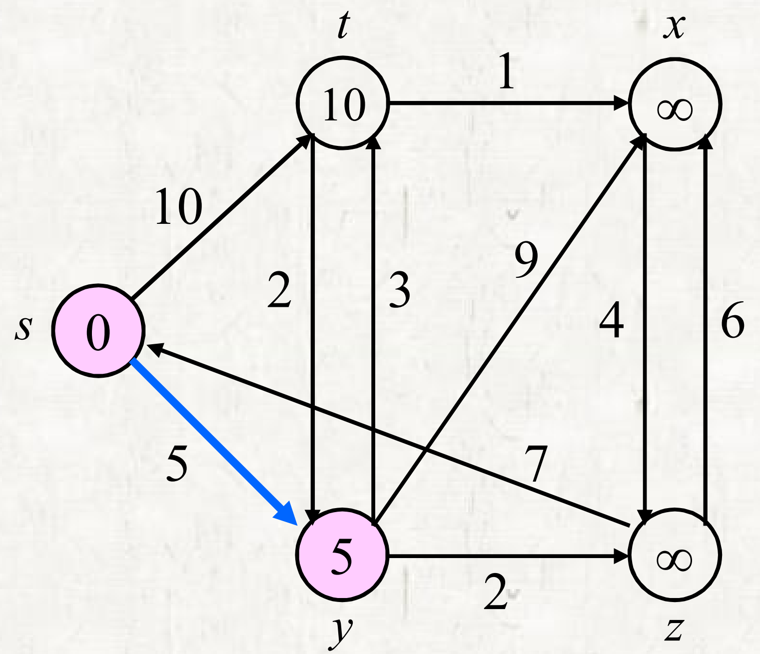


$\boldsymbol{S}=\{s\}$

Dijkstra's Algorithm

Q

s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	5	-	-

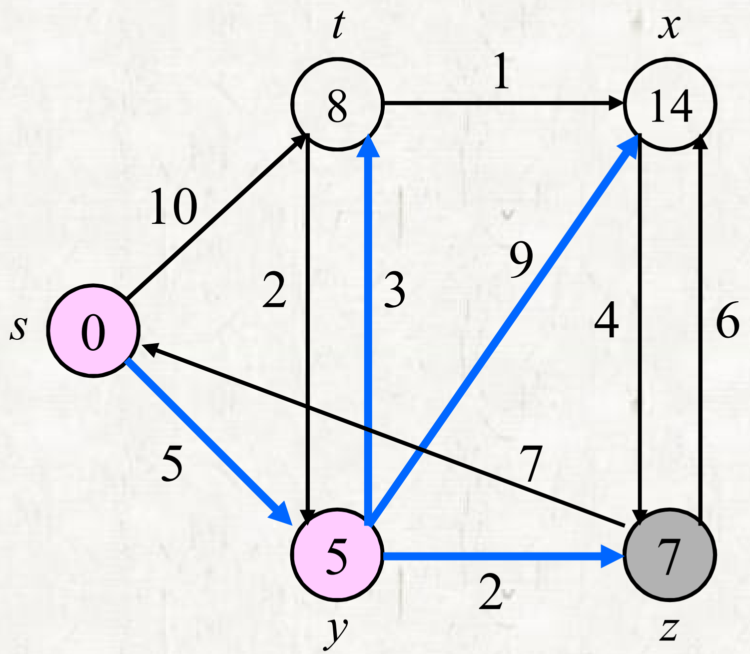


S$=\{s, y\}$

Dijkstra's Algorithm

Q

s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7

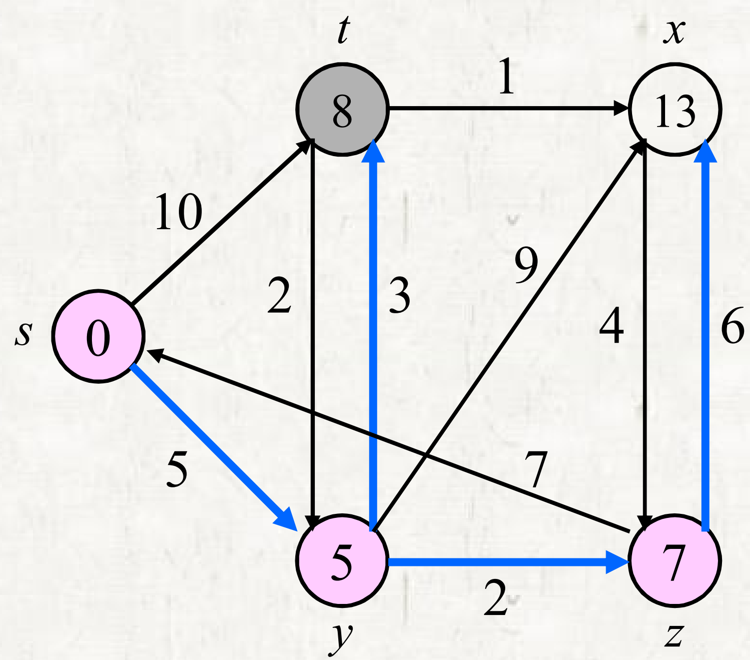


S={*s*, *y*}

Dijkstra's Algorithm

Q

s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7
	8		13	

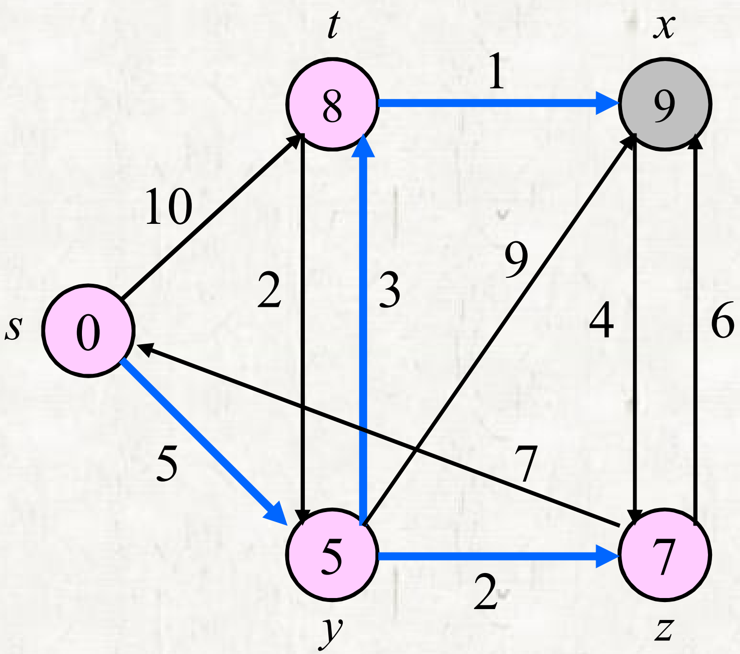


$\boldsymbol{S}=\{s, y, z, t\}$

Dijkstra's Algorithm

Q

s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7
	8		13	
			9	

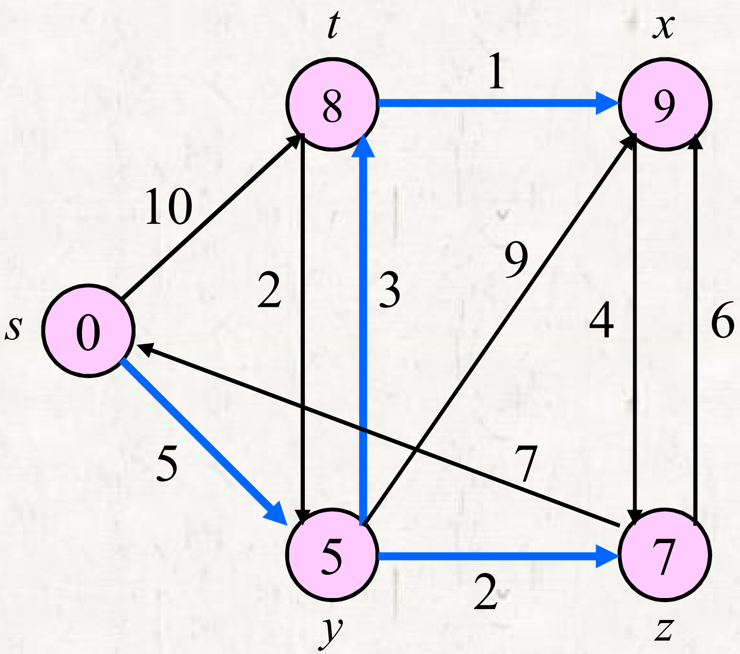


S={*s*, *y*, *z*, *t*}

Dijkstra's Algorithm

Q

s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7
	8		13	
			9	



$\boldsymbol{S}=\{s, y, z, t, x\}$

Dijkstra's algorithm

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = Ø
Q = G.V
while Q ≠ Ø
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Dijkstra's algorithm

- **Running time**
  - $O(V^2)$ if we use an (unsorted) array
  - $O(V \lg V + E \lg V)$ if we use a heap
  - $O(V \lg V + E)$ if we use a Fibonacci heap.

The Bellman-Ford algorithm

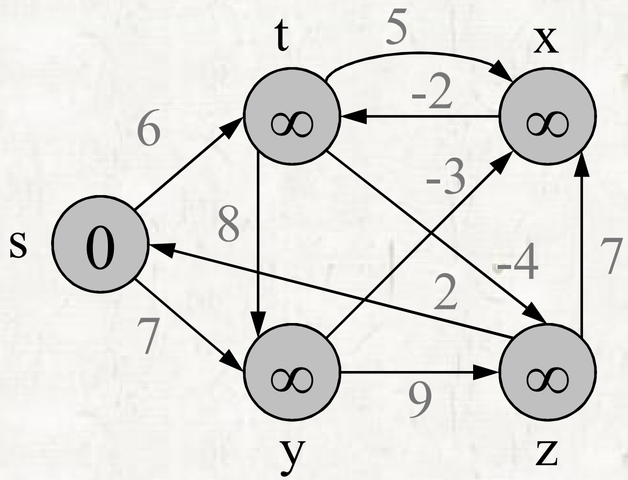
- **The Bellman-Ford algorithm**
  - it solves the single source shortest-paths problem in the general case in which edge weights may be negative.

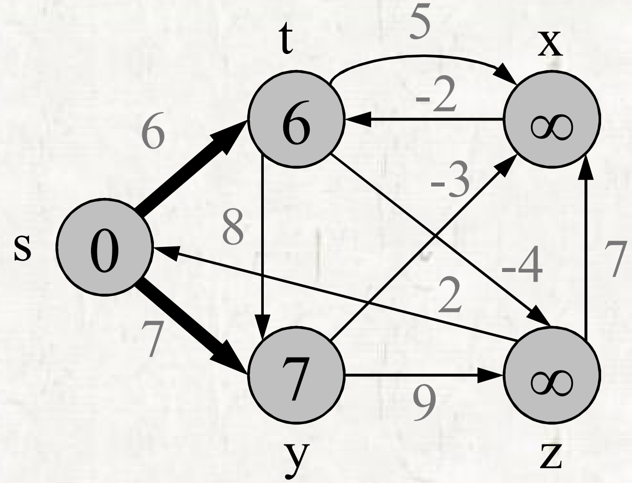
The Bellman-Ford algorithm

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

The Bellman-Ford algorithm

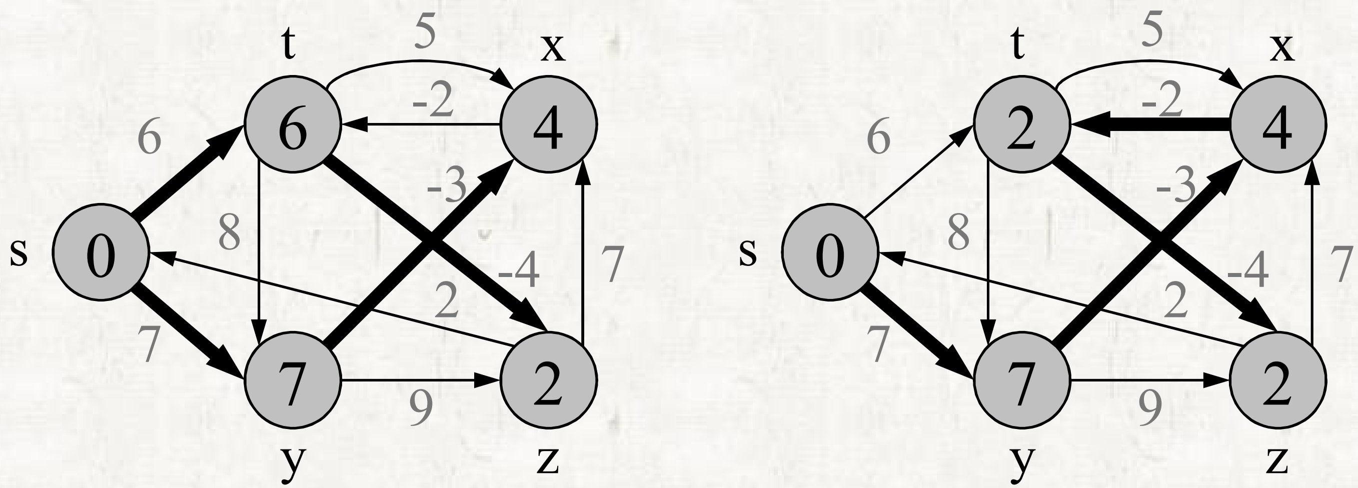
- Relaxation order
  - (t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)





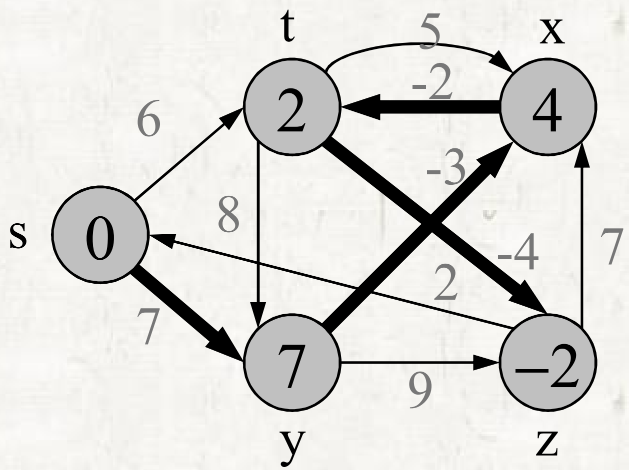
The Bellman-Ford algorithm

- Relaxation order
  - (t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)



The Bellman-Ford algorithm

- Relaxation order
  - (t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)



The Bellman-Ford algorithm

- **The Bellman-Ford algorithm**
  - Running time : $O(VE)$

The Bellman-Ford algorithm

$$\sum_{i=1}^{k} d[v_i] \le \sum_{i=1}^{k} \left(d[v_{i-1}] + w(v_{i-1}, v_i) \right)$$

$$= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

$$\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$$

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

Single-source shortest paths in directed acyclic graphs

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

topologically sort the vertices of G

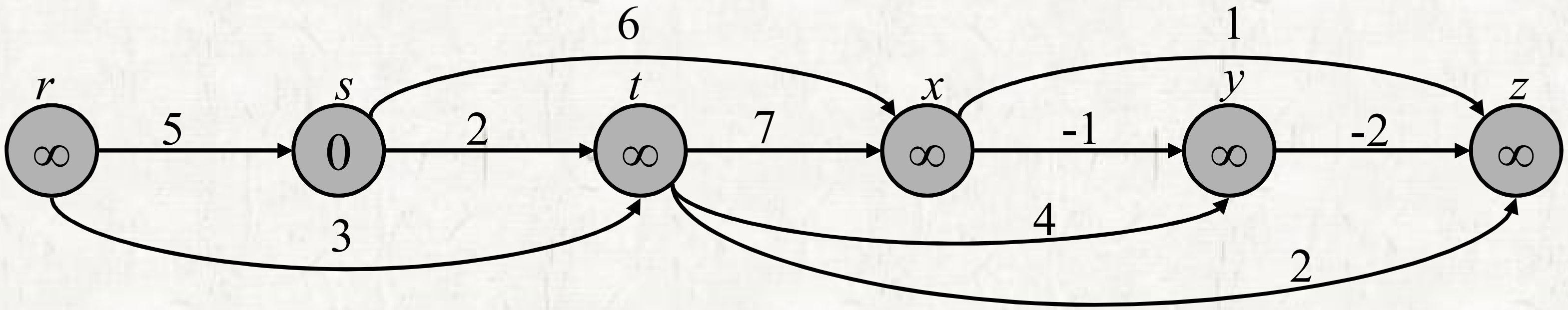
INITIALIZE-SINGLE-SOURCE(G, s)

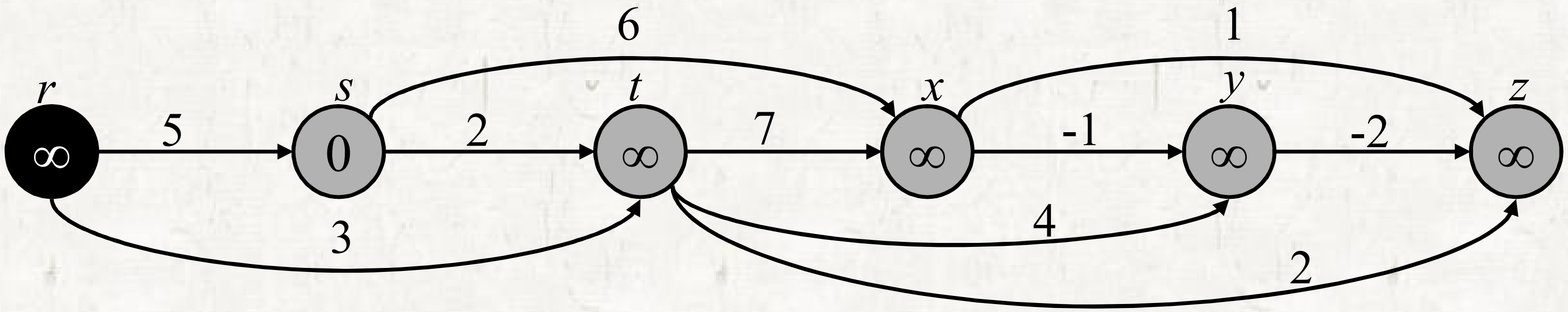
for each vertex u, taken in topologically sorted order

  for each vertex $v \in G.Adj[u]$

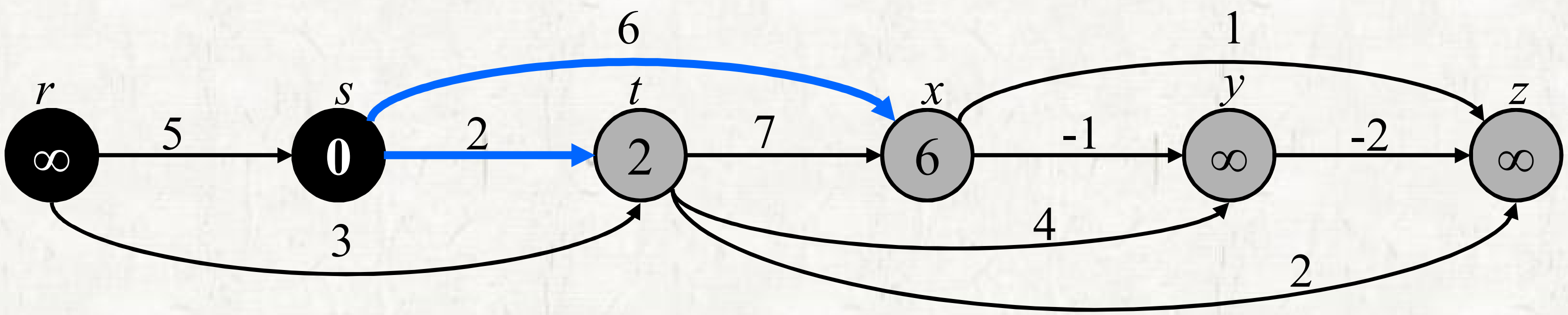
    RELAX(u, v, w)

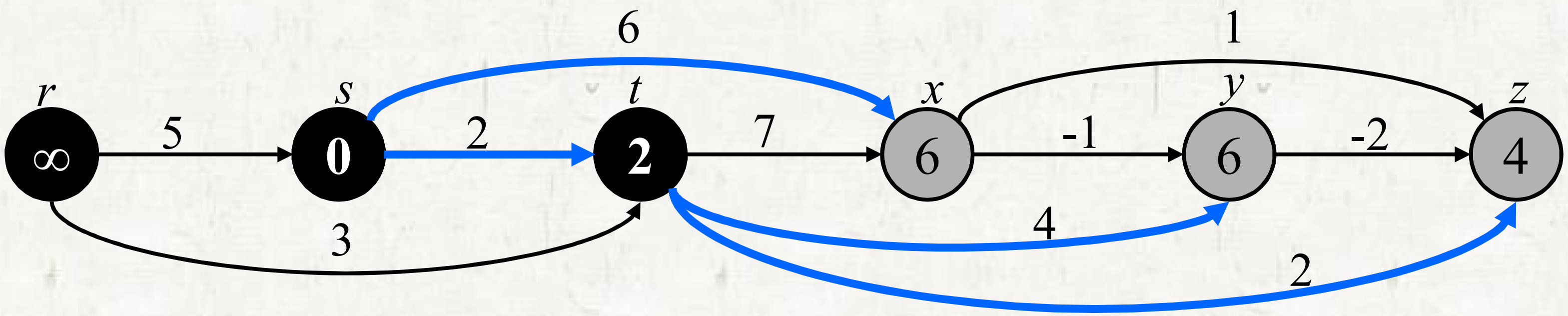
Single-source shortest paths in directed acyclic graphs



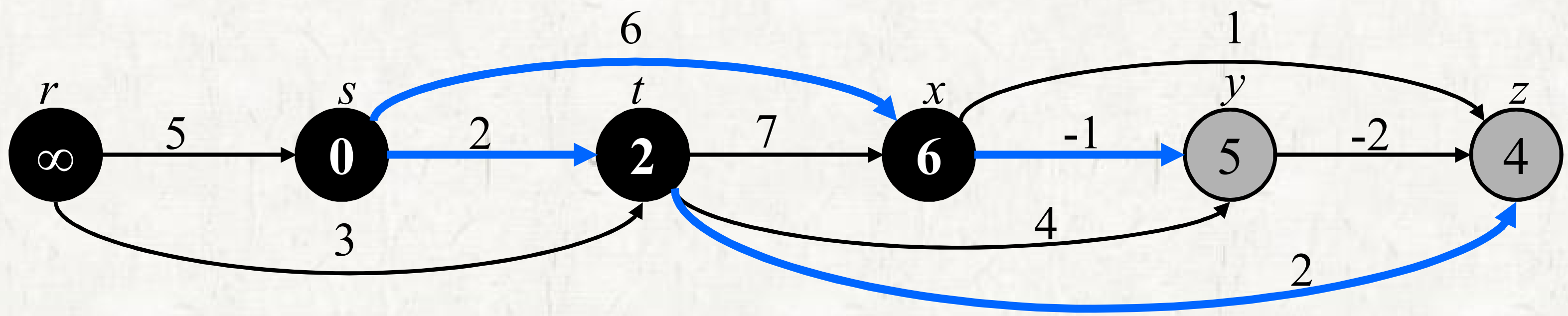


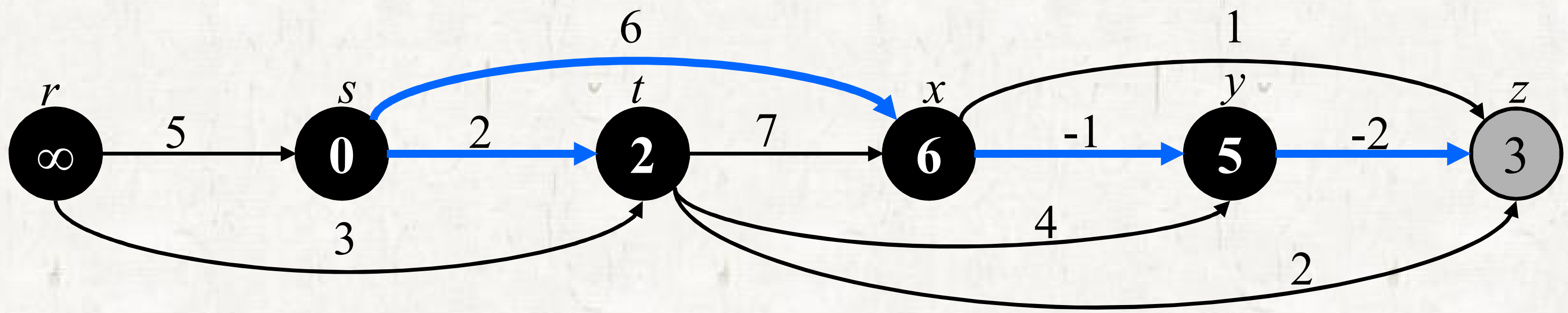
Single-source shortest paths in directed acyclic graphs



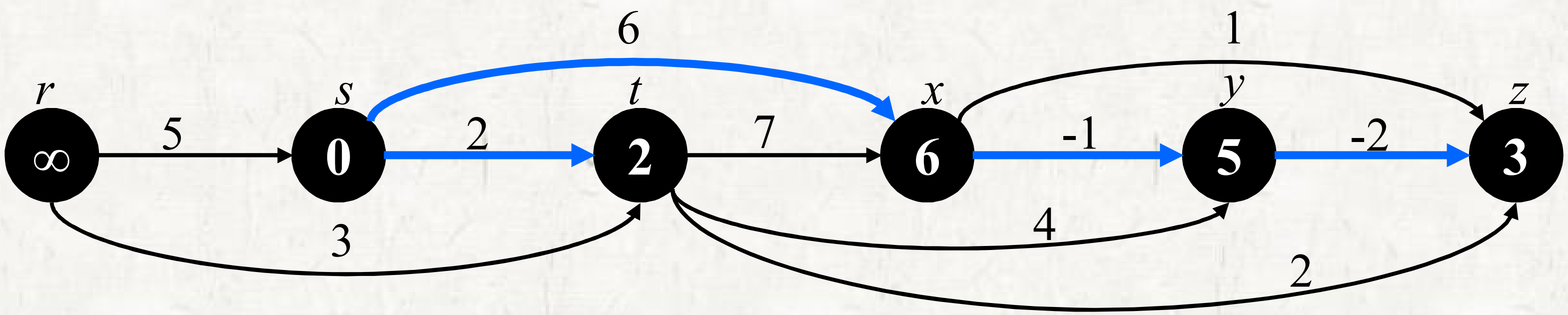


Single-source shortest paths in directed acyclic graphs





Single-source shortest paths in directed acyclic graphs



- Running time: $O(V+E)$ time

PERT chart

- PERT
  - Program evaluation and review technique
  - Edges represent jobs to be performed.
  - Edge weights represent the times required to perform particular jobs.

PERT chart

- PERT
  - If edge (u,v) enters vertex v and edge (v,x) leaves v, then job (u,v) must be performed prior to job (v,x).
  - A path through this dag represents a sequence of jobs that must be performed in a particular order.
  - A ***critical path*** is a longest path through the dag.

PERT chart

- Finding a critical path in a dag
  - Negate the edge weights and run DAG-SHORTEST-PATHS or
  - Run DAG-SHORTEST PATHS, with the modification that we replace "∞" by "$-\infty$" and ">" by "<".