Single-Source Shortest Paths

Slides from Heejin Park

Contents

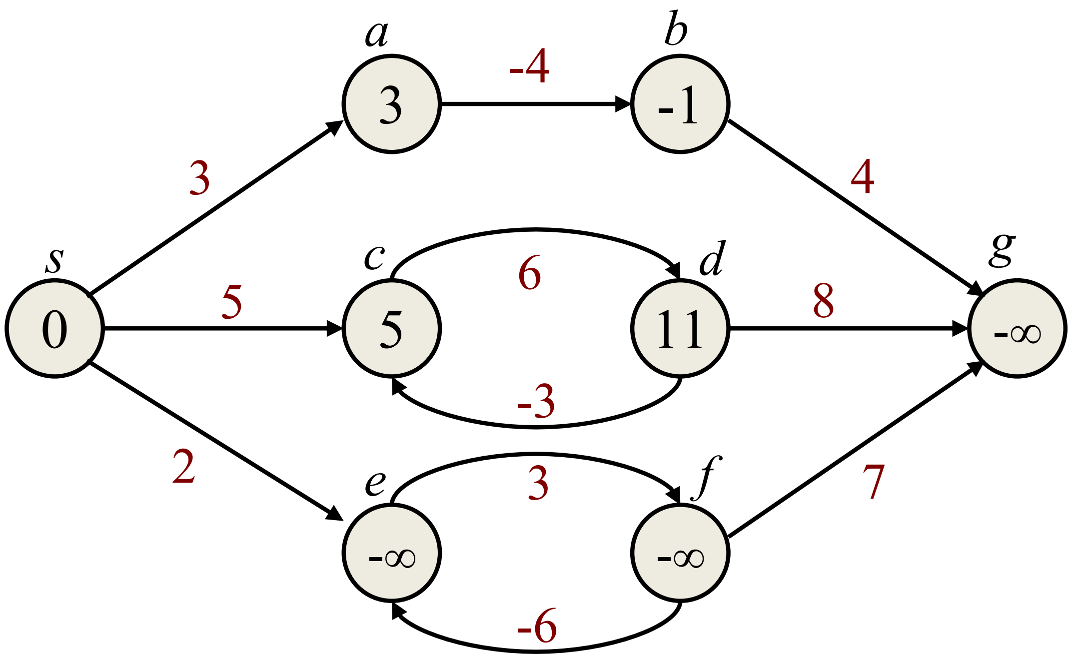
- Definition
- Dijkstra's algorithm
- The Bellman-Ford algorithm
- Single-source shortest paths in directed acyclic graphs

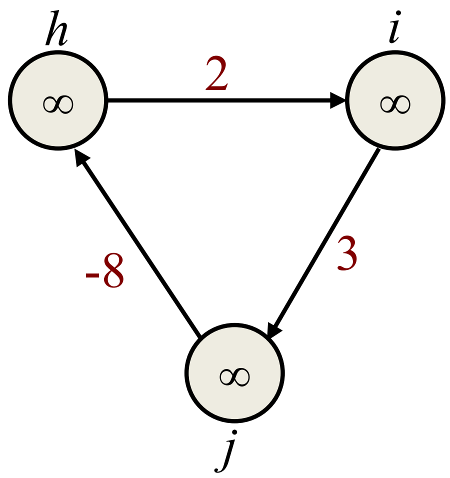
- ***Edge weight***
- ***Path weight***
  - The sum of all edge weights in the path.
- ***A Shortest path*** from u to v.
  - A path from u to v whose weight is the smallest.
  - Vertex u is the ***source*** and v is the ***destination***.
- ***The Shortest-path weight*** from u to v.
  - The weight of a shortest-path from u to v
  - $\delta(u,v)$

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

  - An algorithm for single-source (& all destinations) problem can be used to solve all the other problems.

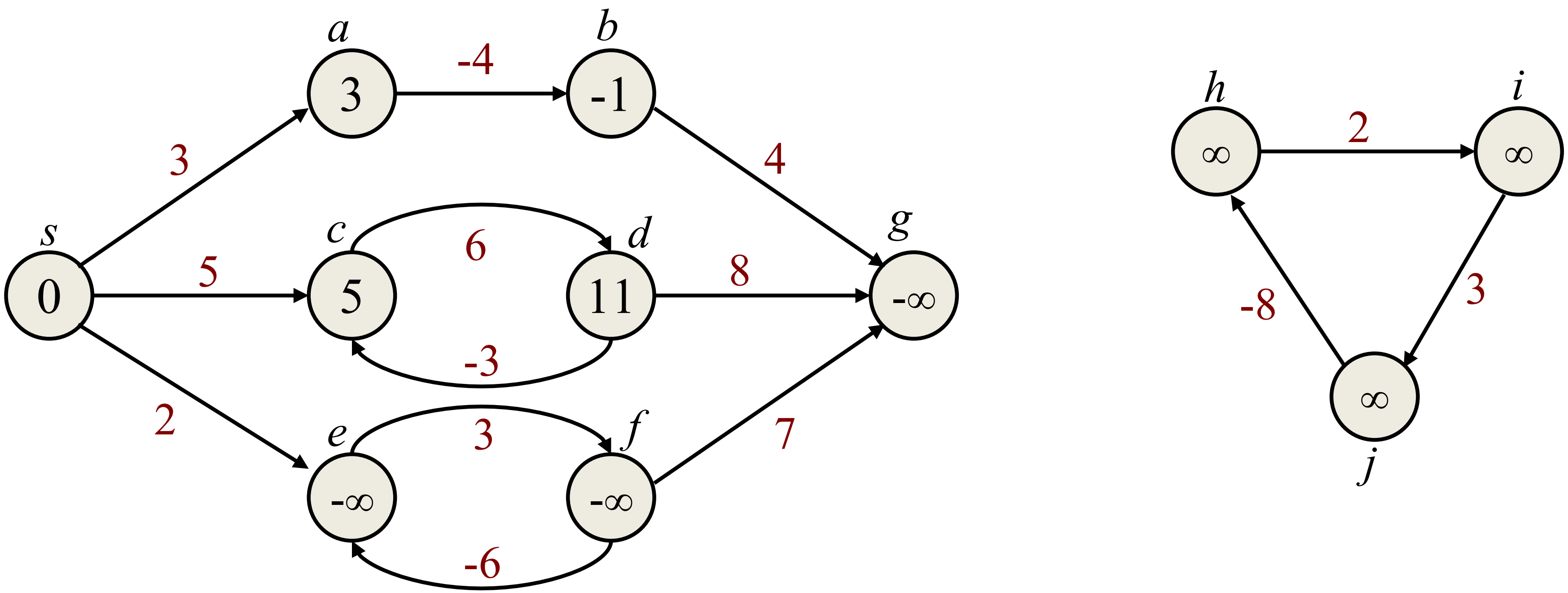
- What is a shortest path from s to g?



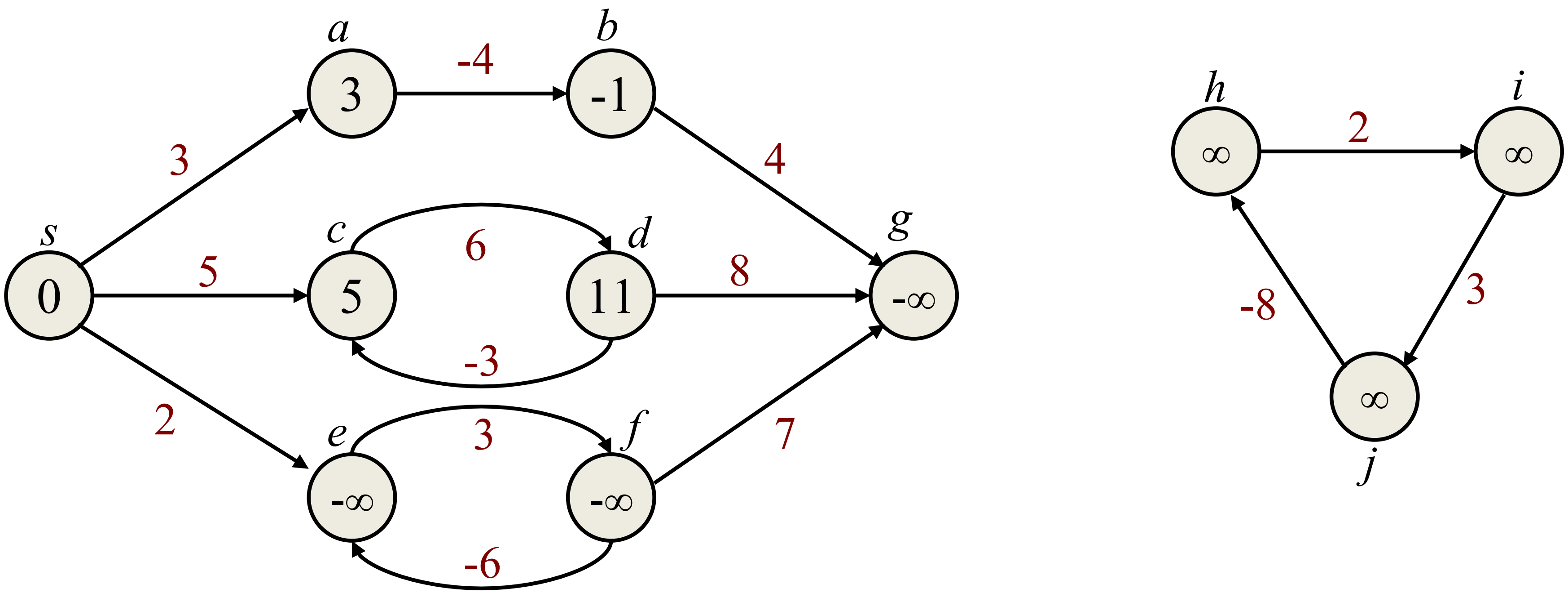


- Do all negative-weight edges cause a problem?

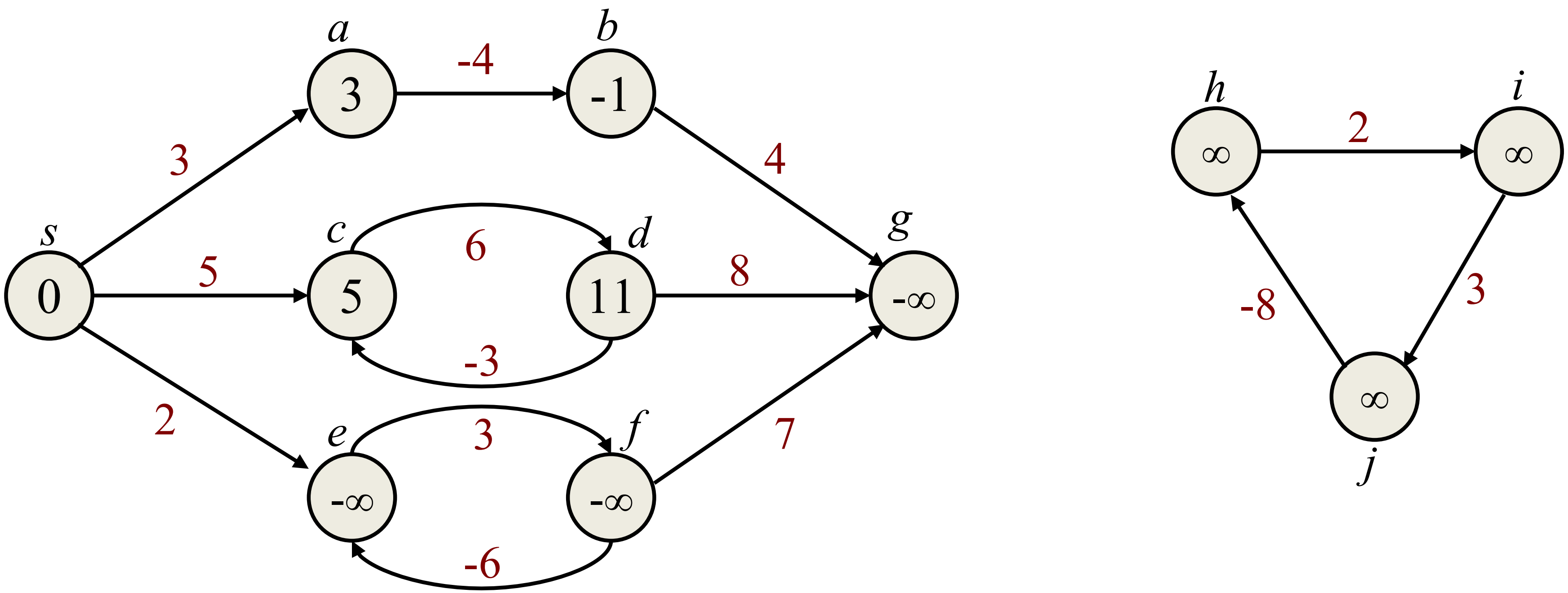
- Do all negative-weight cycles cause a problem?



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

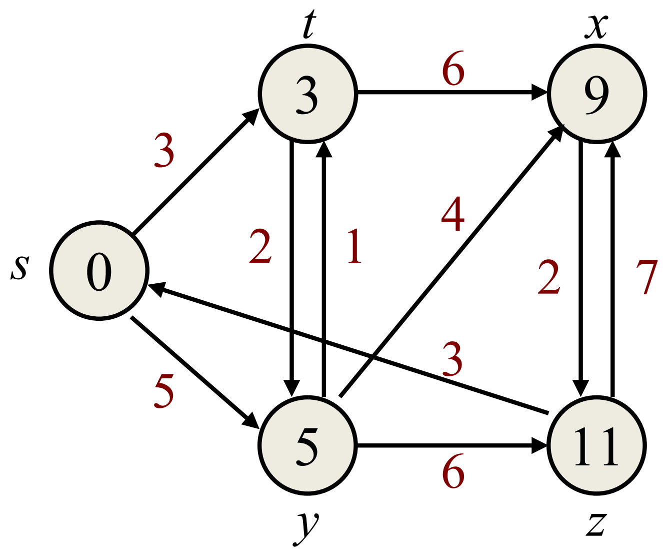


- Single-source shortest paths can be defined if there are not any ***negative-weight cycles reachable from the source.***

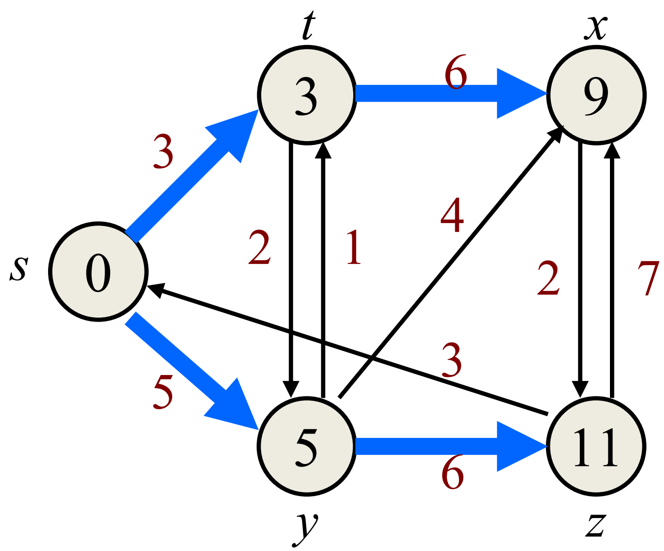


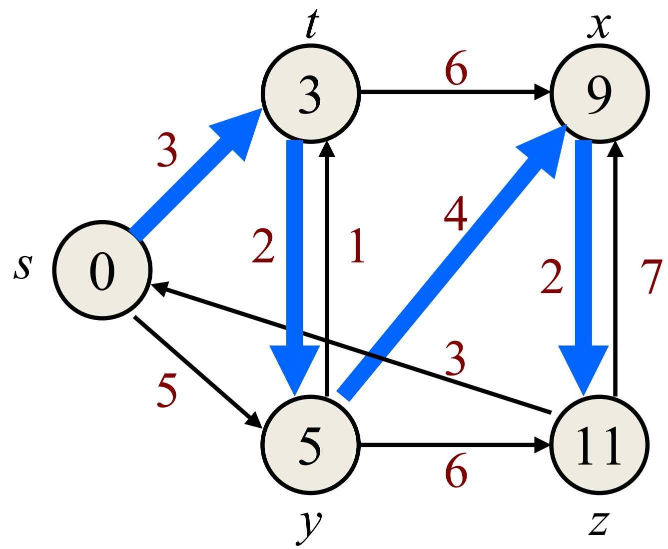
- **Cycles**
  - There is a shortest path that does not include cycles.
  - A shortest-path length is at most $|V| - 1$.

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

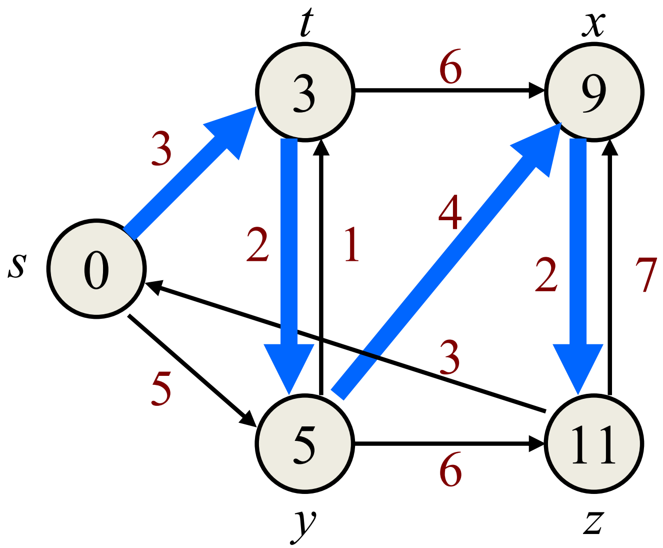


Predecessor subgraph





Predecessor subgraph



t: $s \rightarrow t$

y: $s \rightarrow t \rightarrow y$

x: $s \rightarrow t \rightarrow y \rightarrow x$

z: $s \rightarrow t \rightarrow y \rightarrow x \rightarrow z$

$O(V^2)$ space

t: s

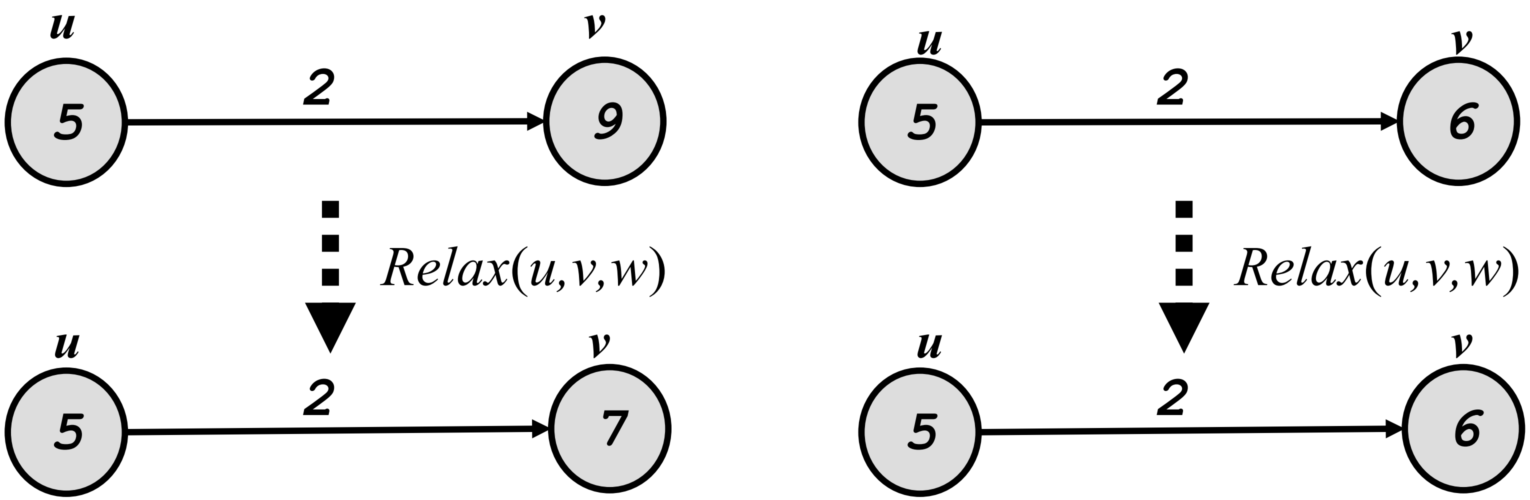
y: t

x: y

z: x

$O(V)$ space

- **Relaxation**



RELAX(*u*, *v*, *w*)

1 **if** $d[v] > d[u] + w[u, v]$

2 **then** $d[v] \leftarrow d[u] + w[u, v]$

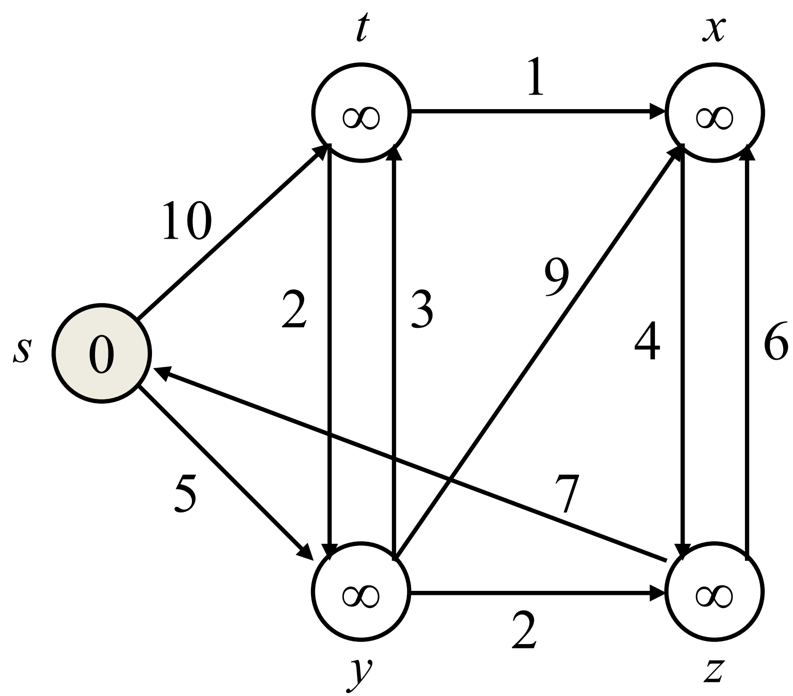
3 $\pi[v] \leftarrow u$

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

```
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)
```

Q

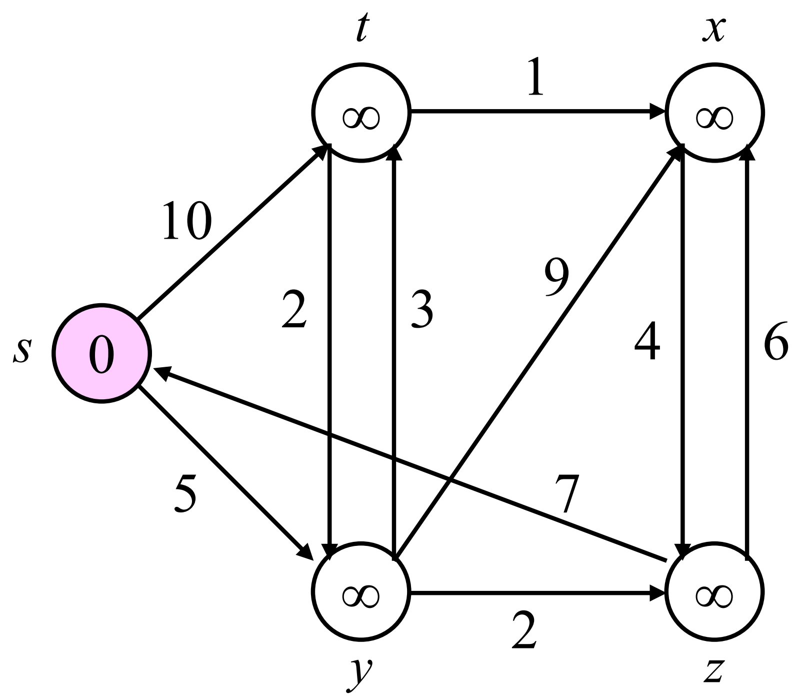
s	t	y	x	z
0	∞	∞	∞	∞



S

Q

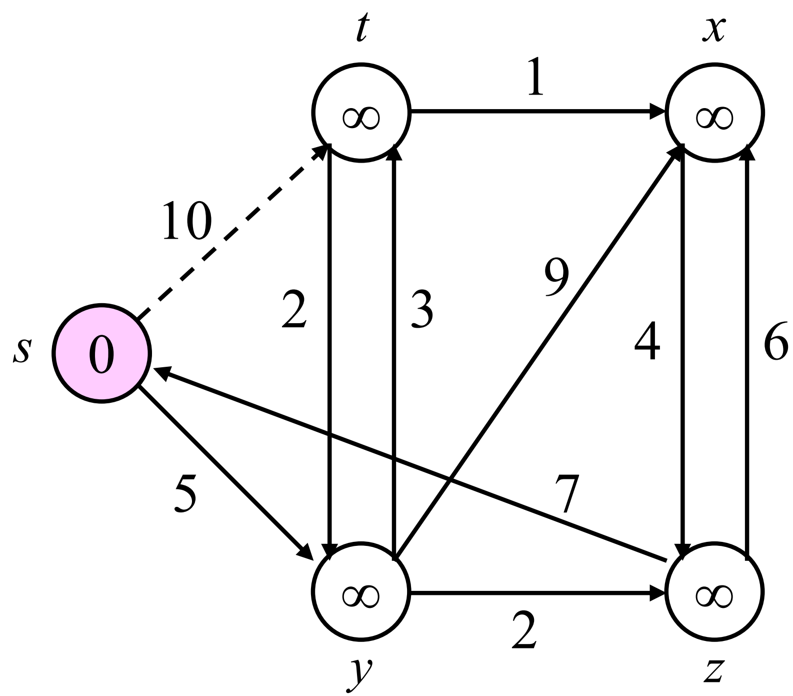
s	t	y	x	z
0	∞	∞	∞	∞



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

Q

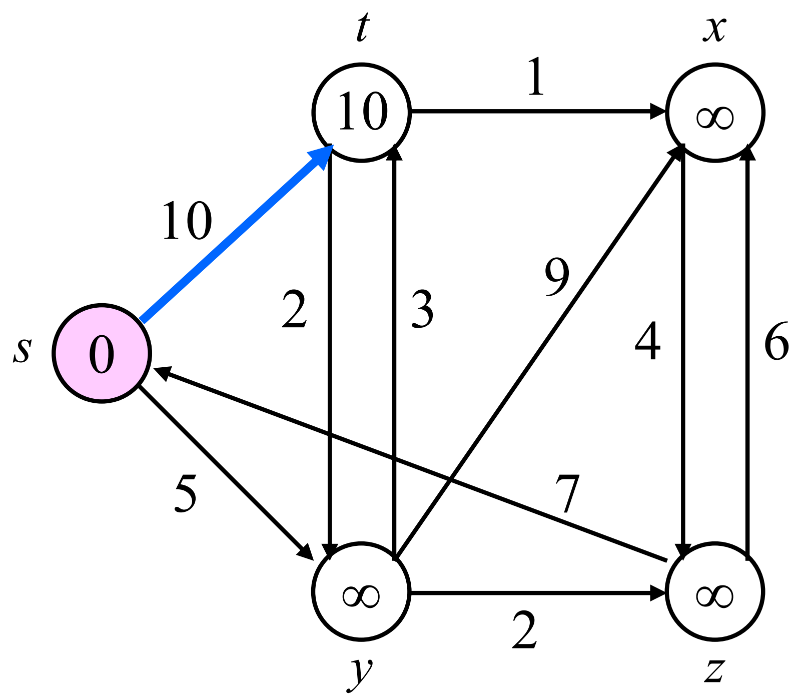
s	t	y	x	z
0	∞	∞	∞	∞



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

Q

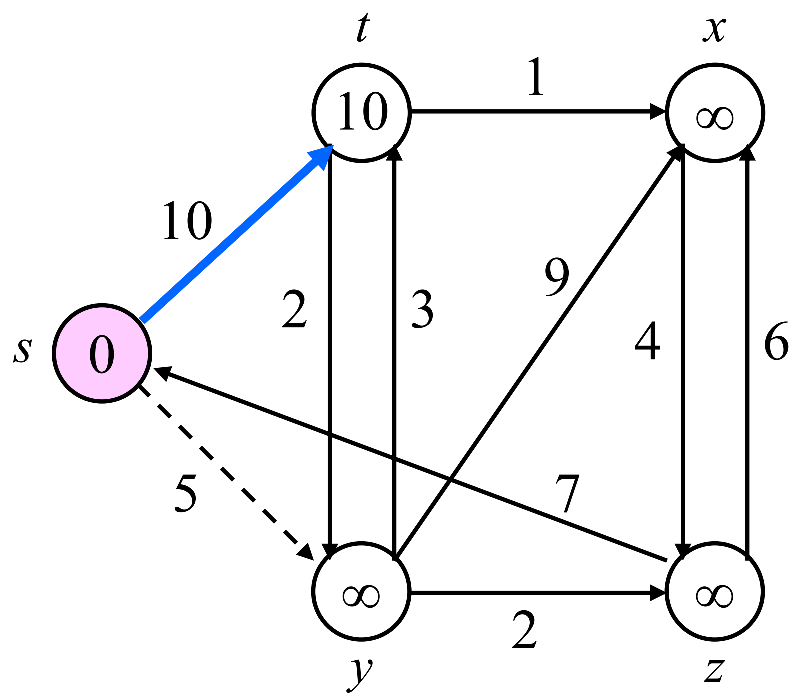
s	t	y	x	z
0	∞	∞	∞	∞
	10	-	-	-



$S = \{s\}$

Q

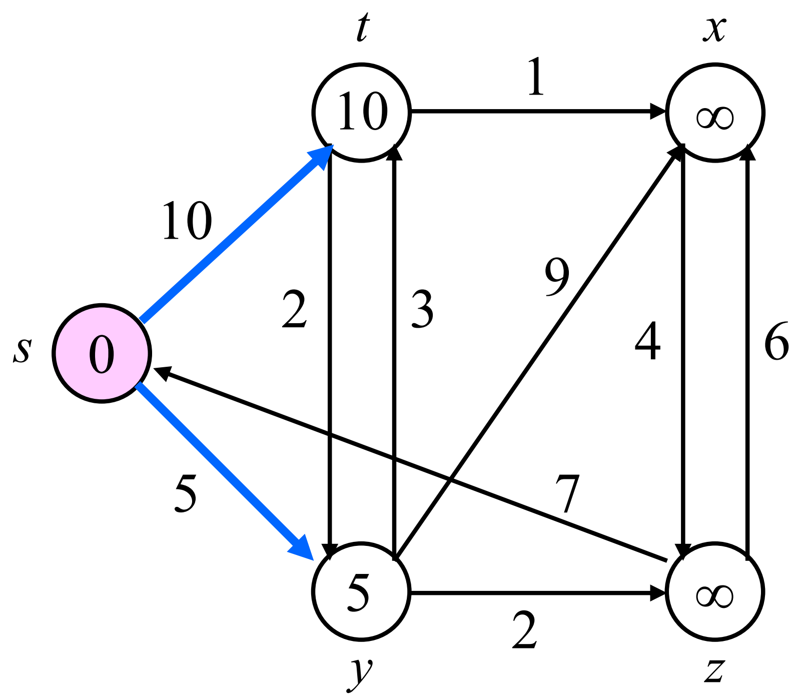
s	t	y	x	z
0	∞	∞	∞	∞
	10	-	-	-



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

Q

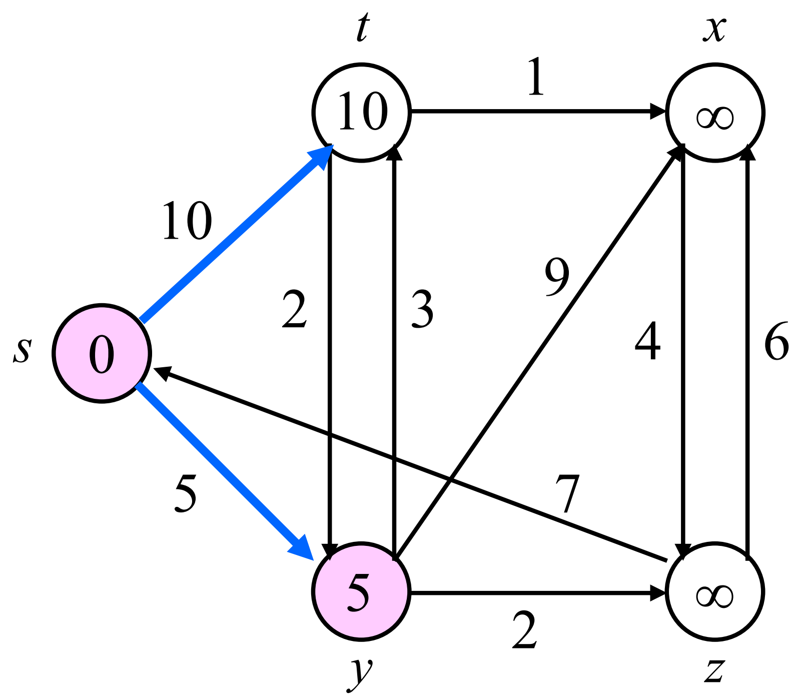
s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-



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

Q

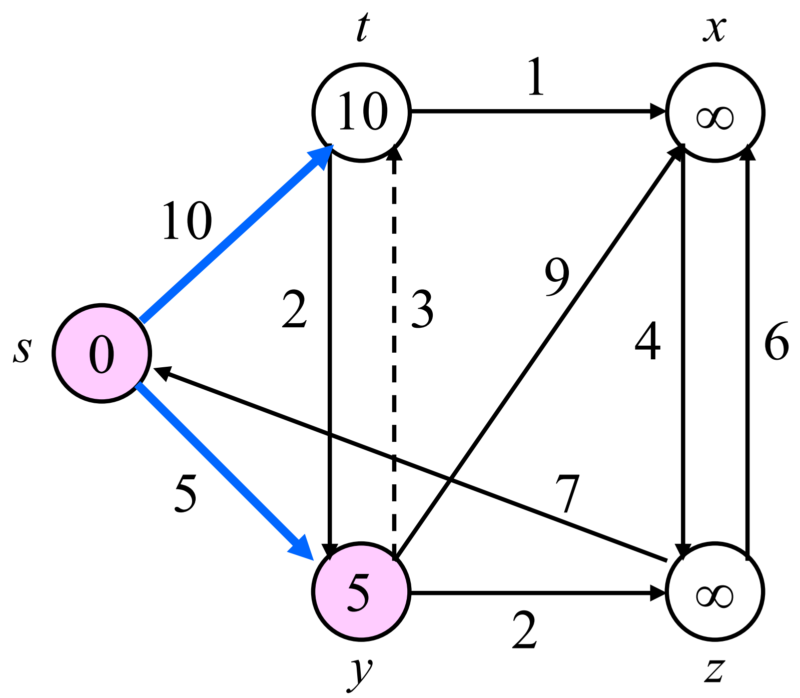
s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-



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

Q

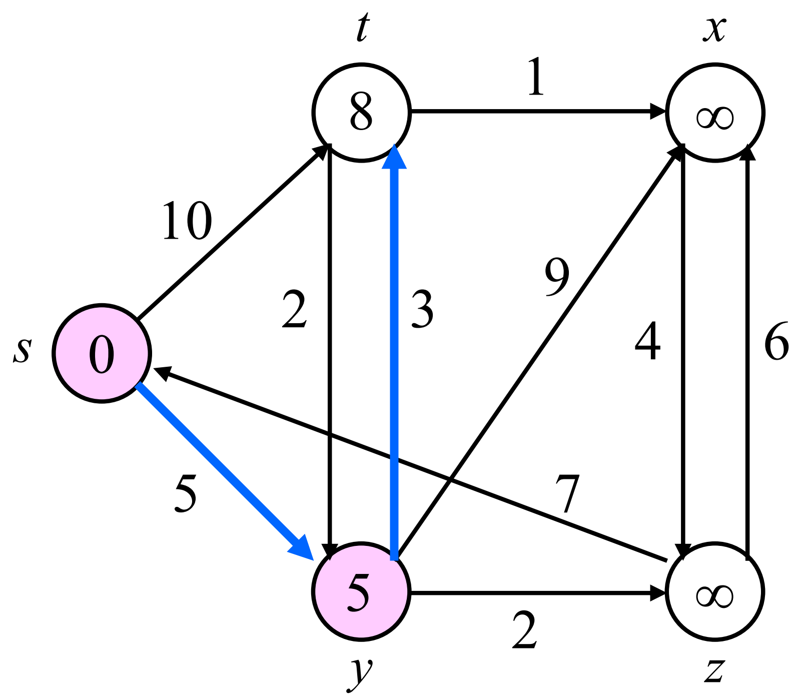
s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	5	-	-



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

Q

s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-
	8		-	-

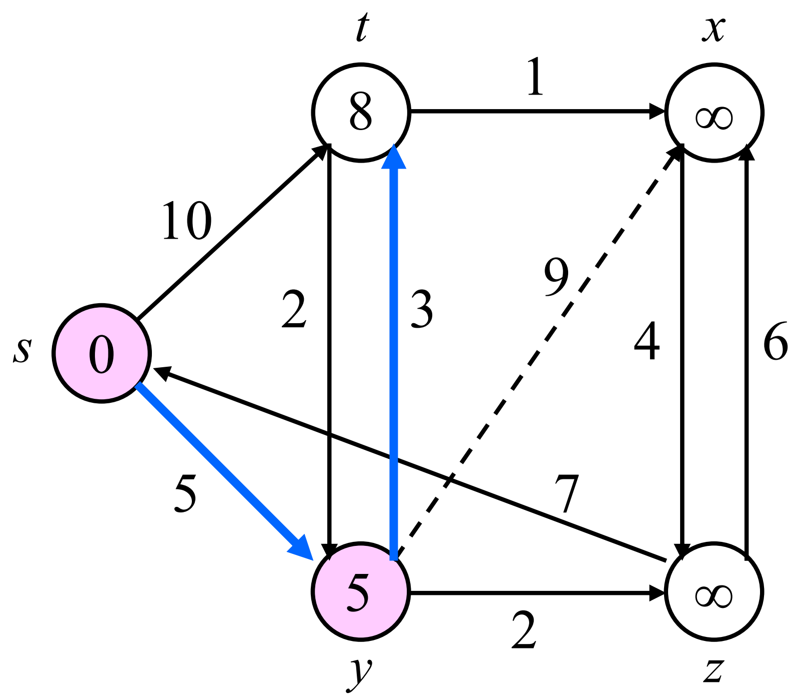


$\boldsymbol{S} = \{s, y\}$

Dijkstra's Algorithm

Q

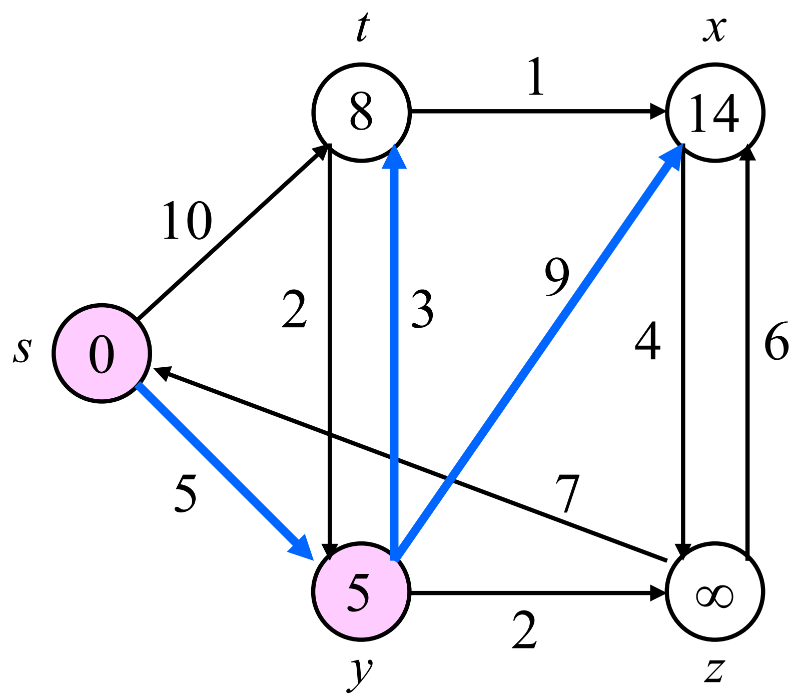
s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-
	8		-	-



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

Q

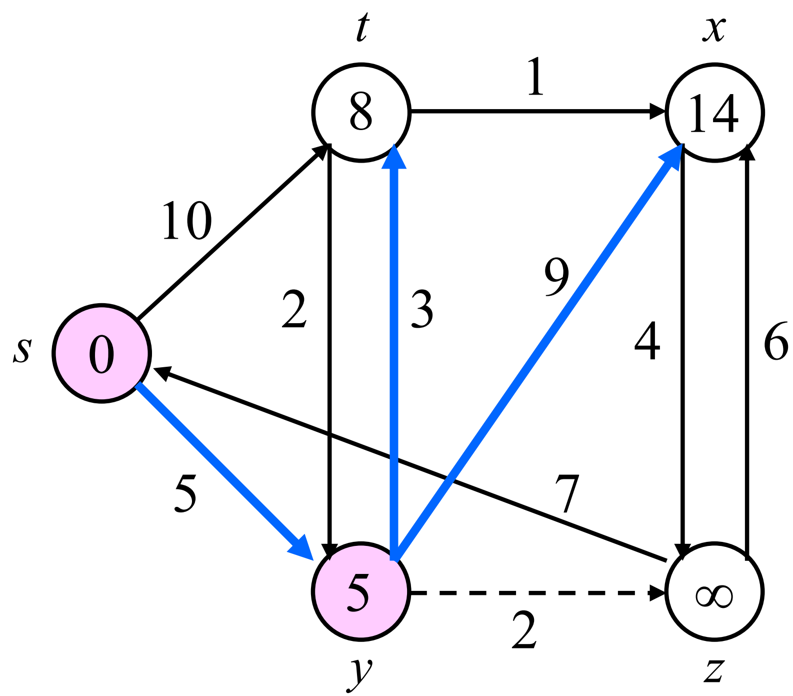
s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	5	-	-
	8		14	-



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

Q

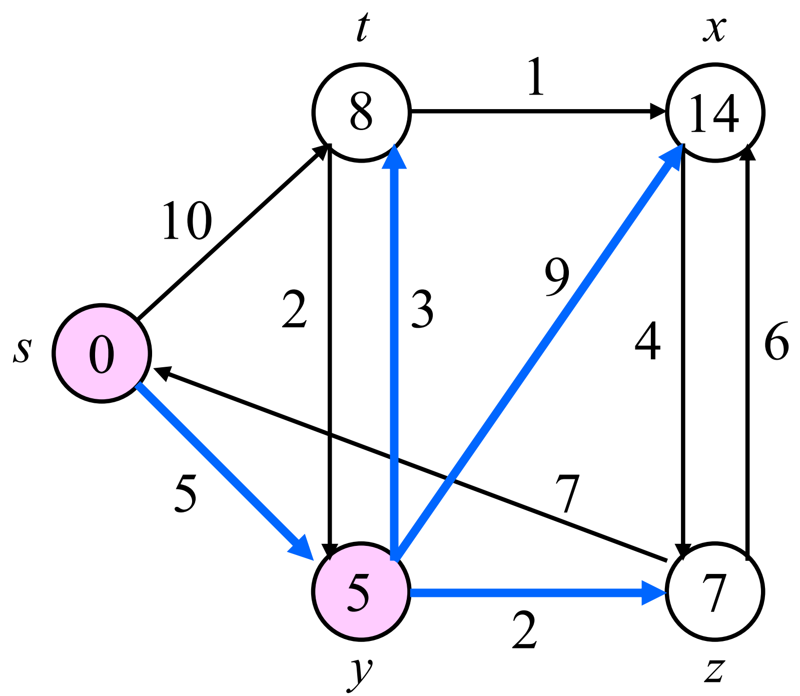
s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-
	8		14	-



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

Q

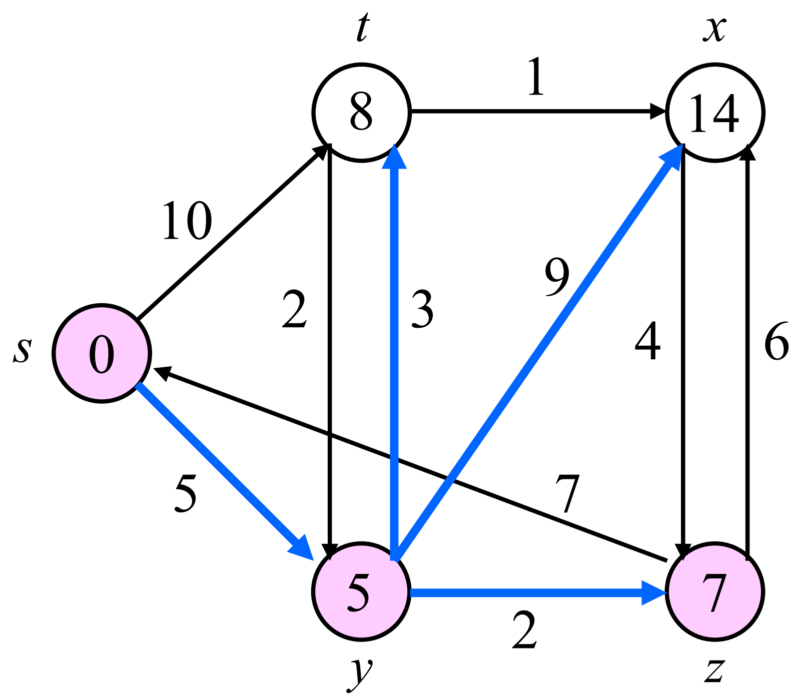
s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7



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

Q

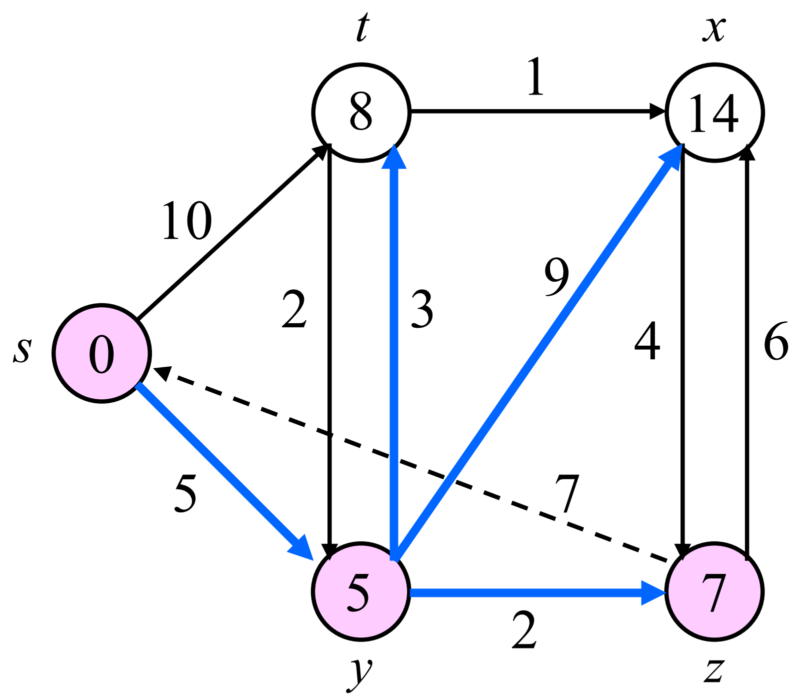
s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7



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

Q

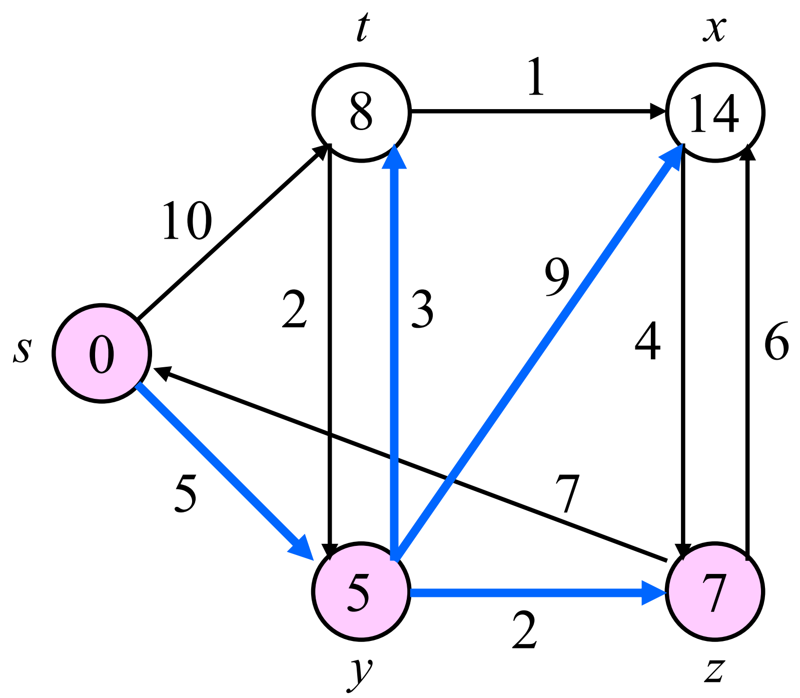
s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7



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

Q

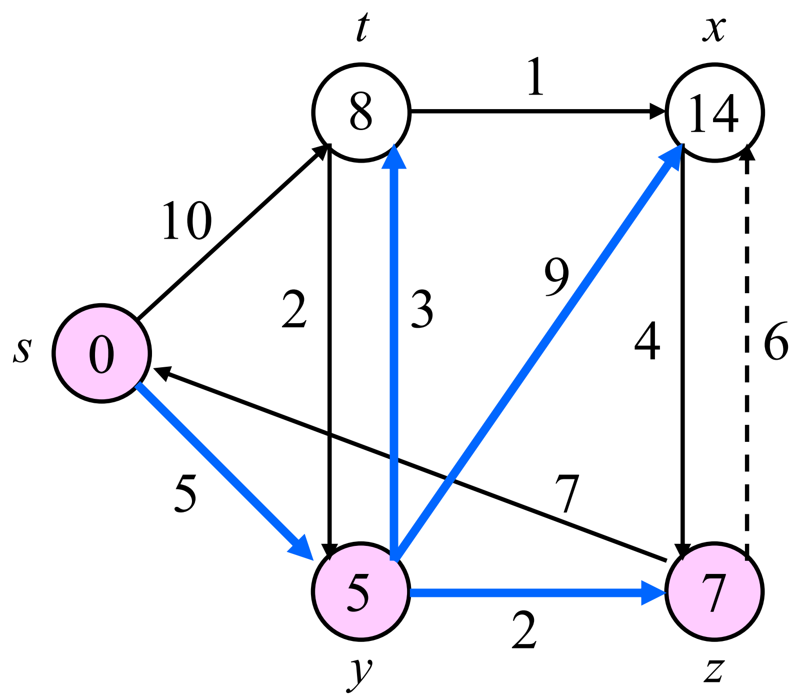
s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7



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

Q

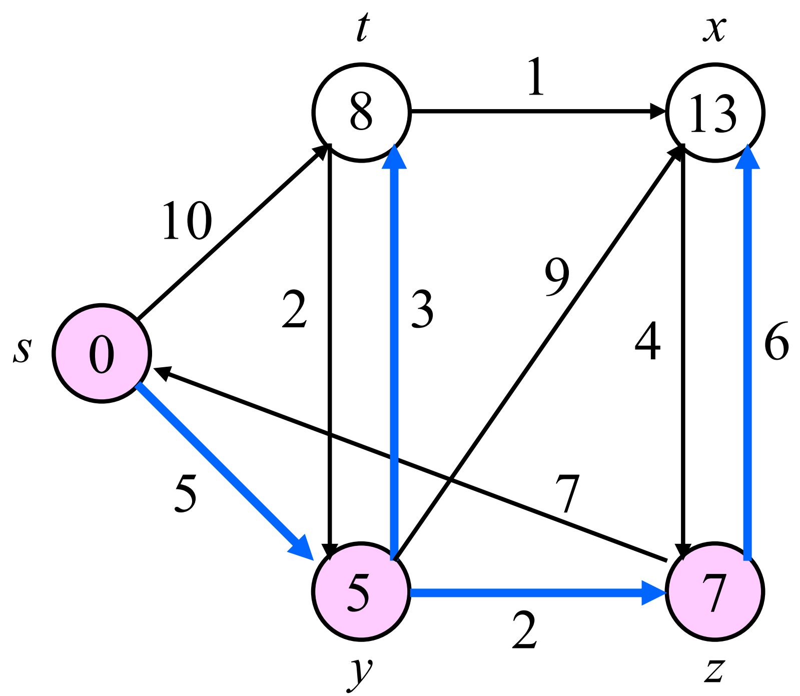
s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7



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

Q

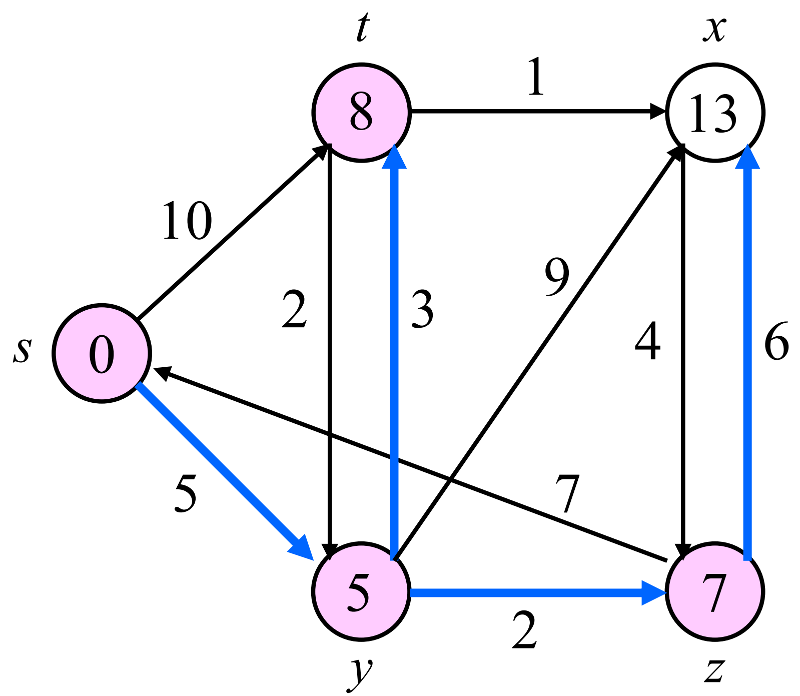
s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7
	8		13	



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

Q

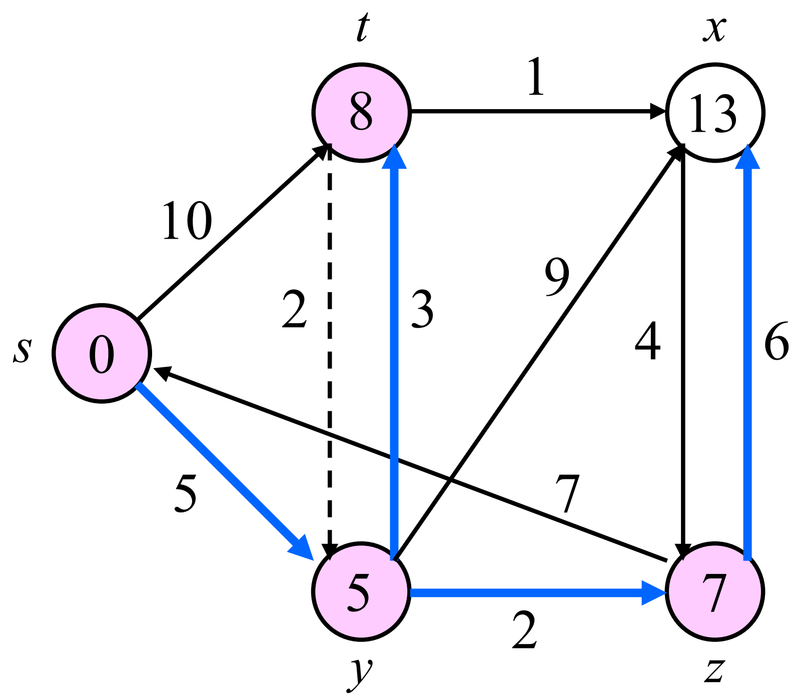
s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7
	8		13	



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

Q

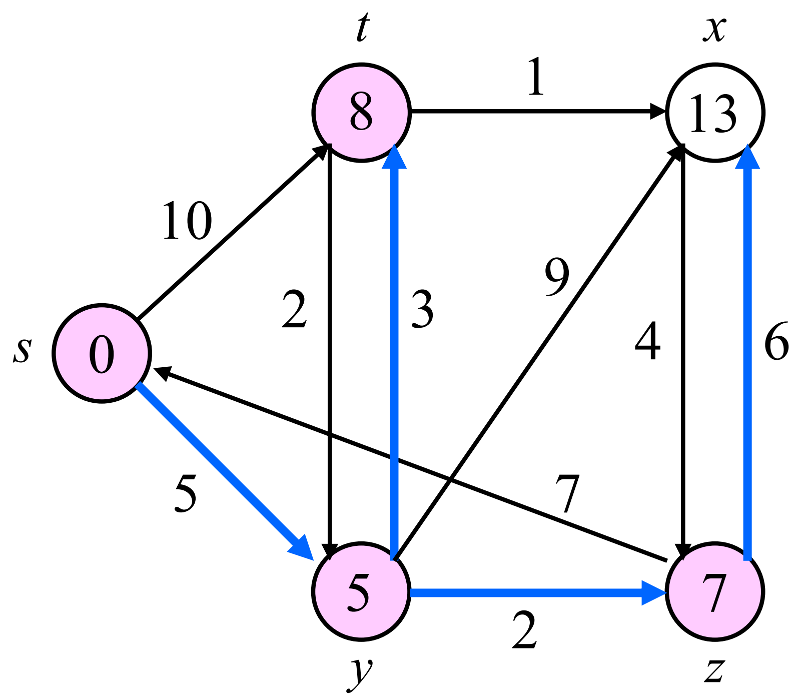
s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7
	8		13	



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

Q

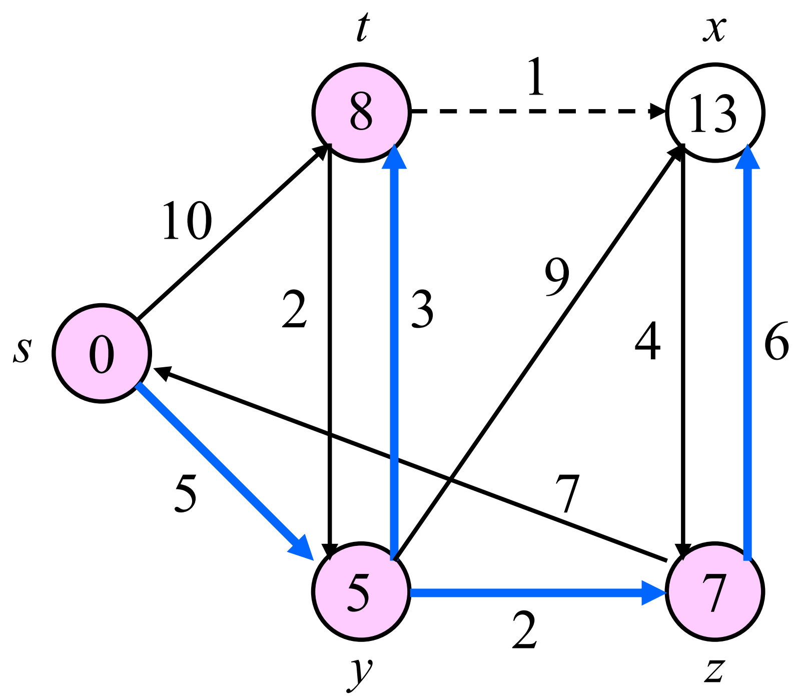
s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7
	8		13	



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

Q

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

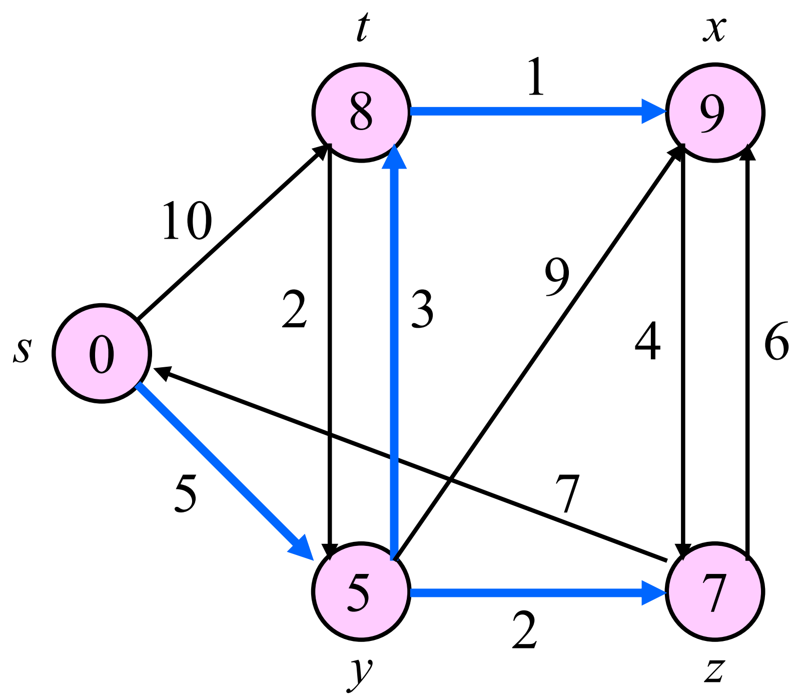


$\boldsymbol{S} = \{s, y, z, t\}$

Q

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

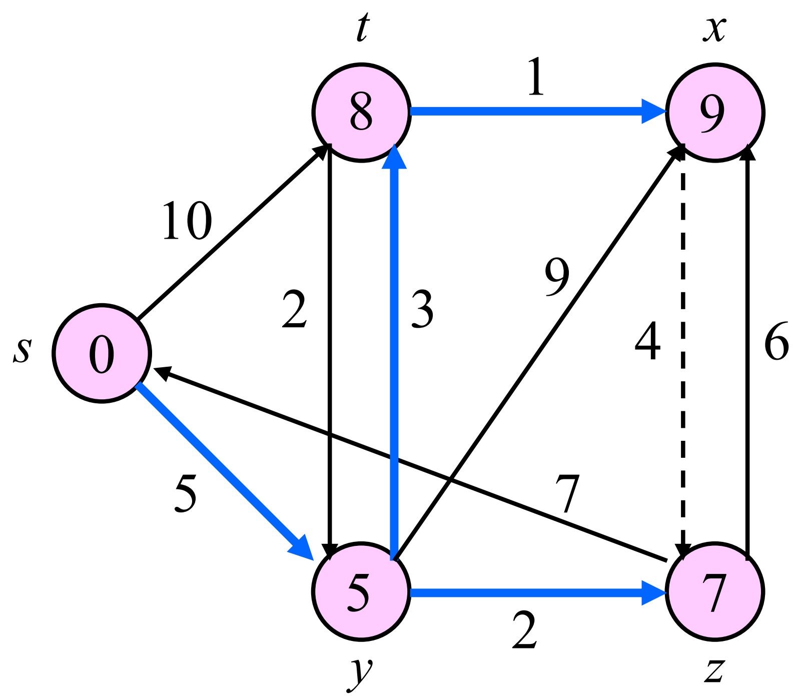
$\boldsymbol{S} = \{s, y, z, t, x\}$



Q

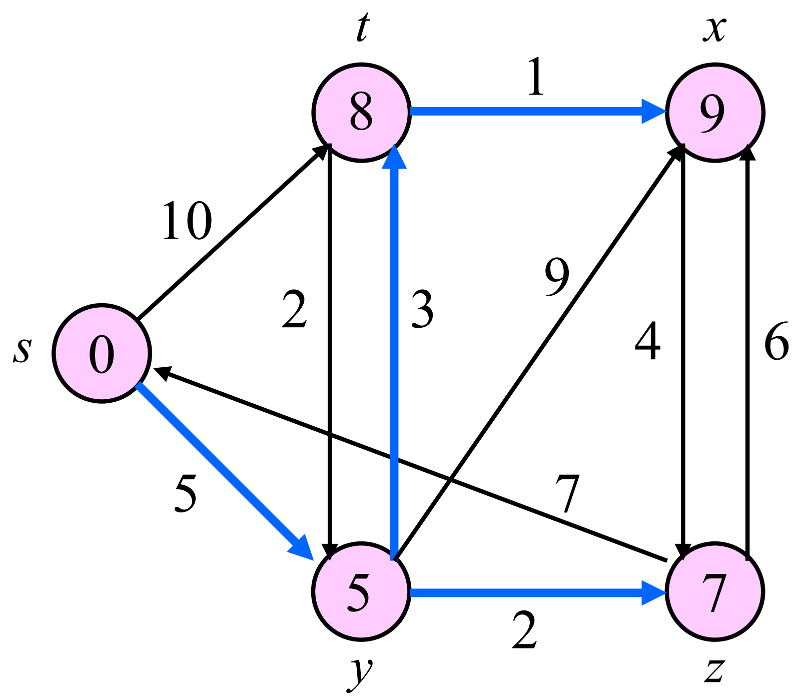
s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7
	8		13	
			9	

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



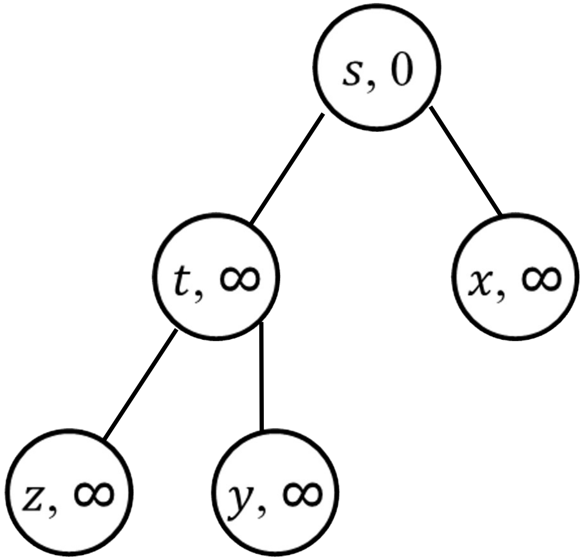
Q

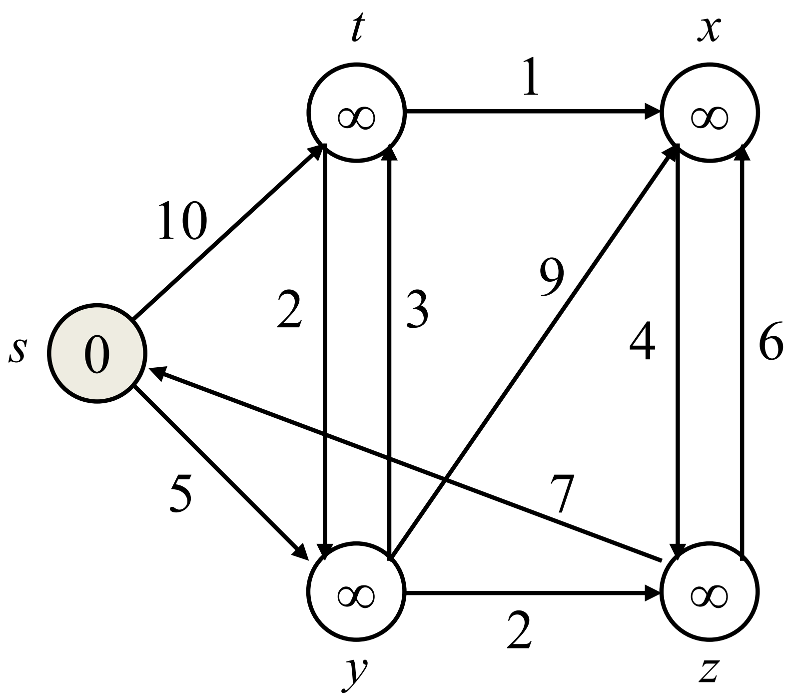
s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-
	8		14	7
	8		13	
			9	



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

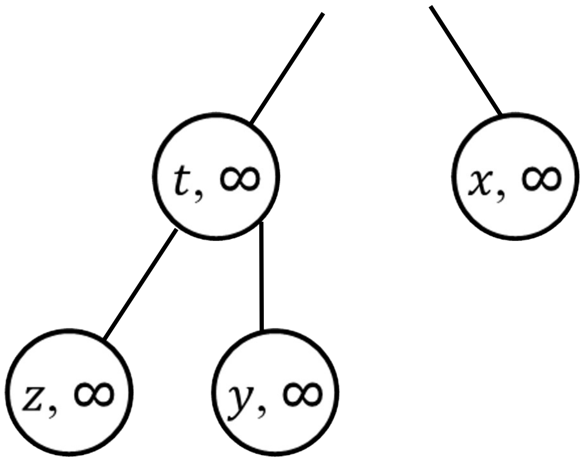
s	t	y	x	z

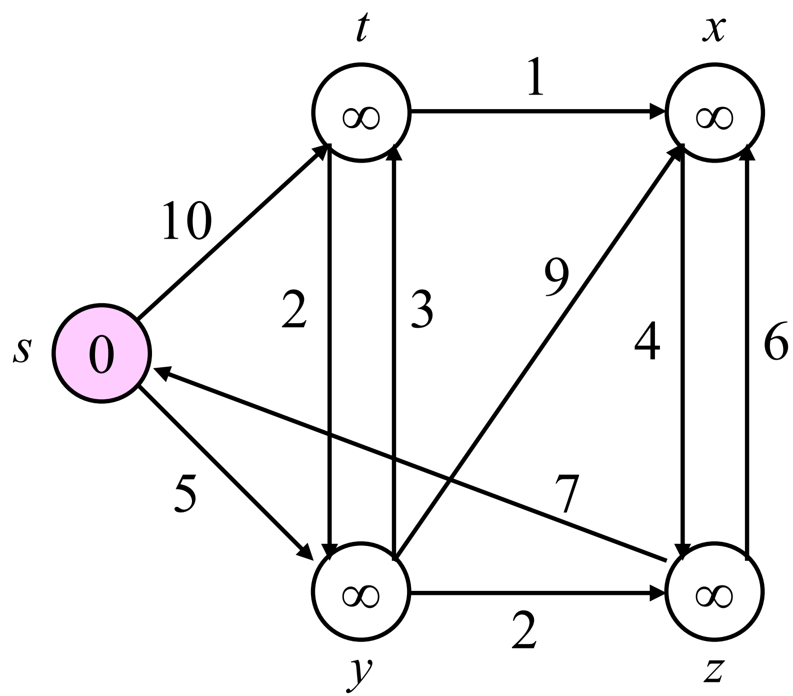




Dijkstra’s Algorithm

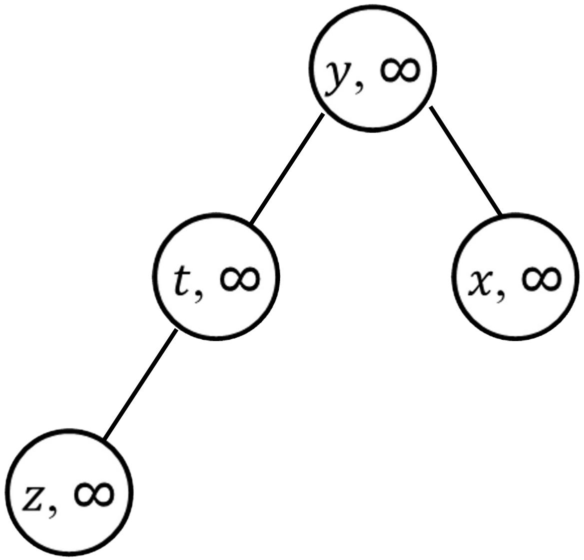
s	*t*	*y*	*x*	*z*

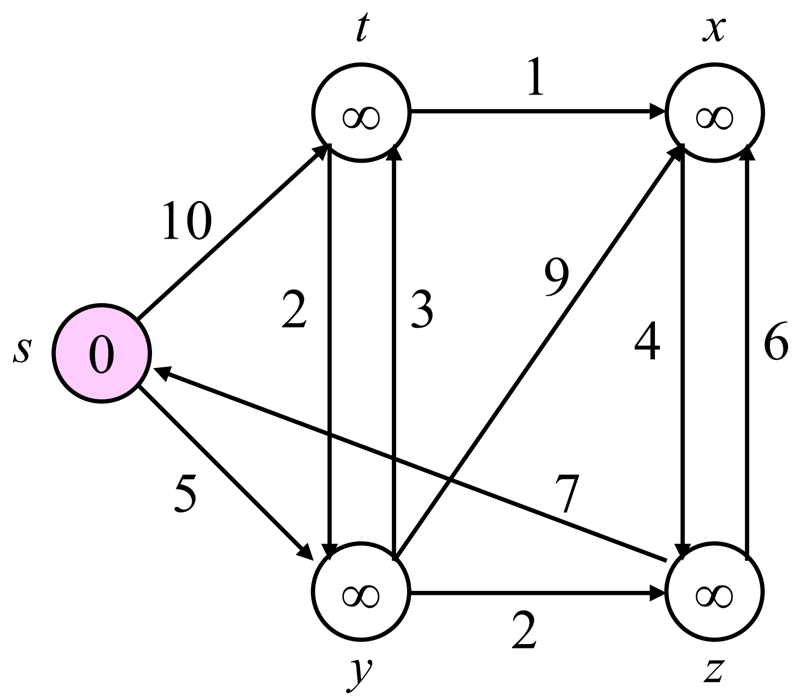




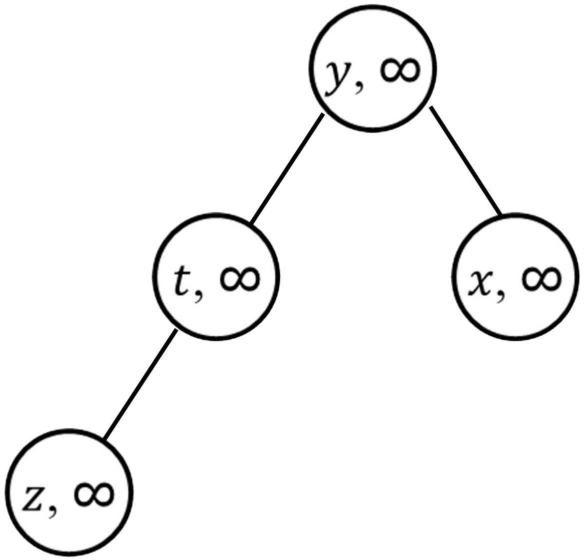
Dijkstra's Algorithm

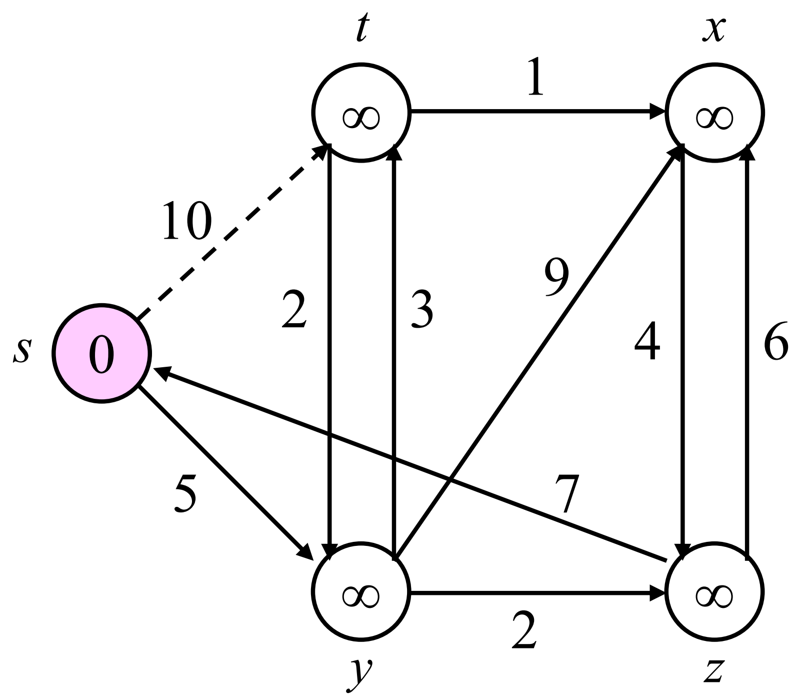
s	*t*	*y*	*x*	*z*



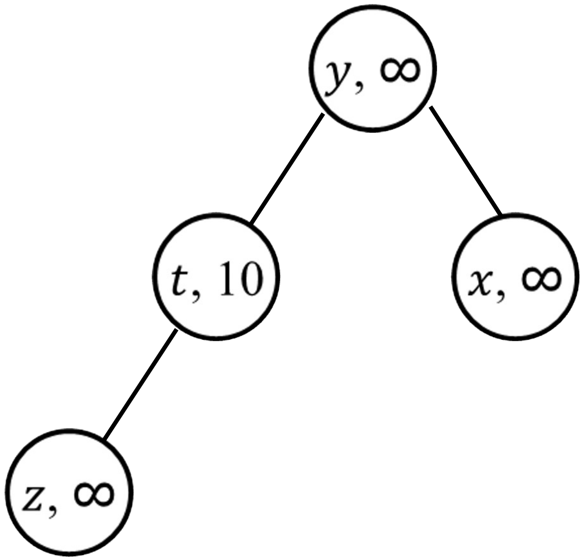


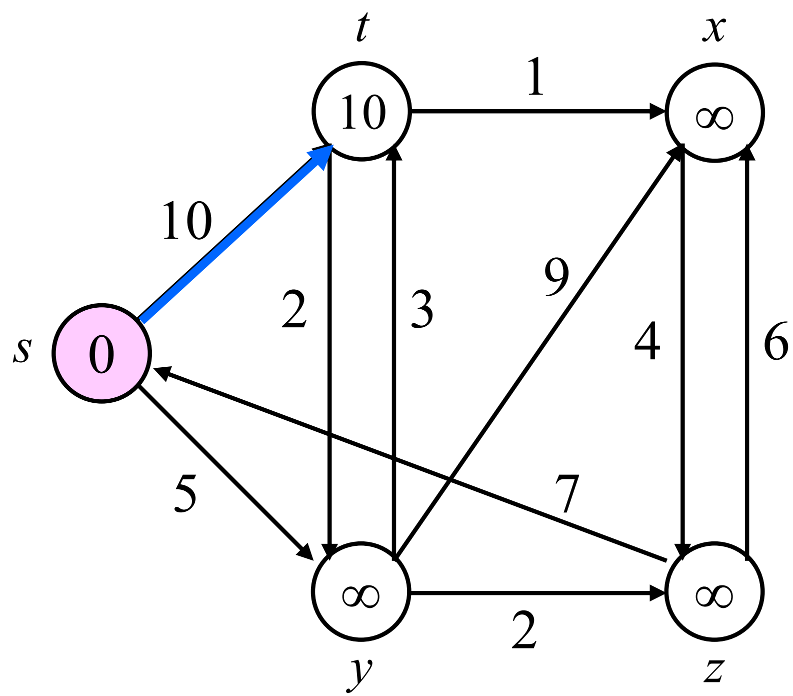
s	t	y	x	z



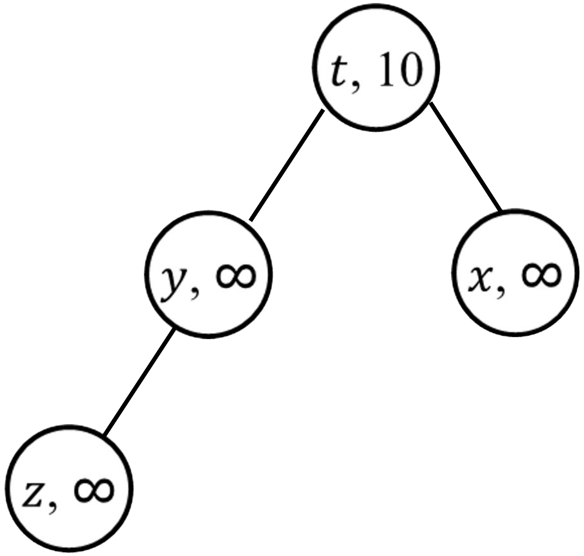


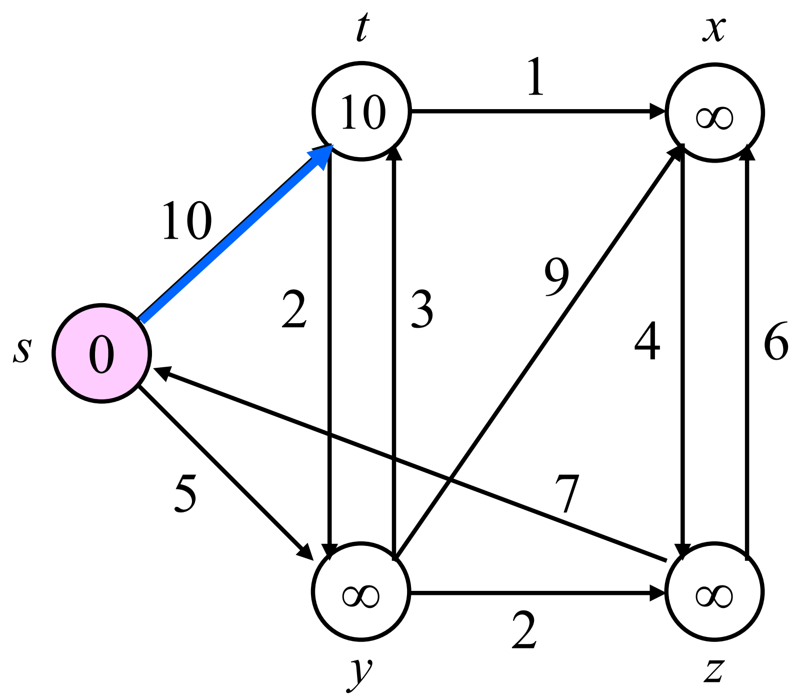
s	*t*	*y*	*x*	*z*
	s			



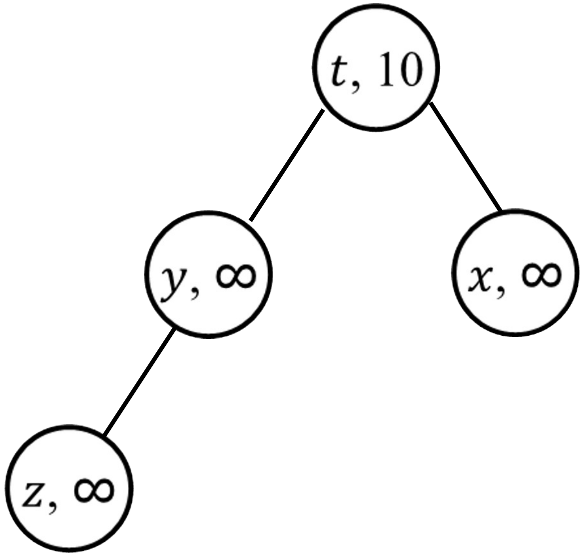


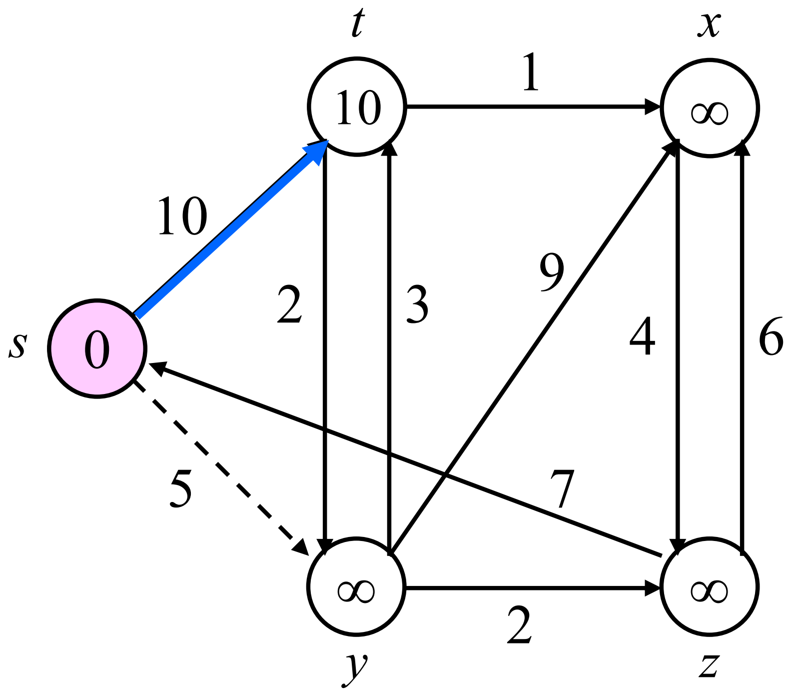
s	t	y	x	z
	s			



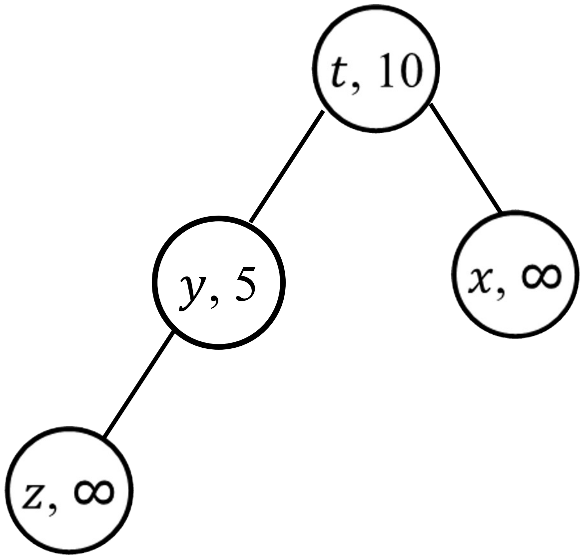


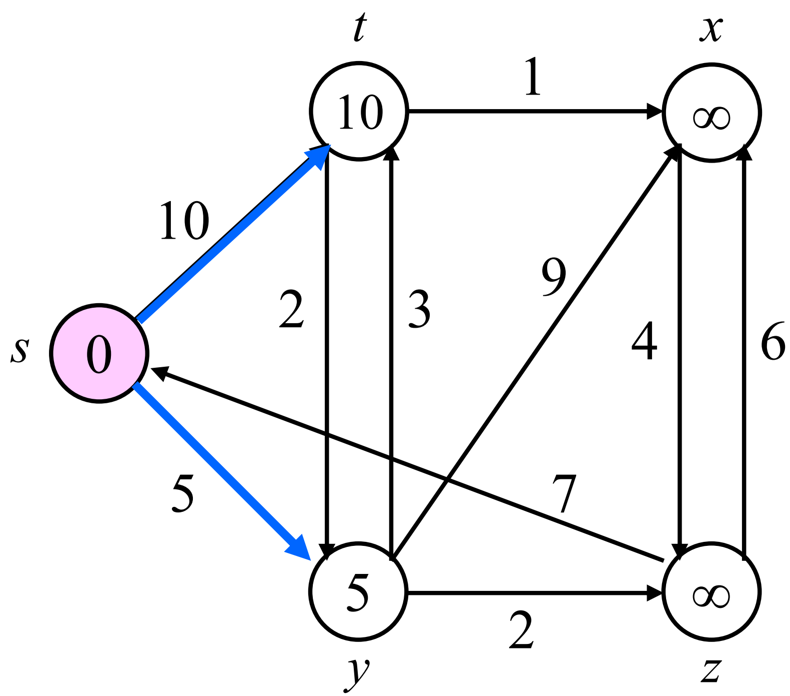
s	t	y	x	z
	s			



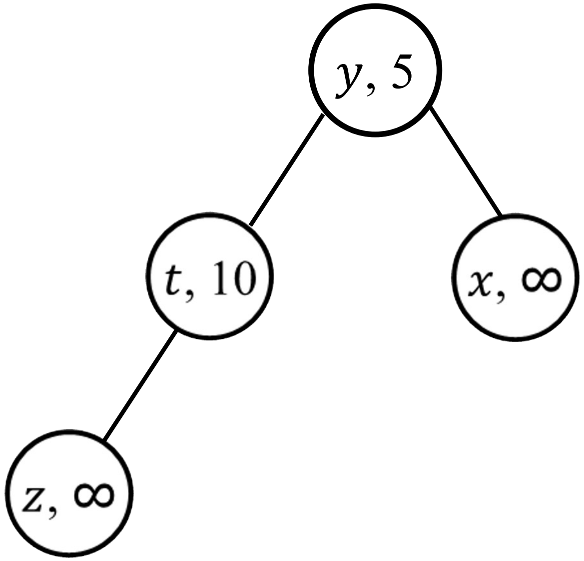


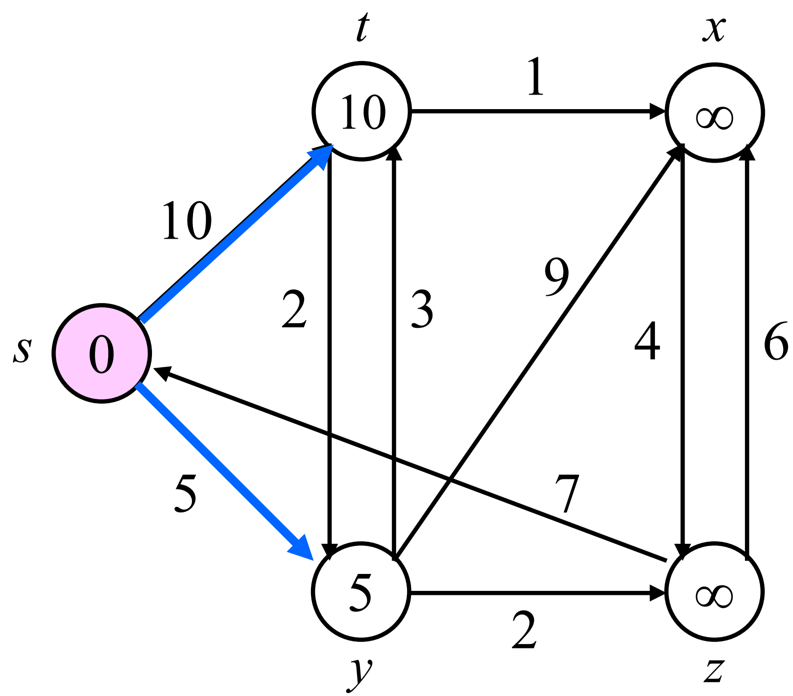
s	*t*	*y*	*x*	*z*
	s	*s*		



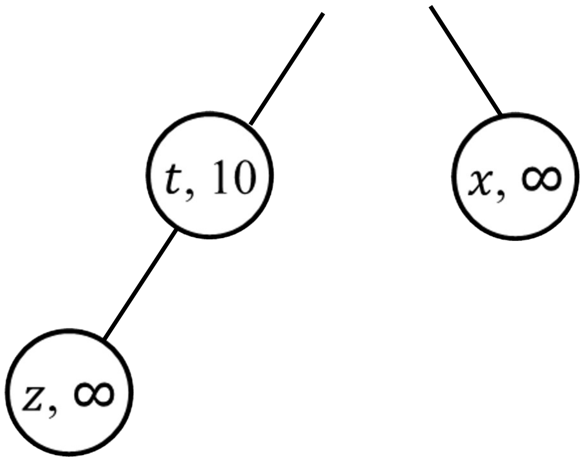


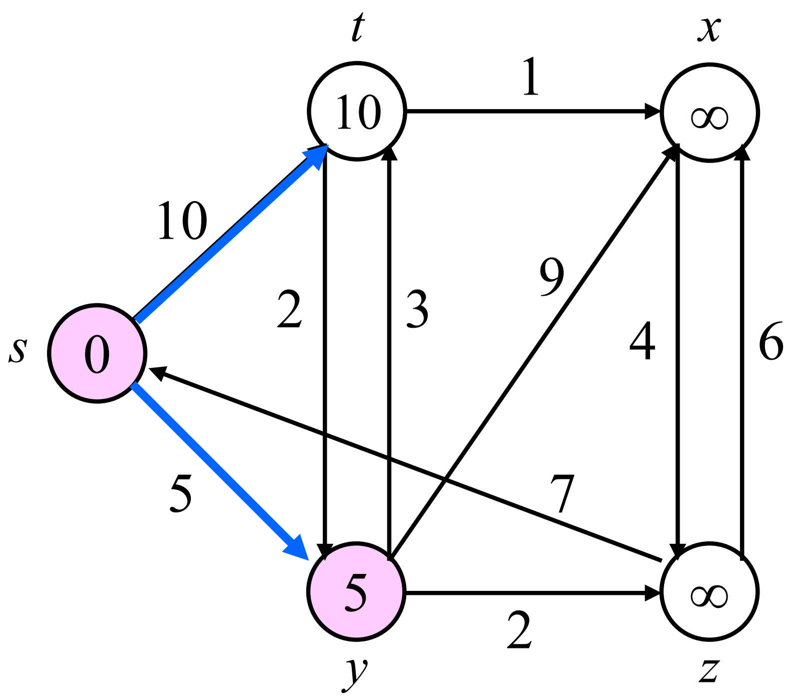
s	t	y	x	z
	s	s		



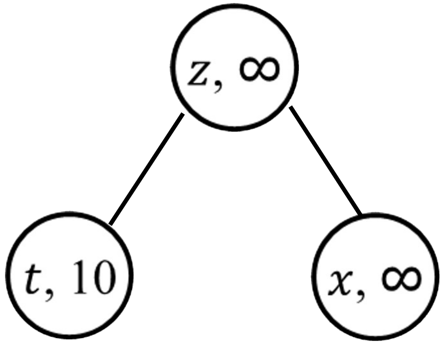


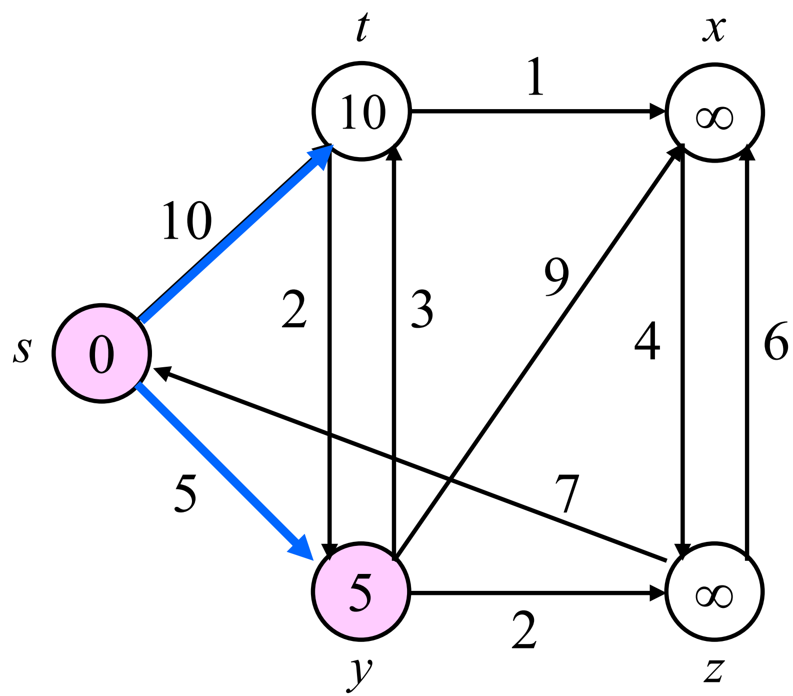
s	t	y	x	z
	s	s		





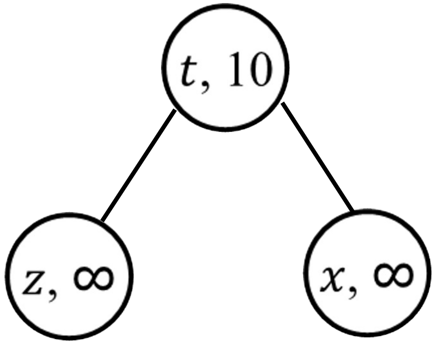
s	t	y	x	z
	s	s		

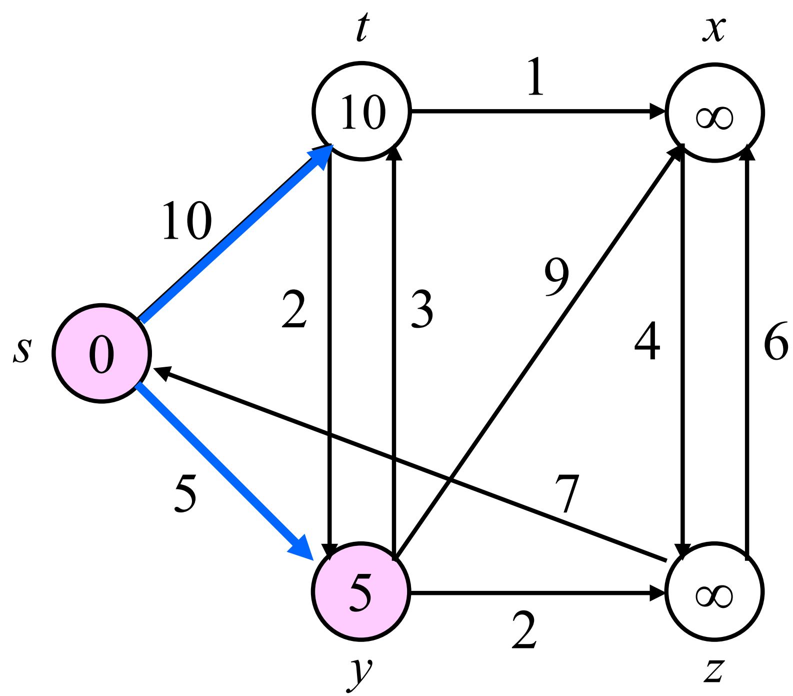




Dijkstra's Algorithm

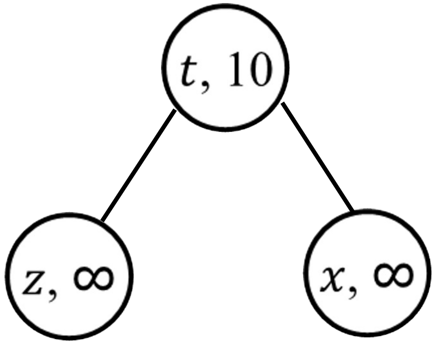
s	*t*	*y*	*x*	*z*
	s	*s*		

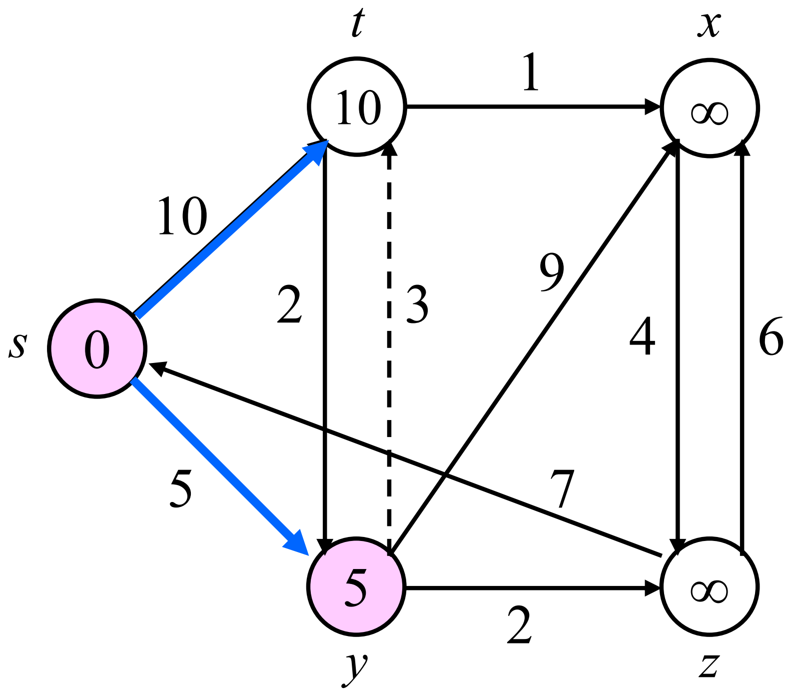




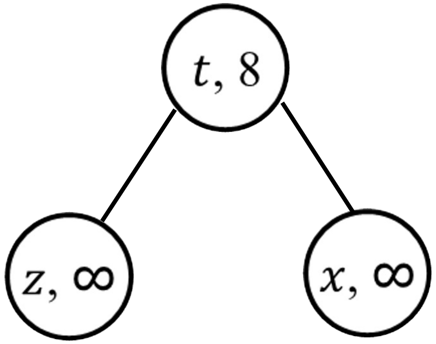
Dijkstra's Algorithm

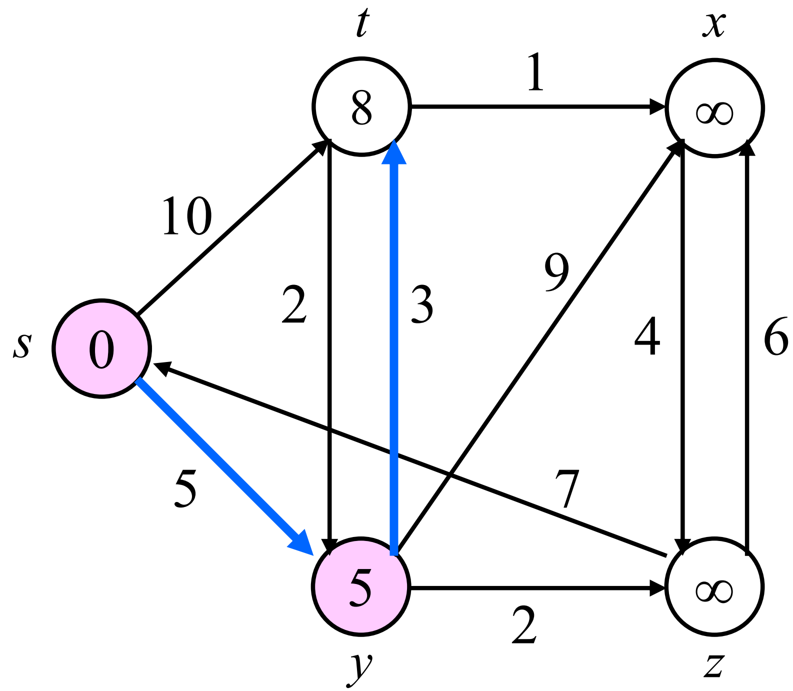
s	*t*	*y*	*x*	*z*
	s	*s*		





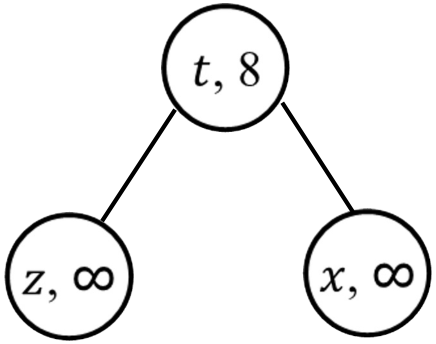
s	*t*	*y*	*x*	*z*
	y	*s*		

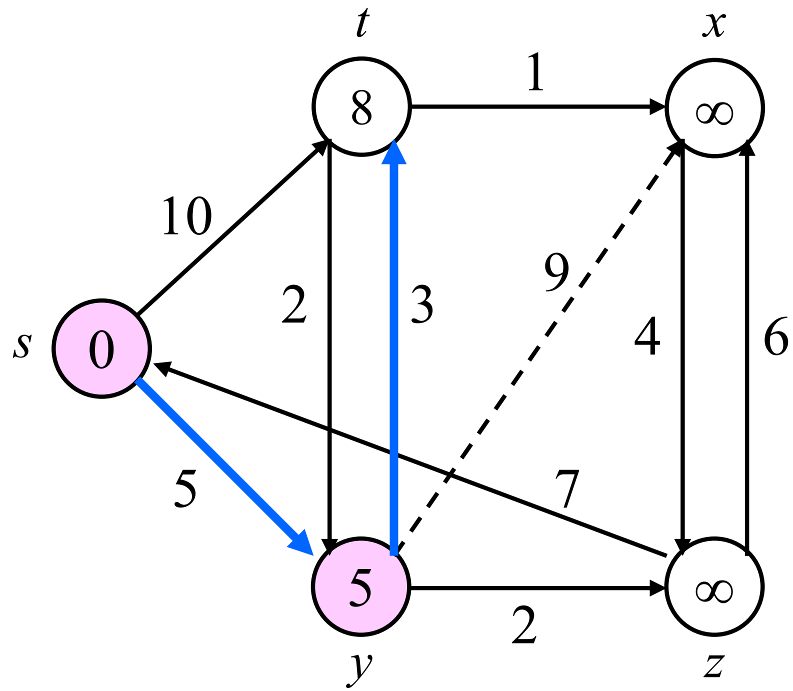




Dijkstra's Algorithm

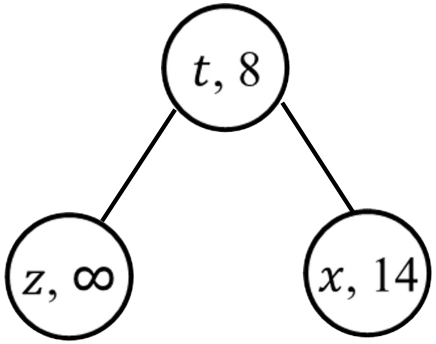
s	*t*	*y*	*x*	*z*
	y	*s*		

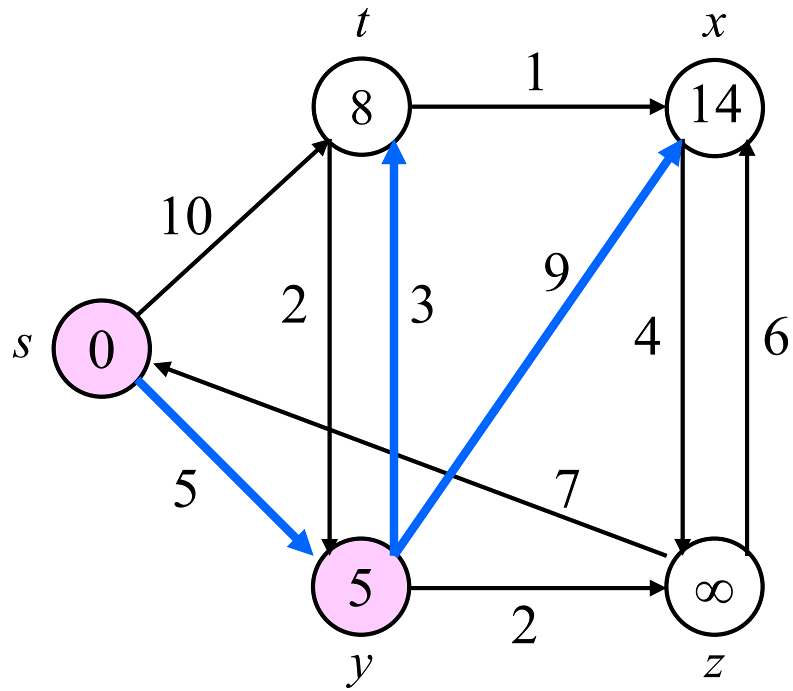




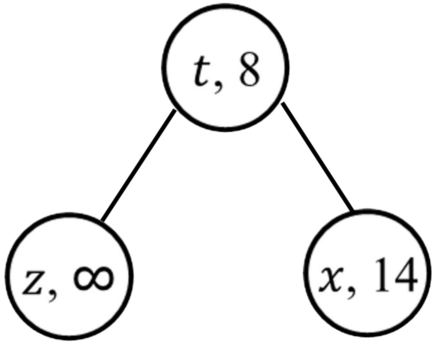
Dijkstra’s Algorithm

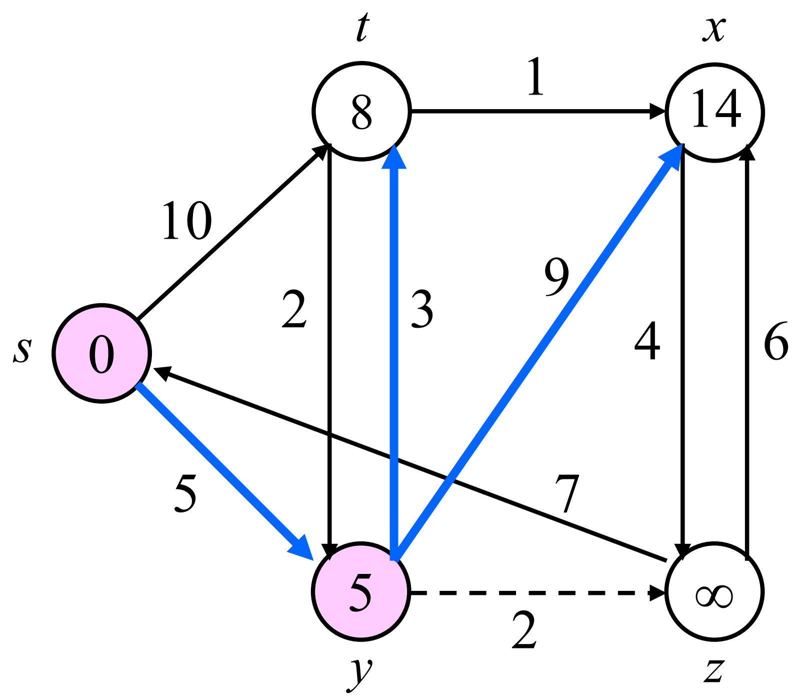
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	



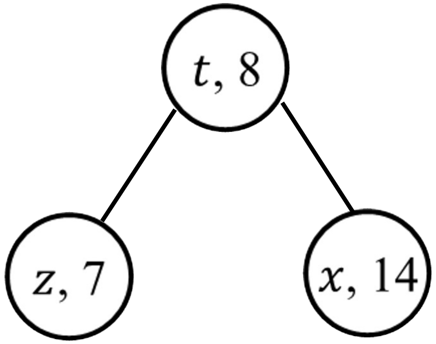


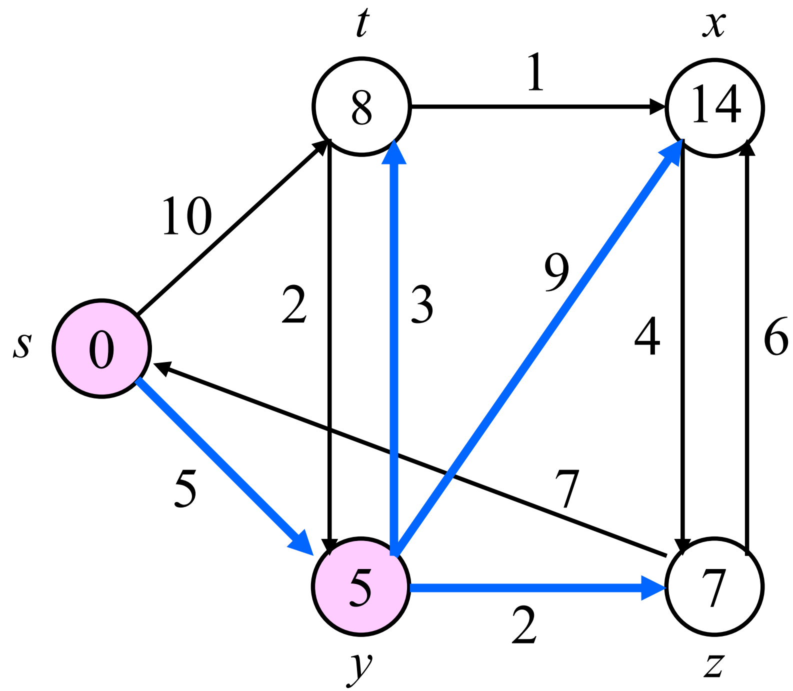
s	t	y	x	z
	y	s	y	



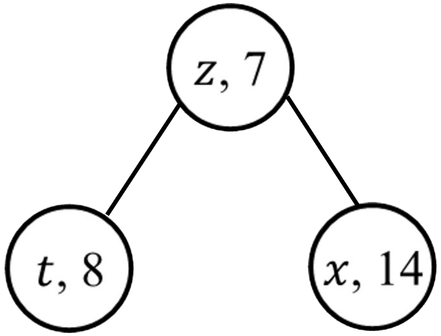


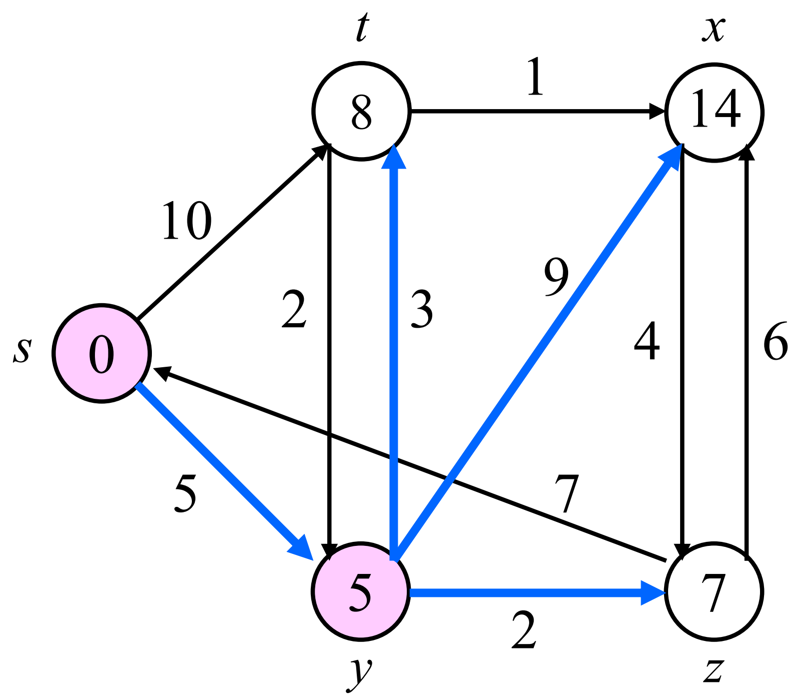
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	*y*



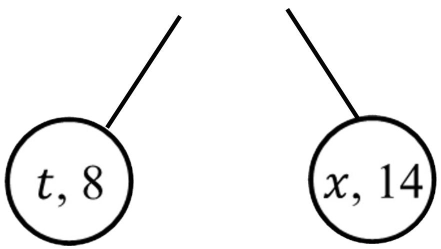


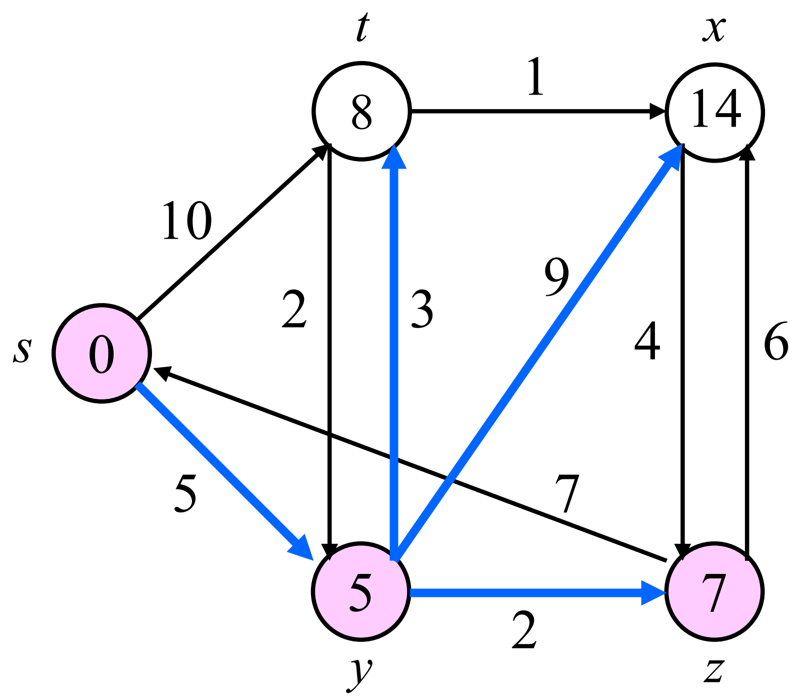
s	t	y	x	z
	y	s	y	y





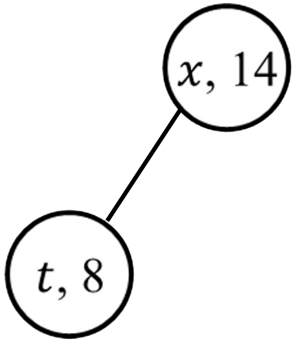
s	t	y	x	z
	y	s	y	y

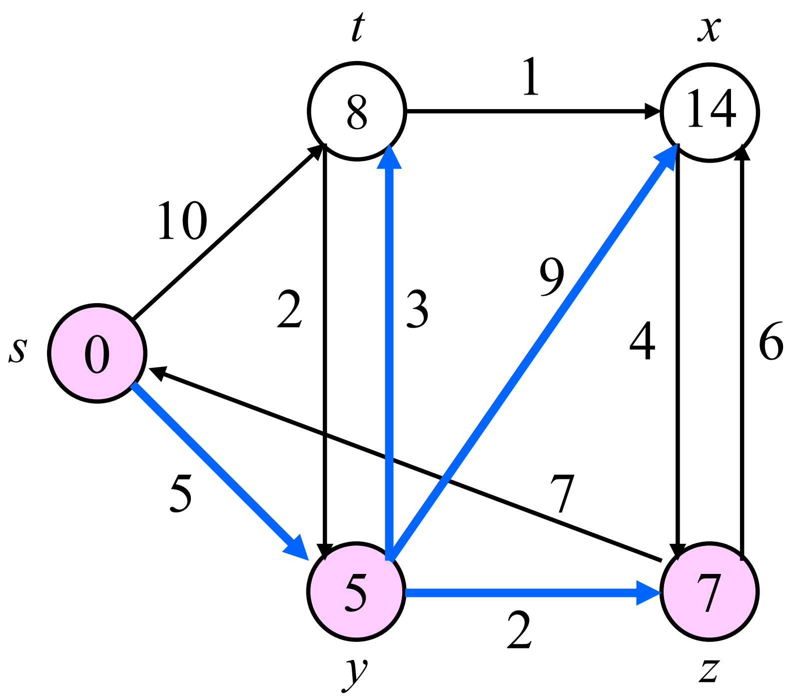




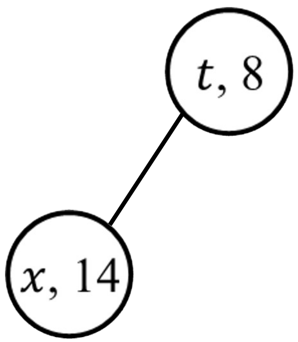
Dijkstra's Algorithm

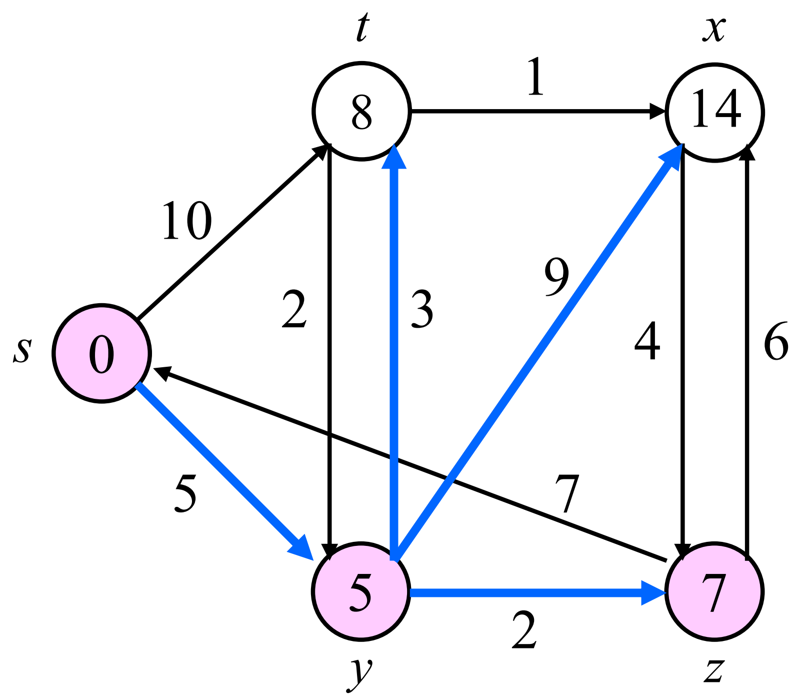
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	*y*



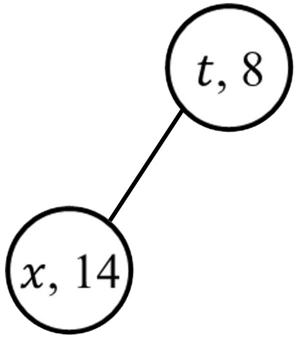


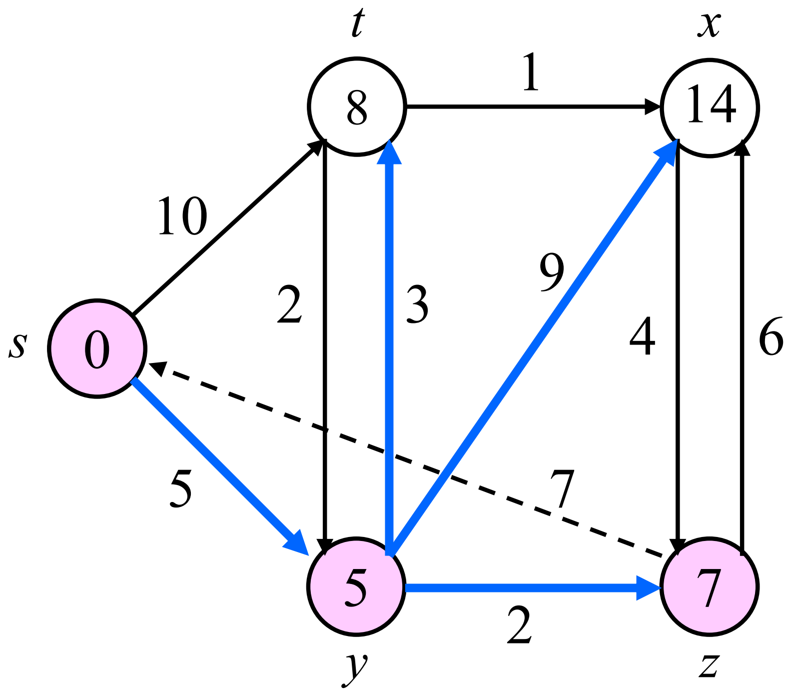
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	*y*





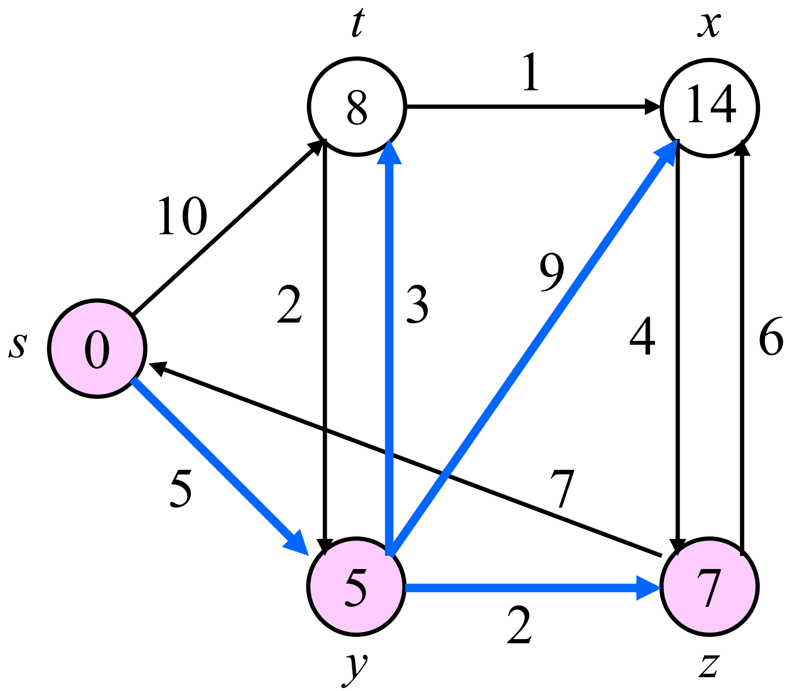
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	*y*



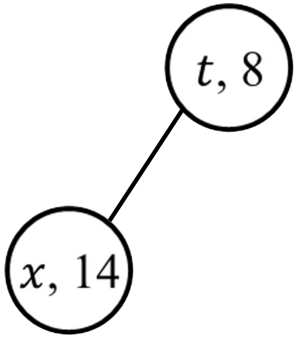


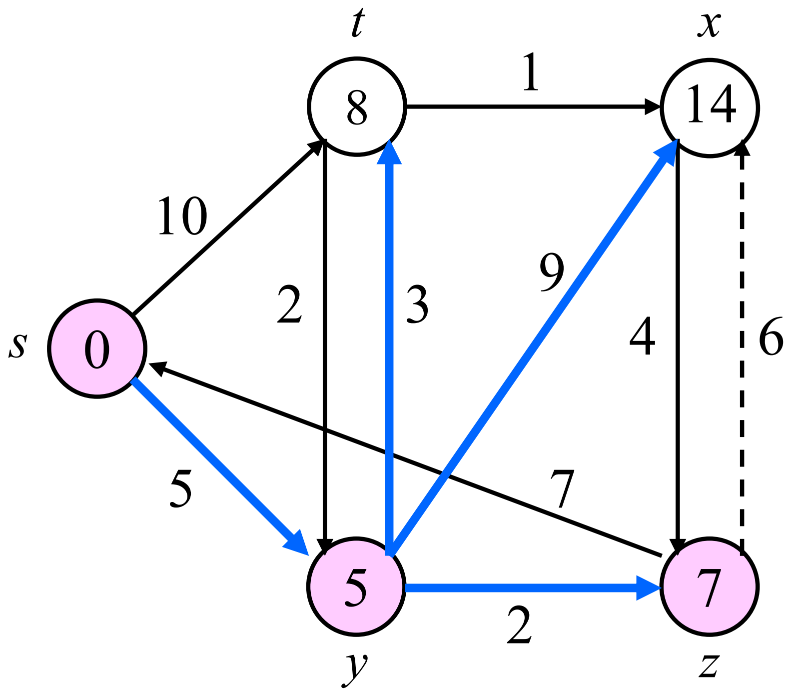
Dijkstra's Algorithm

s	t	y	x	z
	y	s	y	y



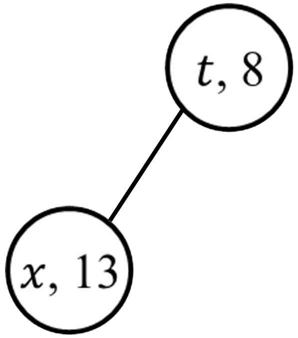
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	*y*

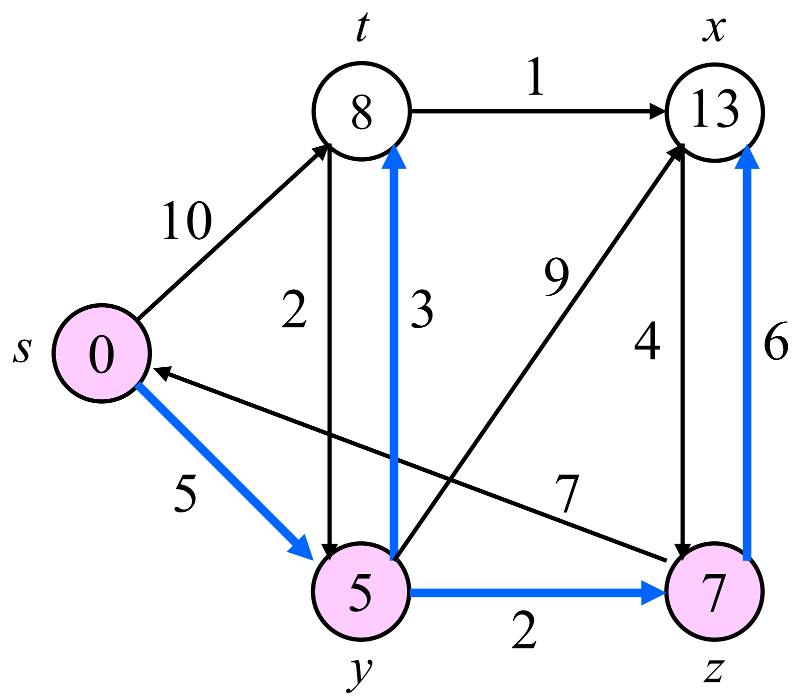




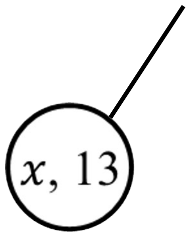
Dijkstra's Algorithm

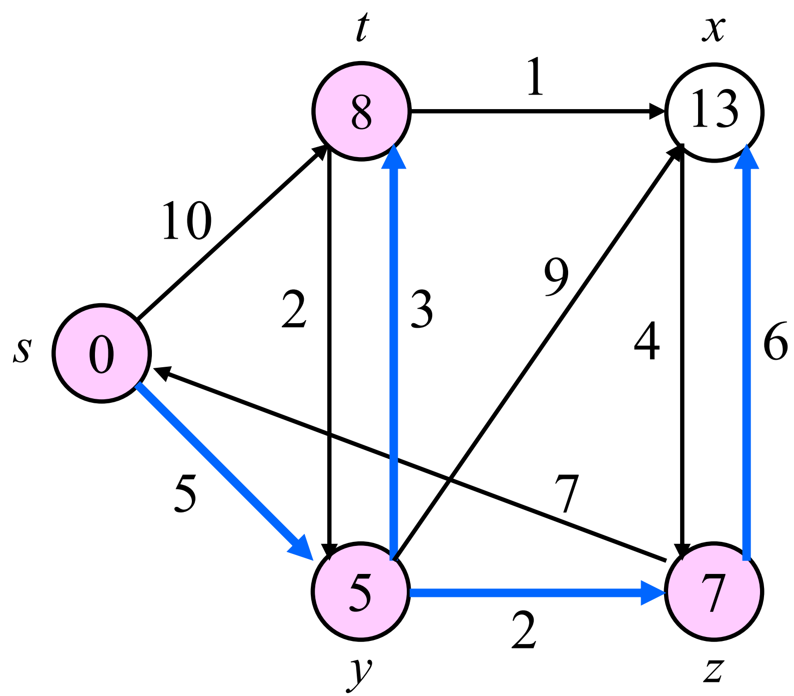
s	*t*	*y*	*x*	*z*
	y	*s*	*z*	*y*



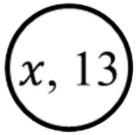


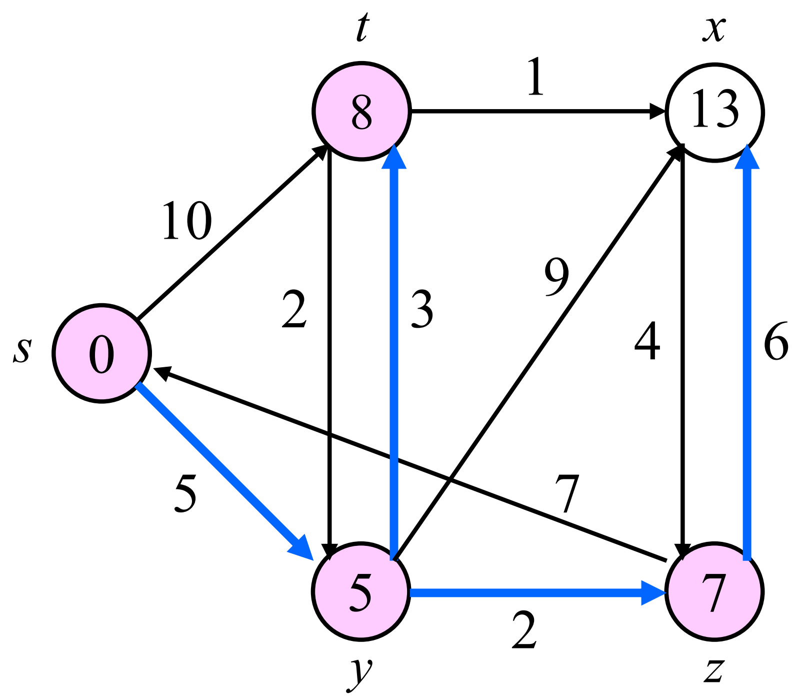
s	*t*	*y*	*x*	*z*
	y	*s*	*z*	*y*





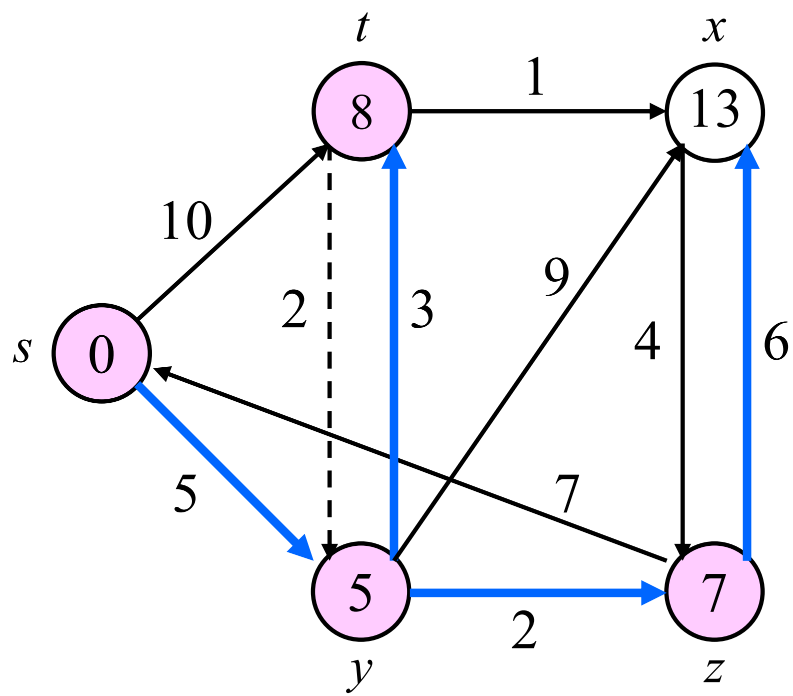
s	*t*	*y*	*x*	*z*
	y	*s*	*z*	*y*





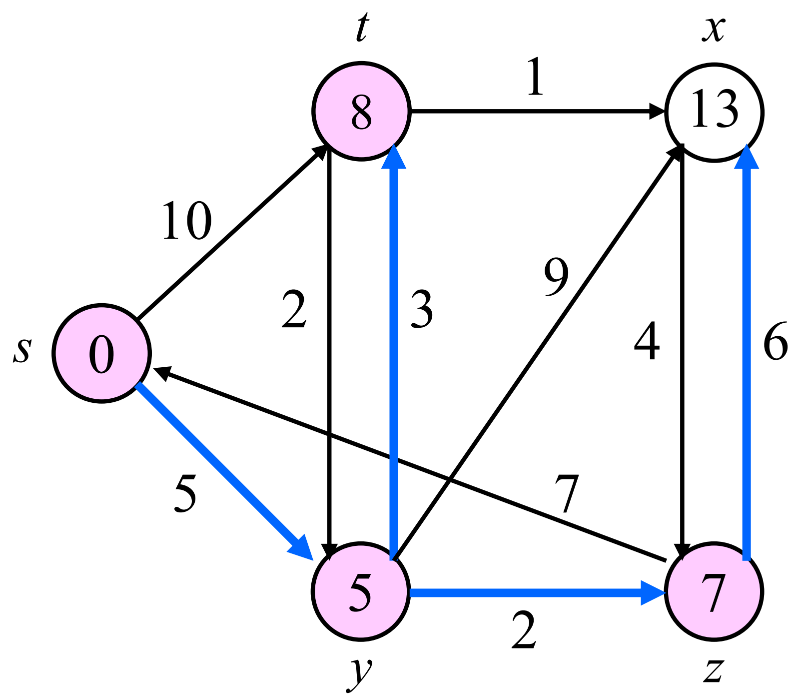
s	*t*	*y*	*x*	*z*
	y	*s*	*z*	*y*

x, 13



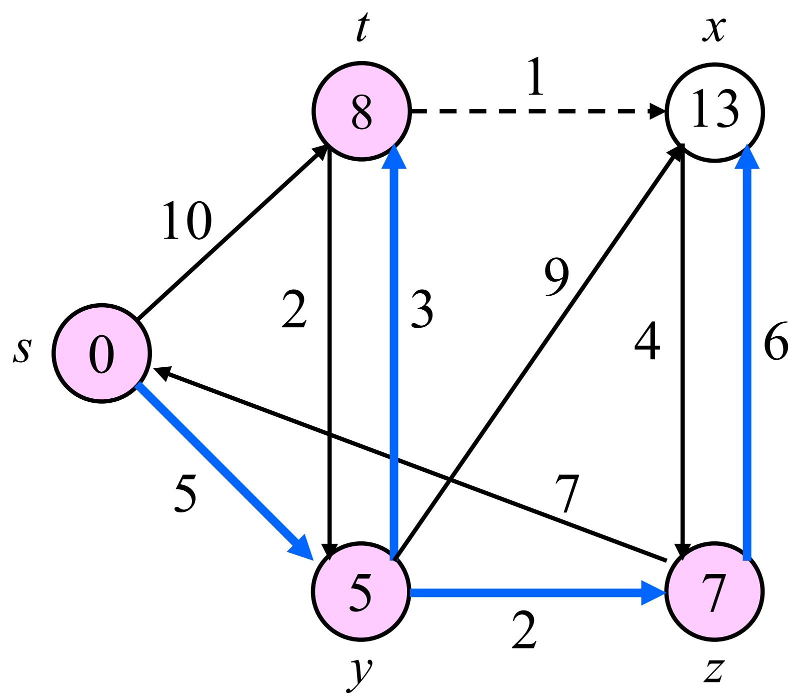
s	t	y	x	z
	y	s	z	y

x, 13



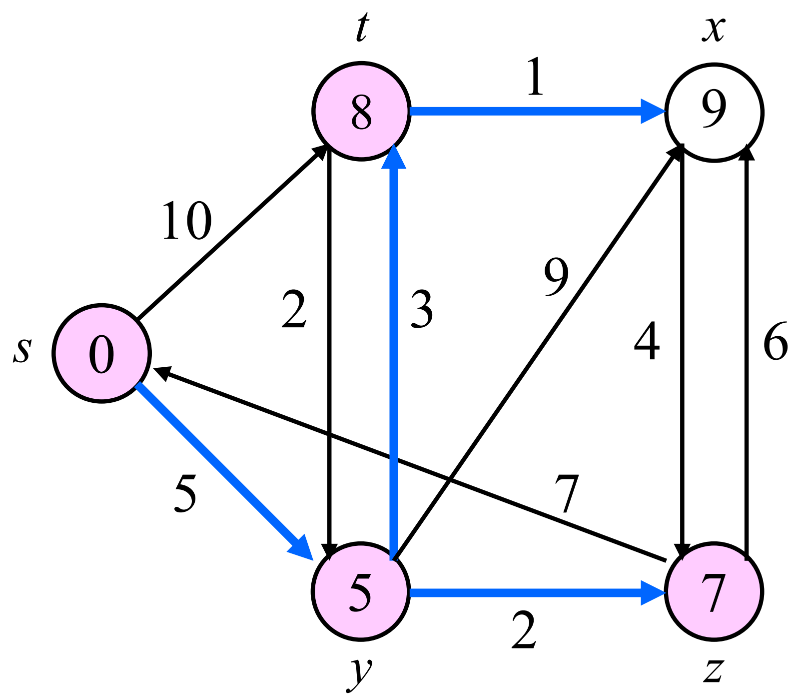
s	*t*	*y*	*x*	*z*
	y	*s*	*z*	*y*

(*x*, 13)

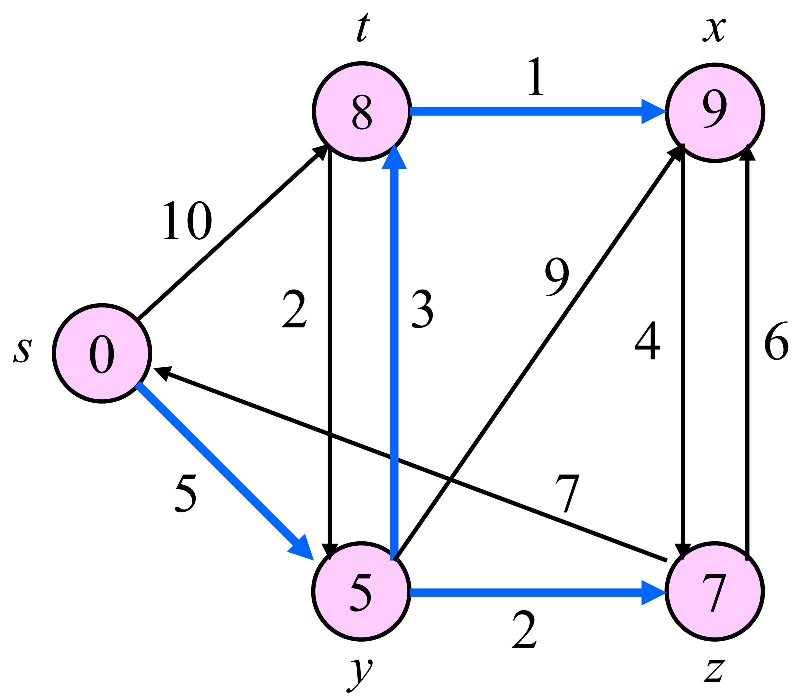


s	t	y	x	z
	y	s	t	y

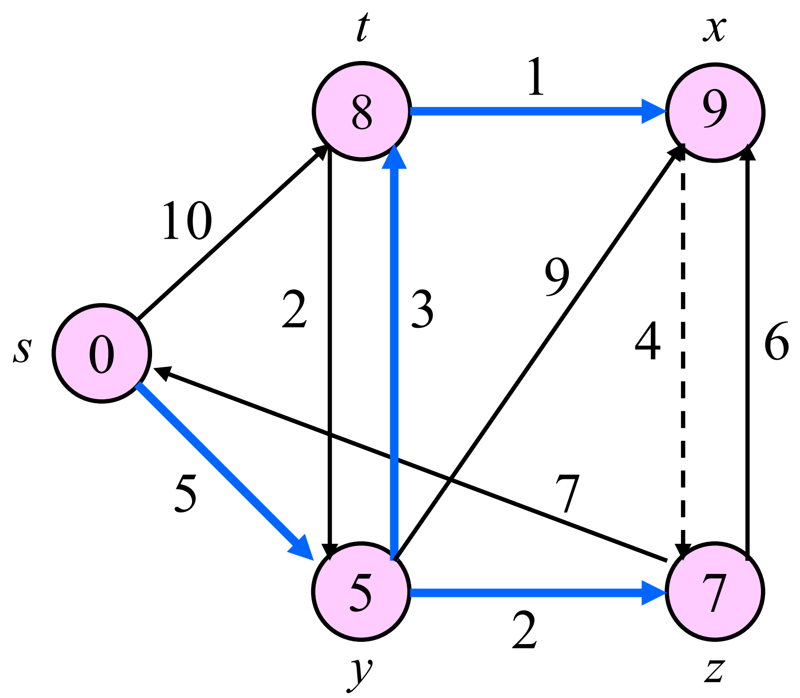
x, 9



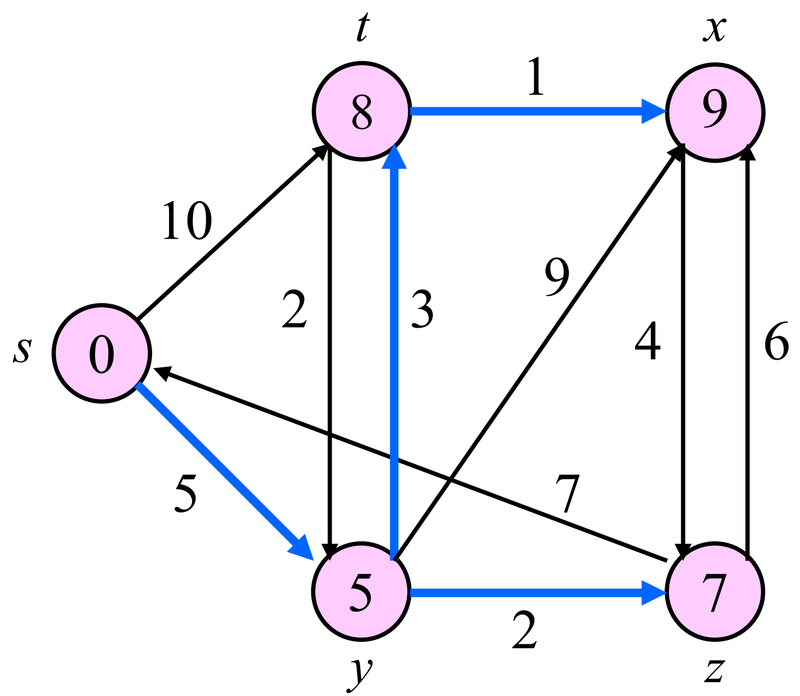
s	*t*	*y*	*x*	*z*
	y	*s*	*t*	*y*



s	*t*	*y*	*x*	*z*
	y	*s*	*t*	*y*



s	t	y	x	z
	y	s	t	y



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)
```

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

- Definitions
  - length of a path: sum of edge weights along the path
  - distance from u to v, $\delta(u, v)$: minimum length
- Problem: Given a directed graph with NONNegative edge weights $G = (V, E)$, and a special source vertex $s \in V$, determine the distance from the source vertex to every vertex in G.
  - $d[v]$: estimate the shortest path
  - $\pi[v]$: predecessor pointer of the path

- Principle Observation
  - Any subpath of a shortest path must also be a shortest path. Maintain an Estimate of shortest path for each vertex $d[v]$
  - Initially, $d[s] = 0$ and $d[v] = \infty$
  - $d[v] \geq \delta(s, v)$: As the algorithm goes on, it updates $d[v]$ until all $d[v]$ converge to $\delta(s, v)$ (This update process is called relaxation.)

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

$d[v] = d[u] + w[u, v];$

$\pi[v] = u;$

- Maintain a subset of vertices $S \subseteq V$, for which we claim we “know” the shortest distance, $d[u] = \delta(s, u)$.
- Initially, $S = \{\}$ and one by one we selected vertices from $V - S$ to add to S at each stage.
- We select the vertex whose $d[u]$ is minimum. We implement this on a *priority* queue where every operation (Insert, Delete_min, Decrease_key) can be done in $O(\lg n)$ time.
- At each stage
  - select a vertex u, which has the smallest $d[u]$ among all the unknown vertices.
  - declare that the shortest path from s to u is known
  - update $d[v]$: $d[v] = d[u] + w[u, v]$ if this value for $d[v]$ is an improvement. (decide if it is a good idea to use u on the path to v.)

Lemma When a vertex u is added to S, $d[u] = \delta(s, u)$.

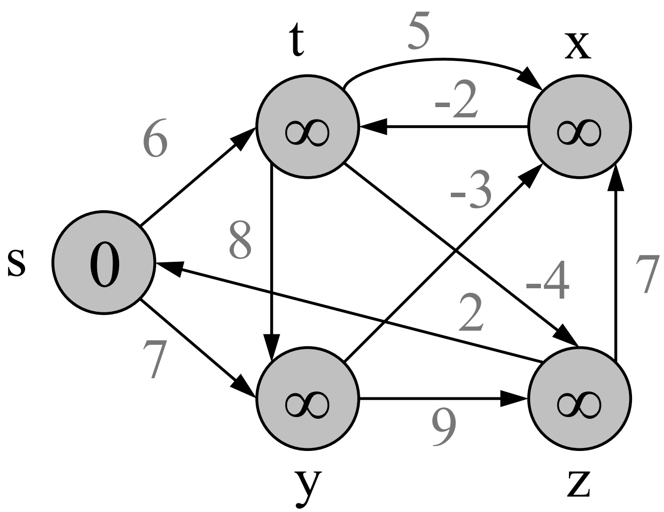
Proof: We assume all edge weights are STRICTLY positive. Suppose the algorithm FIRST attempts to add a vertex u to S for which $d[u] \neq \delta(s, u)$, so $d[u] > \delta(s, u)$. Consider the situation JUST PRIOR to the insertion of u. Consider the true shortest path from s to u. Since $s \in S$ and $u \in V - S$, at some point this path takes a jump out of S. Let (x, y) be the edge taken by the path where $x \in S$ and $y \in V - S$. We argue $y \neq u$. (Why? Since $d[x] = \delta(s, x)$ and we applied relaxation when we add x, we would have set $d[u] = d[x] + w(x, u) = \delta(s, u)$, but we assumed this is not the case.) Now since y appears midway on the path from s to u and all subsequent edges are positive, we have $\delta(s, y) < \delta(s, u)$, and thus, $d[y] = \delta(s, y) < \delta(s, u) < d[u]$. Thus, y would have been added BEFORE u, in contradiction to our assumption.

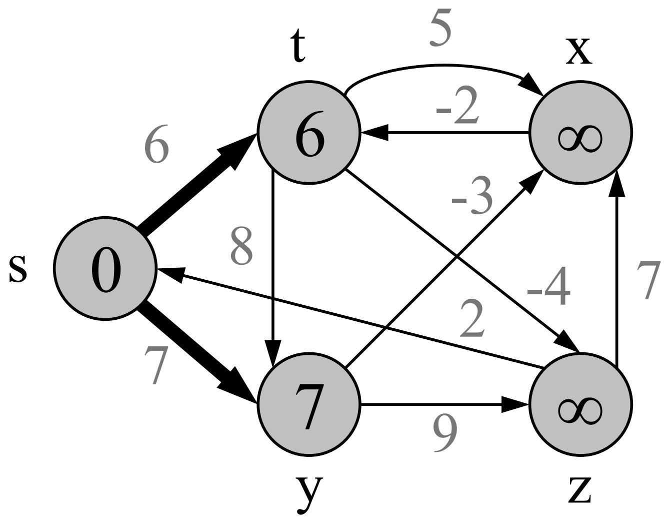
- **The Bellman-Ford algorithm**
  - it solves the single source shortest-paths problem in the general case in which edge weights may be negative.

The Bellman-Ford algorithm

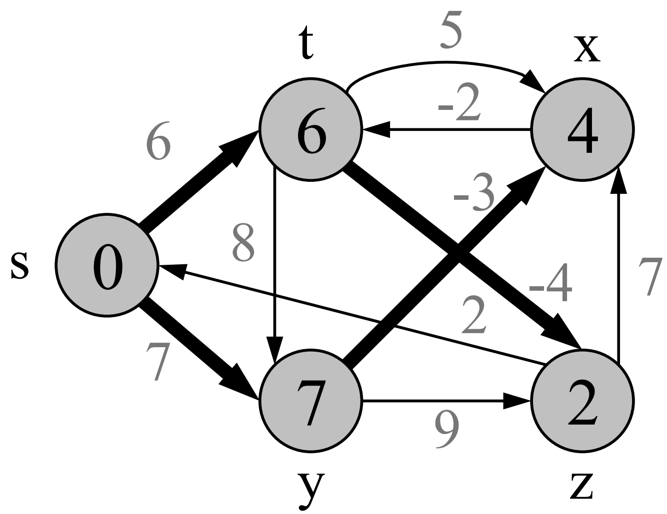
```
BELLMAN-FORD(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    for i = 1 to |G.V| - 1
            for each edge(u, v) ∈ G.E
                    RELAX(u, v, w)
    for each edge(u, v) ∈ G.E
        if v.d > u.d + w(u, v)
            return FALSE
    return TRUE
```

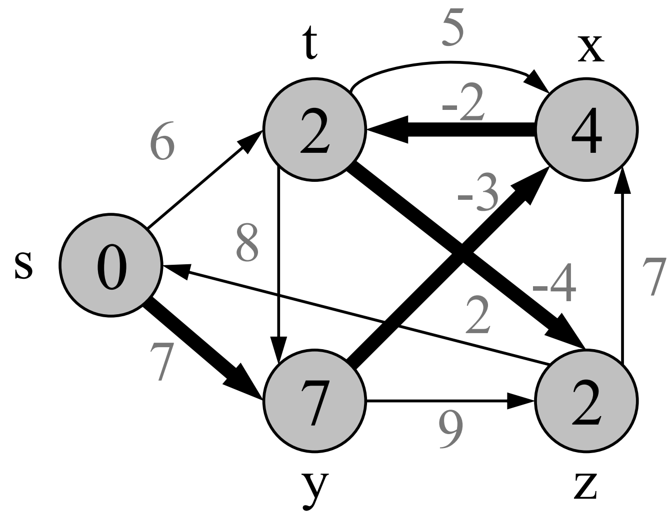
- Relaxation order
  - (t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)



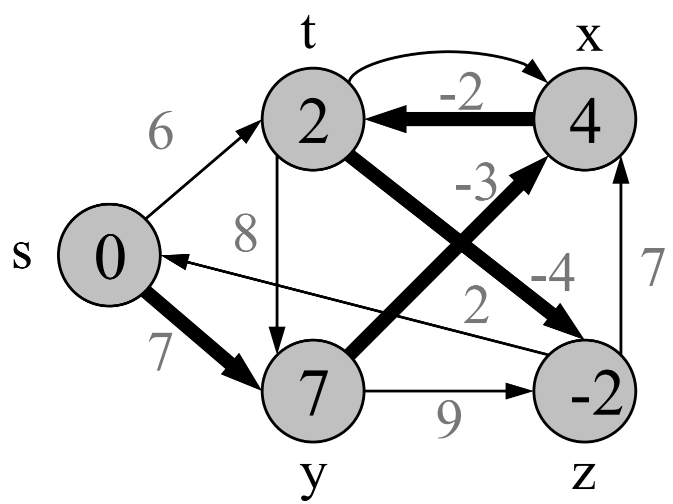


- Relaxation order
  - (t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)





- 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**
  - Running time : $O(VE)$

Assume there is a negative cycle, $< v_0, v_1, \ldots, v_k >$ where $v_0 = v_k$, and yet the Bellman-Ford returns True.
Then, for all $i = 1, 2, .., k$ $d[v_i] \leq d[v_{i-1}] + w[v_{i-1}, v_i]$.

Summing for all nodes in a cycle,

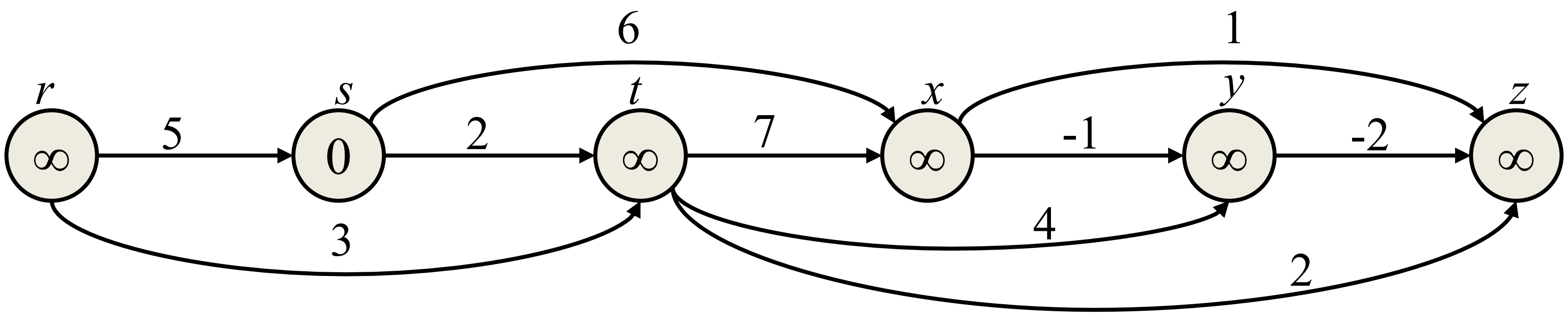
$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} (d[v_{i-1}] + w[v_{i-1}, v_i])$$
$$= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w[v_{i-1}, v_i]$$

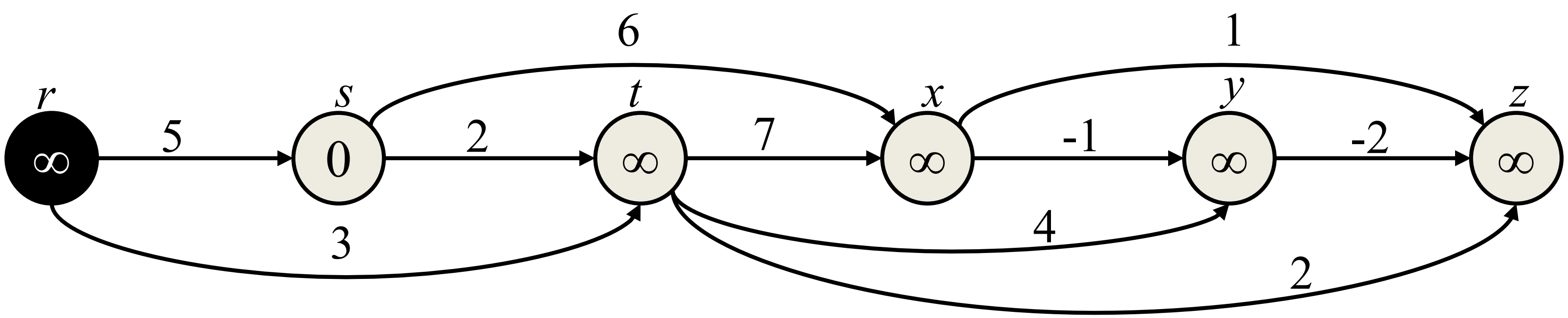
Because $v_0 = v_k$, $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$
Thus, $0 \leq \sum_{i=1}^{k} w[v_{i-1}, v_i]$. Contradiction!

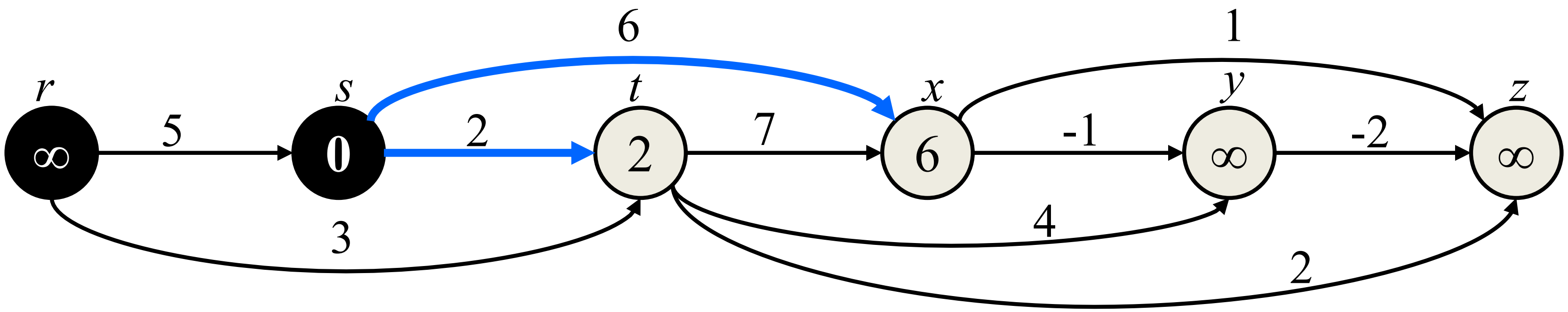
DAG-SHORTEST-PATHS(G, w, s)

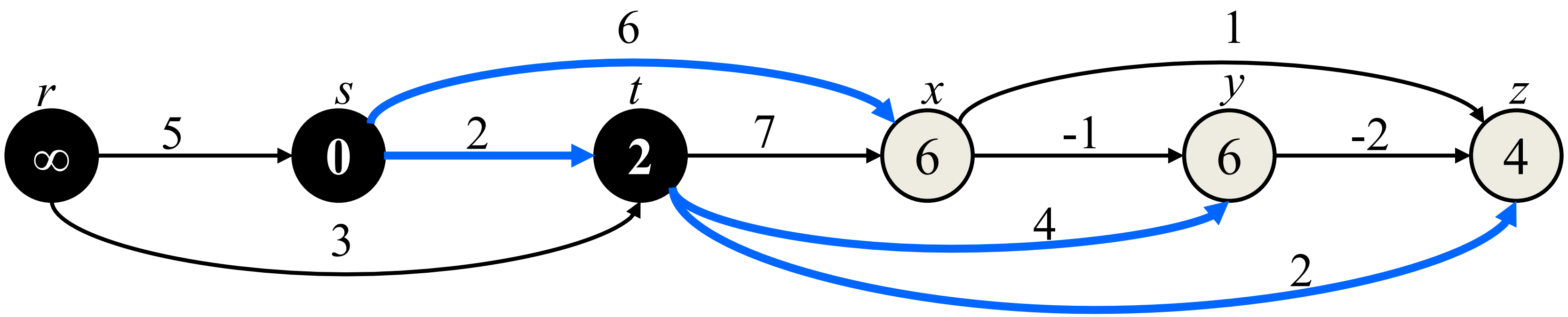
```
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Single-source shortest paths in directed acyclic graphs

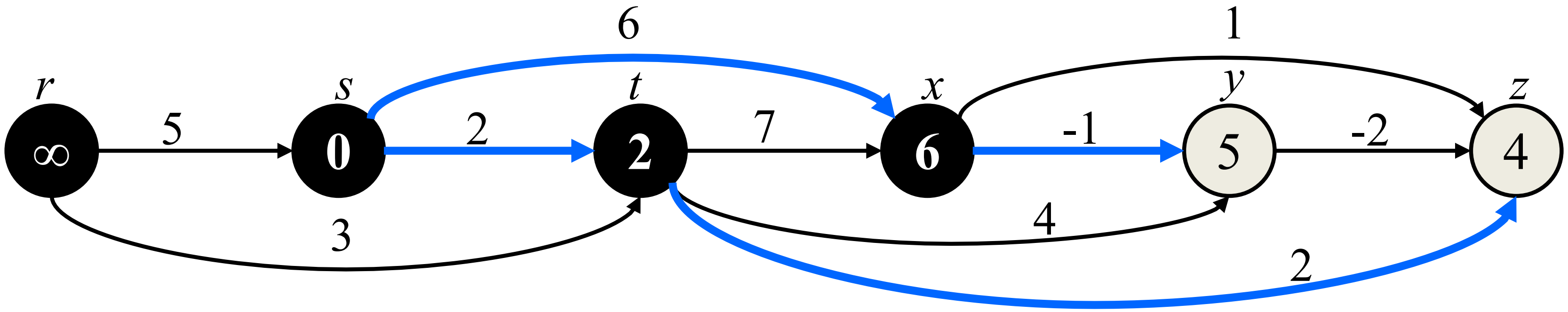


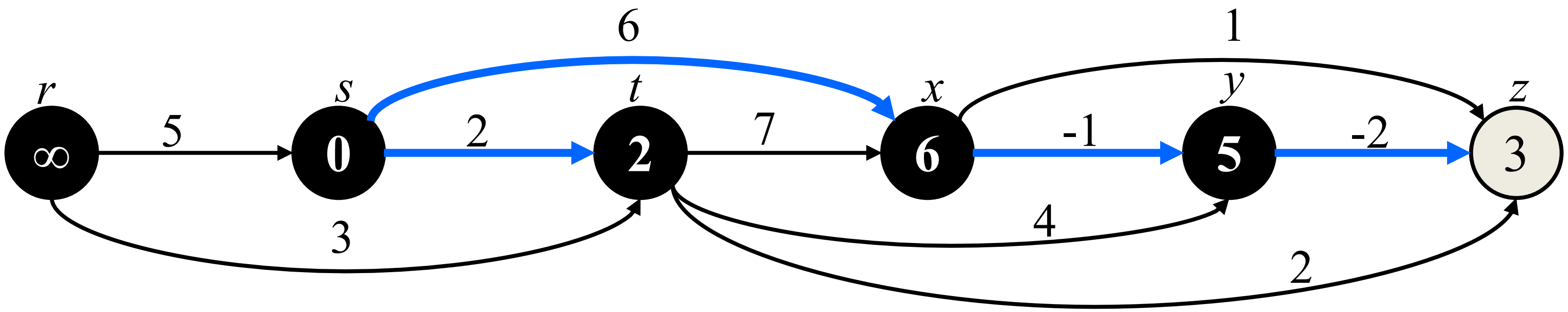


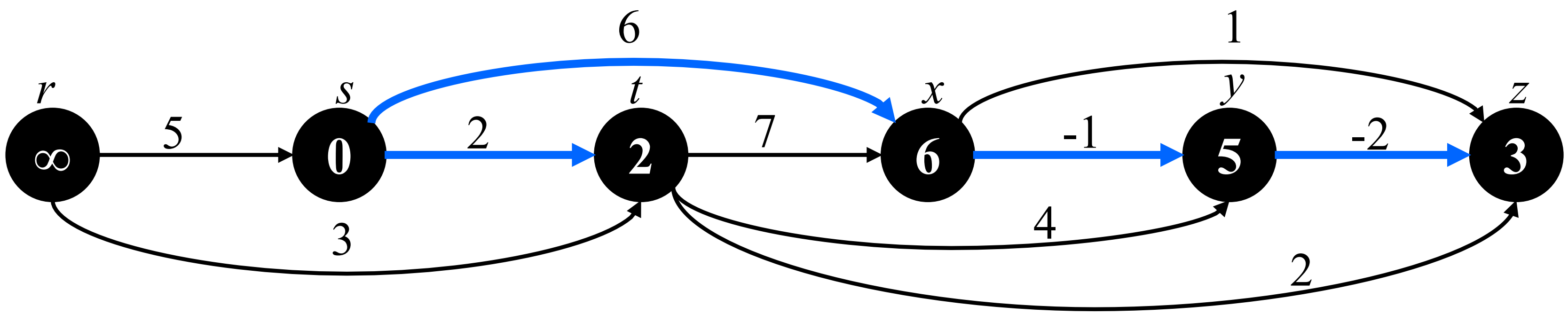

6
1
r
s
t
x
y
z
5
2
7
-1
-2
∞
0
2
6
∞
∞
3
4
2


6
1
r
s
t
x
y
z
5
2
7
-1
-2
∞
0
2
6
6
4
3
4
2

Single-source shortest paths in directed acyclic graphs







- Running time: O(V+E) time

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

- **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.

- **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 “-∞” and “>” by “<”.