Single-Source Shortest Paths

Heejin Park
Hanyang University

Contents

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

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

Definition

- **Shortest-path problems**
  - Single-source & single-destination

    Subset
  - Single-source (& all destinations) → $T(n)$ ← 이거 해결시 모두 해결
  - Single-destination (& all sources)
  - All pairs → $n \cdot T(n)$

    ↳ $G \to G^T$; single destination이라 동일

    transpose 해버리면

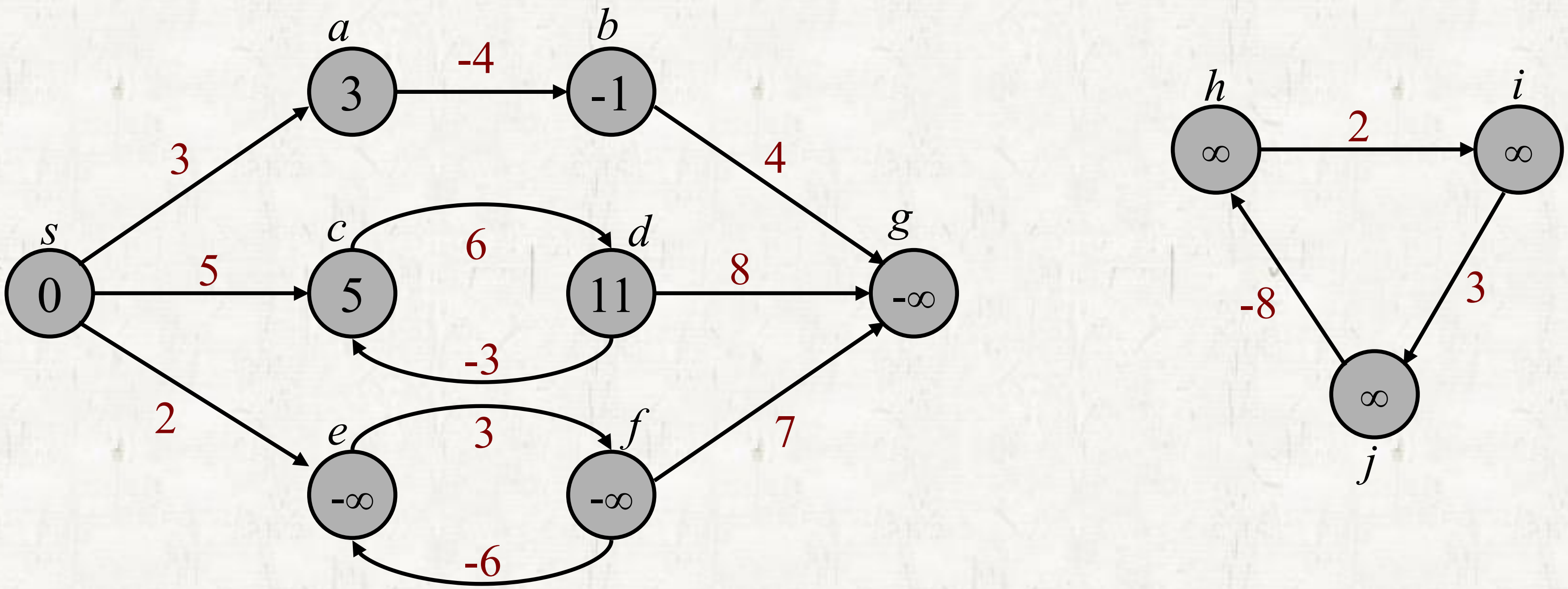
- An algorithm for single-source (& all destinations) problem can be used to solve all the other problems.

Negative-weight edges

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

→ $e \rightleftarrows f$ 100바퀴면

$-\infty$까지 가능 → 정의할 수 없다



Negative-weight edges

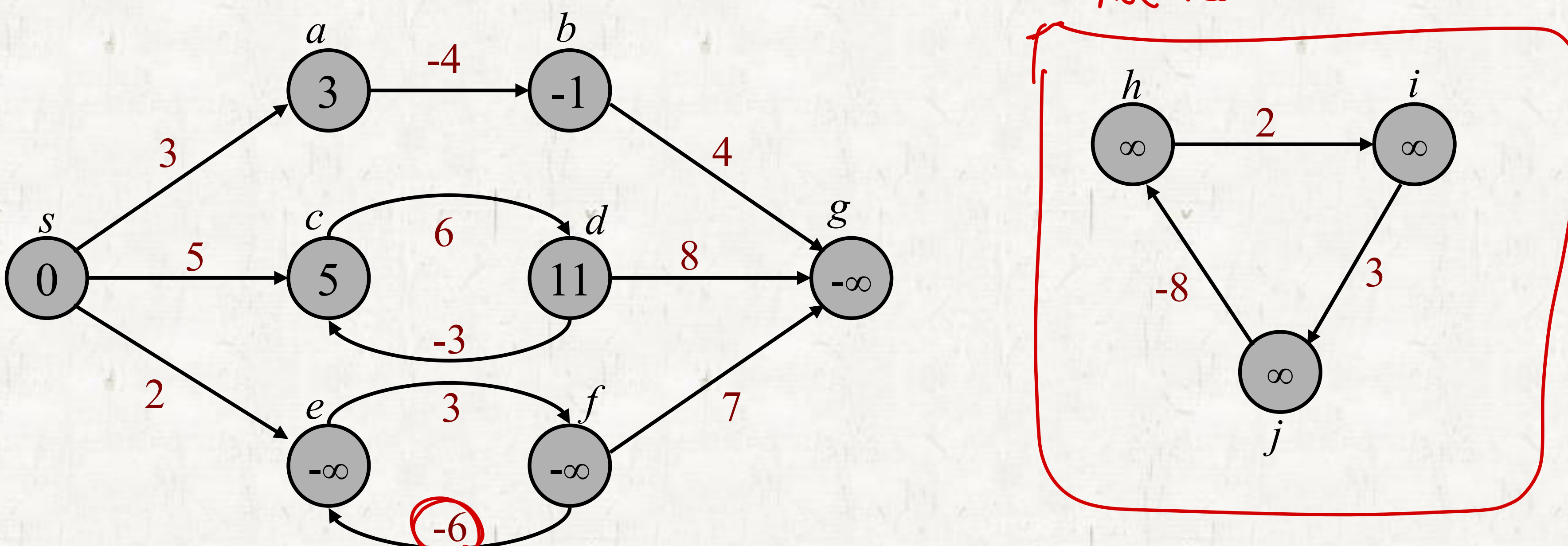
- Do all negative-weight edges cause a problem?

→ (f, e) 만

- Do all negative-weight cycles cause a problem?

→ NO, reachable이 아니라면 상관없다.

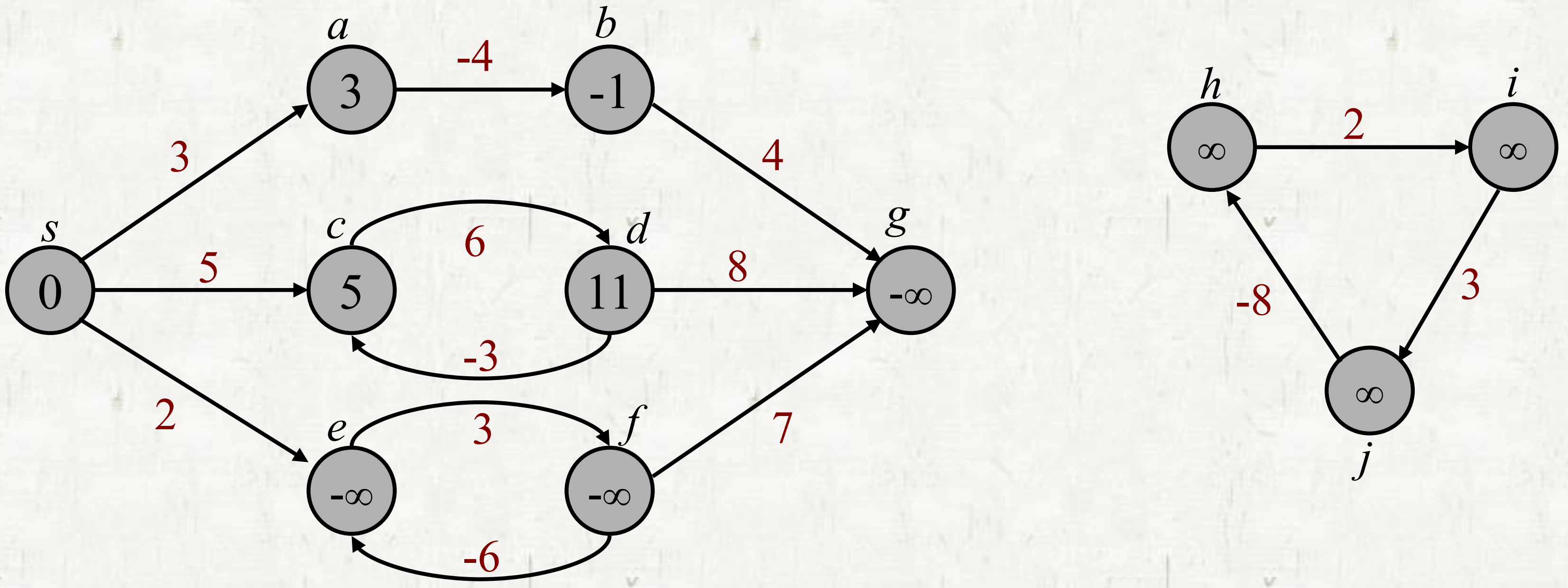
not reachable



cause a problem

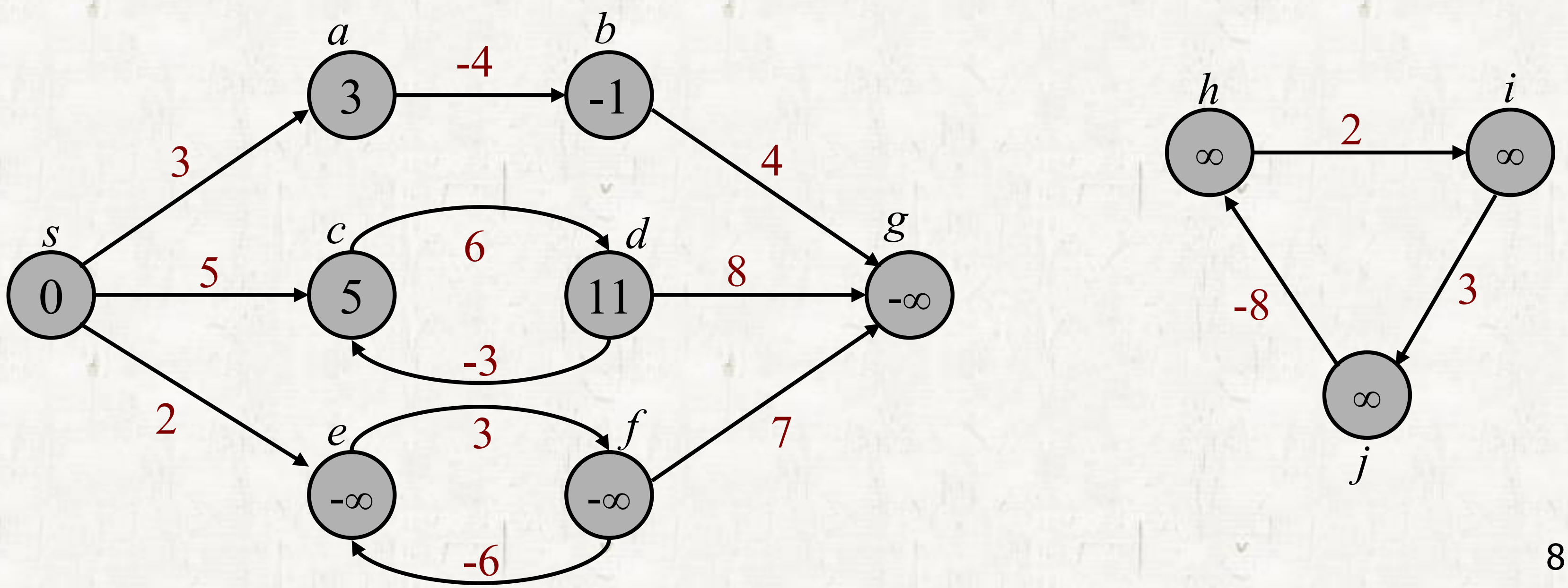
Negative-weight edges

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



Negative-weight edges

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



Cycles

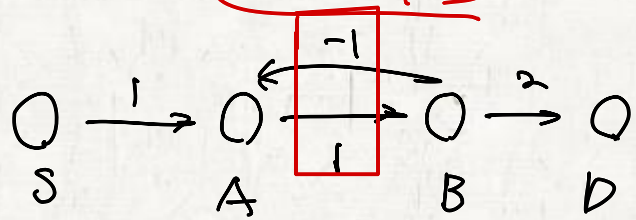
- **Cycles**
  - There is a shortest path that does not include cycles.
  - A shortest-path length is at most |*V*|-1.

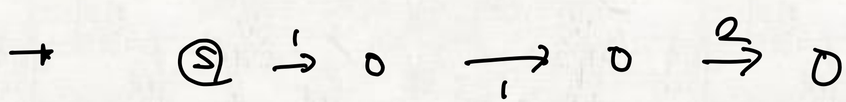
↳ 이러한 cycle 제거

↳ Cycle이 있다면 Cycle의 weight는 0이어야 한다.

↳ Shortest path 존재

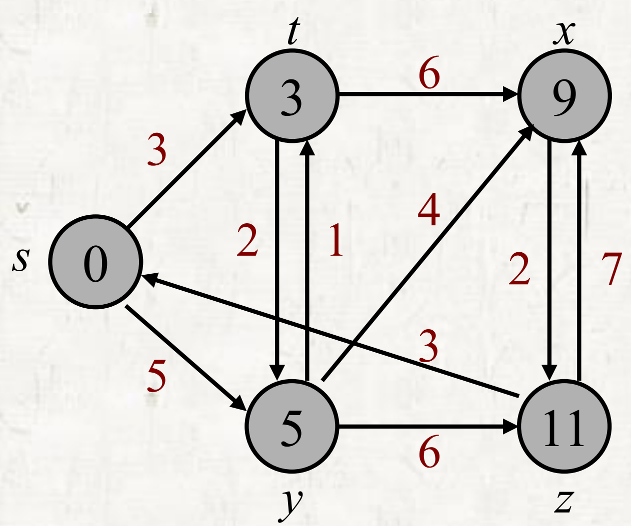
zero in cycles → 무시하므로 존재



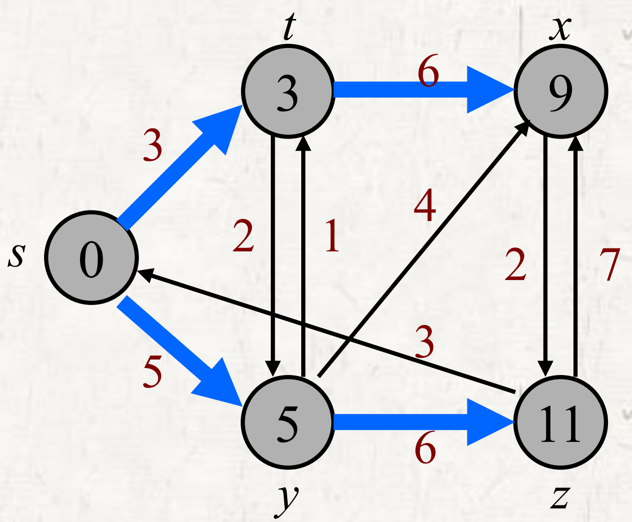


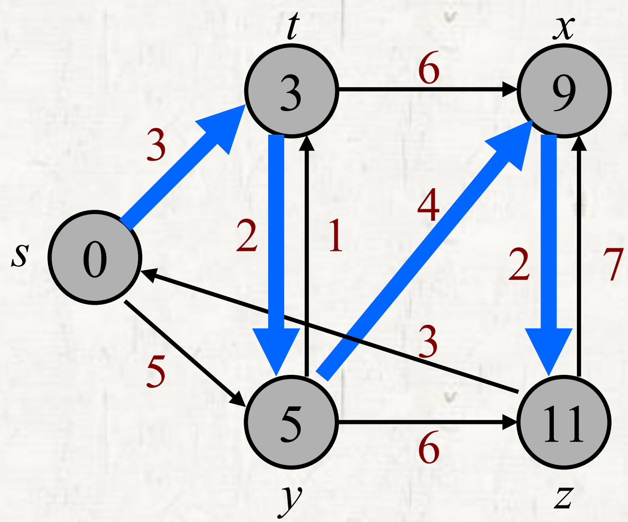
Predecessor subgraph

- **Predecessor subgraph**
  - Shortest-path tree $|E|=|V|-1$ (stores all SSSPs compactly.)
  - Optimal substructure



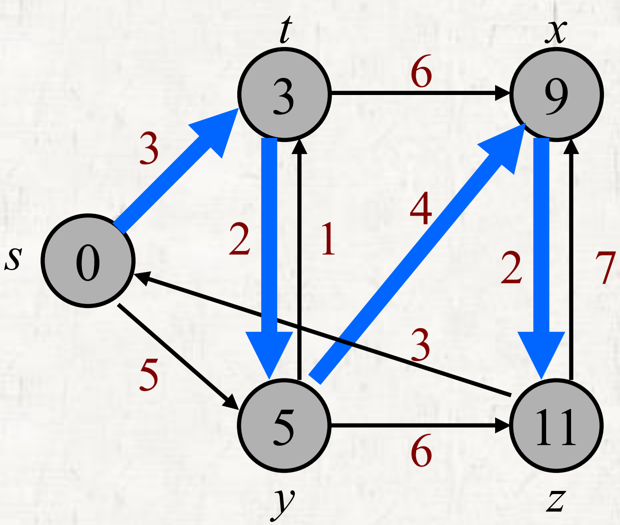
Predecessor subgraph





형태가 달라도
distance는 동일하다

Predecessor subgraph



t : s → t
y: s → t → y
x: s → t → y → x
z: s → t → y → x → z

→ 비효율

↳ worst case

$O(V^2)$ space

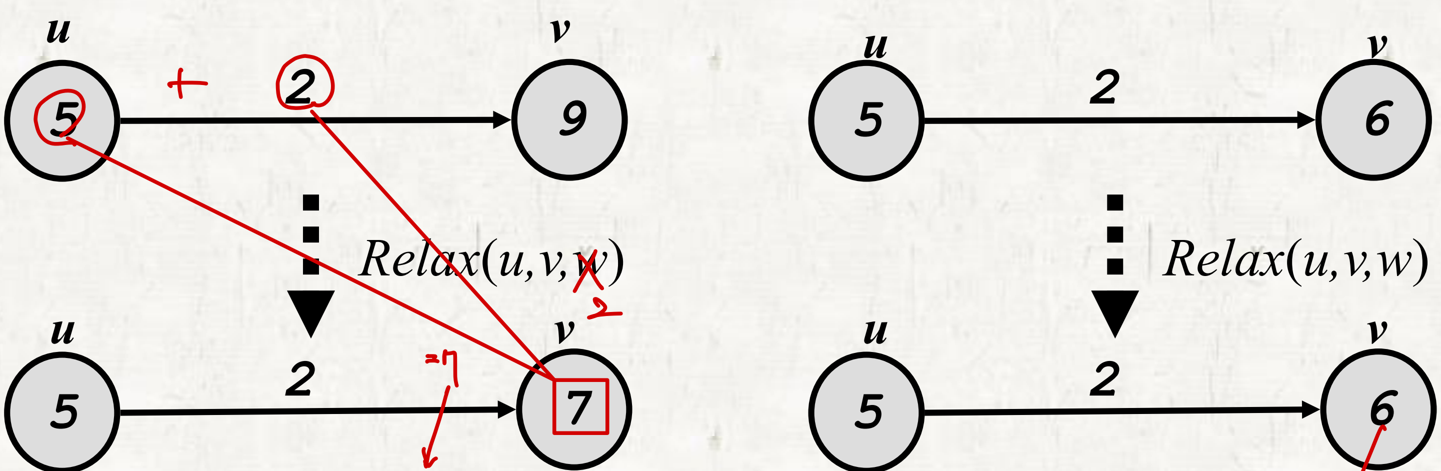
t : s
y: t
x: y
z: x

→ predecessor만 저장

$O(V)$ space

Relaxation

- **Relaxation**



9보다 작으므로 update

→ find a shortest tree

5+2 보다 작음
update X

Dijkstra's algorithm

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

Dijkstra's algorithm

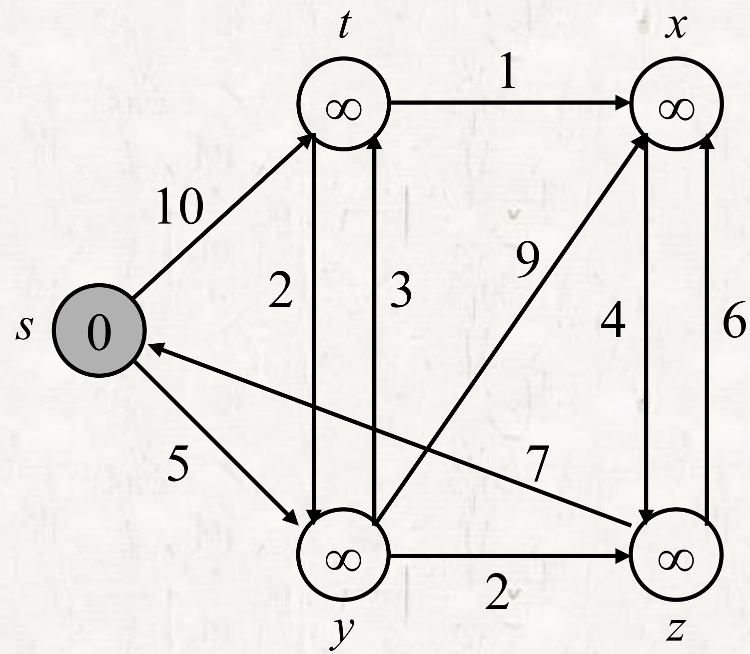
```
DIJKSTRA(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    S = Ø            → Shortest Path Tree (node 저장)
    Q = G.V          Priority Queue → min heap
    while Q ≠ Ø
        u = EXTRACT-MIN(Q)
        S = S ∪ {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u, v, w)
```

Dijkstra's Algorithm

세팅

Q

	s	t	y	x	z
distance	0	∞	∞	∞	∞

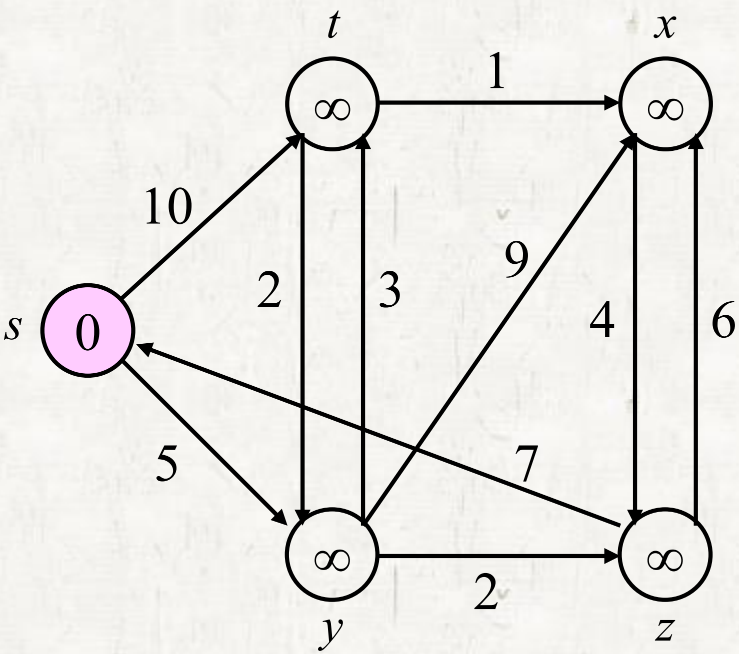


S

Dijkstra's Algorithm

Q

s	t	y	x	z
0	∞	∞	∞	∞

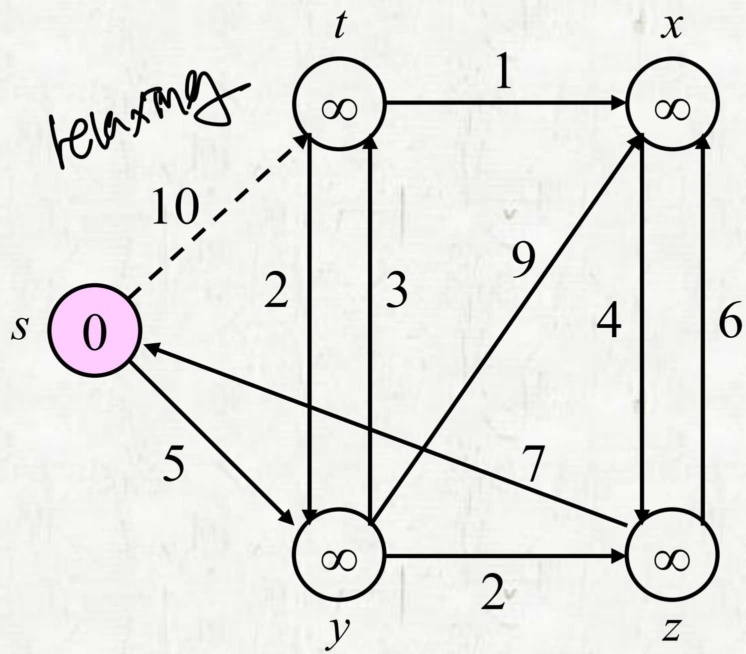


$S = \{s\}$

Dijkstra's Algorithm

Q

s	t	y	x	z
0	∞	∞	∞	∞

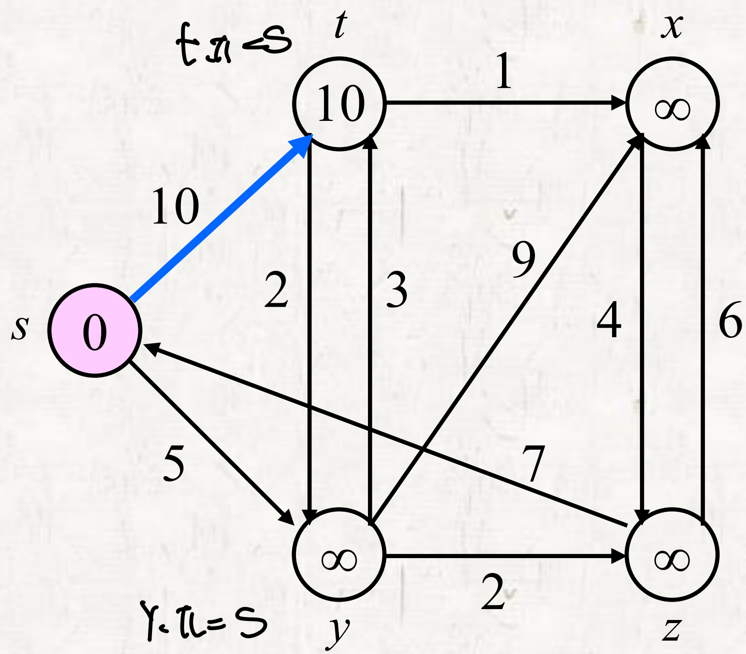


$\boldsymbol{S} = \{s\}$

Dijkstra's Algorithm

Q

s	*t*	*y*	*x*	*z*
0	∞	∞	∞	∞
	10	-	-	-

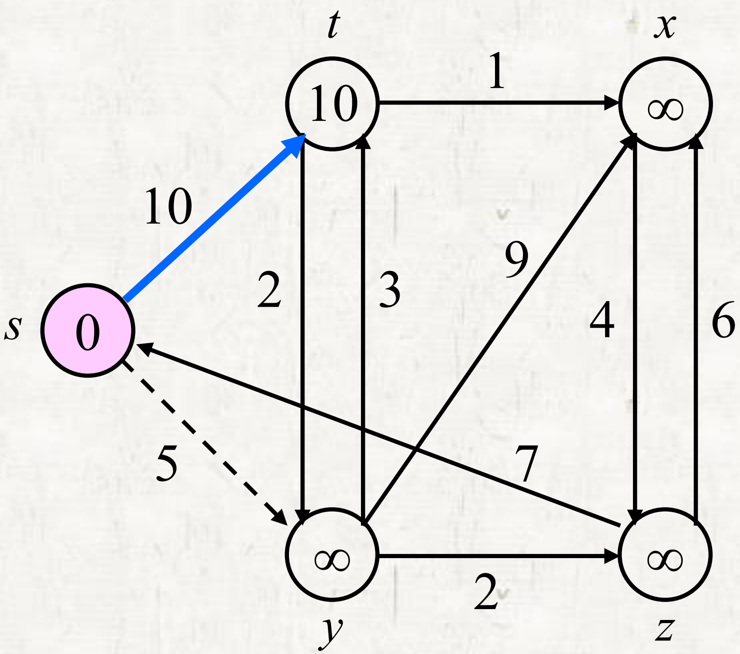


$\boldsymbol{S} = \{s\}$

Dijkstra's Algorithm

Q

s	t	y	x	z
0	∞	∞	∞	∞
	10	-	-	-

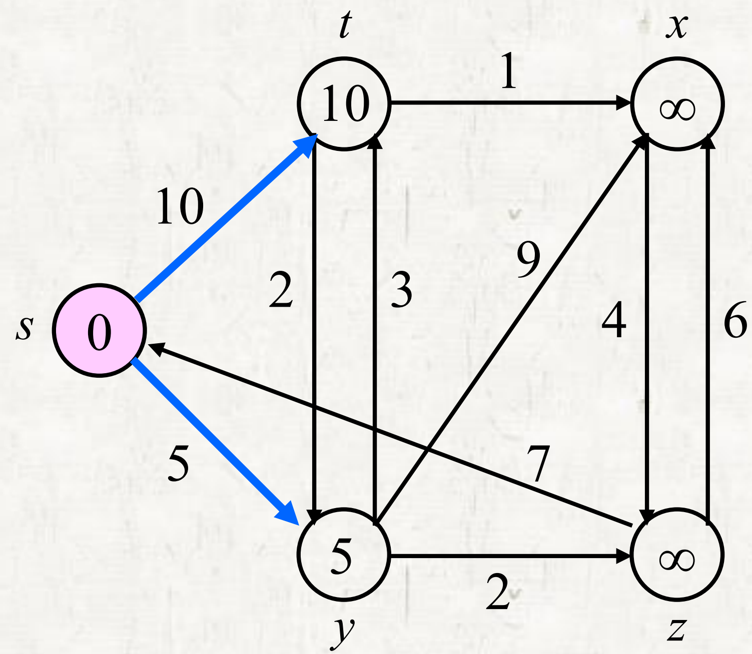


$\boldsymbol{S} = \{s\}$

Dijkstra's Algorithm

Q

s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-



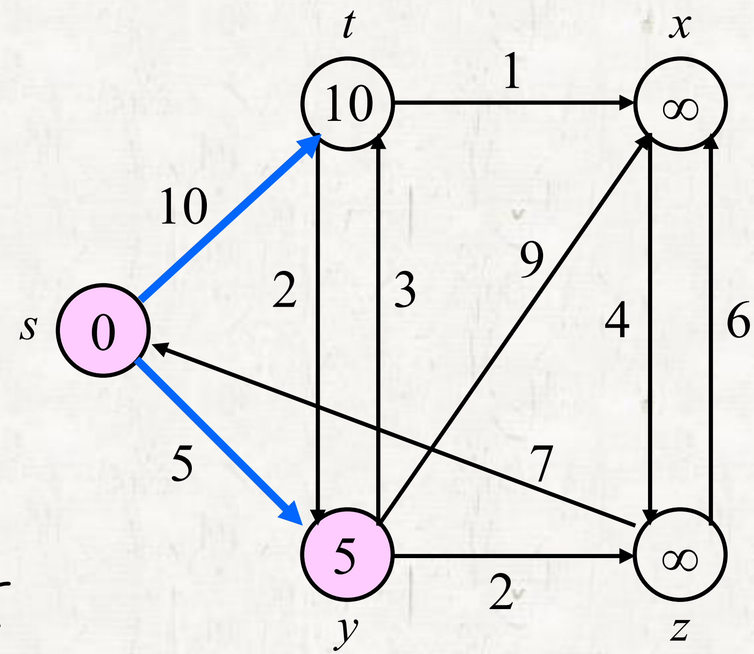
$\boldsymbol{S} = \{s\}$

Dijkstra's Algorithm

Q

s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-

Next smallest vertex

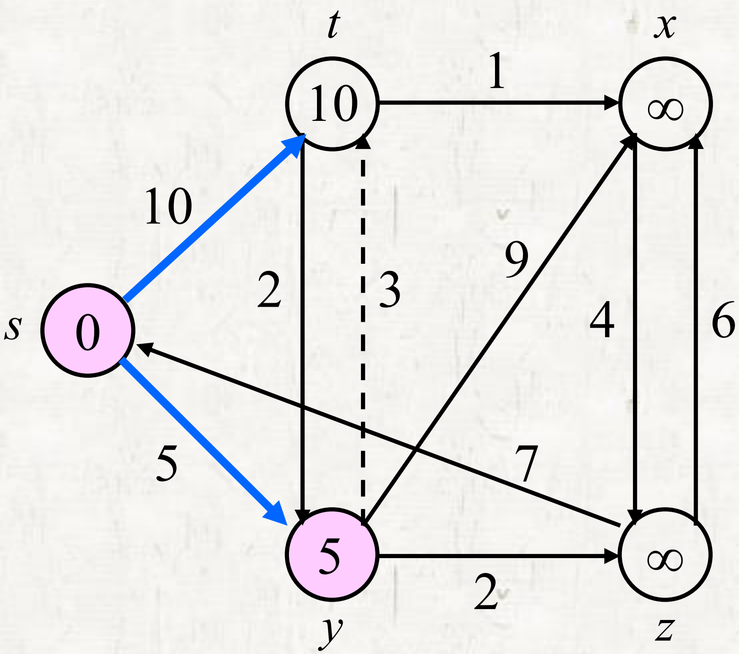


$\boldsymbol{S} = \{s, y\}$

Dijkstra's Algorithm

Q

s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-



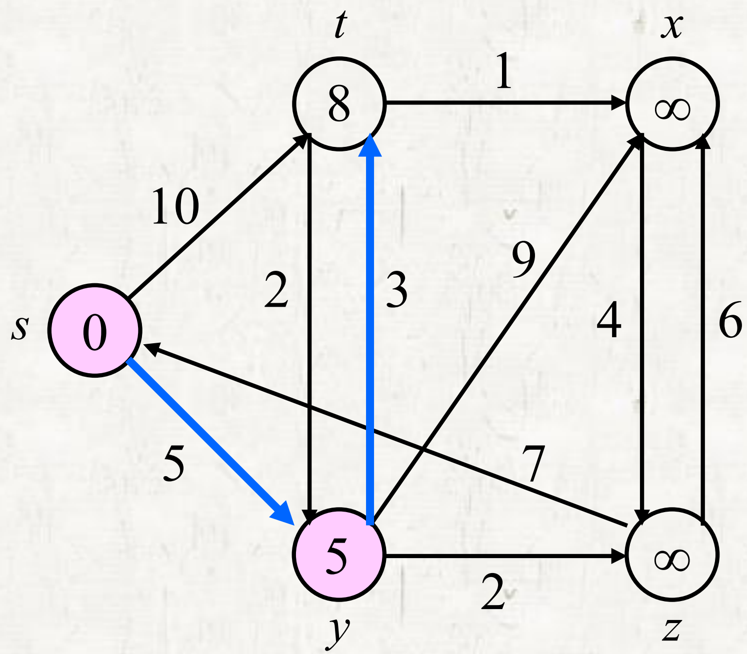
$\boldsymbol{S} = \{s, y\}$

Dijkstra's Algorithm

Q

s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-
	8		-	-

update

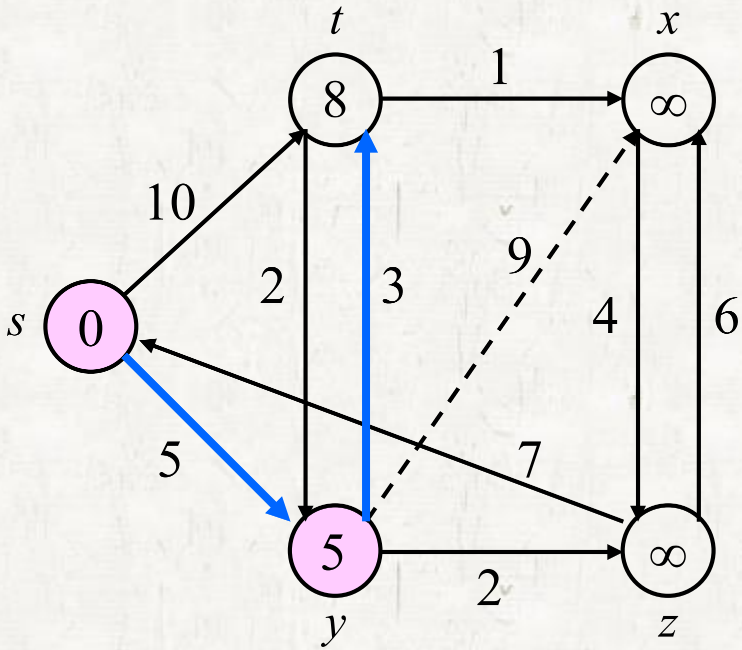


$\boldsymbol{S} = \{s, y\}$

Dijkstra's Algorithm

Q

s	t	y	x	z
0	∞	∞	∞	∞
	10	5	-	-
	8		-	-

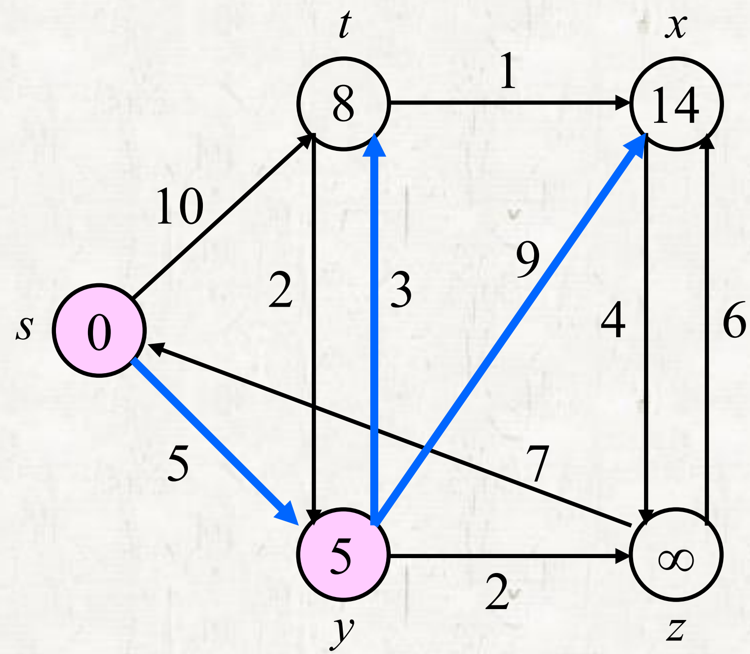


$\boldsymbol{S} = \{s, y\}$

Dijkstra's Algorithm

Q

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

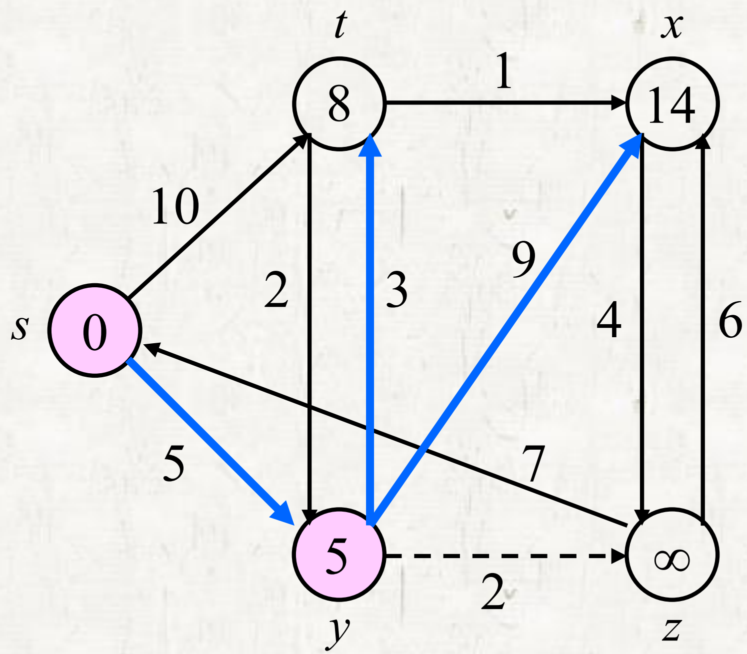


$\boldsymbol{S} = \{s, y\}$

Dijkstra's Algorithm

Q

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

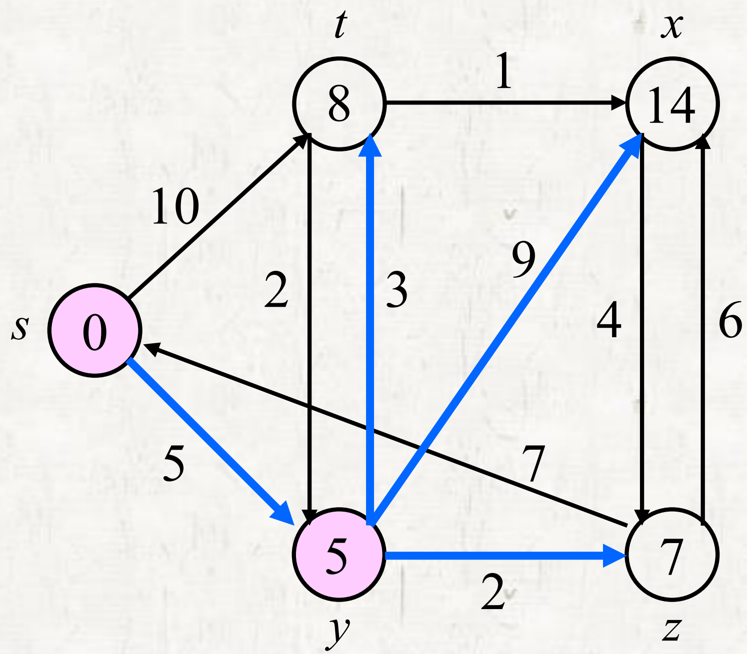


$\boldsymbol{S} = \{s, y\}$

Dijkstra's Algorithm

Q

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

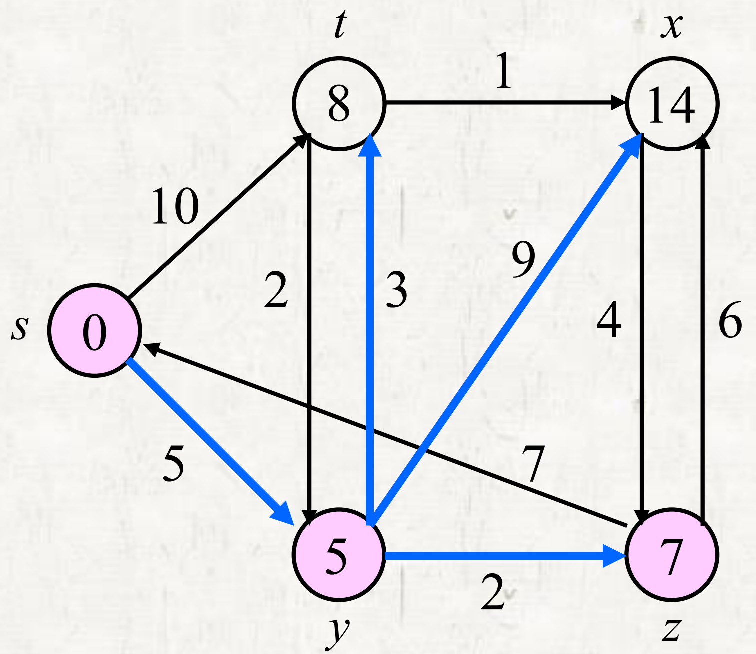


$\boldsymbol{S} = \{s, y\}$

Dijkstra's Algorithm

Q

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

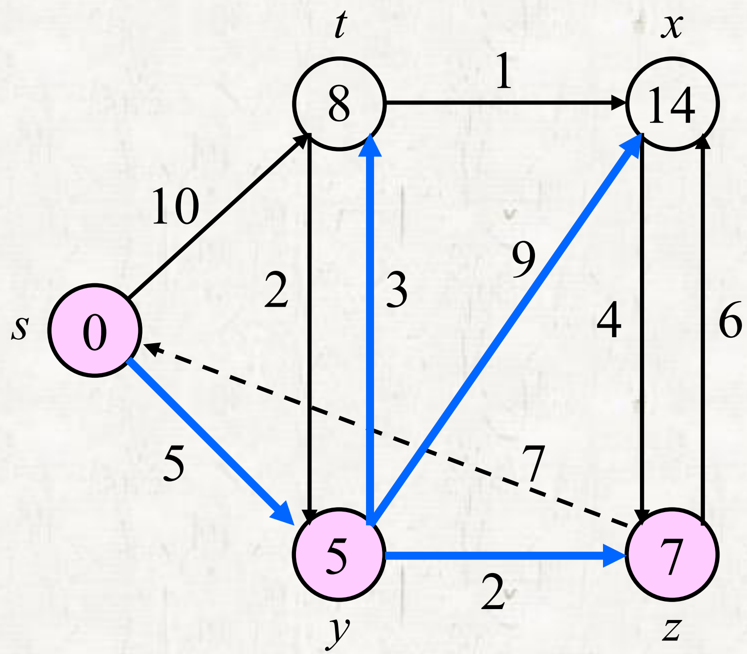


$\boldsymbol{S} = \{s, y, z\}$

Dijkstra's Algorithm

Q

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

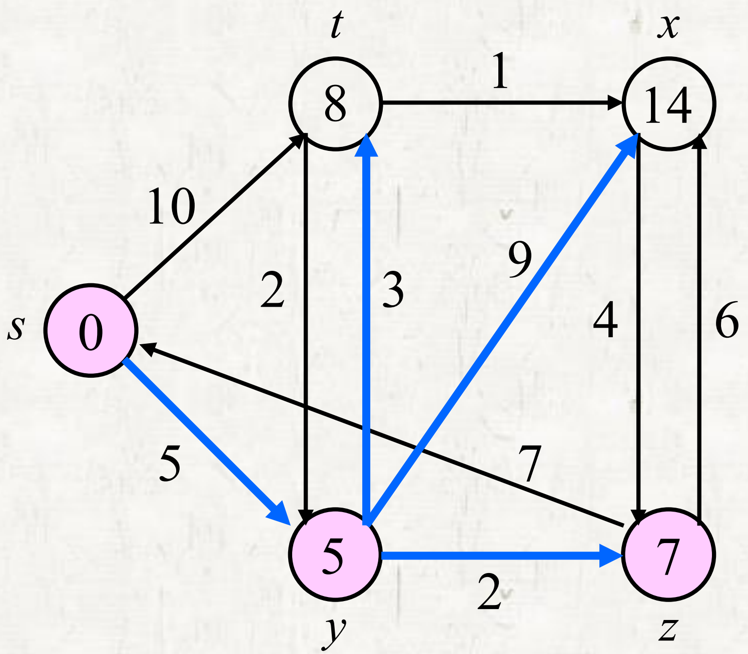


$\boldsymbol{S} = \{s, y, z\}$

Dijkstra's Algorithm

Q

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

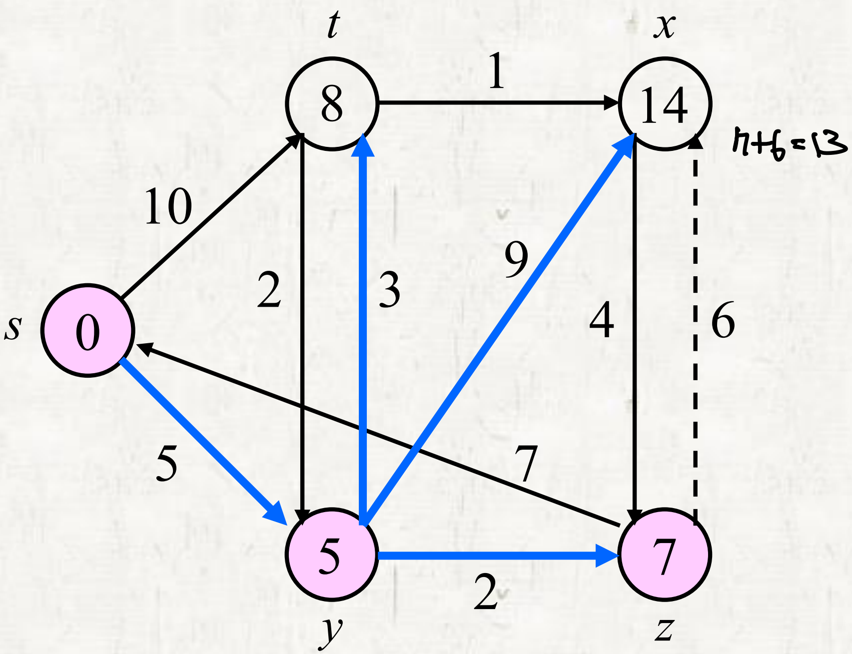


$\boldsymbol{S} = \{s, y, z\}$

Dijkstra's Algorithm

Q

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

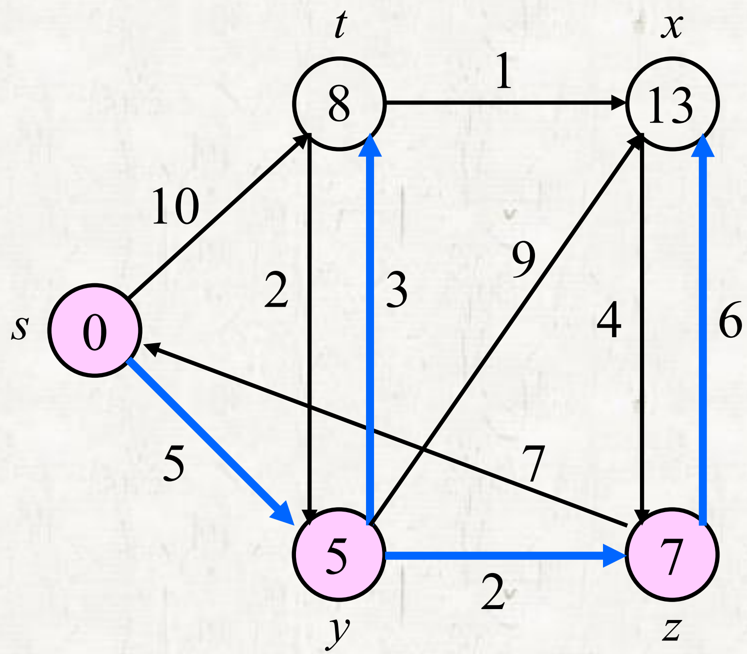


$\boldsymbol{S} = \{s, y, z\}$

Dijkstra's Algorithm

Q

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

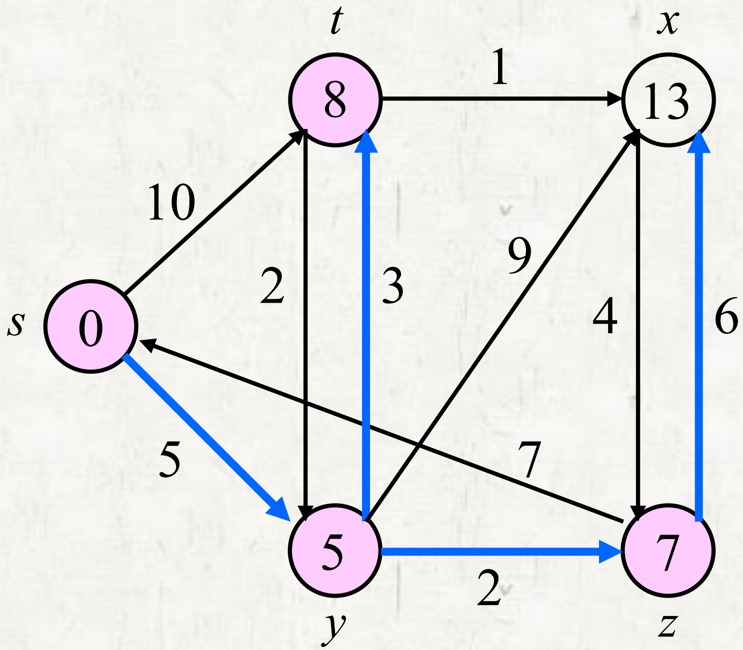


$S = \{s, y, z\}$

Dijkstra's Algorithm

Q

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

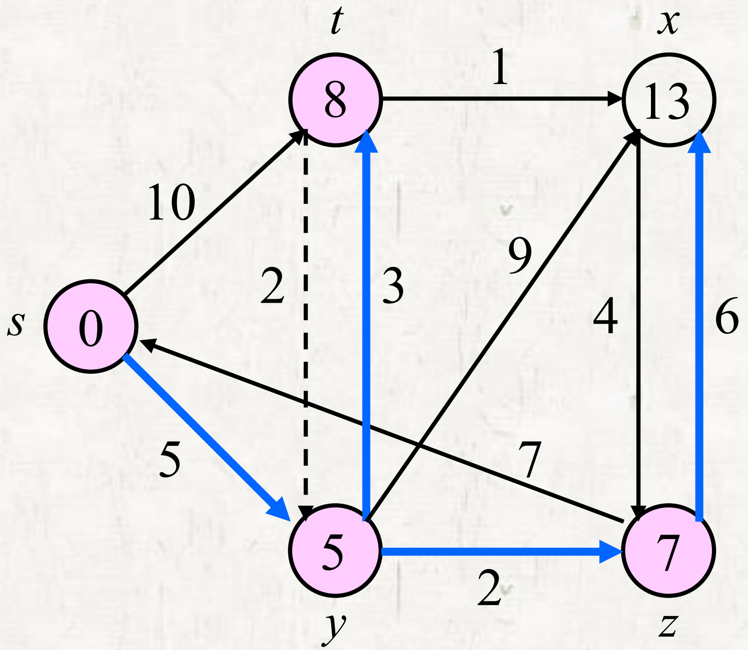


$\boldsymbol{S} = \{s, y, z, t\}$

Dijkstra's Algorithm

Q

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

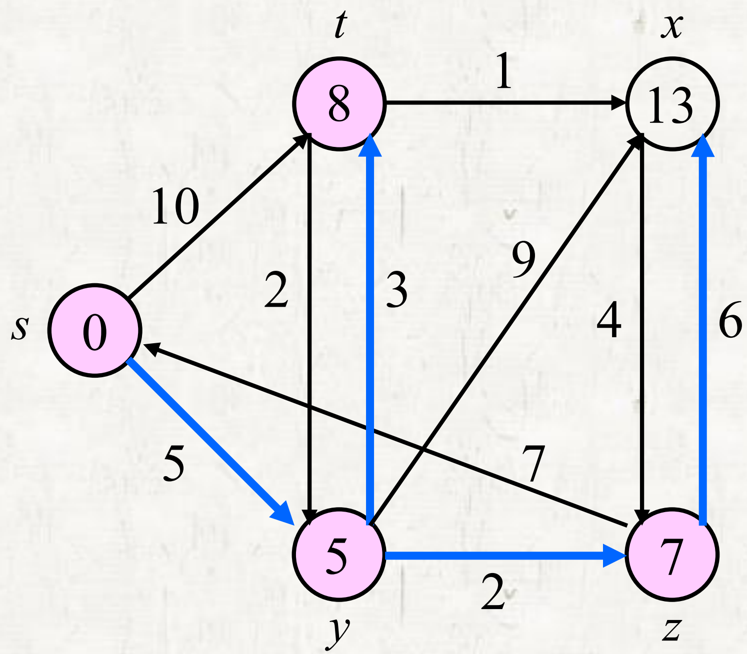


$\boldsymbol{S} = \{s, y, z, t\}$

Dijkstra's Algorithm

Q

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

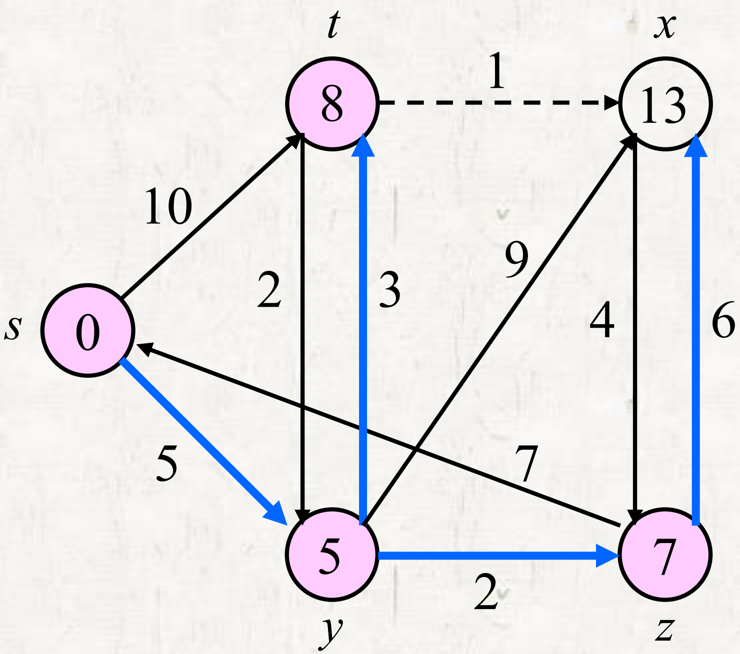


$\boldsymbol{S} = \{s, y, z, t\}$

Dijkstra's Algorithm

Q

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

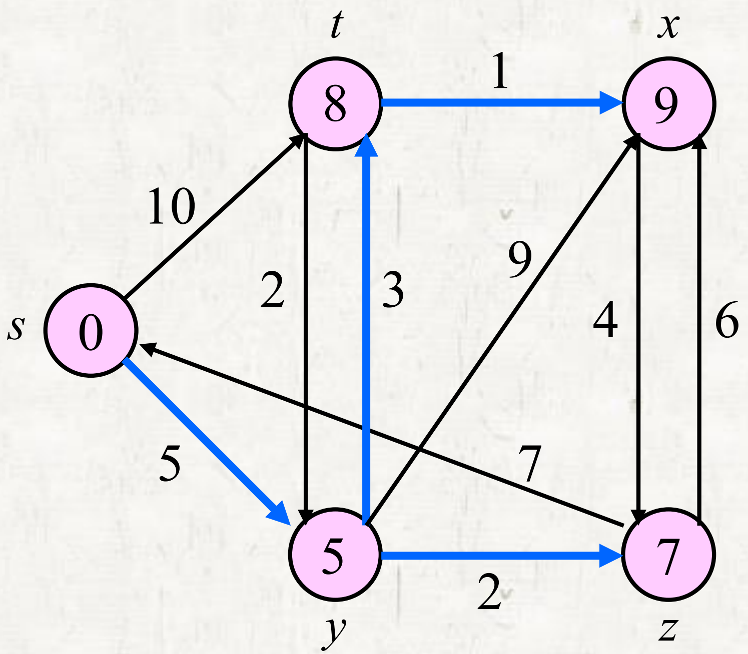


$\boldsymbol{S} = \{s, y, z, t\}$

Dijkstra's Algorithm

Q

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

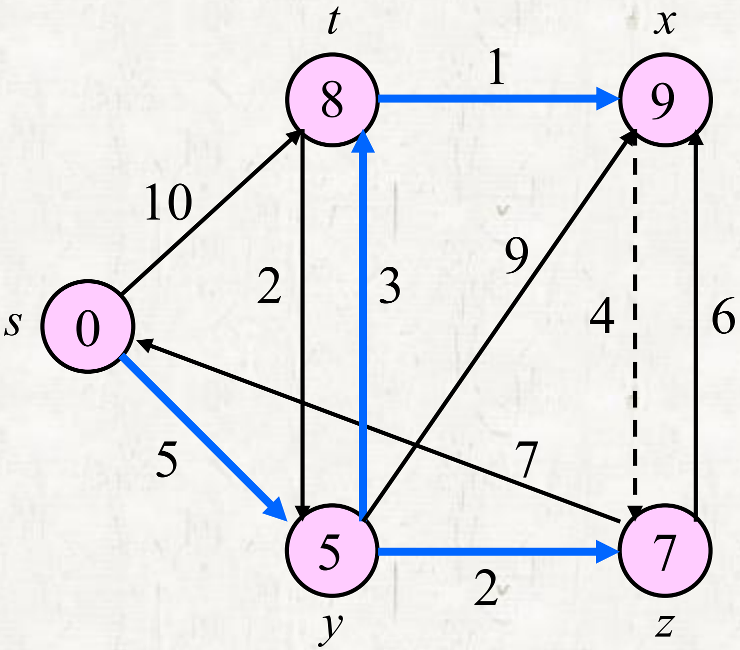


$\boldsymbol{S} = \{s, y, z, t, x\}$

Dijkstra's Algorithm

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

Running Time

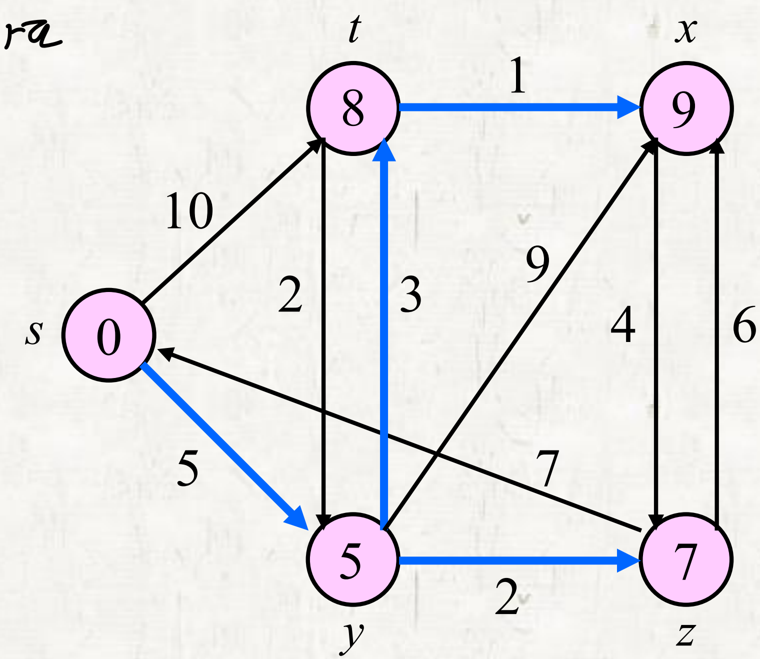
$V \times O(V)$: 각 vertex마다

$\therefore O(V^2)$

Q

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

V



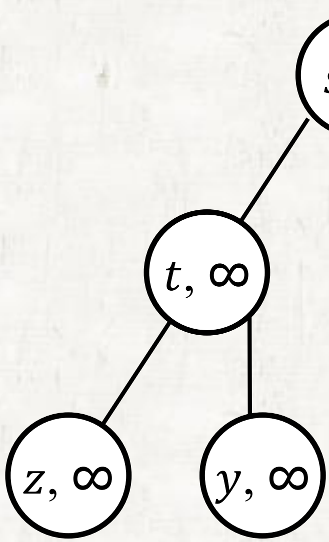
→ 모든 vertex 선택되었으면 종료

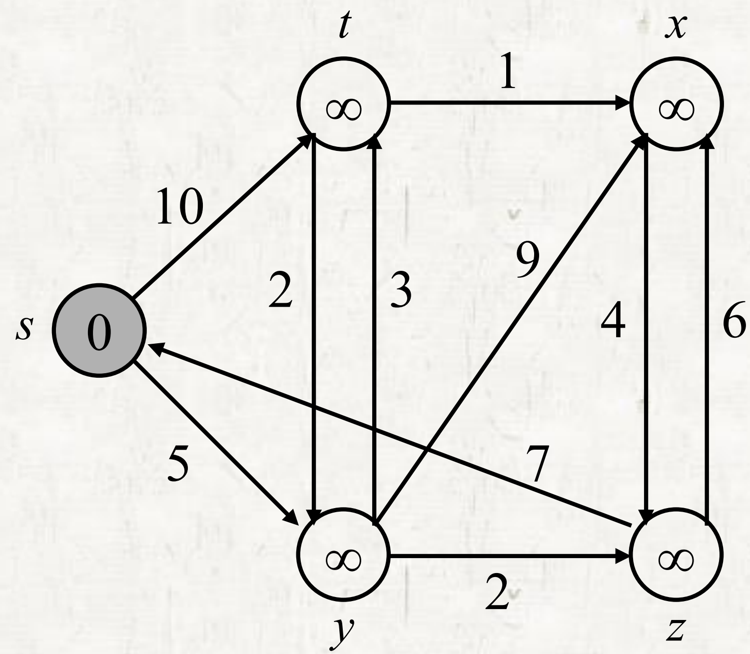
$\boldsymbol{S} = \{s, y, z, t, x\}$

Dijkstra's Algorithm

Reduce Running Time → Using Min heap

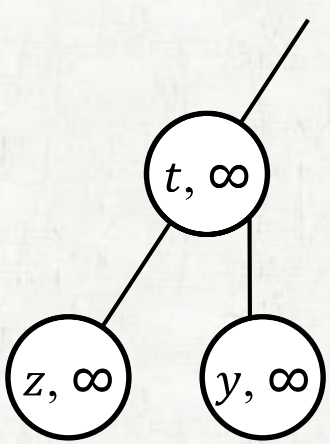
s	t	y	x	z

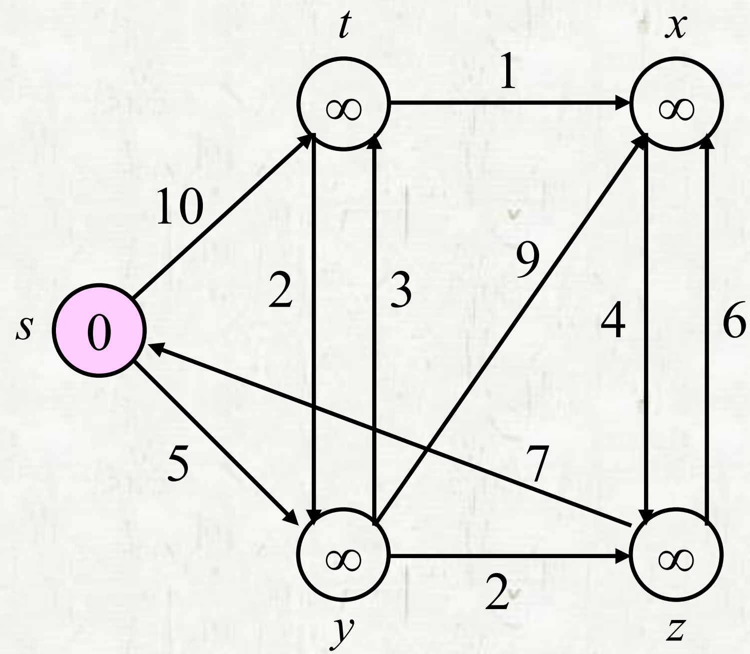




Dijkstra's Algorithm

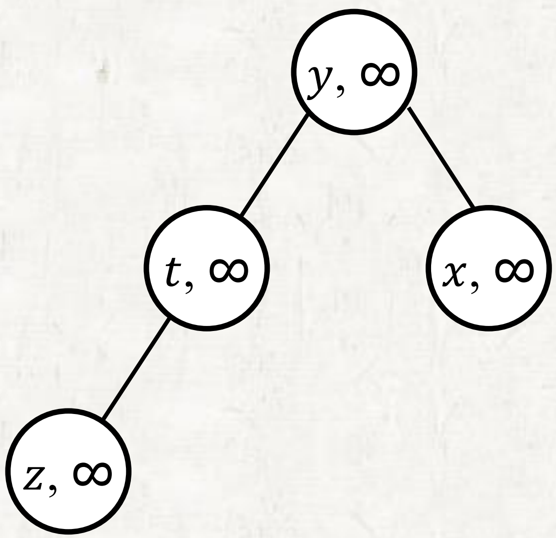
s	t	y	x	z

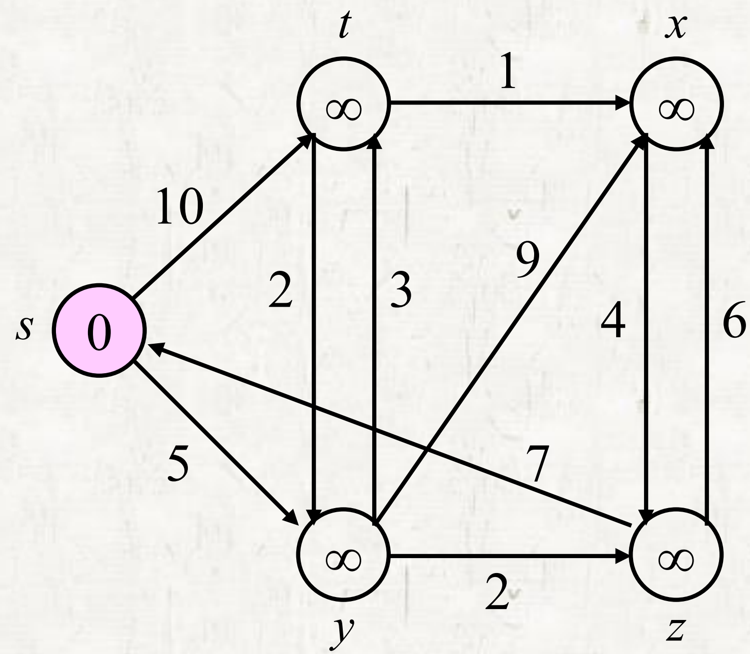




Dijkstra's Algorithm

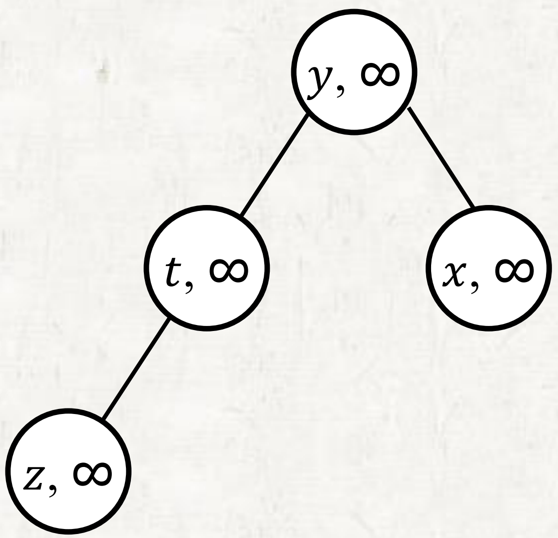
s	*t*	*y*	*x*	*z*

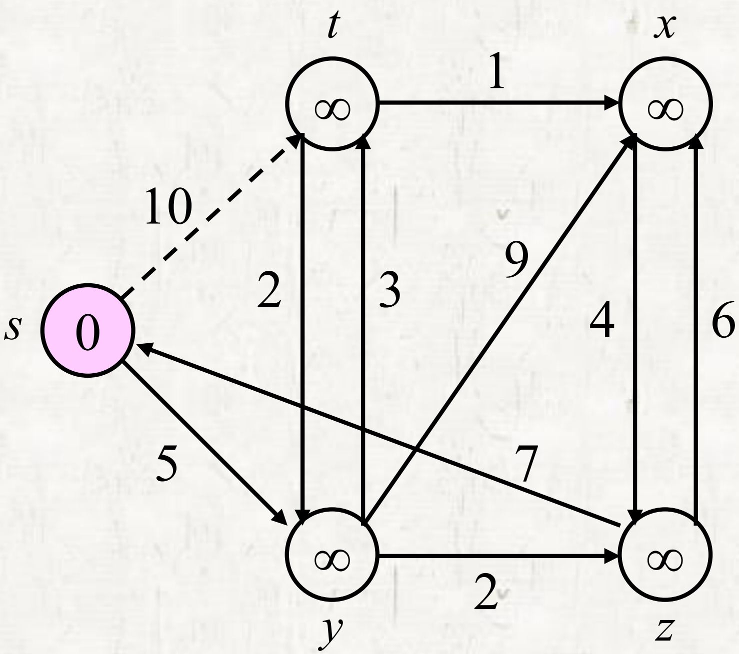




Dijkstra's Algorithm

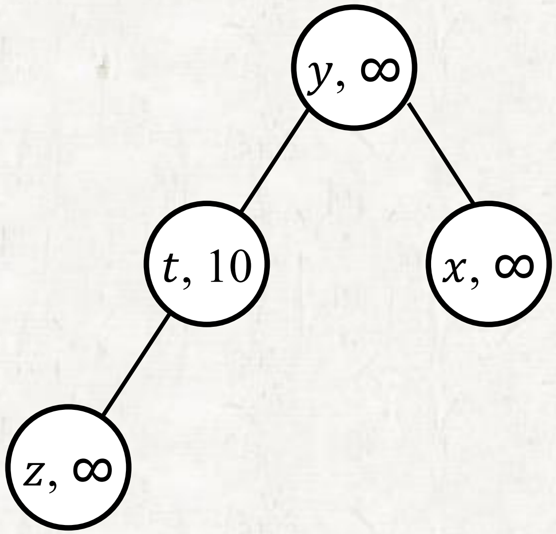
s	t	y	x	z

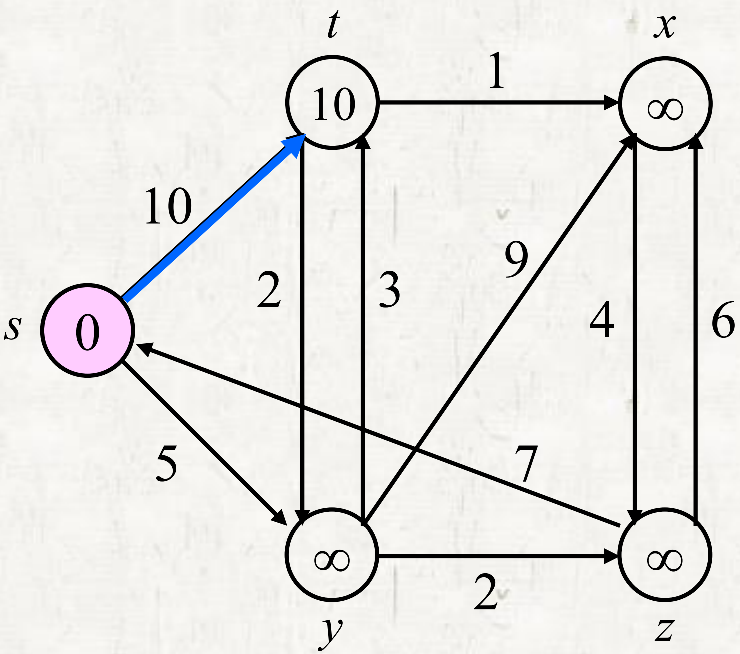




Dijkstra's Algorithm

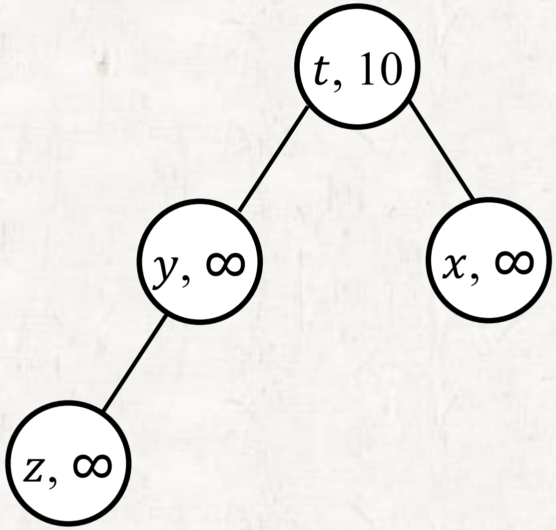
s	t	y	x	z
	s			

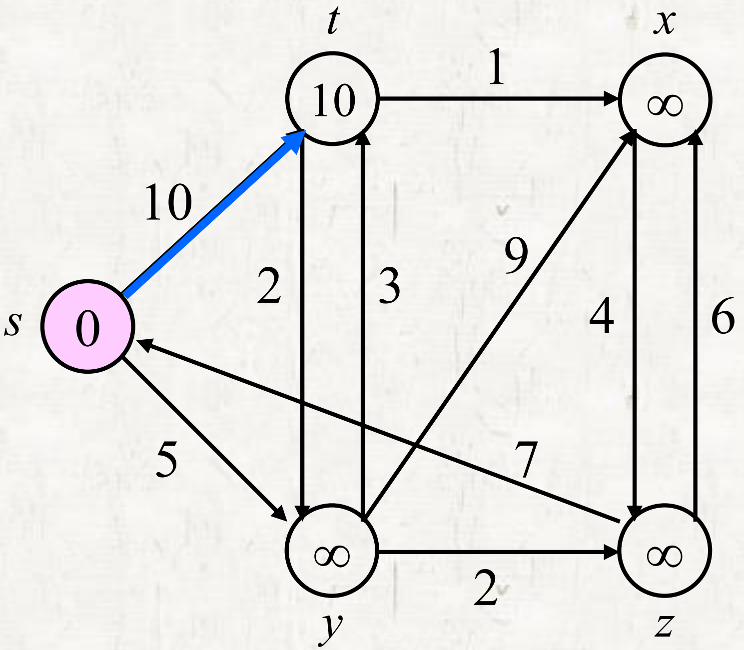




Dijkstra's Algorithm

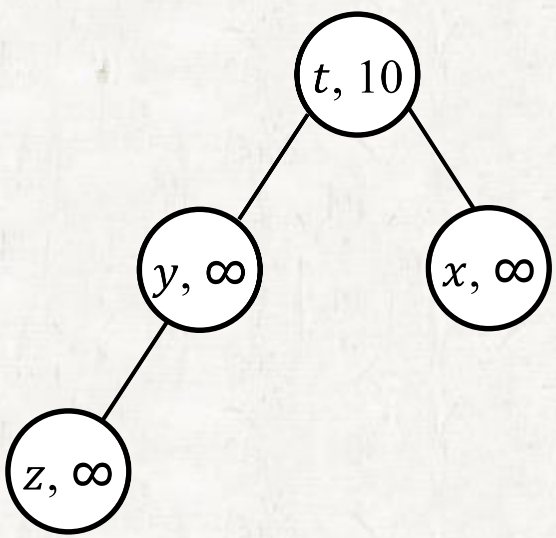
s	*t*	*y*	*x*	*z*
	s			

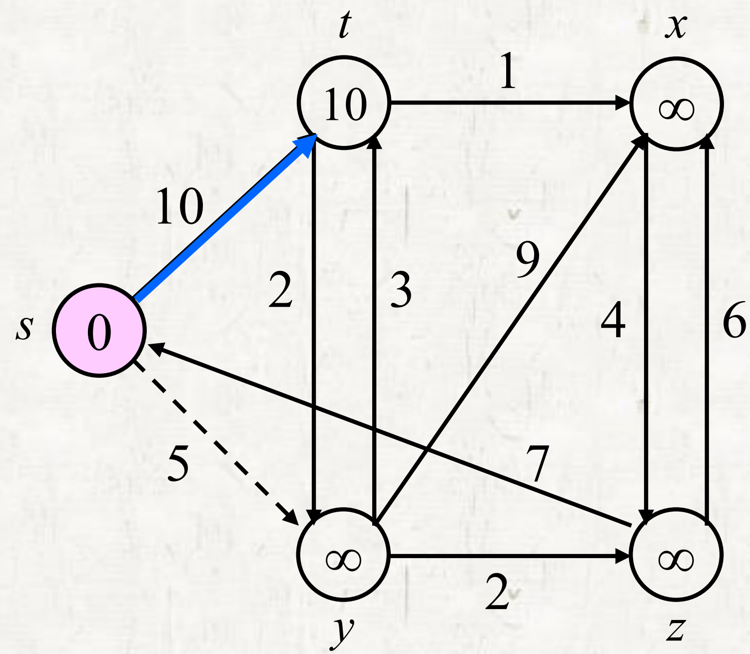




Dijkstra's Algorithm

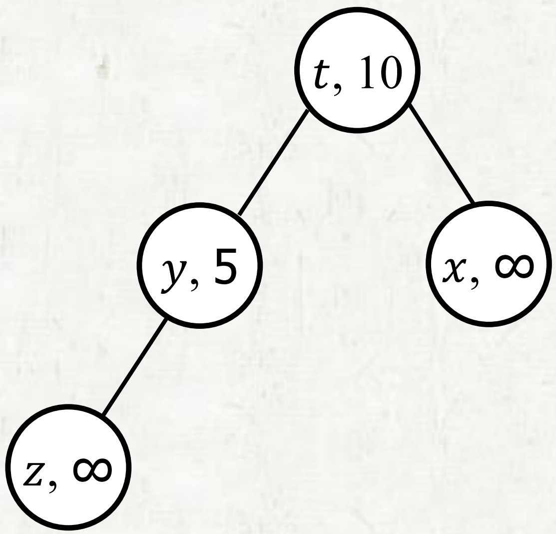
s	t	y	x	z
	s			

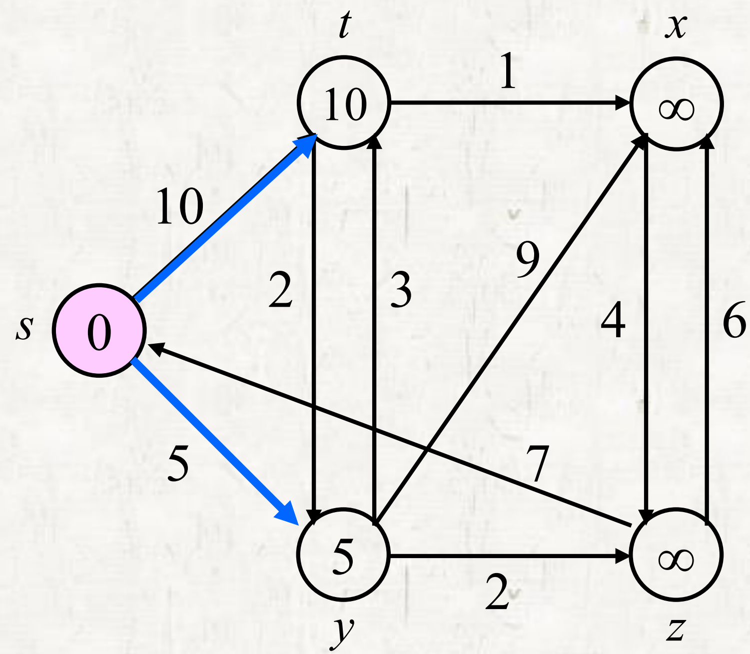




Dijkstra's Algorithm

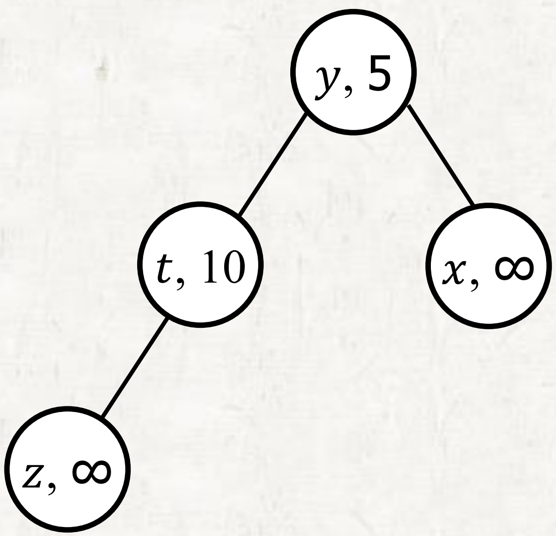
s	*t*	*y*	*x*	*z*
	s	*s*		

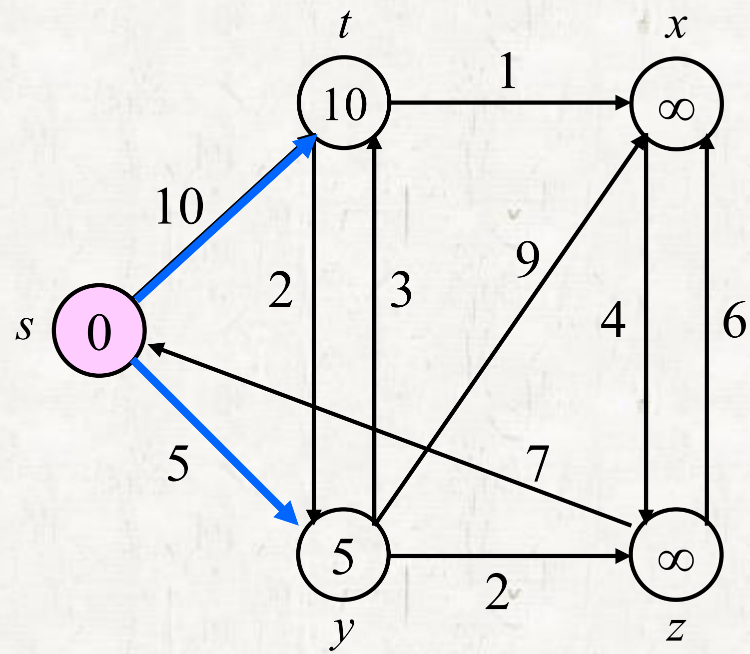




Dijkstra's Algorithm

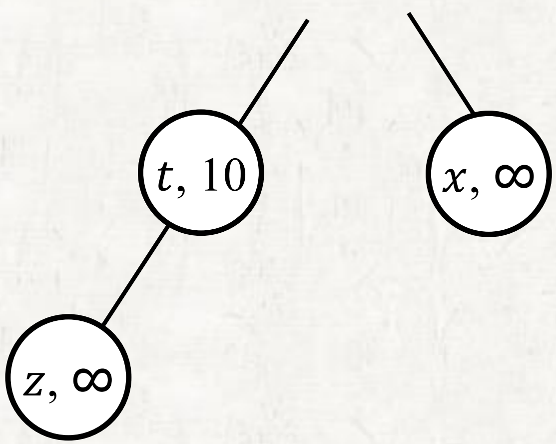
s	t	y	x	z
	s	s		

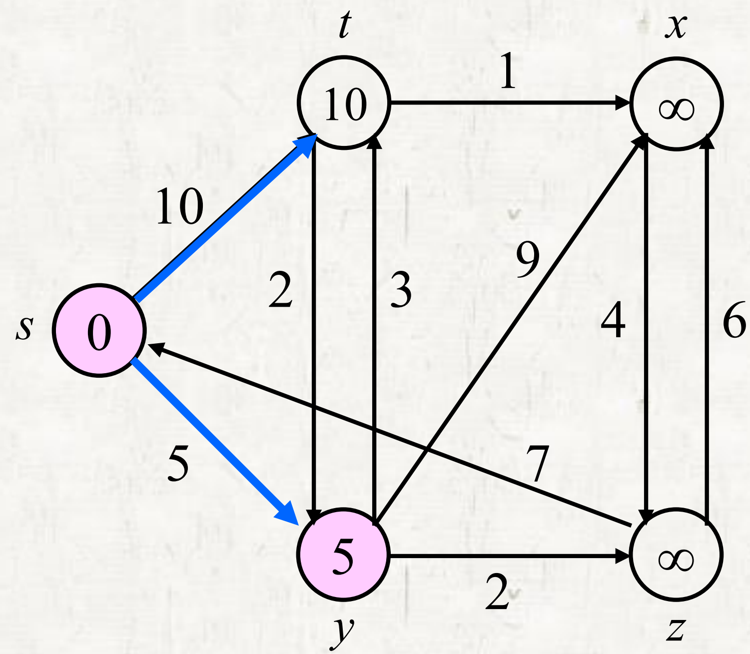




Dijkstra's Algorithm

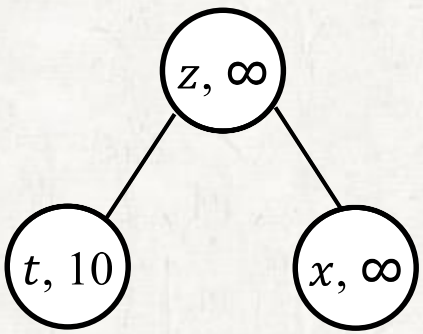
s	*t*	*y*	*x*	*z*
	s	*s*		

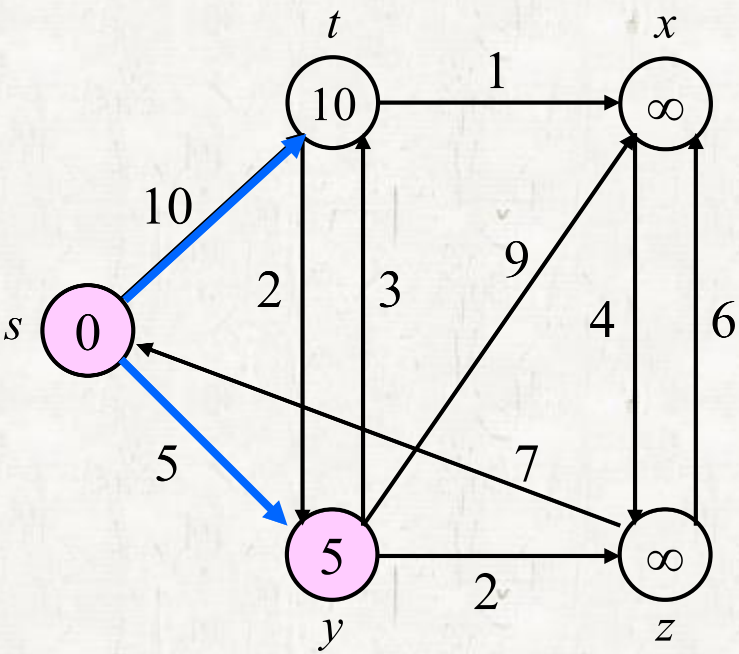




Dijkstra's Algorithm

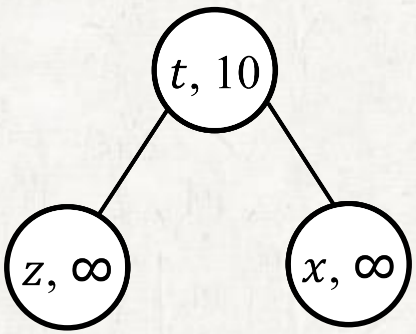
s	t	y	x	z
	s	s		

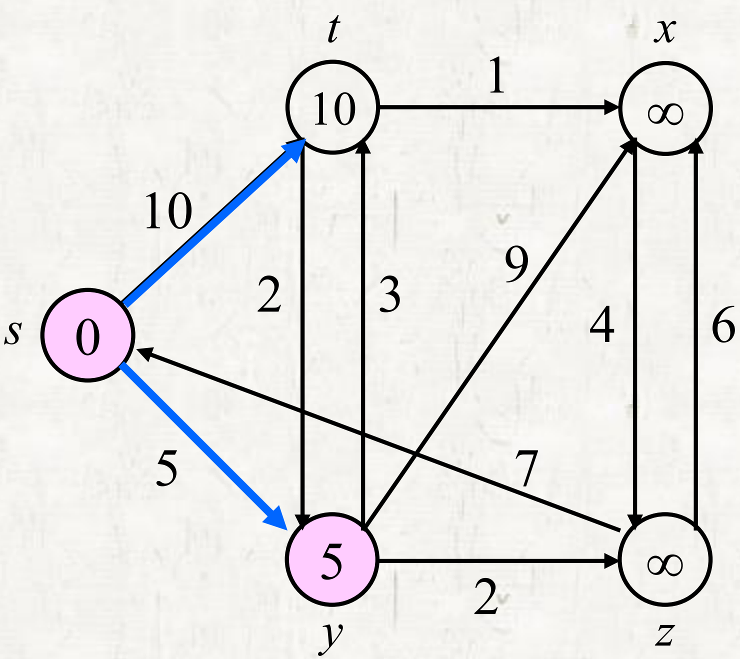




Dijkstra's Algorithm

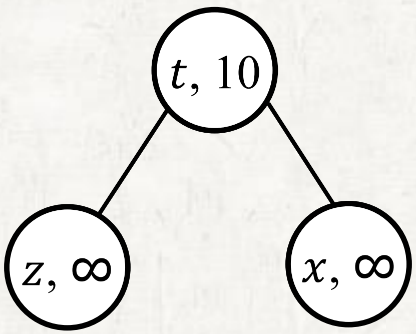
s	*t*	*y*	*x*	*z*
	s	*s*		

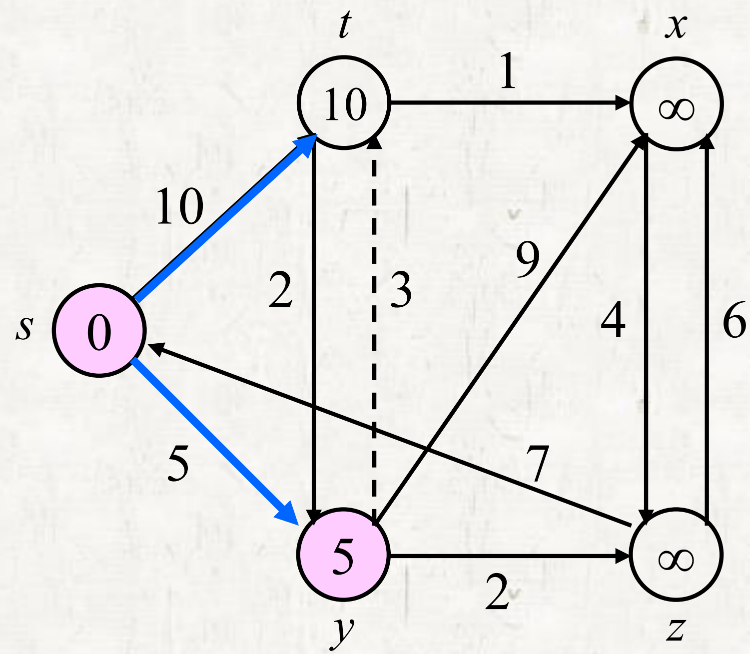




Dijkstra's Algorithm

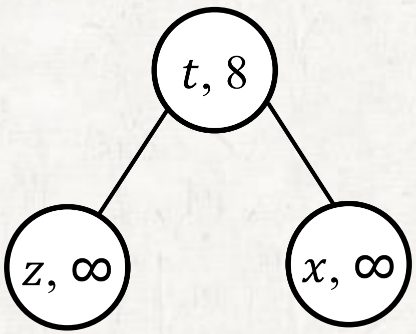
s	t	y	x	z
	s	s		

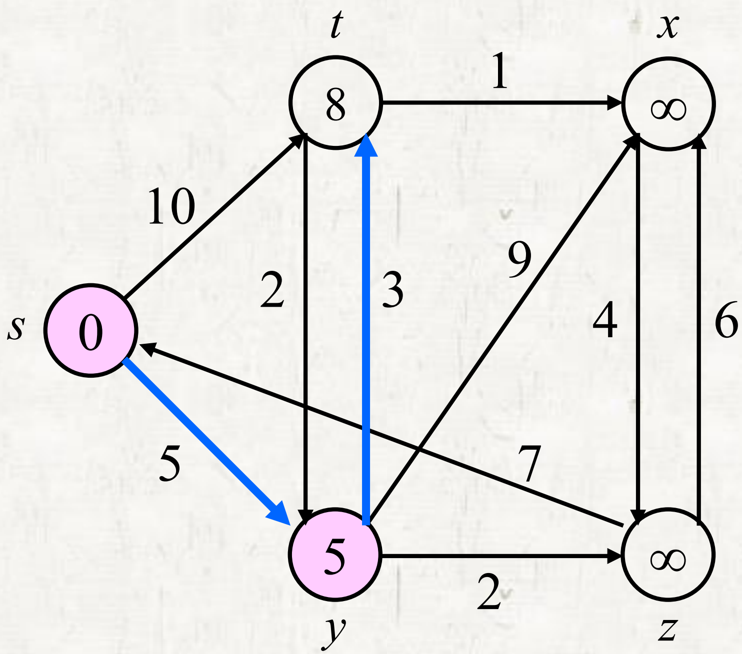




Dijkstra's Algorithm

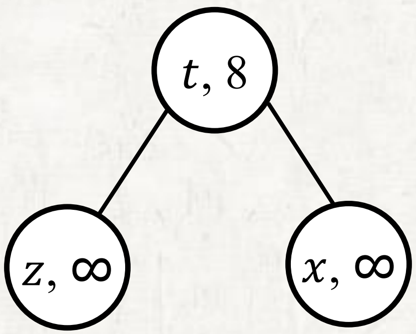
s	*t*	*y*	*x*	*z*
	y	*s*		

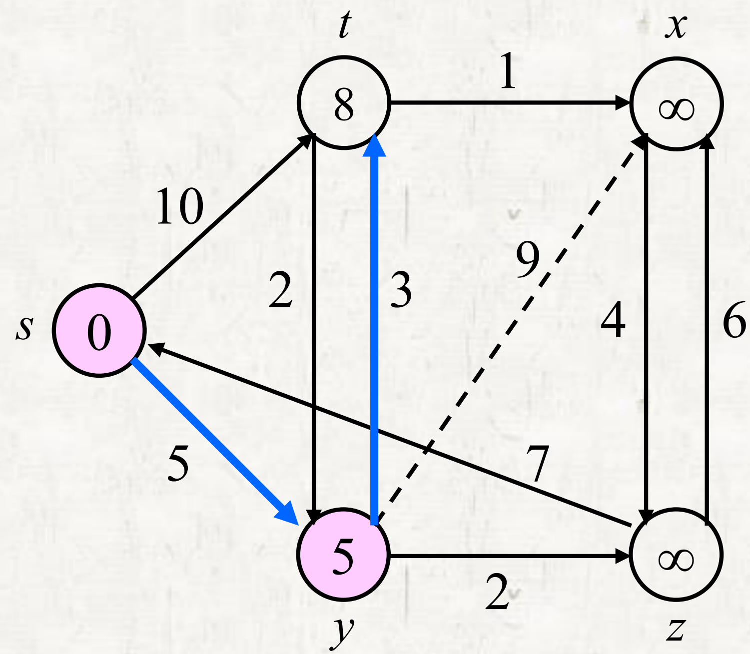




Dijkstra’s Algorithm

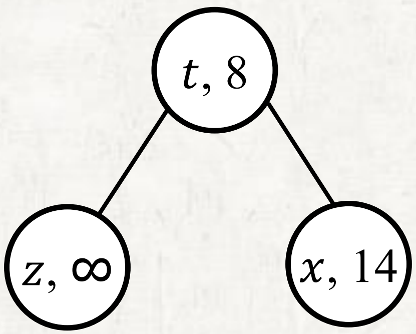
s	t	y	x	z
	y	s		

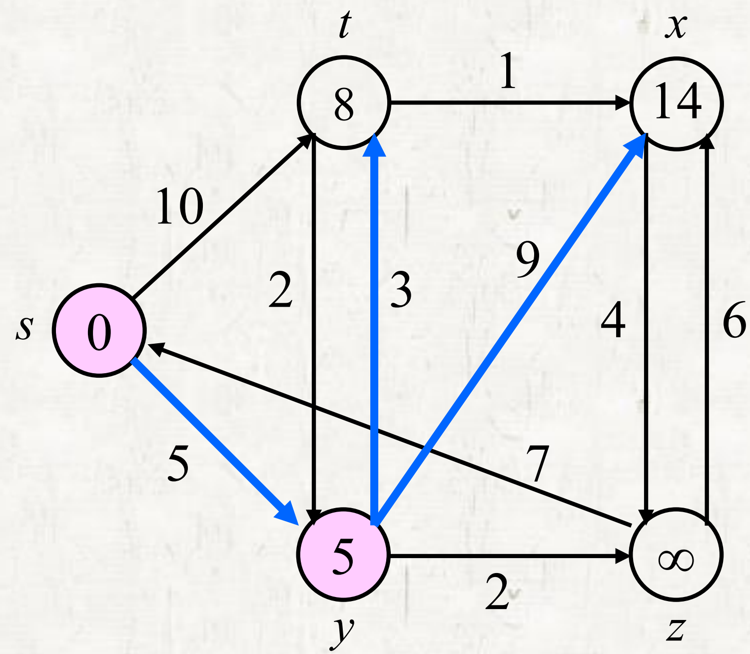




Dijkstra's Algorithm

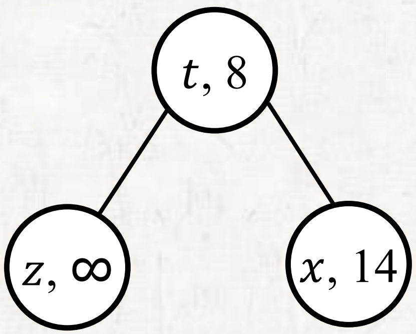
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	

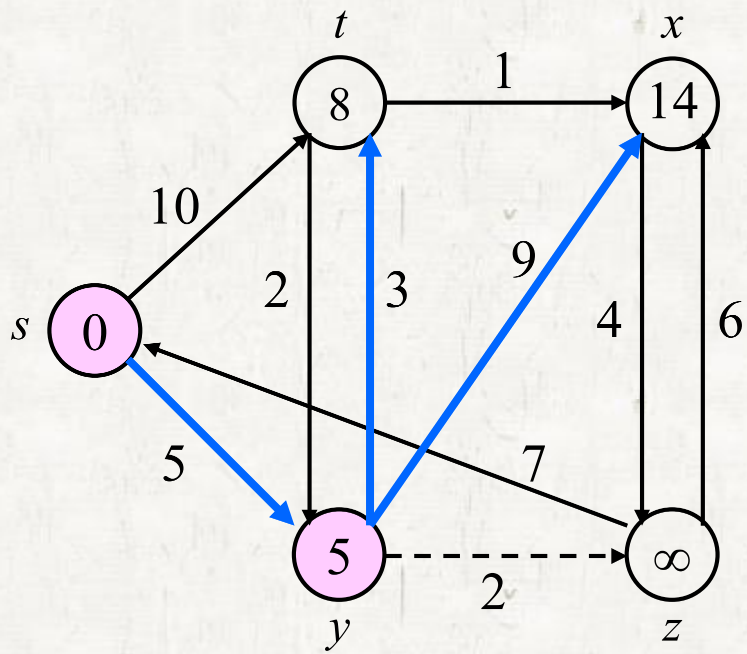




Dijkstra's Algorithm

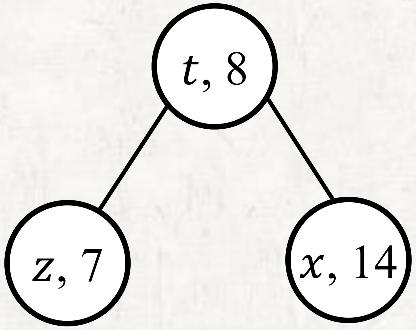
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	

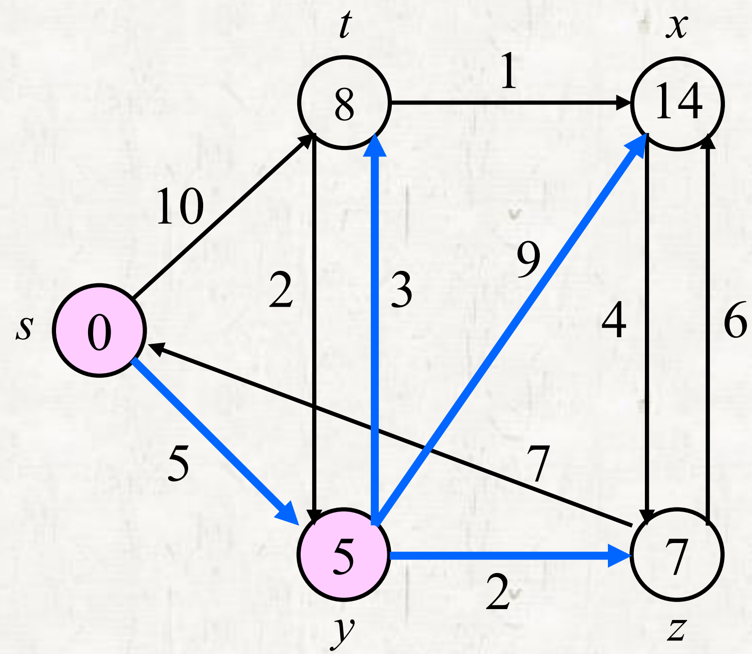




Dijkstra's Algorithm

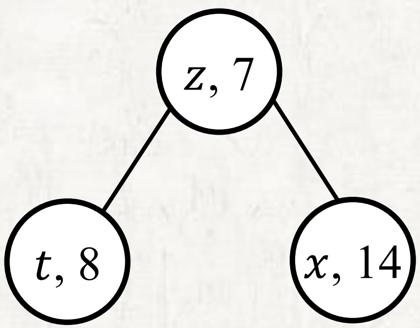
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	*y*

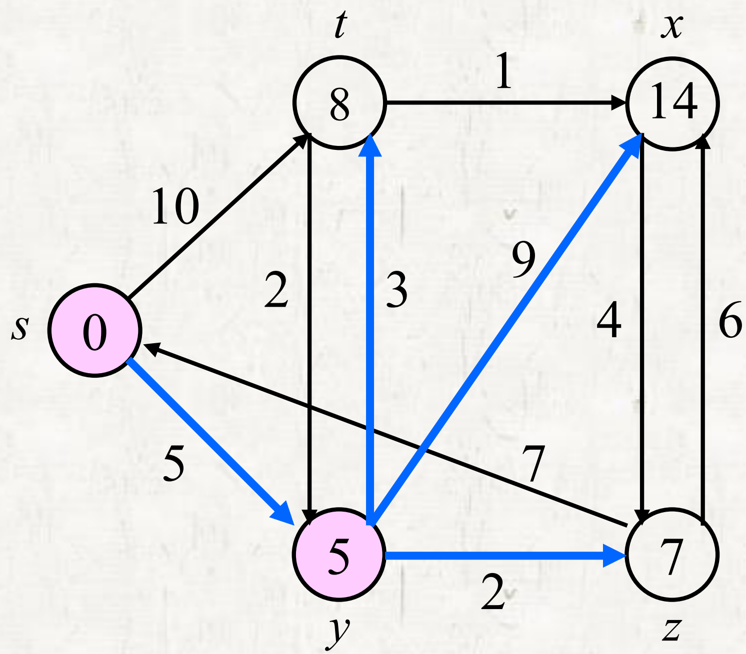




Dijkstra's Algorithm

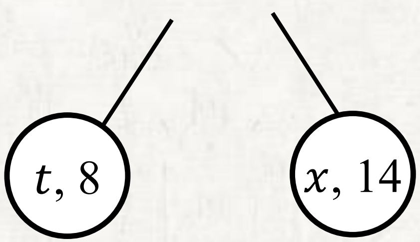
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	*y*

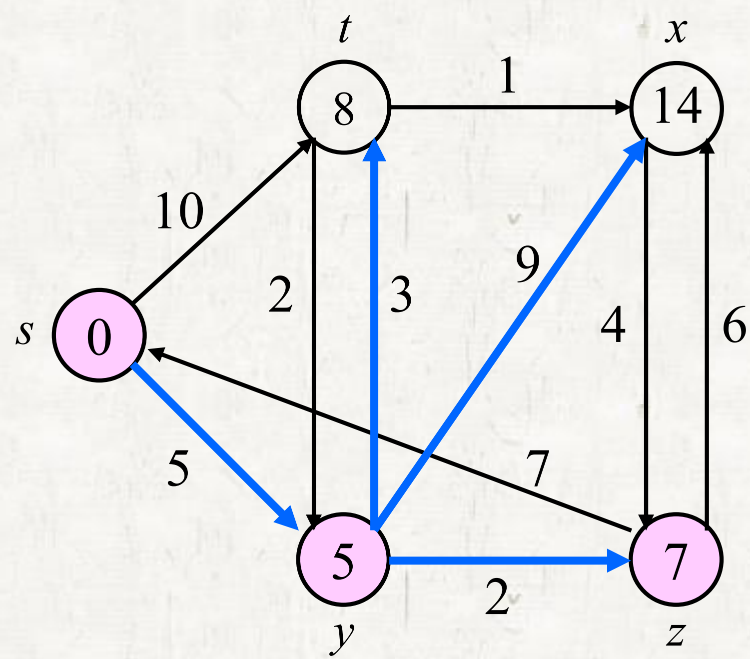




Dijkstra's Algorithm

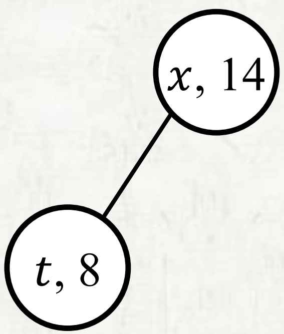
s	t	y	x	z
	y	s	y	y

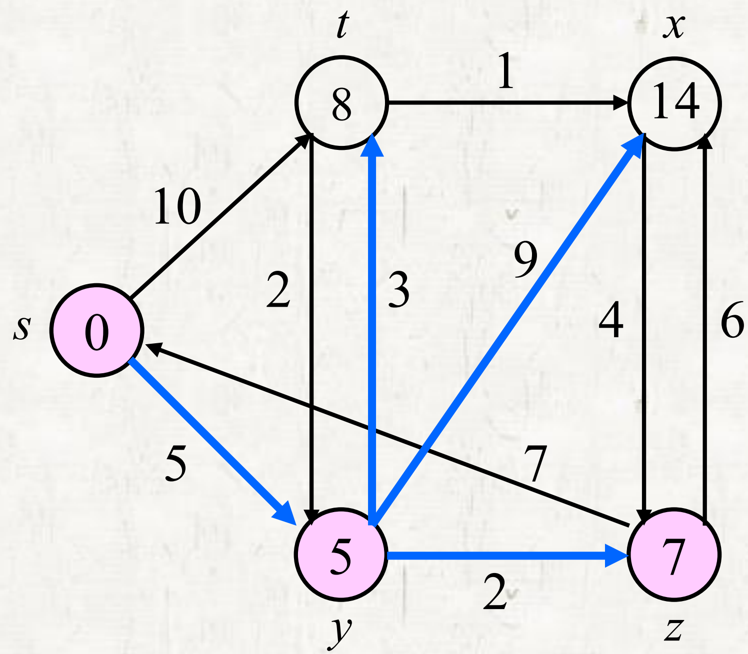




Dijkstra’s Algorithm

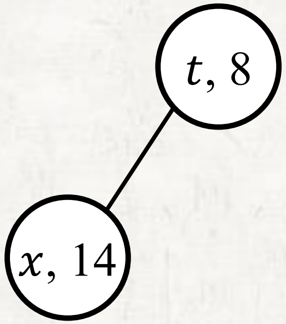
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	*y*

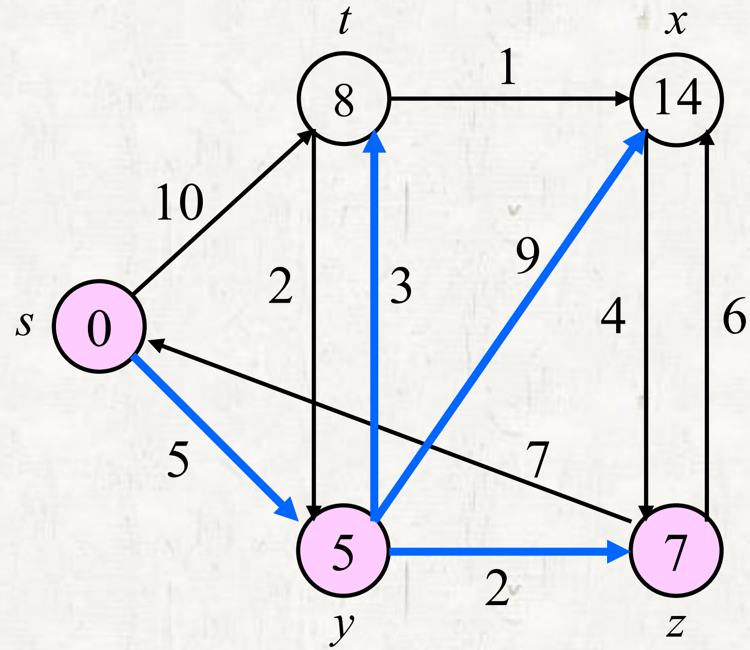




Dijkstra's Algorithm

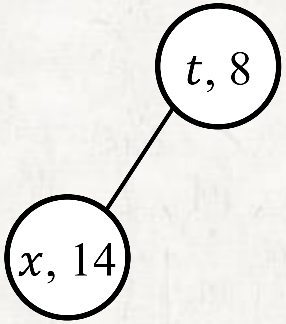
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	*y*

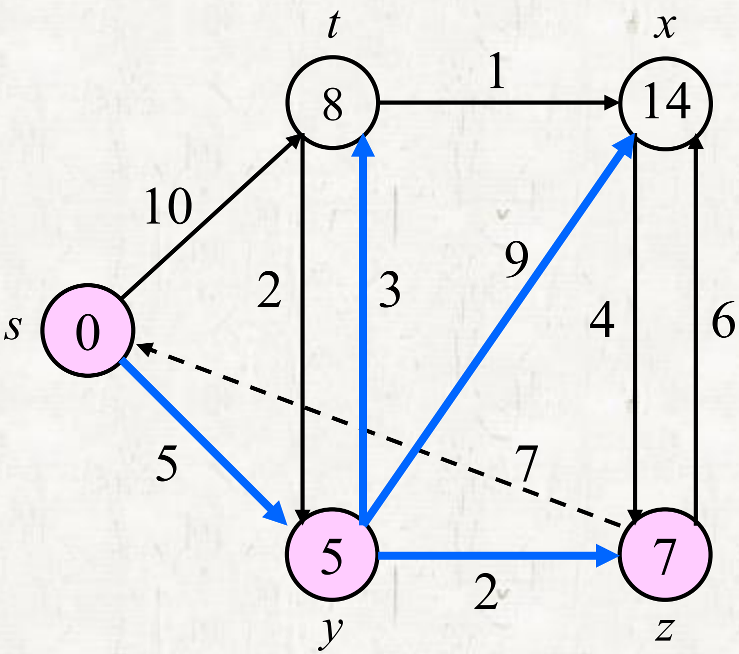




Dijkstra's Algorithm

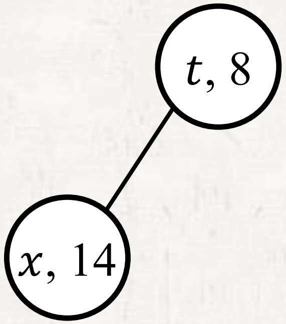
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	*y*

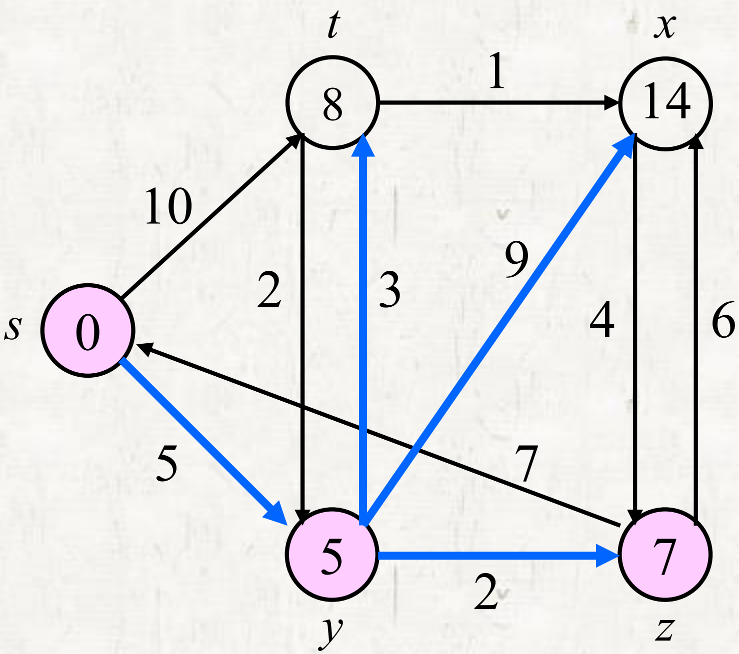




Dijkstra's Algorithm

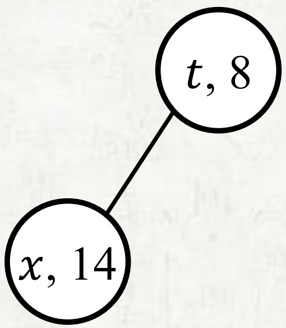
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	*y*

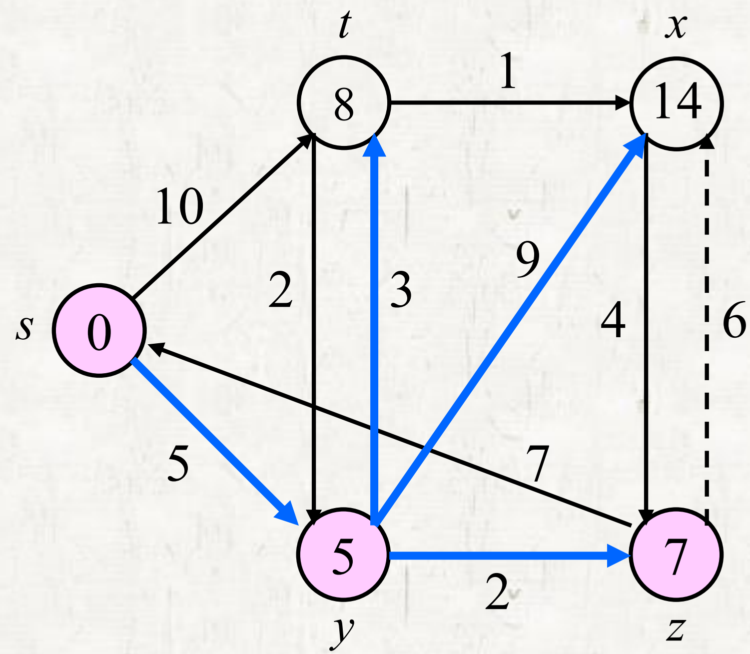




Dijkstra's Algorithm

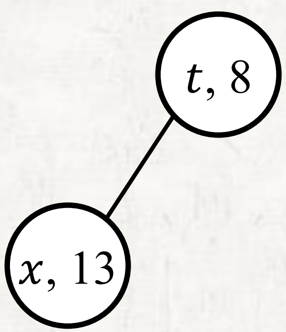
s	*t*	*y*	*x*	*z*
	y	*s*	*y*	*y*

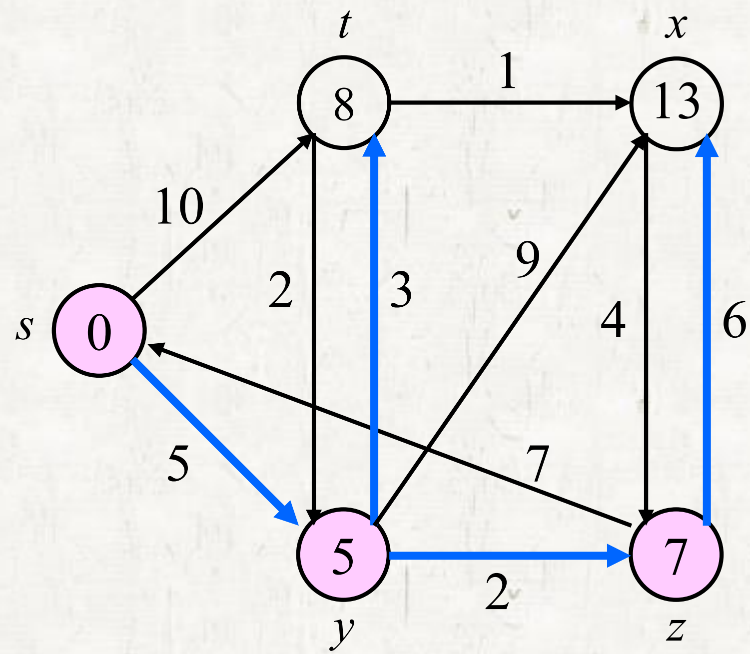




Dijkstra’s Algorithm

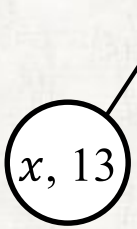
s	*t*	*y*	*x*	*z*
	y	*s*	*z*	*y*

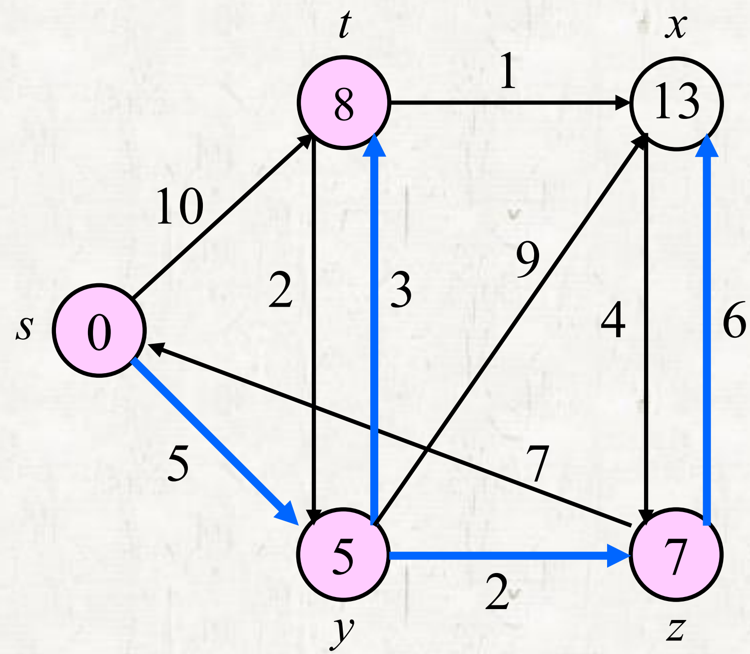




Dijkstra's Algorithm

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

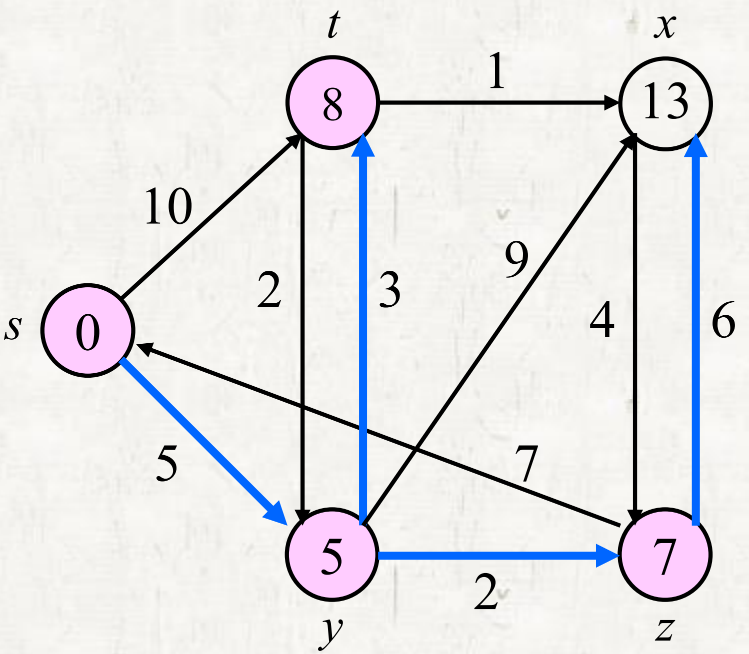




Dijkstra's Algorithm

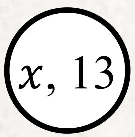
s	*t*	*y*	*x*	*z*
	y	*s*	*z*	*y*

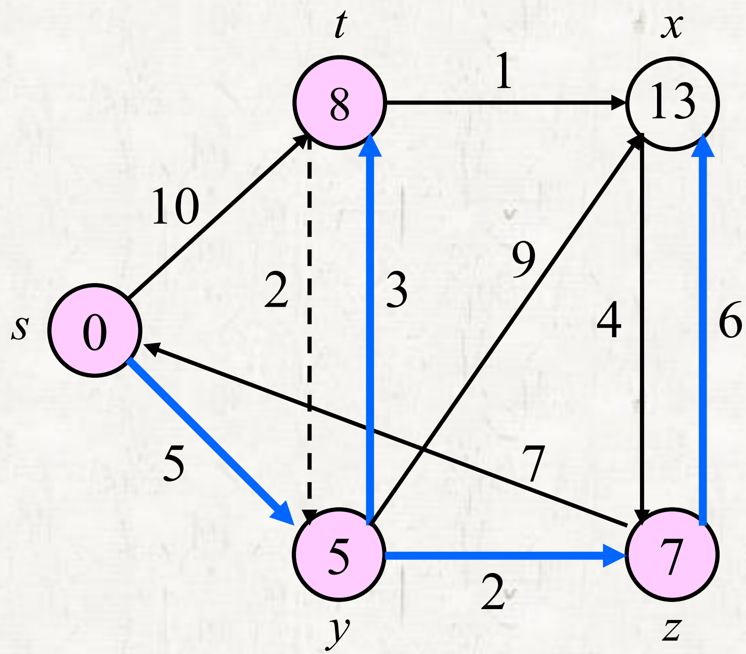
x, 13



Dijkstra's Algorithm

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

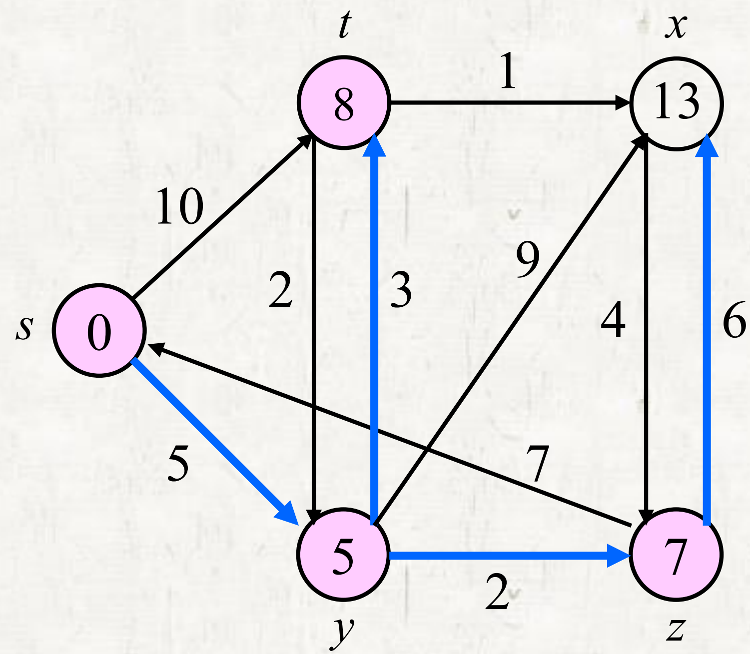




Dijkstra's Algorithm

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

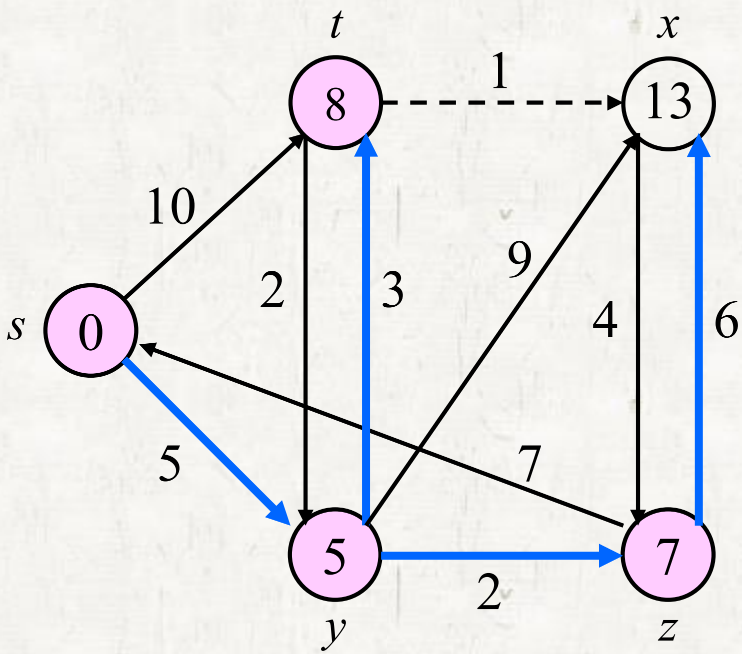
x, 13



Dijkstra's Algorithm

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

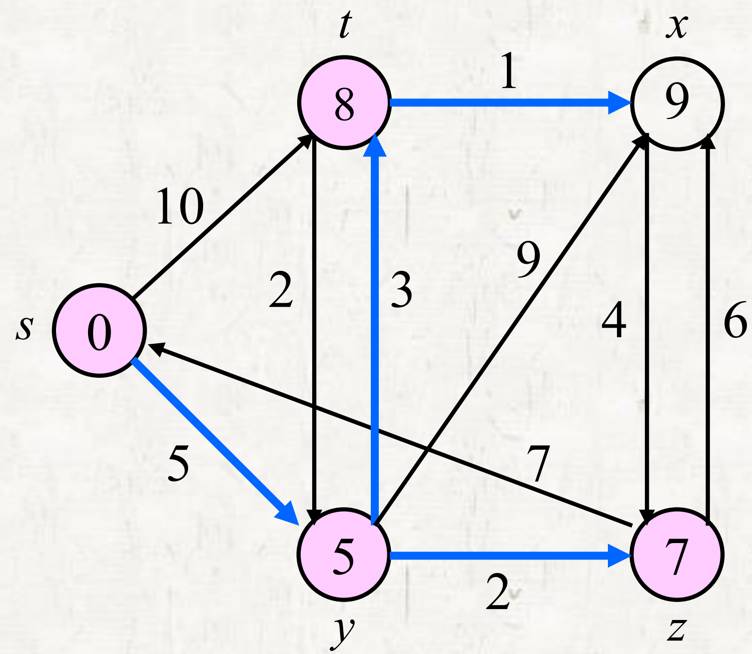
x, 13



Dijkstra's Algorithm

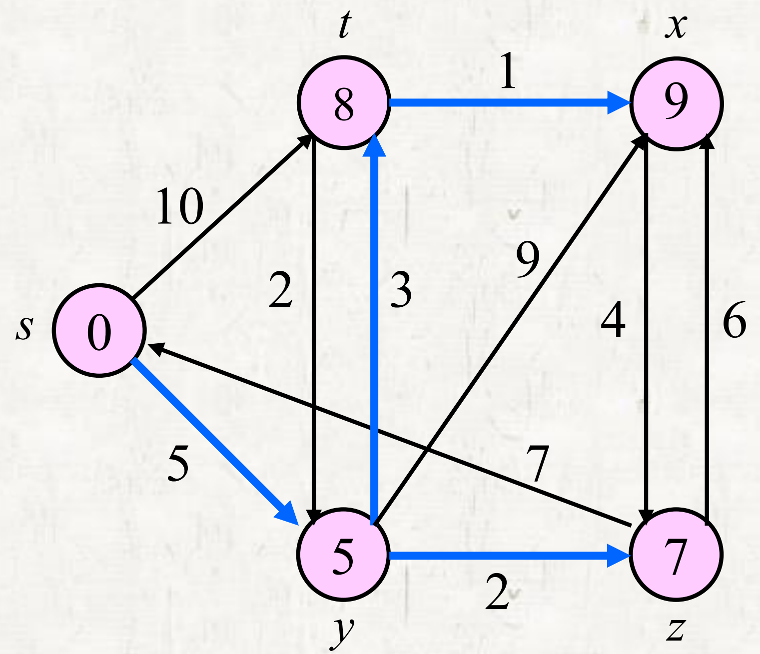
s	*t*	*y*	*x*	*z*
	y	*s*	*t*	*y*

x, 9



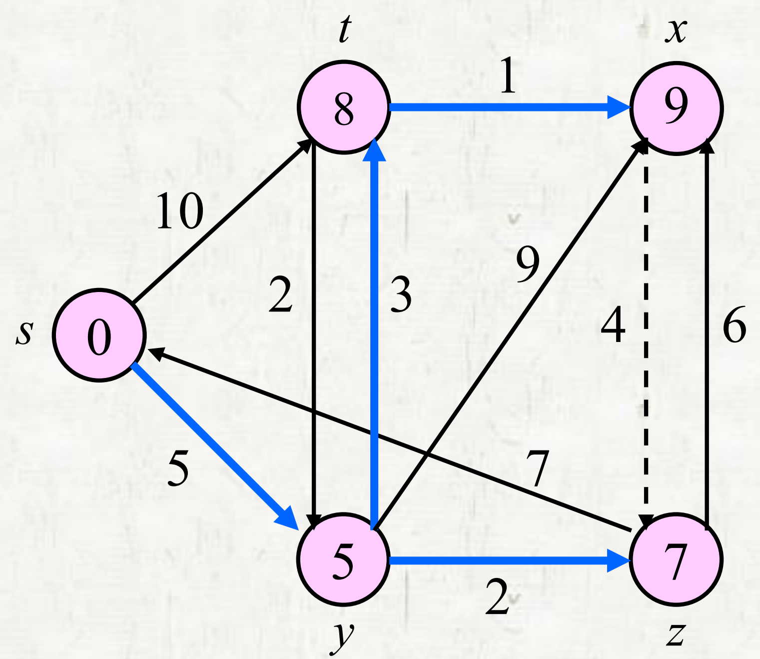
Dijkstra's Algorithm

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



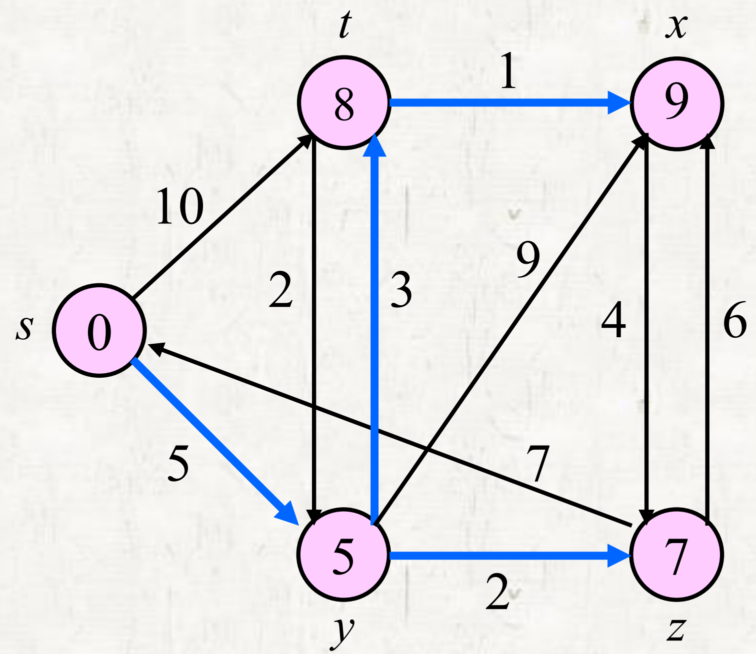
Dijkstra's Algorithm

s	t	y	x	z
	y	s	t	y



Dijkstra's Algorithm

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



Dijkstra's algorithm

using heap

DIJKSTRA(G, w, s)

O(V+E)
INITIALIZE-SINGLE-SOURCE(G, s) — Initialize set
$S = \emptyset$
$Q = G.V$

while $Q \neq \emptyset$

V log V ←
   u = EXTRACT-MIN(Q)

O(V+E)
   $S = S \cup \{u\}$
   for each vertex $v \in G.Adj[u]$

O(E log V) ←
      RELAX(u, v, w) → DECREASE_KEY(v, d[v])를 수행

Dijkstra's algorithm

- **Running time**
  - ① $O(V^2)$ if we use an (unsorted) array
  - ② $O(V\lg V + E\lg V)$ if we use a heap → 모두 connected 되었다는 가정 하에 $E \approx V-1$ 이 가능 → $V\log V$ 유지
  - $O(V\lg V + E)$ if we use a Fibonacci heap.

Best case, 어려움

보통 Heap 사용시 좋음

하지만 Edge가 매우 많다면 $E \approx V^2$ → Array 사용이 나음

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

weight, source

```
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
```

Θ(VE)

relax all edge

RELAX가 수행

→ $v.d = u.d + w(u, v)$

Θ(E)

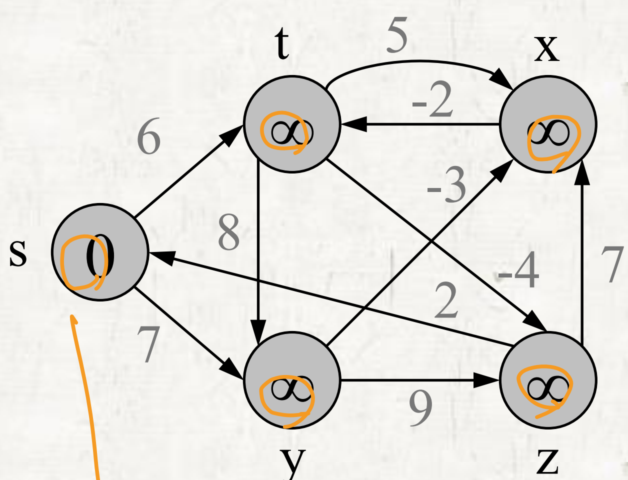
The Bellman-Ford algorithm

- Relaxation order

매 단계마다 이순서로 모든 edge를 Relax

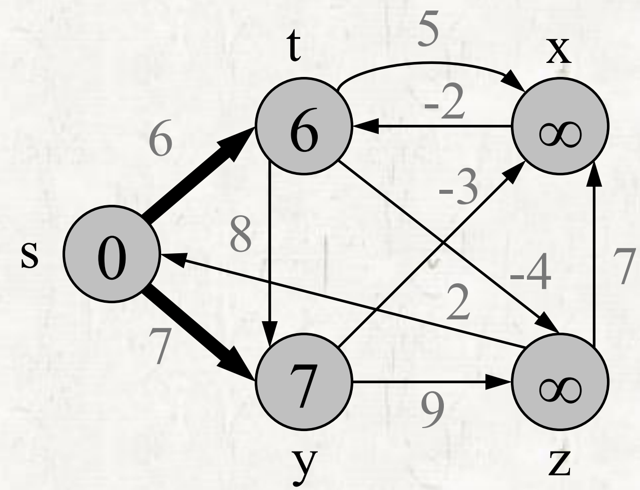
  - (t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)

Relax 진행 순서



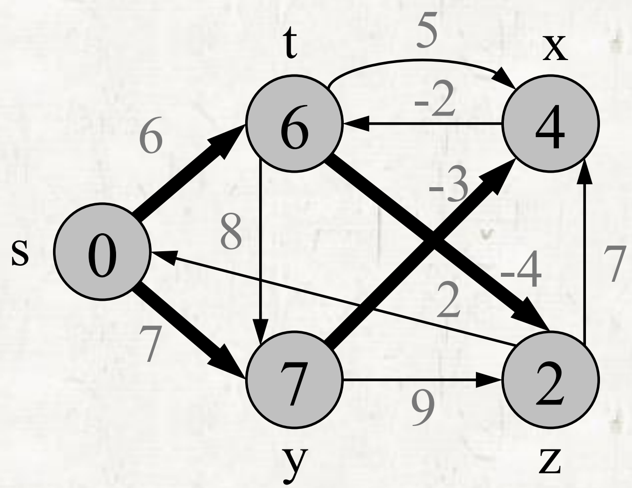
Source는 0

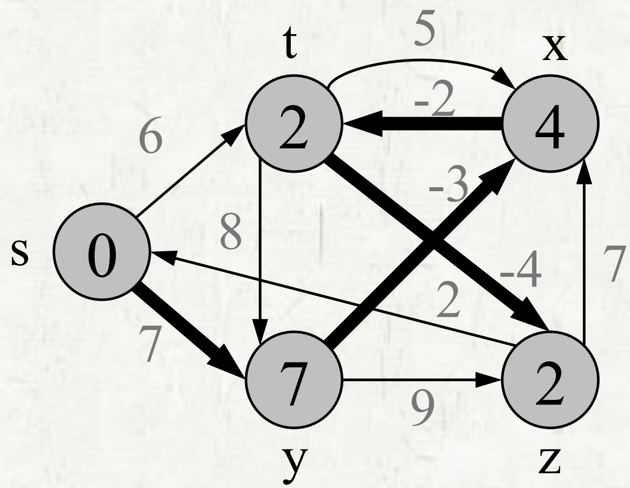
→ Source의 Adj[ ]만 update 된다.



The Bellman-Ford algorithm

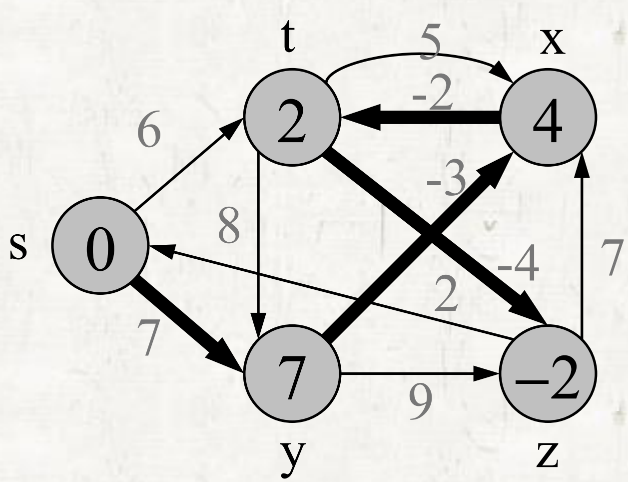
- Relaxation order
  - (t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)





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

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

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

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

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

→ Negative edge가 없다면
이 조건을 만족해야 한다.

Single-source shortest paths in directed acyclic graphs

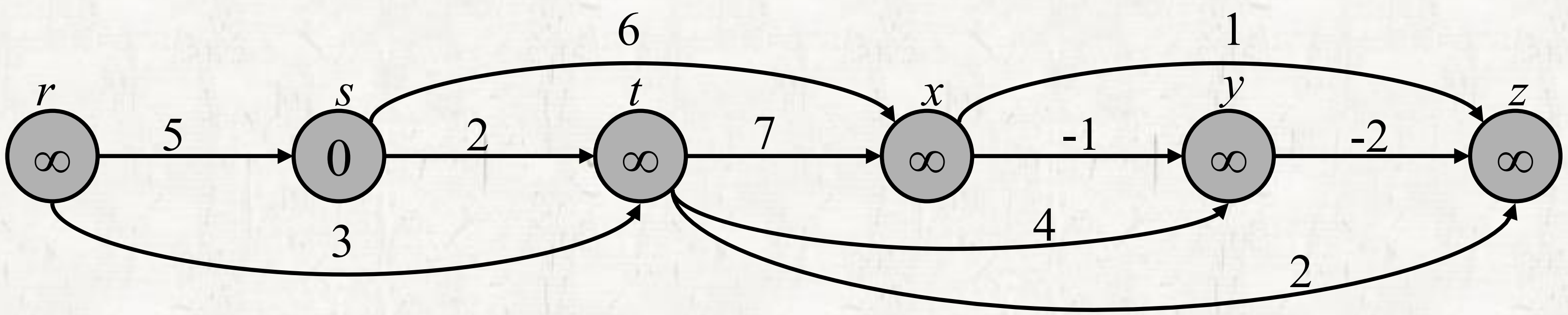
DAG-SHORTEST-PATHS(*G*, *w*, *s*)

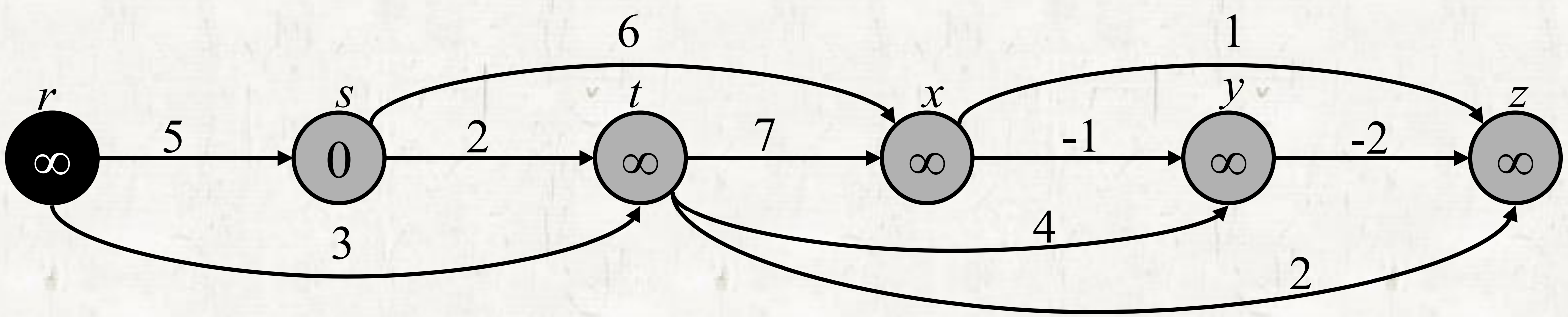
```
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Running Time: $\Theta(V+E)$

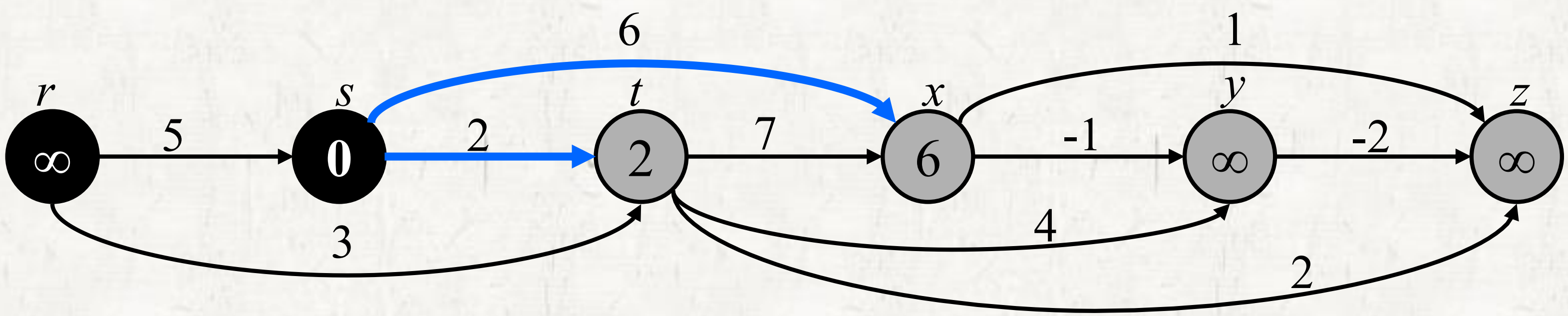
Dijkstra
Bellman-Ford) 보다 빠름

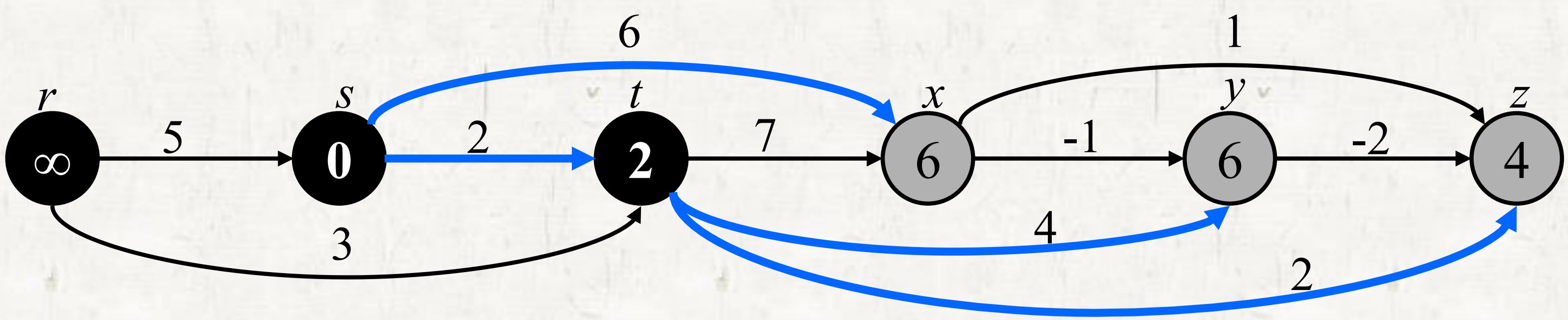
Single-source shortest paths in directed acyclic graphs



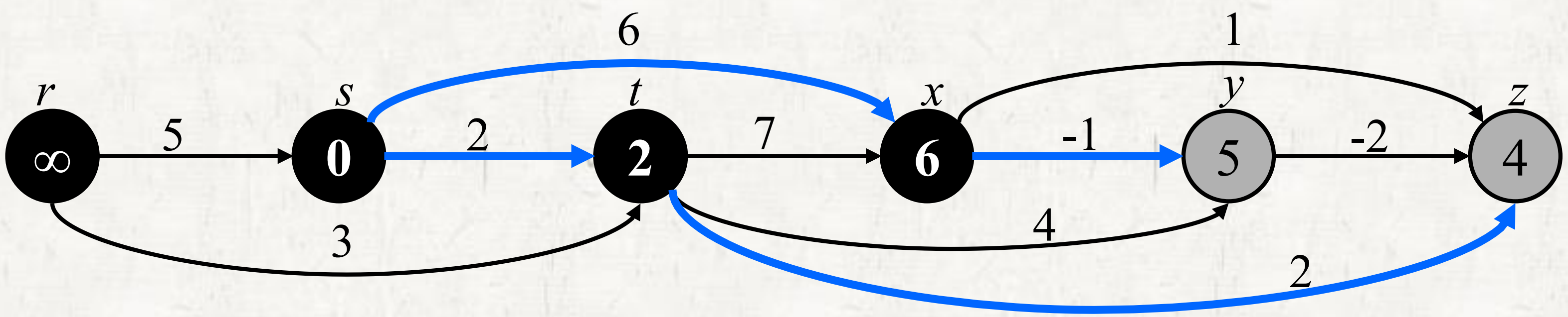


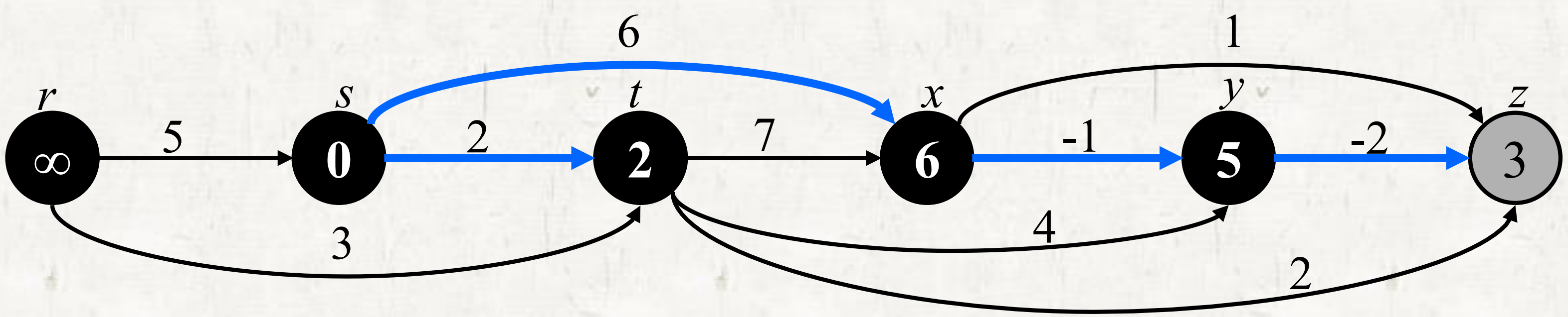
Single-source shortest paths in directed acyclic graphs



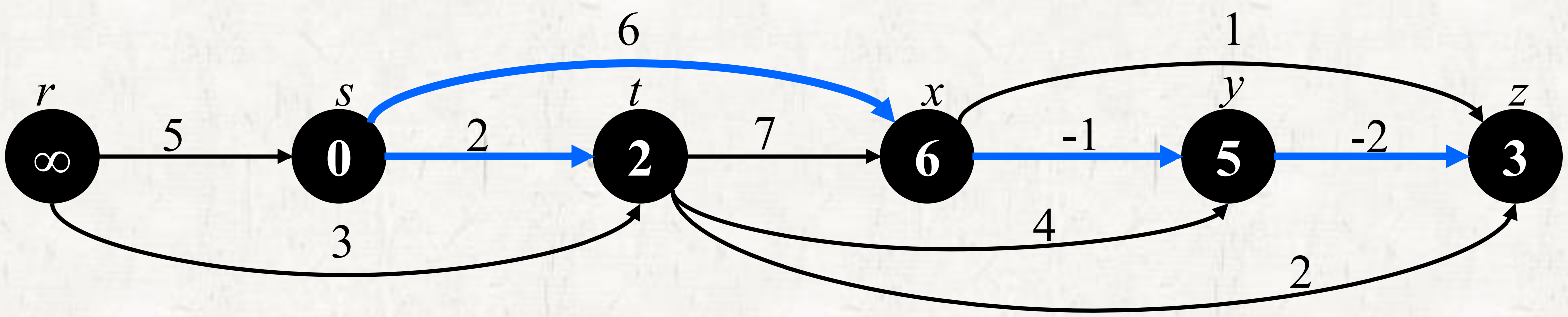


Single-source shortest paths in directed acyclic graphs





Single-source shortest paths in directed acyclic graphs



- Running time: Θ(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 “-∞” and “>” by “<”.