

CME 2001

Data Structures and Algorithms

Zerrin Işık
zerrin@cs.deu.edu.tr

Shortest Path Algorithms

Introduction

Generalization of BFS to handle weighted graphs

- Direct Graph $G = (V, E)$, edge weight $w : E \rightarrow \mathbb{R}$
- In BFS $w(e)=1$ for all $e \in E$.

Weight of path $p = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$ is

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}) = \text{sum of edge weights on path } p$$

Shortest Path

Shortest Path = Path of minimum weight

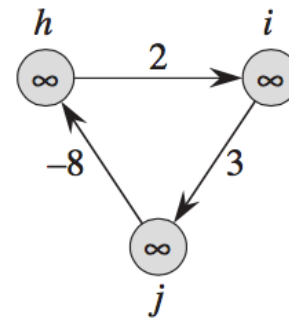
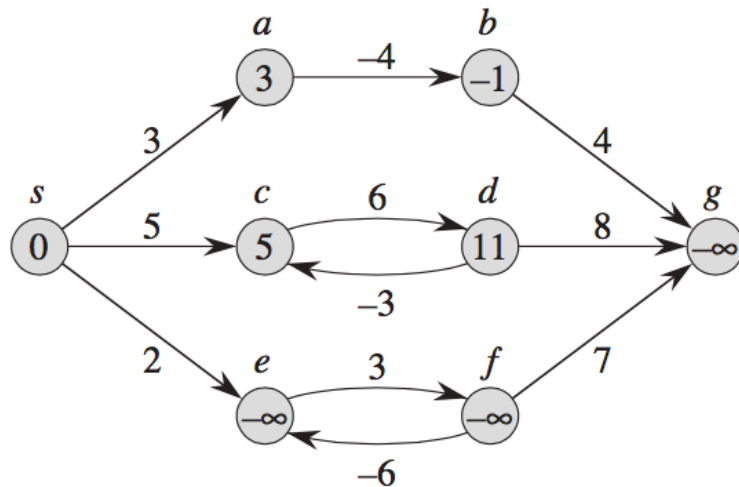
$$\delta(u,v) = \begin{cases} \min \{w(p) : u \xrightarrow{p} v\}; & \text{if there is a path from } u \text{ to } v, \\ \infty & \text{otherwise.} \end{cases}$$

Shortest-Path Variants

- **Single-source shortest-paths problem:** Find the shortest path from s to each vertex v . (e.g. BFS)
- **Single-destination shortest-paths problem:** Find a shortest path to a given *destination* vertex t from each vertex v .
- **Single-pair shortest-path problem:** Find a shortest path from u to v for given vertices u and v .
- **All-pairs shortest-paths problem:** Find a shortest path from u to v for every pair of vertices u and v .

Negative-weight edges

- If we have a negative-weight cycle, just keep going around it, and get $w(s, v) = -\infty$ for all v on the cycle.
- There is no problem, as long as no negative-weight cycles are reachable from the source.



Cycles

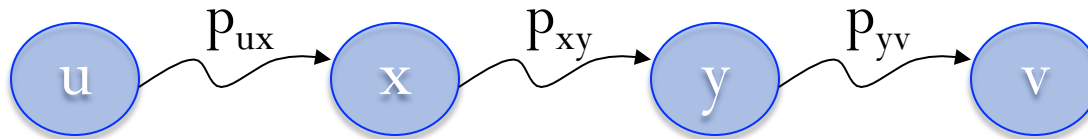
Shortest paths can't contain cycles:

- Already ruled out negative-weight cycles.
- Positive-weight \Rightarrow we can get a shorter path by omitting the cycle.
- Zero-weight: no reason to use them \Rightarrow assume that our solutions won't use them.

Optimal Substructure Property

Theorem: Any subpath of a shortest path is also a shortest path

Proof: By cut and paste.



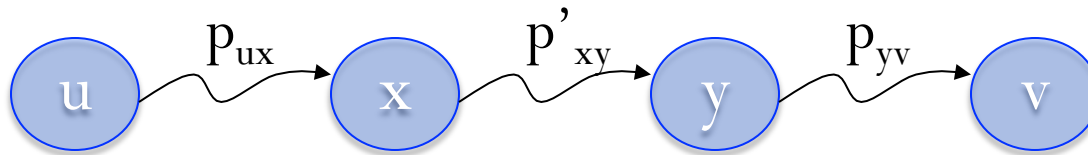
Suppose this path p is a shortest path from u to v . Then

$$\delta(u, v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv}).$$

Optimal Substructure Property

Now suppose there exists a shorter path $x \overset{p'_{xy}}{\rightsquigarrow} y$.

Then $w(p'_{xy}) < w(p_{xy})$. Construct p' :



Then

$$w(p') = w(p_{ux}) + w(p'_{xy}) + w(p_{yv})$$

$$w(p') < w(p_{ux}) + w(p_{xy}) + w(p_{yv})$$

$$w(p') = w(p)$$

Contradicts the assumption that p is a shortest path. ■

Relaxation

- All the shortest-paths algorithms start with INIT-SINGLE-SOURCE.

INIT(G, s)

for each $v \in V$ do

$v.d = \infty$

$v.\pi = \text{NIL}$

$s.d = 0$

$v.d = \delta(s, v)$

$v.\pi =$ predecessor of v on a shortest path from s .

- Can we improve the shortest-path estimate for v by going through u and taking (u, v) ?

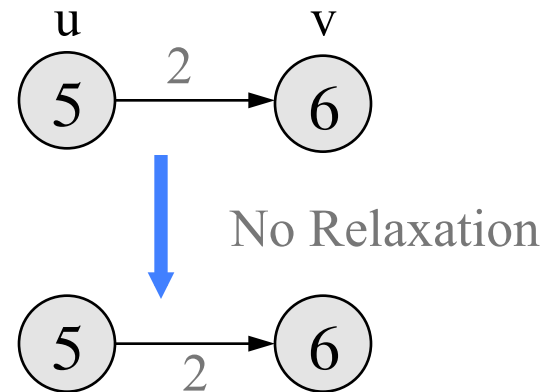
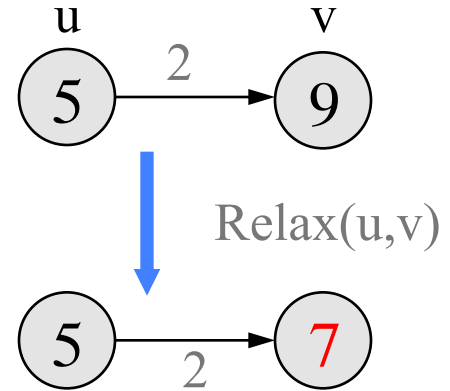
Relaxation

$RELAX(u, v)$

if $v.d > u.d + w(u, v)$

$v.d = u.d + w(u, v)$

$v.\pi = u$



Properties of Relaxation

Triangle Inequality:

For a given vertex $s \in V$ and for every edge $(u, v) \in E$,

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

Upper-bound property:

We always have $v.d \geq \delta(s, v)$ for all vertices $v \in V$.

Once $v.d$ achieves the $\delta(s, v)$ value, it never changes.

No-path property:

If there is no path from s to v , then always have $v.d = \delta(s, v) = \infty$

Properties of Relaxation

Covergence property:

If $s \rightsquigarrow u \rightarrow v$ be a **shortest path** from s to v for some $u, v \in V$ and if $u.d = \delta(s, u)$ at any time prior to **Relax**(u, v) then $v.d = \delta(s, v)$ all times afterward.

Predecessor-subgraph property:

Once $v.d = \delta(s, v)$ for all $v \in V$, the predecessor-subgraph is a shortest-paths tree rooted at s .

Single-Source Shortest Path Algorithms

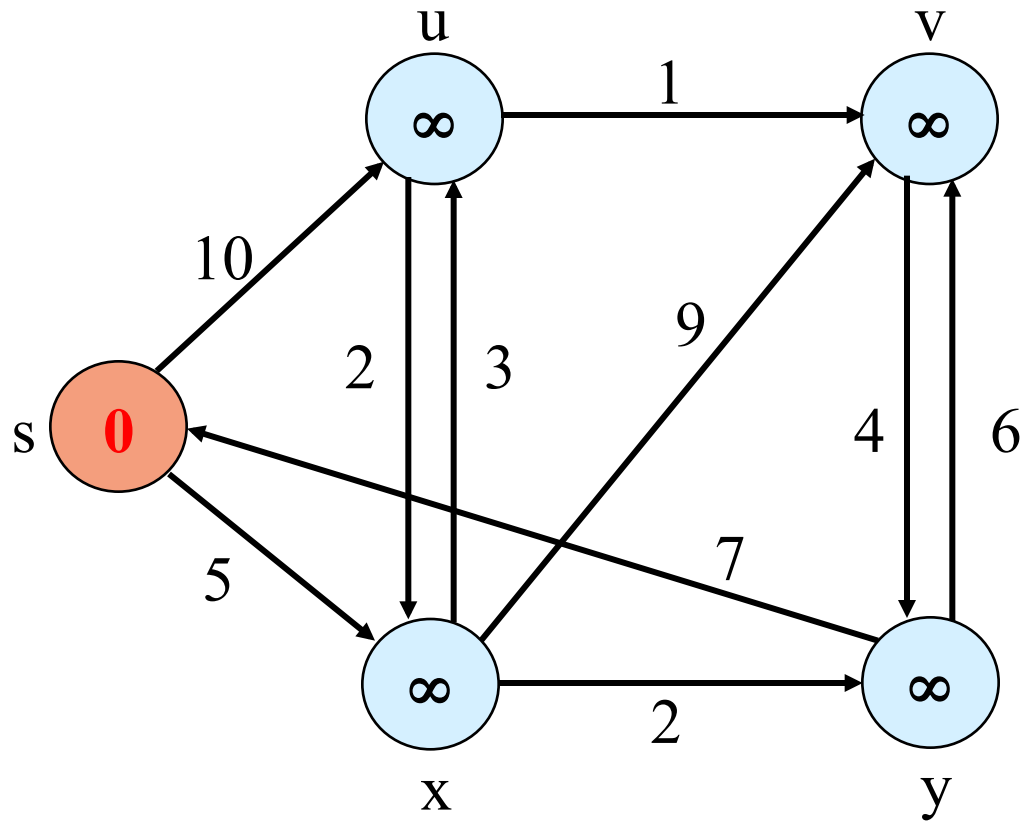
- Dijkstra's Algorithm:
 - greedy approach
 - similar to BFS
 - works for no negative-weight edges
- Bellman-Ford Algorithm:
 - edge weights can be negative
 - can detect negative-weight cycles and report them

Dijkstra's Algorithm

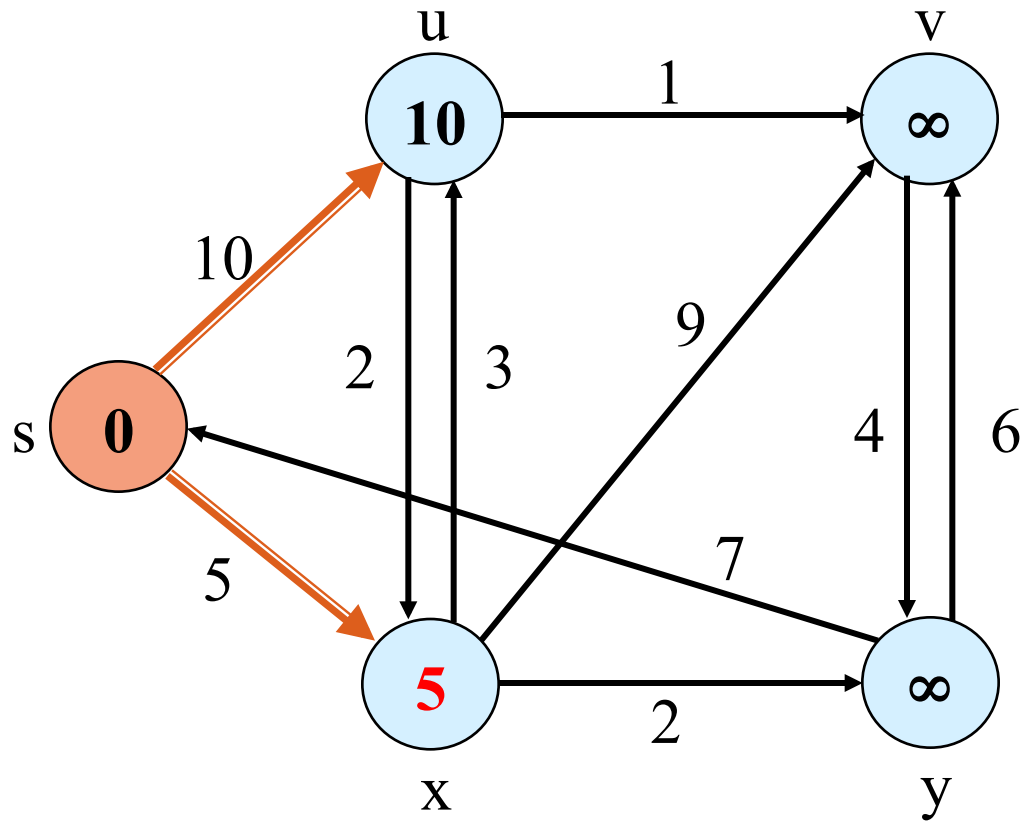
- no negative-weight edges
- weighted version of BFS
- Keys are shortest-path weights: $v.d$
- Two sets of vertices:
 - S: vertices whose final SP-path weights are found.
 - Q: priority queue = $V - S$
- Repeatedly selects u in Q with minimum SP estimate (greedy choice).

```
DIJKSTRA (G, w, s)
INIT (G, s)
S =  $\emptyset$ 
Q = G.V           //insert all vertices into Q
while Q  $\neq \emptyset$ 
    u = EXTRACT-MIN (Q)
    S = S  $\cup$  {u}
    for each v  $\in$  G.Adj[u]
        RELAX (u, v)
```

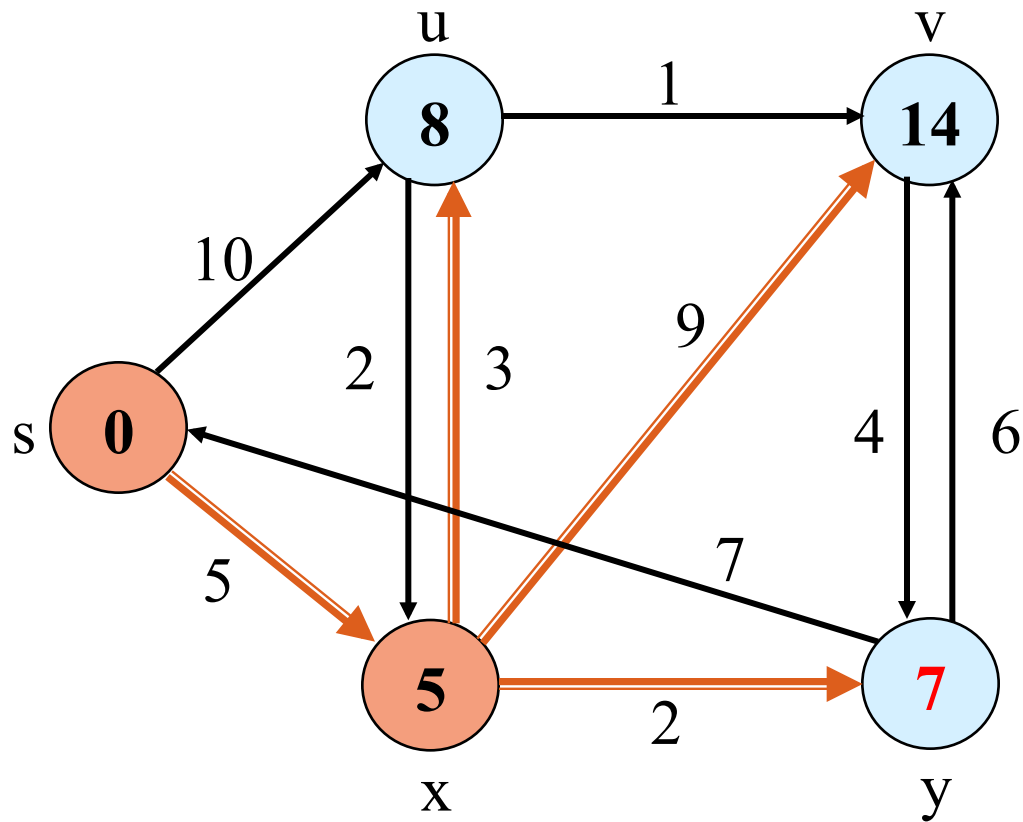
Example



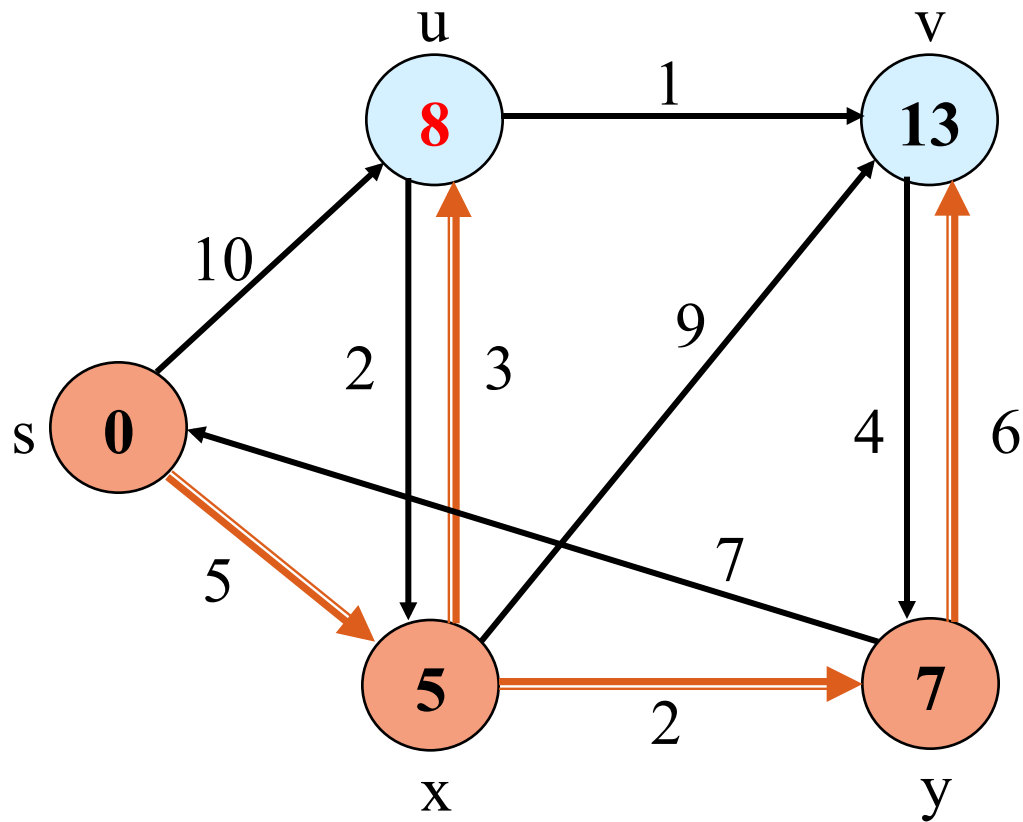
Example



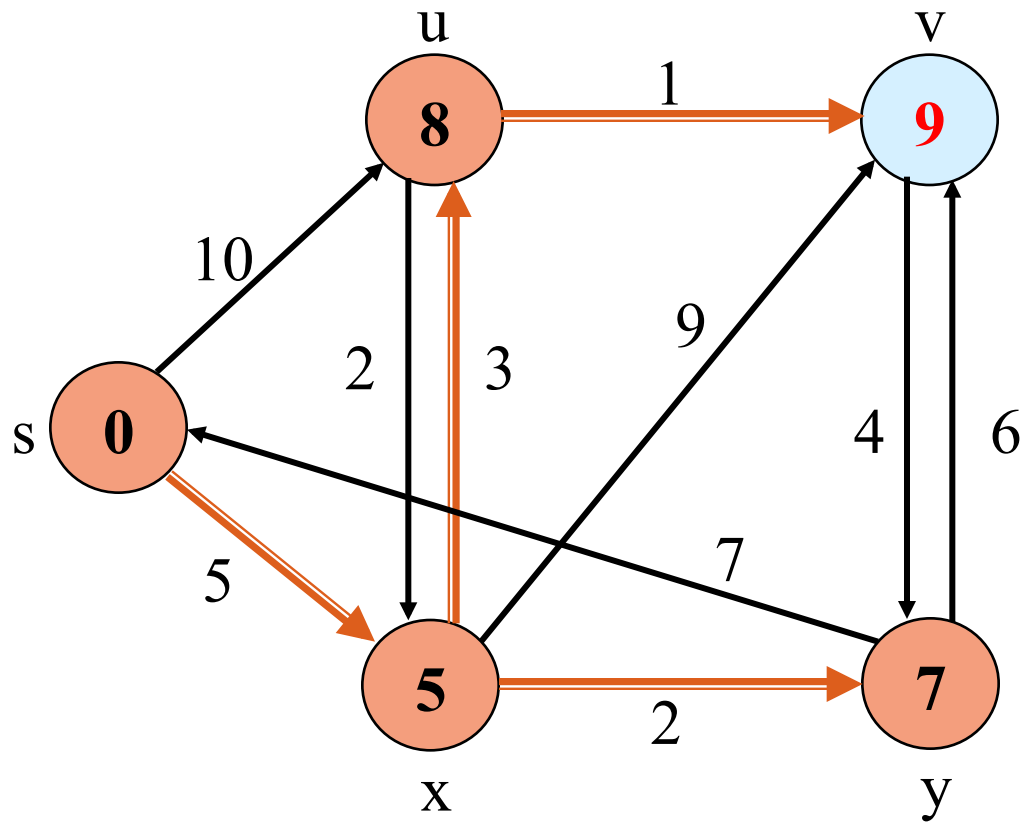
Example



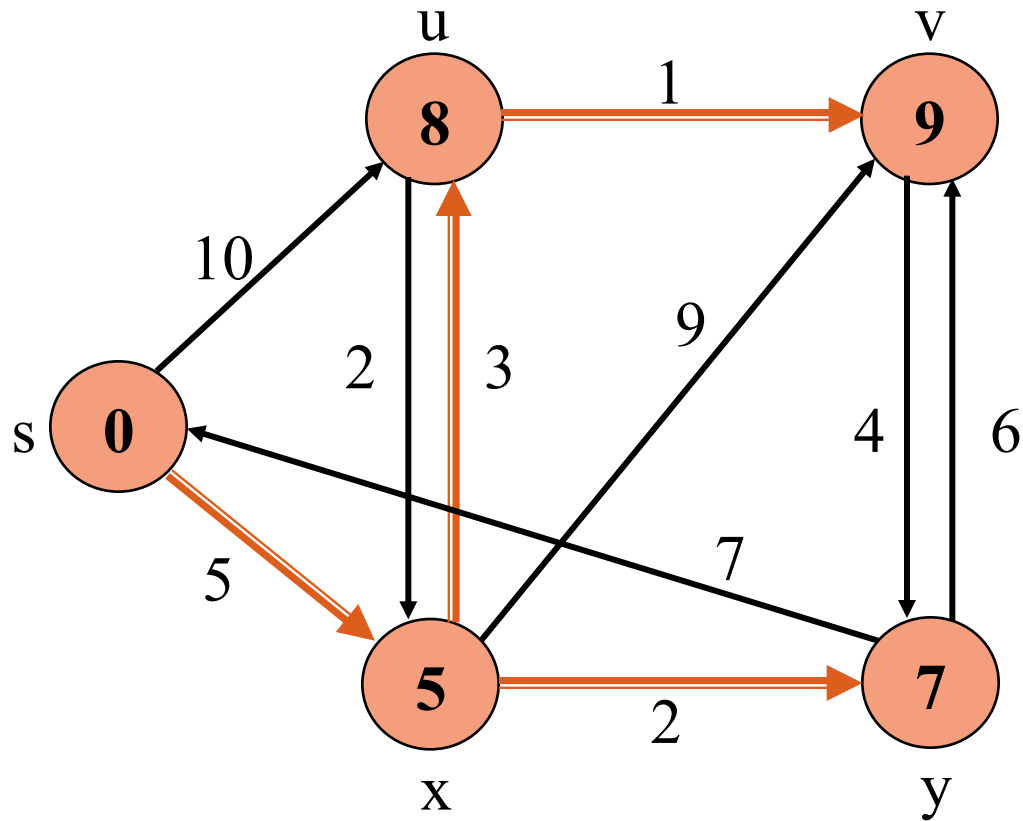
Example



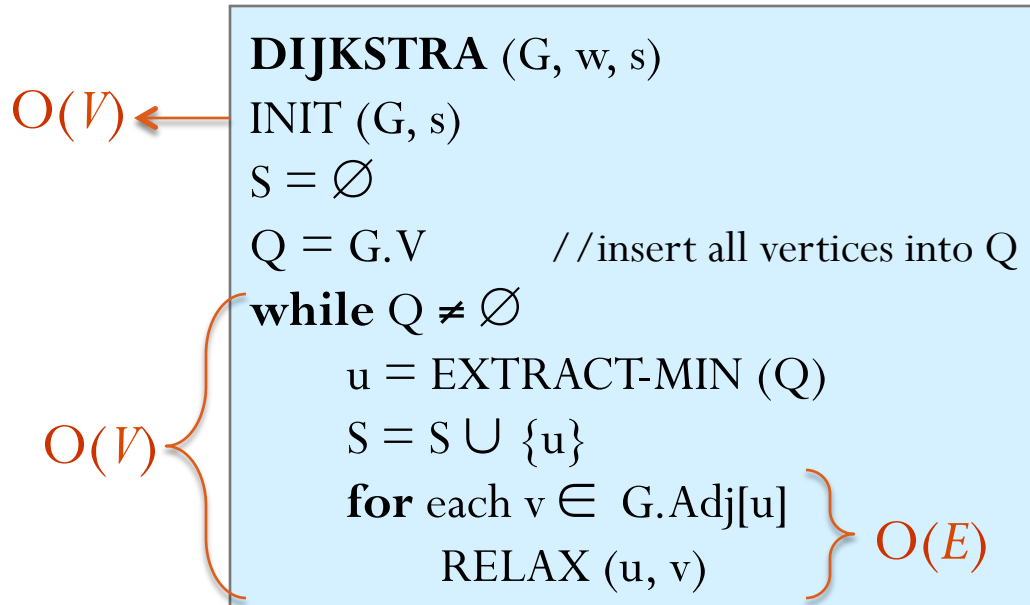
Example



Example

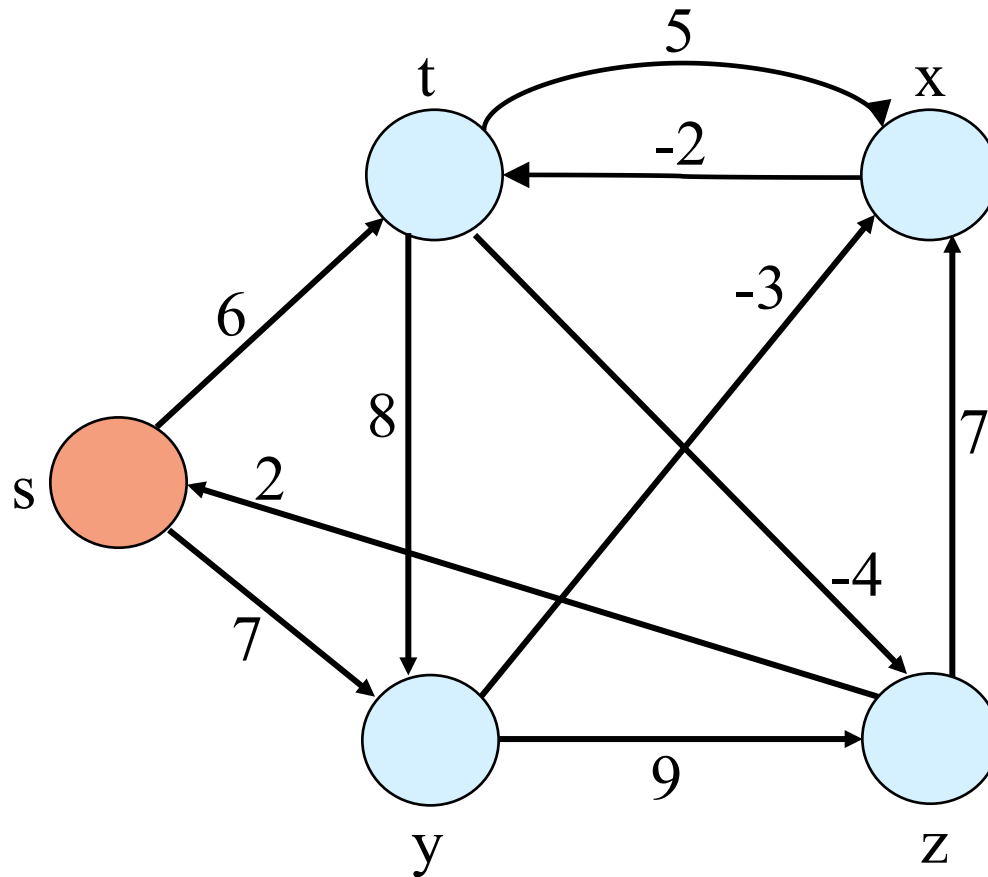


Analysis



- Running time depends on implementation of priority queue:
- Total time :
 - $O(V^2 + E) = O(V^2)$ using linear array for priority queue.
 - $O((V + E) \lg V) = O(E \lg V)$ using min-binary heap.

Negative weight graph



Bellman-Ford Algorithm

BELLMAN-FORD(G, w, s)

INIT-SINGLE-SOURCE(G, s)  Initialize $v.d = \infty$, $v.\pi = \text{NIL}$ for $v \in V$

for $i = 1$ to $|G.V| - 1$

for each edge $(u, v) \in G.E$

RELAX(u, v, w)

for each edge $(u, v) \in G.E$

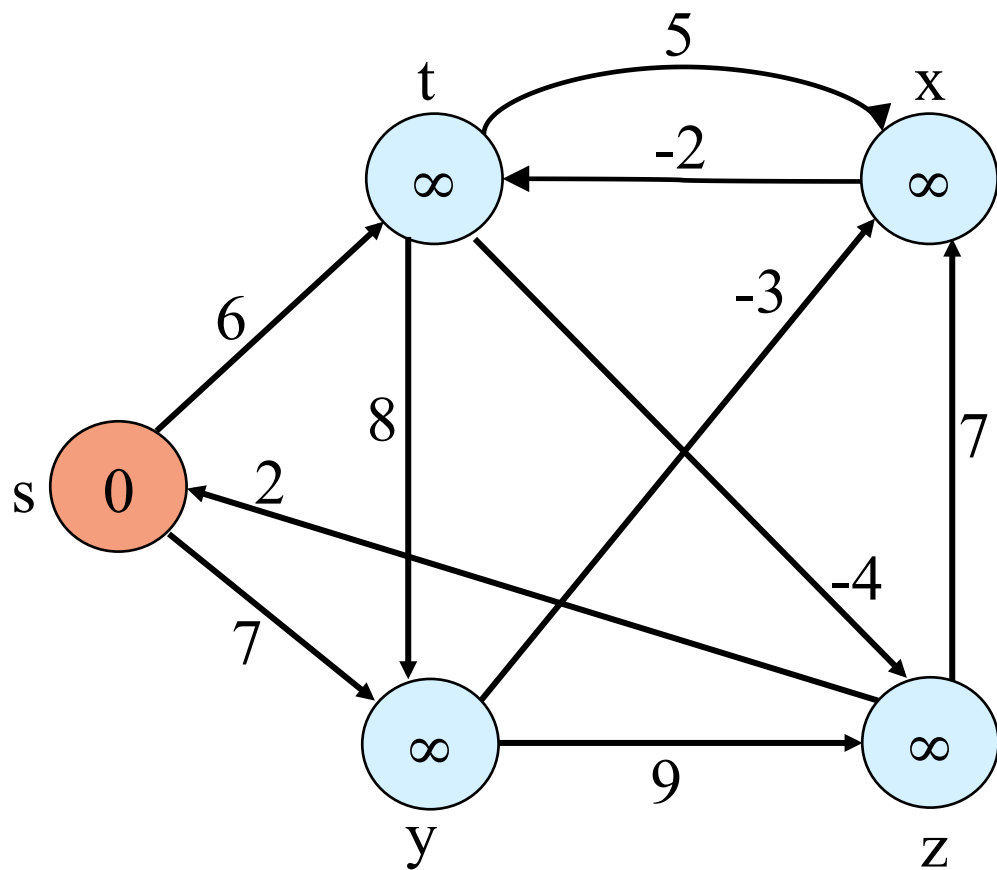
if $v.d > u.d + w(u, v)$

return FALSE

return TRUE

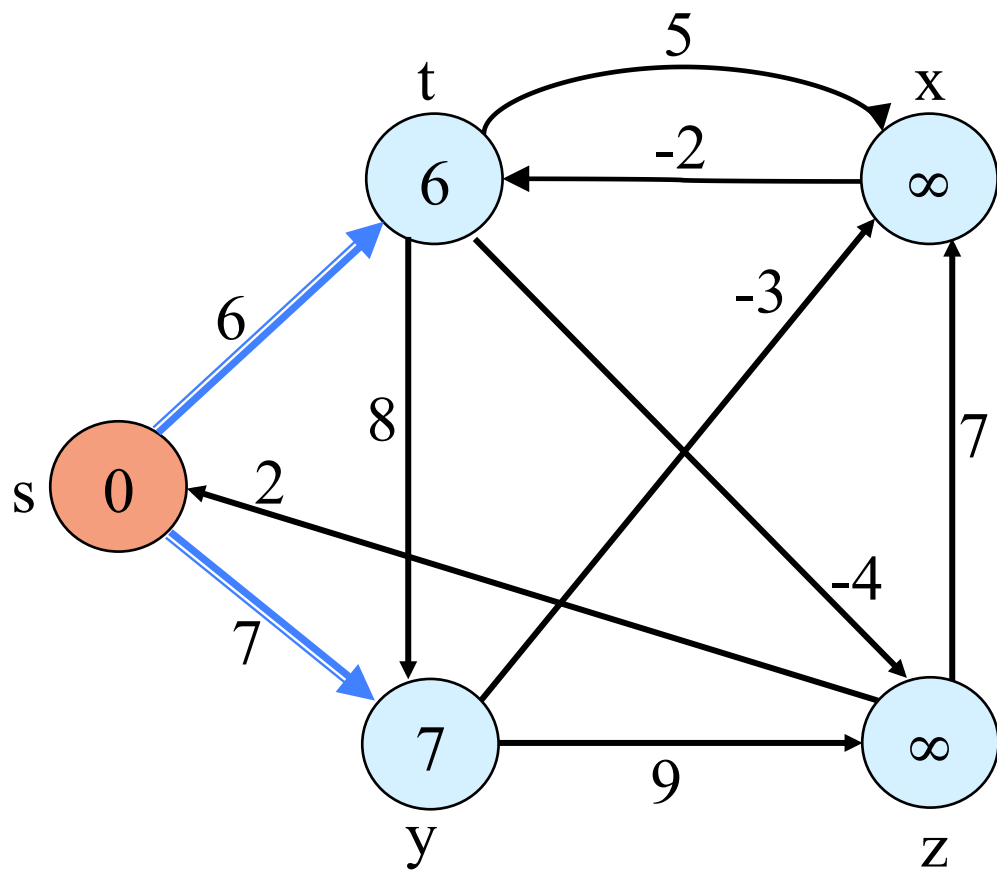
The algorithm returns TRUE if and only if the graph contains no negative-weight cycles that are reachable from the source s .

Example



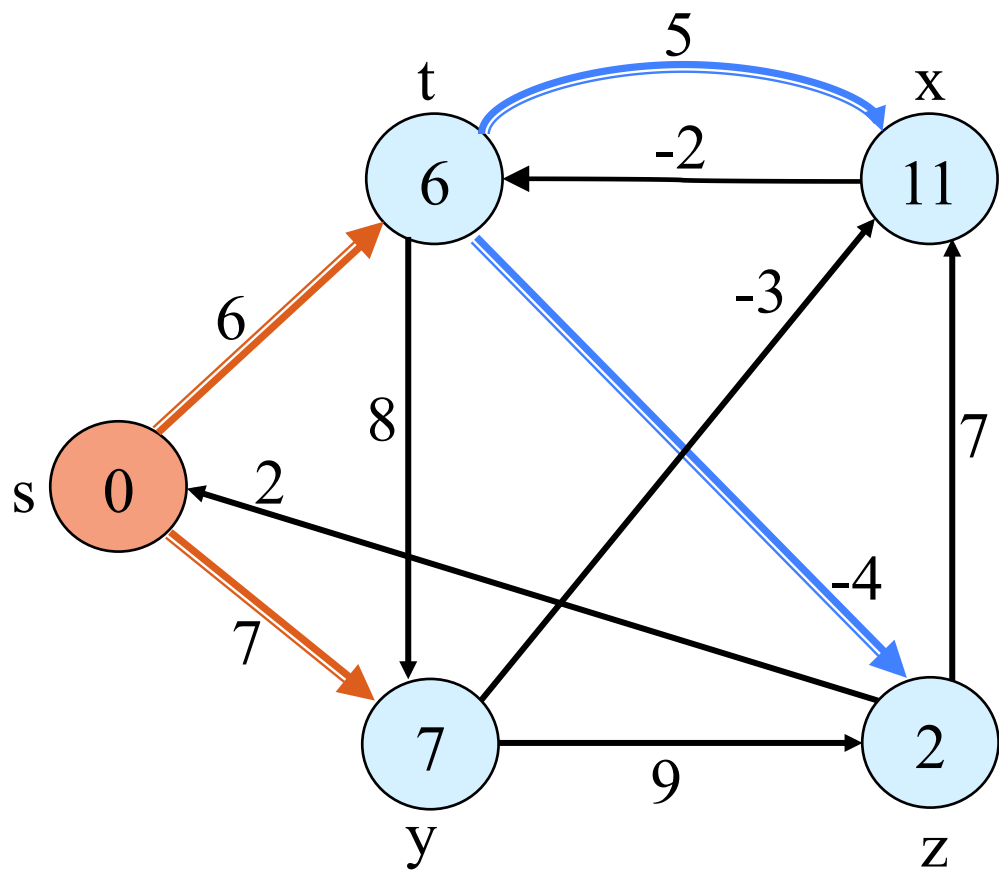
i	s	t	y	x	z
0	0	∞	∞	∞	∞

Example



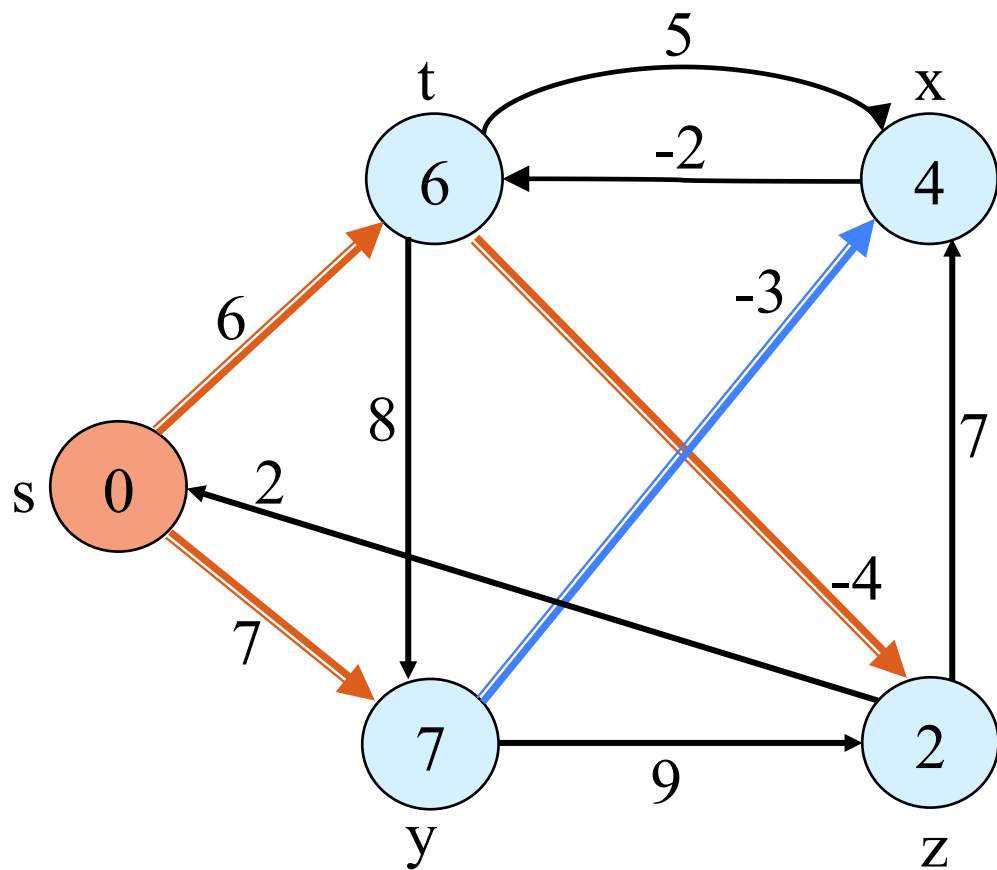
i	s	t	y	x	z
1	0	6	7	∞	∞

Example



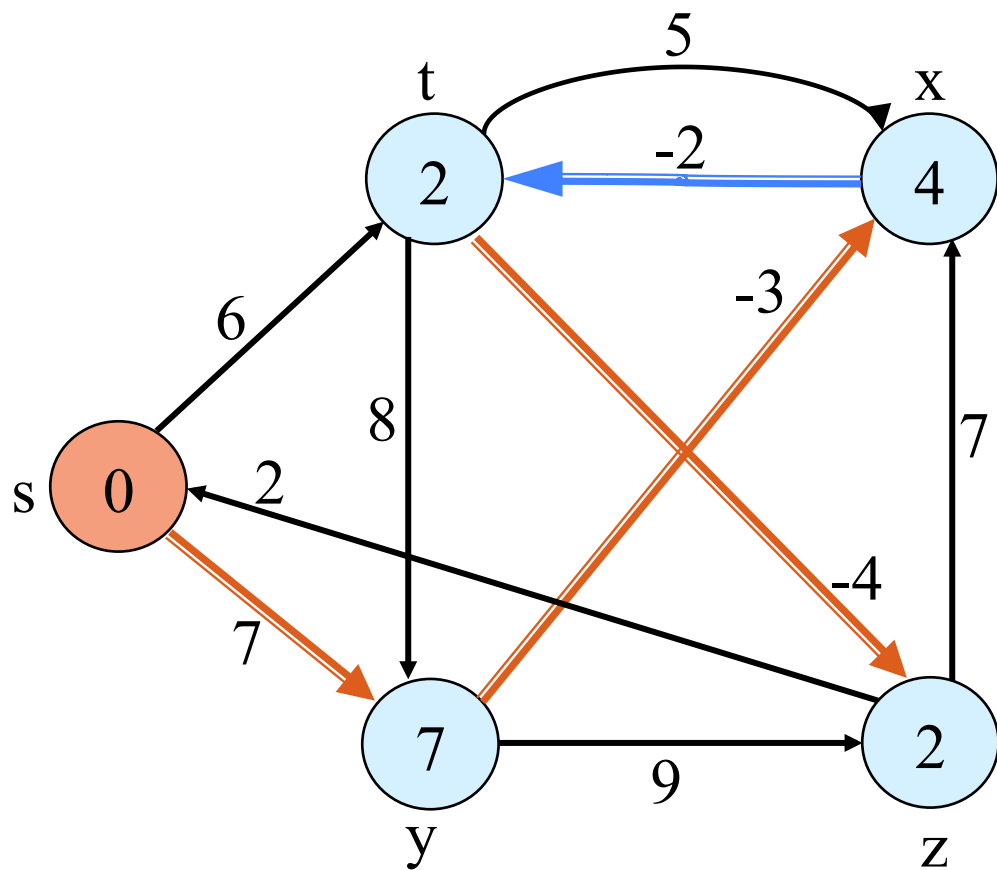
i	s	t	y	x	z
1	0	6	7	11	2

Example



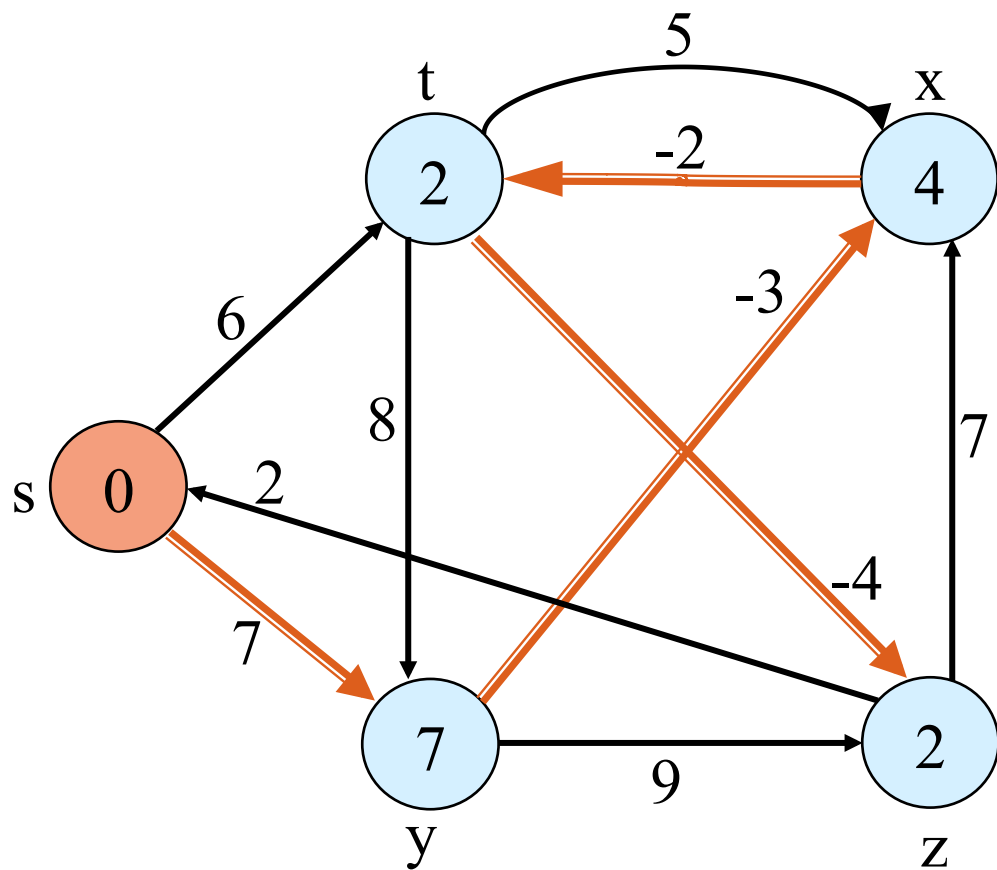
i	s	t	y	x	z
1	0	6	7	4	2

Example



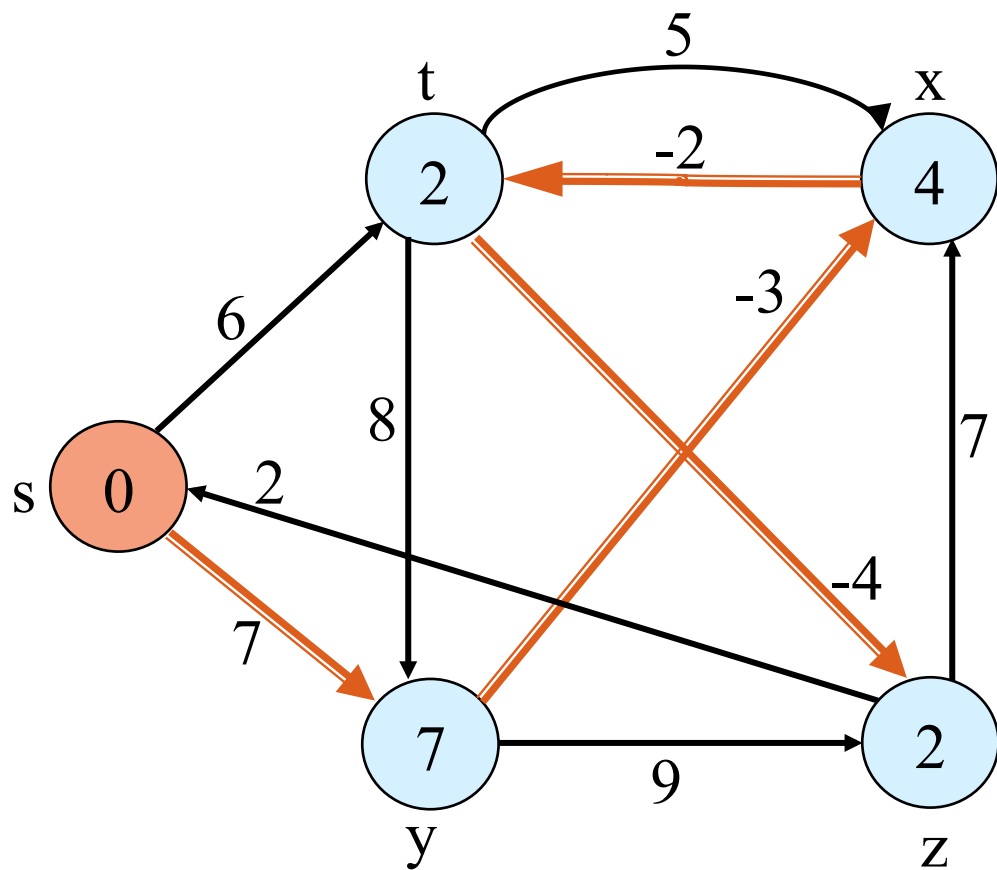
i	s	t	y	x	z
1	0	2	7	4	2

Example



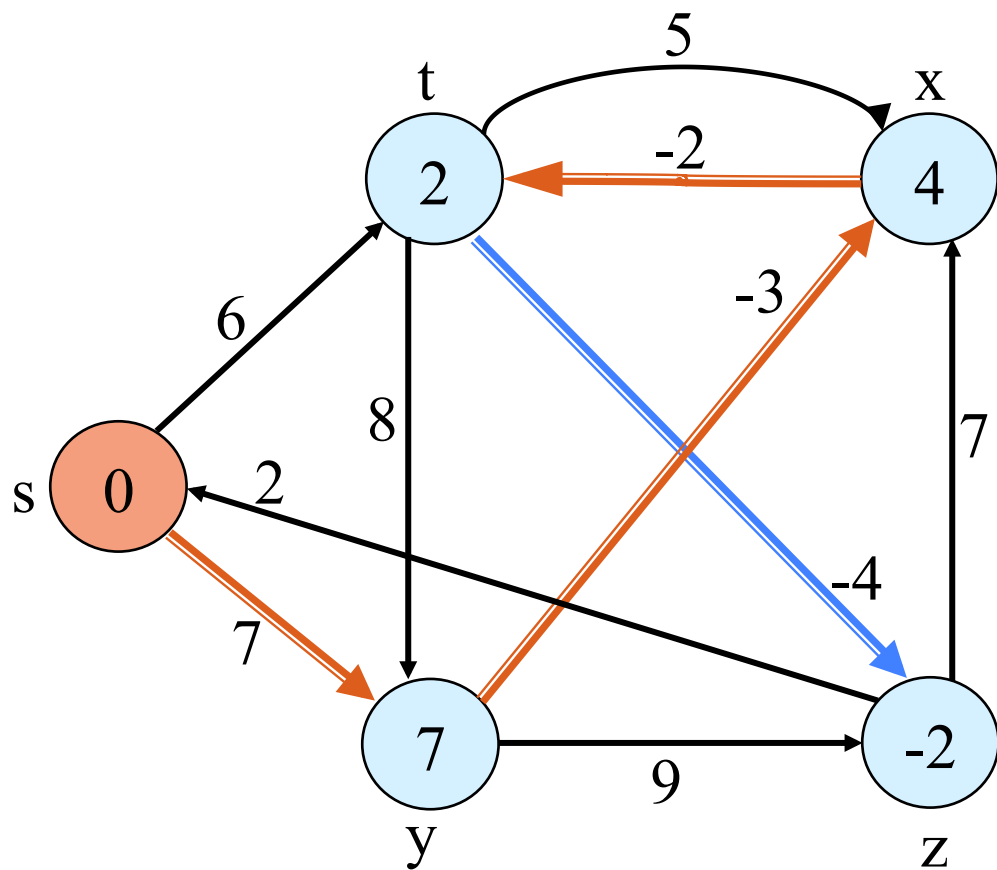
i	s	t	y	x	z
1	0	2	7	4	2

Example



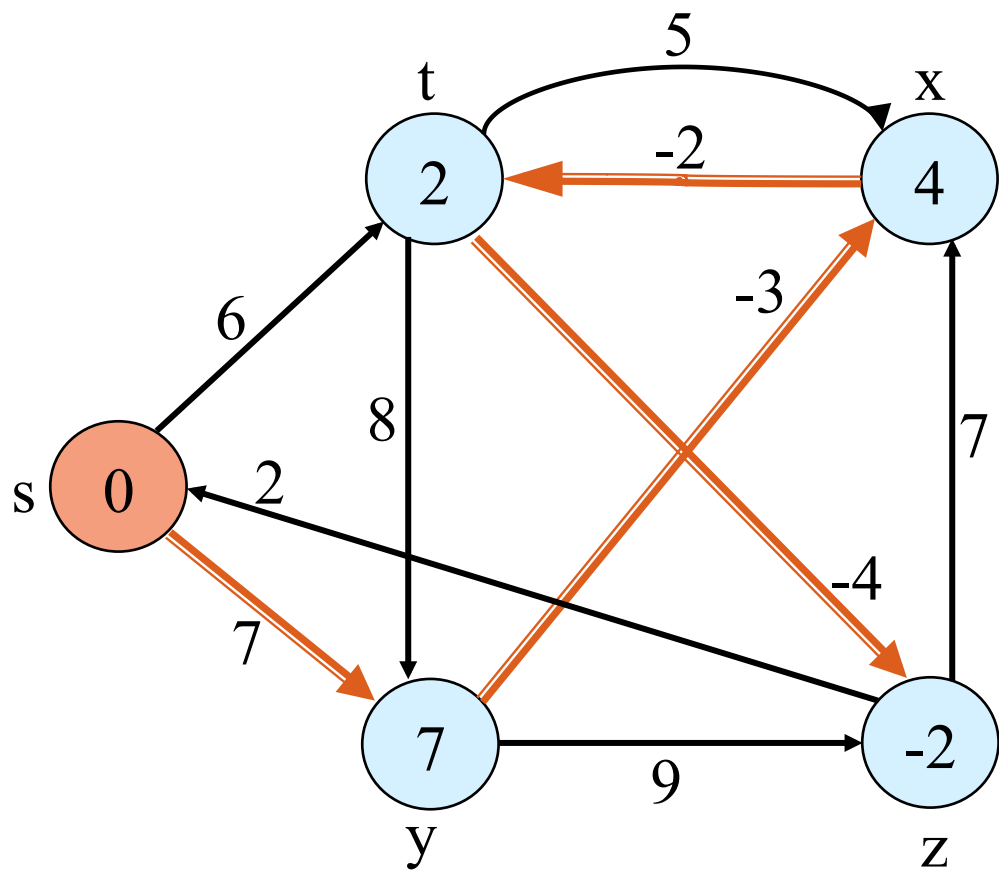
i	s	t	y	x	z
2	0	2	7	4	2

Example



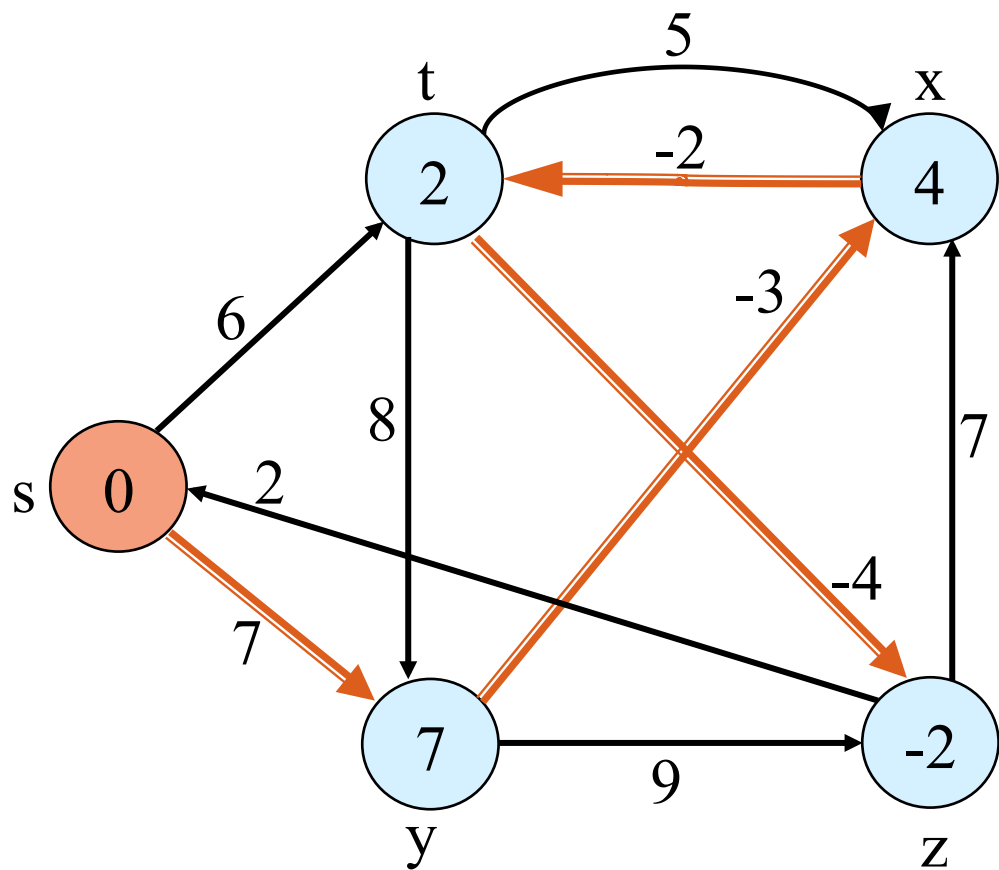
i	s	t	y	x	z
2	0	2	7	4	-2

Example



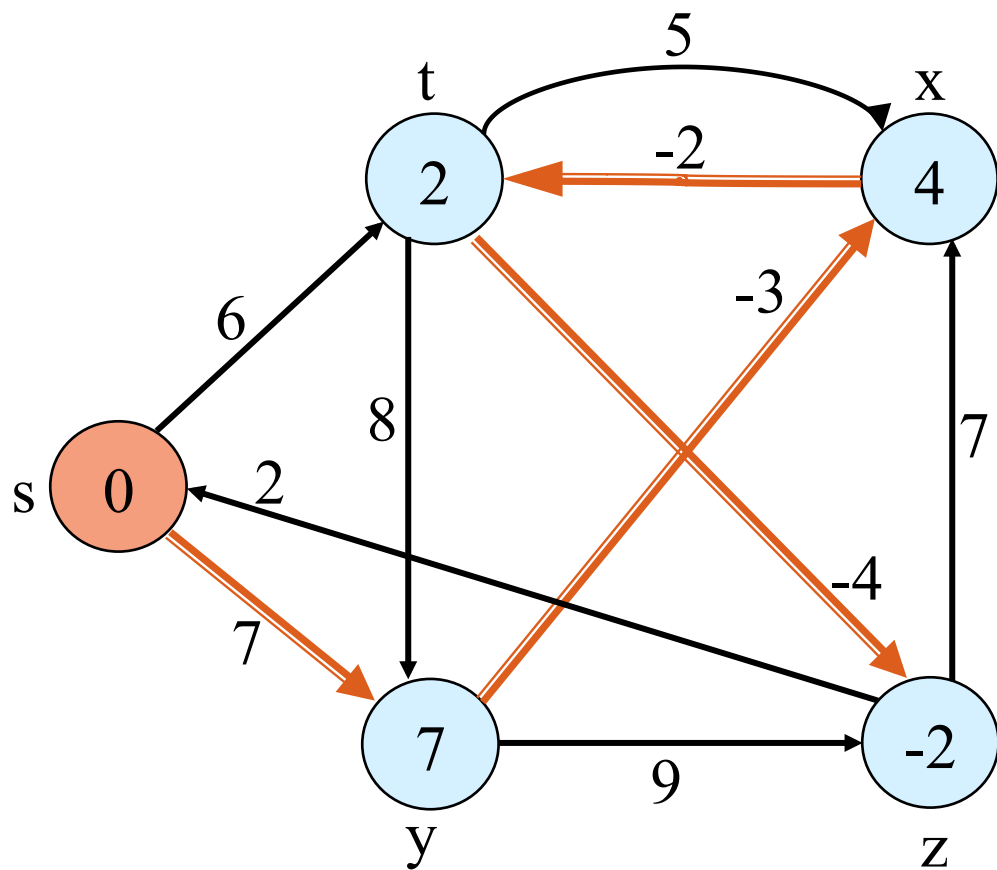
i	s	t	y	x	z
2	0	2	7	4	-2

Example



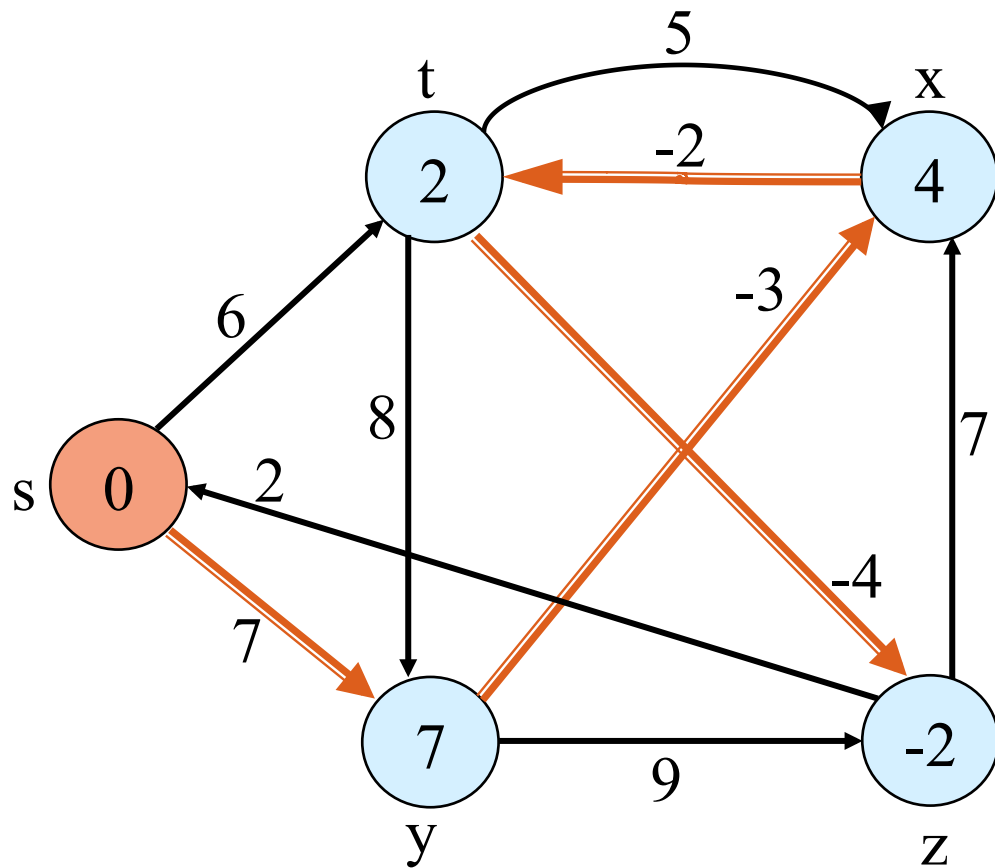
i	s	t	y	x	z
2	0	2	7	4	-2

Example



i	s	t	y	x	z
2	0	2	7	4	-2

Example



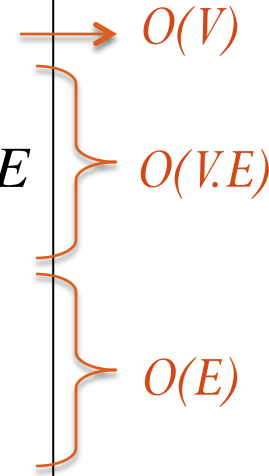
for other iterations ($i=3,4$),
the path lengths do not change.

i	s	t	y	x	z
4	0	2	7	4	-2

shortest path lengths
from **s** to other vertices

Analysis

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



$O(V)$

$O(V.E)$

$O(E)$

- Total Run-time : $O(V + VE + E) = O(VE)$
- Bellman-Ford Algorithm is not as *efficient* as Dijkstra's Algorithm, but it can work on negative weighted edges!

Final Exam Topics

Insertion and Merge Sorts, Complexity Analysis

Stack, Queue, Linked List

Hash Table, Resolving Hashing Problems

Heap Sort Algorithm

Quick Sort Algorithm

Linear Sorting Algorithms

Binary Search Trees

AVL-Trees

B-Trees

Graph Traversal Algorithms (BFS, DFS)

Minimum Spanning Tree Algorithm

Single Source Shortest Path Algorithms

Can use a cheat sheet 😊

- **2 sides** of **one A4** paper
with your own handwriting!