

an SERIES	School of Computer Engineering, KIIT University, Bhubaneswar	CS-2012 : Design & Analysis of Algorithms	DAA	2022 AUTUMN
Lecture No – 34 to 37				
SHORTEST PATH PROBLEM				
Branch/Section: CSE-3,CSE-10 & IT-6, 5th Sem.			Faculty: Prof. Anil Kumar Swain	

Lecture Summary

1. Introduction to Shortest Path Problem
2. The Bellman-Ford Algorithm
3. The Dijkstra's algorithm
4. The Floyd-Warshall algorithm

1 Introduction

1.1 Shortest Path Problem

- In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) such that the sum of the weights of its constituent edges is minimized.
- An example is finding the quickest way to get from one location to another on a road map; in this case, the vertices represent locations and the edges represent segments of road and are weighted by the time needed to travel that segment.
- Formally, In a shortest-paths problem, we are given a weighted, directed graph $G = (V, E)$, with weight function $w : E \rightarrow \mathbb{R}$ mapping edges to real-valued-weights. The weight of path $p = v_0, v_1, \dots, v_k$ is the sum of the weights of its constituent edges:

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

We define the shortest-path weight from u to v by

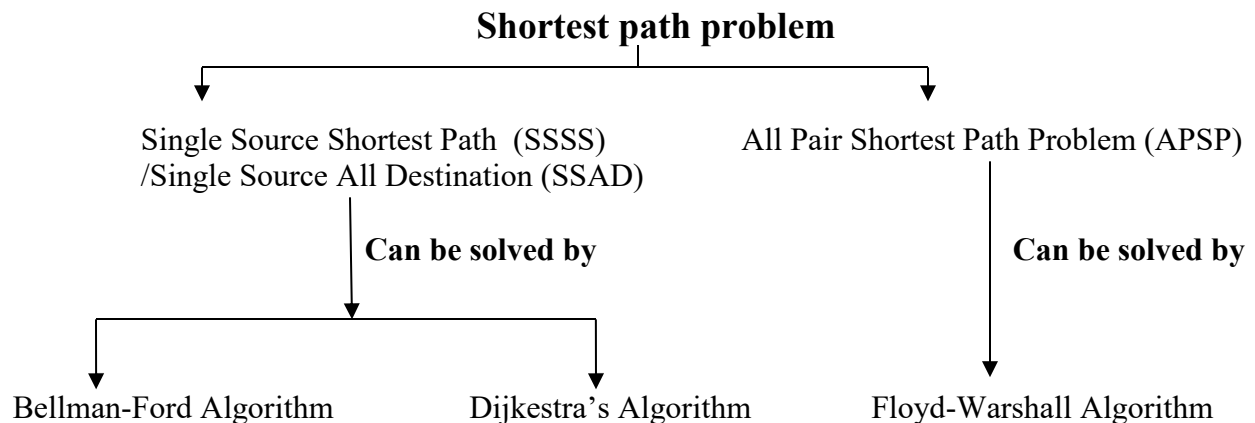
$$\delta(u, v) = \begin{cases} \min\{w(p) : u \stackrel{p}{\rightsquigarrow} v\} & \text{if there is a path from } u \text{ to } v , \\ \infty & \text{otherwise .} \end{cases}$$

A shortest path from vertex u to vertex v is then defined as any path p with weight $w(p) = \delta(u, v)$.

- Edge weights can be interpreted as metrics other than distances. They are often used to represent time, cost, penalties, loss, or any other quantity that accumulates linearly along a path and that one wishes to minimize.

1.2 Single Source Shortest Path(SSSS) Problem

- The problem is also sometimes called Single source all destination (SSAD) problem.
- Many other problems can be solved by the algorithm for the single-source problem, including the following variants.
 - Single-destination shortest-paths problem:** Find a shortest path to a given destination vertex t from each vertex v . By reversing the direction of each edge in the graph, we can reduce this problem to a single-source problem.
 - Single-pair shortest-path problem:** Find a shortest path from u to v for given vertices u and v . If we solve the single-source problem with source vertex u , we solve this problem also.
 - All-pairs shortest-paths (APSP) problem:** Find a shortest path from u to v for every pair of vertices u and v . Although this problem can be solved by running a single-source algorithm once from each vertex, it can usually be solved faster.



1.3 Initialize Single Source Shortest Path

- Before calling Single source shortest path algorithm (Bellman_Ford or Dijkstra), we initialize $d[v]$ the weights from source to all vertices as infinity except the source vertex as zero.
- We do this because before considering any edges, the weights from source to other vertex is infinity as vertices are disjointed. But source to source weight is zero as this is the starting point, it is obvious.

```

INITIALIZE-SINGLE-SOURCE(G, s)
{
    for each vertex v ∈ V[G]
    {
        d[v] ← ∞
        π[v] ← NIL
    }
    d[s] ← 0
}
  
```

Where $\pi[v]$ is the parent or v 's predecessor field and $d[v]$ is the shortest path estimate that is the total minimum weight or distance or cost from source vertex s to v .

1.4 Relaxation Technique

- The process of relaxing an edge (u, v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating $d[v]$ and $\pi[v]$.
- A relaxation step may decrease the value of the shortest-path estimate $d[v]$ and update v 's predecessor field $\pi[v]$.
- The following code performs a relaxation step on edge (u, v) .

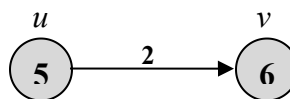
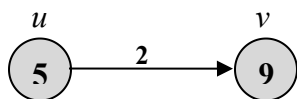
```

RELAX( $u, v, w$ )
{
  If  $d[u] + w(u, v) < d[v]$ 
  {
     $d[v] \leftarrow d[u] + w[u, v]$ 
     $\pi[v] \leftarrow u$ 
  }
}

```

- Where $d[v]$ is the minimum weight of a shortest path from source s to v .
- We call $d[v]$ a shortest-path estimate.

- **Example :** Show the relaxation technique on the given figure (a) and (b)



Answer:

Figure - a

In this case

$$d[u] = 5 \quad d[v] = 9 \quad w(u, v) = 2$$

Apply RELAX(u, v, w)

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

That is $(5+2) < 9 \Rightarrow 7 < 9$ (true) so

Update $d[v]$, $d[v] = 5+2=7$

This means after considering the edge (u, v) we found $d[v]$ is no more minimum. $d[v]$ is minimum through u , that is the edge (u, v) may decrease the $d[v]$ value.

Figure – b

In this case

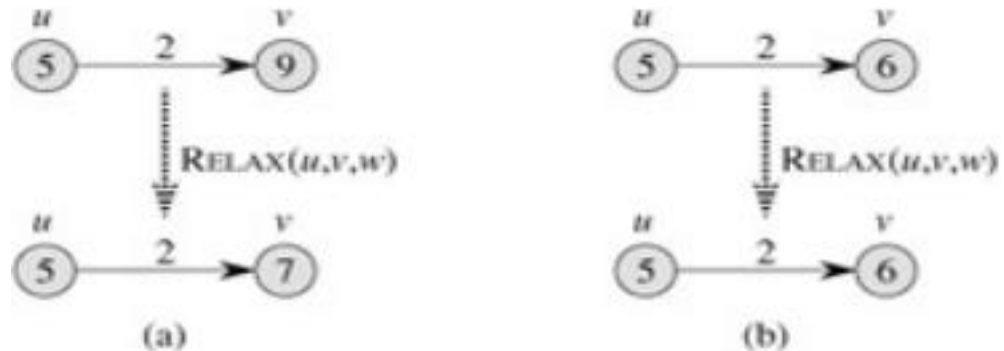
$$d[u] = 5 \quad d[v] = 6 \quad w(u, v) = 2$$

Apply RELAX(u, v, w)

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

$(5+2) < 6 \Rightarrow 7 < 6$ (false) so no need

to update. This means the vertex v has already been assigned a minimum value. (s to v is minimum). The edge (u, v) may not decrease the $d[v]$ value.



2 The Bellman-Ford Algorithm

2.1 Introduction

- The Bellman-Ford algorithm solves the single-source shortest-paths problem in the general case in which edge weights may be negative.
- Given a weighted, directed graph $G = (V, E)$ with source s and weight function $w : E \rightarrow \mathbb{R}$, the Bellman-Ford algorithm returns a Boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source. **If there is such a cycle, the algorithm indicates that no solution exists.**
- If there is no such cycle, the algorithm produces the shortest paths and their weights.
- The algorithm uses relaxation, progressively decreasing an estimate $d[v]$ on the weight of a shortest path from the source s to each vertex $v \in V$ until it achieves the actual shortest-path weight $\delta(s, v)$.
- The algorithm returns TRUE if and only if the graph contains no negative weight cycles that are reachable from the source, Otherwise returns FALSE. False indicates there is a negative-weight cycle that is reachable from the source
- Bellman-Ford Algorithm

```

BELLMAN-FORD( $G, w, s$ )
{
    //Initialize Step
    INITIALIZE-SINGLE-SOURCE( $G, s$ )
    //Relax Step
    for  $i \leftarrow 1$  to  $|V[G]| - 1$ 
    {
        for each edge  $(u, v) \in E[G]$ 
            RELAX( $u, v, w$ )
    }
}

```

```
//Testing Step for -ve weight cycle
for each edge (u, v) ∈ E[G]
{
    if d[v] > d[u] + w(u, v)
        then return FALSE
}
return TRUE
}
```

- The Bellman-Ford runs in time $O(EV)$

2.2 Example (The Bellman-Ford Algorithm)

Run the Bellman-Ford algorithm on the directed graph of Figure-1, using vertex s as the source. In each pass, relax edges in the order (t, x) , (t, y) , (t, z) , (x, t) , (y, x) , (y, z) , (z, x) , (z, s) , (s, t) , (s, y) . and show the d and π values after each pass. Or

You may be asked to find out the minimum distance from vertex s to x or a given vertex s to any other vertex on the the directed graph of Figure-1.

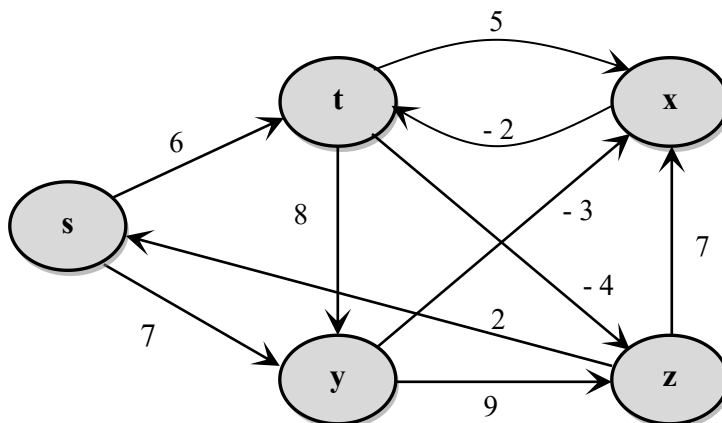
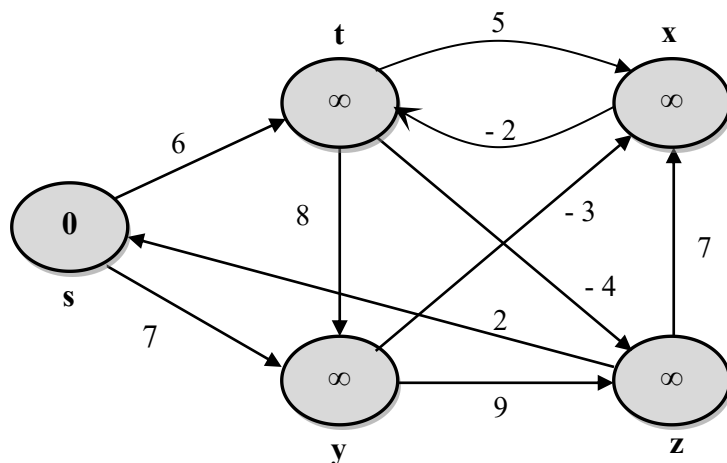


Figure-1 : Bellman-Ford

Solution:

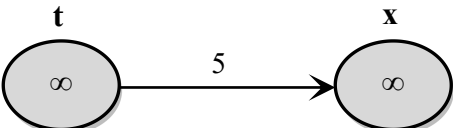
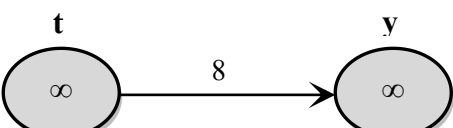
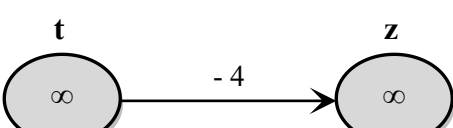
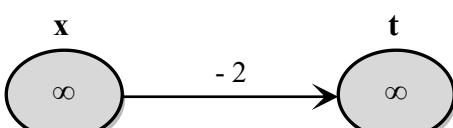
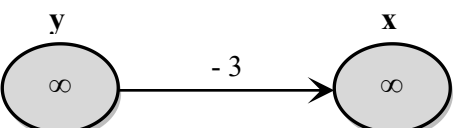
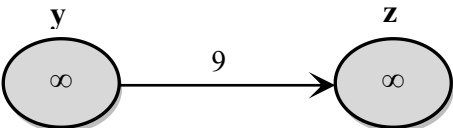
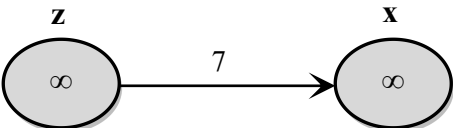
Initialize Step: Initialize single source that is set all vertices have infinite cost except the source vertex s , which has zero cost.



(Initialize step figure)

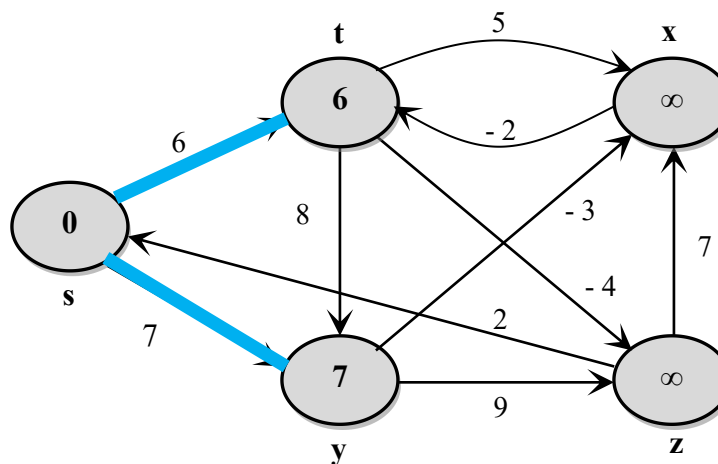
Relax Step (This step will contain $(n-1)$ passes for a given graph with n vertices, in this case it is $n=5$, so total pass is 4)

For Pass-1 : Refer initialize-step figure and the current d value of the vertices after relaxation (if any changes). Relax all edges in the order (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y).

Steps	Relax edges	Result (After Relaxation)	Remarks
1		No change	
2		No change	
3		No change	
4		No change	
5		No change	
6		No change	
7		No change	
8			

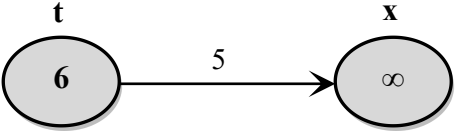
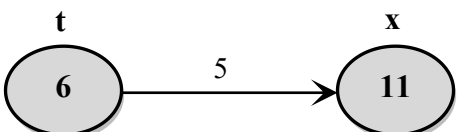
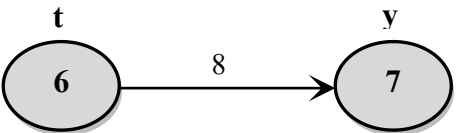
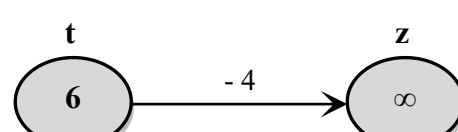
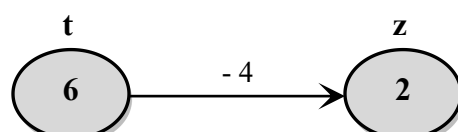
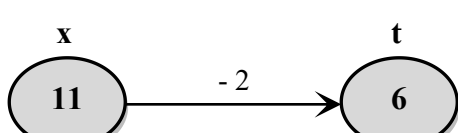
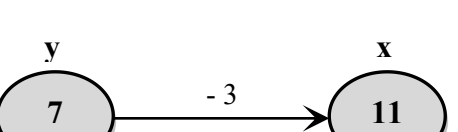
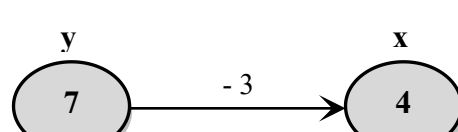
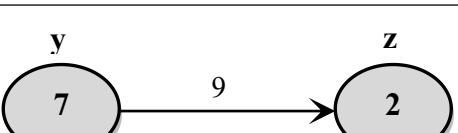
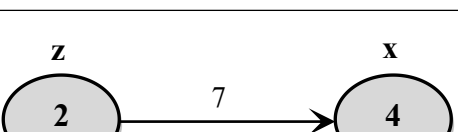
		No change	
9			Update d value of t (d[t]) => replace ∞ by 6 on initialize-step figure. Shade the edge (s,t)
10			Update d value of y (d[y]) => replace ∞ by 7 on initialize-step figure. Shade the edge (s,y)

Now the initialize-step figure becomes



(Pass-1 figure)

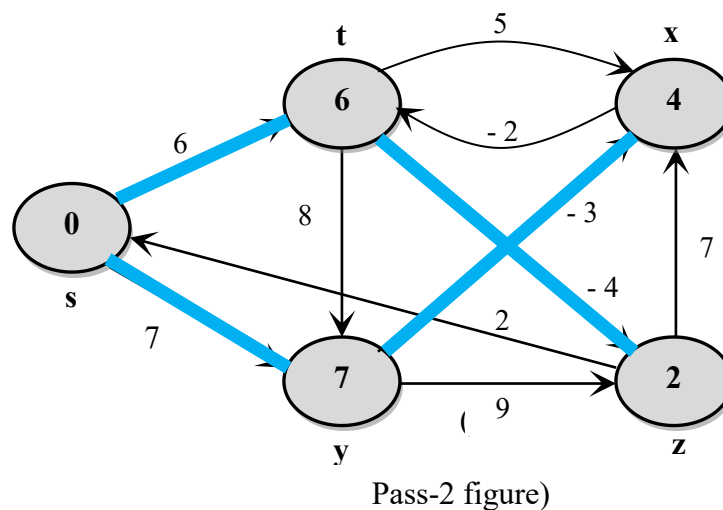
For Pass-2 : Refer pass-1 figure and the current d value of the vertices after relaxation (if any changes). Relax all edges in the order (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y).

Steps	Relax edges	Result (After Relaxation)	Remarks
1			
2		No change	
3			
4		No change	Here d value of x is not ∞ , but 11 as it is updated in step-1.
5			
6		No change	Here d value of z is not ∞ as mentioned in pass-1 fig., but 11 as it is updated in step-2.
7		No change	Here use d value of x as 4 as it is 2 nd time updated in step-5
8			

		No change	
9		No change	
10		No change	

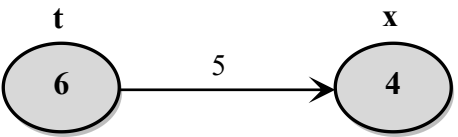
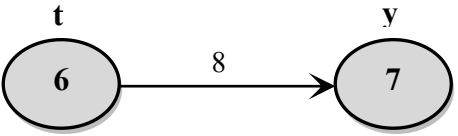
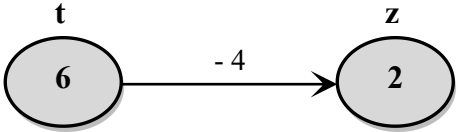
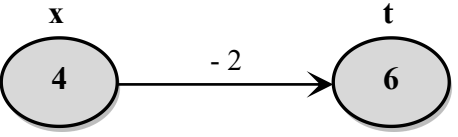
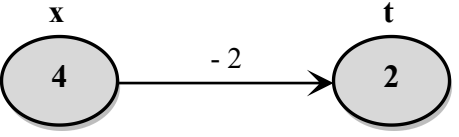
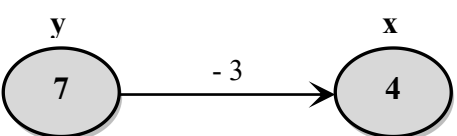
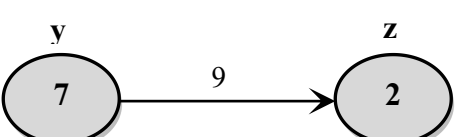

As the d value of vertices x and z are updated, we will update it in pass-1 figure. Also we will shade the edges (y,x) and (t,z), not (t, x) as in step-1 x's processor (parent) is t=>edge (t,x). but in step-5 x's processor is changed to y=>edge (y,x). So no more exists the edge (t,x).

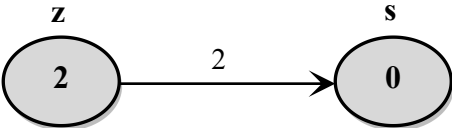
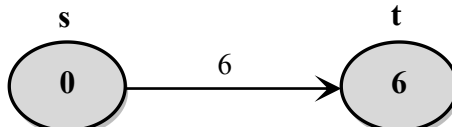
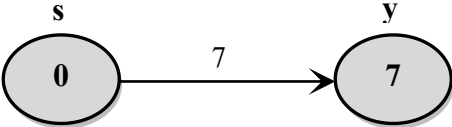
Now the pass-1 figure becomes



For Pass-3 : Refer pass-2 figure and the current d value of the vertices after relaxation (if any changes).

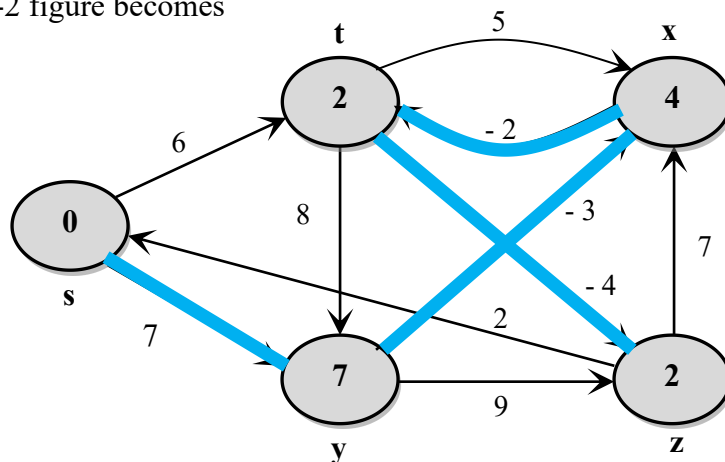
Relax all edges in the order (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y).

St ep s	Relax edges	Result (After Relaxation)	Remarks
1		No change	
2		No change	
3		No change	
4			T's predecessor was s in pass-2 fig. now t's predecessor is x. so shaded edge (s,t) will be removed.
5		No change	
6		No change	
7		No change	
8			

		No change	
9		No change	
10		No change	

Here only change is on the d value of x ($d[x]$) of edge (x, t). shade it. But remove the shade from the edge (s,t). Notice that on vertex t, there is a shaded edge (s, t), now we found another edge (x, t) to be shaded. But no two edges can be shaded incident on a vertex. As edge (x, t) is discovered recently, shade it, remove shade from edge (s, t).

Now the pass-2 figure becomes



(Pass-3 figure)

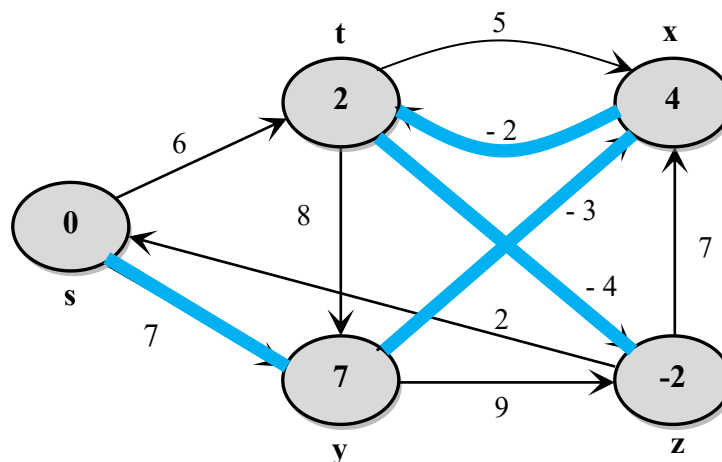
For Pass-4 : Refer pass-3 figure and the current d value of the vertices after relaxation (if any changes). Relax all edges in the order (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y).

Steps	Relax edges	Result (After Relaxation)	Remarks
1		No change	
2		No change	
3			
4		No change	
5		No change	
6		No change	
7		No change	
8			

		No change	
9		No change	
10		No change	

Here only change is on edge (t, z). Update step-3 in pas-3 figure.

Now the pass-3 figure becomes



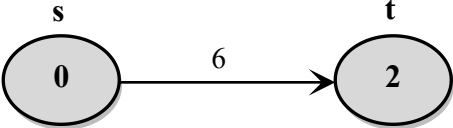
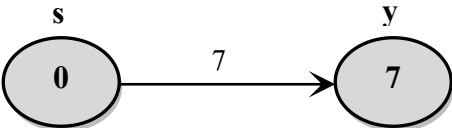
(Pass-4 figure)

Testing Step

Now relax all edges with reference to the last figure (pass-4 figure). This is the testing phase. After relaxation if no changes occur, the Bellman-Ford algorithm returns TRUE that means we can say the given graph does not contain a negative weight cycle reachable from source vertex s and the weights assigned within each vertex are the correct minimum weights of the path from the source vertex to that vertex. If

returned FALSE, we can say the given graph contains a negative weight cycle reachable from source vertex s

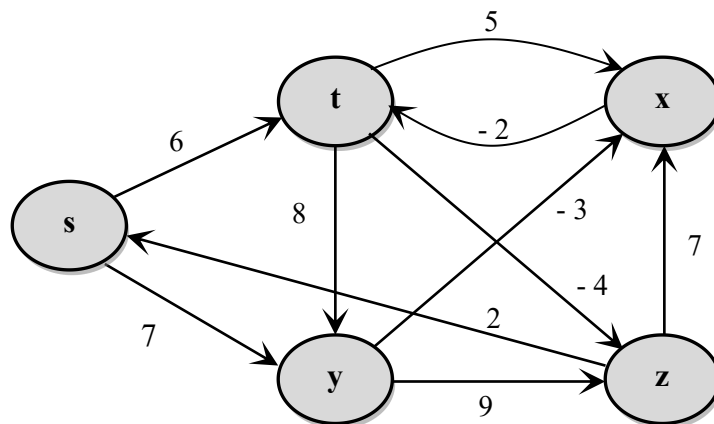
Step s	Relax edges	Result (After Relaxation)	Remarks
1		No change	
2		No change	
3		No change	
4		No change	
5		No change	
6		No change	
7		No change	
8		No change	

9		No change	
10		No change	

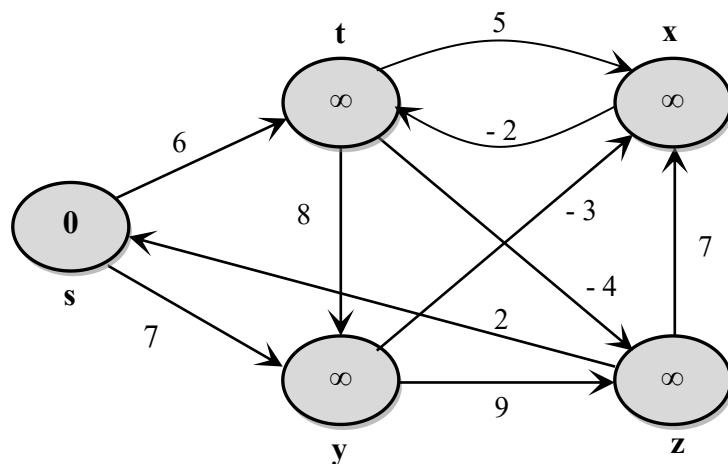
From the above table, it is observed that there is no change in the Result (After Relaxation) column. So The given graph does not contain a -ve weight cycle reachable from source vertex.

All figures

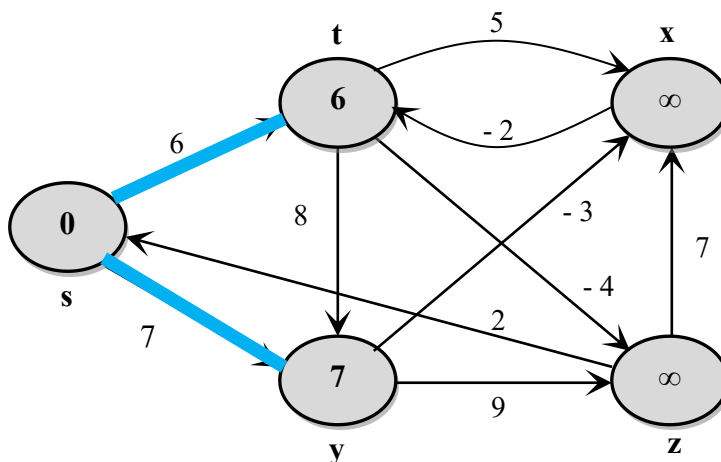
Given figure =>



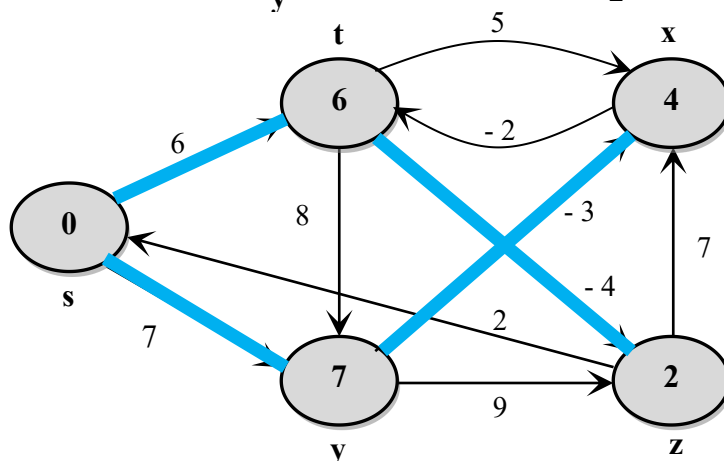
Initilize figure =>



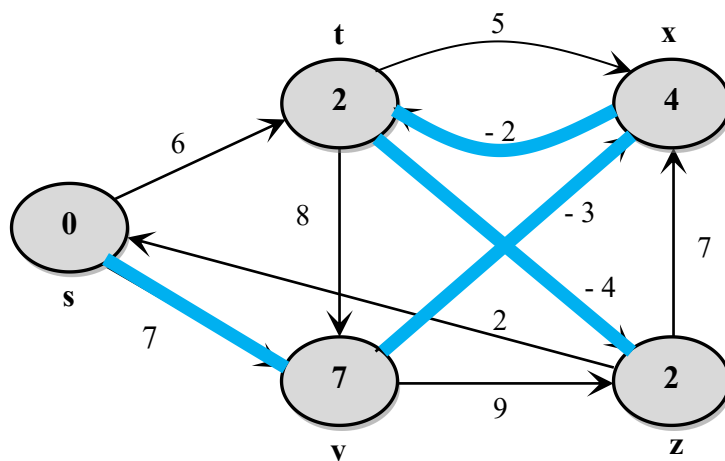
Pass-1 figure

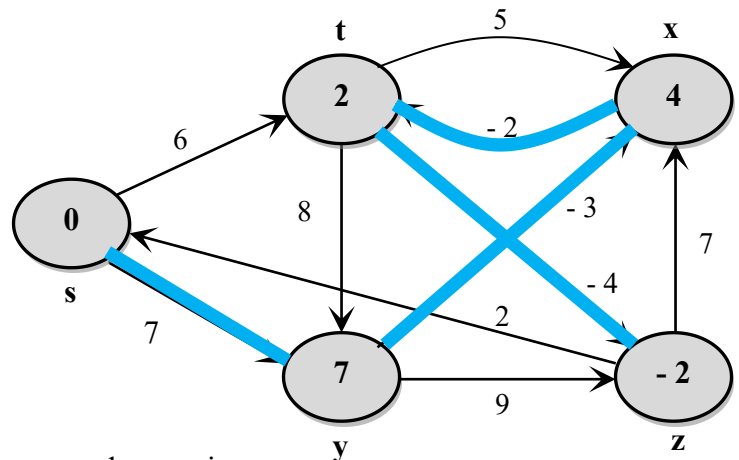


Pass-2 figure



Pass-3 figure



Pass-4 figure
(Last figure)

Min. Weightt/ Cost/ Distance from source vertex s to other vertices

Vertex (v)	s	t	x	y	z
Min. Wt/ Cost/ Distance (d[v])	0	2	4	7	-2

3 The Dijkstra's algorithm

3.1 Introduction

- Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted, directed graph $G = (V, E)$ for the case in which **all edge weights are nonnegative**. Therefore, we assume that $w(u, v) \geq 0$ for each edge $(u, v) \in E$.
- Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, adds u to S , and relaxes all edges leaving u .
- In the implementation, we use a **min-priority queue** Q of vertices, keyed by their d values. The algorithm maintains the invariant that $Q = V - S$ at the start of each iteration

```

DIJKSTRA( $G, w, s$ )
{
    INITIALIZE-SINGLE-SOURCE( $G, s$ )
     $S \leftarrow \emptyset$ 
     $Q \leftarrow V[G]$ 
    while  $Q \neq \emptyset$ 
    {
         $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
         $S \leftarrow S \cup \{u\}$ 
        for each vertex  $v \in \text{Adj}[u]$ 
            do RELAX( $u, v, w$ )
    }
}

```

3.2 The running time of Dijkstra's algorithm depends on how the min-priority queue.

- If we use the linear-array implementation of the min-priority queue, the running time is $O(E + V^2) = O(V^2)$.
- If implemented with a binary min-heap, then running time is $O((V + E) \log V) = E \log V$
- If implemented with Fibonacci heap, then running time is $O(E + V \lg V)$

3.3 Differences between Bellman-Ford and Dijkstra's Algorithms

Bellman-Ford Algorithm	Dijkstra's Algorithm
1. The Bellman-Ford algorithm solves the single-source shortest-paths problem in the general case in which edge weights may be negative.	1. Dijkstra's algorithm solves the single-source shortest-paths problem for the case in which all edge weights are nonnegative.
2. In this case, each edge is relaxed many times.	2. In this case, each edge is relaxed exactly once.
3. The running time larger than Dijkstra's Algorithm.	3. The running time is lower than that of the Bellman- Ford algorithm.

3.4 Example (The Dijkstra's algorithm)

Run Dijkstra's algorithm on the directed graph of Figure-2, using vertex s as the source. Show the d and π values and the vertices in set S after each iteration.

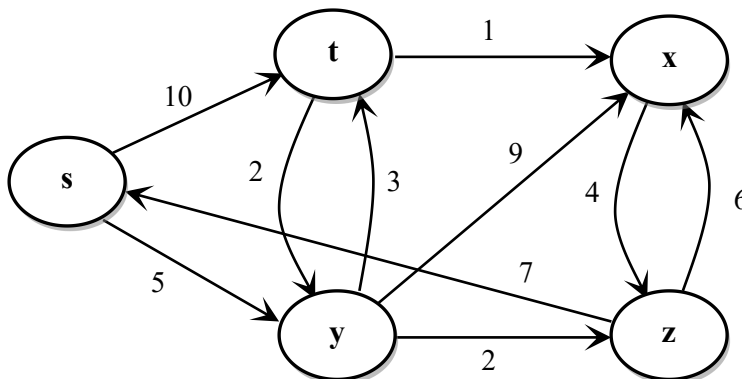


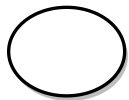
Figure-2 : Given Graph with s as source vertex

Answer:

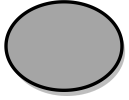
S = Set of vertices which are extracted from min priority queue Q.

π = Predecessor or parent of a vertex

d = minimum weight/cost/distance up to a vertex from source vertex




- This white color means this vertex is inside min priority queue.



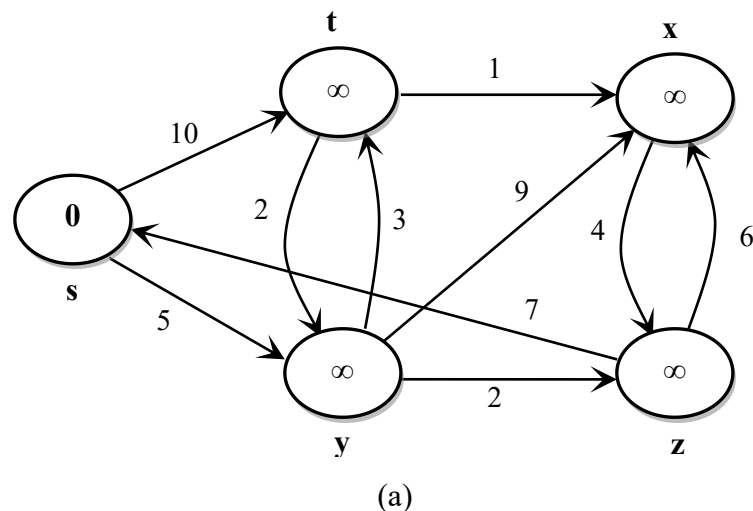
- This shaded color means the minimum cost vertex (d value is minimum). This vertex is currently dequeued.



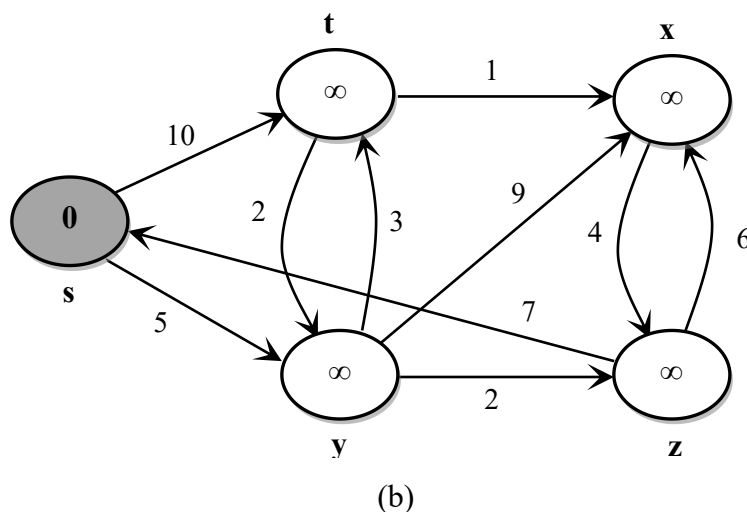
- This black color means vertex currently included in shortest path to the set S (here S is capital).

 - This blue shaded edge indicates predecessor values.

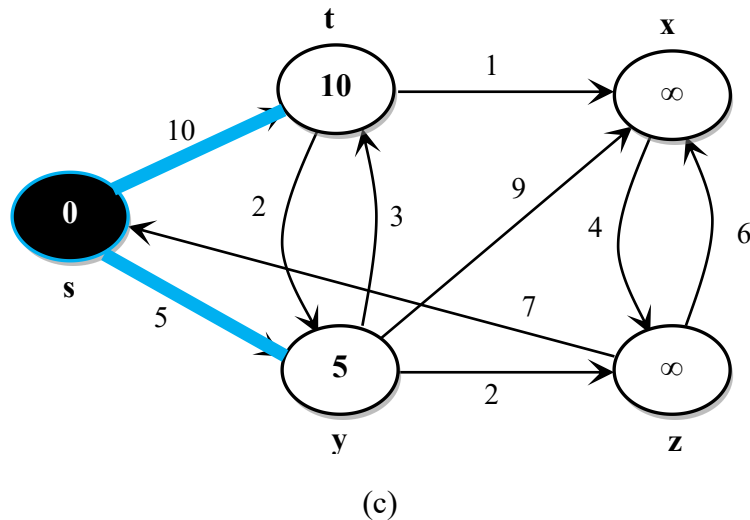
Initilize Step: initialize all vertex by writing infinity inside it except the source vertex. Write zero inside it.



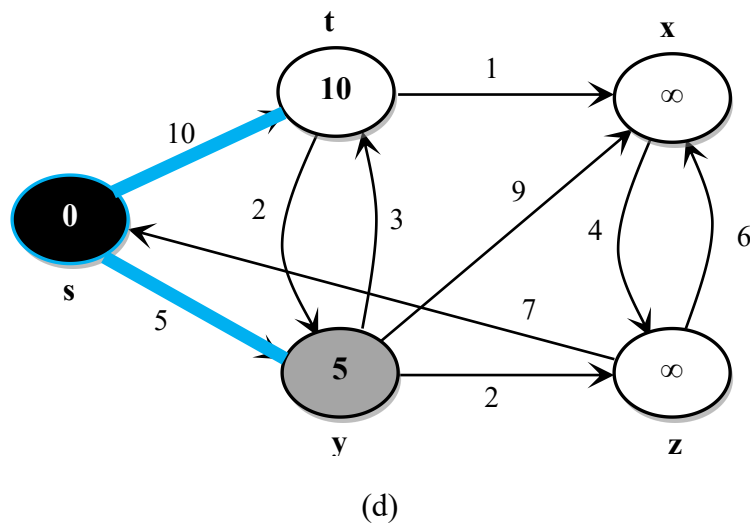
In the above figure all vertices are white that means all are initially inside queue. Find a min vertex and dequeue it. In this case it is the source vertex s. Shade the vertex s.



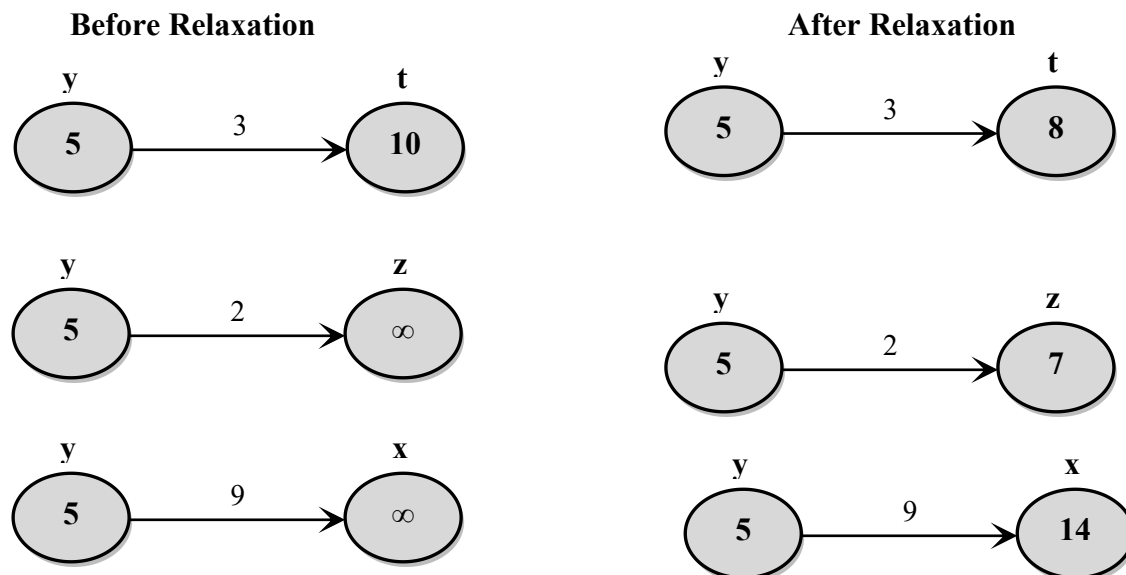
Now add it to set $S \Rightarrow S = \{s\}$, make the color of the vertex as black. Find its adjacent edges, relax them.



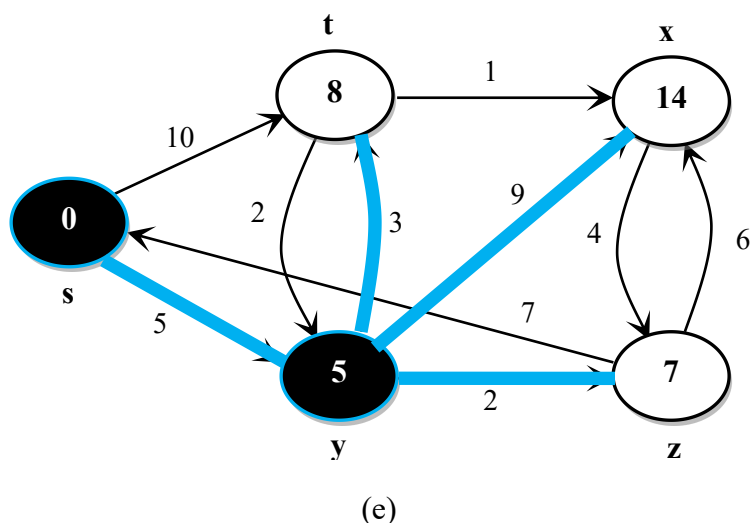
Among the rest 4 while vertices that are inside queue Q , choose a min vertex. Make dequeue operation. $Q = \{d[t], d[x], d[y], d[z]\} = \{10, \infty, \mathbf{5}, \infty\}$, here 5 is minimum. it is the d value corresponds to vertex y , so dequeue it, shade vertex y .



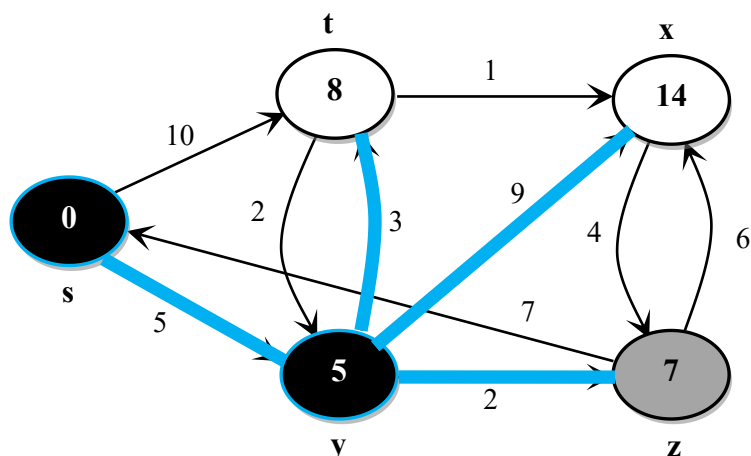
As y is the min vertex now, find all adjacent edges from it. Relax them. The edges are (y, t) , (y, z) , (y, x) . After relaxation make the color of y as black and color of the relaxed edges as blue.



As d value of vertices t, z and x have been changed (See after relaxation figure), update these in the above step. Mark that there is a blue shade edge (s, t) in the previous figure, that means t's predecessor is s. Now we found edge (y, t) to be shaded. That means t's predecessor will be y. one vertex's predecessor can not be two. So remove the old shade edge (s, t).

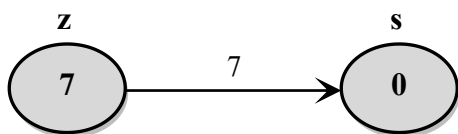


Now left 3 white vertices. Choose a min vertex among 3 vertices. Which vertex is the min vertex? It is vertex z. So make it shaded.

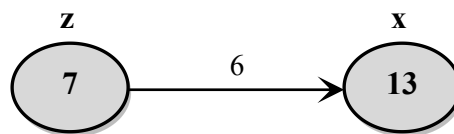
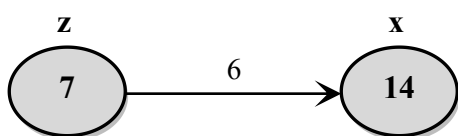


(f)

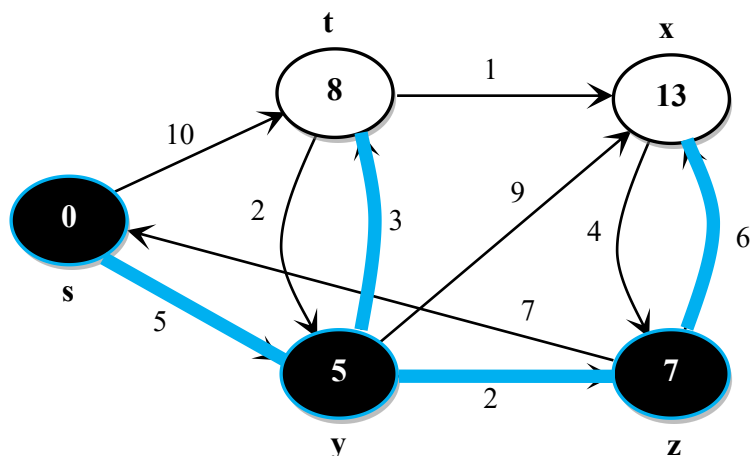
Now concentrate on vertex z, find out its adjacent edges. We found two from z, they are (z, s) and (z, x). relax them. Relaxing edge (z, s) there is no change, but there is a change on vertex x of edge (z, x).



No change as $(7+7) < 0$

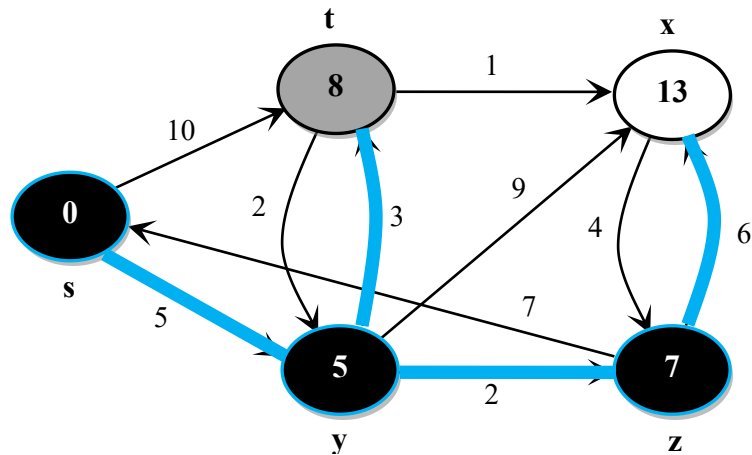


(You may omit these relaxation diagrams in examination to save time)



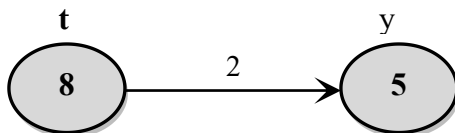
(g)

Now Q has only two values of white vertices. Min vertex is t. So make it shaded.

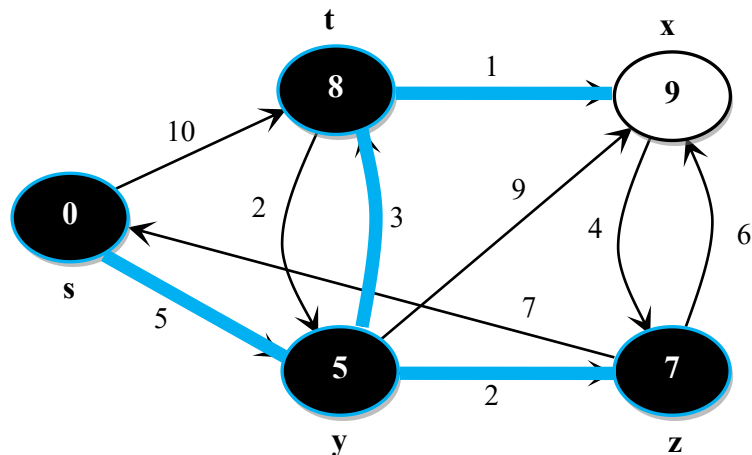
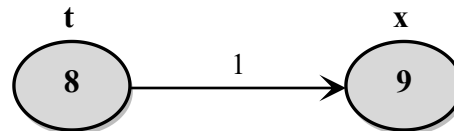
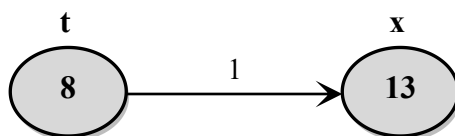


(h)

Now find the adjacent of t. Two edges found, (t, y) and (t, x). Relax them.

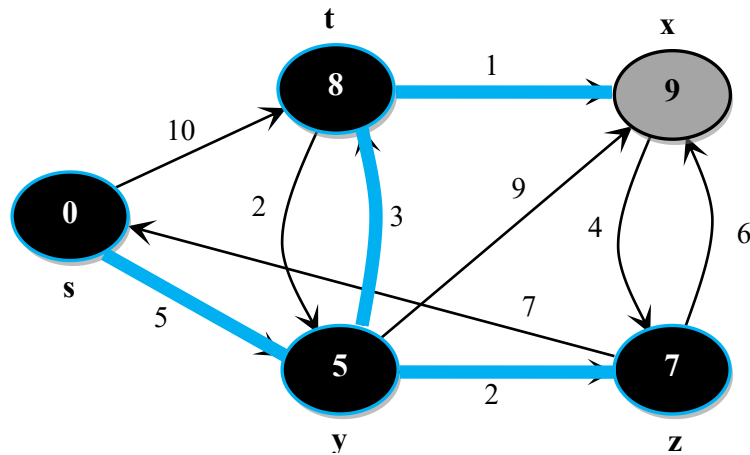


- No change



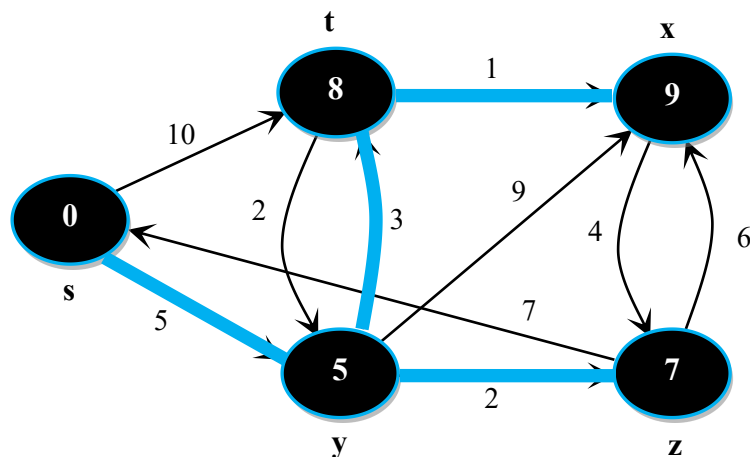
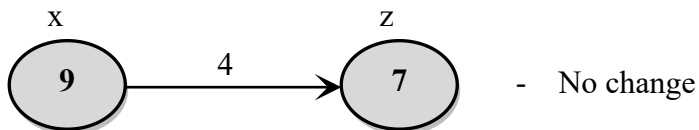
(i)

Only one white vertex is left. So it is minimum. Shade it.



(j)

Find the adjacent vertices of the min vertex x. Only one edge is found that is (x, z). relax it. After relaxation change the color to black.



(k)

(Final Figure)

As no white vertex is left, so the above figure is the final figure.

Min. Weightt/ Cost/ Distance from source vertex s to other vertices

Vertex (v)	s	t	x	y	z
Min. Wt/ Cost/ Distance (d[v])	0	8	9	5	7

4 All-Pairs Shortest Path (APSP) Problem

4.1 Introduction

- The problem of finding shortest paths between all pairs of vertices in a graph is called APSP problem.
- We can solve an all-pairs shortest-paths problem by running a single-source shortest-paths algorithm $|V|$ times, once for each vertex as the source. If all edge weights are nonnegative, we can use Dijkstra's algorithm. If we use the linear-array implementation of the min-priority queue, the running time is $O(V^3 + V E) = O(V^3)$. The binary min-heap implementation of the min-priority queue yields a running time of $O(V E \lg V)$, which is an improvement if the graph is sparse. Alternatively, we can implement the min-priority queue with a Fibonacci heap, yielding a running time of $O(V^2 \lg V + V E)$.
- If negative-weight edges are allowed, Dijkstra's algorithm can no longer be used. Instead, we must run the slower Bellman-Ford algorithm once from each vertex. The resulting running time is $O(V^2 E)$, which on a dense graph is $O(V^4)$.
- The running time can be better by implementing APSP problem by Floyd-Warshall's algorithms.

4.2 The Floyd-Warshall algorithm

- By using dynamic-programming formulation to solve the all pairs shortest-paths problem on a directed graph $G = (V, E)$, in $\Theta(V^3)$ time, is known as Floyd-Warshall algorithm.
- A recursive solution to the all-pairs shortest-paths problem
- Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j for which all intermediate vertices are in the set $\{1, 2, \dots, k\}$. When $k = 0$, a path from vertex i to vertex j with no intermediate vertex numbered higher than 0 has no intermediate vertices at all. Such a path has at most one edge, and hence $d_{ij}^{(0)} = w_{ij}$. A recursive definition following the above discussion is given by

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

- Because for any path, all intermediate vertices are in the set $\{1, 2, \dots, n\}$, the matrix $D_{ij}^{(n)} = d_{ij}^{(n)}$ gives the final answer $d_{ij}^{(n)} = \delta(i, j)$ for all $i, j \in V$.

```
FLOYD-WARSHALL( $W$ )
{
   $n \leftarrow \text{rows}[W]$ 
   $D^{(0)} \leftarrow W$ 
  for  $k \leftarrow 1$  to  $n$ 
  {
    for  $i \leftarrow 1$  to  $n$ 
      do for  $j \leftarrow 1$  to  $n$ 
        do  $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$ 
    }
  }
  return  $D^{(n)}$ 
}
```

4.3 Example (The Floyd-Warshall algorithm)

Run the Floyd-Warshall algorithm on the weighted, directed graph of Figure-3. Show the matrix $D^{(k)}$ that results for each iteration of the outer loop.

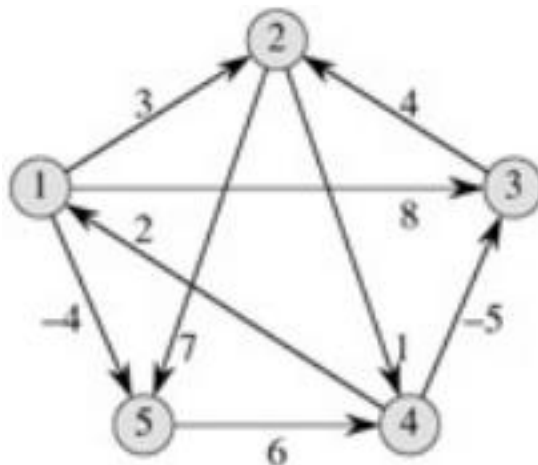


Figure-3 : Floyd-Warshall algorithm

Answer

$D^{(0)}$	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

$D^{(1)}$	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

$D^{(2)}$	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

$D^{(3)}$	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	∞	∞	∞	6	0

$D^{(4)}$	1	2	3	4	5
1	0	3	-1	4	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

$D^{(5)}$	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

$\pi^{(0)}$	1	2	3	4	5
1	NIL	1	1	NIL	1
2	NIL	NIL	NIL	2	2
3	NIL	3	NIL	NIL	NIL
4	4	NIL	4	NIL	NIL
5	NIL	NIL	NIL	5	NIL

$\pi^{(1)}$	1	2	3	4	5
1	NIL	1	1	NIL	1
2	NIL	NIL	NIL	2	2
3	NIL	3	NIL	NIL	NIL
4	4	1	4	NIL	1
5	NIL	NIL	NIL	5	NIL

$\pi^{(2)}$	1	2	3	4	5
1	NIL	1	1	2	1
2	NIL	NIL	NIL	2	2
3	NIL	3	NIL	2	2
4	4	1	4	NIL	1
5	NIL	NIL	NIL	5	NIL

$\pi^{(3)}$	1	2	3	4	5
1	NIL	1	1	2	1
2	NIL	NIL	NIL	2	2
3	NIL	3	NIL	2	2
4	4	3	4	NIL	1
5	NIL	NIL	NIL	5	NIL

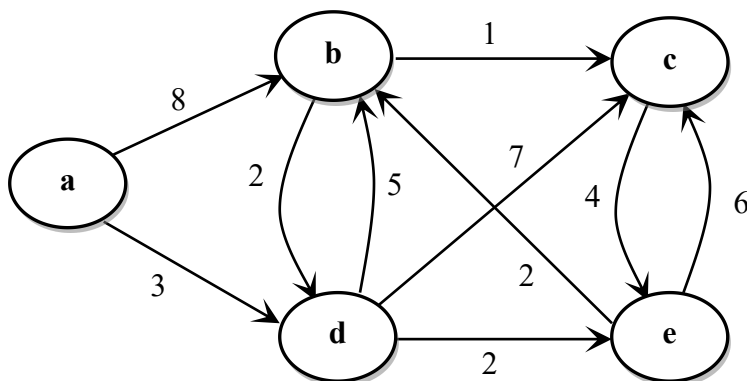
$\pi^{(4)}$	1	2	3	4	5
1	NIL	1	4	2	1
2	4	NIL	4	2	1
3	4	3	NIL	2	1
4	4	3	4	NIL	1
5	4	3	4	5	NIL

$\pi^{(5)}$	1	2	3	4	5
1	NIL	3	4	5	1
2	4	NIL	4	2	1
3	4	3	NIL	2	1
4	4	3	4	NIL	1
5	4	3	4	5	NIL

 Refer the class note how to fill up the D and π matrix.

5 Problems & Solutions

5.1 Use suitable shortest path algorithm to find out shortest path between a to c and a to e.



Answer:

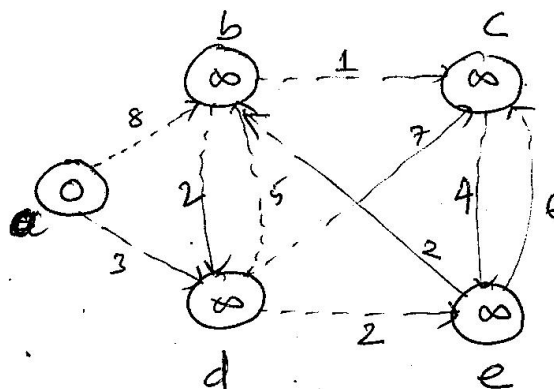
By Dijkstra's Algorithm

Shortest path between a to c : 8 (Route: a-d-e-b-c)

and a to e : 5 (Route: a-d-e)

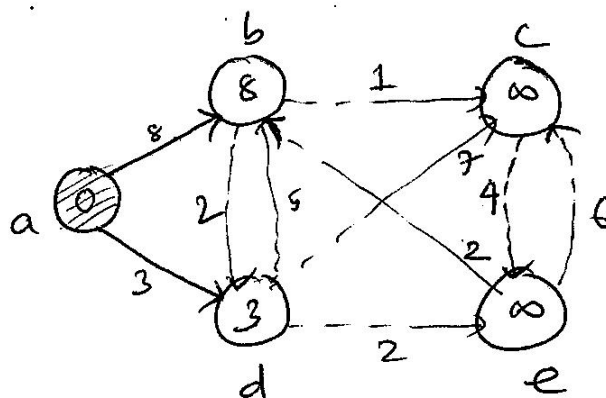
Explanation

By applying Dijkstra's Algorithm to the given graph, the initialize step figure becomes

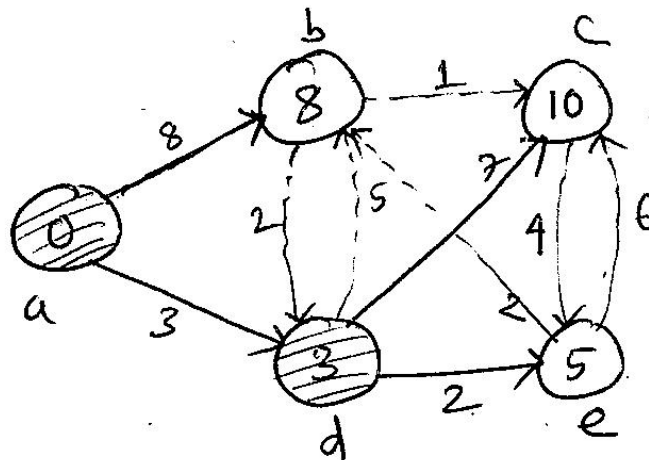


(Figure-0)

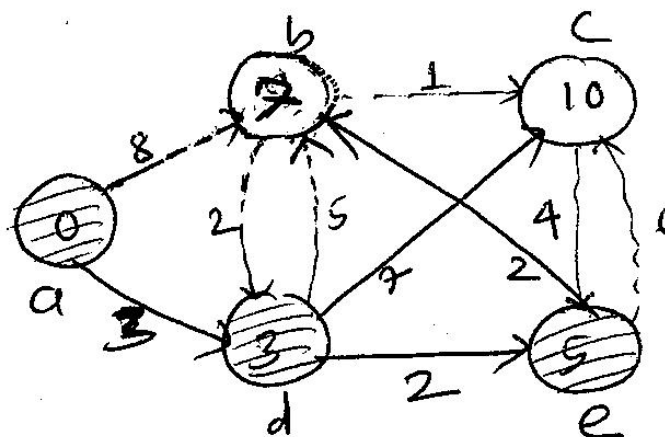
Now applying the rest part of Dijkstra's Algorithm by choosing the min-vertex from the queue followed by relax step, the figure-1 to figure-5 shows the intermediate step to final step. The value inside the circle indicates the minimum distance found so far from the given start vertex a.



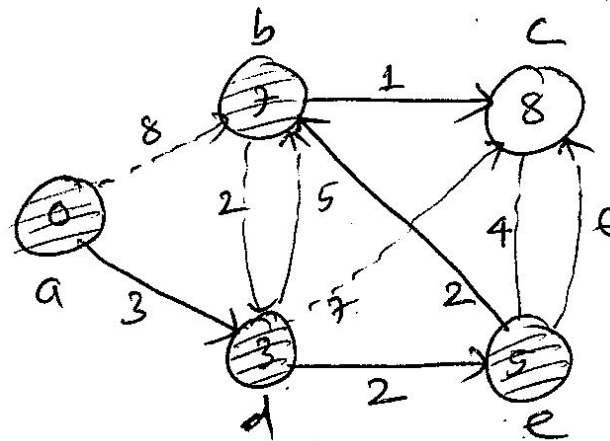
(Figure-1)



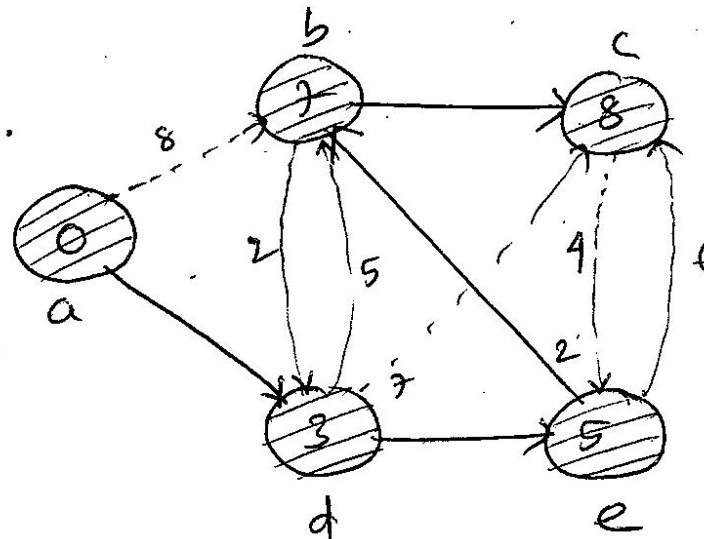
(Figure-2)



(Figure-3)



(Figure-4)



(Figure-5)