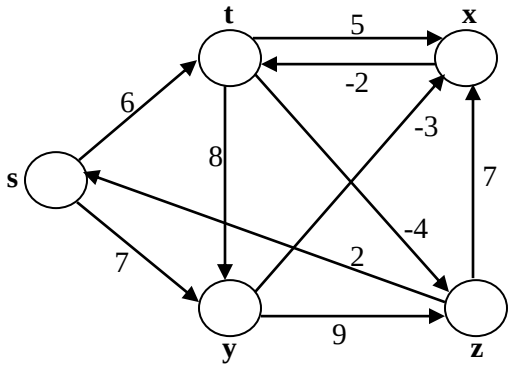
Let us assume that we have a weighted directed graph G = (V, E) given below:




Edges are: (s, t), (s, y), (t, x), (t, y), (t, z), (x, t), (z, s), (z, x), (y, x), (y, z)

Three Edges (t, z), (x, t) and (y, x) are holding Negative Edge Weights.

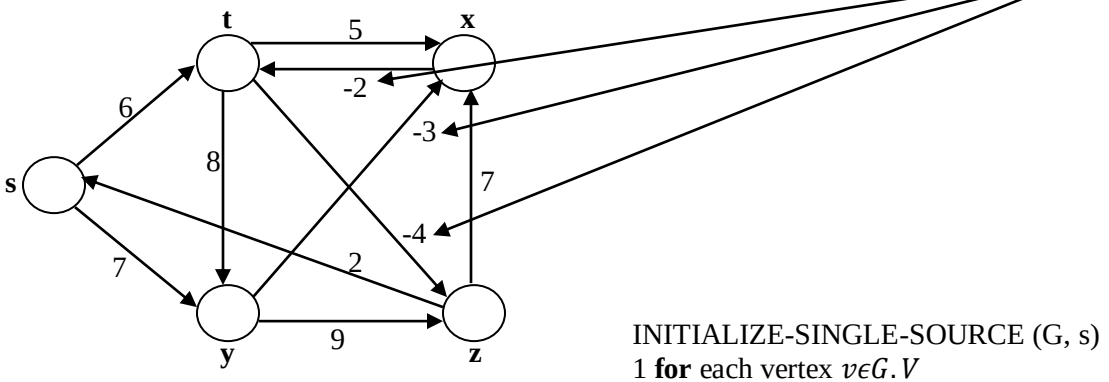
Assuming that, we have to apply Single Source Shortest Path Algorithm on this Graph.

Unfortunately the graph consists of some negative edge weights; hence we can't apply Dijkstra's Algorithm.

An alternative algorithm is Bellman Ford Algorithm which allows negative edge weights. So, we can solve this graph using Bellman Ford Algorithm.

# Bellman Ford Algorithm (Online Lecture)

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




**Algorithm**

BELLMAN-FORD (G, w, s)

1. INITIALIZE-SINGLE-SOURCE (G, s)
2. **for** i= 1 to |G.V| - 1
3. **for** each edge $(u, v) \epsilon\ G.E$
4. RELAX(u, v, w)
5. **for** each edge $(u, v) \epsilon G.E$
6. **if v.d > u.d + w (u, v)**
7. Return FALSE
8. Return TRUE

INITIALIZE-SINGLE-SOURCE (G, s)
1 **for** each vertex $v \epsilon G.V$
2 v.d = ∞
3 v.π = NIL
4 s.d = 0

Given a weighted, directed graph G= (V, E) with source s and weight function w: E -> R, the algorithm relaxes edges, progressively decreasing an estimate v.d on the weight of a shortest path from the source s to each vertex $v \epsilon V$ until it achieves the actual shortest-path weight $\delta(s, v)$.

RELAX (u, v, w)
1 **if v.d > u.d + w (u, v)**
2 v.d = u.d + w (u, v)
3 v.π = u

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

Problem and Solution

Let ‘s’ is the source vertex

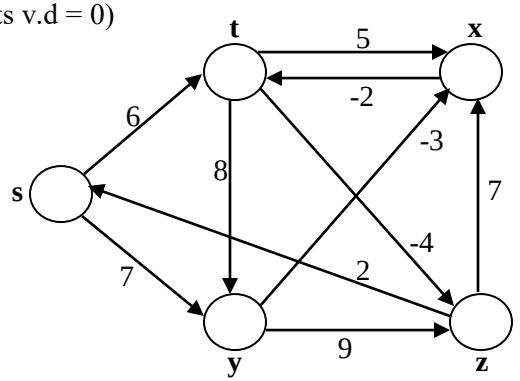
We have to calculate shortest paths from ‘s’ to other vertices

Step 1: Initialization of the Graph (for all vertices, v.d = ∞ and v.π = NIL // Identify source and assign its v.d = 0)

| | s | t | x | z | y |
|---|---|---|---|---|---|
| v.d | ∞ | ∞ | ∞ | ∞ | ∞ |
| v.π | NIL | NIL | NIL | NIL | NIL |

→

| | s | t | x | z | y |
|---|---|---|---|---|---|
| v.d | 0 | ∞ | ∞ | ∞ | ∞ |
| v.π | NIL | NIL | NIL | NIL | NIL |




Step 2:

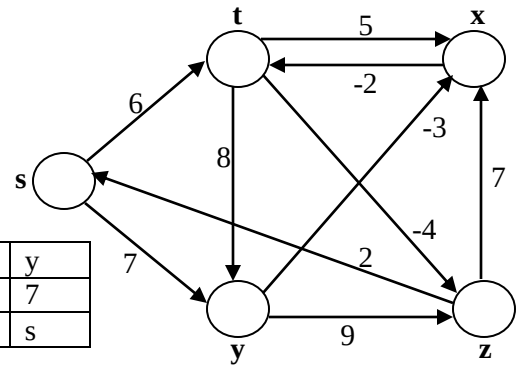
We have total 5 vertices, so total number of iteration will be (5 – 1) = 4 and for each iteration the algorithm relaxes edges, progressively decreasing an estimate v.d on the weight of a shortest path from the source s to each vertex $v \epsilon V$ until it achieves the actual shortest-path weight $\delta(s, v)$

**1st iteration**

| | s | t | x | z | y |
|---|---|---|---|---|---|
| v.d | 0 | ∞ | ∞ | ∞ | ∞ |
| v.π | NIL | NIL | NIL | NIL | NIL |

s.d = 0 // **(s, t), (s, y)** // t.d = ∞, t.π = NIL | y.d = ∞, y.π = NIL
Apply relaxation w.r.t ‘s’ to its adjacent
Check t.d > s.d + w(s,t) => ∞ > 0 + 6 => So, t.d = 6, t. π = s
Check y.d > s.d + w(s,y) => ∞ > 0 + 7 => So, y.d = 7, y. π = s

| | $s^1$ | t | x | z | y |
|---|---|---|---|---|---|
| v.d | 0 | 6 | ∞ | ∞ | 7 |
| v.π | NIL | s | NIL | NIL | s |




Now use the updated table

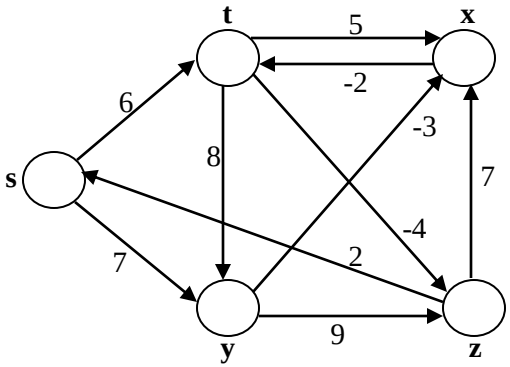
t.d = 6 // **(t, x), (t, y), (t, z)** // x.d = ∞, x.π = NIL | y.d = 7, y.π = s | z.d = ∞, z.π = NIL
Apply relaxation w.r.t ‘t’ to its adjacent
Check x.d > t.d + w(t,x) => ∞ > 6 + 5 => So, x.d = 11, x. π = t
Check y.d > t.d + w(t,y) => 7 > 6 + 8 => So No relaxation. No change of y.d and y. π
Check z.d > t.d + w(t,z) => ∞ > 6 + -4 => So, z.d = 2, z. π = t

→

| | $s^1$ | $t^1$ | x | z | y |
|---|---|---|---|---|---|
| v.d | 0 | 6 | 11 | 2 | 7 |
| v.π | NIL | s | t | t | s |

Now use updated table

| | $s^1$ | $t^1$ | x | z | y |
|---|---|---|---|---|---|
| v.d | 0 | 6 | 11 | 2 | 7 |
| v.π | NIL | s | t | t | s |




x.d = 11 // **(x, t)** // t.d = 6, t.π = s
Apply relaxation w.r.t 'x' to its adjacent
Check t.d > x.d + w(x,t) => 6 > 11 + -2 => So No relaxation. No change of t.d and t. π

No change of the last table

| | $s^1$ | $t^1$ | $x^1$ | z | y |
|---|---|---|---|---|---|
| v.d | 0 | 6 | 11 | 2 | 7 |
| v.π | NIL | s | t | t | s |

z.d = 2 // **(z, s), (z, x)** // s.d = 0, s.π = NIL | x.d = 11, x. π = t
Apply relaxation w.r.t 'z' to its adjacent
Check s.d > z.d + w(z,s) => 0 > 2 + 2 => So No relaxation. No change of s.d and s. π
Check x.d > z.d + w(z,x) => 11 > 2 + 7 => x.d = 9 and x. π = z

| | $s^1$ | $t^1$ | $x^1$ | $z^1$ | y |
|---|---|---|---|---|---|
| v.d | 0 | 6 | 9 | 2 | 7 |
| v.π | NIL | s | z | t | s |

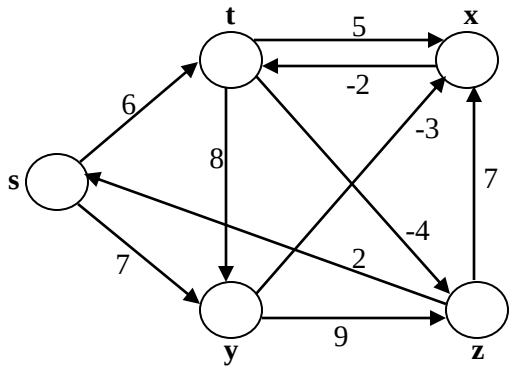
y.d = 7 // **(y, x), (y, z)** // x.d = 9, x.π = z | z.d = 2, z. π = t
Apply relaxation w.r.t 'y' to its adjacent
Check x.d > y.d + w(y,x) => 9 > 7 + -3 => So x.d = 4 and x. π = y
Check z.d > y.d + w(y,z) => 2 > 7 + 9 => So No relaxation. No change of z.d and z. π

| | $s^1$ | $t^1$ | $x^1$ | $z^1$ | $y^1$ |
|---|---|---|---|---|---|
| v.d | 0 | 6 | 4 | 2 | 7 |
| v.π | NIL | s | y | t | s |

**Now 1st iteration is complete. Use the last updated table for entering into 2nd iteration.**

**2nd Iteration**

| | $s^1$ | $t^1$ | $x^1$ | $z^1$ | $y^1$ |
|---|---|---|---|---|---|
| v.d | 0 | 6 | 4 | 2 | 7 |
| v.π | NIL | s | y | t | s |




s.d = 0 // **(s, t), (s, y)** // t.d = 6, t.π = s | y.d = 7, y.π = s
Apply relaxation w.r.t 's' to its adjacent
Check t.d > s.d + w(s,t) => 6 > 0 + 6 => So No relaxation
Check y.d > s.d + w(s,y) => 7 > 0 + 7 => So No relaxation

| | $s^2$ | $t^1$ | $x^1$ | $z^1$ | $y^1$ |
|---|---|---|---|---|---|
| v.d | 0 | 6 | 4 | 2 | 7 |
| v.π | NIL | s | y | t | s |

t.d = 6 // **(t, x), (t, y), (t, z)** // x.d = 4, x.π = y | y.d = 7, y.π = s | z.d = 2, z.π = t
Apply relaxation w.r.t 't' to its adjacent
Check x.d > t.d + w(t,x) => 4 > 6 + 5 => So, No relaxation
Check y.d > t.d + w(t,y) => 7 > 6 + 8 => So No relaxation
Check z.d > t.d + w(t,z) => 2 > 6 + -4 => So No relaxation

| | $s^2$ | $t^2$ | $x^1$ | $z^1$ | $y^1$ |
|---|---|---|---|---|---|
| v.d | 0 | 6 | 4 | 2 | 7 |
| v.π | NIL | s | y | t | s |

x.d = 4 // **(x, t)** // t.d = 6, t.π = s
Apply relaxation w.r.t 'x' to its adjacent
Check t.d > x.d + w(x,t) => 6 > 4 + -2 => So t.d = 2and t. π = x

| | $s^2$ | $t^2$ | $x^2$ | $z^1$ | $y^1$ |
|---|---|---|---|---|---|
| v.d | 0 | 2 | 4 | 2 | 7 |
| v.π | NIL | x | y | t | s |

z.d = 2 // **(z, s), (z, x)** // s.d = 0, s.π = NIL | x.d = 4, x. π = y
Apply relaxation w.r.t 'z' to its adjacent
Check s.d > z.d + w(z,s) => 0 > 2 + 2 => So No relaxation
Check x.d > z.d + w(z,x) => 4 > 2 + 7 => So No relaxation

| | $s^2$ | $t^2$ | $x^2$ | $z^2$ | $y^1$ |
|---|---|---|---|---|---|
| v.d | 0 | 2 | 4 | 2 | 7 |
| v.π | NIL | x | y | t | s |

y.d = 7 // **(y, x), (y, z)** // x.d = 4, x.π = y | z.d = 2, z. π = t
Apply relaxation w.r.t 'y' to its adjacent
Check x.d > y.d + w(y,x) => 4 > 7 + -3 => So No relaxation
Check z.d > y.d + w(y,z) => 2 > 7 + 9 => So No relaxation

Hence the updated table is

| | $s^2$ | $t^2$ | $x^2$ | $z^2$ | $y^2$ |
|---|---|---|---|---|---|
| v.d | 0 | 2 | 4 | 2 | 7 |
| v.π | NIL | x | y | t | s |

**Now 2nd iteration is complete. Use the last updated table for entering into 3rd iteration.**

**3rd Iteration**

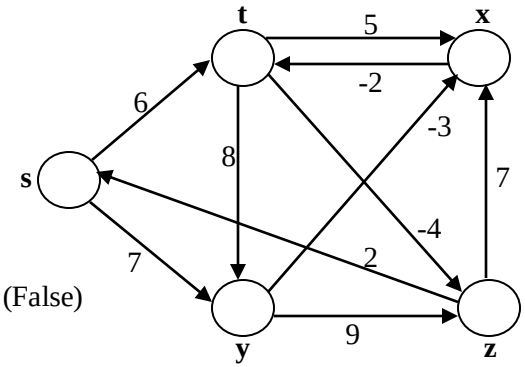
If you go for detailed calculation under 3rd Iteration, you shall find a single change for vertex 'z'. New z.d = -2 but z. π = t . So it means, with respect to the vertex't', again there is a successful relaxation for vertex 'z'. If you follow the last updated table from **iteration 2**, you will notice that **z. π = t**. As, again under **3rd iteration** there is a **successful relaxation** for the vertex '**z' w.r.t. the vertex 't'**, so **z. π is unchanged** but **z.d is changed from 2 to -2**. Hence updated table after 3rd iteration is

| | $s^3$ | $t^3$ | $x^3$ | $\mathbf{z^3}$ | $y^3$ |
|---|---|---|---|---|---|
| v.d | 0 | 2 | 4 | **-2** | 7 |
| v.π | NIL | x | y | **t** | s |

**4th Iteration**

You will not get any change in 4th iteration. 3rd and 4th iteration result will be same. Hence updated table after 4th iteration is

| | $s^4$ | $t^4$ | $x^4$ | $z^4$ | $y^4$ |
|---|---|---|---|---|---|
| v.d | 0 | 2 | 4 | -2 | 7 |
| v.π | NIL | x | y | t | s |




Now check for negative weigh cycle (Refer Final Table)

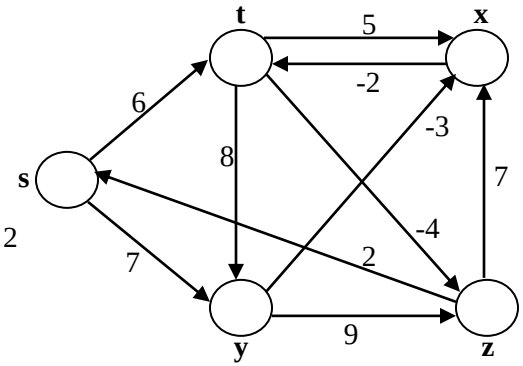
**for** each edge $(u, v) \epsilon G.E$
  **if v.d > u.d + w (u, v)**
    Return FALSE

(t, x), Check x.d > t.d + w (t, x) => 4 > 2 + 5 => 4 > 7 (False)
***So, No negative weight cycle is found***
Similarly check for others.

**It is observed that this graph, Bellman-Ford algorithm returns TRUE.**

**Refer Final Table**

| | $s^4$ | $t^4$ | $x^4$ | $z^4$ | $y^4$ |
|---|---|---|---|---|---|
| v.d | 0 | 2 | 4 | -2 | 7 |
| v.π | NIL | x | y | t | s |




Path formation logic is same as Dijkstra's method. [Go through my youtube lecture]

Now, shortest distance from s to t is t.d value from the final table generated after 4th iteration, i.e. 2

Path from s to t is formed as s -> y -> x -> t

Check for other vertices from source 's'.

Complexity Analysis: Consider the algorithm again (shown below)

BELLMAN-FORD (G, w, s)

1. INITIALIZE-SINGLE-SOURCE (G, s)
2. **for** i= 1 to |G.V| - 1
3. **for** each edge $(u, v) \epsilon\ G.E$
4. RELAX(u, v, w)
5. **for** each edge $(u, v) \epsilon G.E$
6. **if v.d > u.d + w (u, v)**
7. Return FALSE
8. Return TRUE

The Bellman-Ford algorithm runs in time O(VE), since the initialization in line 1 takes Θ(V) time, each of the |V| - 1 passes over the edges in lines 2–4 takes Θ(E) time, and the **for** loop of lines 5–7 takes O(E) time.

**To prove the correctness of the Bellman-Ford algorithm, we start by showing that if there are no negative-weight cycles, the algorithm computes correct shortest-path weights for all vertices reachable from the source.**