Johnson’s algorithm for All-pairs shortest paths

- Johnson's Algorithm uses both Dijkstra's Algorithm and Bellman-Ford Algorithm.
- Johnson's Algorithm uses the technique of **"reweighting."**
- If all edge weights w in a graph G = (V, E) are nonnegative, we can find the shortest paths between all pairs of vertices by running Dijkstra's Algorithm once from each vertex.
- If G has negative - weight edges, we compute a new - set of non - negative edge weights that allows us to use the same method.

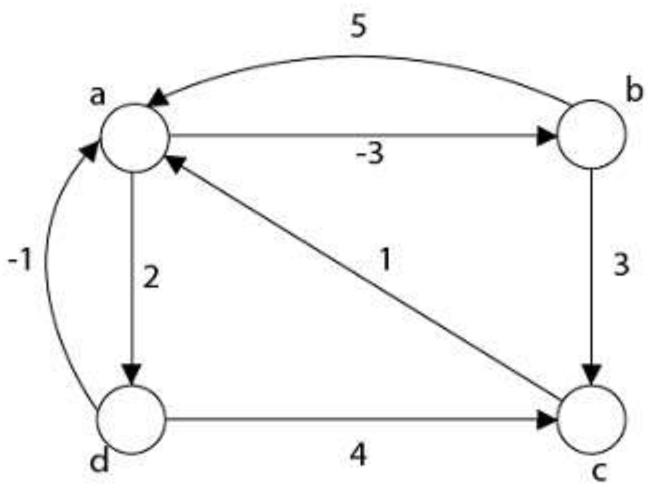
The new set of edges weight $\widehat{w}$ must satisfy two essential properties:

1. For all pairs of vertices $u, v \in V$, a path p is a shortest path from u to v using weight function w if and only if p is also a shortest path from u to v using weight function $\widehat{w}$.
2. For all edges (u, v), the new weight $\widehat{w}(u, v)$ is nonnegative.

- For each edge (u, v) ∈ E define

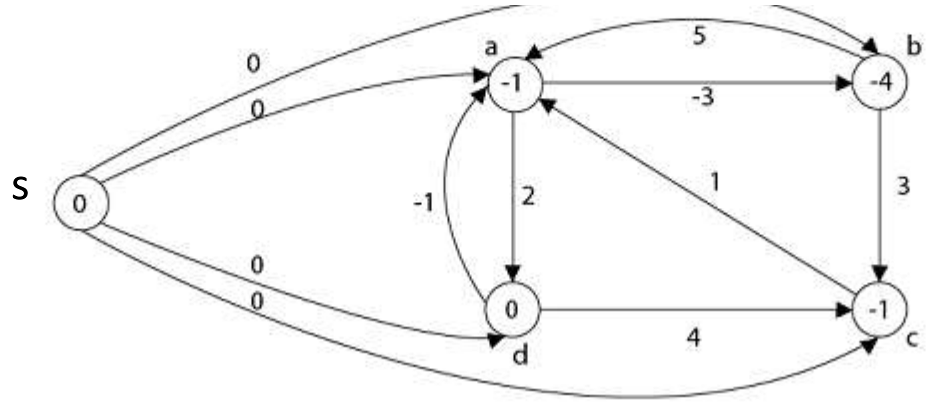
$$\widehat{w}(u, v) = w(u, v) + h(u) - h(v) .$$

- Where h (u) = label of u
  h (v) = label of v



Step1: Take any source vertex's' outside the graph and make distance from 's' to every vertex '0'.
Step2: Apply Bellman-Ford Algorithm and calculate minimum weight on each vertex.

$$dist^k[u] = \min\ \{dist^{k-1}[u],\ \min_i\ \{dist^{k-1}[i] + cost[i,u]\}\}\quad k = 2, 3, \ldots,\ n-1.$$



Step3: w (a, b) = w (a, b) + h (a) - h (b)

= -3 + (-1) - (-4)

= 0

w (b, a) = w (b, a) + h (b) - h (a)

= 5 + (-4) - (-1)

= 2

w (b, c) = w (b, c) + h (b) - h (c)

w (b, c) = 3 + (-4) - (-1)

= 0

w (c, a) = w (c, a) + h (c) - h (a)

w (c, a) = 1 + (-1) - (-1)

= 1

$w(d, c) = w(d, c) + h(d) - h(c)$

$w(d, c) = 4 + 0 - (-1)$

$= 5$

$w(d, a) = w(d, a) + h(d) - h(a)$

$w(d, a) = -1 + 0 - (-1)$

$= 0$

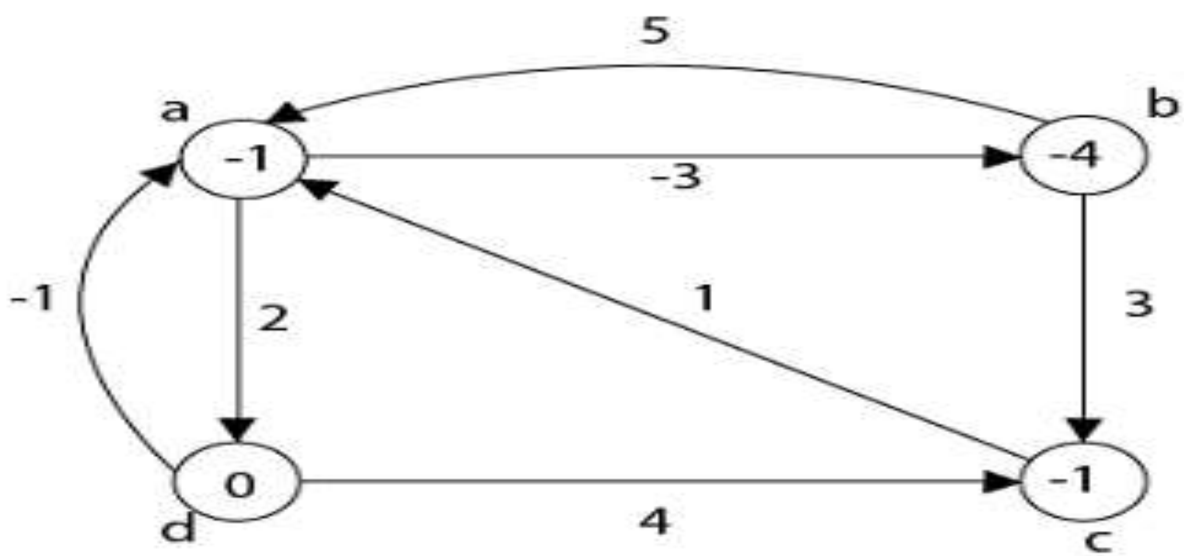
$w(a, d) = w(a, d) + h(a) - h(d)$

$w(a, d) = 2 + (-1) - 0 = 1$

- **Step 4:** Now all edge weights are positive and now we can apply Dijkstra's Algorithm on each vertex and make a matrix corresponds to each vertex in a graph

	a	b	c	d
a	0	0	0	1
b	1	0	0	2
c	1	1	0	2
d	0	0	0	0

Step5:



$d_{uv} \leftarrow \delta(u, v) + h(v) - h(u)$

$d(a, a) = 0 + (-1) - (-1) = 0$

$d(a, b) = 0 + (-4) - (-1) = -3$

$d(a, c) = 0 + (-1) - (-1) = 0$

$d(a, d) = 1 + (0) - (-1) = 2$

$d(b, a) = 1 + (-1) - (-4) = 4$

$d(b, b) = 0 + (-4) - (-4) = 0$

$d(c, a) = 1 + (-1) - (-1) = 1$

$d(c, b) = 1 + (-4) - (-1) = -2$

$d(c, c) = 0$

$d(c, d) = 2 + (0) - (-1) = 3$

$d(d, a) = 0 + (-1) - (0) = -1$

$d(d, b) = 0 + (-4) - (0) = -4$

$d(d, c) = 0 + (-1) - (0) = -1$

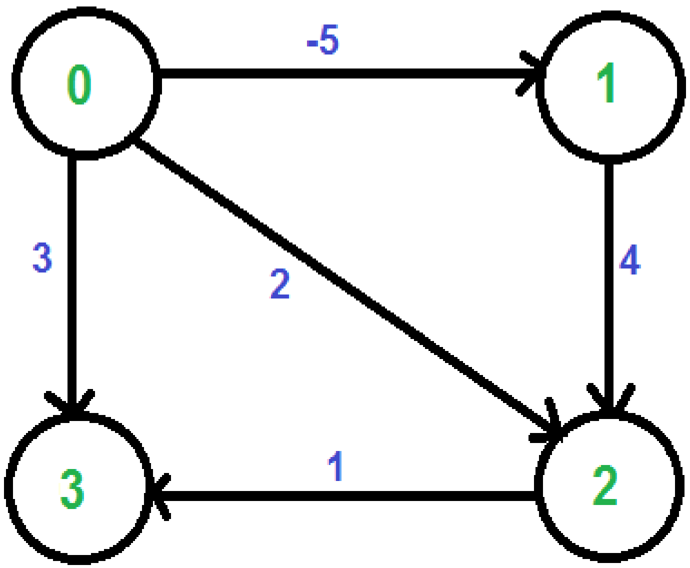
$d(d, d) = 0$

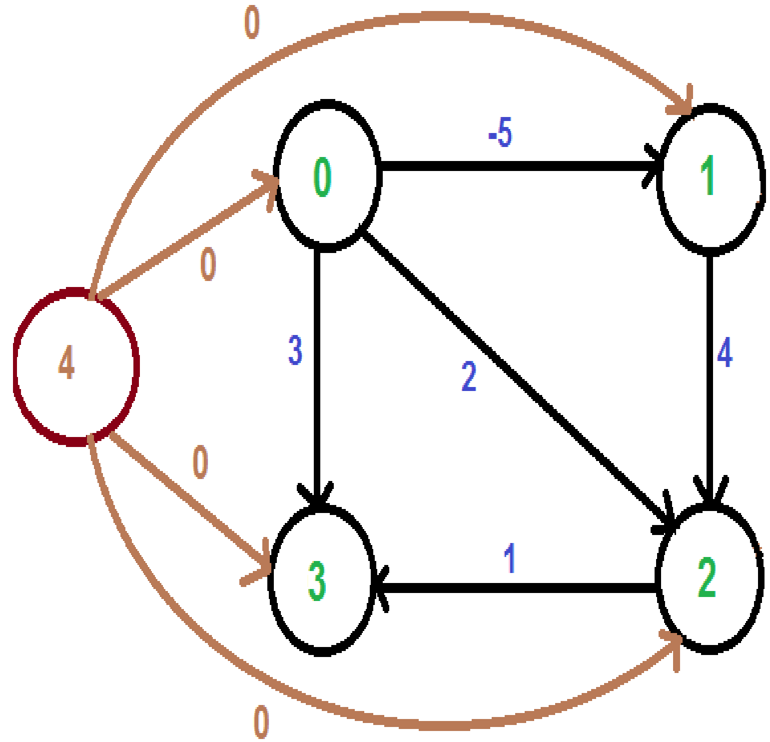
	a	b	c	d
a	0	-3	0	2
b	4	0	3	6
c	1	-2	0	3
d	-1	-4	-1	0

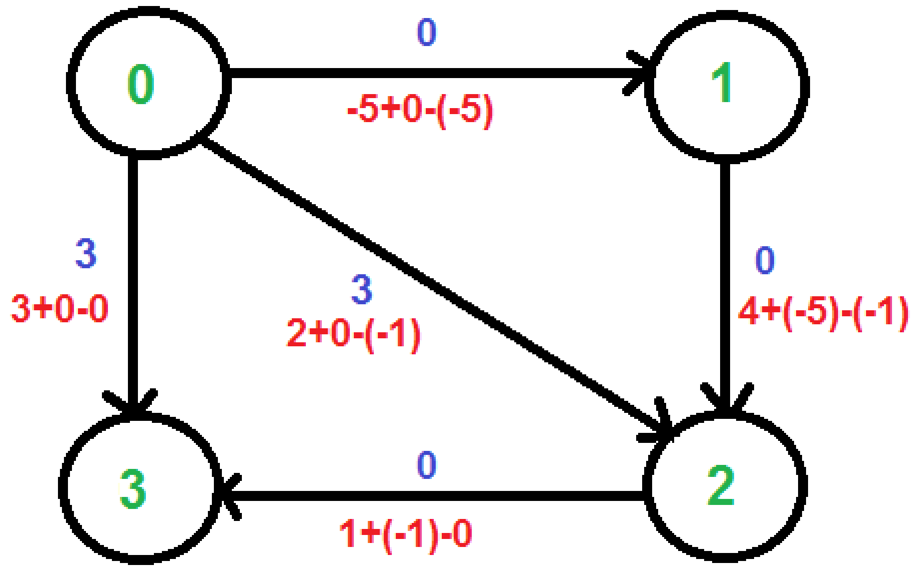
Algorithm

```
JOHNSON (G)
 1. Compute G' where V [G'] = V[G] ∪ {S} and
E [G'] = E [G] ∪ {(s, v): v ∈ V [G] }

 2. If BELLMAN-FORD (G',w, s) = FALSE
     then "input graph contains a negative weight cycle"
   else
     for each vertex v ∈ V [G']
      do h (v) ← δ(s, v)
   Computed by Bellman-Ford algorithm
  for each edge (u, v) ∈ E[G']
    do w (u, v) ← w (u, v) + h (u) - h (v)
  for each edge u ∈ V [G]
  do run DIJKSTRA (G, w, u) to compute
     δ (u, v) for all v ∈ V [G]
   for each vertex v ∈ V [G]
  do duv← δ (u, v) + h (v) - h (u)
Return D.
```


0
1
2
3
-5
3
2
4
1


4
0
1
2
3
-5
3
2
4
1
0
0
0
0



Distances from 4 to 0, 1, 2 and 3 are 0, -5, -1 and 0 respectieveły.