DIGRAFOS: CAMINO MAS CORTO

ISIS 1206

Camino mas corto

$$G = \{V, E\} \qquad w : E \to \Re$$

$$P = \langle v_0, v_1, ..v_k \rangle \qquad w(P) = \sum_{i=1}^{k} w(v_{i-1} \to v_i)$$

Camino mas corto desde u hasta v

$$\delta(u, v) = \left\{ \begin{array}{ll} \min\{w(P) : u \overset{P}{\rightsquigarrow} v\}, & \text{Si } u \rightsquigarrow v \\ \infty, & \text{en caso contrario} \end{array} \right\}$$

Camino mas corto (Variantes)

Unica fuente

Dado una fuente `s`, encontrar los caminos más cortos a todos los destinos

Unico destino (Single destination)

Dado un destino `t`, encontrar los caminos más cortos desde el resto de vértices

Unica fuente, unico destino (Single pair)

Dado una fuente `s` y un destino `t`, encontrar el caminos más corto entre ambos vértices

Todas las parejas (All pairs)

Encontrar los caminos más cortos entre todas las parejas de vértices

Arbol de caminos más cortos (Shortest-paths tree - SPT)

Dado un origen *s* en

$$G = \{V, E\}$$

$$G' = \{V', E'\}$$

$$V' \subseteq V$$

$$E' \subseteq E$$

1. *V'* es el conjunto de vertices alcanzable desde *s* en el grafo original

2. *G'* es un árbol con raíz *s*

4. $\forall v \in V'$, cualquier camino desde *s* a *v*, es el camino más corto

CÓMO SE CONSTRUYE UN CAMINO MAS CORTO DESDE S -> V, USANDO UN SPT ?

Algoritmo Genérico

Sub-estructura optima

Dado $P_{u \to v} = u \to e \to ... \to v$, el camino mas corto entre *u* y *v*

$P_{u \to x} \subseteq P_{u \to v}$, es también el camino mas corto entre u y x

Desigualdad triángulo

$$\delta(u, v) \leq \delta(u, x) + \delta(x, v)$$

PRUEBE EL LEMA DE LA SUB-ESTRUCTURA OPTIMA

Relaxation

http://benralston.blogspot.com.co/2010/05/relaxation-1-what-is-relaxation.html

Relaxation

Técnica usada en los algoritmos de **relajación** (grafos) para estimar una cota superior del peso total más corto entre dos vértices

1. Inicialización

$$\delta(s,v) = \begin{cases} 0, & \text{Si } v = s \\ \infty, & \text{en caso contrario} \end{cases}$$

$\pi(v) = \emptyset, \forall v \in V$ //Predecesor de *v* en el SPT

Relaxation

2. Paso de “relajación”

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

$$\delta(s, v) = d \qquad x \rightsquigarrow v \qquad \pi(v) = x$$

Relaxation

2. Paso de “relajación”

La cota superior es menor

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

Qué pasa si hay otro vértice *k* tal que ?

$$\delta(s,x) + w(x,v) \geq \delta(s,k) + w(k,v)$$

$$s \rightsquigarrow k, k \rightsquigarrow v$$

Relaxation

2. Paso de “relajación”

La cota superior es menor

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

$$\delta(s,x) + w(x,v) \geq \delta(s,k) + w(k,v)$$

$$s \rightsquigarrow k, k \rightsquigarrow v$$

$$\delta(s,v) \leq \delta(s,k) + w(k,v) \qquad \delta(s,v) = d' < d \qquad \pi(v) = k$$

Relaxation

2. Paso de “relajación”

La cota superior es menor

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

$$\delta(s,x) + w(x,v) \geq \delta(s,k) + w(k,v)$$

$$s \rightsquigarrow k, k \rightsquigarrow v$$

$$\delta(s,v) \leq \delta(s,k) + w(k,v) \qquad \delta(s,v) = d' < d \qquad \pi(v) = k$$

Relaxation

```
private void relax(DirectedEdge e)
{
   int v = e.from(), w = e.to();
   if (distTo[w] > distTo[v] + e.weight())
   {
      distTo[w] = distTo[v] + e.weight();
      edgeTo[w] = e;
   }
}
```

Tomado de: Algorithms (4th ed.) by Robert Sedgewick and Kevin Wayne

Relaxation

```
private void relax(DirectedEdge e)
{
   int v = e.from(), w = e.to();
   if (distTo[w] > distTo[v] + e.weight())
   {
      distTo[w] = distTo[v] + e.weight();
      edgeTo[w] = e;
   }
}
```

$\pi(w)$

Tomado de: Algorithms (4th ed.) by Robert Sedgewick and Kevin Wayne

Relaxation

```
private void relax(DirectedEdge e)
{
   int v = e.from(), w = e.to();
   if (distTo[w] > distTo[v] + e.weight())
   {
      distTo[w] = distTo[v] + e.weight();
      edgeTo[w] = e;
   }
}
```

$\delta(s, w)$

Tomado de: Algorithms (4th ed.) by Robert Sedgewick and Kevin Wayne

Propiedades de caminos cortos (consecuencia del concepto de relajación)

Desigualdad triángulo $\delta^*(s,v) \leq \delta(s,k) + w(k,v)$

Cota superior $\delta(s,v) \geq \delta^*(s,v)$

No-camino $\nexists(s \rightsquigarrow v) \implies \delta(s,v) = \infty$

Subgrafo del predecesor (subcamino óptimo) Si $\delta(s,v) = \delta^*(s,v), \forall k$ en el camino de predecesores de $v, \delta(s,k)$es el camino mas corto

Algoritmo Genérico

Relaje cada vértice, hasta que no hay mas vértices en la cola/lista de trabajo

Algoritmos: Dijkstra (greedy), Bellman-Ford (prog. dinámica)

Algoritmo de Dijkstra [dɛikstra] (Codicioso)

(Asume que todos los pesos son no-negativos)

ALGORITHM 4.9 Dijkstra's shortest-paths algorithm

```
public class DijkstraSP
{
   private DirectedEdge[] edgeTo;
   private double[] distTo;
   private IndexMinPQ<Double> pq;

   public DijkstraSP(EdgeWeightedDigraph G, int s)
   {
      edgeTo = new DirectedEdge[G.V()];
      distTo = new double[G.V()];
      pq = new IndexMinPQ<Double>(G.V());

      for (int v = 0; v < G.V(); v++)
         distTo[v] = Double.POSITIVE_INFINITY;
      distTo[s] = 0.0;

      pq.insert(s, 0.0);
      while (!pq.isEmpty())
         relax(G, pq.delMin())
   }

   private void relax(EdgeWeightedDigraph G, int v)
   {
      for(DirectedEdge e : G.adj(v))
      {
         int w = e.to();
         if (distTo[w] > distTo[v] + e.weight())
         {
            distTo[w] = distTo[v] + e.weight();
            edgeTo[w] = e;
            if (pq.contains(w)) pq.change(w, distTo[w]);
            else                pq.insert(w, distTo[w]);
         }
      }
   }
```

Tomado de: Algorithms (4th ed.) by Robert Sedgewick and Kevin Wayne

Algoritmo de Dijkstra (Codicioso)

(Asume que todos los pesos son no-negativos)

ALGORITHM 4.9 Dijkstra's shortest-paths algorithm

```
public class DijkstraSP
{
   private DirectedEdge[] edgeTo;
   private double[] distTo;
   private IndexMinPQ<Double> pq;

   public DijkstraSP(EdgeWeightedDigraph G, int s)
   {
      edgeTo = new DirectedEdge[G.V()];
      distTo = new double[G.V()];
      pq = new IndexMinPQ<Double>(G.V());

      for (int v = 0; v < G.V(); v++)
         distTo[v] = Double.POSITIVE_INFINITY;
      distTo[s] = 0.0;

      pq.insert(s, 0.0);
      while (!pq.isEmpty())
         relax(G, pq.delMin())
   }
```

Tomado de: Algorithms (4th ed.) by Robert Sedgewick and Kevin Wayne

Algoritmo de Dijkstra (Codicioso)

(Asume que todos los pesos son no-negativos)

ALGORITHM 4.9 Dijkstra's shortest-paths algorithm

```
public class DijkstraSP
{
   private DirectedEdge[] edgeTo;
   private double[] distTo;
   private IndexMinPQ<Double> pq;

   public DijkstraSP(EdgeWeightedDigraph G, int s)
   {
      edgeTo = new DirectedEdge[G.V()];
      distTo = new double[G.V()];
      pq = new IndexMinPQ<Double>(G.V());

      for (int v = 0; v < G.V(); v++)
         distTo[v] = Double.POSITIVE_INFINITY;
      distTo[s] = 0.0;

      pq.insert(s, 0.0);
      while (!pq.isEmpty())
         relax(G, pq.delMin());
   }
```

Codicioso !!!

Tomado de: Algorithms (4th ed.) by Robert Sedgewick and Kevin Wayne

Algoritmo de Dijkstra (Codicioso)

(Asume que todos los pesos son no-negativos)

```
private void relax(EdgeWeightedDigraph G, int v)
{
   for(DirectedEdge e : G.adj(v))
   {
      int w = e.to();
      if (distTo[w] > distTo[v] + e.weight())
      {
         distTo[w] = distTo[v] + e.weight();
         edgeTo[w] = e;
         if (pq.contains(w)) pq.change(w, distTo[w]);
         else                pq.insert(w, distTo[w]);
      }
   }
}
```

Tomado de: Algorithms (4th ed.) by Robert Sedgewick and Kevin Wayne

Algoritmo de Dijkstra (Codicioso)

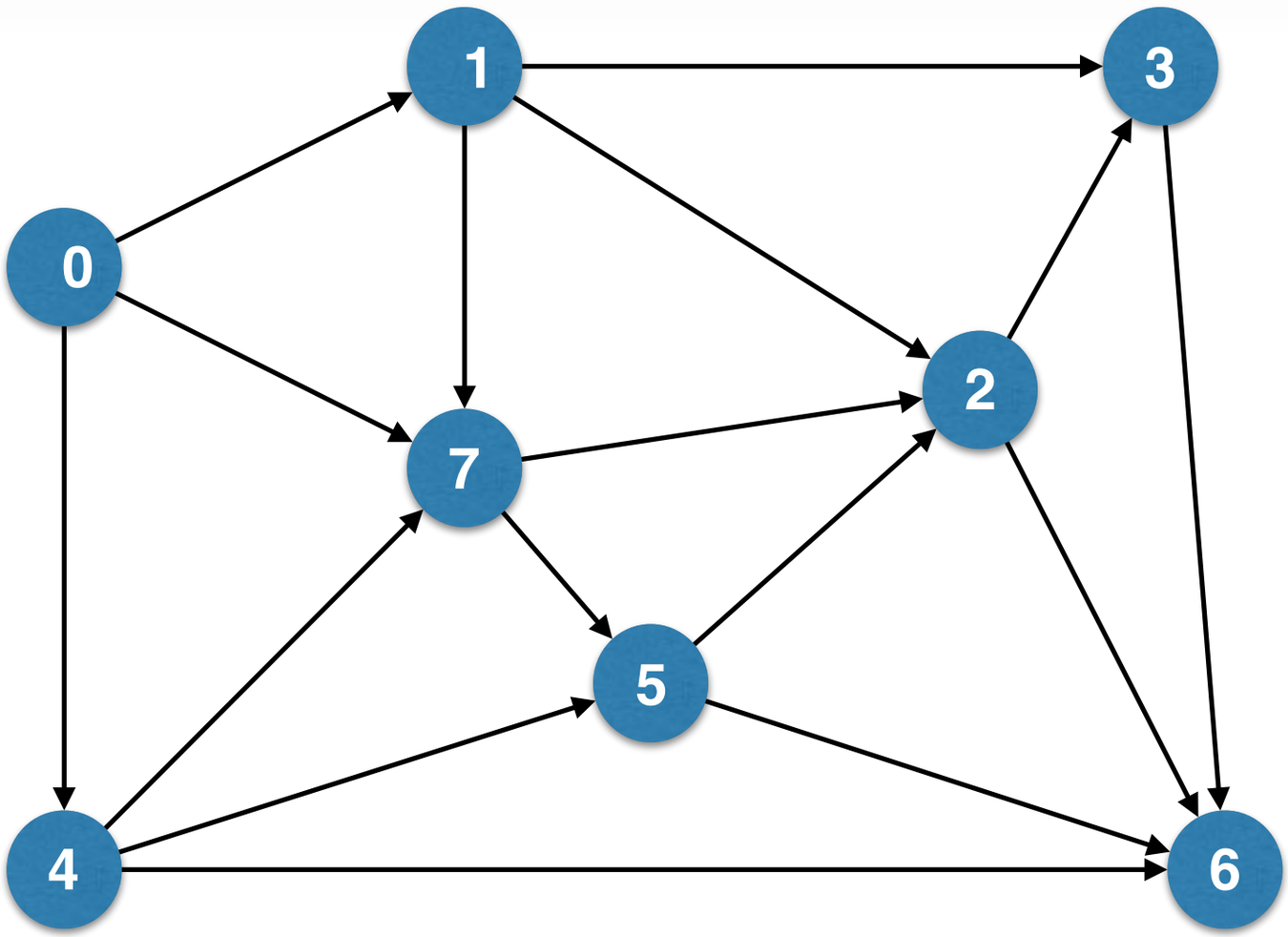
(Asume que todos los pesos son no-negativos)

```
private void relax(EdgeWeightedDigraph G, int v)
{
   for(DirectedEdge e : G.adj(v))
   {
      int w = e.to();
      if (distTo[w] > distTo[v] + e.weight())
      {
         distTo[w] = distTo[v] + e.weight();
         edgeTo[w] = e;
         if (pq.contains(w)) pq.change(w, distTo[w]);
         else                pq.insert(w, distTo[w]);
      }
   }
}
```

$\delta(s, w)$ $\pi(w)$

Tomado de: Algorithms (4th ed.) by Robert Sedgewick and Kevin Wayne

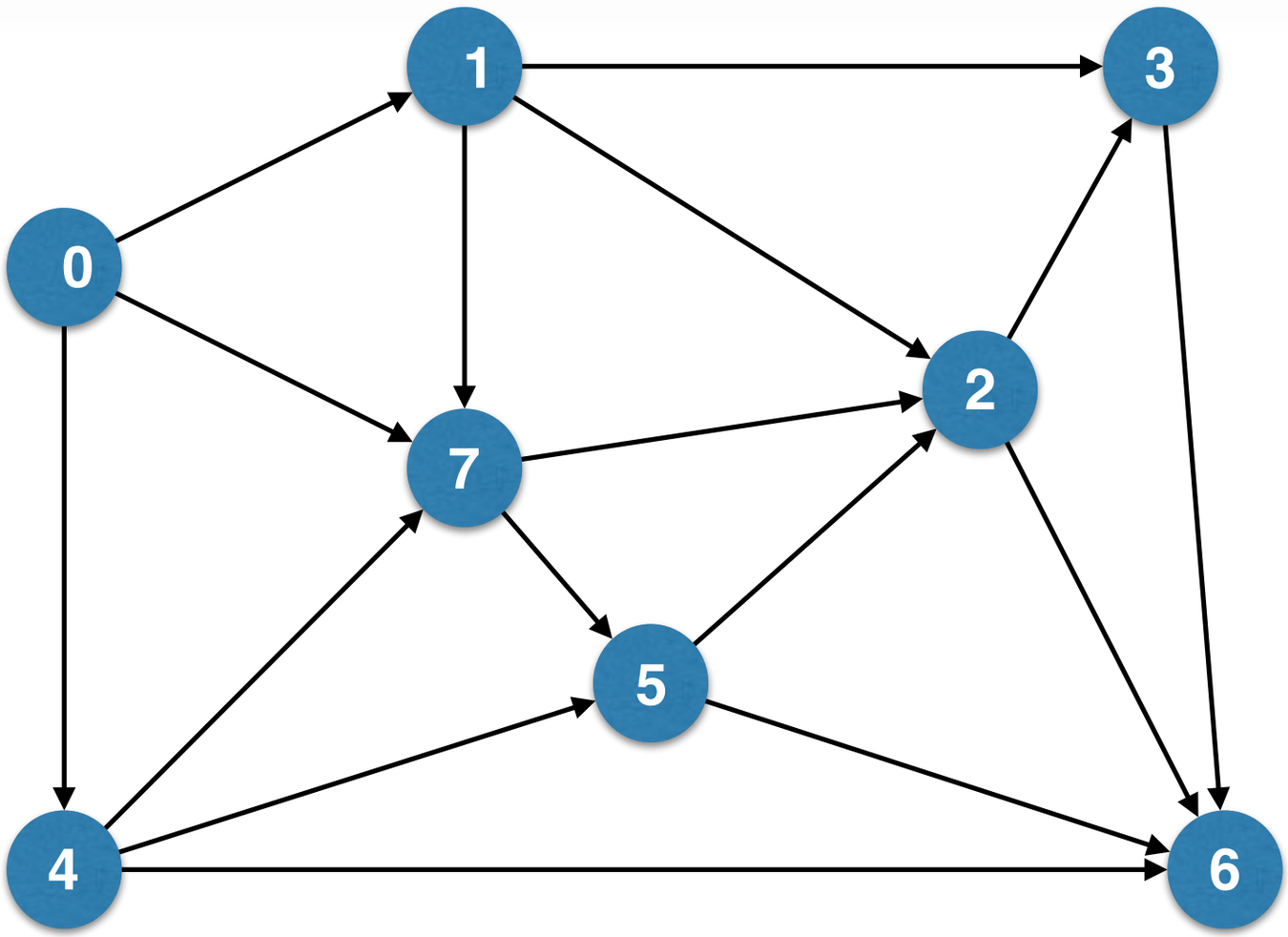
Algoritmo de Dijkstra (Codicioso)



v	distTo[v] $\delta(s,v)$	edge[v] $\pi(v)$
0		
1		
2		
3		
4		
5		
6		
7		

Algoritmo de Dijkstra (Codicioso)

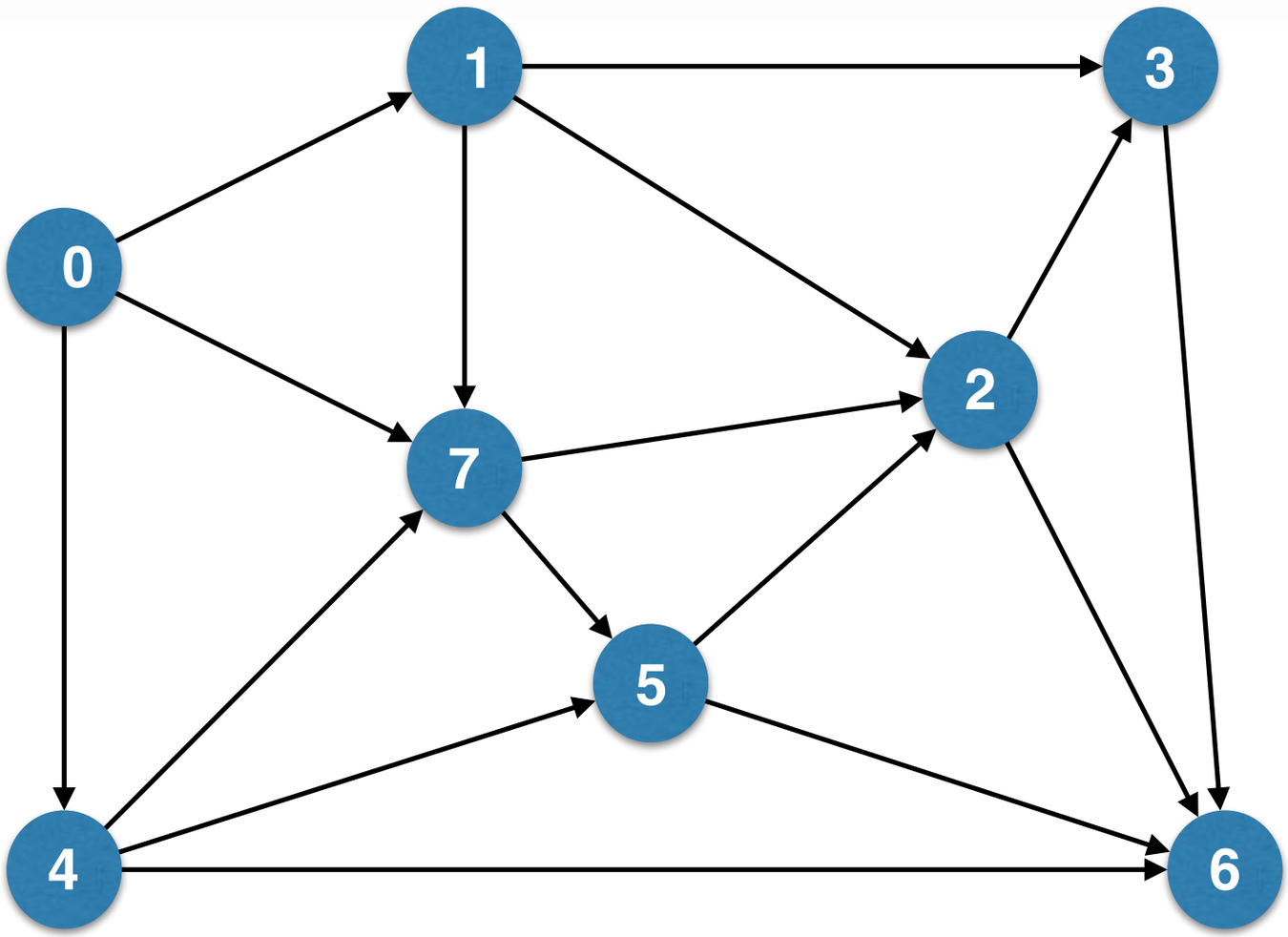
1. Inicialización



	$\delta(s,v)$	$\pi(v)$
v	distTo[v]	edge[v]
0	0	null
1	INFTY	null
2	INFTY	null
3	INFTY	null
4	INFTY	null
5	INFTY	null
6	INFTY	null
7	INFTY	null

Algoritmo de Dijkstra (Codicioso)

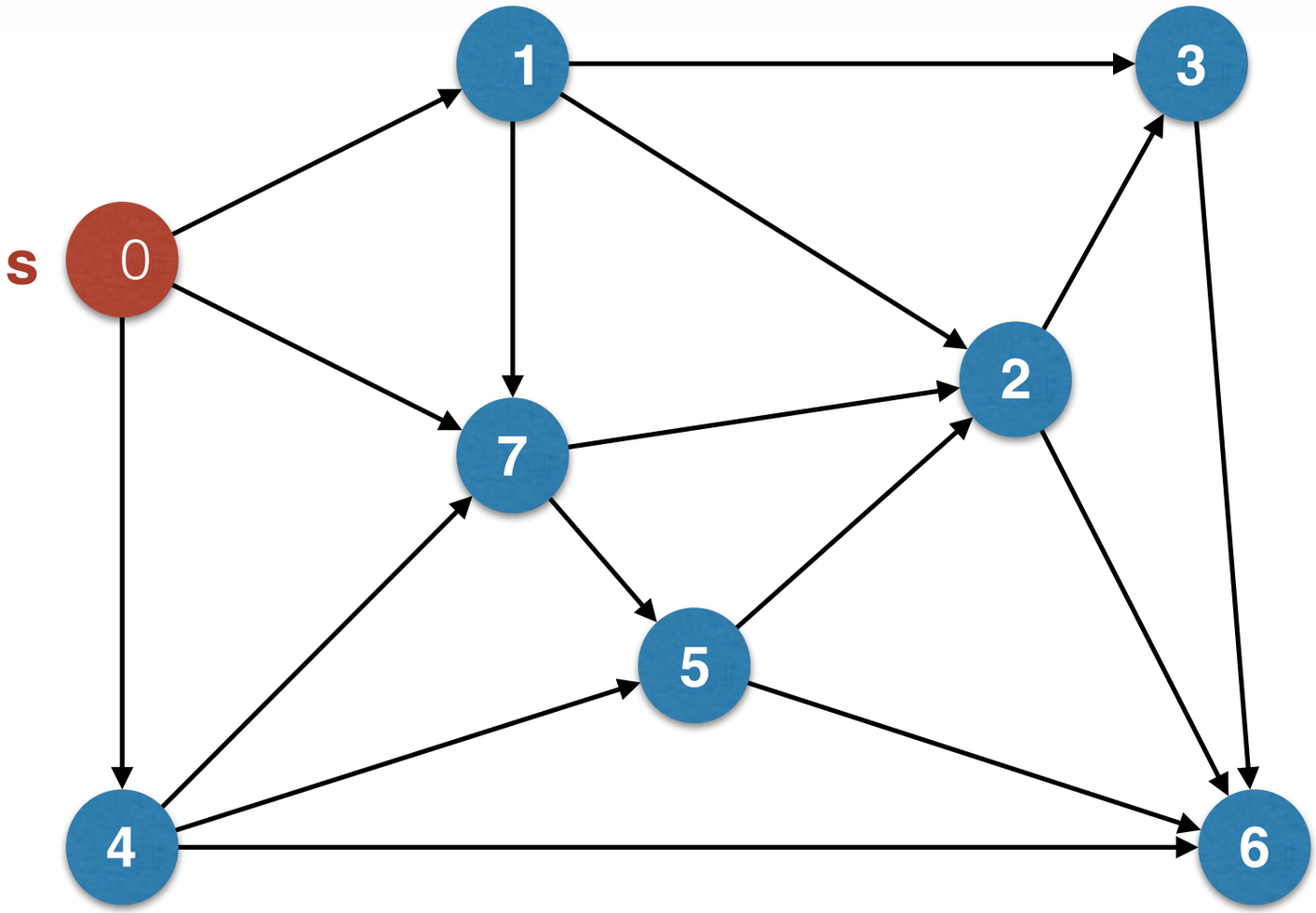
2. RELAX !!



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
0	0	null
1	INFTY	null
2	INFTY	null
3	INFTY	null
4	INFTY	null
5	INFTY	null
6	INFTY	null
7	INFTY	null

Algoritmo de Dijkstra (Codicioso)

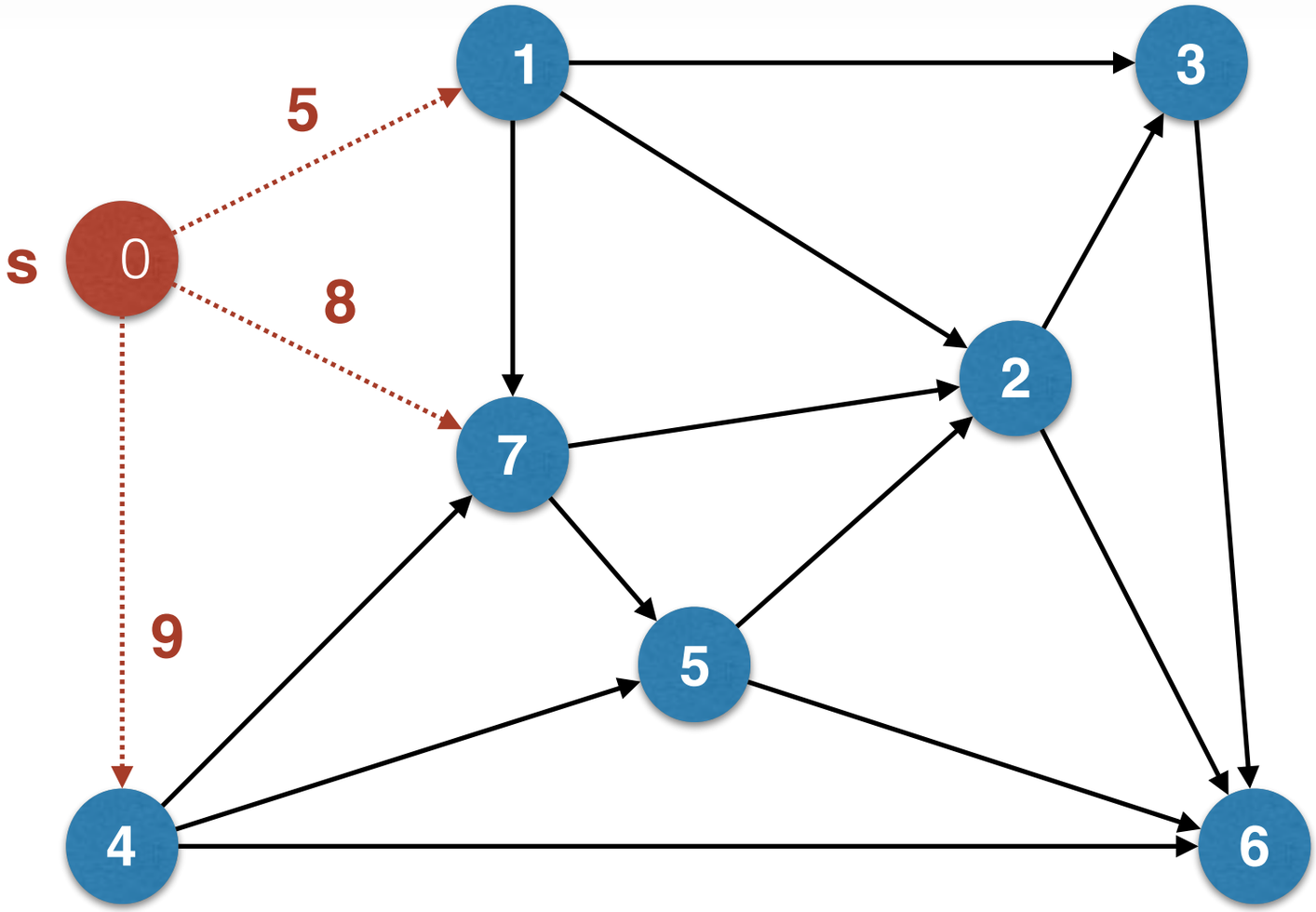
2. RELAX !!



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
0	0	null
1	INFTY	null
2	INFTY	null
3	INFTY	null
4	INFTY	null
5	INFTY	null
6	INFTY	null
7	INFTY	null

Algoritmo de Dijkstra (Codicioso)

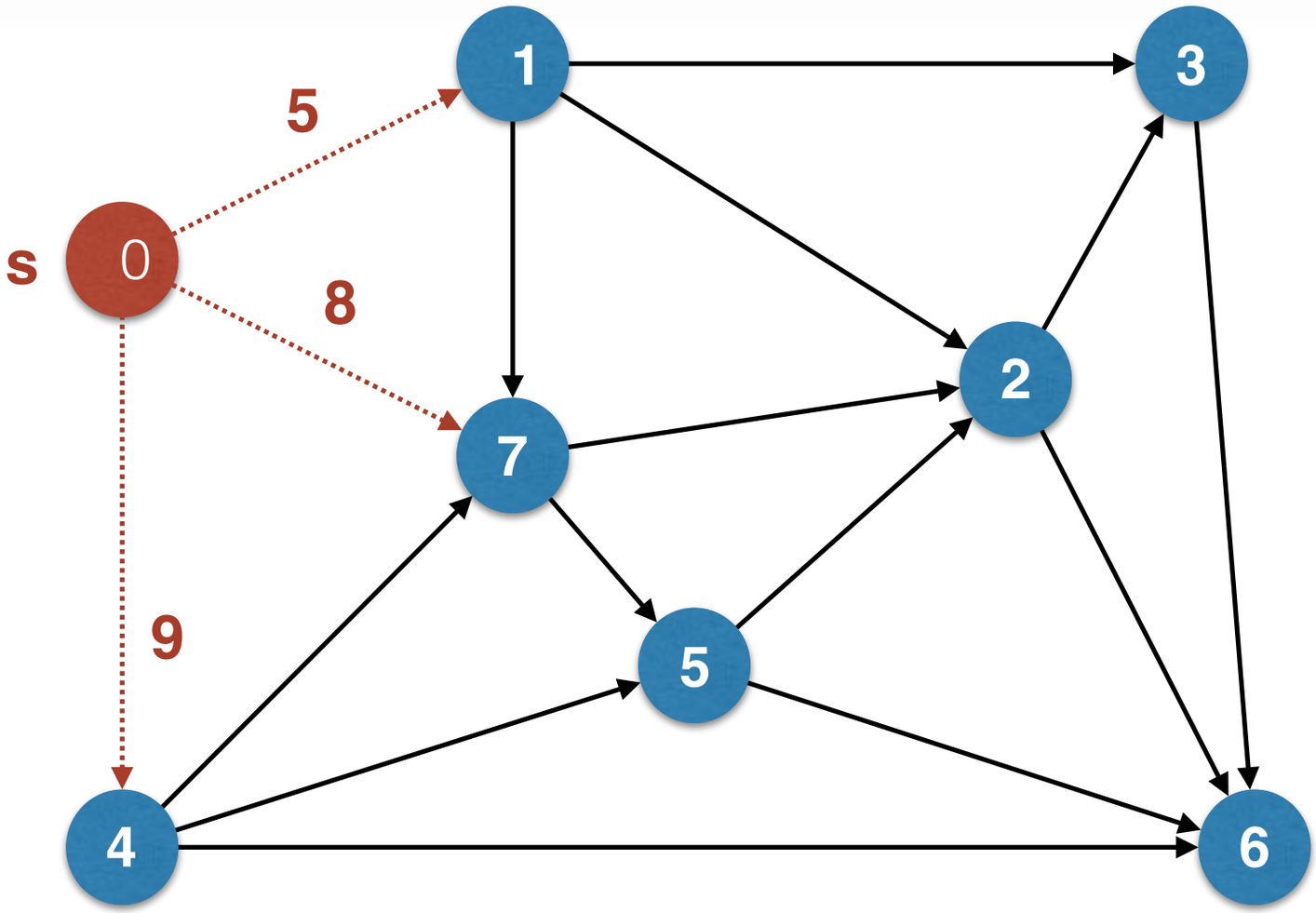
2. RELAX !! Aristas desde 0



v	distTo[v] $\delta(s,v)$	edge[v] $\pi(v)$
✖ 0	0	null
1	INFTY	null
2	INFTY	null
3	INFTY	null
4	INFTY	null
5	INFTY	null
6	INFTY	null
7	INFTY	null

Algoritmo de Dijkstra (Codicioso)

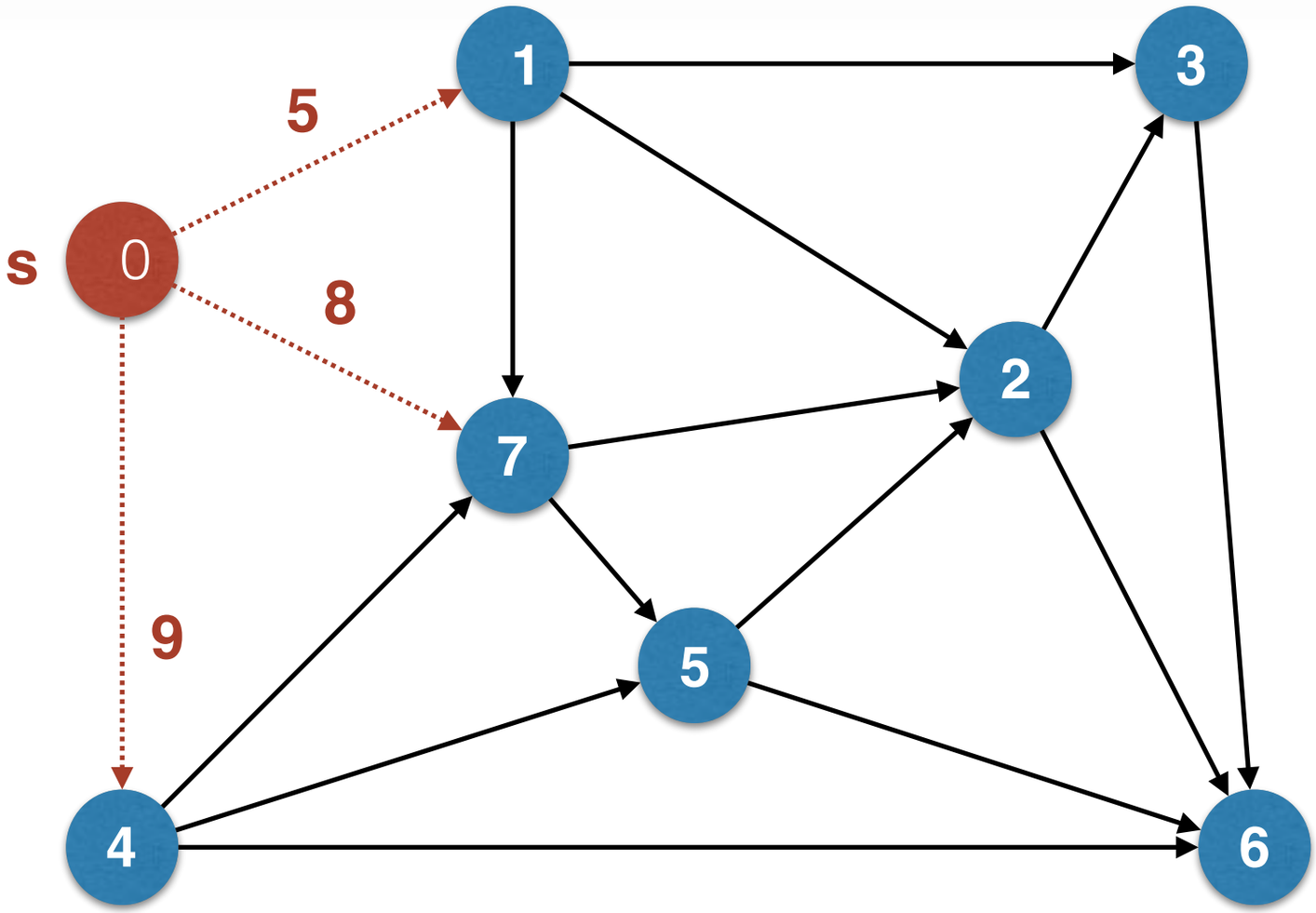
2. RELAX !! Aristas desde 0



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✖ 0	0	null
1	5	0->1
2	INFTY	null
3	INFTY	null
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

Algoritmo de Dijkstra (Codicioso)

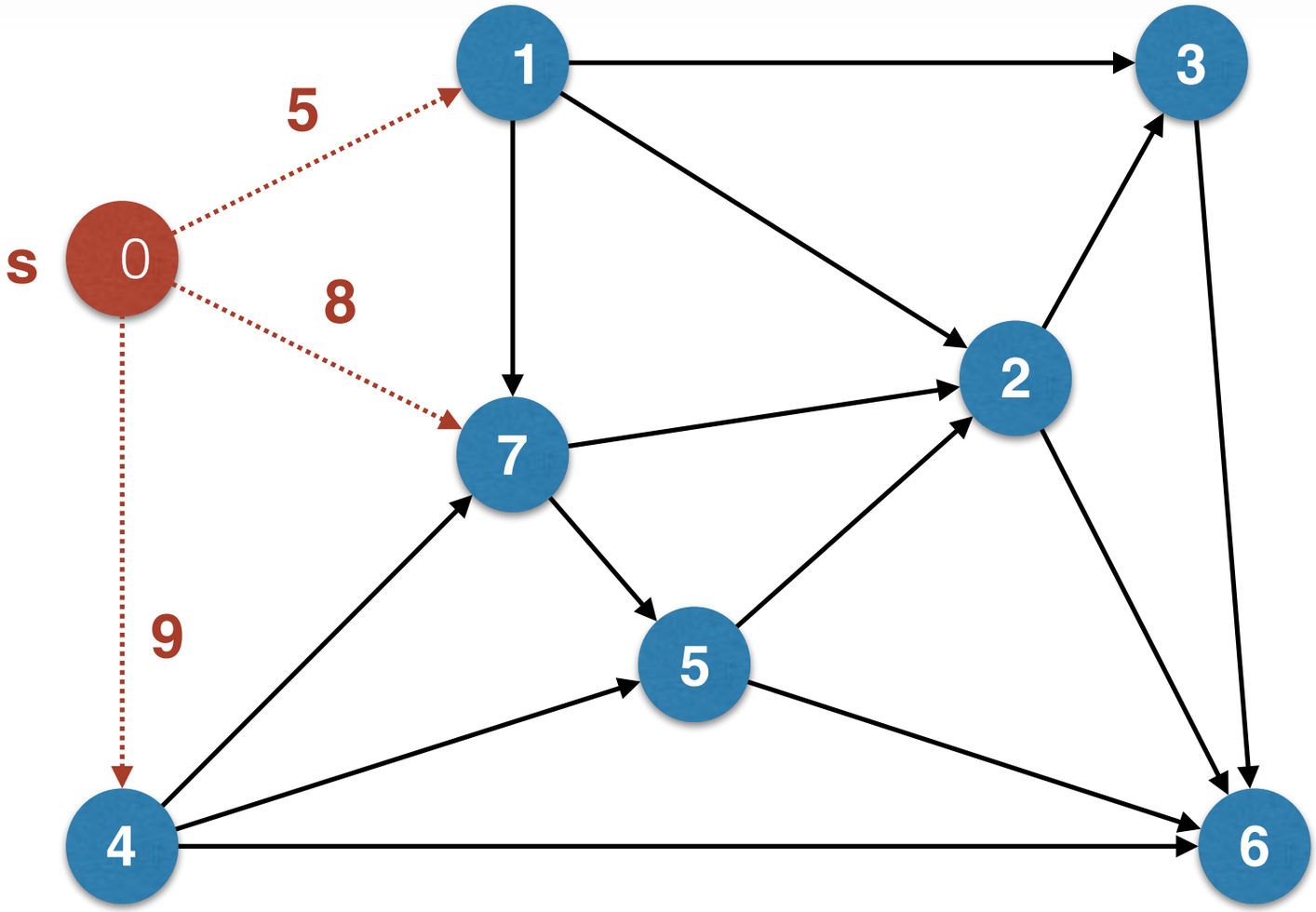
3. RELAX !!



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✖ 0	0	null
1	5	0->1
2	INFTY	null
3	INFTY	null
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

Algoritmo de Dijkstra (Codicioso)

3. RELAX !!



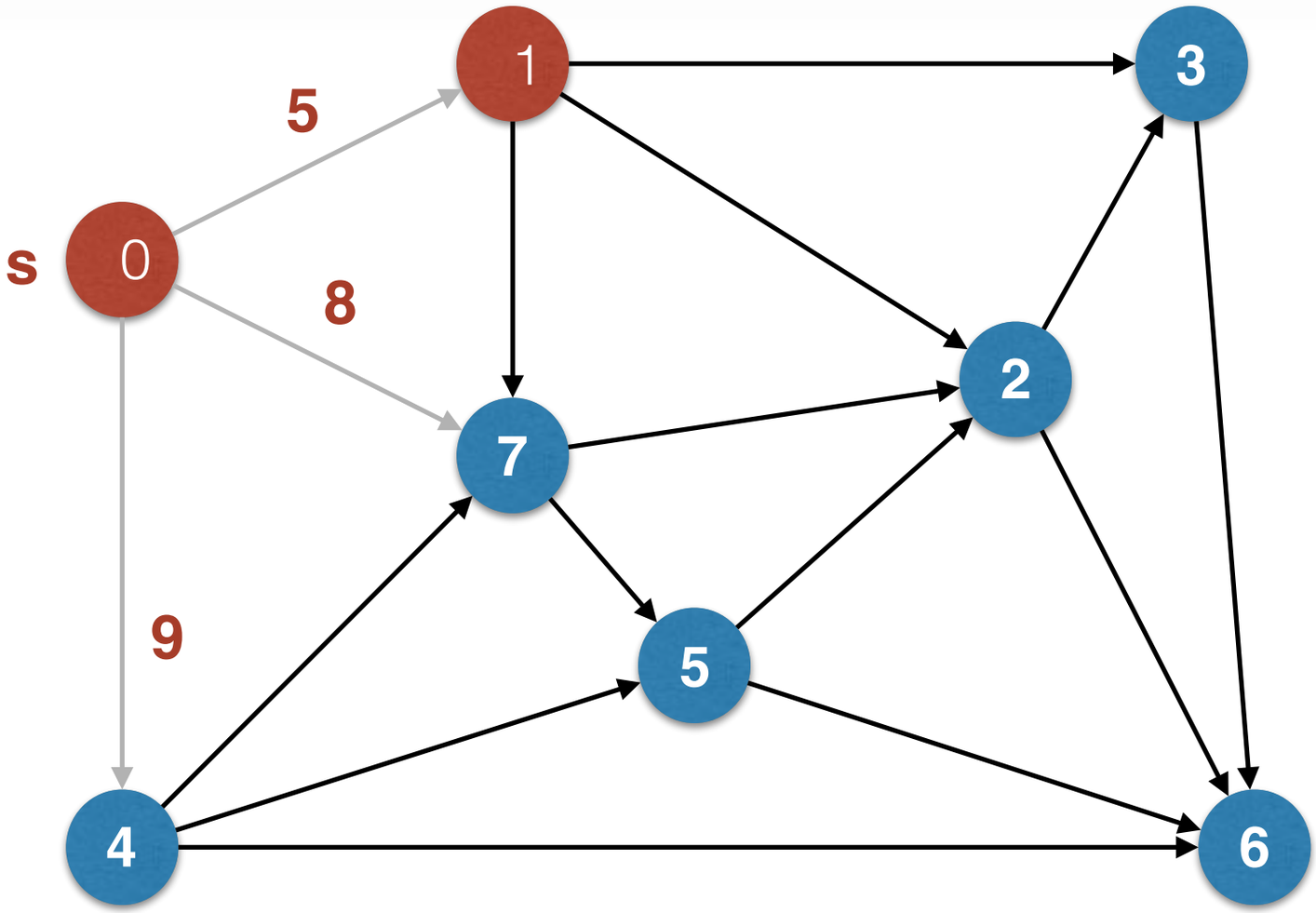
v	distTo[v]	edge[v]
✖ 0	0	null
1	5	0->1
2	INFTY	null
3	INFTY	null
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

$\delta(s, v)$ $\pi(v)$

(Seleccione el vértice con min costo desde s)

Algoritmo de Dijkstra (Codicioso)

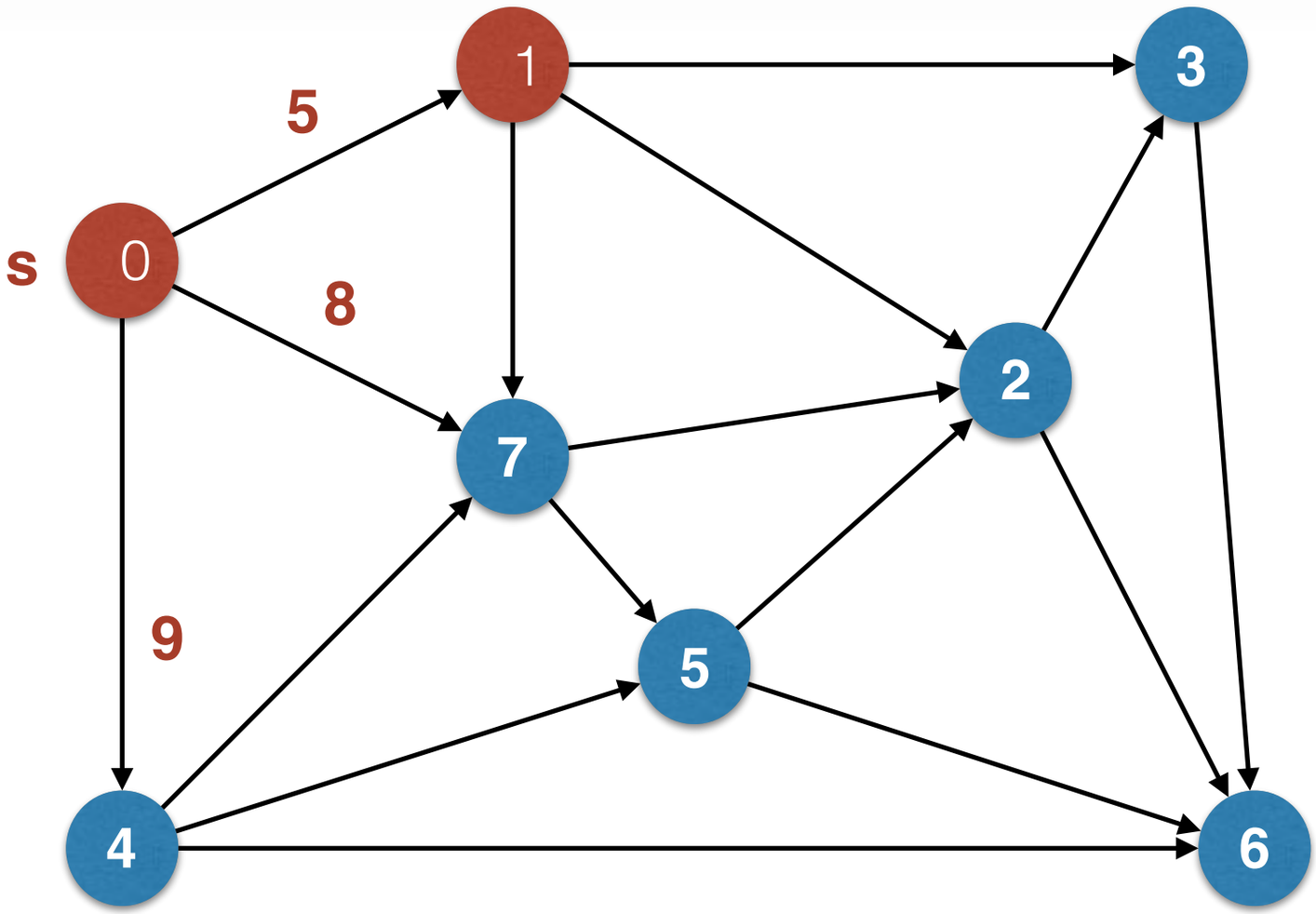
3. RELAX !!



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
0	0	null
1	5	0->1
2	INFTY	null
3	INFTY	null
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

Algoritmo de Dijkstra (Codicioso)

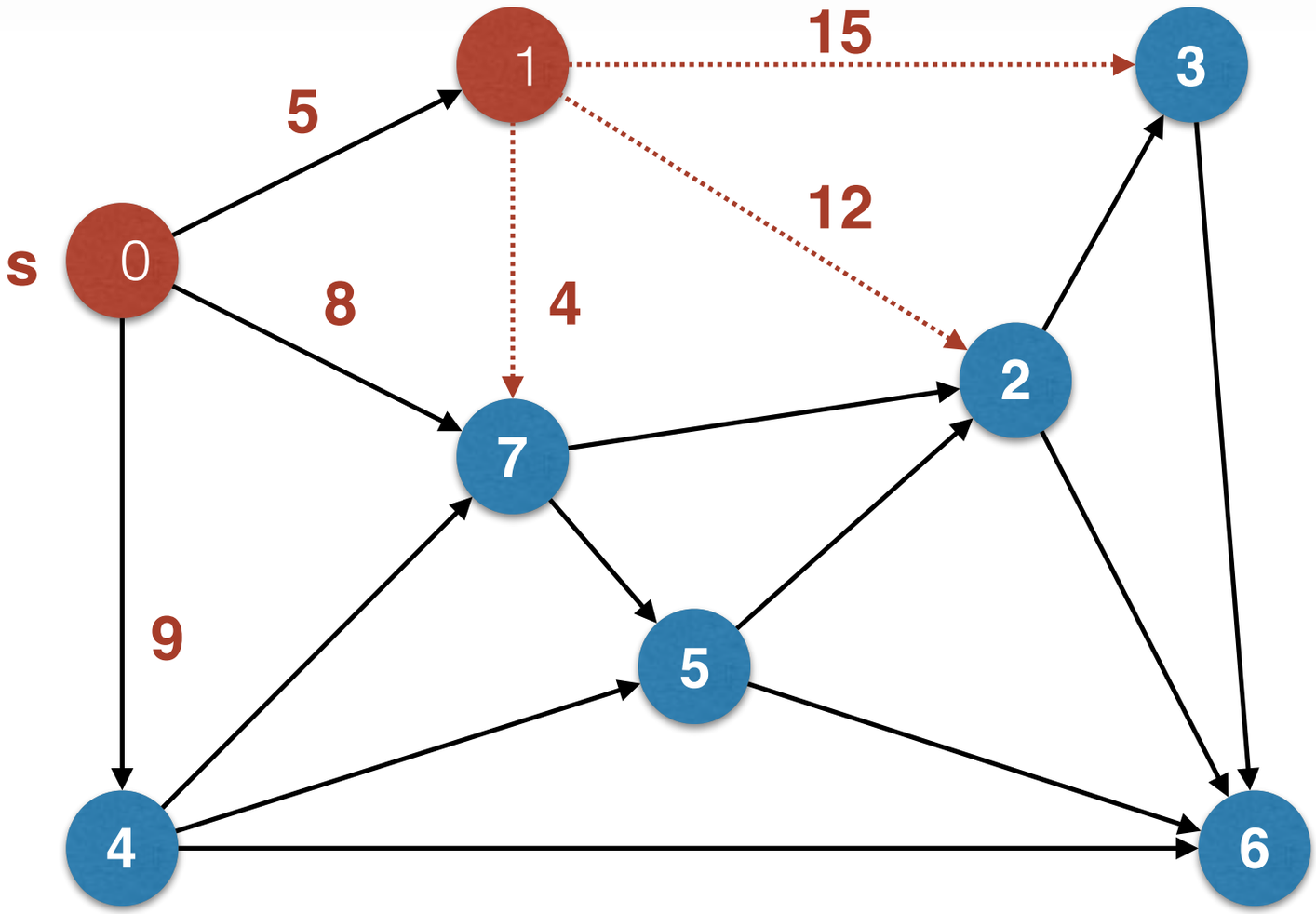
3. RELAX !! Aristas desde 1



	$\delta(s,v)$	$\pi(v)$
v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	INFTY	null
3	INFTY	null
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

Algoritmo de Dijkstra (Codicioso)

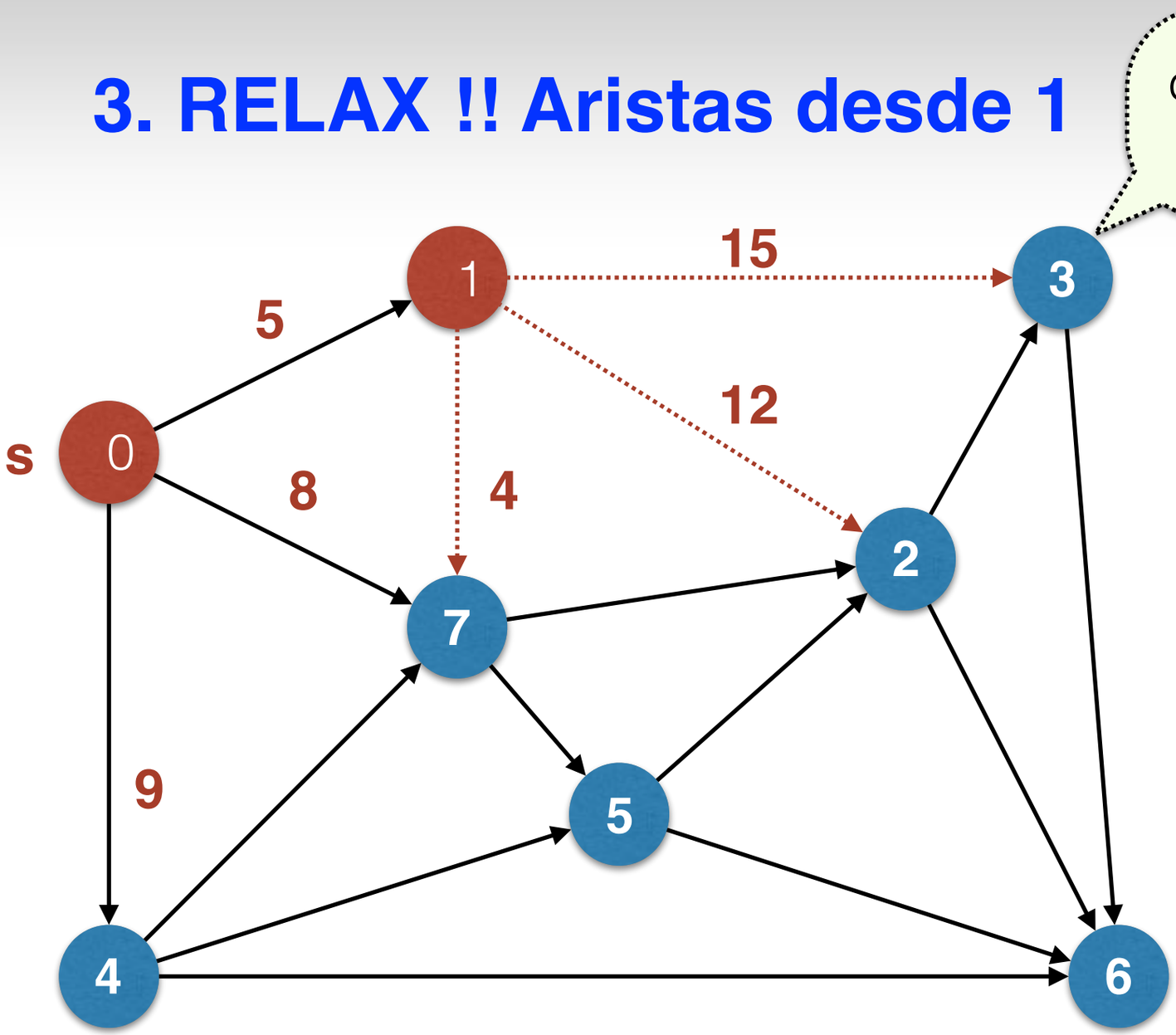
3. RELAX !! Aristas desde 1



v	distTo[v] $\delta(s,v)$	edge[v] $\pi(v)$
✗ 0	0	null
✗ 1	5	0->1
2	INFTY	null
3	INFTY	null
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

Algoritmo de Dijkstra (Codicioso)

3. RELAX !! Aristas desde 1

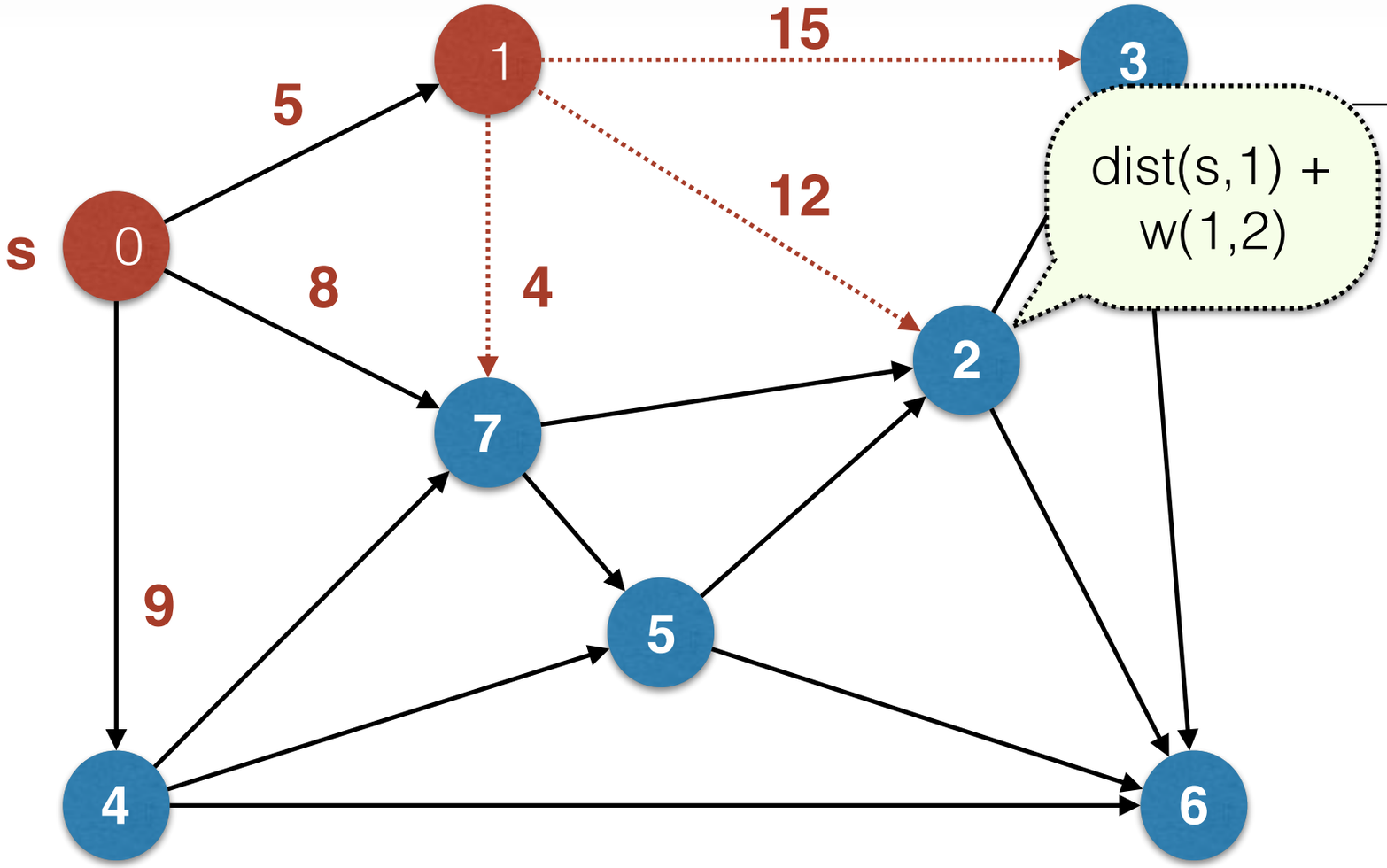


v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	INFTY	null
3	INFTY	null
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

$\delta(s, v)$ $\pi(v)$

Algoritmo de Dijkstra (Codicioso)

3. RELAX !! Aristas desde 1

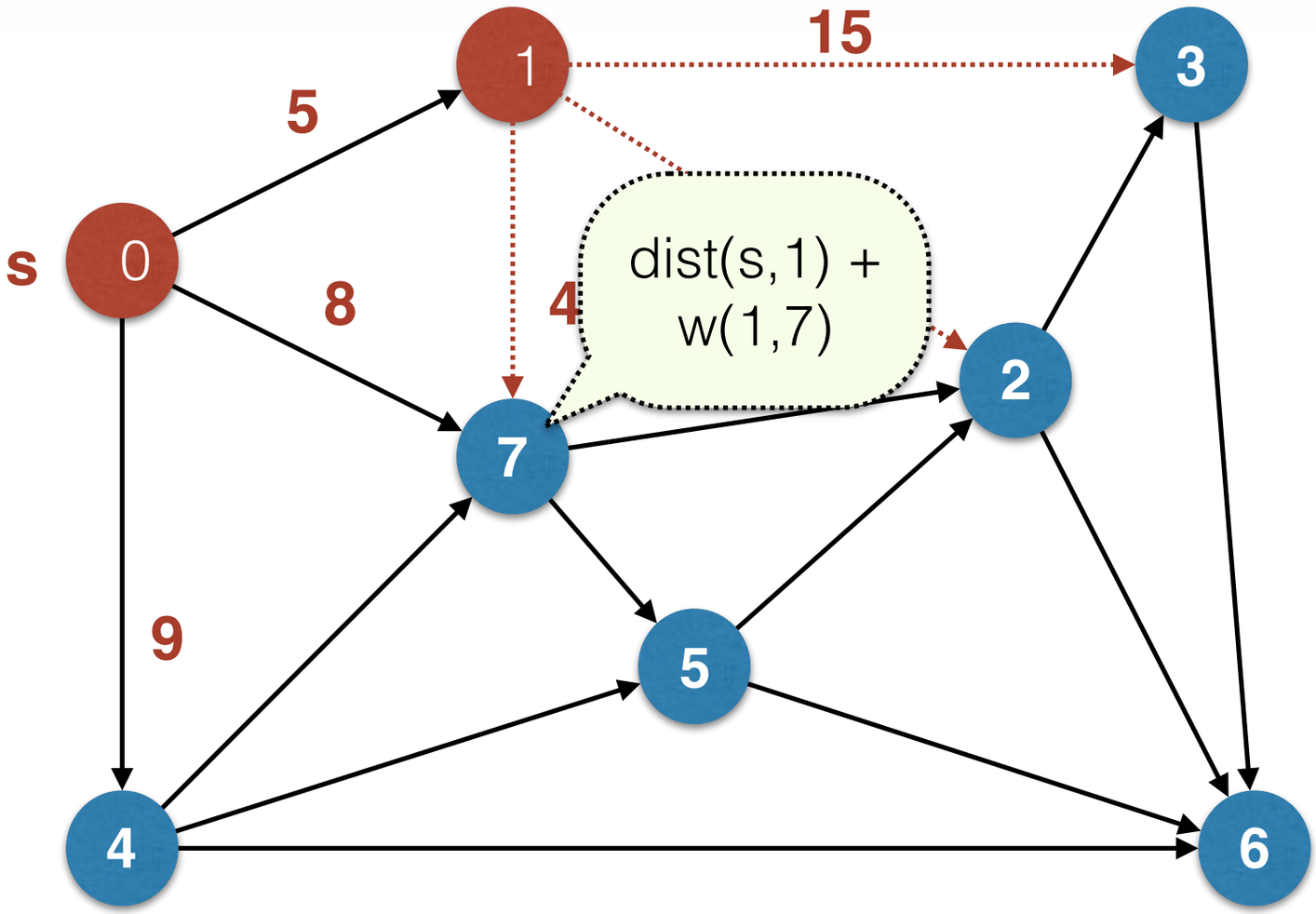


v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	INFTY	null
3	INFTY	null
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

$\delta(s,v)$ $\pi(v)$

Algoritmo de Dijkstra (Codicioso)

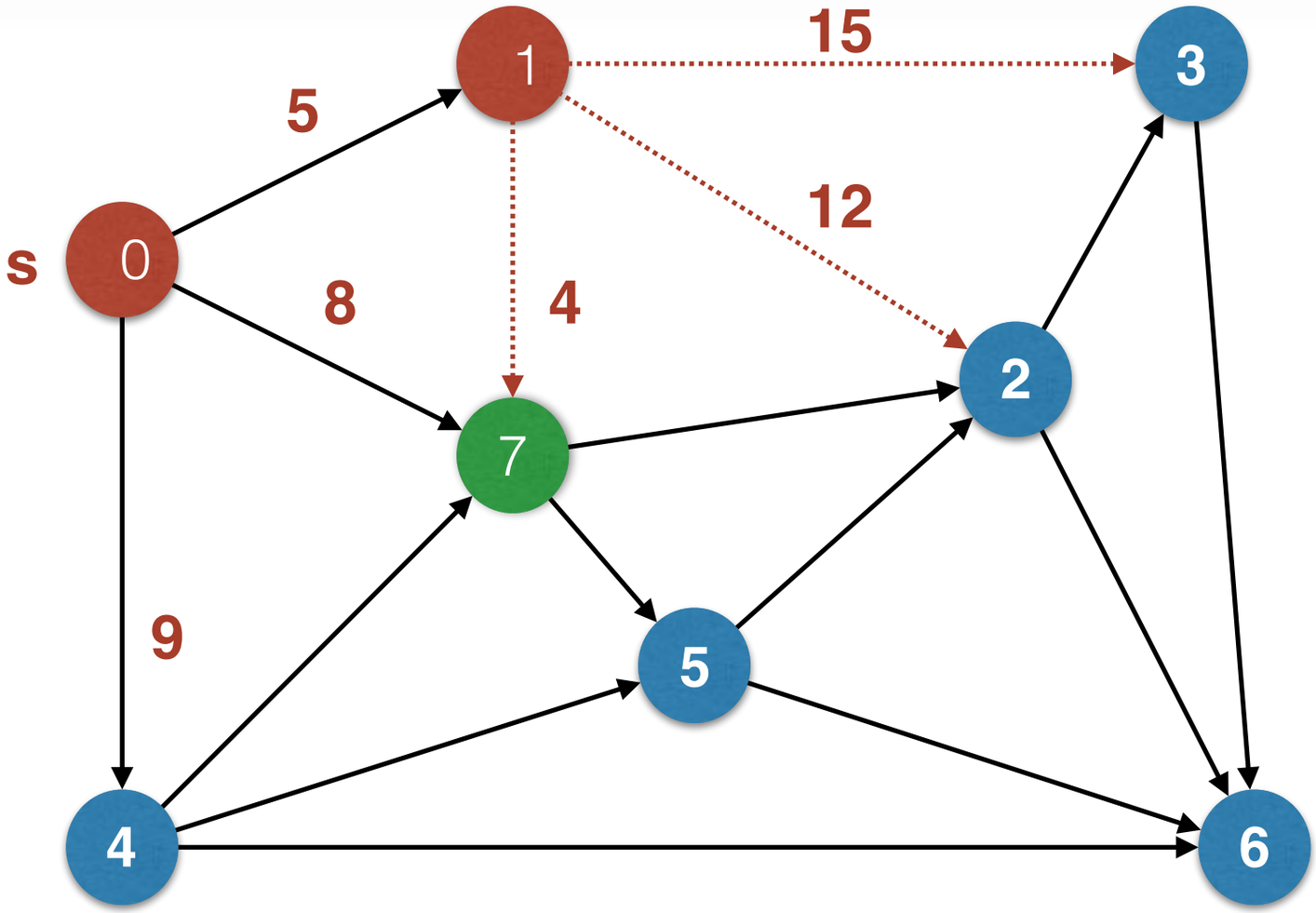
3. RELAX !! Aristas desde 1



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✖ 0	0	null
✖ 1	5	0->1
2	INFTY	null
3	INFTY	null
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

Algoritmo de Dijkstra (Codicioso)

3. RELAX !! Aristas desde 1

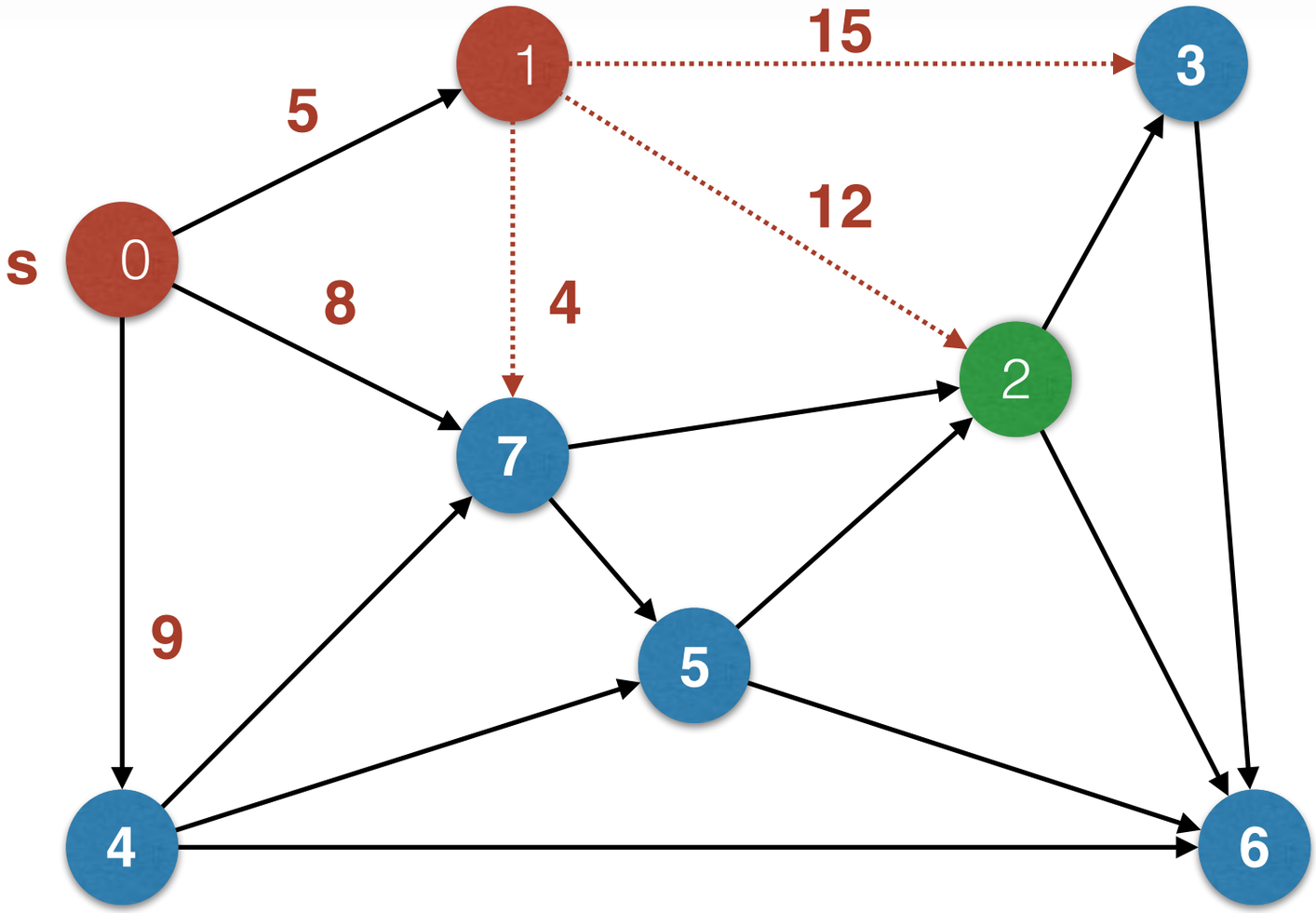


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	INFTY	null
3	INFTY	null
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

5 + 4 < 8 ?

Algoritmo de Dijkstra (Codicioso)

3. RELAX !! Aristas desde 1

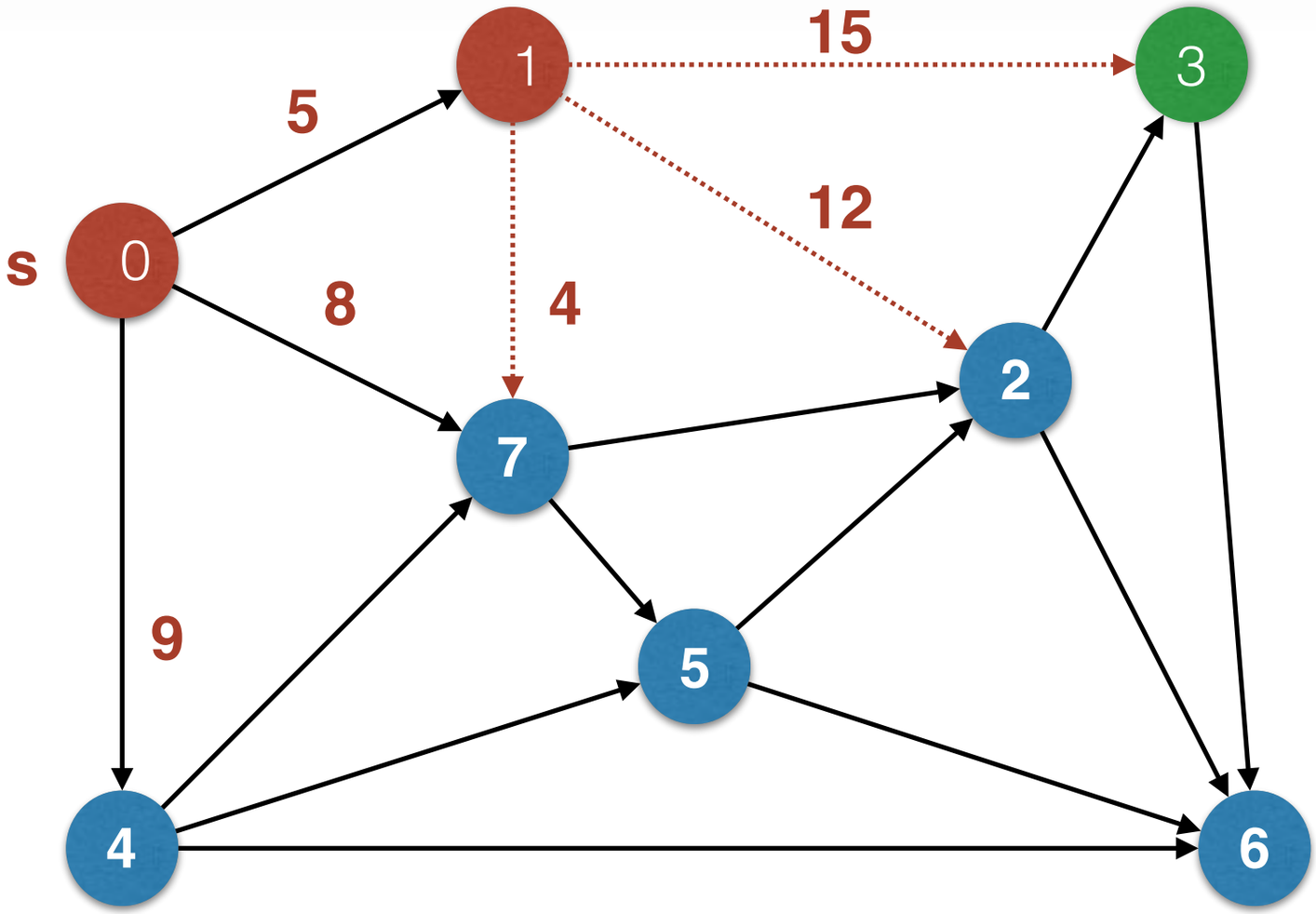


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	INFTY	null
3	INFTY	null
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

5 + 12 < INFTY ?

Algoritmo de Dijkstra (Codicioso)

3. RELAX !! Aristas desde 1

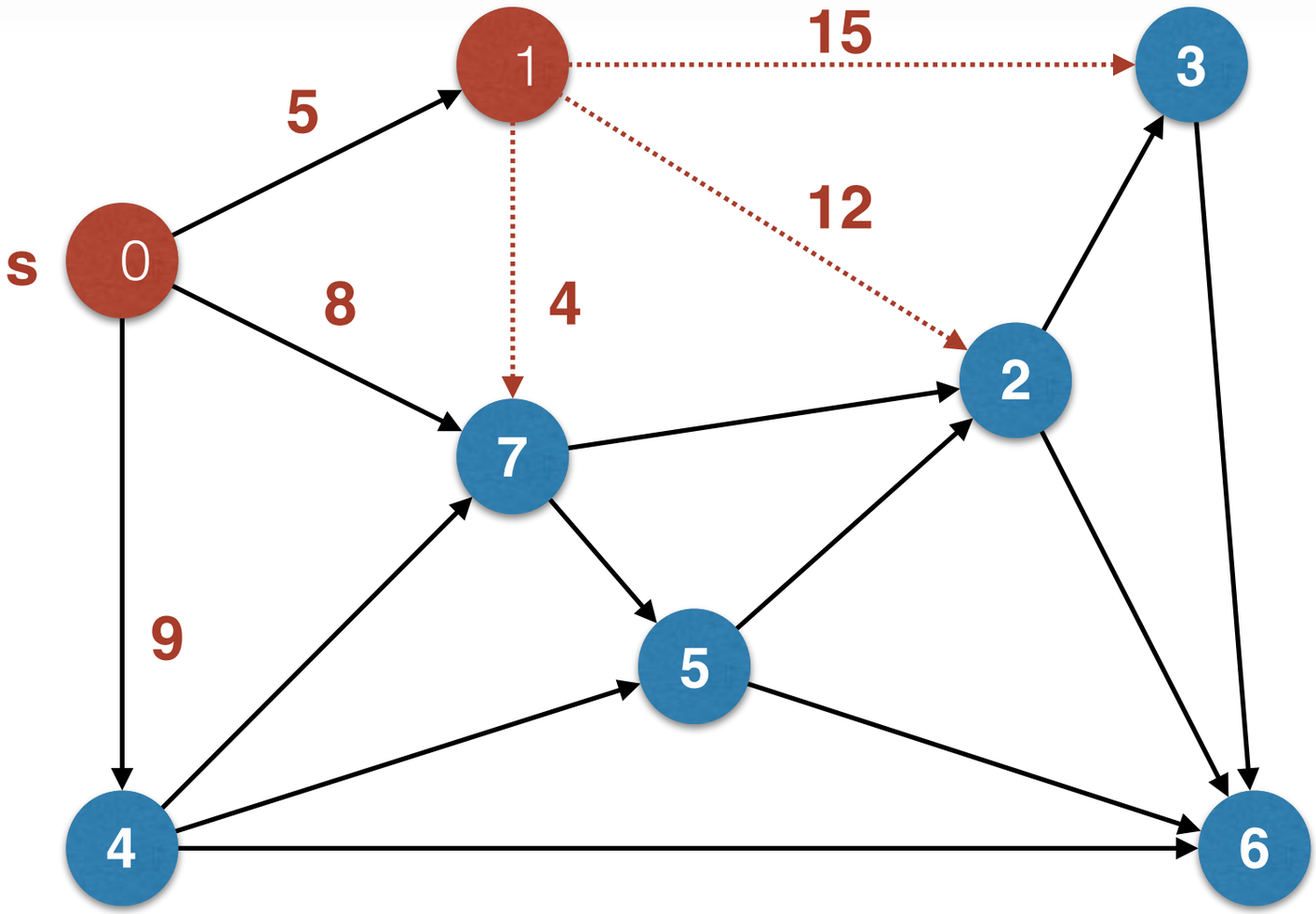


	$\delta(s,v)$	$\pi(v)$
v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	17	1->2
3	INFTY	null
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

5 + 15 < INFTY ?

Algoritmo de Dijkstra (Codicioso)

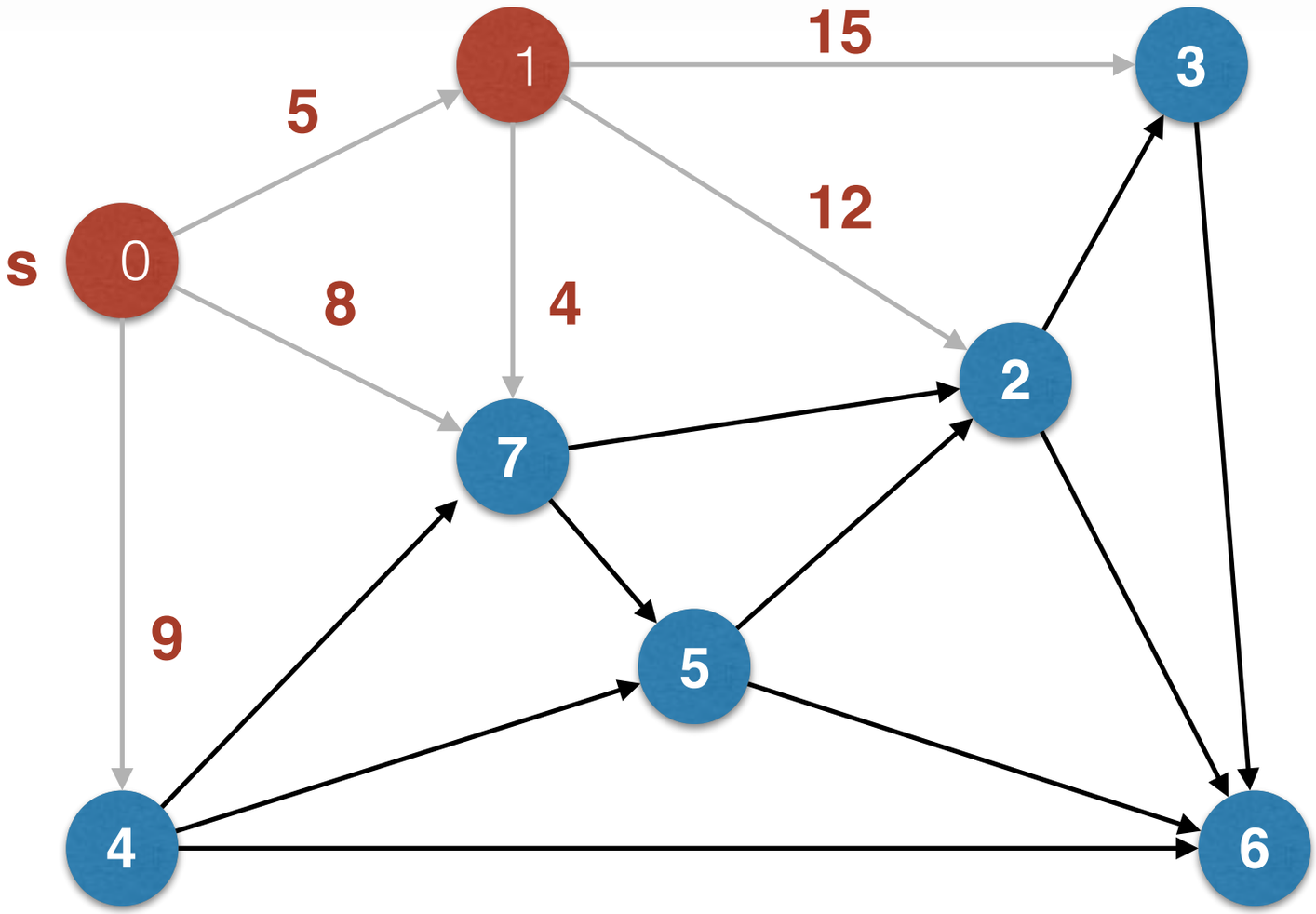
3. RELAX !! Aristas desde 1



v	distTo[v] $\delta(s,v)$	edge[v] $\pi(v)$
✗ 0	0	null
✗ 1	5	0->1
2	17	1->2
3	20	1->3
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

Algoritmo de Dijkstra (Codicioso)

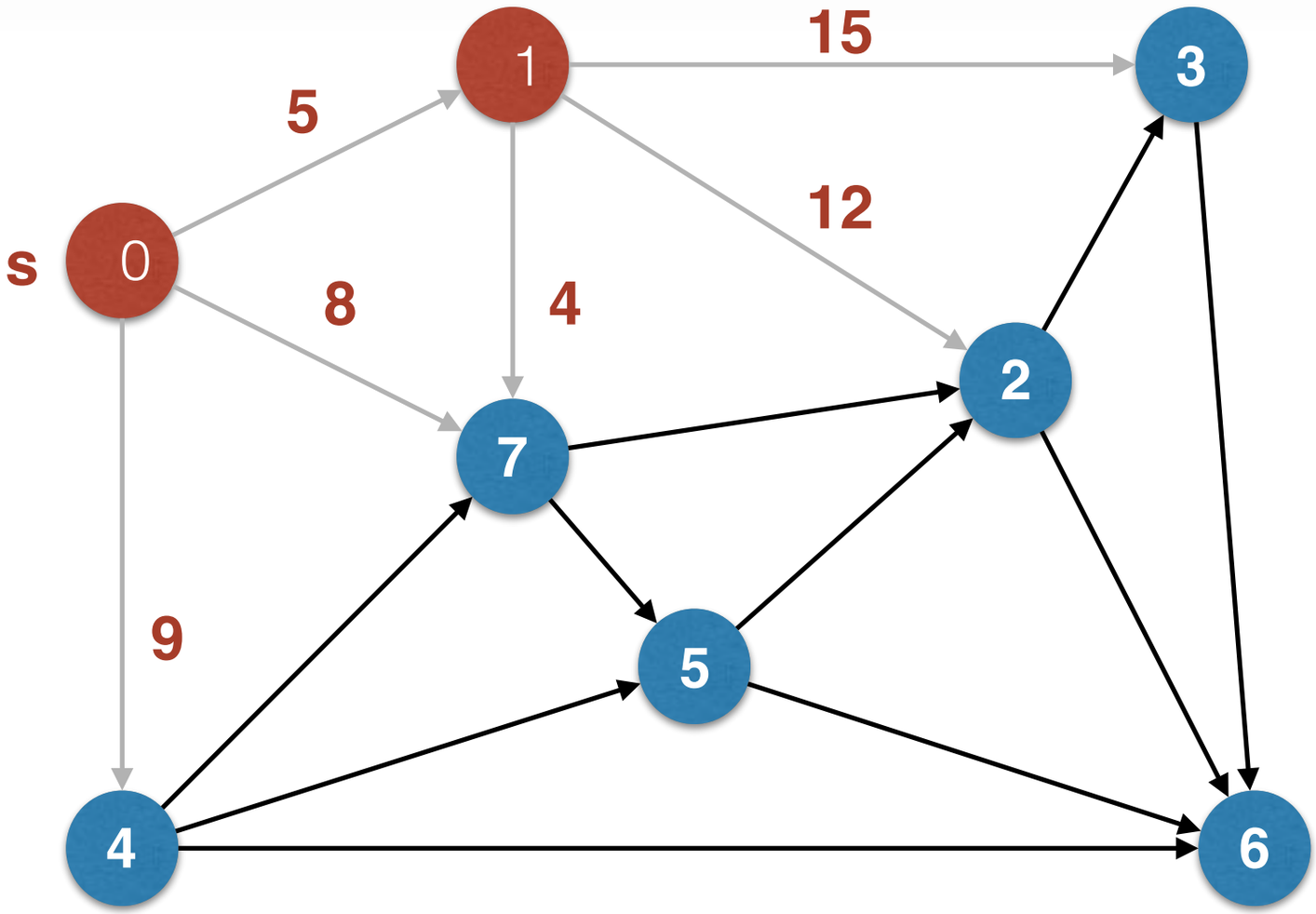
4. RELAX !!



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	17	1->2
3	20	1->3
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

Algoritmo de Dijkstra (Codicioso)

4. RELAX !!

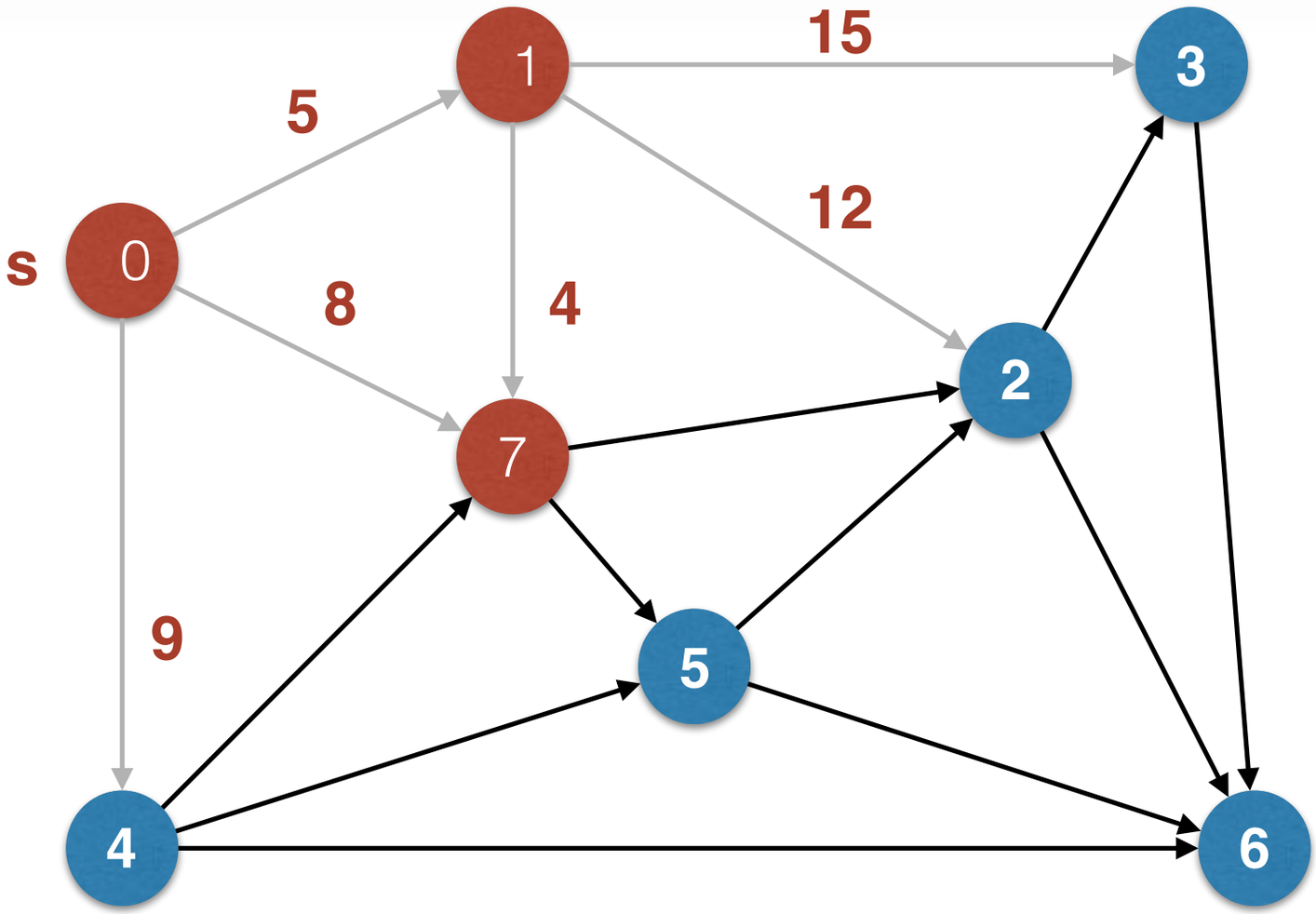


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	17	1->2
3	20	1->3
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

(Seleccione el vértice con min costo desde s)

Algoritmo de Dijkstra (Codicioso)

4. RELAX !!

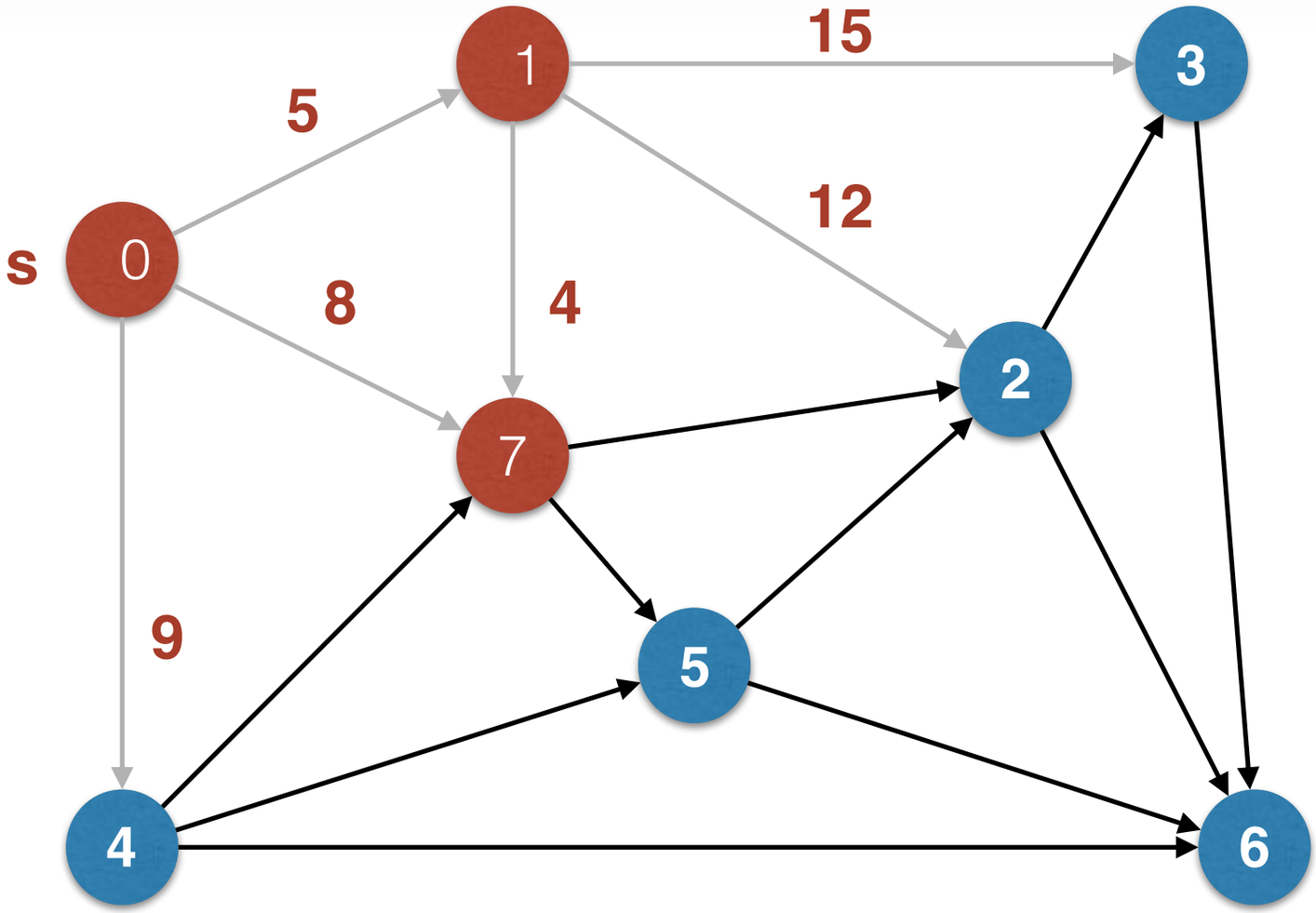


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	17	1->2
3	20	1->3
4	9	0->4
5	INFTY	null
6	INFTY	null
7	8	0->7

(Seleccione el vértice con min costo desde s)

Algoritmo de Dijkstra (Codicioso)

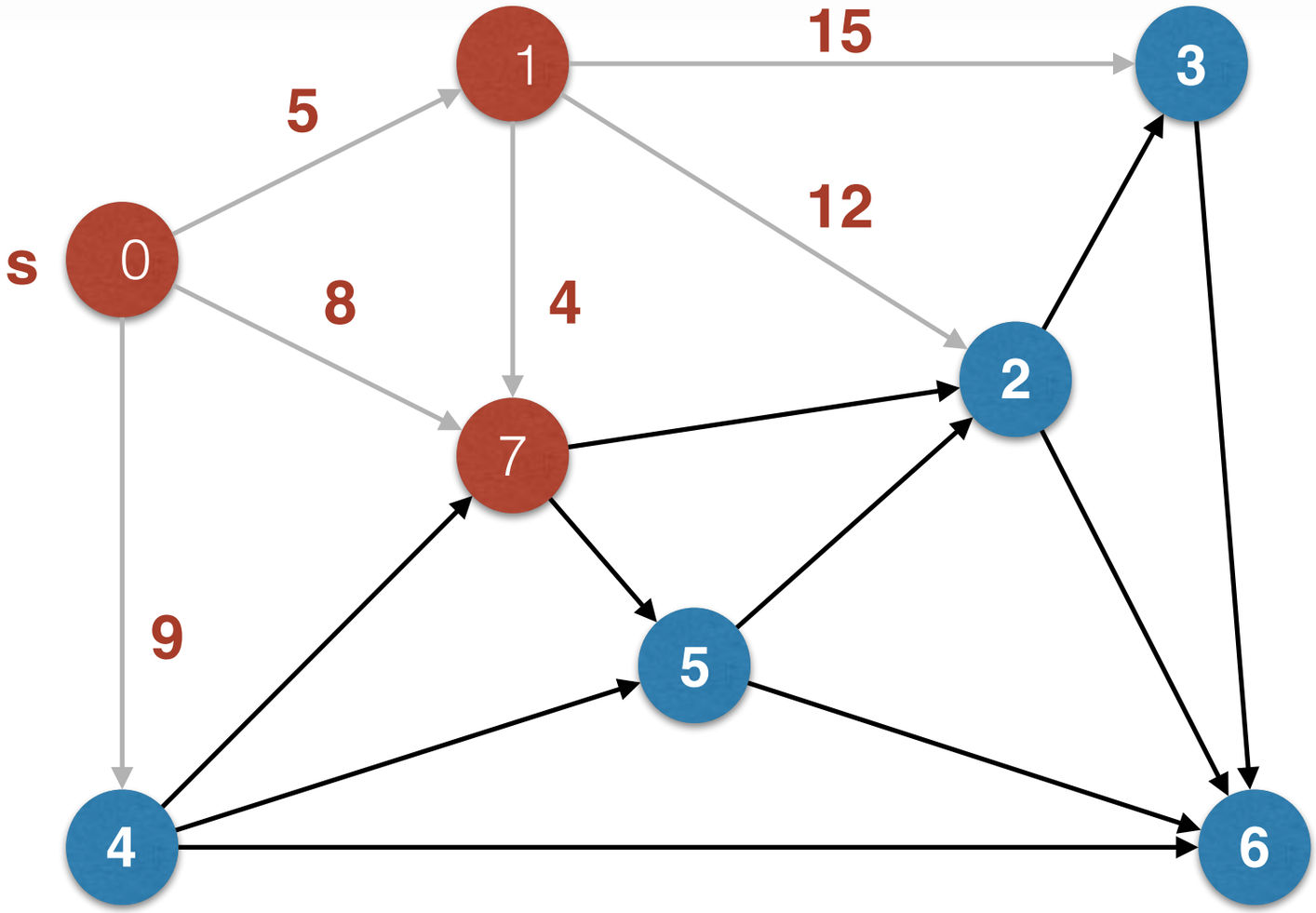
4. RELAX !!



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	17	1->2
3	20	1->3
4	9	0->4
5	INFTY	null
6	INFTY	null
✗ 7	8	0->7

Algoritmo de Dijkstra (Codicioso)

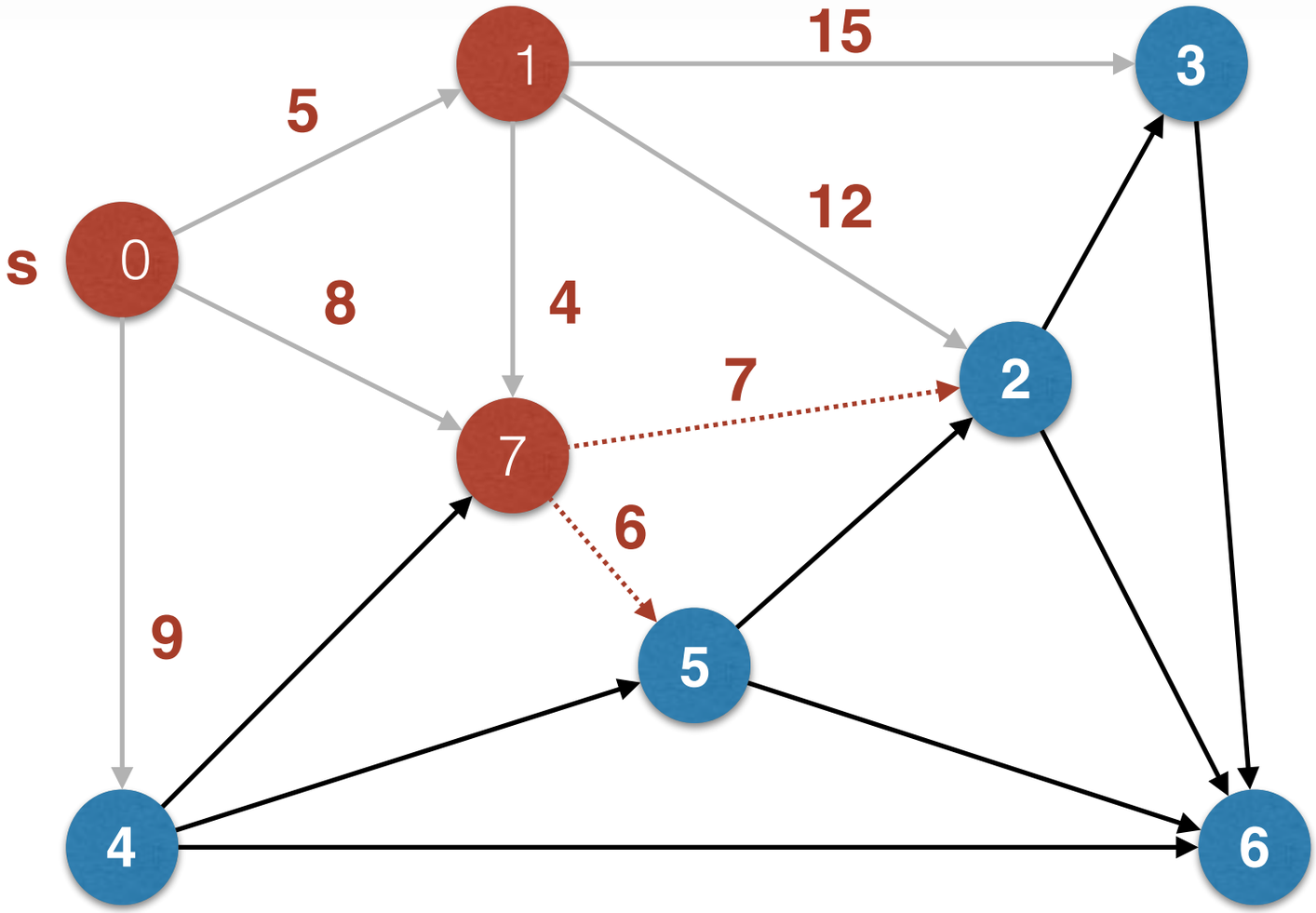
4. RELAX !! Aristas desde 7



	v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗	0	0	null
✗	1	5	0->1
	2	17	1->2
	3	20	1->3
	4	9	0->4
	5	INFTY	null
	6	INFTY	null
✗	7	8	0->7

Algoritmo de Dijkstra (Codicioso)

4. RELAX !! Aristas desde 7

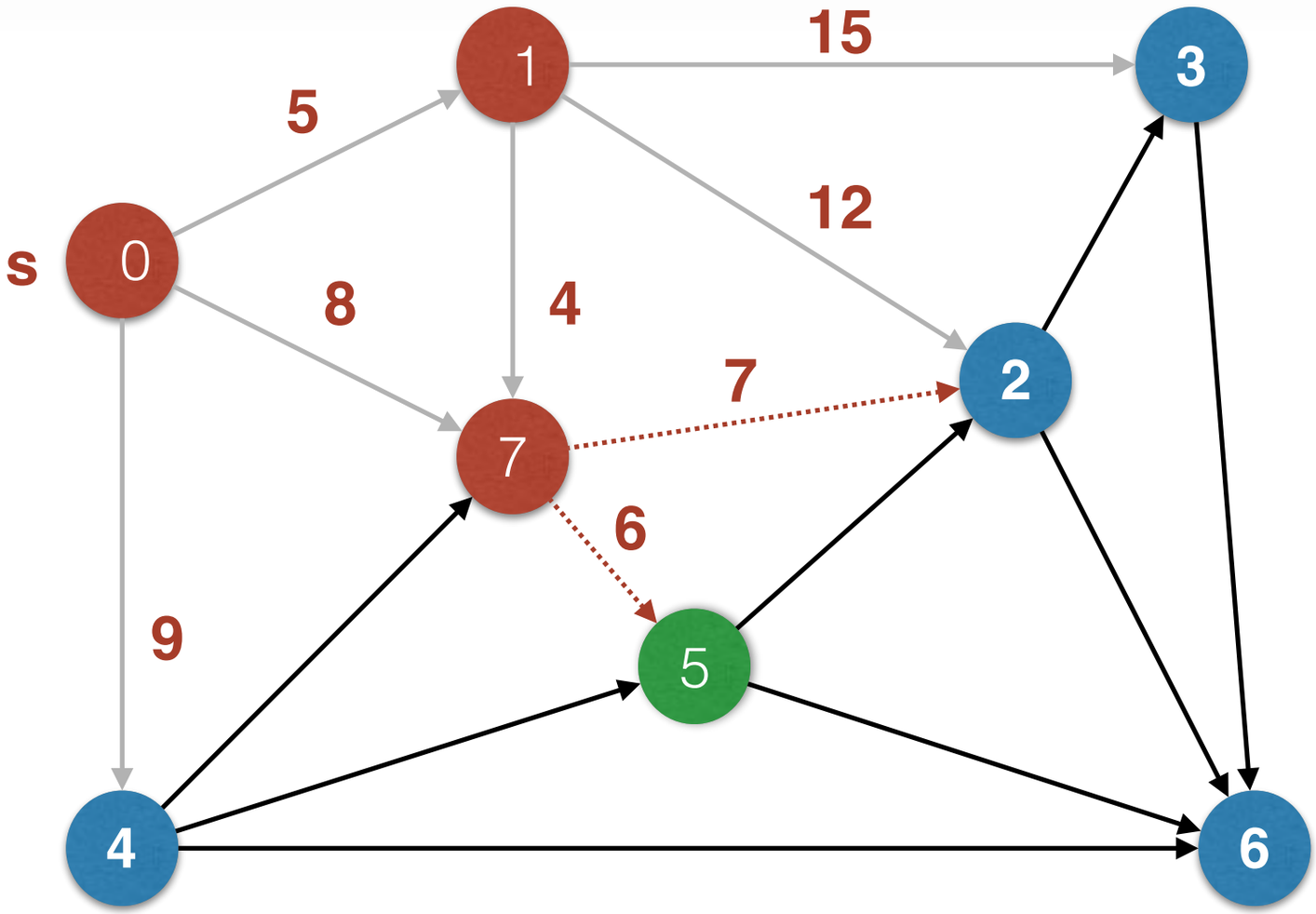


v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	17	1->2
3	20	1->3
4	9	0->4
5	INFTY	null
6	INFTY	null
✗ 7	8	0->7

$\delta(s, v)$ $\pi(v)$

Algoritmo de Dijkstra (Codicioso)

4. RELAX !!

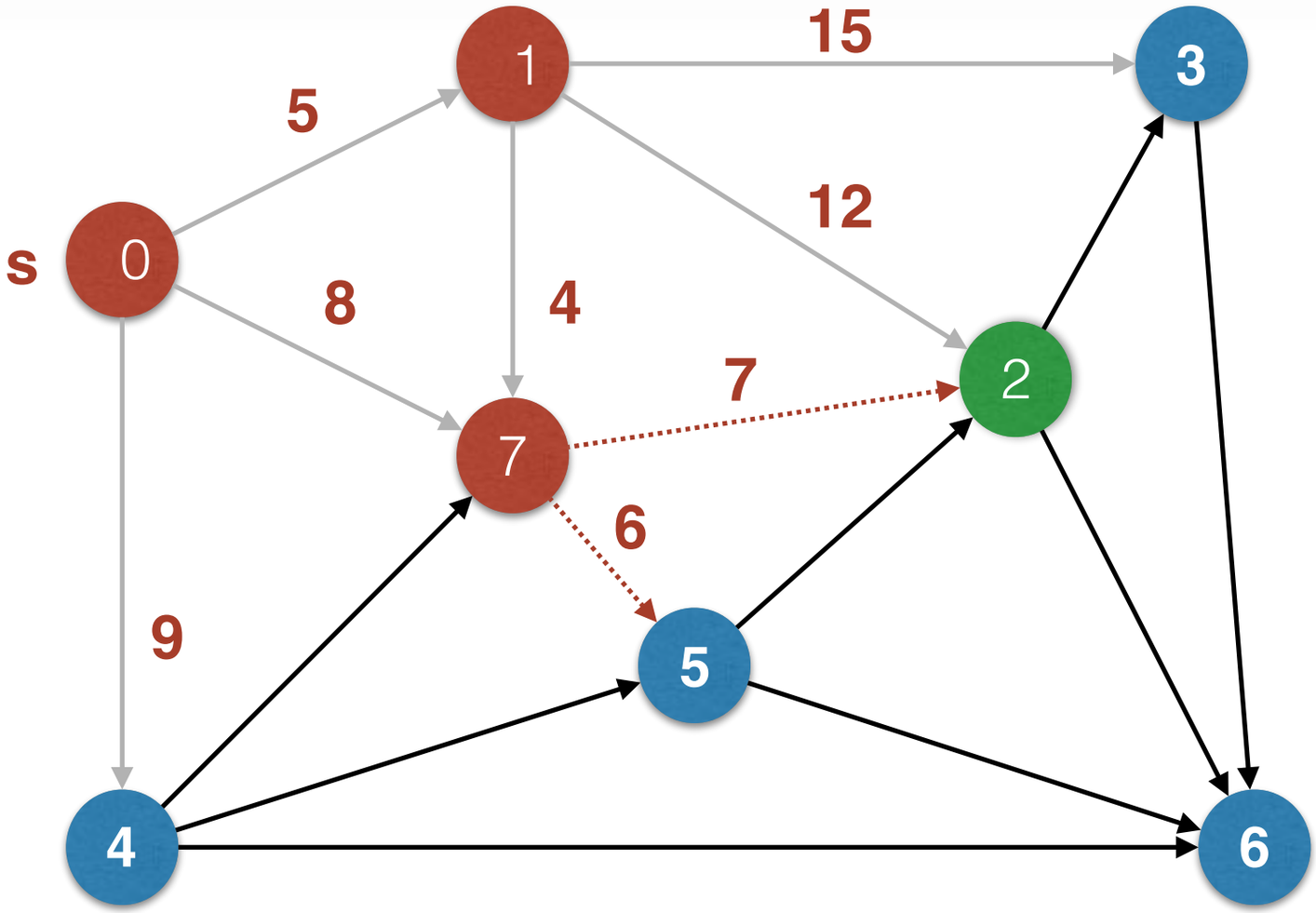


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	17	1->2
3	20	1->3
4	9	0->4
5	INFTY	null
6	INFTY	null
✗ 7	8	0->7

8 + 6 < INFTY ?

Algoritmo de Dijkstra (Codicioso)

4. RELAX !!

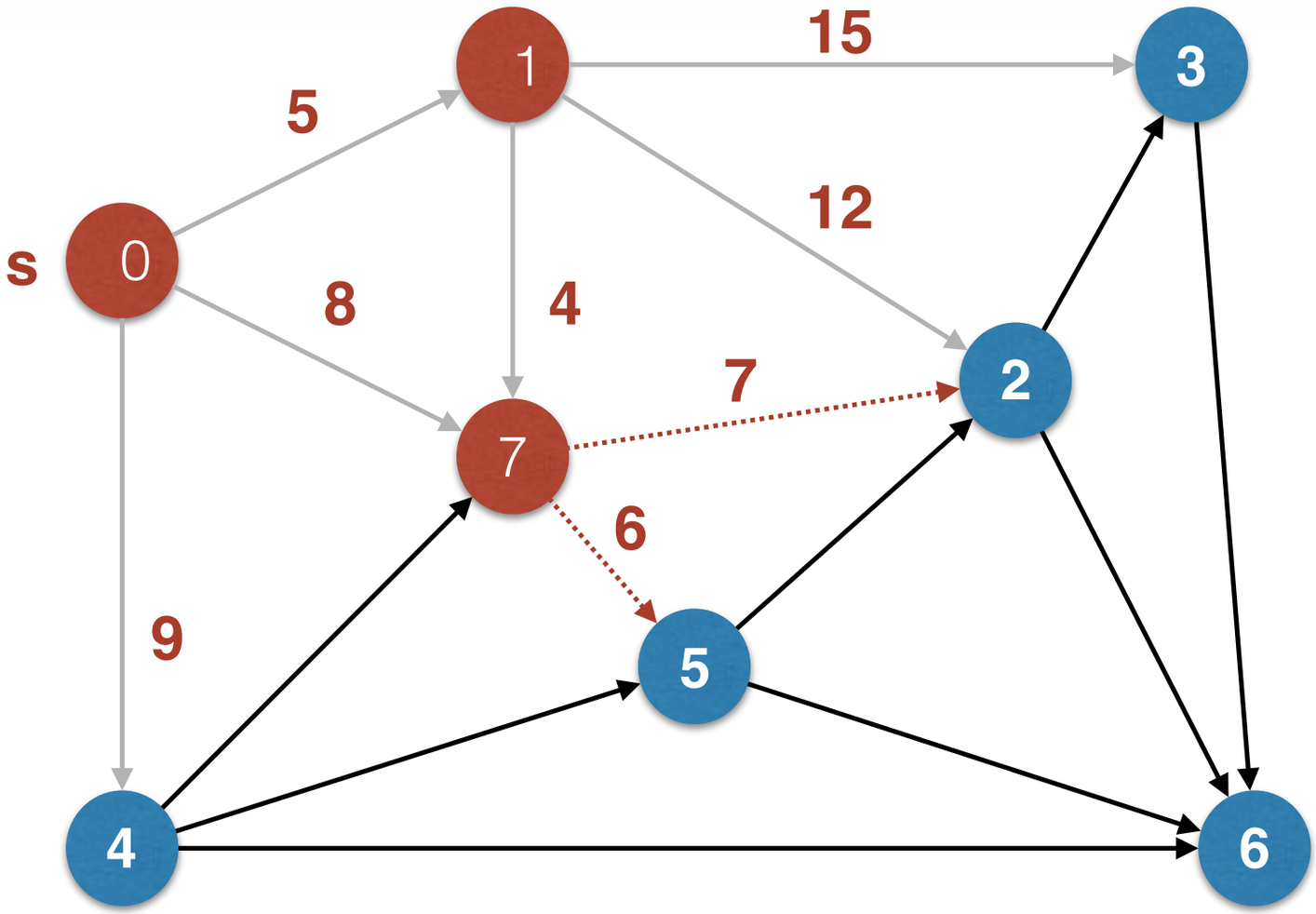


v	distTo[v]	edge[v]	$\delta(s,v)$ / $\pi(v)$
✗ 0	0	null	
✗ 1	5	0->1	
2	17	1->2	
3	20	1->3	
4	9	0->4	
5	14	7->5	
6	INFTY	null	
✗ 7	8	0->7	

8 + 7 < 17 ?

Algoritmo de Dijkstra (Codicioso)

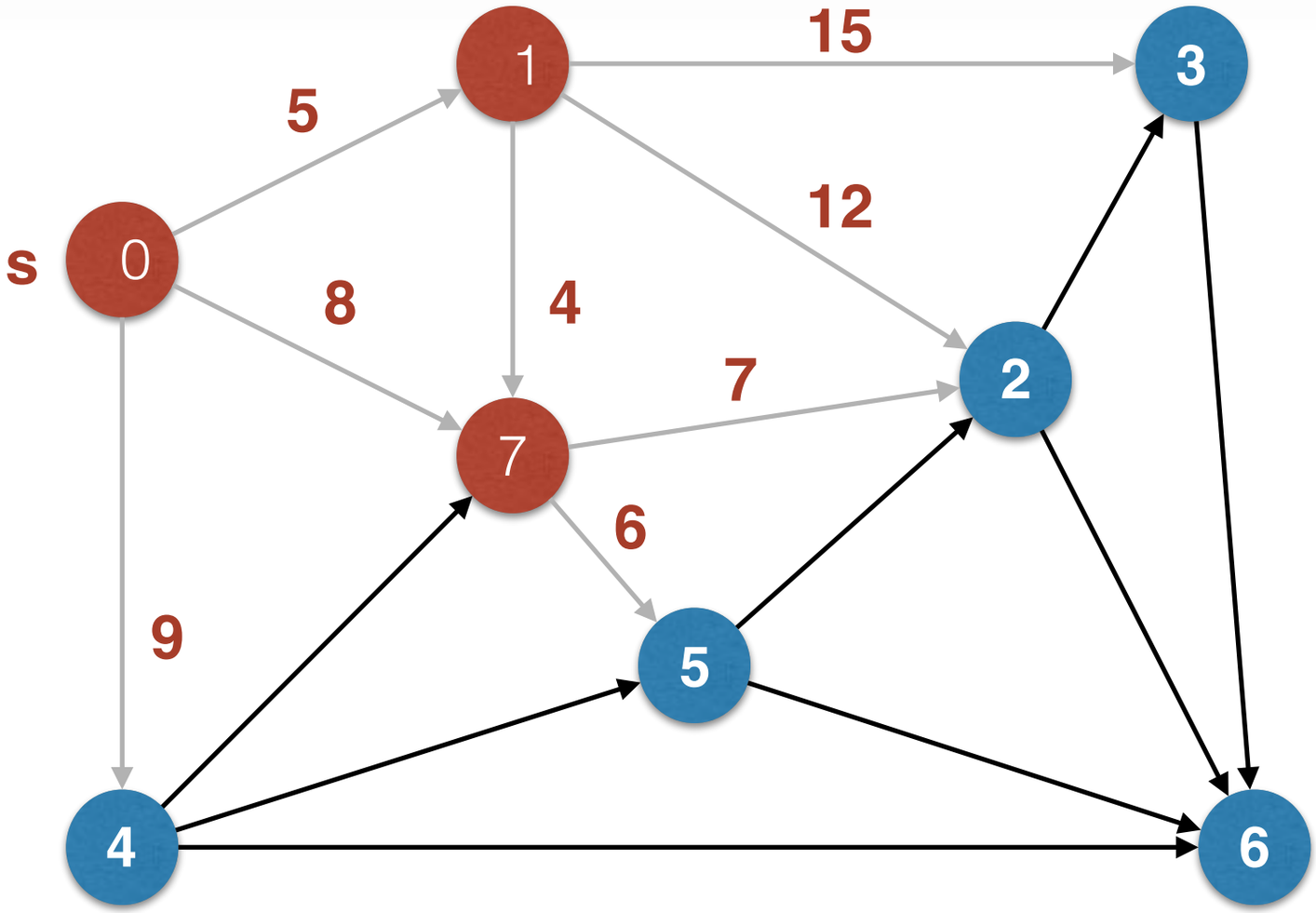
4. RELAX !!



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
4	9	0->4
5	14	7->5
6	INFTY	null
✗ 7	8	0->7

Algoritmo de Dijkstra (Codicioso)

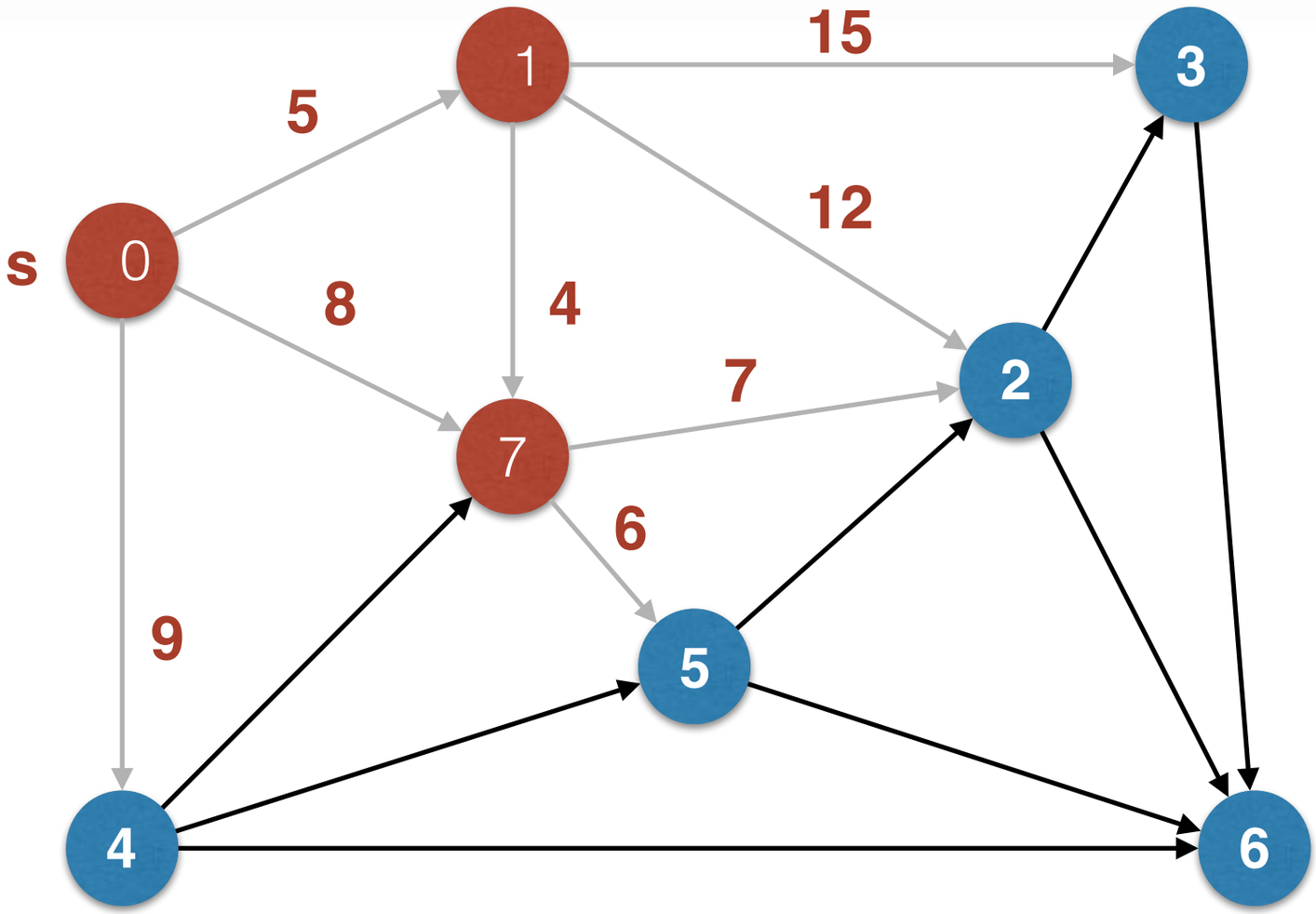
5. RELAX !!



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
4	9	0->4
5	14	7->5
6	INFTY	null
✗ 7	8	0->7

Algoritmo de Dijkstra (Codicioso)

5. RELAX !!

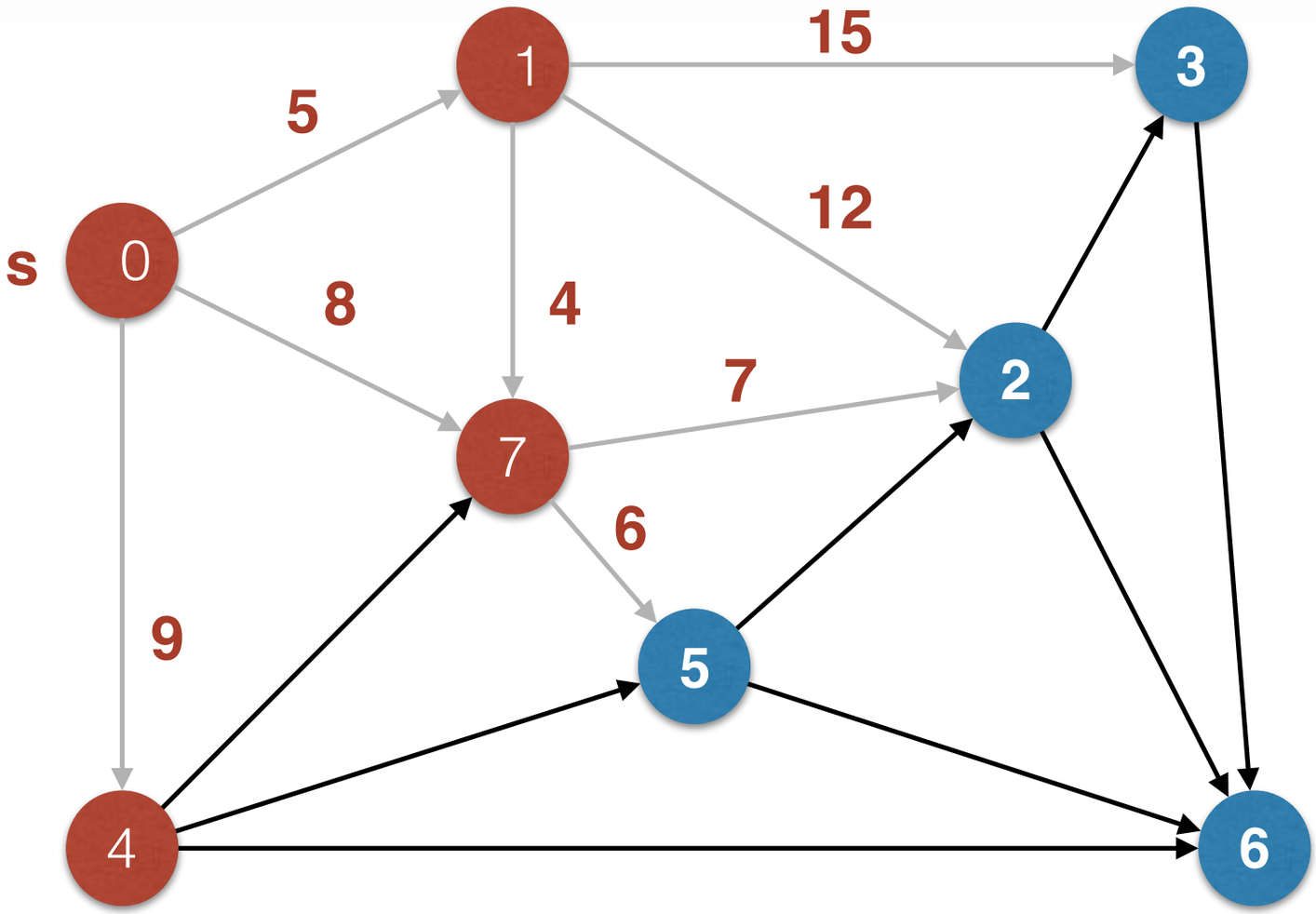


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✖ 0	0	null
✖ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
4	9	0->4
5	14	7->5
6	INFTY	null
✖ 7	8	0->7

(Seleccione el vértice con min costo desde s)

Algoritmo de Dijkstra (Codicioso)

5. RELAX !!

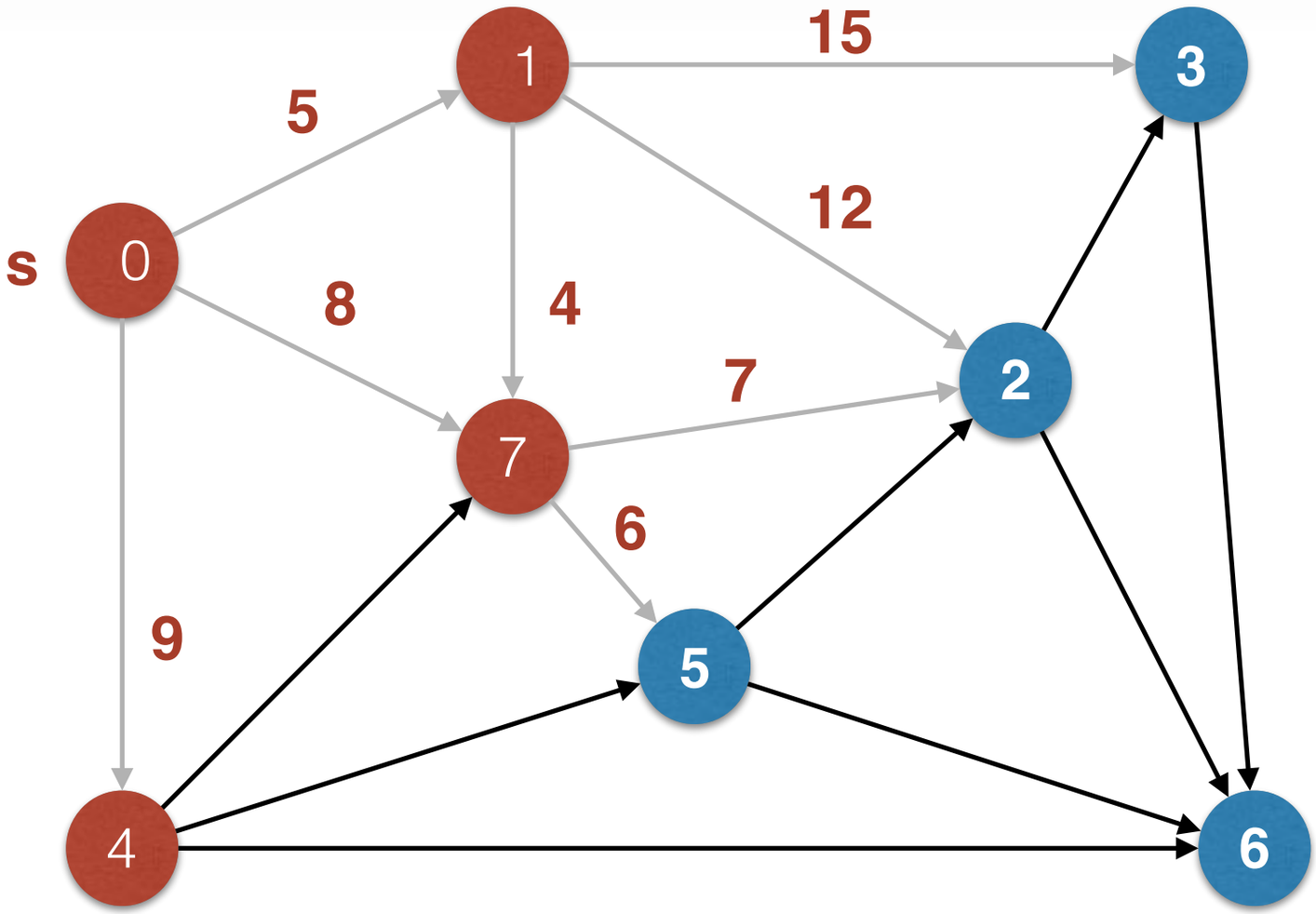


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
4	9	0->4
5	14	7->5
6	INFTY	null
✗ 7	8	0->7

(Seleccione el vértice con min costo desde s)

Algoritmo de Dijkstra (Codicioso)

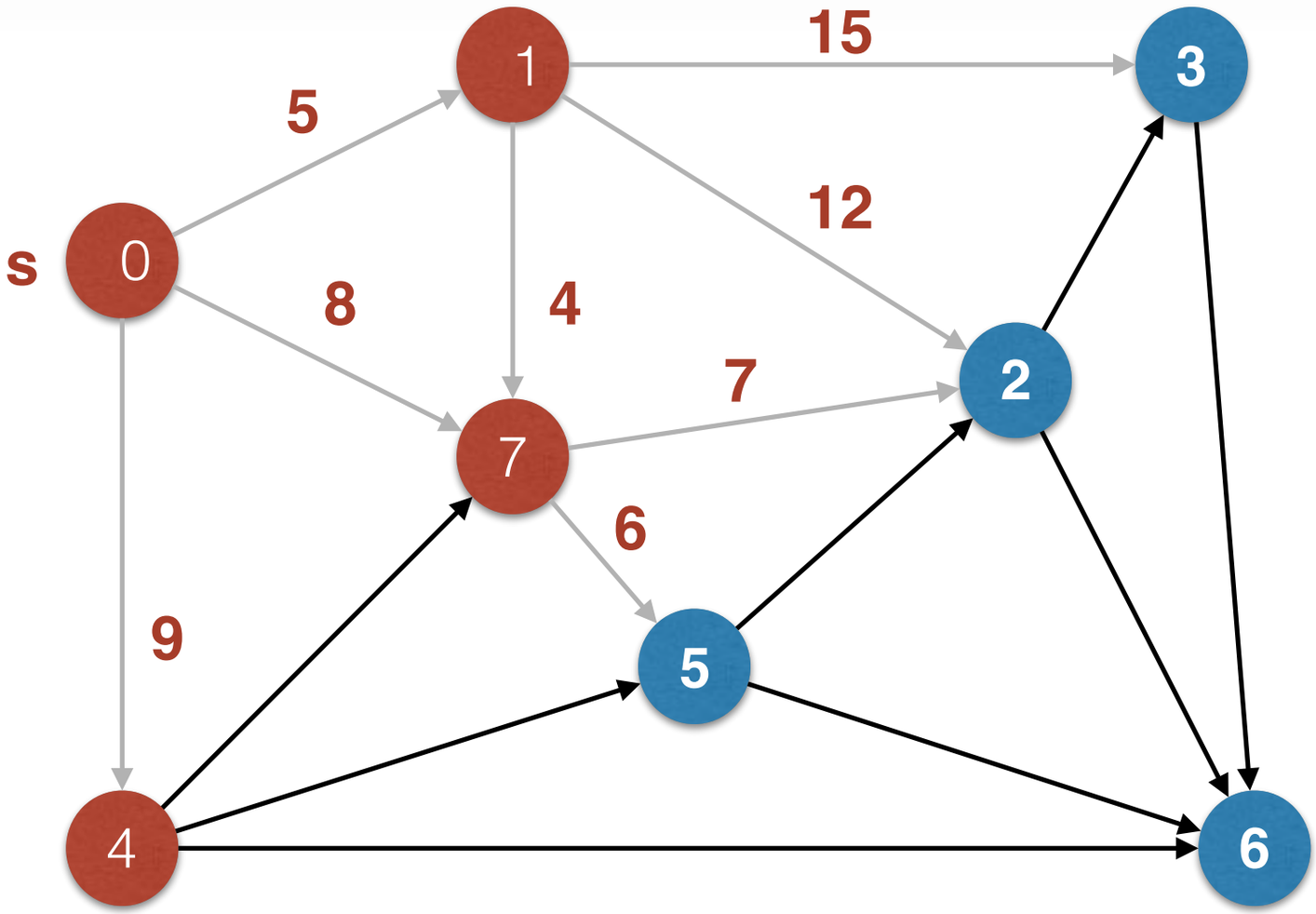
5. RELAX !! Aristas desde 4



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✖ 0	0	null
✖ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
✖ 4	9	0->4
5	14	7->5
6	INFTY	null
✖ 7	8	0->7

Algoritmo de Dijkstra (Codicioso)

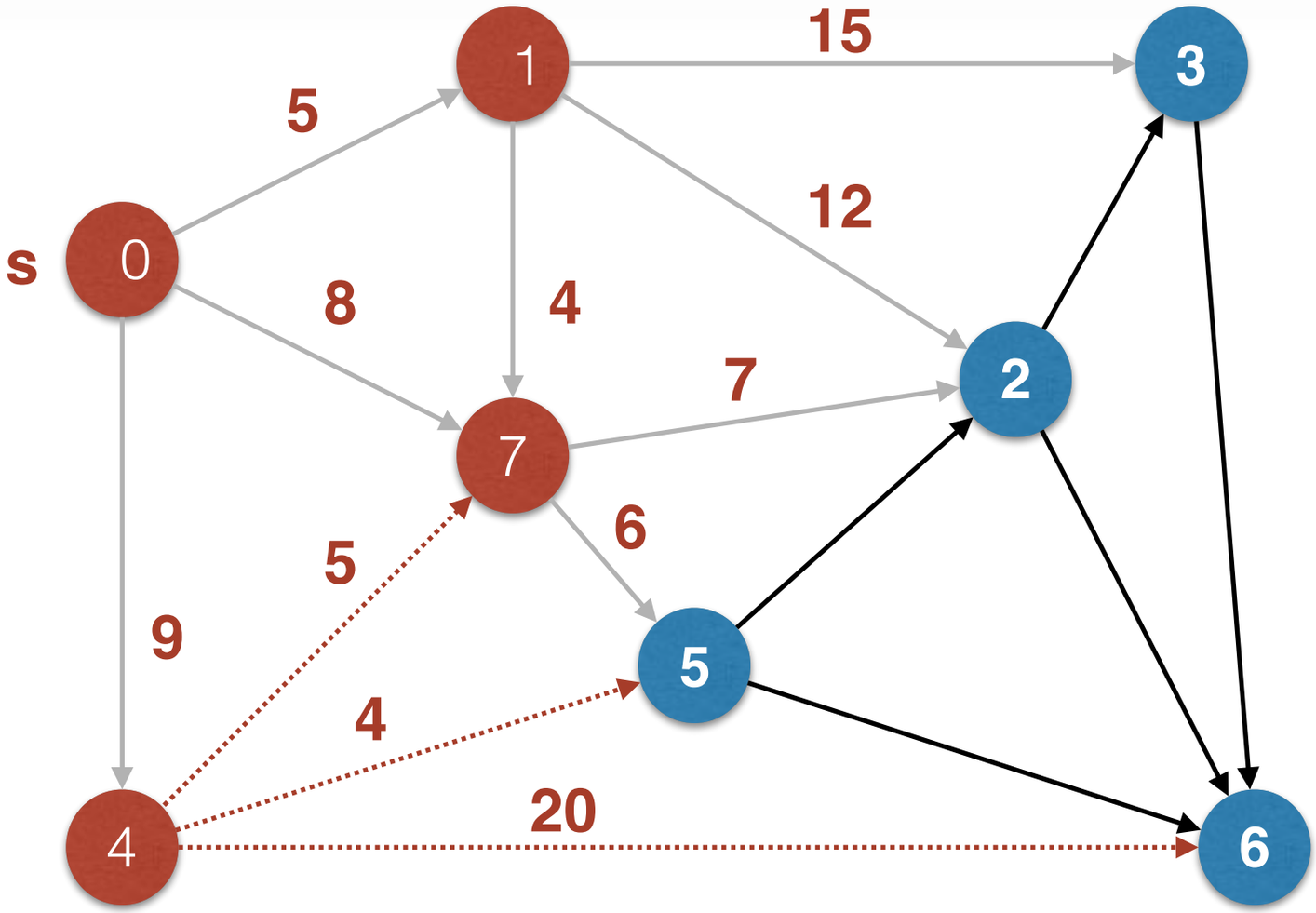
5. RELAX !! Aristas desde 4



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✖ 0	0	null
✖ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
✖ 4	9	0->4
5	14	7->5
6	INFTY	null
✖ 7	8	0->7

Algoritmo de Dijkstra (Codicioso)

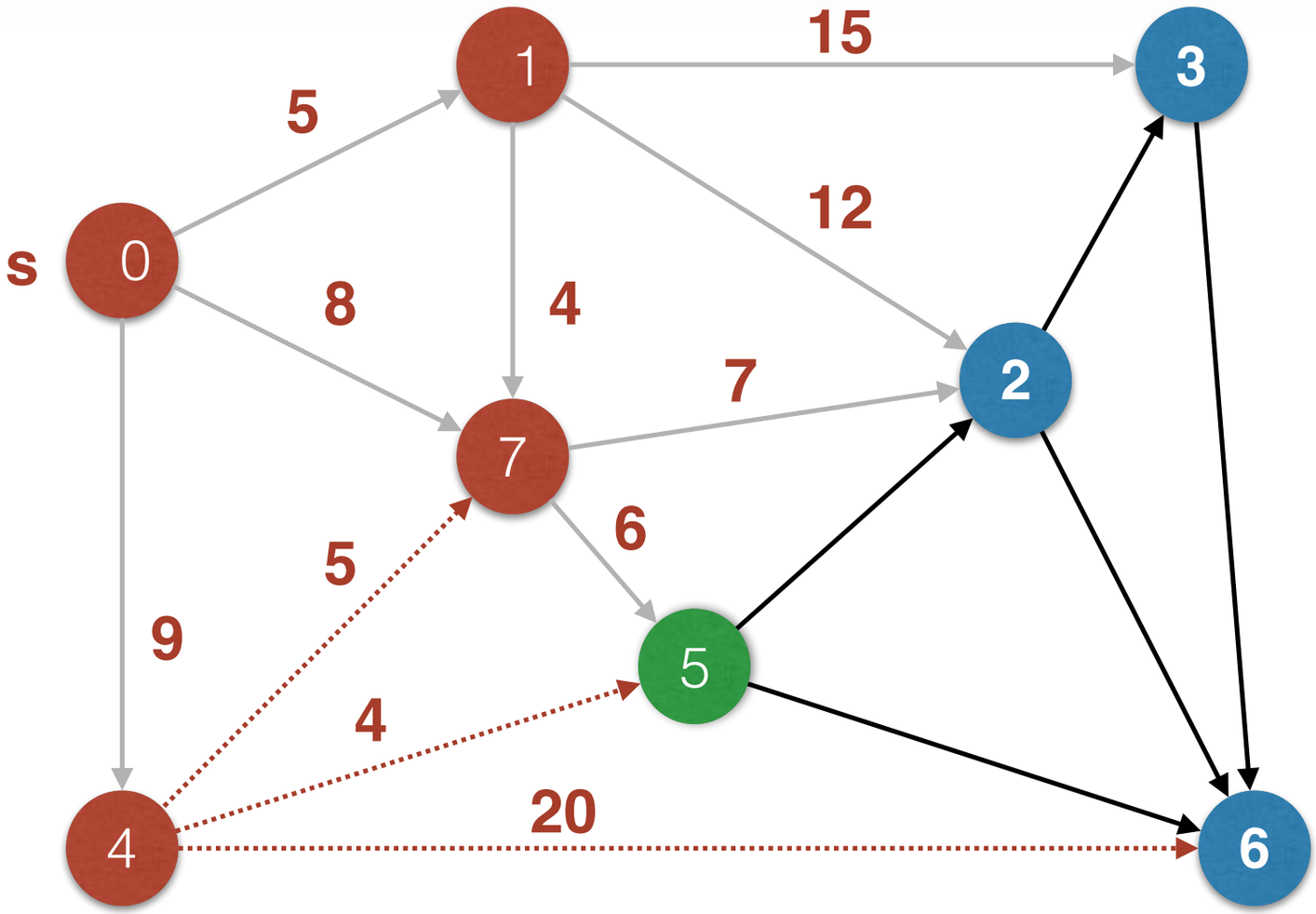
5. RELAX !!



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
✗ 4	9	0->4
5	14	7->5
6	INFTY	null
✗ 7	8	0->7

Algoritmo de Dijkstra (Codicioso)

5. RELAX !!

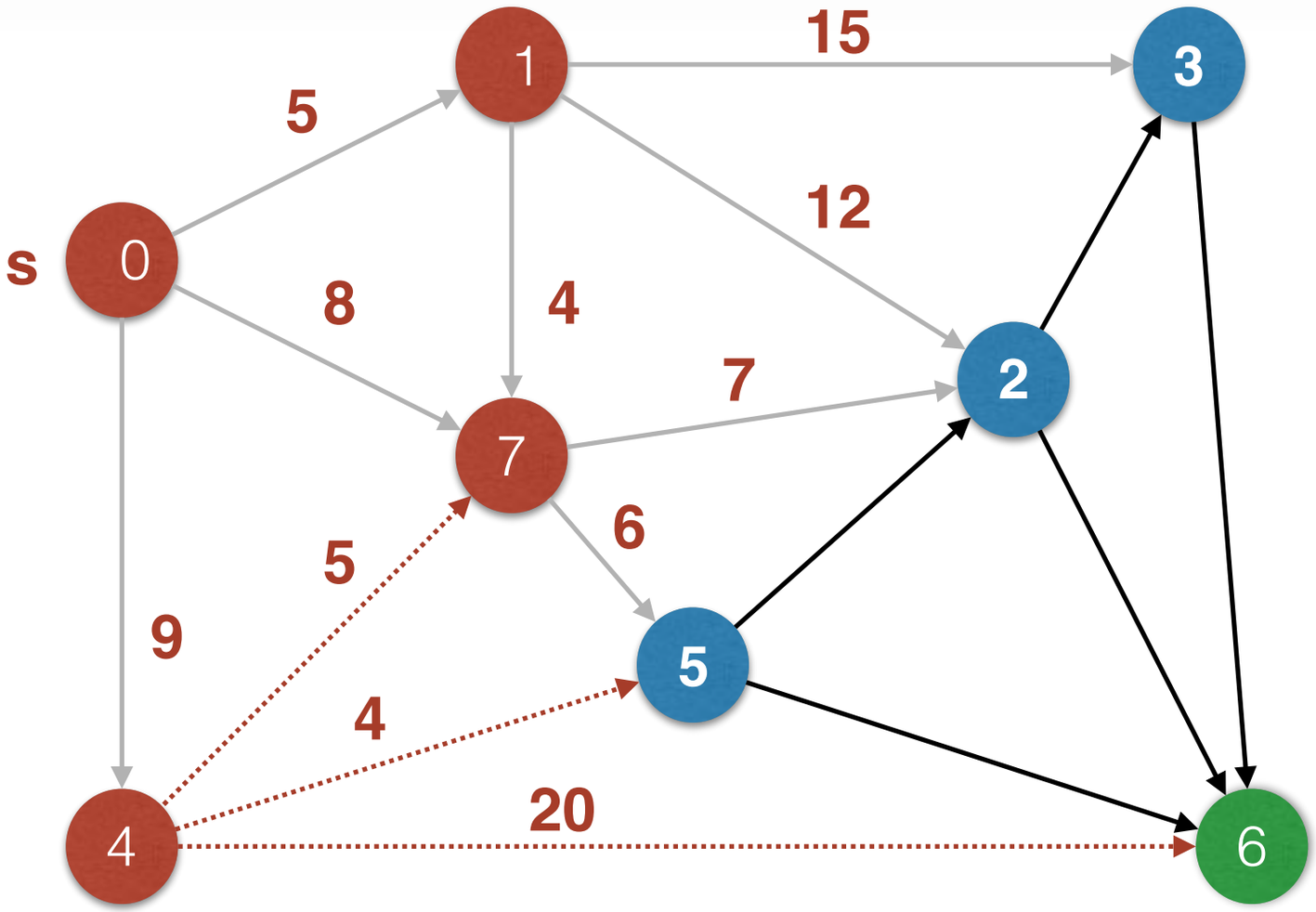


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
✗ 4	9	0->4
5	14	7->5
6	INFTY	null
✗ 7	8	0->7

9 + 4 < 14 ?

Algoritmo de Dijkstra (Codicioso)

5. RELAX !!

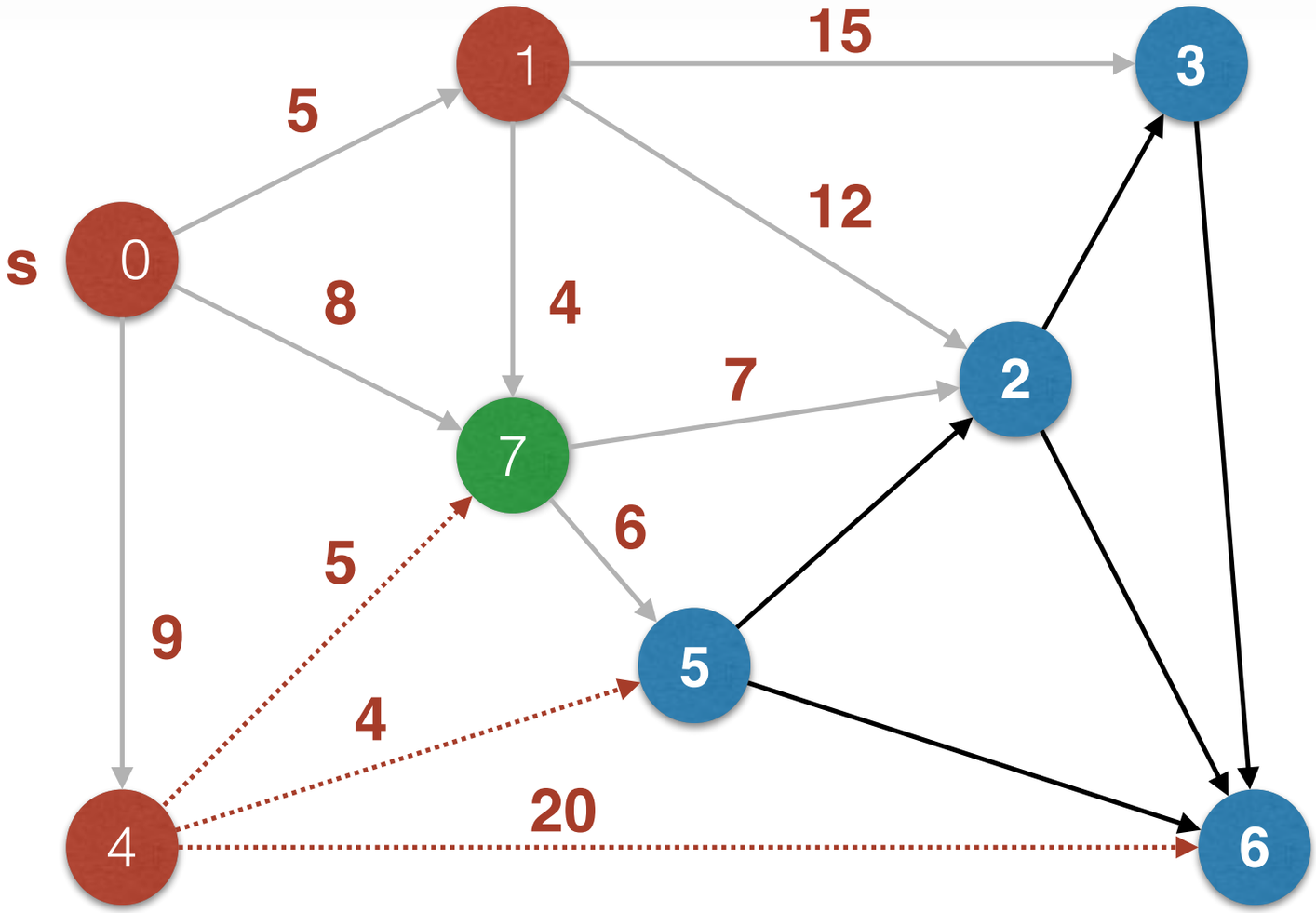


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
✗ 4	9	0->4
5	13	4->5
6	INFTY	null
✗ 7	8	0->7

9 + 29 < INFTY ?

Algoritmo de Dijkstra (Codicioso)

5. RELAX !!

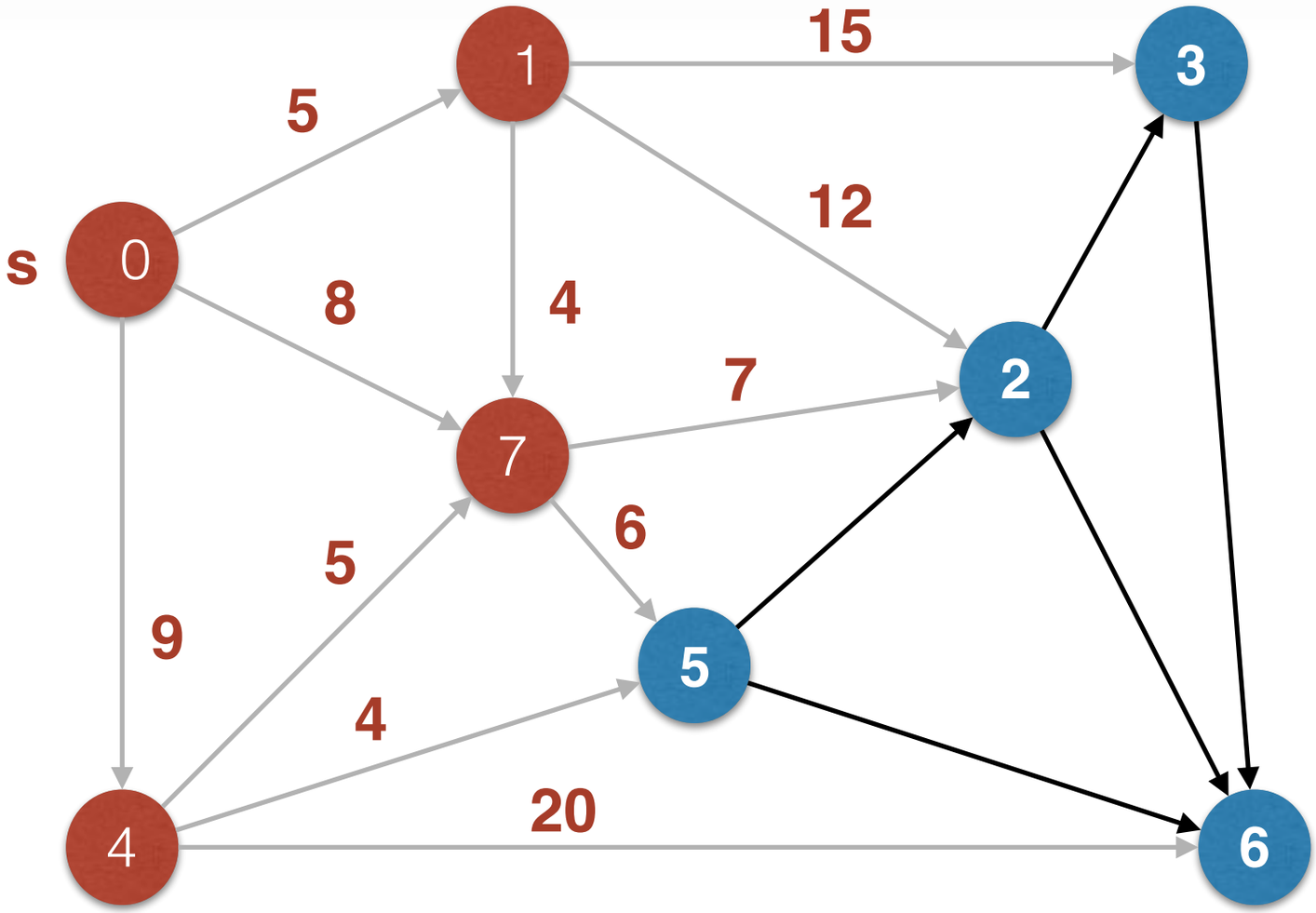


$9 + 5 < 8$?

v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
✗ 4	9	0->4
5	13	4->5
6	29	4->6
✗ 7	8	0->7

Algoritmo de Dijkstra (Codicioso)

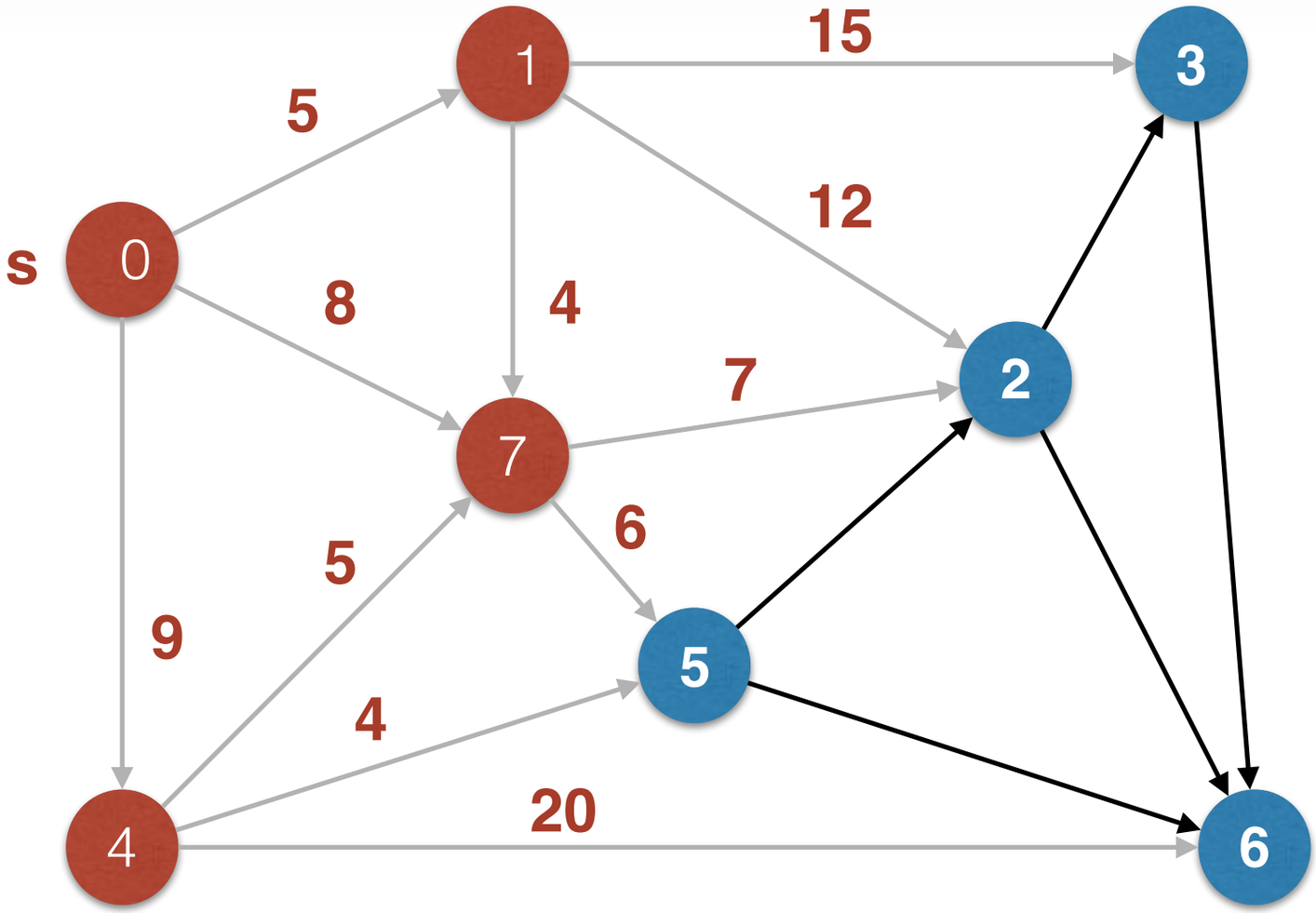
6. RELAX !!



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
✗ 4	9	0->4
5	13	4->5
6	29	4->6
✗ 7	8	0->7

Algoritmo de Dijkstra (Codicioso)

6. RELAX !!

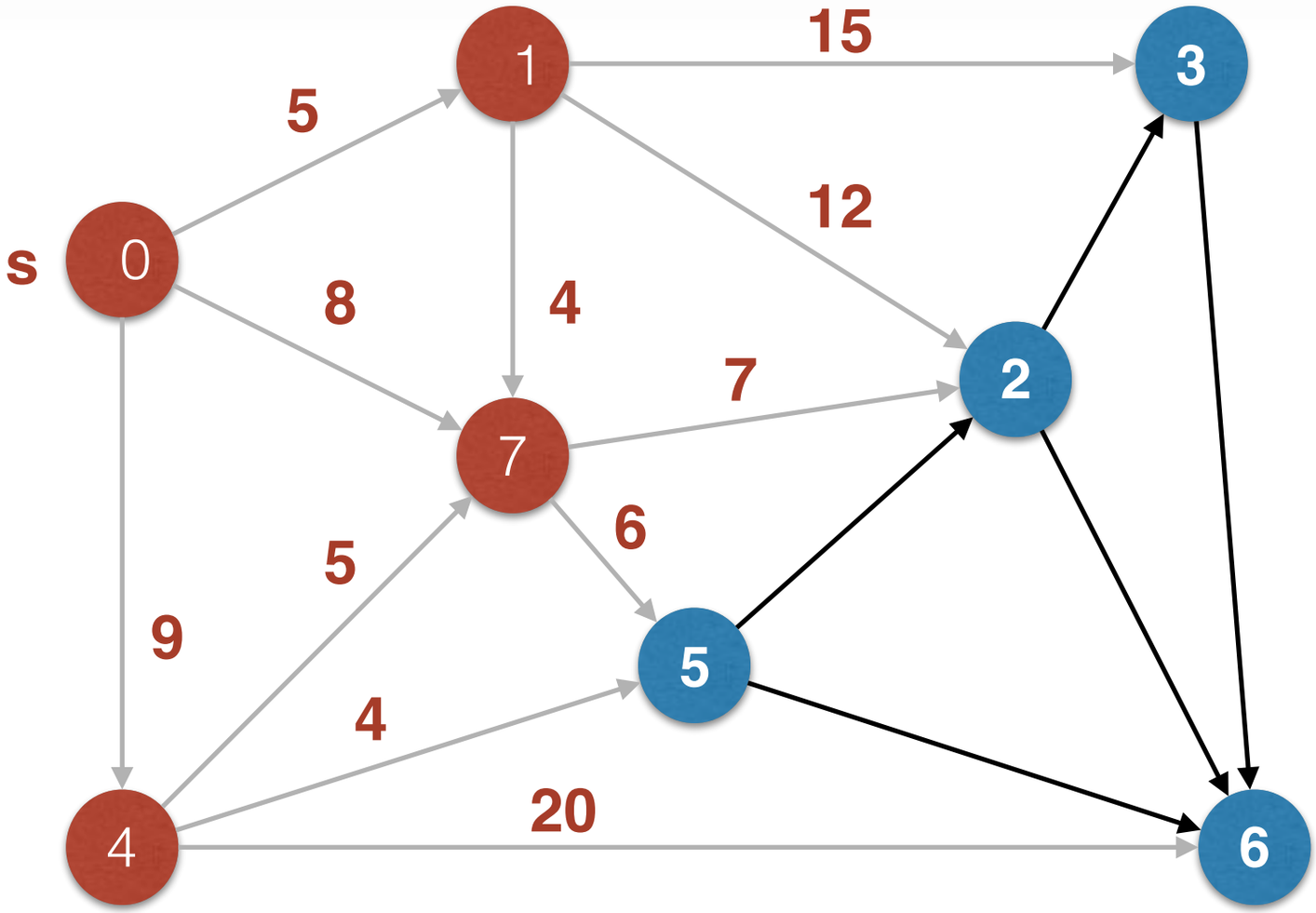


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
✗ 4	9	0->4
5	13	4->5
6	29	4->6
✗ 7	8	0->7

(Seleccione el vértice con min costo desde s)

Algoritmo de Dijkstra (Codicioso)

6. RELAX !!

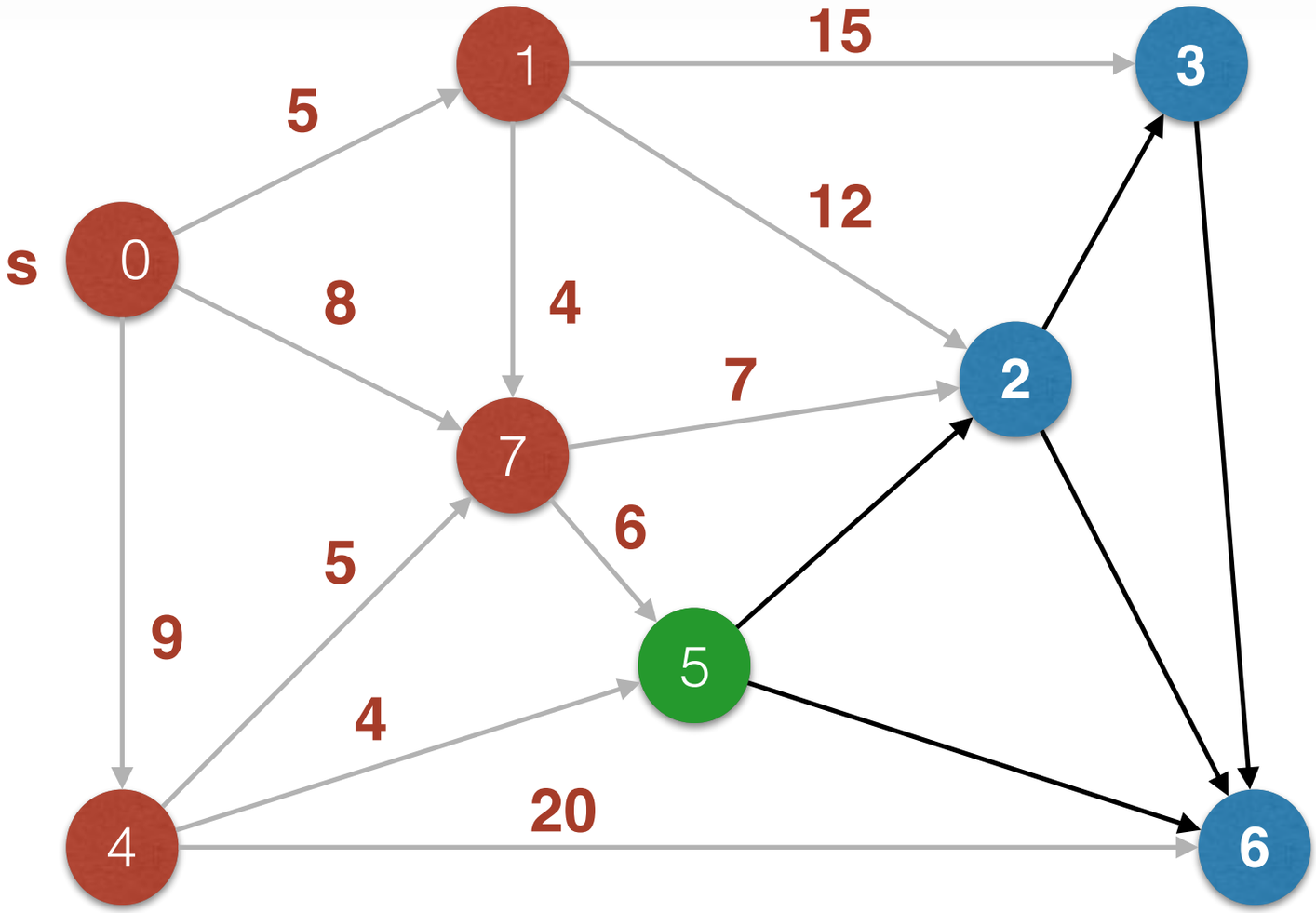


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
✗ 4	9	0->4
5	13	4->5
6	29	4->6
✗ 7	8	0->7

(Seleccione el vértice con min costo desde s)

Algoritmo de Dijkstra (Codicioso)

6. RELAX !! Aristas desde 5

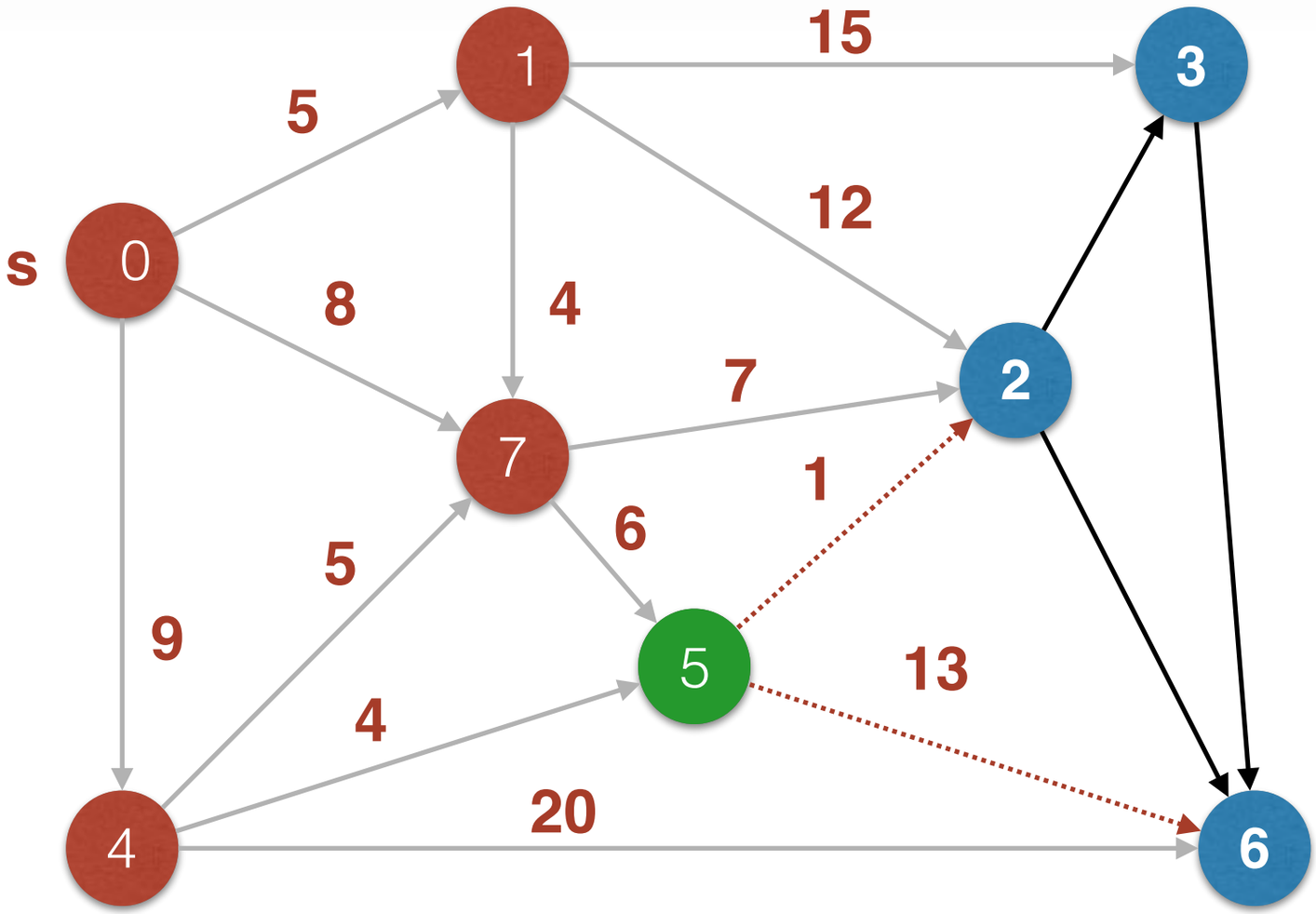


v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
✗ 4	9	0->4
5	13	4->5
6	29	4->6
✗ 7	8	0->7

$\delta(s, v)$ $\pi(v)$

Algoritmo de Dijkstra (Codicioso)

6. RELAX !! Aristas desde 5

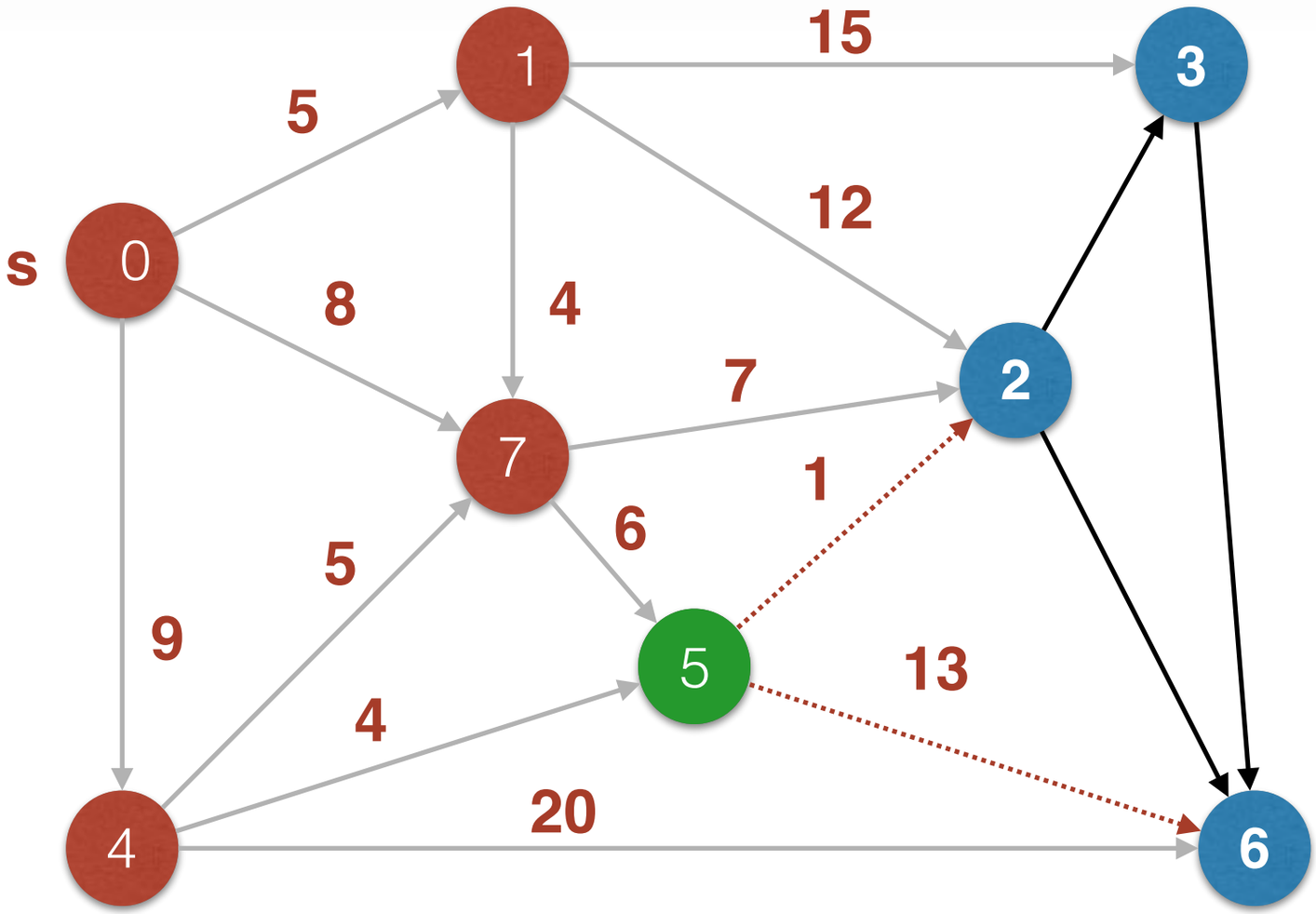


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**15**	**7->2**
3	20	1->3
✗ 4	9	0->4
✗ 5	13	4->5
6	29	4->6
✗ 7	8	0->7

13 + 1 < 15 ?

Algoritmo de Dijkstra (Codicioso)

6. RELAX !! Aristas desde 5



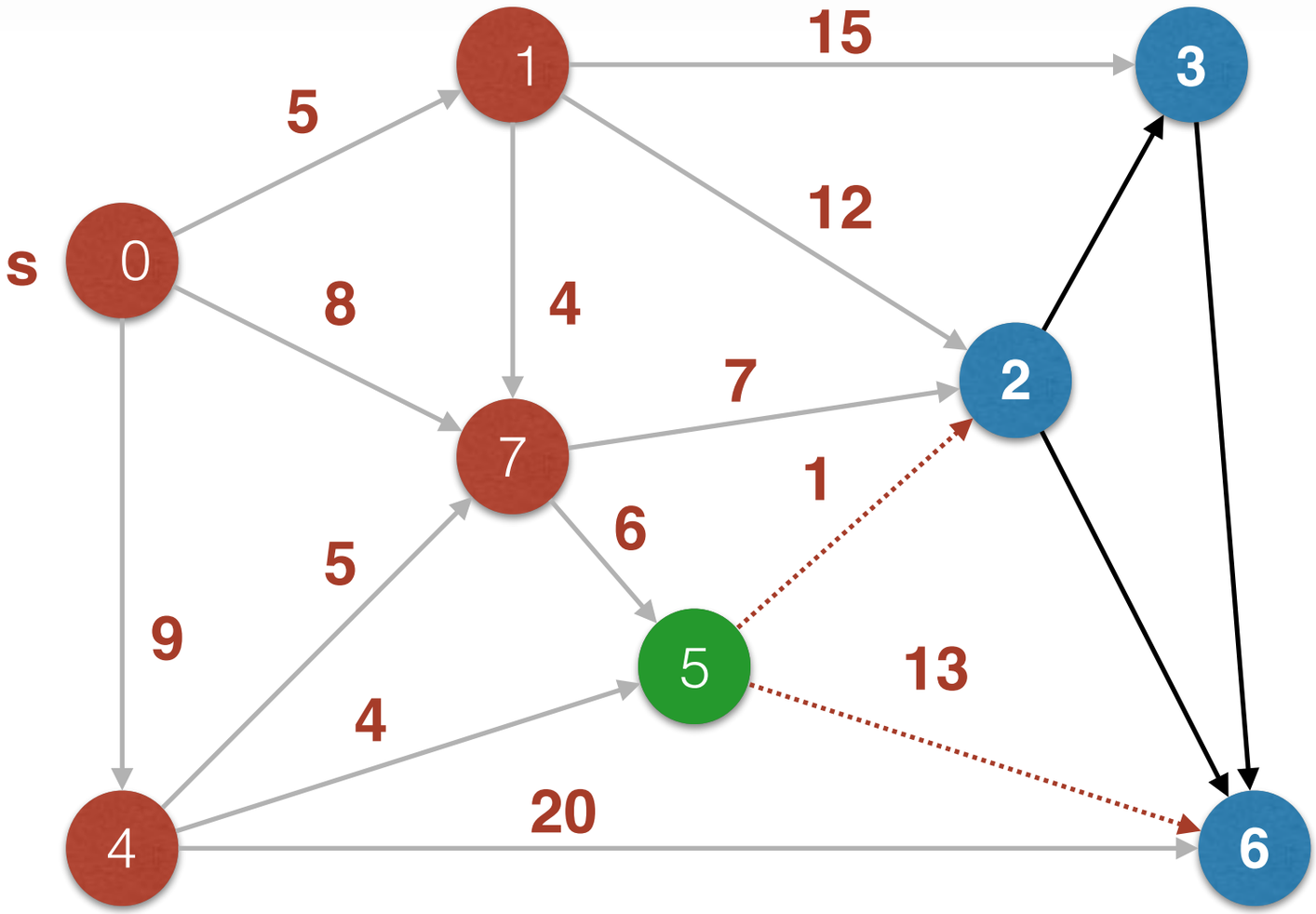
v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	**14**	**5->2**
3	20	1->3
✗ 4	9	0->4
✗ 5	13	4->5
6	29	4->6
✗ 7	8	0->7

$\delta(s, v)$ $\pi(v)$

13 + 1 < 15 ?

Algoritmo de Dijkstra (Codicioso)

6. RELAX !! Aristas desde 5



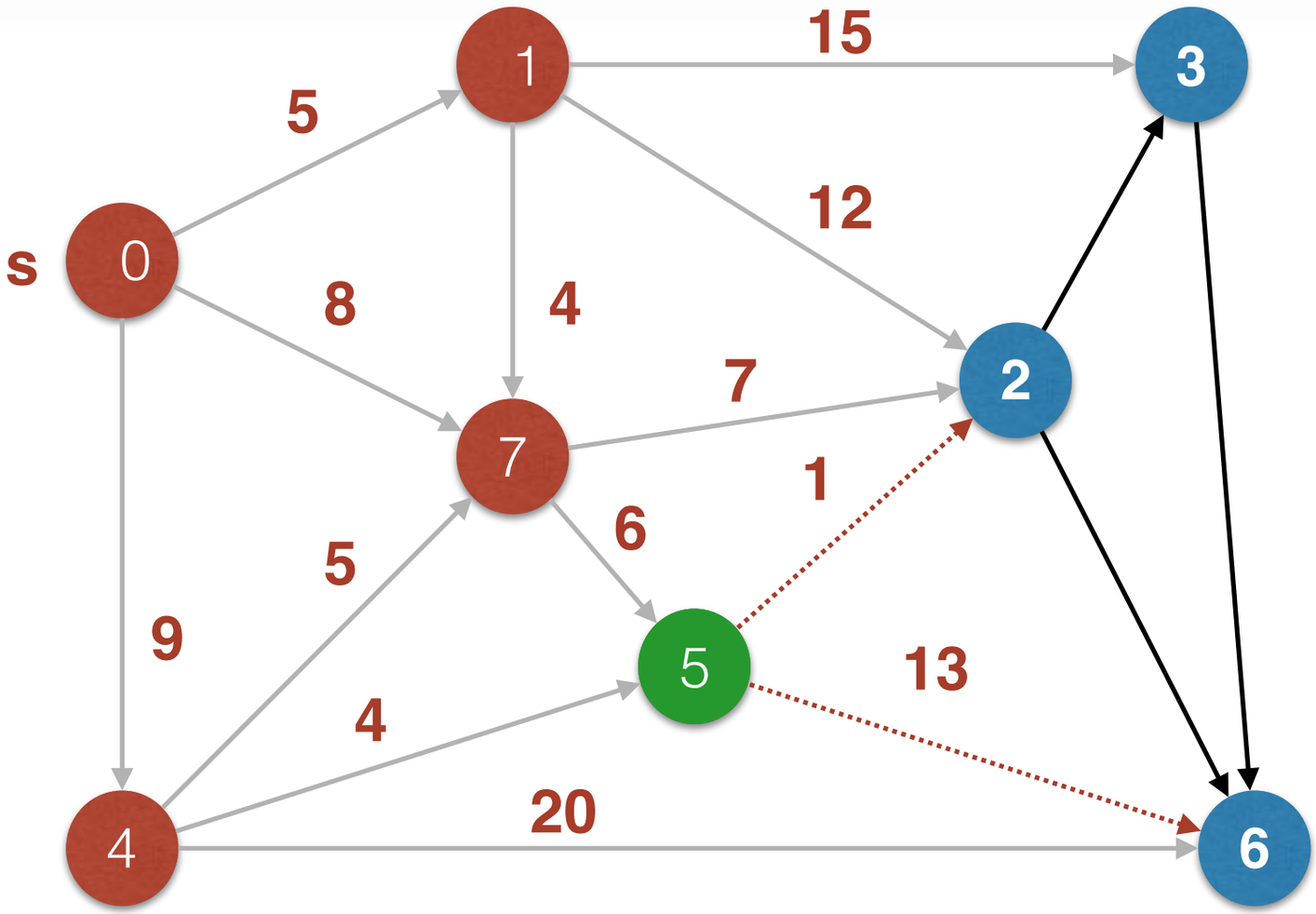
v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	14	5->2
3	20	1->3
✗ 4	9	0->4
✗ 5	13	4->5
6	**29**	**4->6**
✗ 7	8	0->7

$\delta(s,v)$ $\pi(v)$

13 + 13 < 29 ?

Algoritmo de Dijkstra (Codicioso)

6. RELAX !! Aristas desde 5

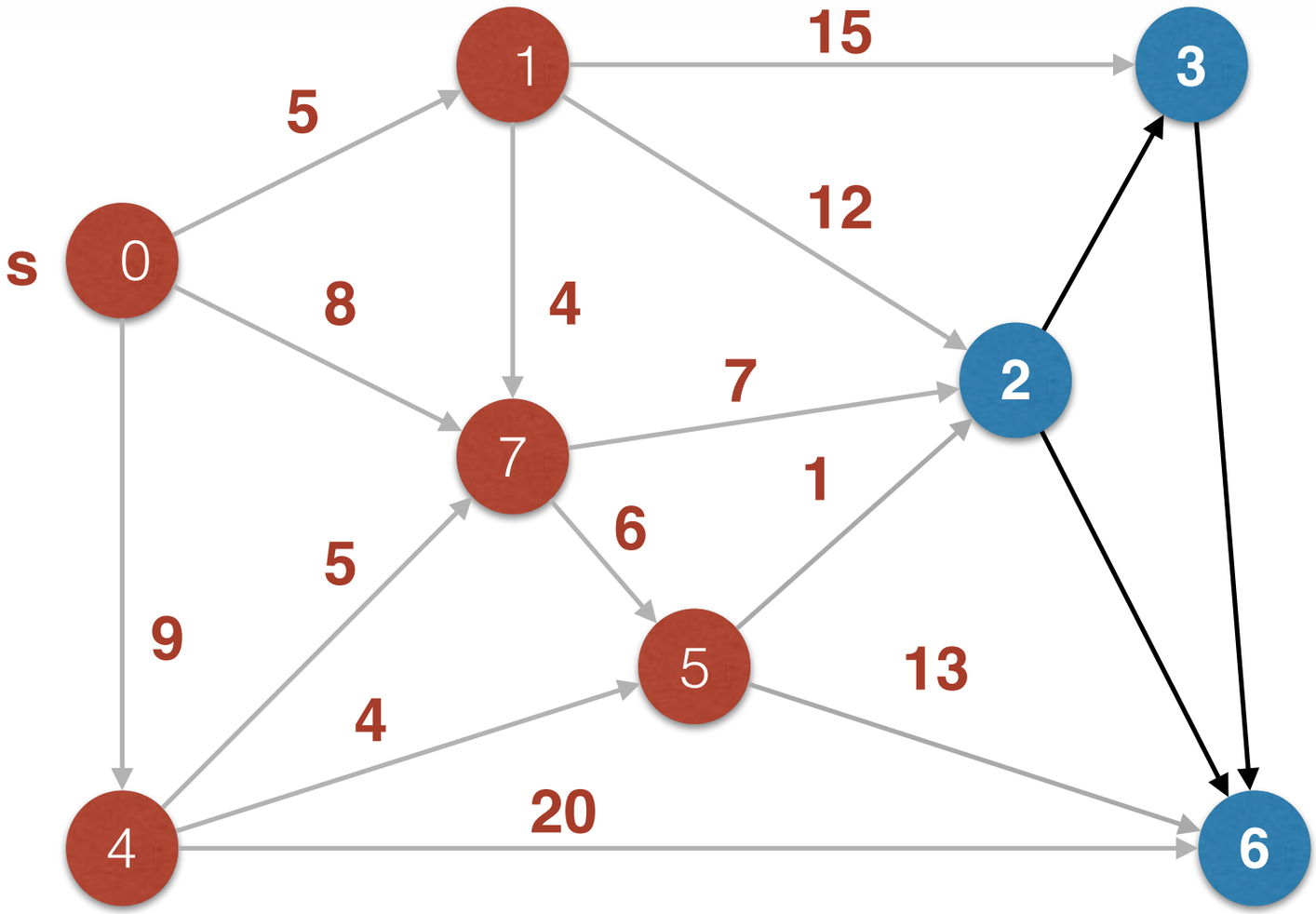


v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	14	5->2
3	20	1->3
✗ 4	9	0->4
✗ 5	13	4->5
6	**26**	**5->6**
✗ 7	8	0->7

13 + 13 < 29 ?

Algoritmo de Dijkstra (Codicioso)

6. RELAX !! Aristas desde 5

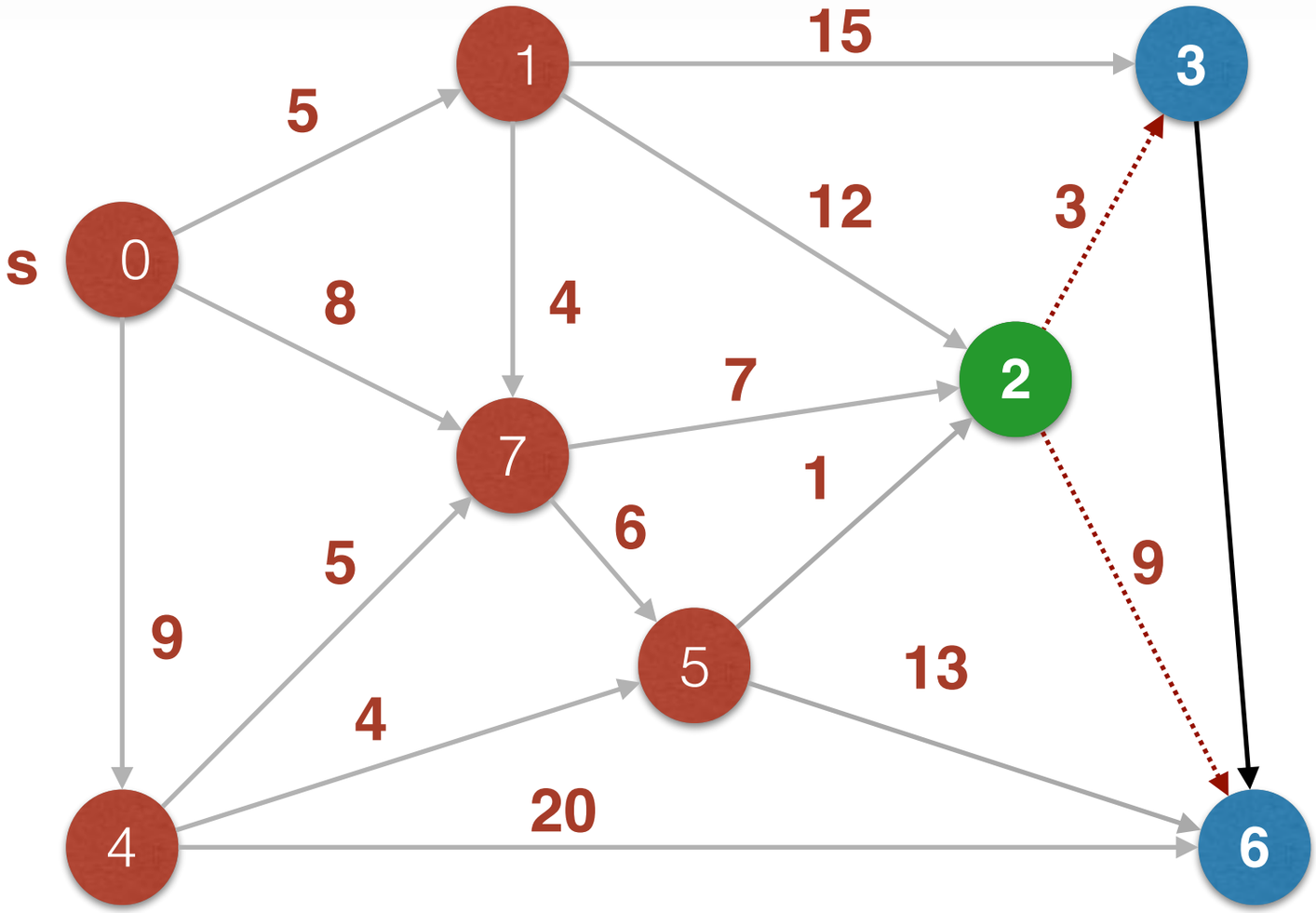


v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
2	14	5->2
3	20	1->3
✗ 4	9	0->4
✗ 5	13	4->5
6	26	4->6
✗ 7	8	0->7

$\delta(s, v)$ $\pi(v)$

Algoritmo de Dijkstra (Codicioso)

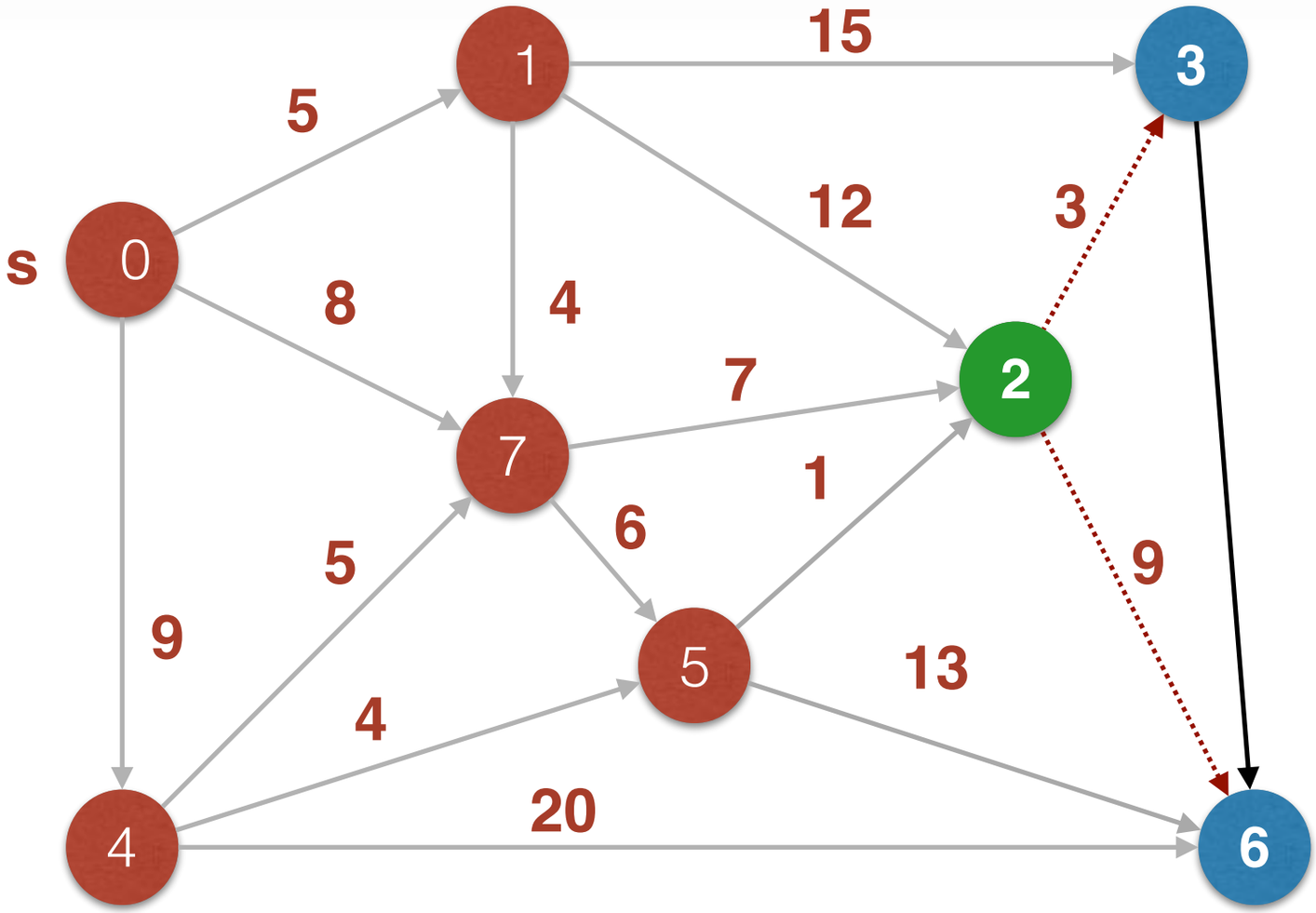
7. RELAX !! Aristas desde 2



v	distTo[v] $\delta(s,v)$	edge[v] $\pi(v)$
✗ 0	0	null
✗ 1	5	0->1
✗ 2	14	5->2
3	20	1->3
✗ 4	9	0->4
✗ 5	13	4->5
6	26	4->6
✗ 7	8	0->7

Algoritmo de Dijkstra (Codicioso)

7. RELAX !! Aristas desde 2

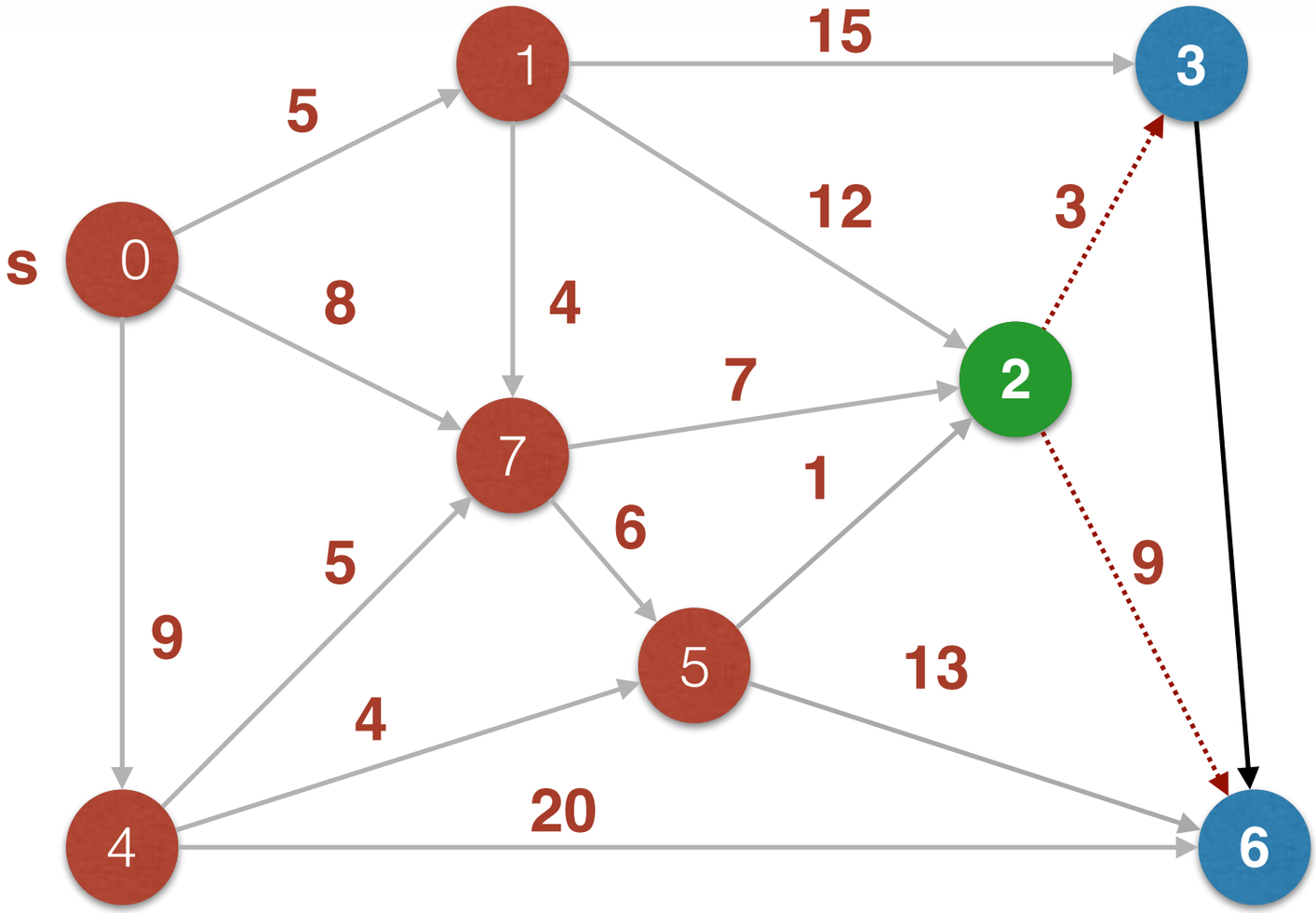


v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
✗ 2	14	5->2
3	**20**	**1->3**
✗ 4	9	0->4
✗ 5	13	4->5
6	26	4->6
✗ 7	8	0->7

14 + 3 < 20 ?

Algoritmo de Dijkstra (Codicioso)

7. RELAX !! Aristas desde 2

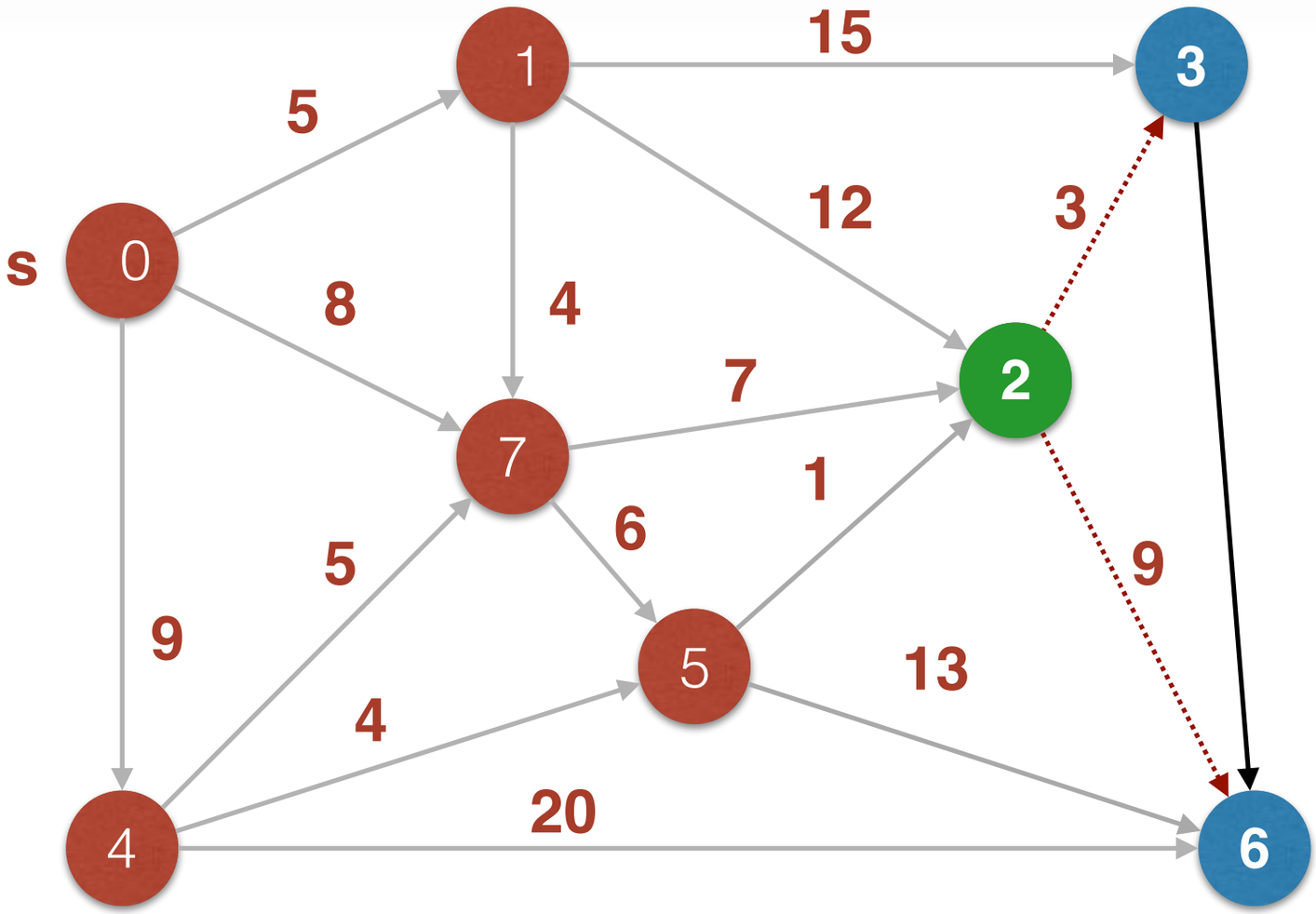


v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
✗ 2	14	5->2
3	**17**	**1->3**
✗ 4	9	0->4
✗ 5	13	4->5
6	26	4->6
✗ 7	8	0->7

14 + 3 < 20 ?

Algoritmo de Dijkstra (Codicioso)

7. RELAX !! Aristas desde 2



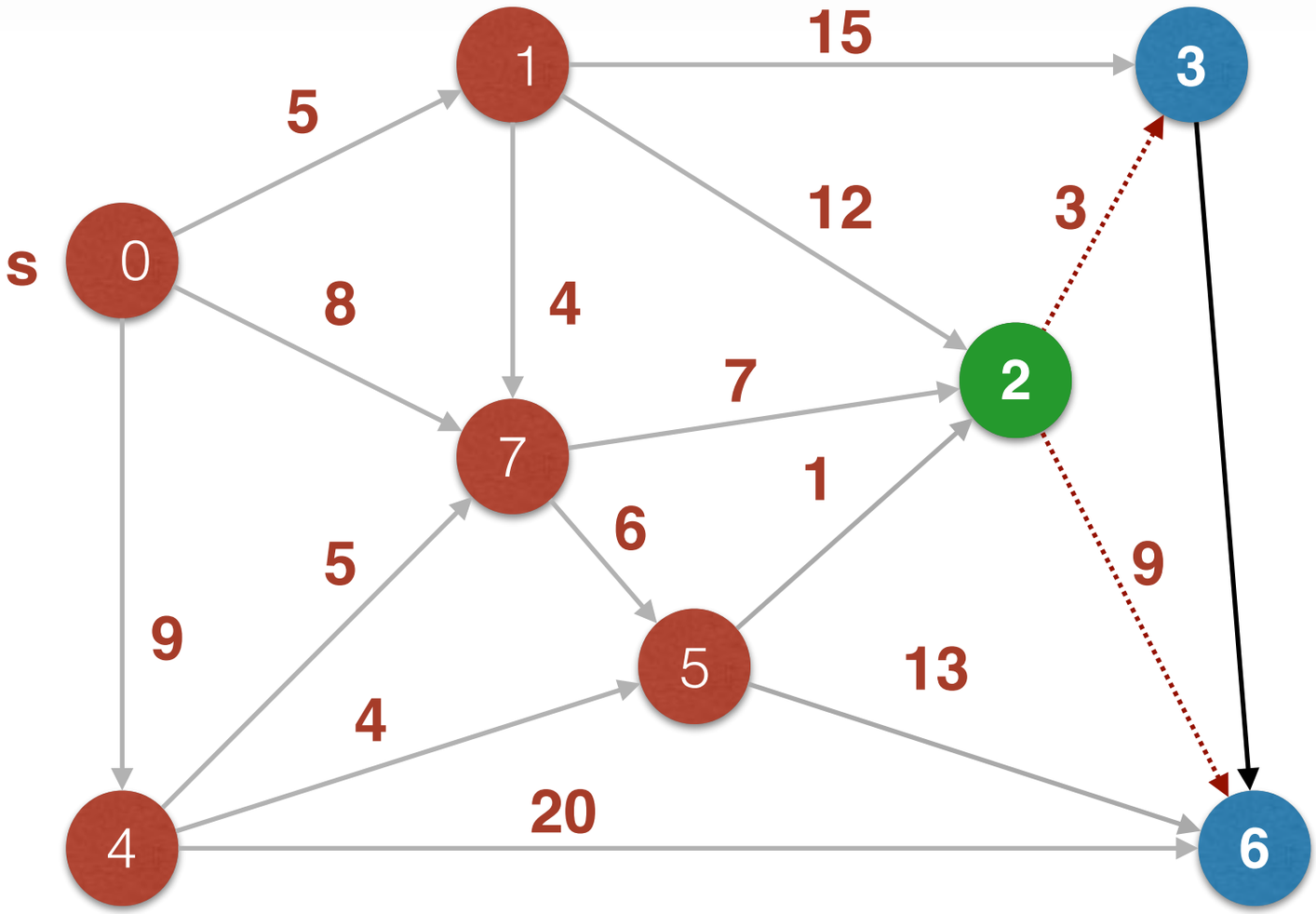
v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
✗ 2	14	5->2
3	17	2->3
✗ 4	9	0->4
✗ 5	13	4->5
6	**26**	**4->6**
✗ 7	8	0->7

$\delta(s,v)$ $\pi(v)$

14 + 9 < 26 ?

Algoritmo de Dijkstra (Codicioso)

7. RELAX !! Aristas desde 2



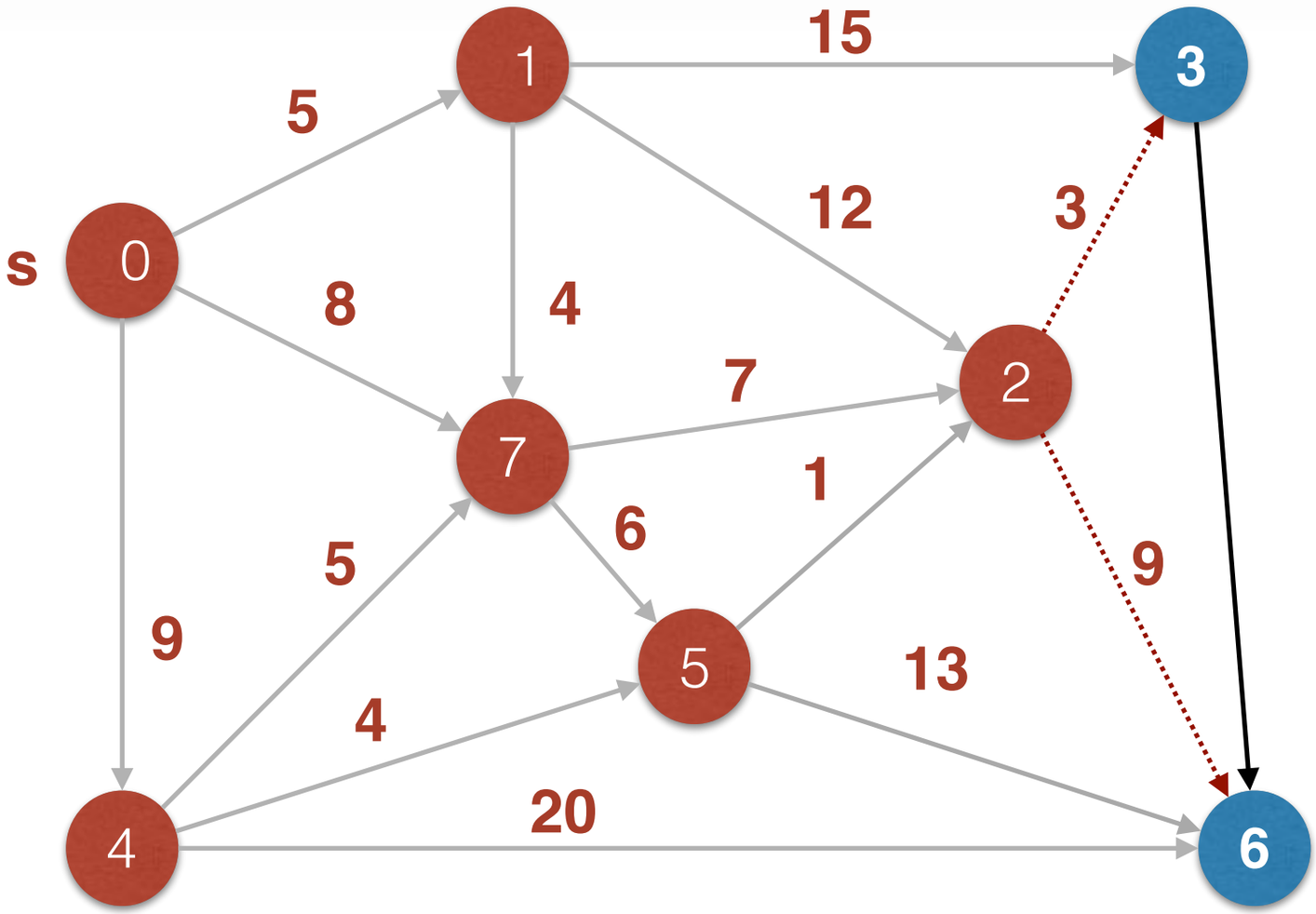
v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
✗ 2	14	5->2
3	17	2->3
✗ 4	9	0->4
✗ 5	13	4->5
6	**23**	**4->6**
✗ 7	8	0->7

$\delta(s,v)$ $\pi(v)$

14 + 9 < 26 ?

Algoritmo de Dijkstra (Codicioso)

7. RELAX !! Aristas desde 2

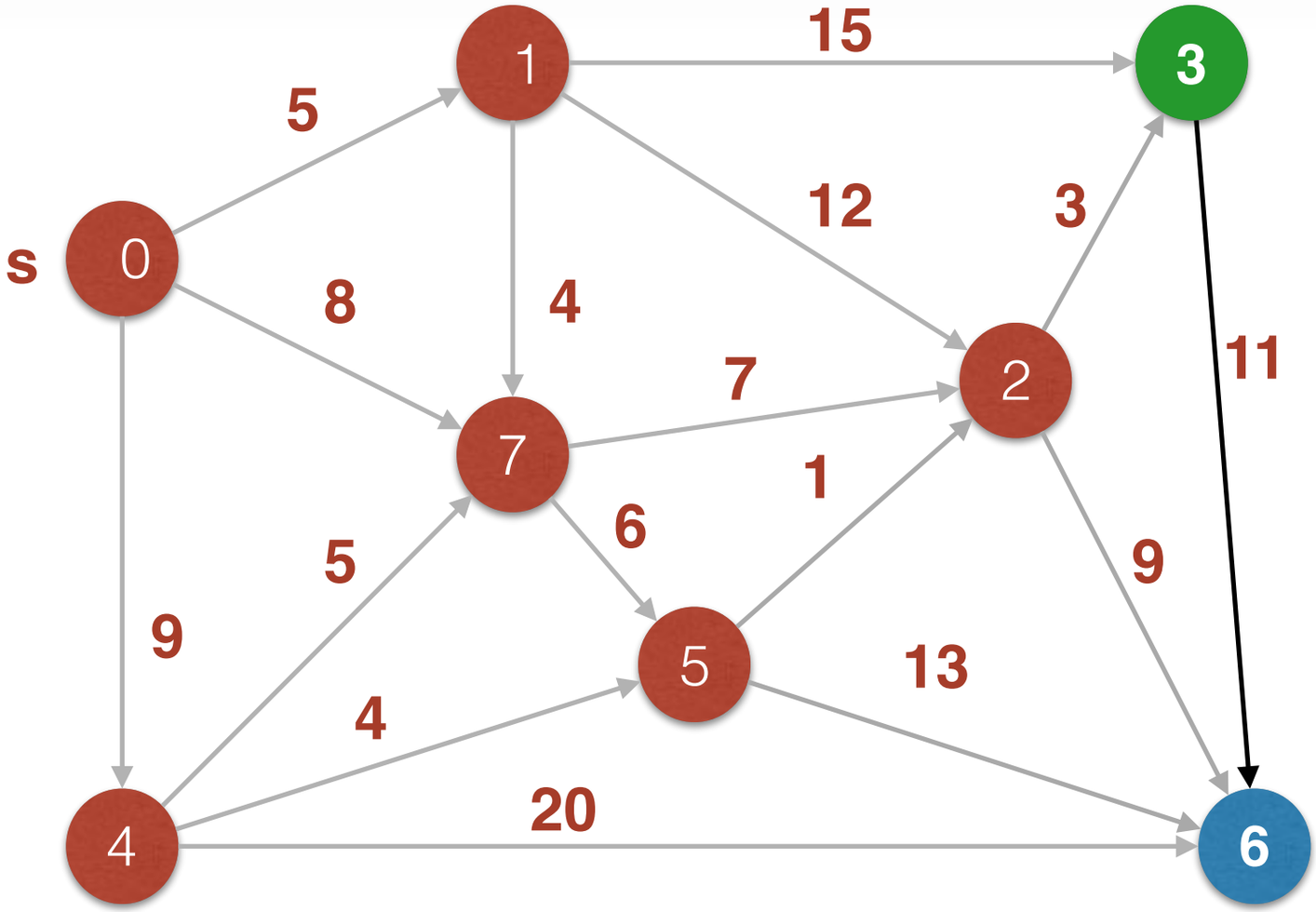


$14 + 9 < 26$?

v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
✗ 2	14	5->2
3	17	2->3
✗ 4	9	0->4
✗ 5	13	4->5
6	23	2->6
✗ 7	8	0->7

Algoritmo de Dijkstra (Codicioso)

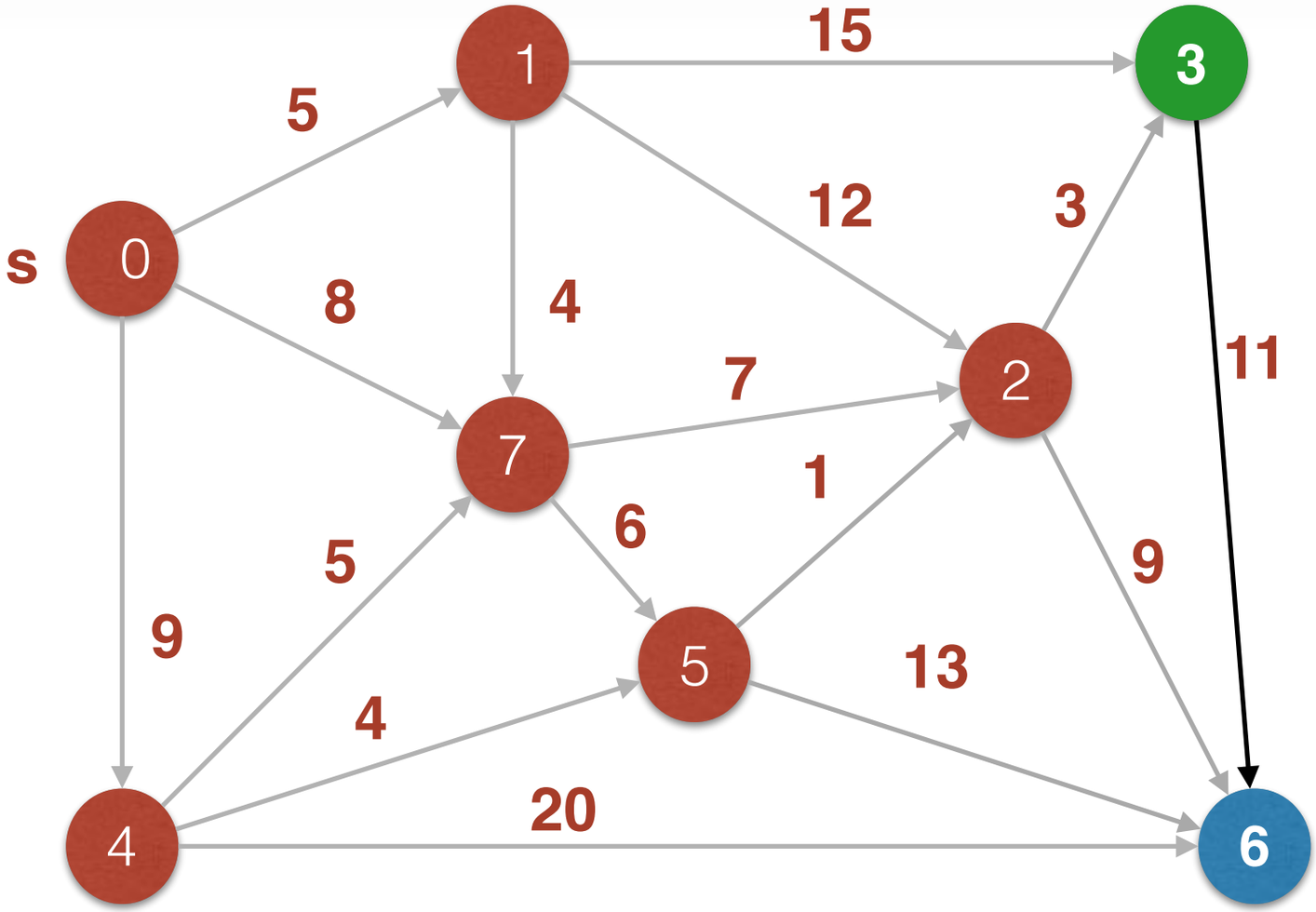
8. RELAX !! Aristas desde 3



v	distTo[v] $\delta(s,v)$	edge[v] $\pi(v)$
✗ 0	0	null
✗ 1	5	0->1
✗ 2	14	5->2
✗ 3	17	2->3
✗ 4	9	0->4
✗ 5	13	4->5
6	**23**	**2->6**
✗ 7	8	0->7

Algoritmo de Dijkstra (Codicioso)

8. RELAX !! Aristas desde 3



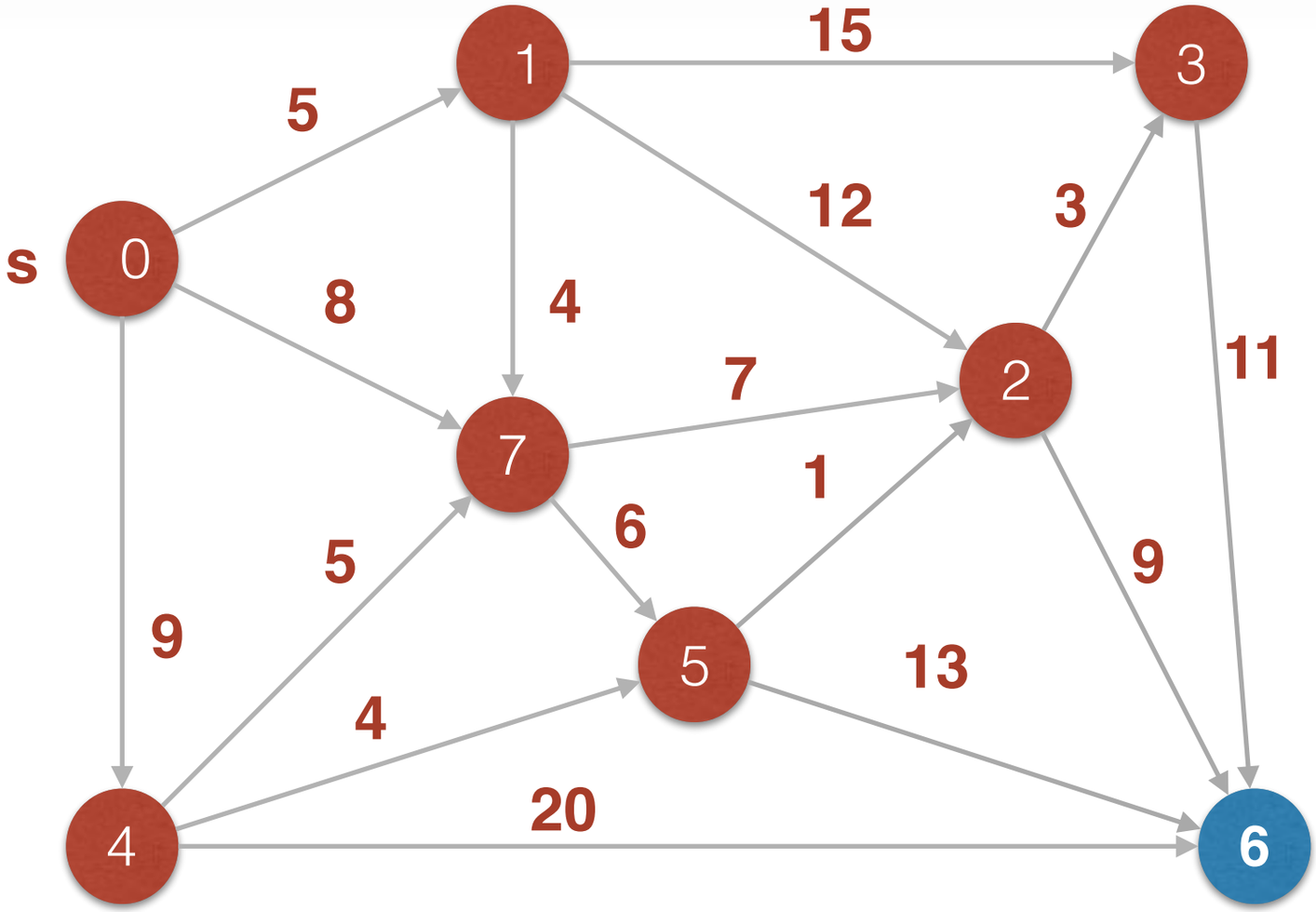
v	distTo[v]	edge[v]
0	0	null
1	5	0->1
2	14	5->2
3	17	2->3
4	9	0->4
5	13	4->5
6	**23**	**2->6**
7	8	0->7

$\delta(s, v)$ $\pi(v)$

17 + 11 < 23 ?

Algoritmo de Dijkstra (Codicioso)

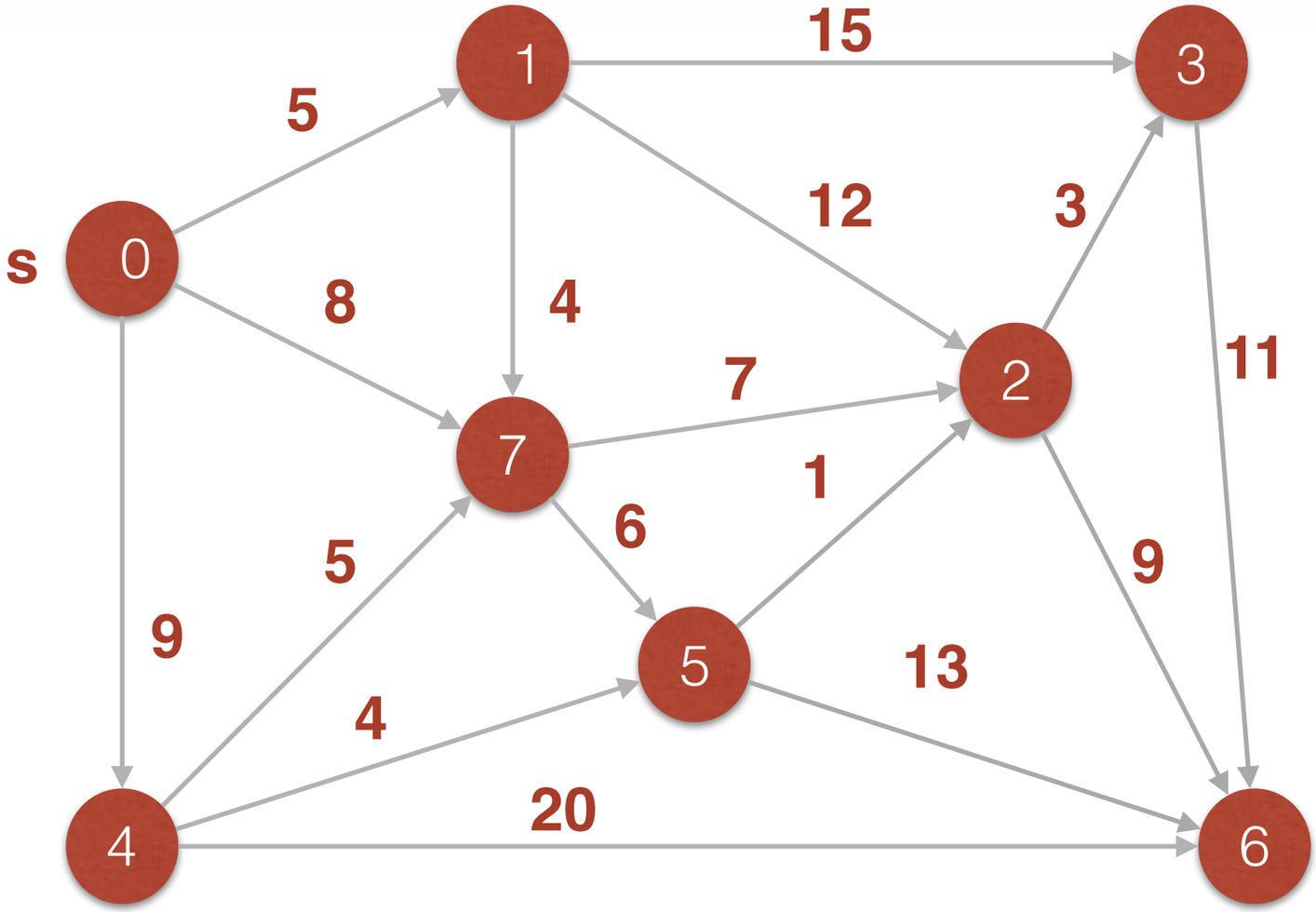
8. RELAX !! Aristas desde 3



	v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗	0	0	null
✗	1	5	0->1
✗	2	14	5->2
✗	3	17	2->3
✗	4	9	0->4
✗	5	13	4->5
	6	23	2->6
✗	7	8	0->7

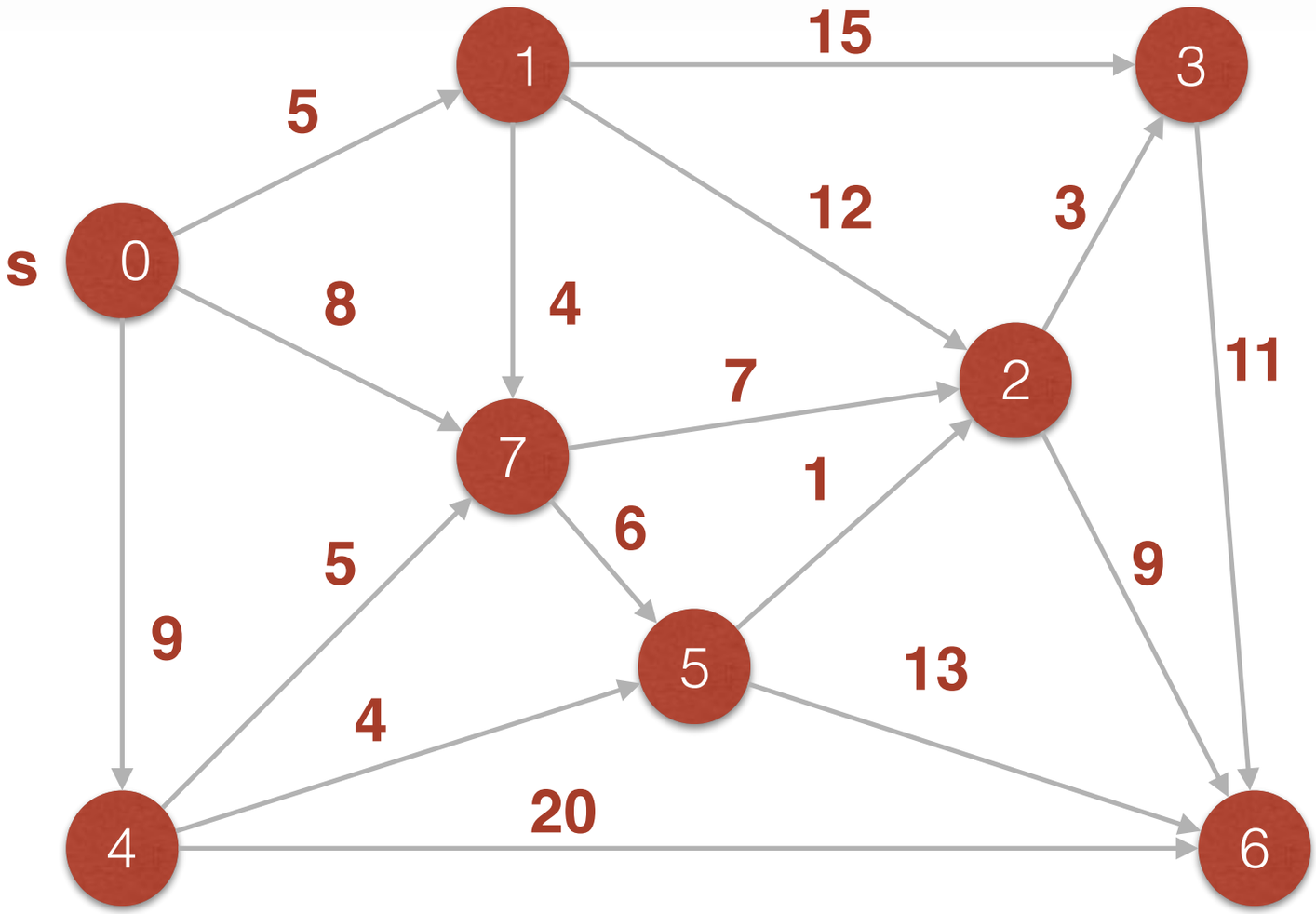
Algoritmo de Dijkstra (Codicioso)

9. RELAX !! Aristas desde 6



v	distTo[v]	edge[v]	$\delta(s,v)$	$\pi(v)$
✗ 0	0	null		
✗ 1	5	0->1		
✗ 2	14	5->2		
✗ 3	17	2->3		
✗ 4	9	0->4		
✗ 5	13	4->5		
✗ 6	23	2->6		
✗ 7	8	0->7		

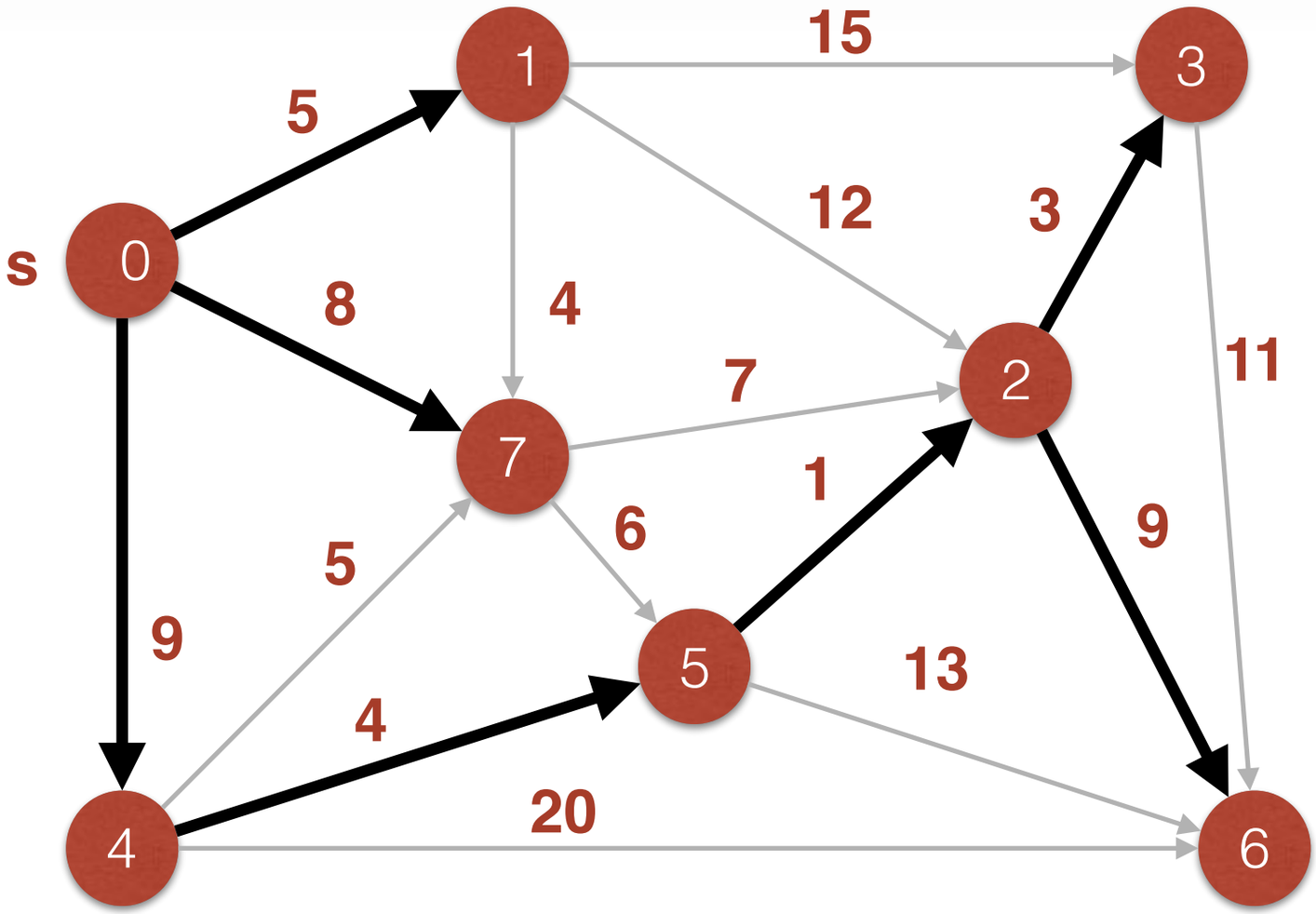
Algoritmo de Dijkstra (Codicioso)



v	distTo[v]	edge[v]
✗ 0	0	null
✗ 1	5	0->1
✗ 2	14	5->2
✗ 3	17	2->3
✗ 4	9	0->4
✗ 5	13	4->5
✗ 6	23	2->6
✗ 7	8	0->7

$\delta(s, v)$ $\pi(v)$

Algoritmo de Dijkstra (Codicioso)



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
✗ 0	0	null
✗ 1	5	0->1
✗ 2	14	5->2
✗ 3	17	2->3
✗ 4	9	0->4
✗ 5	13	4->5
✗ 6	23	2->6
✗ 7	8	0->7

COMPLEJIDAD DE DIJKSTRA

Dijkstra

(Asume que todos los pesos son no-negativos)

ALGORITHM 4.9 Dijkstra's shortest-paths algorithm

```
public class DijkstraSP
{
   private DirectedEdge[] edgeTo;
   private double[] distTo;
   private IndexMinPQ<Double> pq;

   public DijkstraSP(EdgeWeightedDigraph G, int s)
   {
      edgeTo = new DirectedEdge[G.V()];
      distTo = new double[G.V()];
      pq = new IndexMinPQ<Double>(G.V());

      for (int v = 0; v < G.V(); v++)
         distTo[v] = Double.POSITIVE_INFINITY;
      distTo[s] = 0.0;

      pq.insert(s, 0.0);
      while (!pq.isEmpty())
         relax(G, pq.delMin())
   }

   private void relax(EdgeWeightedDigraph G, int v)
   {
      for(DirectedEdge e : G.adj(v))
      {
         int w = e.to();
         if (distTo[w] > distTo[v] + e.weight())
         {
            distTo[w] = distTo[v] + e.weight();
            edgeTo[w] = e;
            if (pq.contains(w)) pq.change(w, distTo[w]);
            else                pq.insert(w, distTo[w]);
         }
      }
   }
```

Dijkstra

(Asume que todos los pesos son no-negativos)

ALGORITHM 4.9 Dijkstra's shortest-paths algorithm

```
public class DijkstraSP
{
   private DirectedEdge[] edgeTo;
   private double[] distTo;
   private IndexMinPQ<Double> pq;

   public DijkstraSP(EdgeWeightedDigraph G, int s)
   {
      edgeTo = new DirectedEdge[G.V()];
      distTo = new double[G.V()];
      pq = new IndexMinPQ<Double>(G.V());
      for (int v = 0; v < G.V(); v++)
         distTo[v] = Double.POSITIVE_INFINITY;
      distTo[s] = 0.0;

      pq.insert(s, 0.0);
      while (!pq.isEmpty())
         relax(G, pq.delMin())
   }

   private void relax(EdgeWeightedDigraph G, int v)
   {
      for(DirectedEdge e : G.adj(v))
      {
         int w = e.to();
         if (distTo[w] > distTo[v] + e.weight())
         {
            distTo[w] = distTo[v] + e.weight();
            edgeTo[w] = e;
            if (pq.contains(w)) pq.change(w, distTo[w]);
            else                pq.insert(w, distTo[w]);
         }
      }
   }
```

O(V)

Dijkstra

(Asume que todos los pesos son no-negativos)

ALGORITHM 4.9 Dijkstra's shortest-paths algorithm

```
public class DijkstraSP
{
   private DirectedEdge[] edgeTo;
   private double[] distTo;
   private IndexMinPQ<Double> pq;

   public DijkstraSP(EdgeWeightedDigraph G, int s)
   {
      edgeTo = new DirectedEdge[G.V()];
      distTo = new double[G.V()];
      pq = new IndexMinPQ<Double>(G.V());
      for (int v = 0; v < G.V(); v++)
         distTo[v] = Double.POSITIVE_INFINITY;
      distTo[s] = 0.0;

      pq.insert(s, 0.0);
      while (!pq.isEmpty())
         relax(G, pq.delMin())
   }

   private void relax(EdgeWeightedDigraph G, int v)
   {
      for(DirectedEdge e : G.adj(v))
      {
         int w = e.to();
         if (distTo[w] > distTo[v] + e.weight())
         {
            distTo[w] = distTo[v] + e.weight();
            edgeTo[w] = e;
            if (pq.contains(w)) pq.change(w, distTo[w]);
            else                pq.insert(w, distTo[w]);
         }
      }
   }
```

O(V)

O(E)

Dijkstra

(Asume que todos los pesos son no-negativos)

ALGORITHM 4.9 Dijkstra's shortest-paths algorithm

```
public class DijkstraSP
{
   private DirectedEdge[] edgeTo;
   private double[] distTo;
   private IndexMinPQ<Double> pq;

   public DijkstraSP(EdgeWeightedDigraph G, int s)
   {
      edgeTo = new DirectedEdge[G.V()];
      distTo = new double[G.V()];
      pq = new IndexMinPQ<Double>(G.V());
      for (int v = 0; v < G.V(); v++)
         distTo[v] = Double.POSITIVE_INFINITY;
      distTo[s] = 0.0;

      pq.insert(s, 0.0);
      while (!pq.isEmpty())
         relax(G, pq.delMin())
   }

   private void relax(EdgeWeightedDigraph G, int v)
   {
      for(DirectedEdge e : G.adj(v))
      {
         int w = e.to();
         if (distTo[w] > distTo[v] + e.weight())
         {
            distTo[w] = distTo[v] + e.weight();
            edgeTo[w] = e;
            if (pq.contains(w)) pq.change(w, distTo[w]);
            else                pq.insert(w, distTo[w]);
         }
      }
   }
```

O(V)

O(Elog(V))

O(E)

Dijkstra

(Asume que todos los pesos son no-negativos)

ALGORITHM 4.9 Dijkstra's shortest-paths algorithm

```
public class DijkstraSP
{
   private DirectedEdge[] edgeTo;
   private double[] distTo;
   private IndexMinPQ<Double> pq;

   public DijkstraSP(EdgeWeightedDigraph G, int s)
   {
      edgeTo = new DirectedEdge[G.V()];
      distTo = new double[G.V()];
      pq = new IndexMinPQ<Double>(G.V());

      for (int v = 0; v < G.V(); v++)
         distTo[v] = Double.POSITIVE_INFINITY;
      distTo[s] = 0.0;

      pq.insert(s, 0.0);
      while (!pq.isEmpty())
         relax(G, pq.delMin())
   }

   private void relax(EdgeWeightedDigraph G, int v)
   {
      for(DirectedEdge e : G.adj(v))
      {
         int w = e.to();
         if (distTo[w] > distTo[v] + e.weight())
         {
            distTo[w] = distTo[v] + e.weight();
            edgeTo[w] = e;
            if (pq.contains(w)) pq.change(w, distTo[w]);
            else                pq.insert(w, distTo[w]);
         }
      }
   }
```

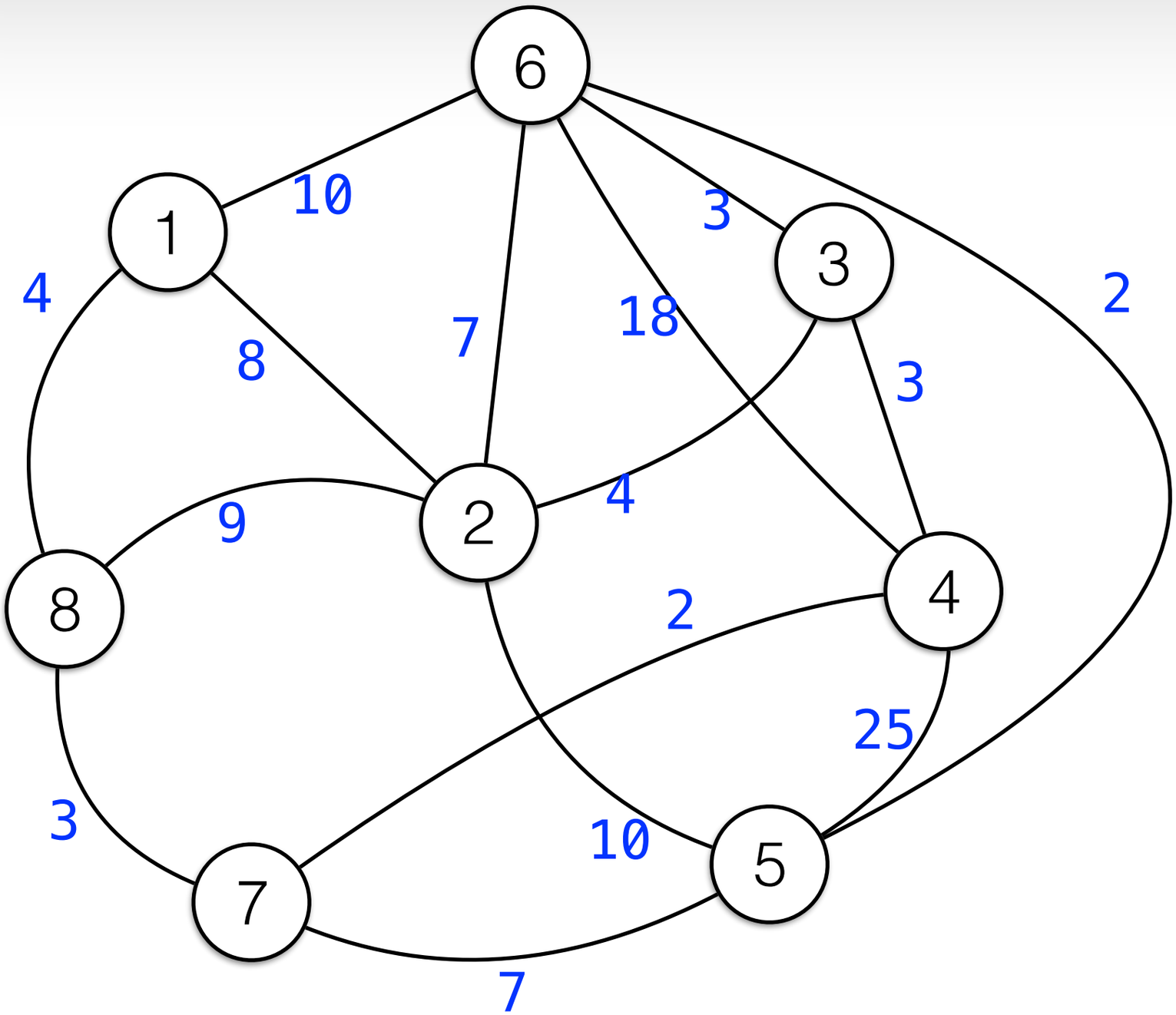
O(V + Elog(V))

Dijkstra

O(E + Vlog(V))

Algoritmo de Dijkstra

Do it yourself!



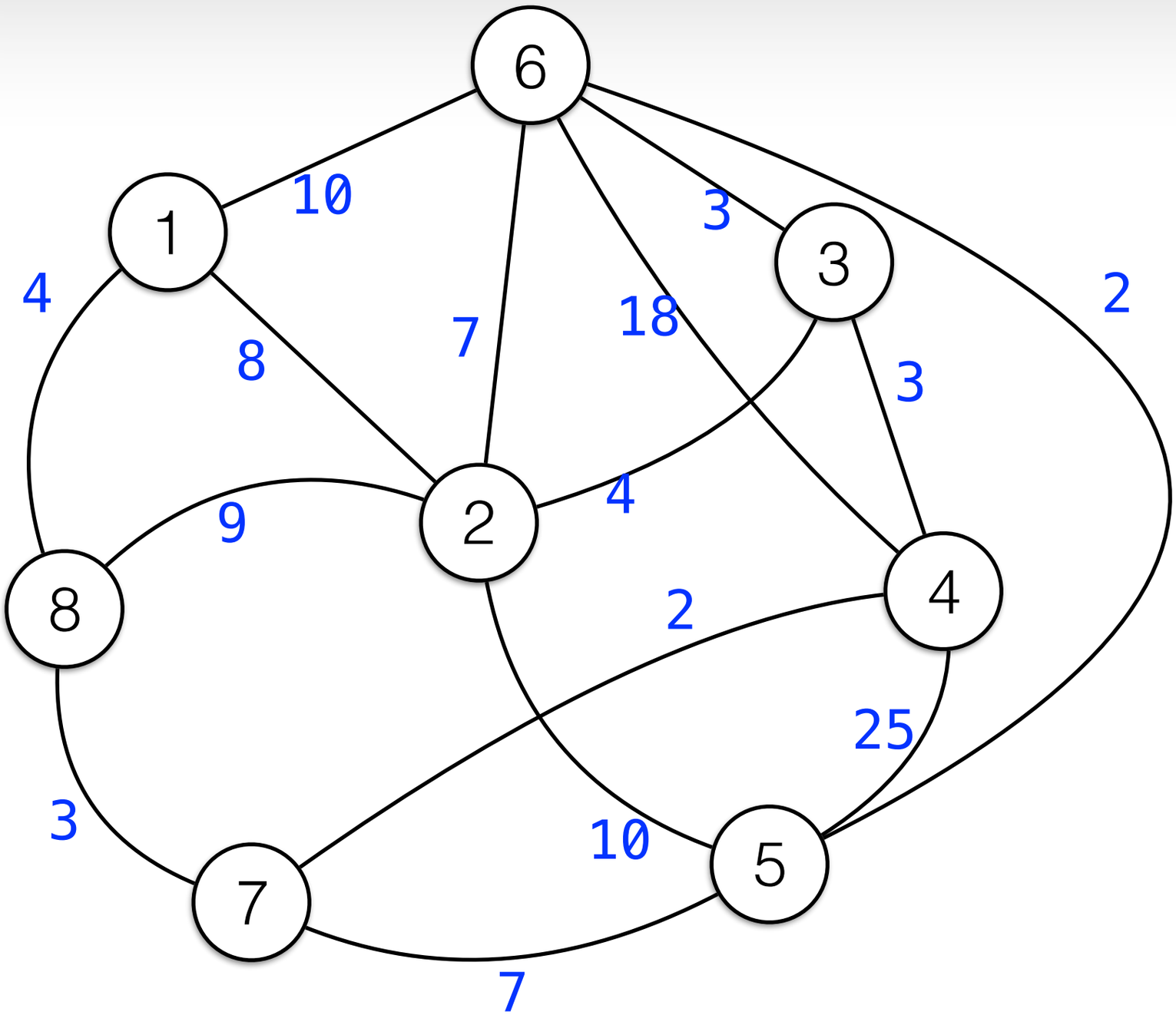
`s = 1`

	$\delta(s,v)$	$\pi(v)$
`v`	`distTo[v]`	`edge[v]`
0		
1		
2		
3		
4		
5		
6		
7		
8		

Algoritmo de Dijkstra

Do it yourself!

s = 4



v	distTo[v]	edge[v]
0		
1		
2		
3		
4		
5		
6		
7		
8		

$\delta(s, v)$ $\pi(v)$

Lectura adicional

Capítulo 24