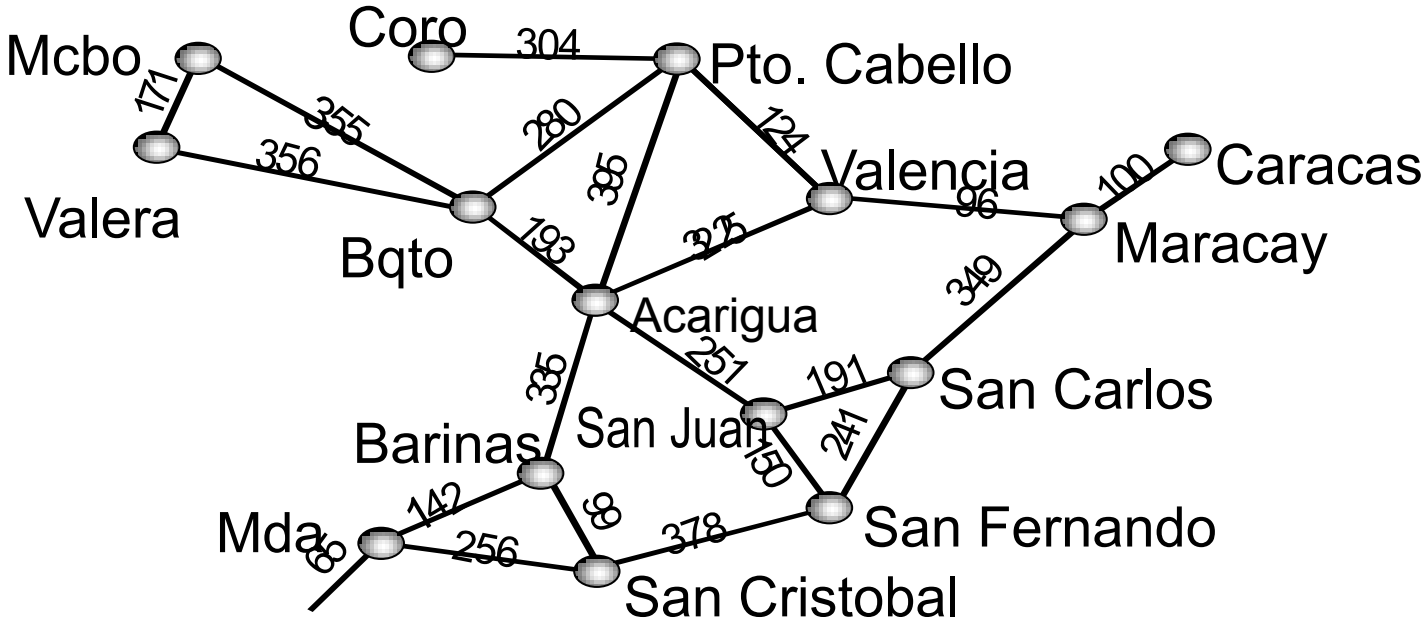
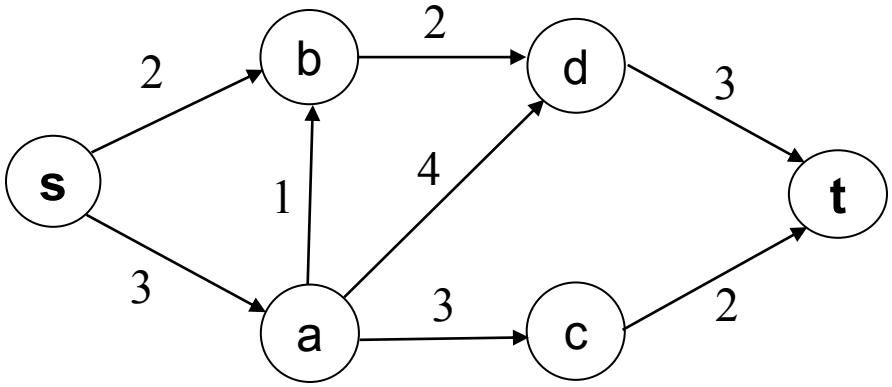
Caminos más cortos: algoritmo de Dijkstra, algoritmo de Bellman-Ford, Caminos más cortos en grafos dirigidos, algoritmo de Floyd-Warshall y algoritmo de Johnson.

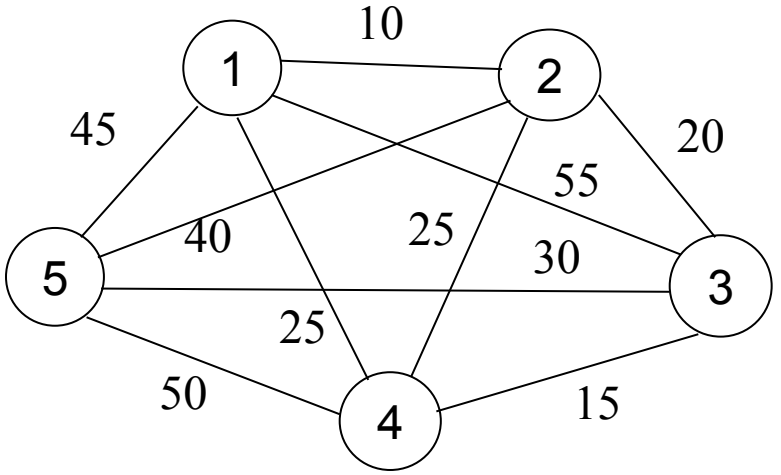
Jose Aguilar

Caminos más cortos

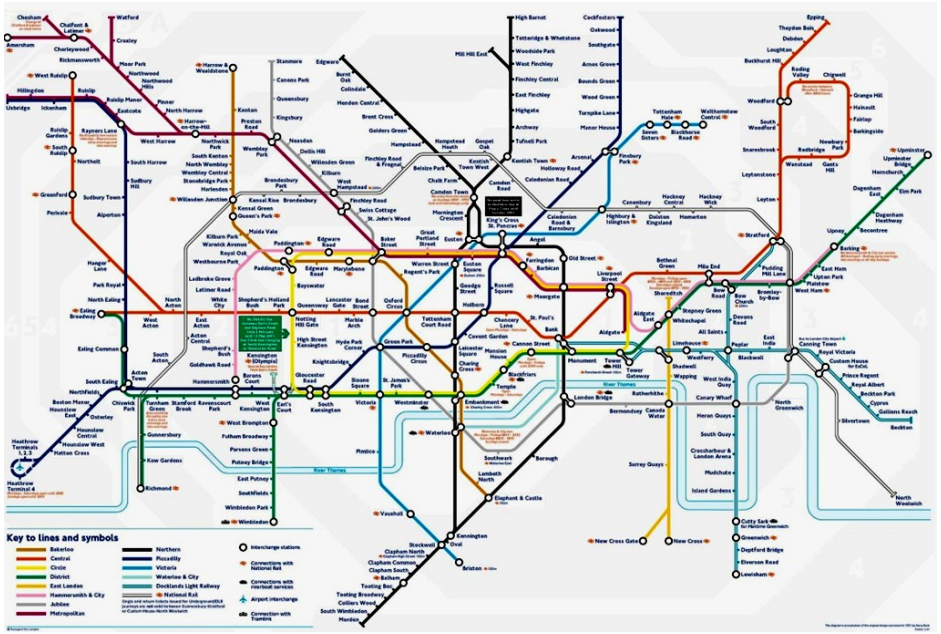
- **Coste de un camino**: suma de los costes de las aristas por las que pasa.
- **Problemas de caminos mínimos:**
  - Camino mínimo entre dos nodos, **v** y **w**.
  - Caminos mínimos entre un nodo **v** y todos los demás.
  - Caminos mínimos entre todos los pares de nodos.







Algoritmos de caminos más cortos



Problema del Camino más Corto

Definición:

Dado un grafo dirigido G=(V,E) con una función de pesos w: E$\rightarrow$R, el peso de un camino $p=<v_0, v_1, \ldots, v_k>$ es la suma de los pesos de los arcos:

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

el camino mínimo será:

$\delta(u,v) = \min \{w(p): u \sim> v\}$, si hay camino de *u* a *v*,

En otro caso será ∞.

Caminos más cortos desde un nodo dado

- Dado un digrafo G = (N, A) **etiquetado y conexo** con pesos w: A → ℜ. Para un camino p = ⟨n0, n1, ..., nk⟩ , w(p)= sumatoria de i=1 hasta k de w(ni-1, ni),

- Se define el **camino con el peso mínimo** de u a v como:
-

δ (u, v) = min { w(p) : longitud del camino p de u a v }, si ∃ p de u a v,
sino ∞

- Los pesos pueden ser cualquier medida: **distancia, tiempo, costos, etc.** que se desea minimizar.

Caminos más cortos

- El recorrido BPA halla los caminos con el menor número de aristas desde el vértice inicial. Por tanto, BPA halla los caminos más cortos asumiendo que las aristas tienen el mismo peso.

En muchas aplicaciones (p.e. redes de transporte) las aristas tienen peso diferentes.

Caminos más cortos

Sea un digrafo etiquetado DG={N, A} cuyas aristas tienen pesos no negativos

Problema: encontrar todos los caminos mínimos desde un nodo origen hacia todos los demás nodos de DG

En este apartado se revisarán los algoritmos de las rutas más cortas de **Dijkstra, Floyd, Bellman-Ford**

Todos los algoritmos usan la desigualdad del triángulo, es decir, tratan de probar si

peso(u,v) > peso(u,i) + peso(i,v).

Variantes del Problema

- Problema del camino más corto desde un nodo s $\subset$ V a cada uno v $\subset$ V-S.

- Problema de todos los caminos cortos a un destino simple (a un único destino). Solución: se invierten los arcos y se convierte en el problema inicial (invertir la salida).

- Problema del camino más corto entre un par de nodos dados.

- Problema del camino más corto para todos los pares de nodos: Solución: Ejecutar el algoritmo para todos los pares desde cada nodo.

Caminos más cortos

- El problema de los caminos más cortos desde un vértice.
  - Algoritmo de Dijkstra.

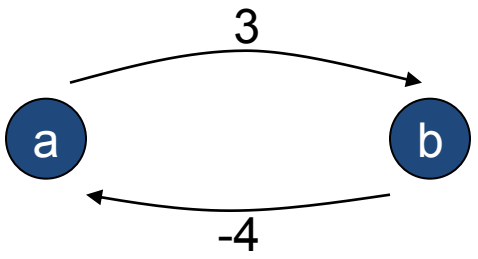
- El problema de los caminos más cortos entre todos los pares de vértices.
  - Algoritmo de Floyd.

- Cierre transitivo.
  - Algoritmo de Warshall

Arcos Negativos

Los caminos más cortos están bien definidos ssi no existen ciclos negativos.

De otra manera no estará bien definido, pues cada vez que demos vuelta a ciclo el camino será más corto.



$\delta(a,b) = -\infty$

Caminos más cortos

- El C+C está siempre bien definido por ser un dag.
- Se relajan los nodos en base a un ordenamiento topológico de los mismos.
- Si hay un camino desde u hasta v en G, entonces u precede a v en el **orden topológico**
- Los algoritmos de Bellman-Ford y de Dijkstra usan los siguientes algoritmos como **inicialización y relajación**

```
Inicialización(G, s)
para cada vértice v en G
        d[v] = INFINITO
        padre[v] = NULO
d[s] = 0
```

Caminos Cortos y Relajación

La técnica de relajación es esencial para comprender los algoritmos de caminos más cortos.

La **Relajación** es el método mediante el cual se decrece la cota superior del peso del camino actual hasta que iguale a la cota del camino más corto.

Relajación

Relajación: ∀ v ∈ N, se mantiene un atributo d(v) que es el límite superior de los pesos del C+C de s a v, d(v) se denomina el estimado del peso del C+C

```
Relajamiento(u,v)
si d[v] > d[u] + peso(u,v)
        d[v] = d[u] + peso(u,v)
        padre[v] = u
```

Propiedades de la relajación: Sea G=(N, A) un digrafo etiquetado con w: A → ℜ

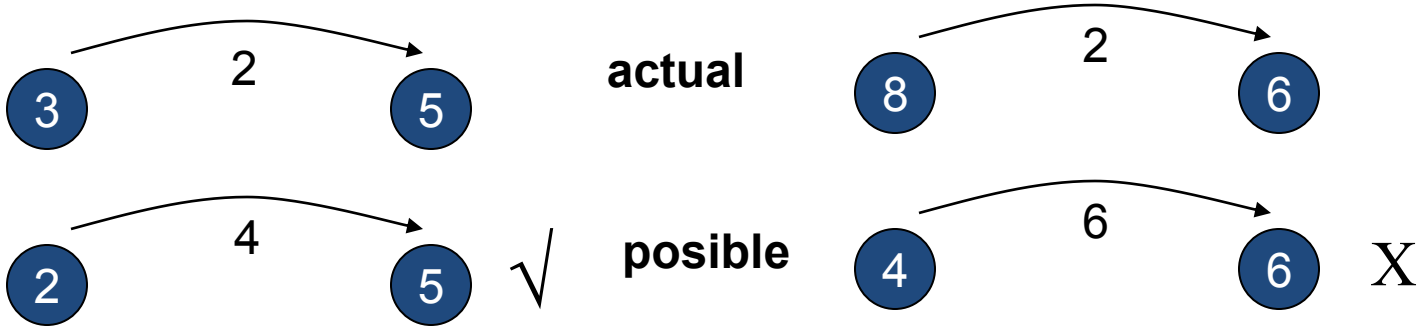
- Sea (u, v) ∈ A luego de la relajación del arco (u, v) por medio de relajar( ) se tiene que **d(v) ≤ d(u) + w(u, v)**
- Sea s ∈ N el nodo fuente y G iniciado con iniciar(s), entonces d(v) ≥ δ(s, v) ∀ v ∈ N y esta invariante se mantiene sobre cualquier secuencia de relajación de los arcos de G. Mas aún, **cuando d(v) alcanza su límite inferior δ(s, v), este nunca cambia.**

Relajación

INITIALIZATION(G,s)
1 for each vertex v $\subset$ V[G]
2 do d[v] $\leftarrow \infty$
3 Π[v] $\leftarrow$ NIL
4 d[s] $\leftarrow$0

Relajar es mejorar la distancia de un nodo a otro. Dado (u,v) se pregunta si el camino calculado hasta ahora hasta *v* es mejor que el camino hasta u y luego w(u,v).

RELAX(u,v,w)
1 if d[v] > d[u] + w(u,v)
2 then d[v] $\leftarrow$ d[u] + w(u,v)
3 Π[v] $\leftarrow$ u



UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Relajación

Versión 1.0

relajar(Nodo: u, v, Real: w)

{pre: n > 0 ∧ (u, v) ∈ A } {pos: n > 0 }

1	Si (v.D() > u.D() + w(u, v)) entonces
	  v.D(u.D() + w(u, v))
	  v.Padre(u)
	fsi
		--D(), Padre(): Operaciones de la clase Nodo.

2	Regrese	

Caminos más cortos

Seudocódigo de las rutas más cortas en un dag.

```
Rutas-cortas-dags(G, s)
Ordenar topológicamente G
Inicialización(G,s)
para cada vértice u, ordenado topológicamente
    para cada vértice v que es adyacente a u Relajamiento(u,v)
```

Algoritmo para caminos mínimos en dag etiquetado

Versión 1.0

{C+Cdag(Nodo: s)

{pre: n > 0 ∧ s ∈ N } {pos: n > 0 }

1	lot = ordenTopologico( )	Θ(N+A)	-iniciar(), relajar(), ordenTopologico(): Operaciones de la clase Grafo. -lot: ListaDe [Entero]. Lista de nodos en orden topológico
2	iniciar(s)	Θ(N)	
3	[[relajar(u, v, w)] v ∈ listaAdy(u)] u ∈ lot		

$T(n) = \Theta(N + A)$.

Algoritmo de Dijkstra

Resuelve el camino mínimo para aquellos grafos dirigidos, con peso no negativo. Entonces se asume $w(u,v) \geq 0 \ \forall \ (u,v) \subset E$.

Al final del algoritmo se tiene que para todos los vértices $d[v] = \delta(s,v)$.

Algoritmo de Dijkstra

- Todos los $w(u, v) \geq 0 \ \forall \ u, v \in N$ y $(u, v) \in A$.

- El algoritmo mantiene un conjunto S de nodos cuyos pesos finales del C+C desde s han sido determinados.
  Esto es, $\forall \ v \in S$ se tiene $d(v) = \delta(s, v)$.

- El algoritmo repetidamente selecciona un nodo $u \in N - S$ con el mínimo C+C estimado, inserta u en S y relaja todos los arcos que salen de u.

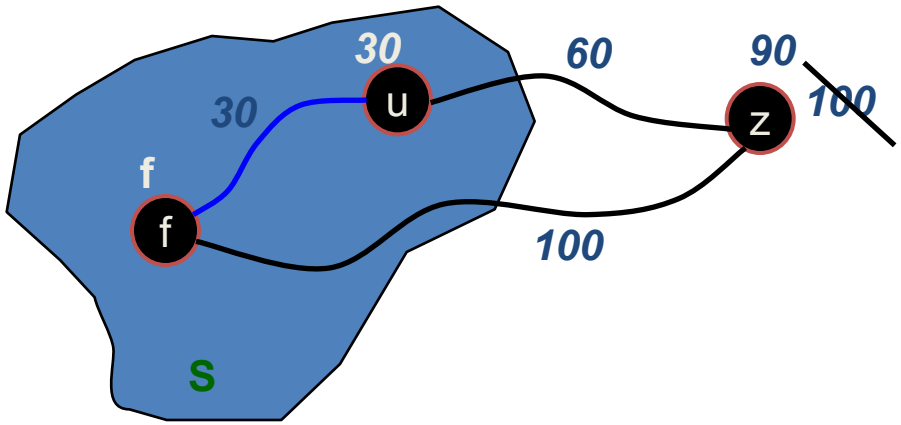
- La cola por prioridad C contiene todos los nodos en N – S cuya clave es d(u).

- Es un algoritmo incremental.

Algoritmo de Dijkstra

- Cuando se añade un vértice al conjunto S, el valor de d[v] contiene la distancia de *f* a v.
- Cuando se añade un nuevo vértice *u* al conjunto S, es necesario comprobar si *u* es una mejor ruta para sus vértices adyacentes *z* que están en el conjunto C.
- Para ello se actualiza d con la ***relajación*** de la arista (*u*, *z*):

$$d[z] = \min(d[z], d[u] + w[u, z])$$



Algoritmo de Dijkstra

Dijkstra(G, f)

S = { f }, C = { V }

d[f] = 0

d[u] = ∞ $\quad\quad$ $\forall$ u $\neq f$

while (C $\neq \emptyset$) {

  seleccionar vértice $w \in$ C / d[w] es mínimo

  S = S $\cup$ {w}, C = C - {w}

  for cada vertex $v \in$ Adyacente[w] {

    if (d[v] > d[w] + w(w, v))

      d[v] = d[w] + w(w, v)

  }

}

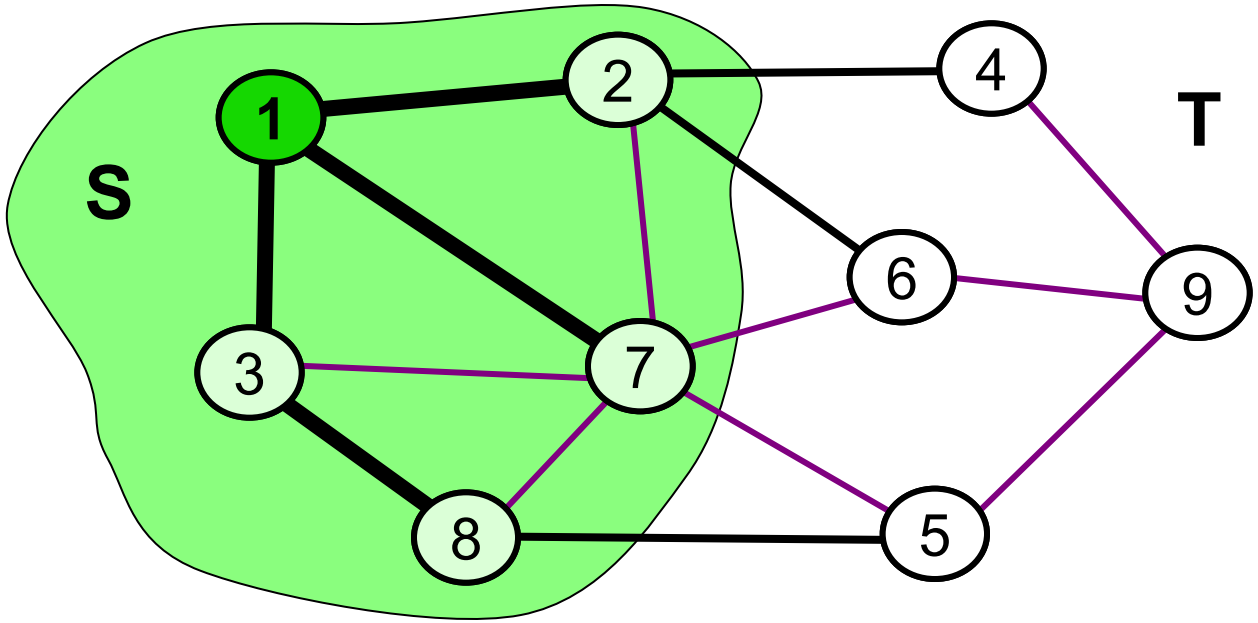
Algoritmo de Dijkstra

- Supongamos un grafo **G**, con pesos positivos y un nodo origen **v**.

- El algoritmo trabaja con dos conjuntos de nodos:
  - **Escogidos: S**. Nodos para los cuales se conoce ya el camino mínimo desde el origen.
  - **Candidatos: T**. Nodos pendientes de calcular el camino mínimo, aunque conocemos los caminos mínimos desde el origen pasando por nodos de **S**.

Algoritmo de Dijkstra

- **Camino especial:** camino desde el origen hasta un nodo, que pasa sólo por nodos escogidos, **S**.



- **Idea:** En cada paso, coger el nodo de **T** con menor distancia al origen. Añadirlo a **S**.
- Recalcular los caminos mínimos de los demás candidatos, pudiendo pasar por el nodo cogido.

Algoritmo de Dijkstra

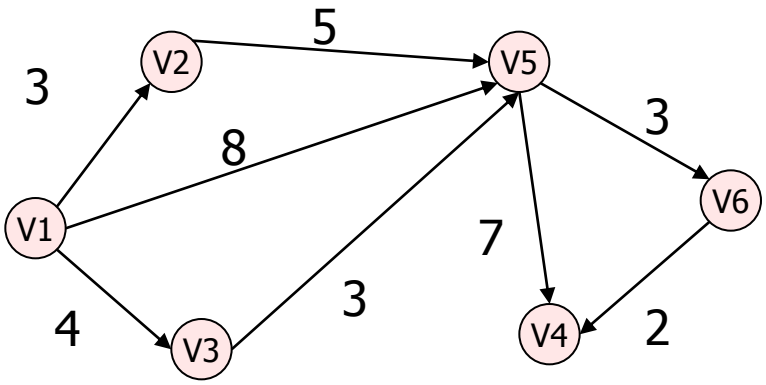
- **Inicialización: S**= {1}, **T**= {2, ..., n}, caminos especiales mínimos = caminos directos.
- **Repetir** n-1 veces:
  - Seleccionar el nodo **v** de **T** con el camino especial más corto.
  - **Proposición:** el camino mínimo para este nodo **v**, coincide con su camino especial.
  - Recalcular los caminos especiales para los nodos de **T**, pudiendo pasar por **v**.

EJEMPLO DE DIJKSTRA

Pasando por V2, Distacia de V1 a V5 seria 8, no hay mejora

Pasando por V3, Distacia de V1 a V5 seria 7, CAMBIAR

De V1 AL RESTO



Escogidos	Vertice Evaluado	V1 D[0]	V2 D[1]	V3 D[2]	V4 D[3]	V5 D[4]	V6 D[5]
V1							
V1,V2							
V1,V2,V3							
V1,V2,V3,V5							
V1,V2,V3,V5,V6							

1. D[] se incializa con F.P. de adyacentes al origen

2. Escoger vertice Vk que no haya sido escogido, con la menor distancia del Vevaluado a Vk

3. Revisar si alguna distancia puede ser mejorada pasando por Vk

Repetir hasta k se hayan visitado todos los vertices

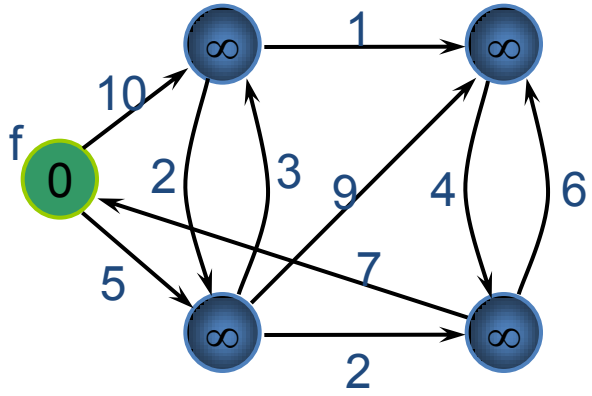
Algoritmo de Dijkstra

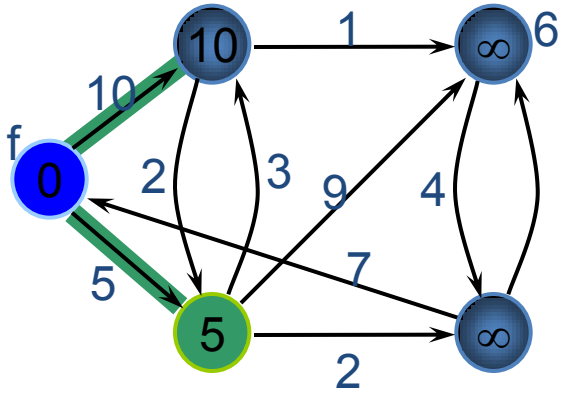
DIJKSTRA(G,s)

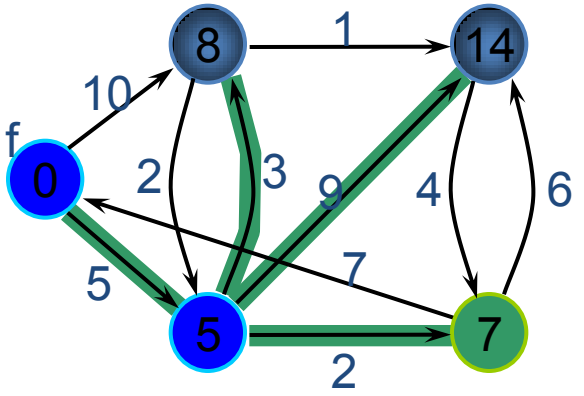
- Inicialización(G,s)
- S es el conjunto vacío
- Meter los vértices u a una cola de prioridad Q de acuerdo a d[u]
- mientras Q no este vacía
  - extraer el minimo de Q en u
  - S = S union {u}
  - para cada v adyacente a u relajamiento(u,v)

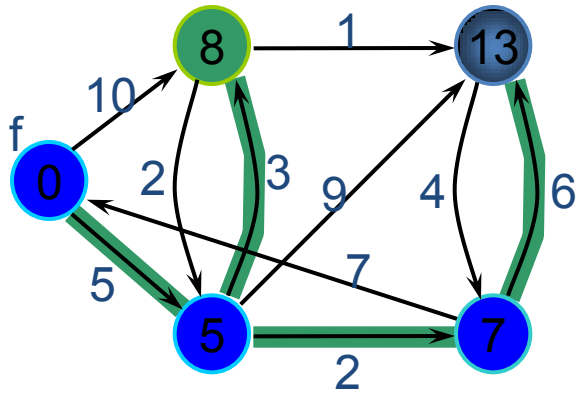
La implementación del algoritmo de Dijkstra es muy similar a la del algoritmo de Prim.

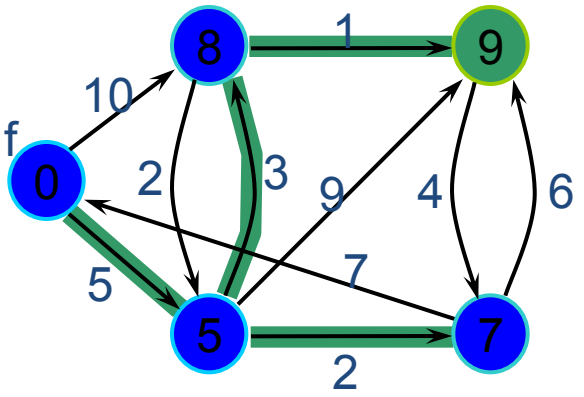
Algoritmo de Dijkstra

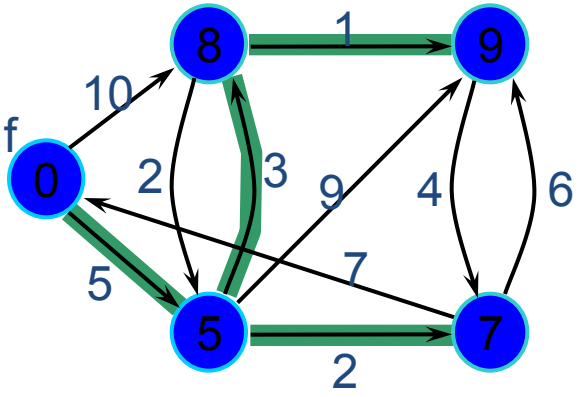












Algoritmo de Dijkstra

Complejidad:

- Con matrices de adyacencia: $O(n^2)$.
- Con listas de adyacencia: $O(n^2)$.
- Se puede usar estructuras de datos más eficientes, como una cola de prioridad para buscar el vértice con menor d[v] se consigue O(e*log n).

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Algoritmo de Dijkstra

Versión 1.0

C+Cdijkstra(Nodo: s): Conjunto

{pre: n > 0 ∧ s ∈ N } {pos: n > 0 }

1	iniciar(s)	-iniciar(), relajar(): Operaciones de la clase Grafo.
2	[C.entrar(u)] u ∈ N	
3	(¬C.vacíaColaP()) [u = C.min( )	-C: ColaPrioridad. Cola de prioridad que contiene los nodos que aún no han sido pasados al C+C.
	C.salir( )	
	S = S U { u }	-S: Conjunto. Contiene el C+C.
	[relajar(u, v, w)] v ∈ listaAdy(u)	-entrar(), vacíaColaP(), min(), salir(). Operaciones de la clase ColaPrioridad
	]	
4	regrese S	

Si C se implanta como un vector, entonces T(n) = O(N2 + A) = O(N2)

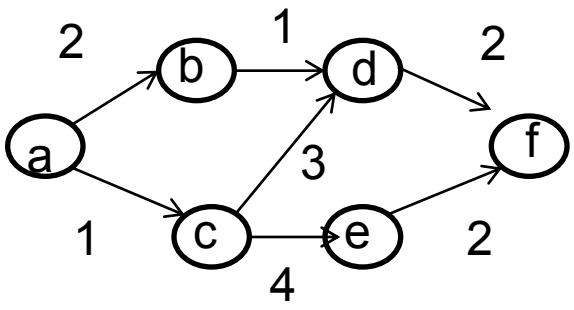
Si C se implanta como un montículo binario, entonces T(n) = O(A lg N)

Si C se implanta como un montículo binomial, entonces T(n) = O(N lg N + A)

Algoritmo de Dijkstra

Origen el nodo 1

Paso	v	C	D	P



Caminos más cortos

- *Problema*: Hallar los caminos mínimos entre cualquier par de nodos de un grafo.

- Se puede usar el algoritmo de Dijkstra tomando cada vértice como fuente.

- Existe una manera más directa: *algoritmo de Floyd*.

Algoritmo de Floyd-Warshall

Caminos más cortos a todos los pares de nodos

Algoritmos que encuentra las distancias más cortas entre todos los pares de vértices.

Este problema se puede resolver **invocando n veces para**
pesos no negativos a Dijkstra

Cola por prioridad basada en Vector T(n)=O(N3),
Montículo binario T(n)= O(N A lg N),
Montículo Fibonacci T(n)= O(N2 lg N + N A))

si hay pesos negativos a Bellman-Ford

T(n)= O(N2 A) y para digrafos densos T(n)= O(N4)

Caminos más cortos a todos los pares de nodos

Para mejorar esos T(n) se tratará el problema desde otro punto de vista

Usando multiplicación de matrices

Algoritmo de Floyd-Warshall:

almacena en cada iteración el mejor camino entre el que pasa por el nodo intermedio k y el que va directamente del nodo i al nodo j.

Algoritmo de Floyd-Warshall

El método consiste en hacer iteraciones del proceso de relajación para cada arista en el grafo, tomando en cada paso un vértice intermedio diferente.

La manera más simple de implementarlo es por medio de **tres ciclos anidados** que iteran sobre un grafo representado por una matriz de adyacencia.

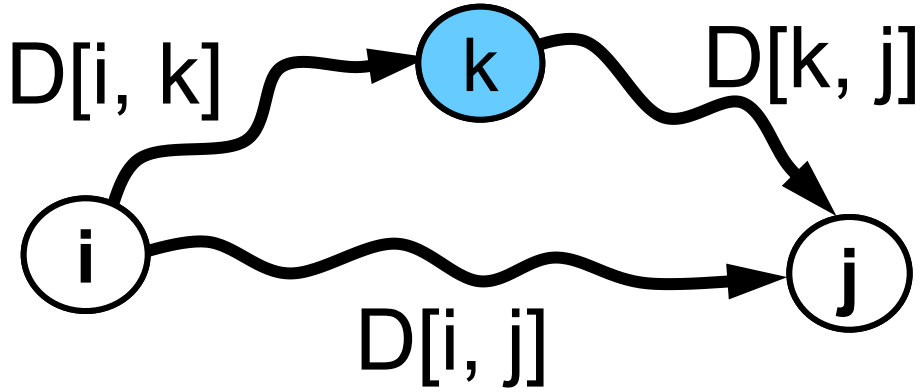
Algoritmo de Floyd

¿En qué se basa el algoritmo de Floyd?

- En cada paso **k**, la matriz **D** almacena los caminos mínimos entre todos los pares pudiendo pasar por los **k** primeros nodos.
- **Inicialización: D** almacena los caminos directos.
- **Paso 1:** Caminos mínimos pudiendo pasar por el 1.
- ...
- **Paso n:** Caminos mínimos pudiendo pasar por cualquier nodo $\rightarrow$ Lo que buscamos.

- En el paso **k**, el nodo **k** actúa de pivote.

Algoritmo de Floyd



- **Camino mínimo entre i y j, en el paso k:**
  - Sin pasar por **k**: D[i, j]
  - Pasando por **k**: D[i, k] + D[k, j]
  - Nos quedamos con el menor.

- **P: array** [1..n, 1..n] **de** entero. **P[i, j]** indica un nodo intermedio en el camino de **i** a **j**.

$$i \rightarrow \ldots \rightarrow P[i, j] \rightarrow \ldots \rightarrow j$$

Algoritmo de Floyd-Warshall

seudocódigo:

```
n = numero de vértices
para k = 1 to n
    para i = 1 to n
        para j = 1 to n
            si w(i,j) > w(i,k) + w(k,j)
              w(i,j) = w(i,k) + w(k,j)
              padre(i,j) = k
```

.

Formalización Algoritmo de Floyd

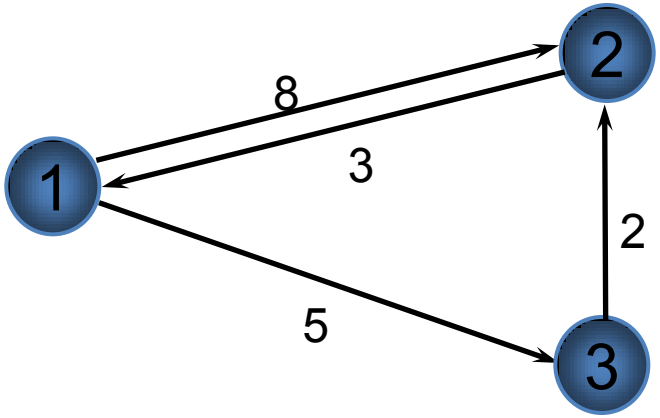
- Utiliza una matriz $A_k[i][j]$, que contiene el camino más corto que pasa por los primeros k primeros vértices.
- Inicialmente $A_k[i][j] = C[i][j] \ \forall \ i \neq j$. Si no hay arista de i a j $C[i][j] = \infty$ y los elementos diagonales se ponen a 0.
- En la iteración k (nodo k como pivote) se calcula, para cada camino de v a w, si es más corto pasando por k aplicando:

  $A_k[i][j] = \min (A_{k-1}[i][j] , A_{k-1}[i][k] + A_{k-1}[k][j]), \ \forall \ i \neq j.$
- Como no varía la fila y la columna k en la iteración k, sólo es necesario una matriz A.

Formalización Algoritmo de Floyd

```
Floyd (n, C, A)
{
    A[i][j] = C[i][j]
    A[i][j] = 0
    for (k =1; k <=n; k ++)
        for (i =1; i <=n; i ++)
            for (j =1; j <=n; j ++)
                if (A[i][k] + A[k][j] < A[i][j] )
                    A[i][j] = A[i][k] + A[k][j]
}
```

Algoritmo de Floyd



C[*i*][*j*]	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

A_1[*i*][*j*]	2	3
2	0	8
3	2	0

A_2[*i*][*j*]	1	3
1	0	5
3	5	0

A_3[*i*][*j*]	1	2
1	0	7
2	3	0

Principio recursividad Algoritmo de Floyd-Warshall

- Un nodo intermedio de un camino simple p = ⟨n1, n2, ..., nh⟩ es cualquier nodo de p diferente de n1 y nh

- Si k es un nodo intermedio de p, entonces p puede dividirse en p' y p'', donde
  - p' es el C+C de i a k y
  - p'' el C+C de k a j.

$$i \longrightarrow p' \longrightarrow k \longrightarrow p'' \longrightarrow j$$

.

Principio recursividad Algoritmo de Floyd-Warshall

Sea $d^{(k)}_{ij}$ el peso del C+C de i a j con todos los nodos intermedios en {1, 2, ..., k}.

Si k = 0, no hay nodos intermedios por lo que $d^{(0)}_{ij} = w(i, j)$.

Se define la recursión

$$d^{(k)}_{ij} = \begin{cases} w(i,j) & \text{si } k=0 \\ \min(d^{(k-1)}_{ij},\ d^{(k-1)}_{ik} + d^{(k-1)}_{kj}) & \text{si } k\geq 0 \end{cases}$$

La matriz $D^{(n)}$ contiene los pesos de los C+C entre todos los pares de nodos de G, pero no contiene los C+C.

Algoritmo de Floyd-Warshall

	Versión 1.0 FloydWarshall(Arreglo(n x n)De [Real]: W): Arreglo(n x n)De [Real] {pre: n > 0 } {pos: n > 0 }	
1	n = W.filas()	-i, j, k: Entero. Indican nodos del grafo y son los subíndices de los lazos. -D(n): Arreglo(n x n)De [Real]. Matriz con los pesos de los C+C actuales.
2	$D^{(0)} = W$	
3	[[[$D^{(k)}(i, j) = \min(D^{(k-1)}(i, j), D^{(k-1)}(i, k) + D^{(k-1)}(k, j)$)] j = 1, n] i = 1, n] k = 1, n regrese $D^{(n)}$	
4		

$$T(n) = \Theta(n3)$$

Algoritmo de Floyd

La complejidad del algoritmo es:

- Con matriz de adyacencia: $O(n^3)$.
- Para grafos dispersos es mejor usar la versión Dijkstra con lista de adyacencia que toma

  $O(ne \log n)$.

Algoritmo de Floyd

El algoritmo de Floyd **es muy versátil** a pesar de ser muy simple y se usa para encontrar la solución de problemas donde es necesario encontrar **el mejor camino basado en restricciones de carga máxima o de esfuerzo mínimo**.

En esos casos es necesario **modificar el proceso de relajación** para sustituirlo por uno que **elija en cada iteración la mejor solución en función de las restricciones**..

Algoritmo de Bellman-Ford

DESCRIPCIÓN DEL PROBLEMA

El algoritmo de Bellman-Ford genera el camino más corto en un grafo dirigido ponderado (en el que el peso de alguna de las aristas puede ser negativo).

El algoritmo de Dijkstra resuelve este mismo problema en un tiempo menor, pero requiere que los pesos de las aristas no sean negativos.

el algoritmo Bellman-Ford normalmente se utiliza cuando hay aristas con peso negativo

EXPLICACIÓN DEL ALGORITMO

- ***En el paso 0****, inicializamos todas las distancias o costos mínimos a infinito.*
- ***En el paso 1****, actualizamos el paso anterior, aplicando la relajación. En este caso ponemos la distancia de los nodos que tienen accesos directos al vértice 1, y se la sumamos a la distancia mínima acumulada que hay hasta el vértice.*
- ***En el paso 2,*** *al haber ya una distancia mínima acumulada desde los nodos 2 y 3 hasta 1, podemos actualizar las distancias mínimas de los nodos 4 y 5.*
- ***En los pasos sucesivos****, se van actualizando las distancias mínimas acumuladas (D) de los distintos vértices hasta 1.*

Final del algoritmo se da cuando no hay ningún cambio de un paso a otro, ya no se puede encontrar un camino más corto.

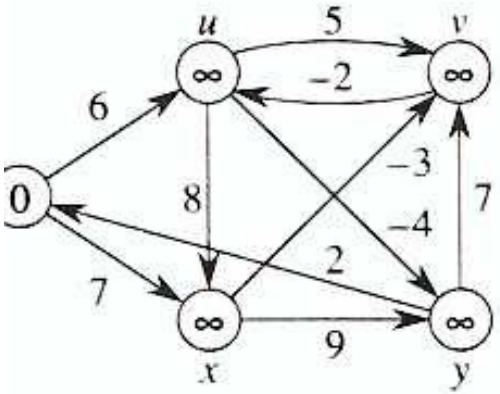
Algoritmo de Bellman-Ford

Se trata de resolver el mismo problema en el caso de pesos negativos y positivos.

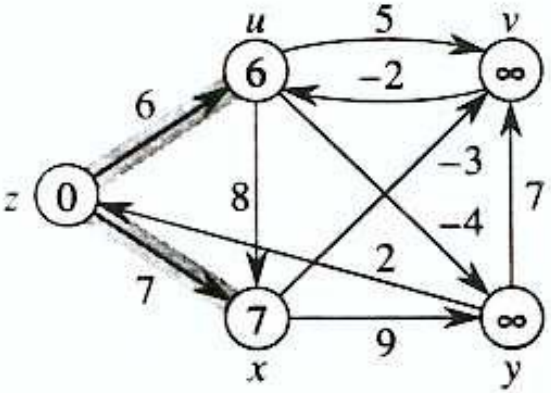
```
Int Bellman-Ford(G, w, s) {
        for (cada vértice v en V[G] ) {
                d [v] = infinito; /* -"MAX_INT" por ejemplo*/
                p [v] = NIL;
        }
        d [s] = 0;
        for (i=1 to |V [G]| -1 )
                for (cada arco (u,v) en E[G] )
                        Relax(u,v,w);
        for ( cada arco (u,v) en  E[G] )
                if(d [v] > d [u]+w(u,v))
                        return 0;  /* False; no existe camino mínimo */
        return 1;
}
```

El tiempo de ejecución es O(EV).

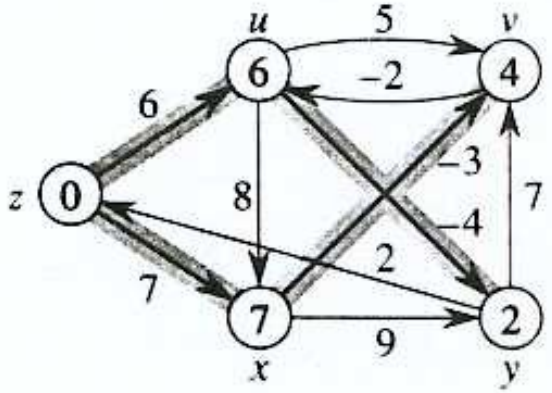
Ejemplo: Algoritmo de Bellman-Ford



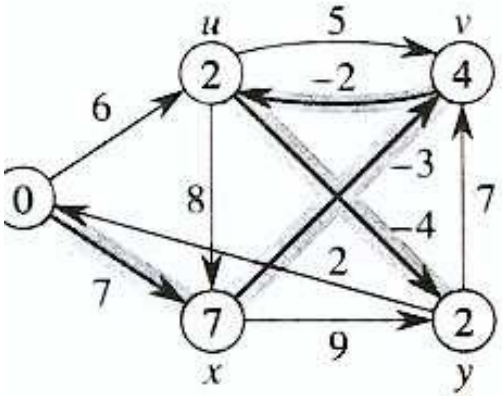
(a)



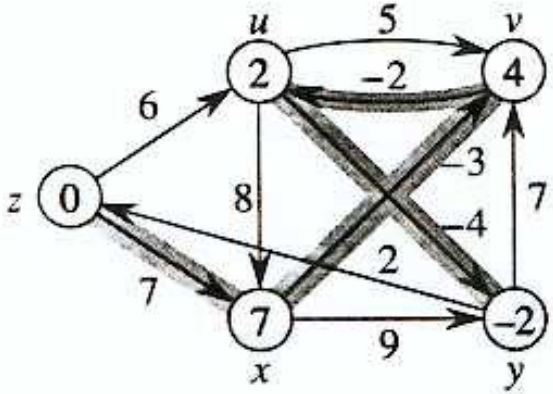
(b)



(c)



(d)



(e)

Caminos más cortos: Algoritmo de Bellman-Ford

```
BELLMAN-FORD(G,s)
Inicializacion(G,s)
para i = 1 hasta nvert – 1                                   Θ(V)
        para cada arista (u,v) en G relajamiento(u,v)        Θ(V)
para cada arista (u,v) en G                                  Θ(E)
        si d[u] > d[u] + peso(u,v) return FALSO  ;  /* False; no existe camino mínimo */
return VERDADERO
```

TOTAL: Θ(VE)

Al terminar de ejecutarse el algoritmo, las distancias más cortas serán almacenadas en el arreglo d y los caminos de s al resto de los vértices puede encontrarse por medio del algoritmo :

```
Camino(u,v)
si u = v imprime u
en caso contrario si padre[v] = NULO
        no existe camino de u a v
si padre[v] es diferente de NULO
        camino(u, padre[v])
        imprime v
```

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Algoritmo de Bellman-Ford

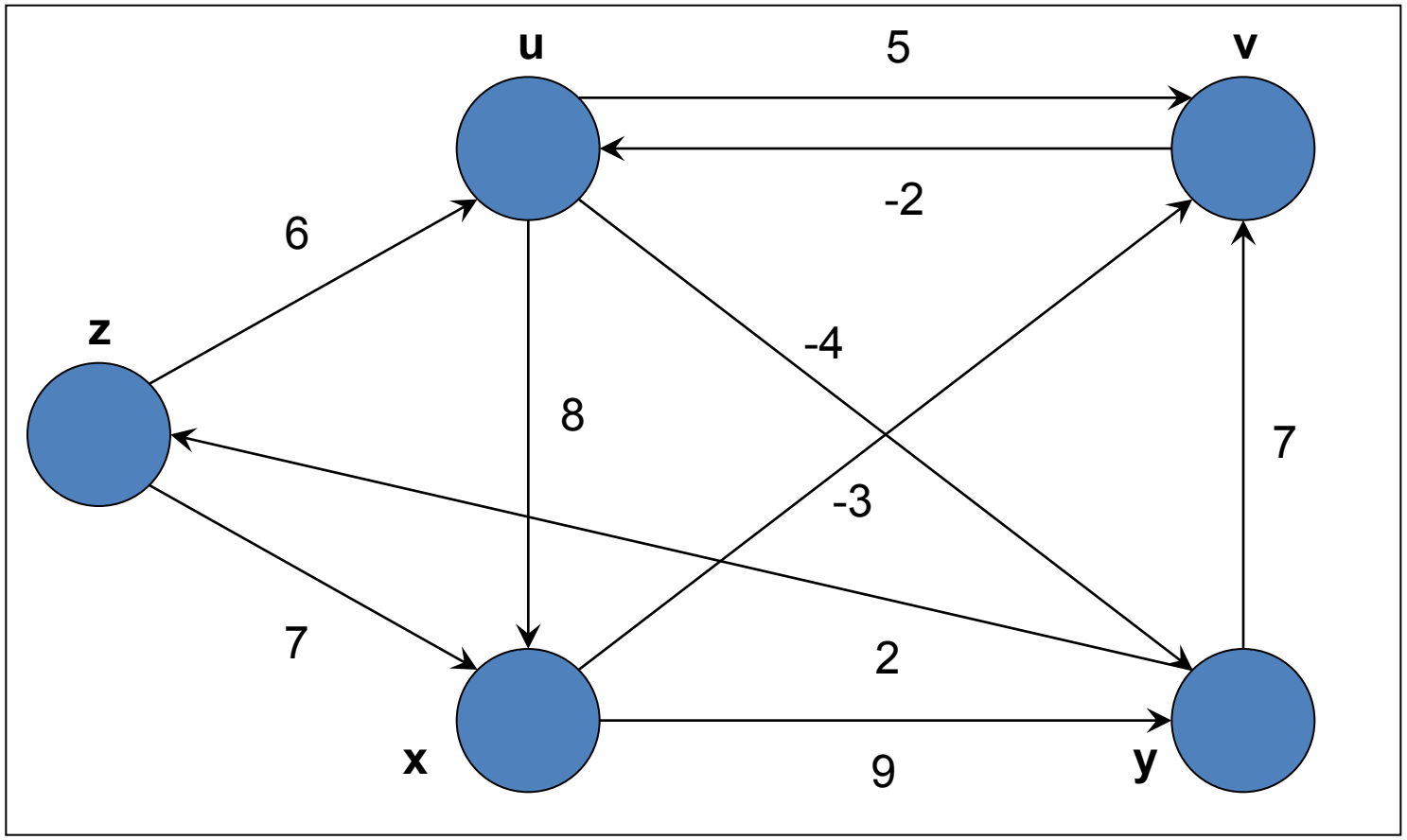
Versión 1.0

C+CBellmanFord(Nodo: s): Lógico

{pre: n > 0 ∧ s ∈ N } {pos: n > 0 }

1	iniciar(s) O(N)	-iniciar(), relajar(): Operaciones de la clase Grafo. -i: Entero. Subíndice. -D(), Padre(). Operaciones de la clase Nodo
2	[[relajar(u, v, w)] (u, v) ∈ A] i = 1, N – 1	
3	[Si (v.D() > u.D() + w(u, v)) entonces regresar Falso fsi	
4	] (u, v) ∈ A O(A) regrese Verdadero	

$$T(n) = O(N\,A)$$



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

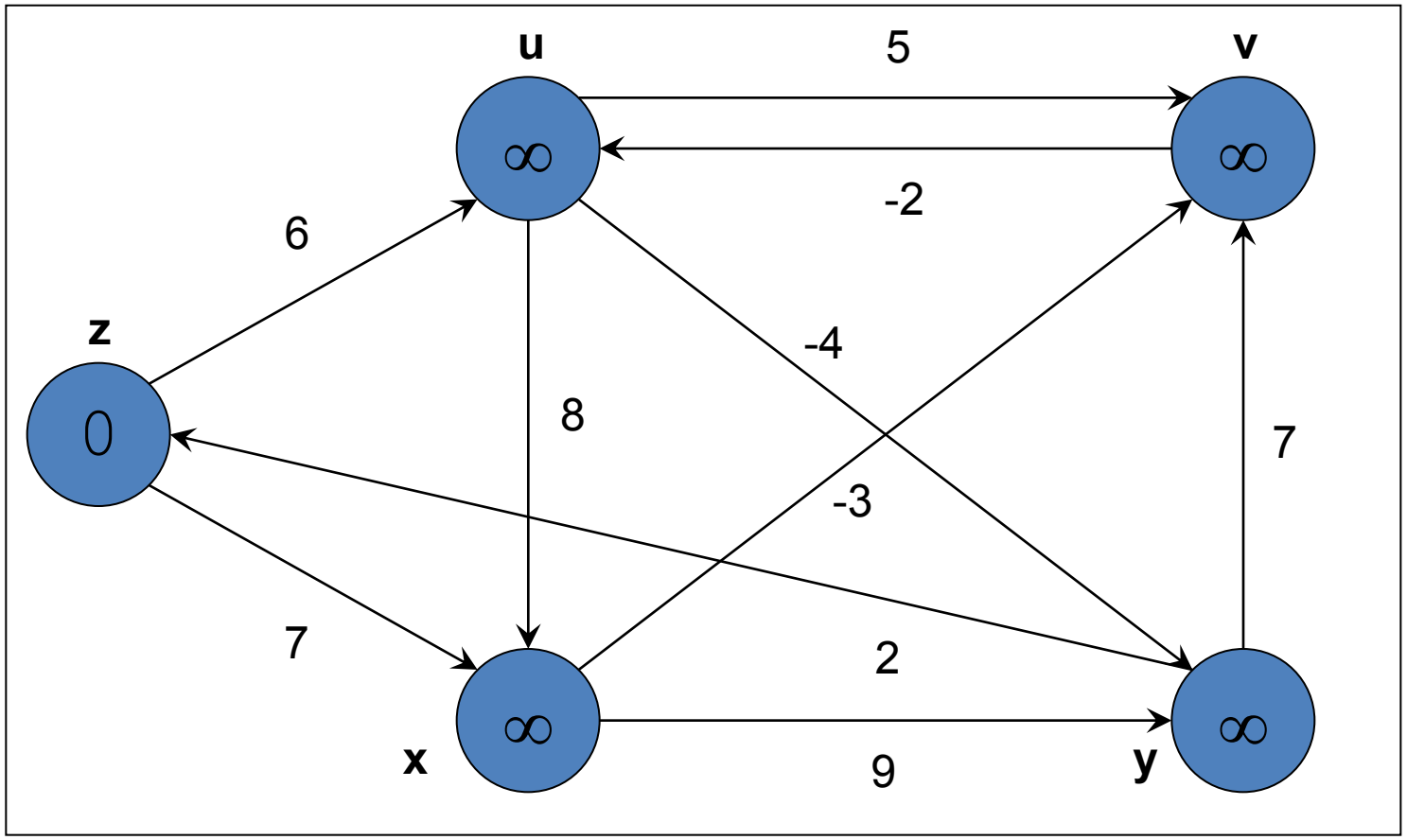
V [ ] = { u v x y z }

d [ ] = { _ _ _ _ _ }

Π [ ] = { _ _ _ _ _ }

Paso 0.0

Encontrar el camino más corto del Vértice z a cada uno de los otros Vértices.



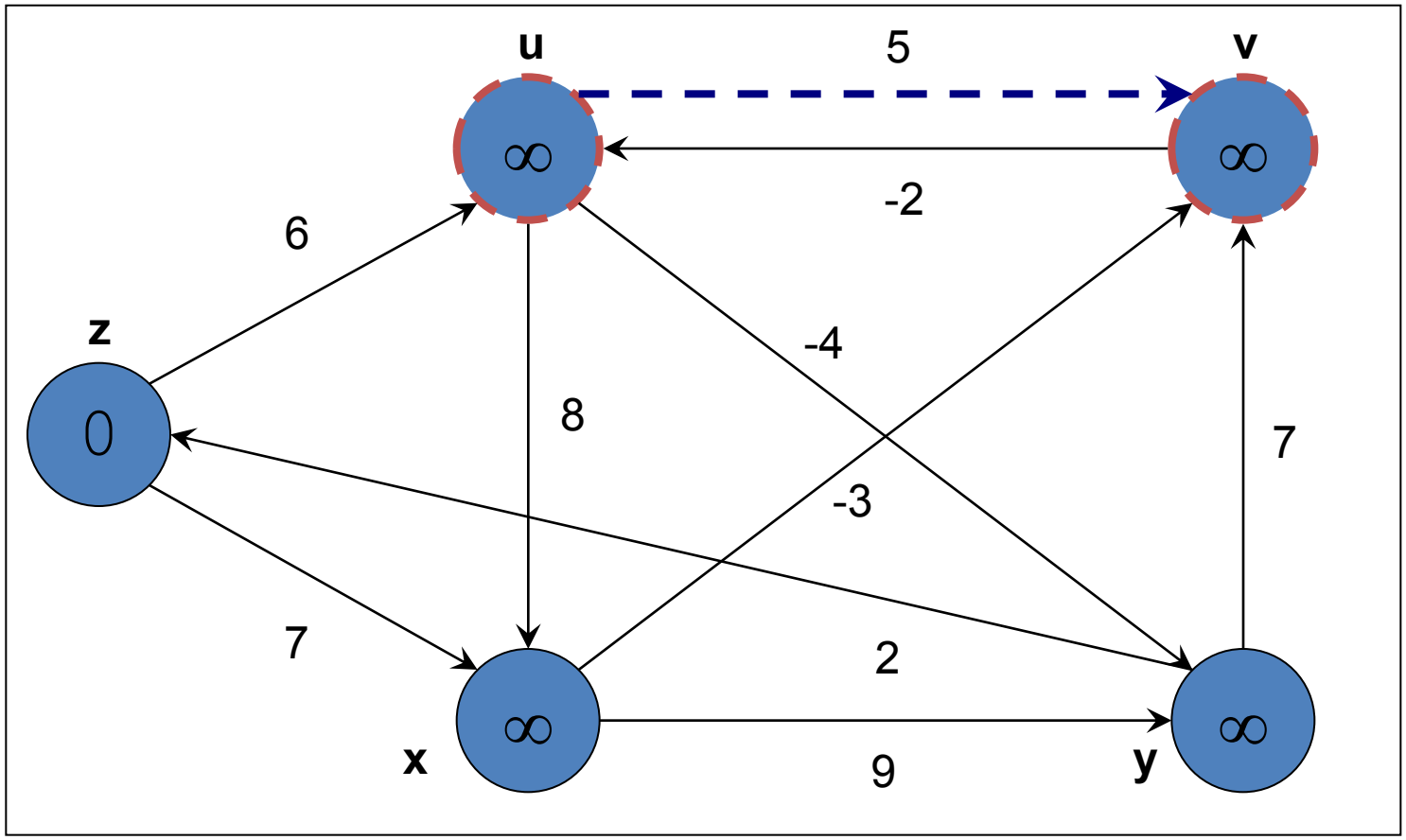
Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	∞	∞	∞	∞	**0**	**}**
Π	**[**	**]**	**=**	**{**						**}**

Paso 0.1

Inicializar los vectores d y Π.



Lista de Arcos

(u,v)←
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	∞	∞	∞	∞	**0**	**}**
Π	**[**	**]**	**=**	**{**						**}**

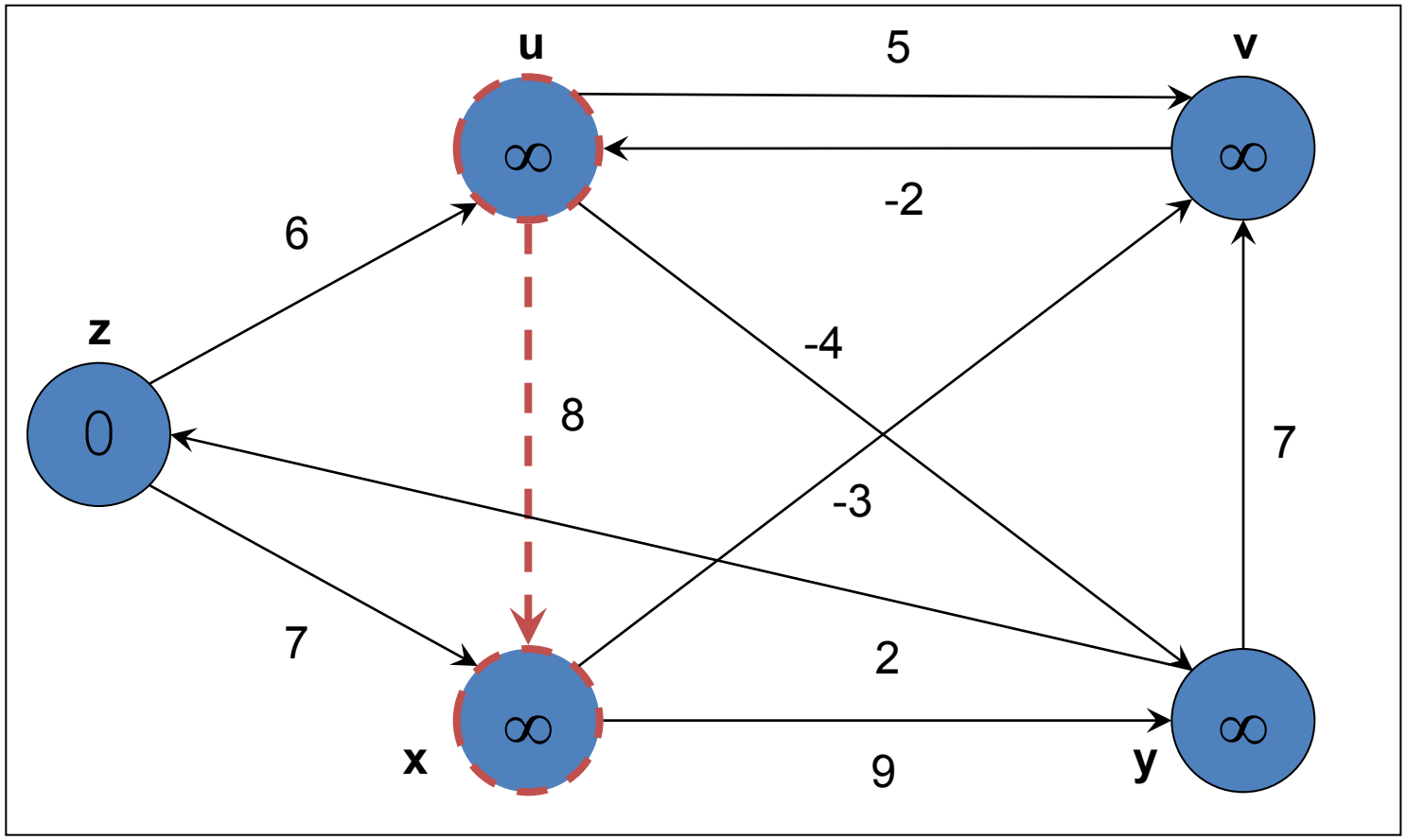
Paso 1.1

Aplicar Relax al Arco (u,v)

Pregunta: ¿ d[v] > d[u] + w(u , v) ?

Respuesta: NO

Proceso: No se hace nada.



Lista de Arcos

(u,v)
(u,x) ←
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	∞	∞	∞	∞	**0**	**}**
Π	**[**	**]**	**=**	**{**						**}**

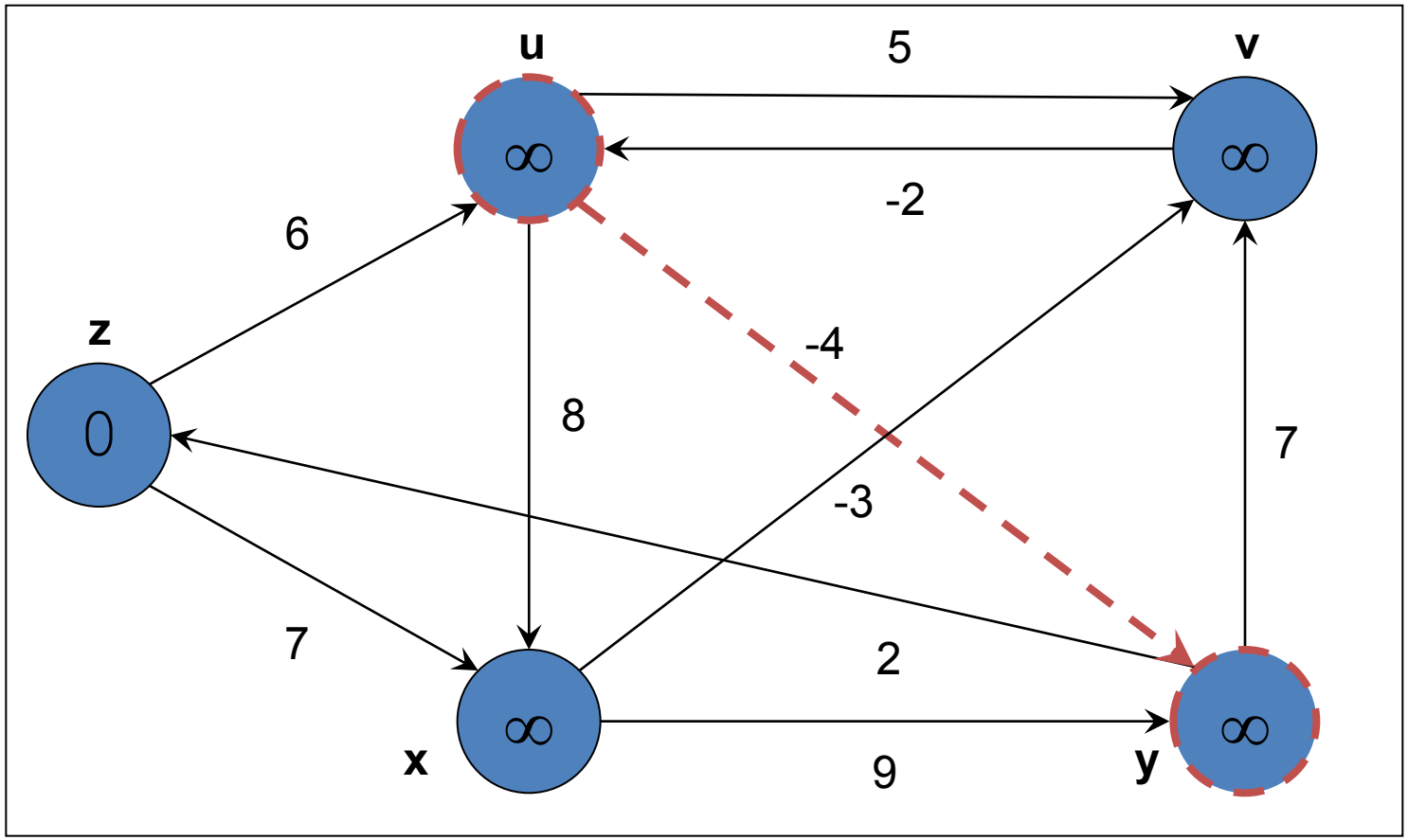
Paso 1.2

Aplicar Relax al Arco (u,x)

Pregunta: ¿ d[x] > d[u] + w(u , x) ?

Respuesta: NO

Proceso: No se hace nada.



Lista de Arcos

(u,v)
(u,x)
(u,y) ←
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	**∞**	**∞**	**∞**	**∞**	**0**	**}**
Π	**[**	**]**	**=**	**{**						**}**

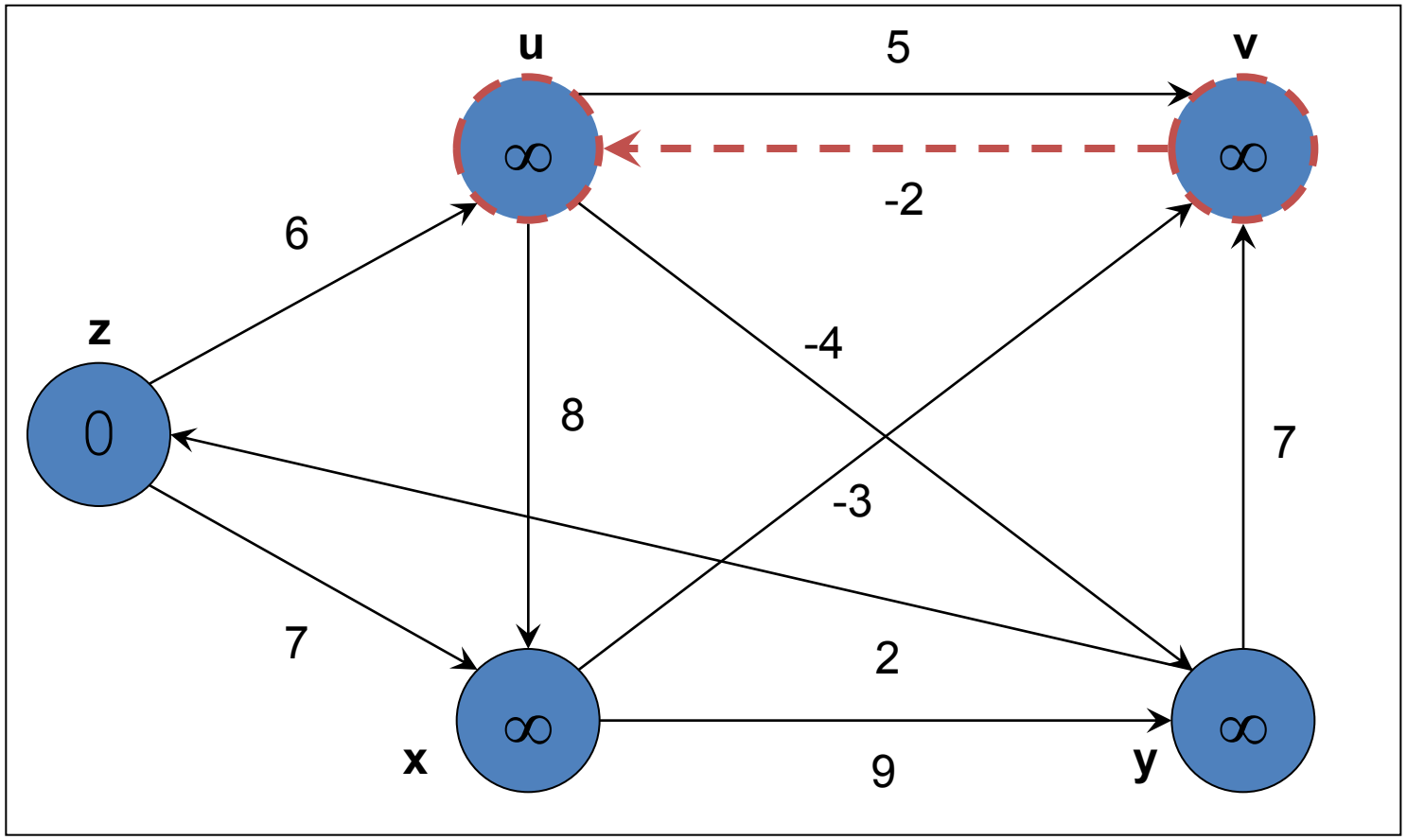
Paso 1.3

Aplicar Relax al Arco (u,y)

Pregunta: ¿ d[y] > d[u] + w(u , y) ?

Respuesta: NO

Proceso: No se hace nada.



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u) ←
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	∞	∞	∞	∞	**0**	**}**
Π	**[**	**]**	**=**	**{**						**}**

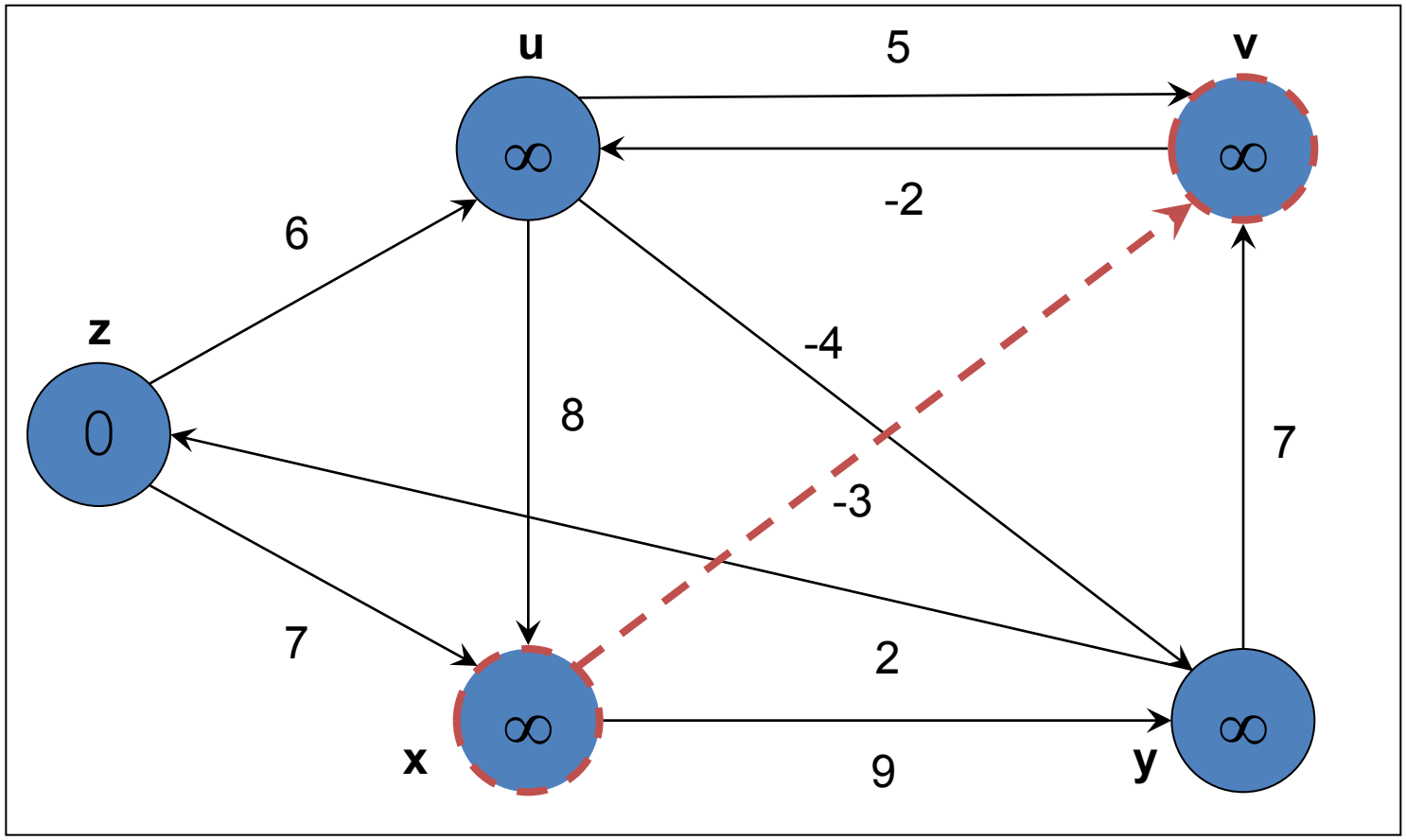
Paso 1.4

Aplicar Relax al Arco (v,u)

Pregunta: ¿ d[u] > d[v] + w(v , u) ?

Respuesta: NO

Proceso: No se hace nada.



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v) ←
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

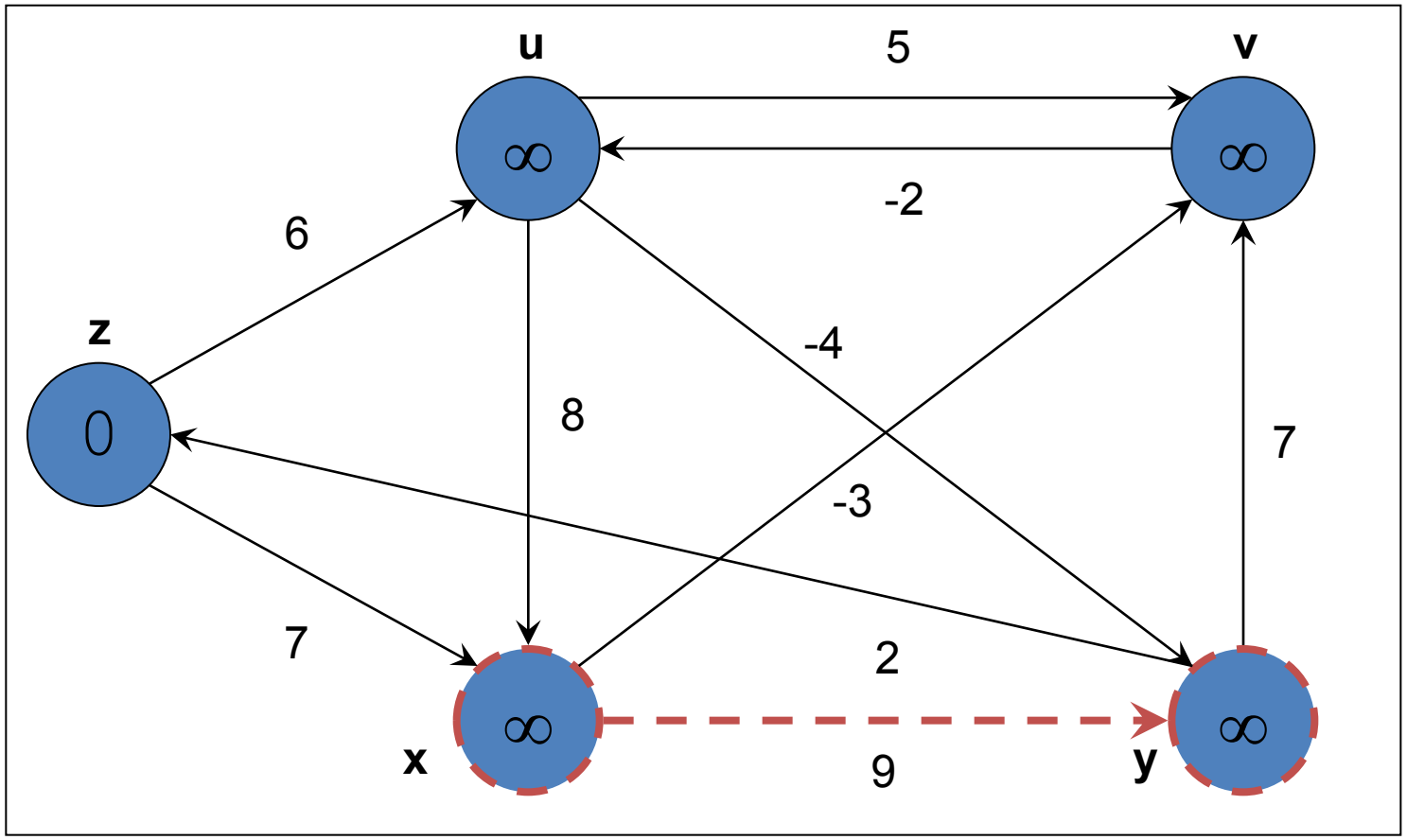
V	[ ]	=	{	u	v	x	y	z	}
d	[ ]	=	{	∞	∞	∞	∞	0	}
Π	[ ]	=	{						}

Paso 1.5 Aplicar Relax al Arco (x,v)

Pregunta: ¿ d[v] > d[x] + w(x , v) ?

Respuesta: NO

Proceso: No se hace nada.



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y) ←
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	∞	∞	∞	∞	**0**	**}**
Π	**[**	**]**	**=**	**{**						**}**

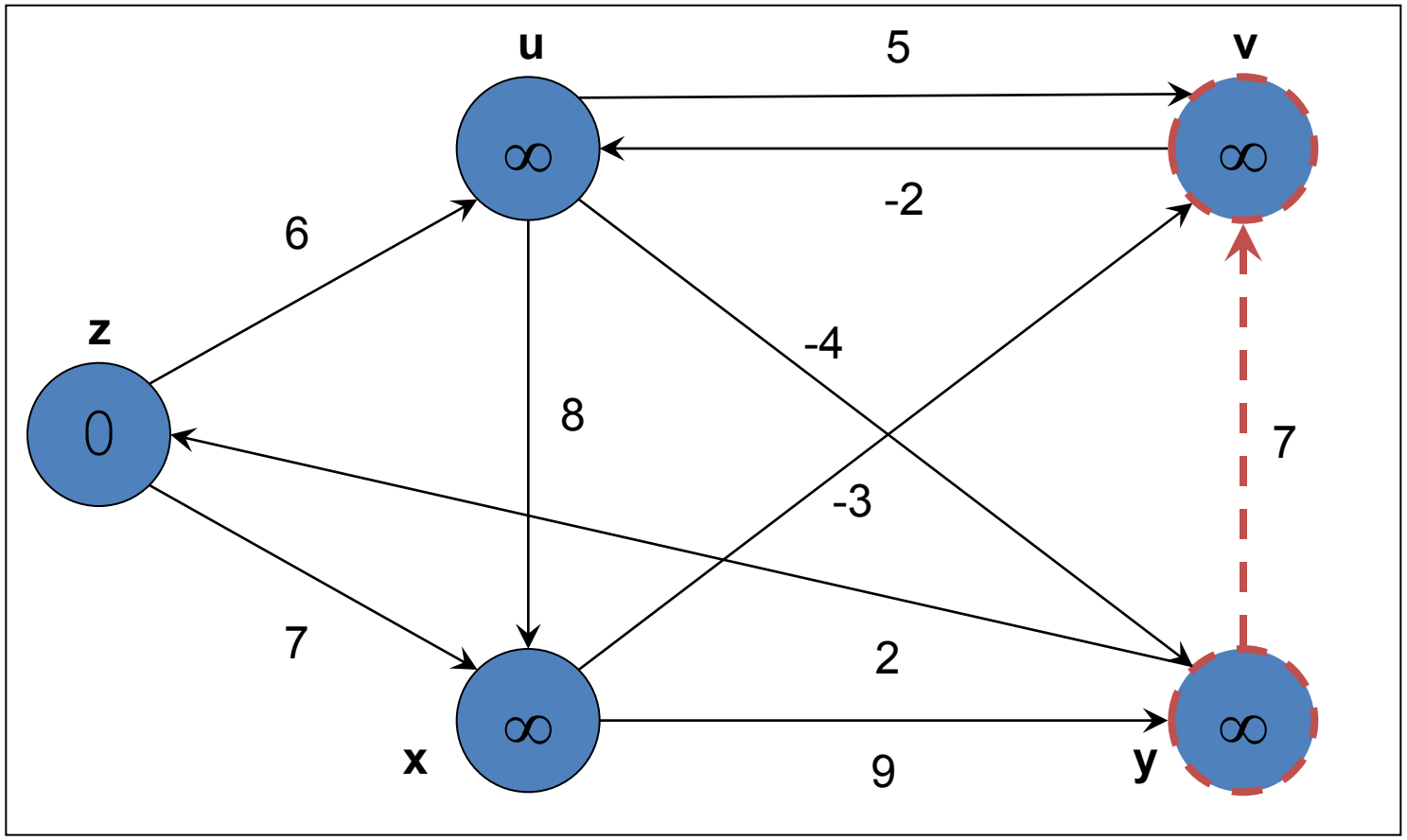
Paso 1.6

Aplicar Relax al Arco (x,y)

Pregunta: ¿ d[y] > d[x] + w(x , y) ?

Respuesta: NO

Proceso: No se hace nada.



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v) ←
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	∞	∞	∞	∞	**0**	**}**
Π	**[**	**]**	**=**	**{**						**}**

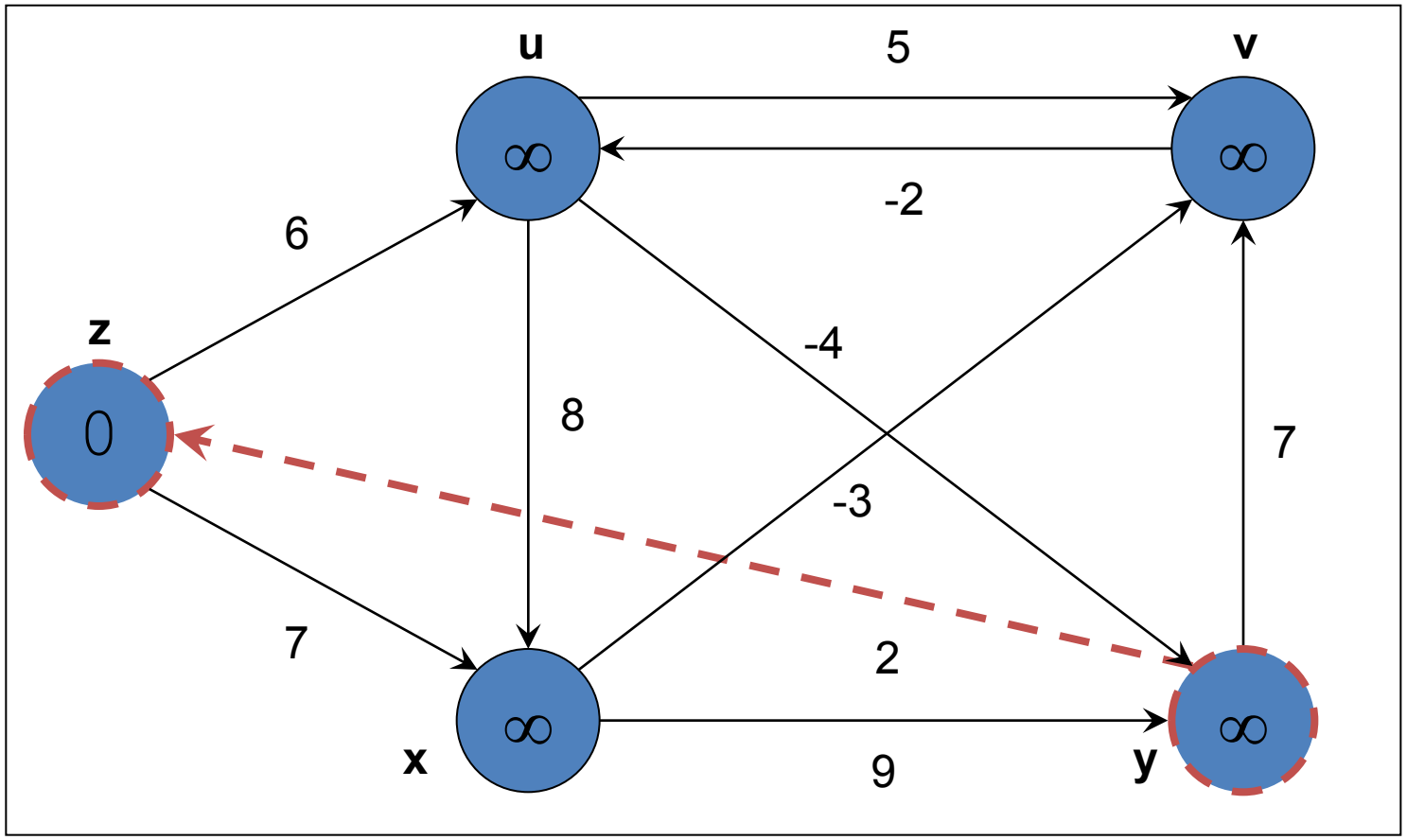
Paso 1.7

Aplicar Relax al Arco (y,v)

Pregunta: ¿ d[v] > d[y] + w(y , v) ?

Respuesta: NO

Proceso: No se hace nada.



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z) ←
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	∞	∞	∞	∞	**0**	**}**
Π	**[**	**]**	**=**	**{**						**}**

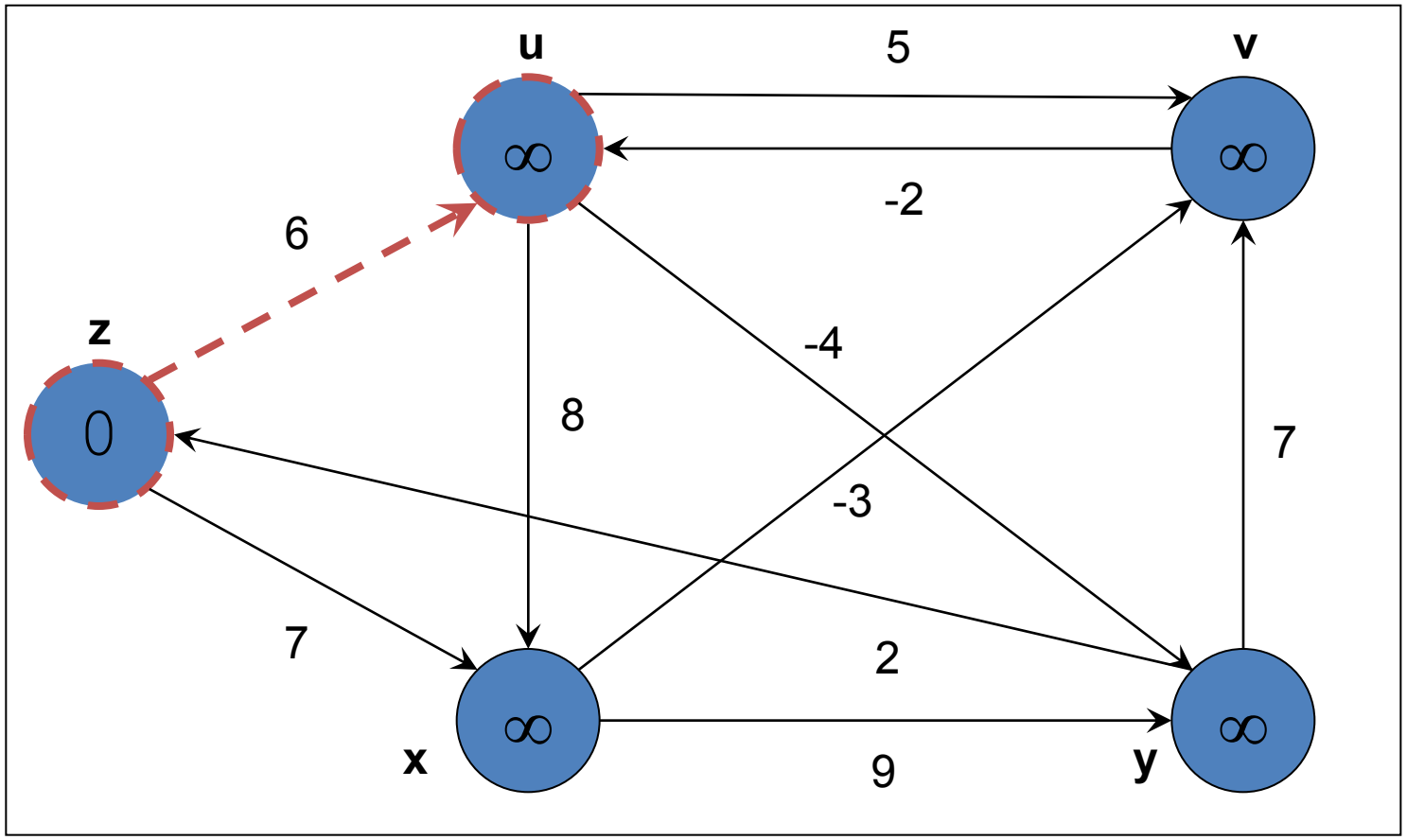
Paso 1.8

Aplicar Relax al Arco (y,v)

Pregunta: ¿ d[z] > d[y] + w(y , z) ?

Respuesta: NO

Proceso: No se hace nada.



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u) ←
(z,x)

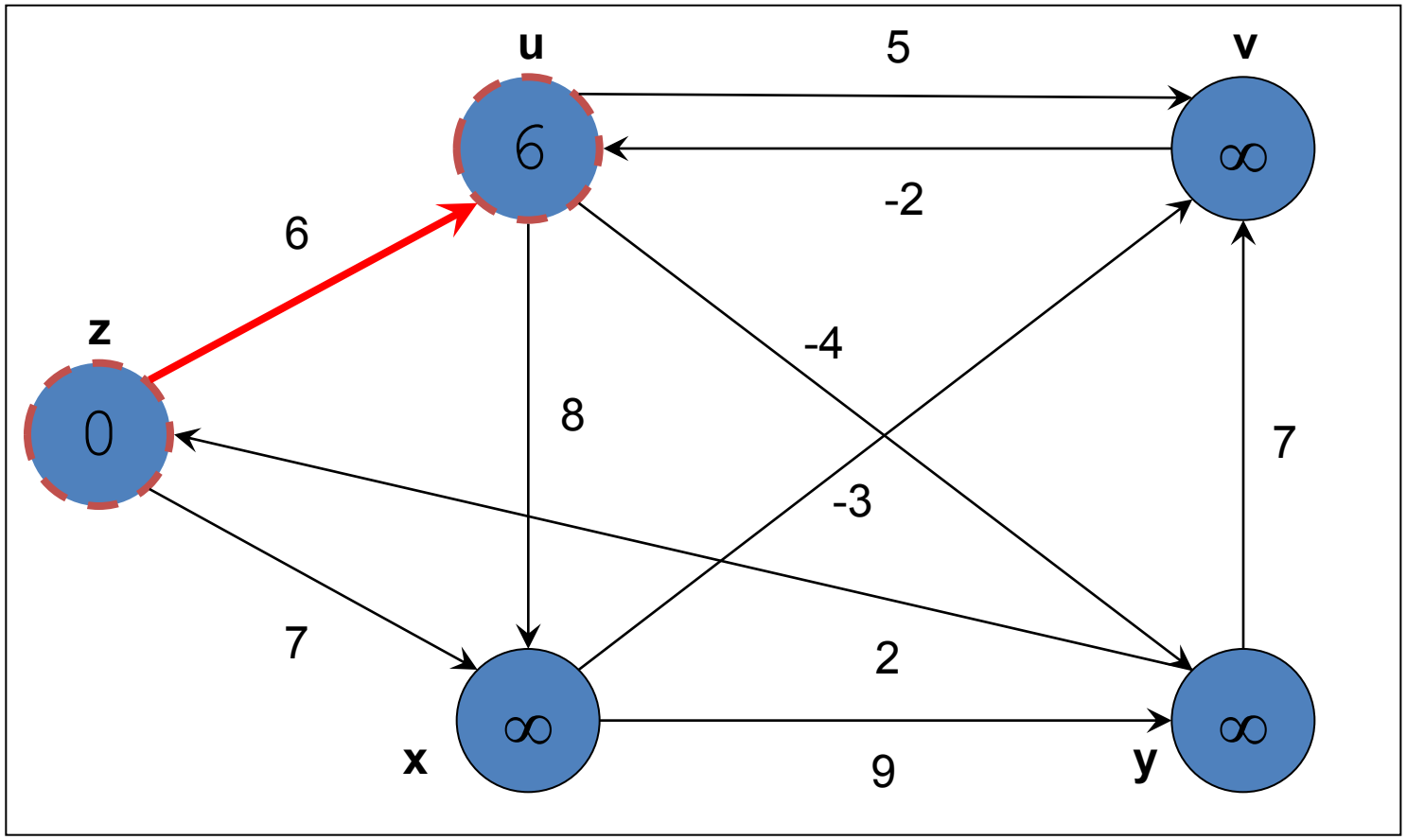
V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	∞	∞	∞	∞	**0**	**}**
Π	**[**	**]**	**=**	**{**						**}**

Paso 1.9 Aplicar Relax al Arco (z,u)

Pregunta: ¿ d[u] > d[z] + w(z , u) ?

Respuesta: SI

Proceso: d[u] = d[z] + w(z, u) y Π[u] = z



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u) ←
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	∞	∞	∞	**0**	**}**
Π	**[**	**]**	**=**	**{**	**z**					**}**

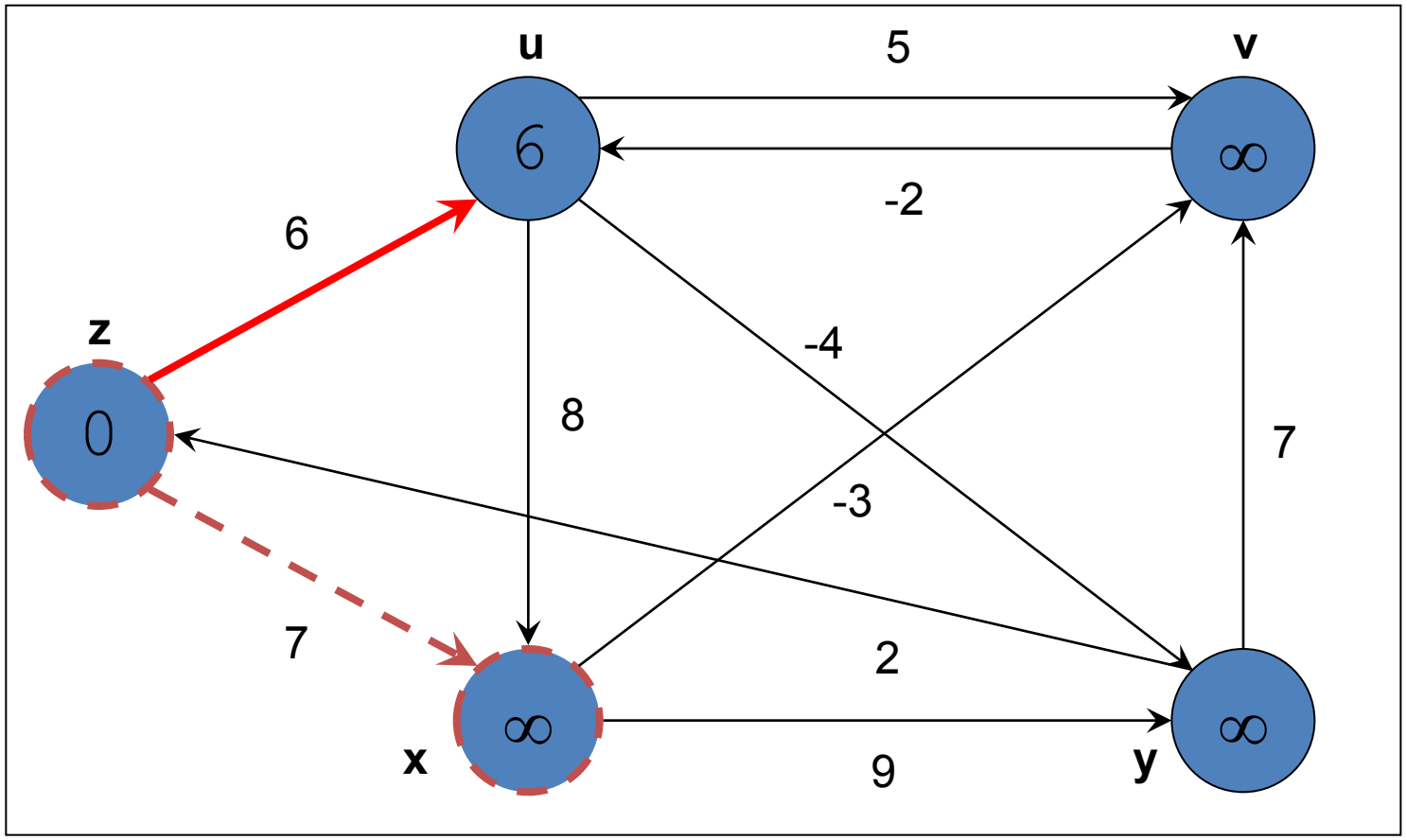
Paso 1.9

Aplicar Relax al Arco (z,u)

Pregunta: ¿ d[u] > d[z] + w(z , u) ?

Respuesta: SI

Proceso: d[u] = d[z] + w(z, u) y Π[u] = z



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x) ←

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	∞	∞	∞	**0**	**}**
Π	**[**	**]**	**=**	**{**	**z**					**}**

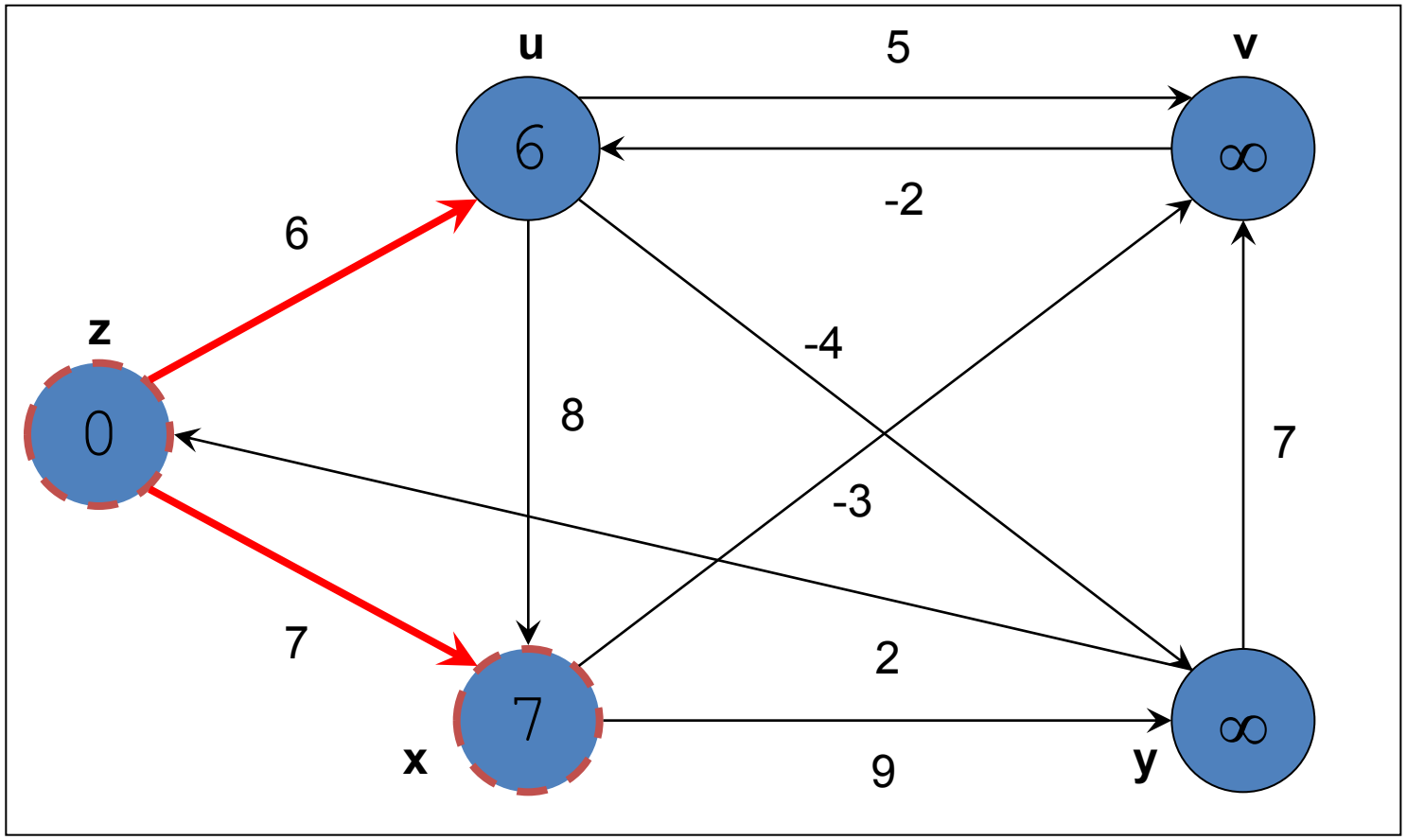
Paso 1.10

Aplicar Relax al Arco (z,x)

Pregunta: ¿ d[x] > d[z] + w(z , x) ?

Respuesta: SI

Proceso: d[x] = d[z] + w(z, x) y Π[x] = z



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x) ←

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	∞	7	∞	0	**}**
Π	**[**	**]**	**=**	**{**	**z**		**z**			**}**

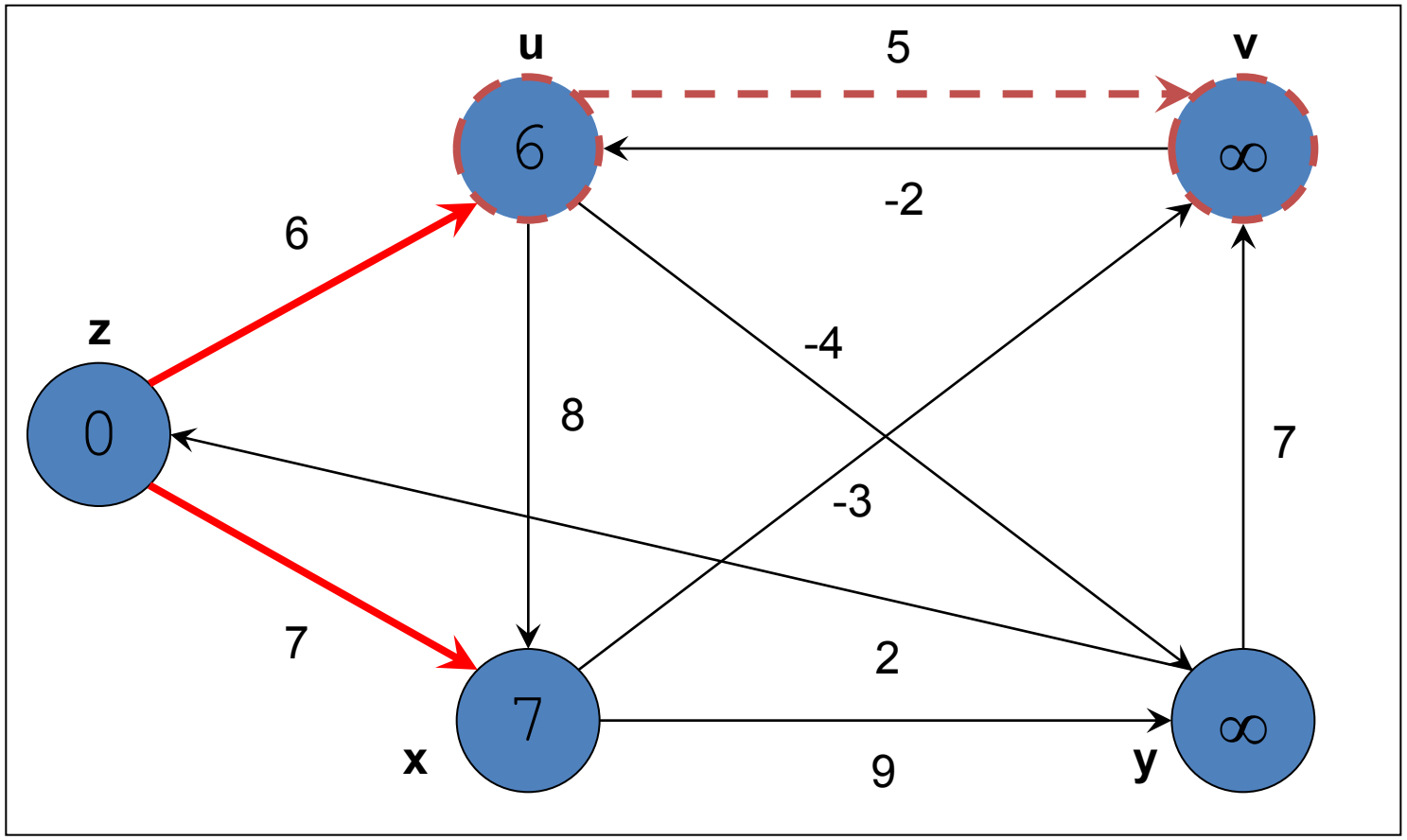
Paso 1.10

Aplicar Relax al Arco (z,x)

Pregunta: ¿ d[x] > d[z] + w(z , x) ?

Respuesta: SI

Proceso: d[x] = d[z] + w(z, x) y Π[x] = z



Lista de Arcos

(u,v) ←
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	[ ]	=	{	u	v	x	y	z	}
d	[ ]	=	{	6	∞	7	∞	0	}
Π	[ ]	=	{	z		z			}

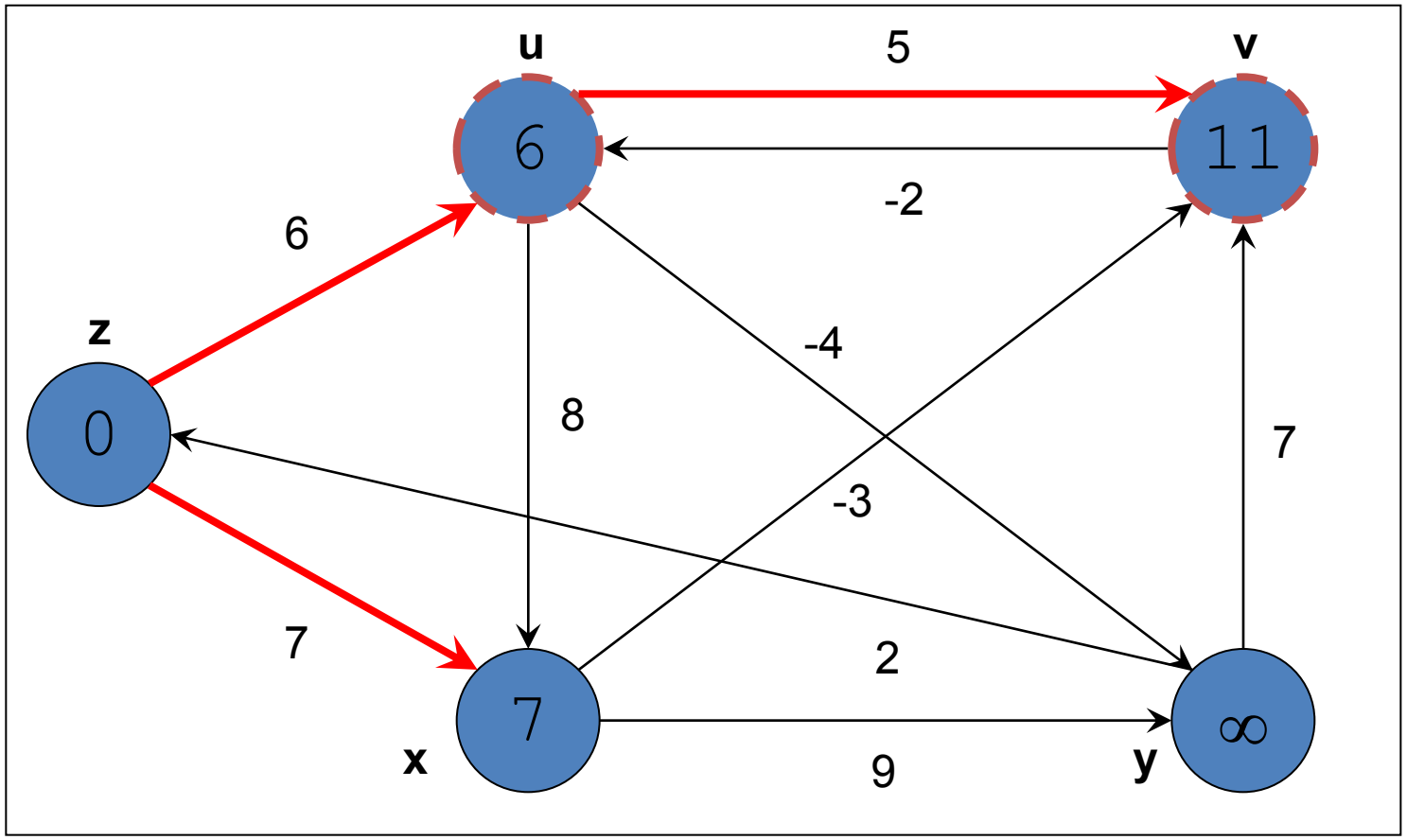
Paso 2.1

Aplicar Relax al Arco (u,v)

Pregunta: ¿ d[v] > d[u] + w(u , v) ?

Respuesta: SI

Proceso: d[v] = d[u] + w(u, v) y Π[v] = u



Lista de Arcos

(u,v) ←
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	11	**7**	∞	**0**	**}**
Π	**[**	**]**	**=**	**{**	**z**	**u**	**z**			**}**

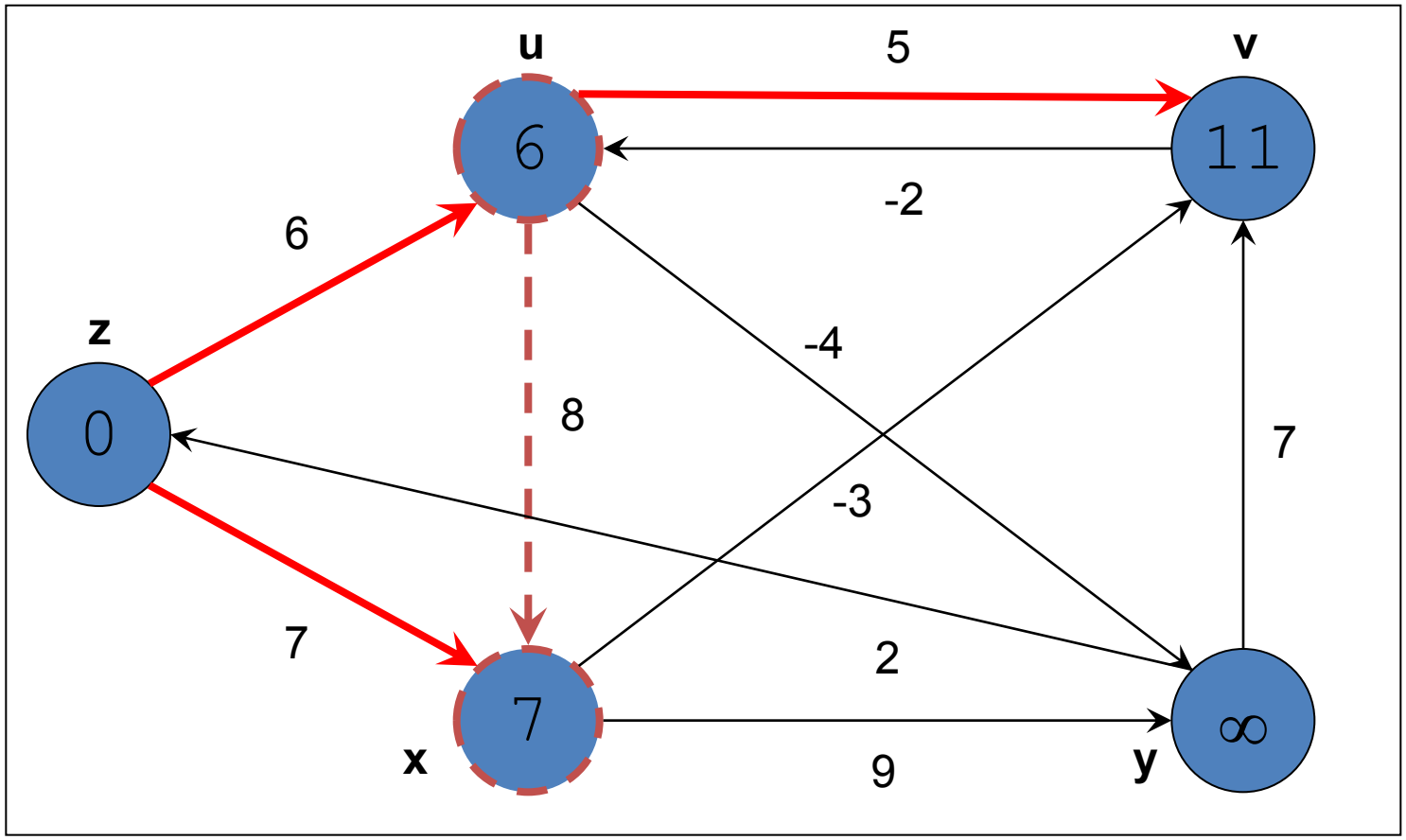
Paso 2.1

Aplicar Relax al Arco (u,v)

Pregunta: ¿ d[v] > d[u] + w(u , v) ?

Respuesta: SI

Proceso: d[v] = d[u] + w(u, v) y Π[v] = u



Lista de Arcos

(u,v)
(u,x) ←
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	[ ]	=	{	u	v	x	y	z	}
d	[ ]	=	{	6	11	7	∞	0	}
Π	[ ]	=	{	z	u	z			}

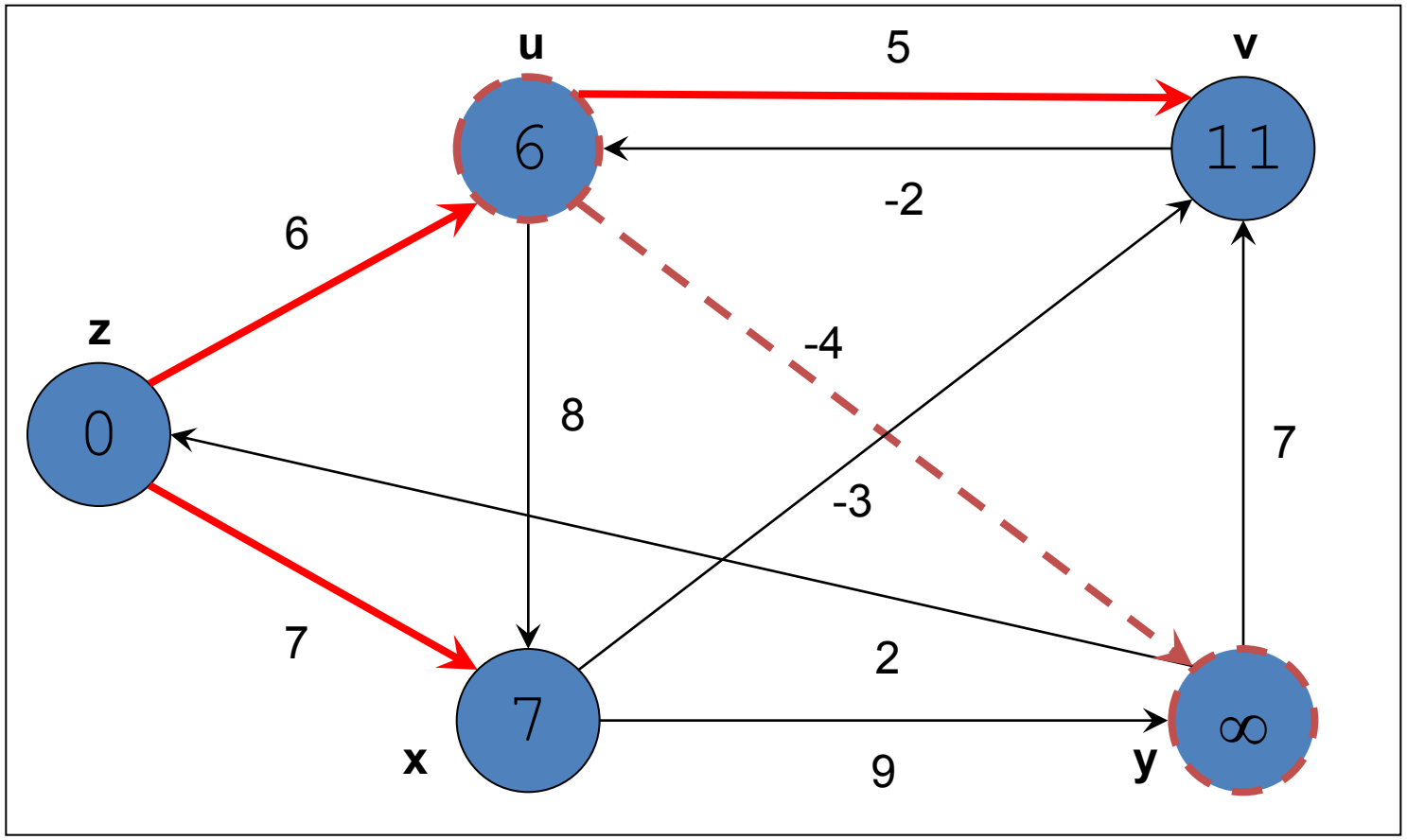
Paso 2.2

Aplicar Relax al Arco (u,x)

Pregunta: ¿ d[x] > d[u] + w(u , x) ?

Respuesta: NO

Proceso: No se hace nada.



Lista de Arcos

(u,v)
(u,x)
(u,y) ←
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	11	7	∞	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**u**	**z**			**}**

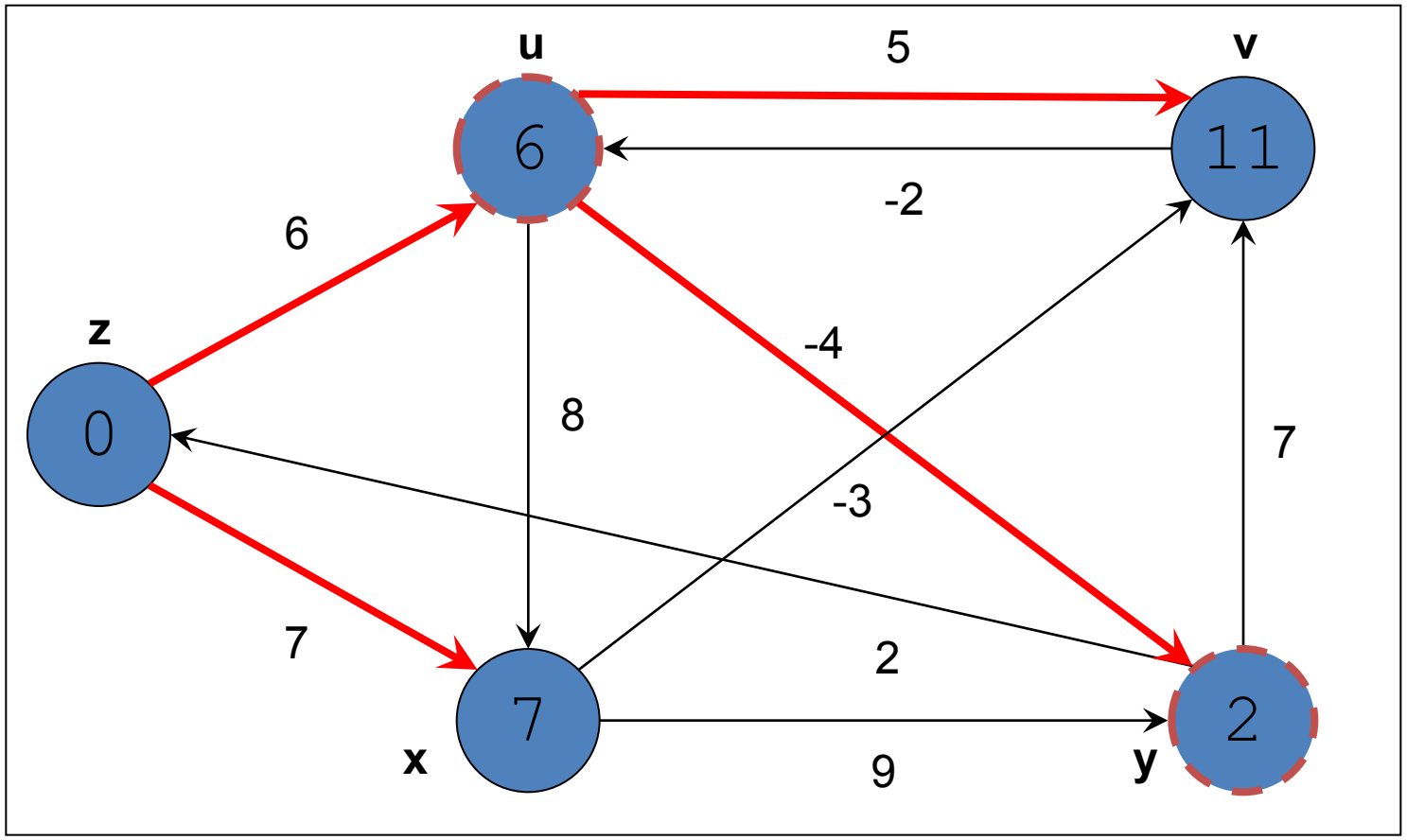
Paso 2.3

Aplicar Relax al Arco (u,y)

Pregunta: ¿ d[y] > d[u] + w(u , y) ?

Respuesta: SI

Proceso: d[y] = d[u] + w(u, y) y Π[y] = u



Lista de Arcos

(u,v)
(u,x)
(u,y) ←
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	11	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**u**	**z**	**u**		**}**

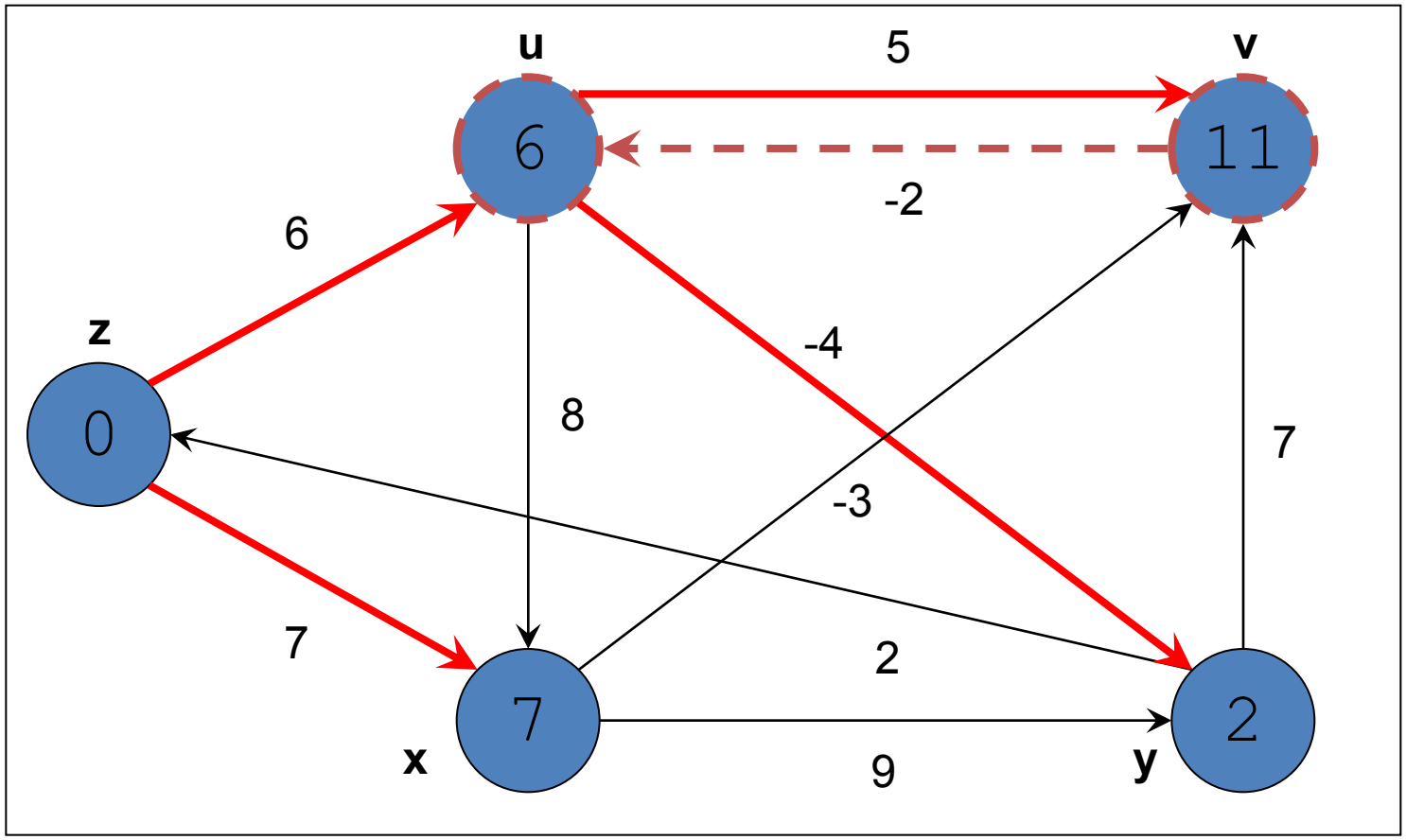
Paso 2.3

Aplicar Relax al Arco (u,y)

Pregunta: ¿ d[y] > d[u] + w(u , y) ?

Respuesta: SI

Proceso: d[y] = d[u] + w(u, y) y Π[y] = u



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u) ←
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	11	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**u**	**z**	**u**		**}**

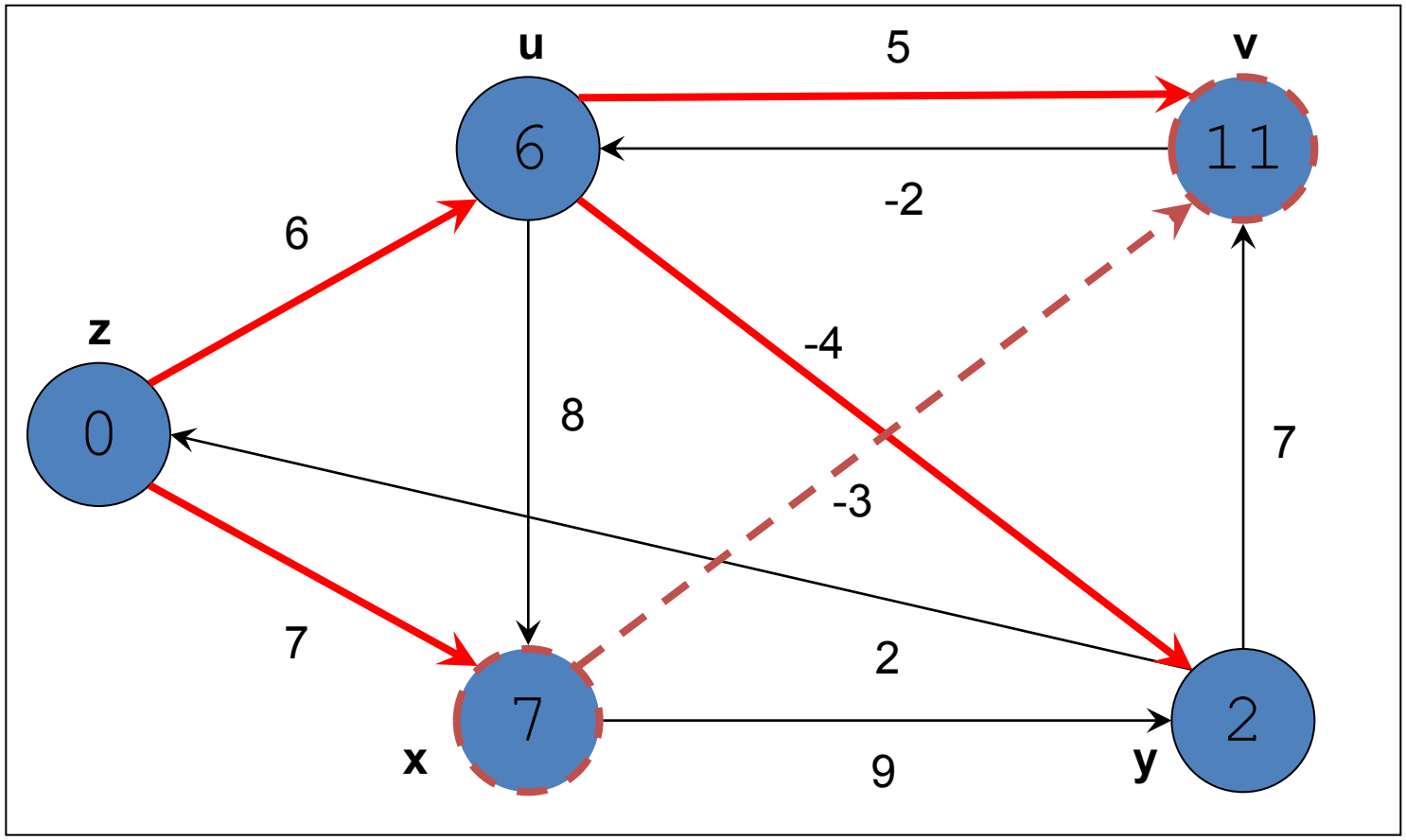
Paso 2.4

Aplicar Relax al Arco (v,u)

Pregunta: **¿ d[u] > d[v] + w(v , u) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v) ←
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	11	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**u**	**z**	**u**		**}**

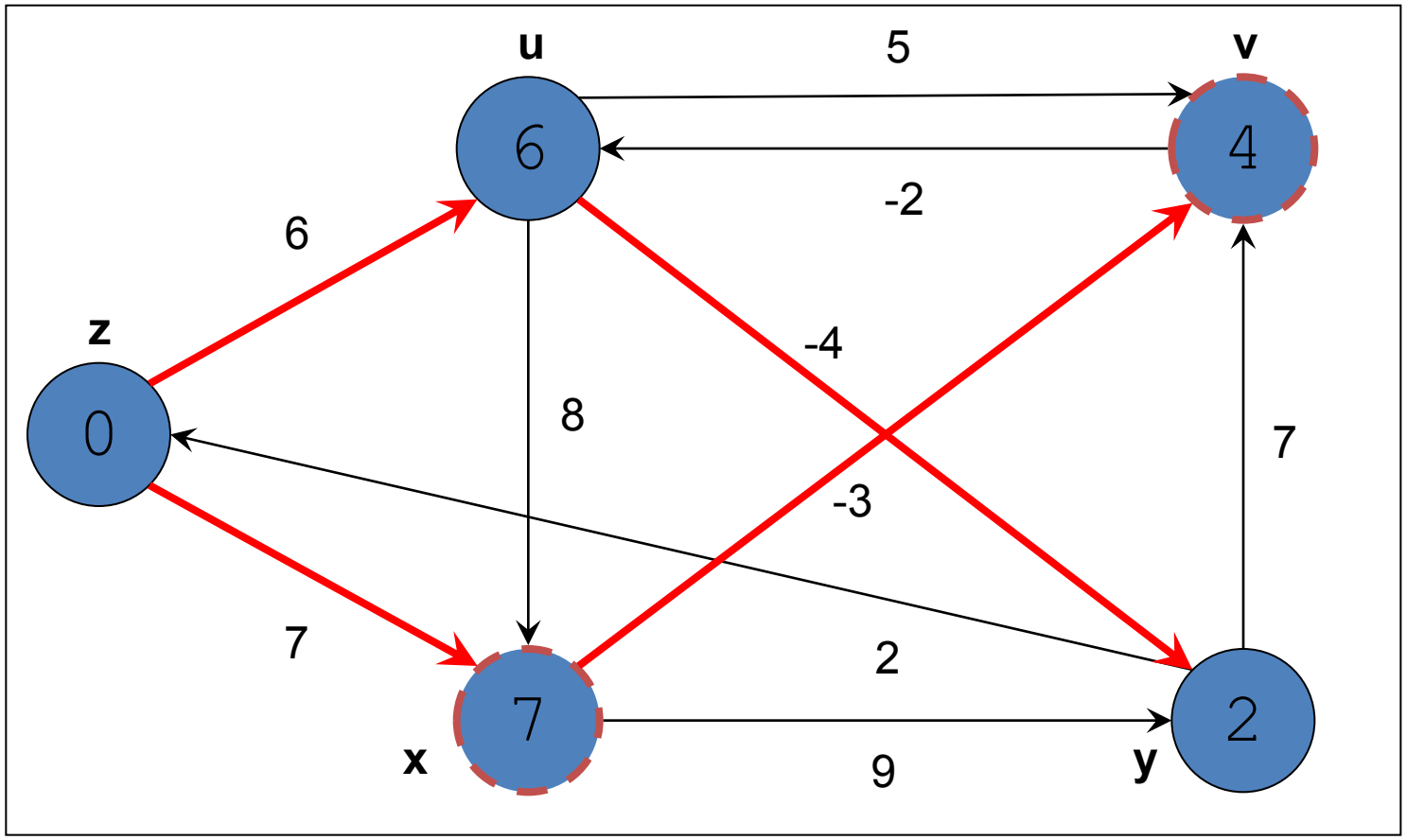
Paso 2.5

Aplicar Relax al Arco (x,v)

Pregunta: ¿ d[v] > d[x] + w(x , v) ?

Respuesta: SI

Proceso: d[y] = d[x] + w(x, v) y Π[y] = x



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v) ←
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**x**	**z**	**u**		**}**

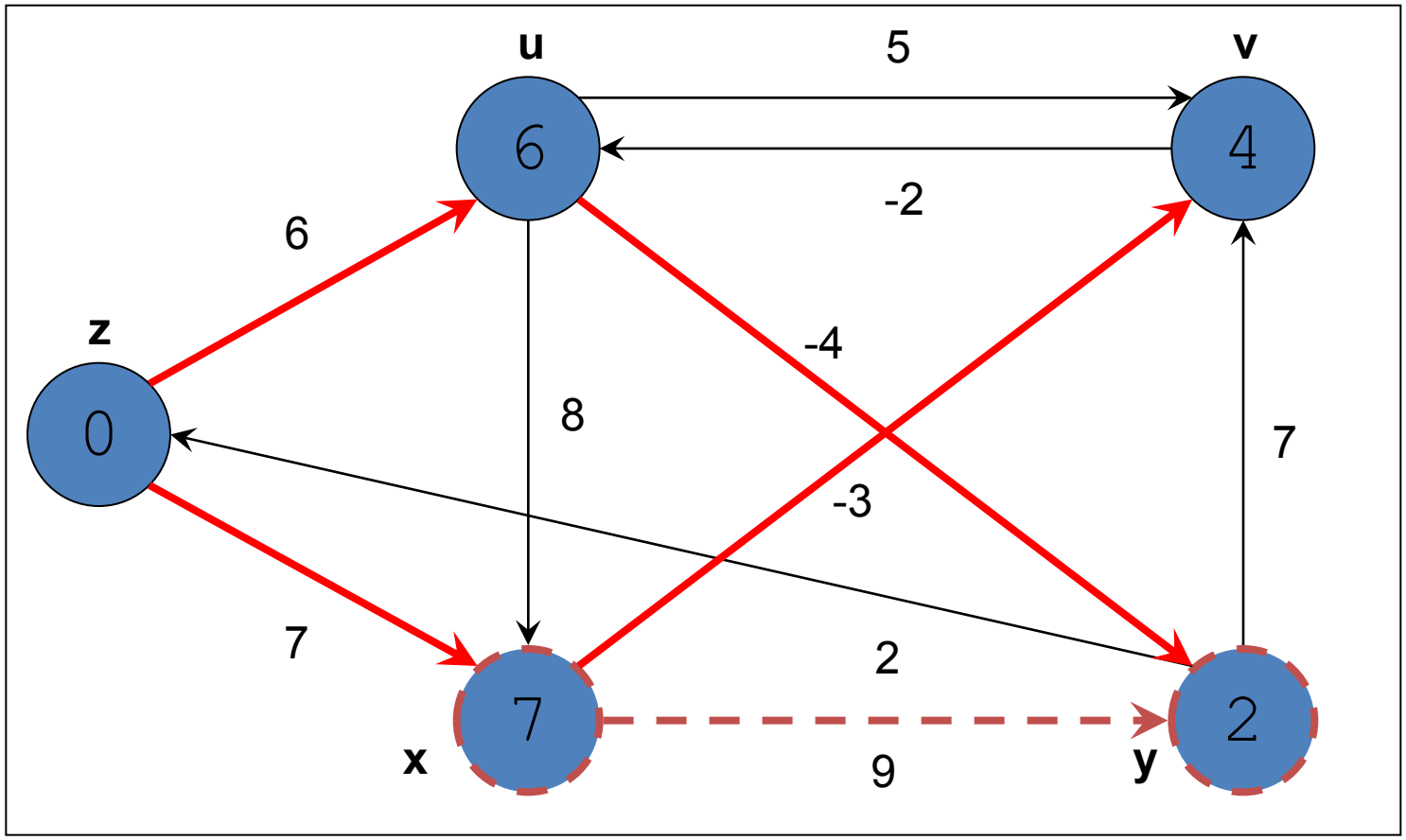
Paso 2.5

Aplicar Relax al Arco (x,v)

Pregunta: ¿ d[v] > d[x] + w(x , v) ?

Respuesta: SI

Proceso: d[y] = d[x] + w(x, v) y Π[y] = x



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y) ←
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**x**	**z**	**u**		**}**

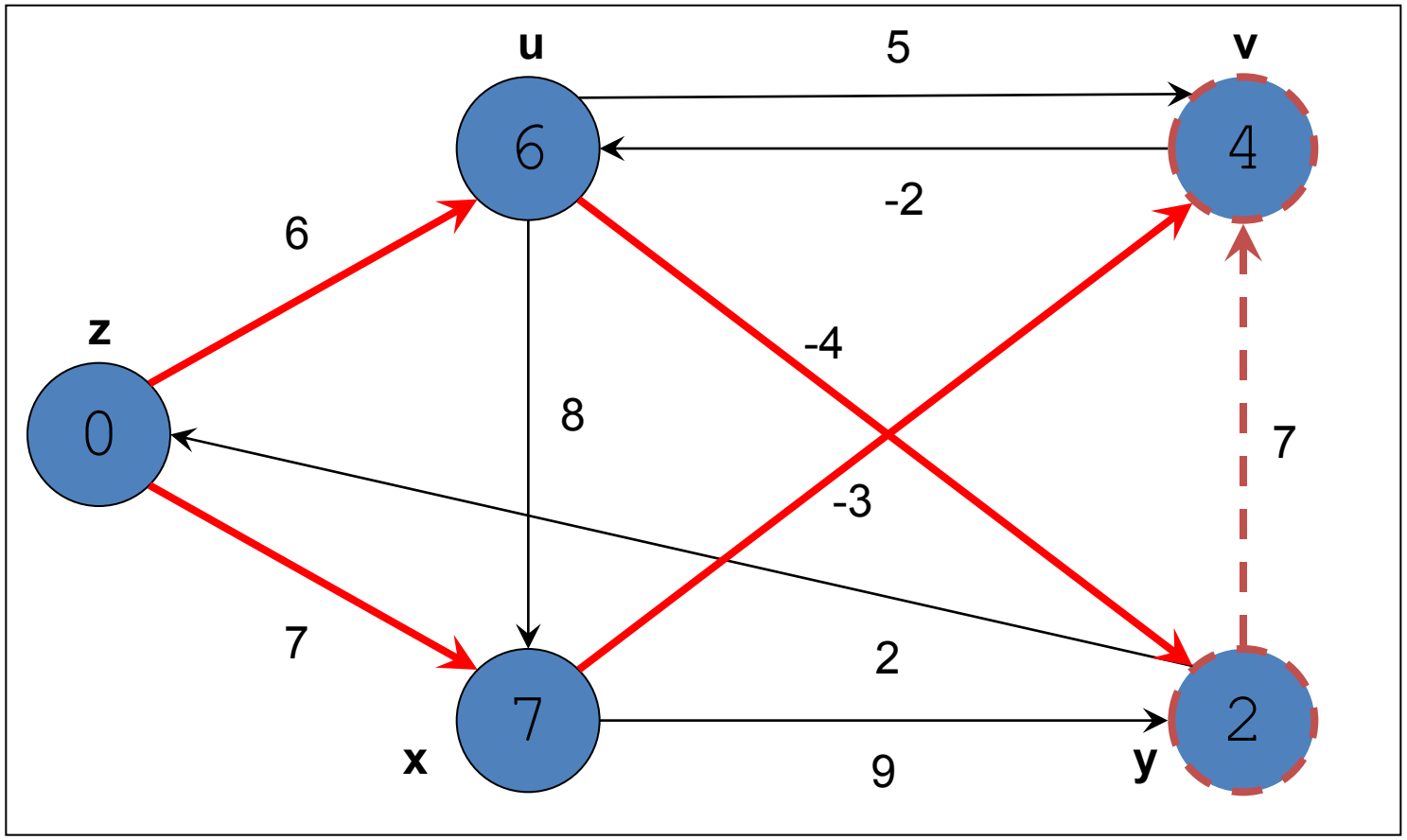
Paso 2.6

Aplicar Relax al Arco (x,y)

Pregunta: **¿ d[y] > d[x] + w(x , y) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v) ←
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**x**	**z**	**u**		**}**

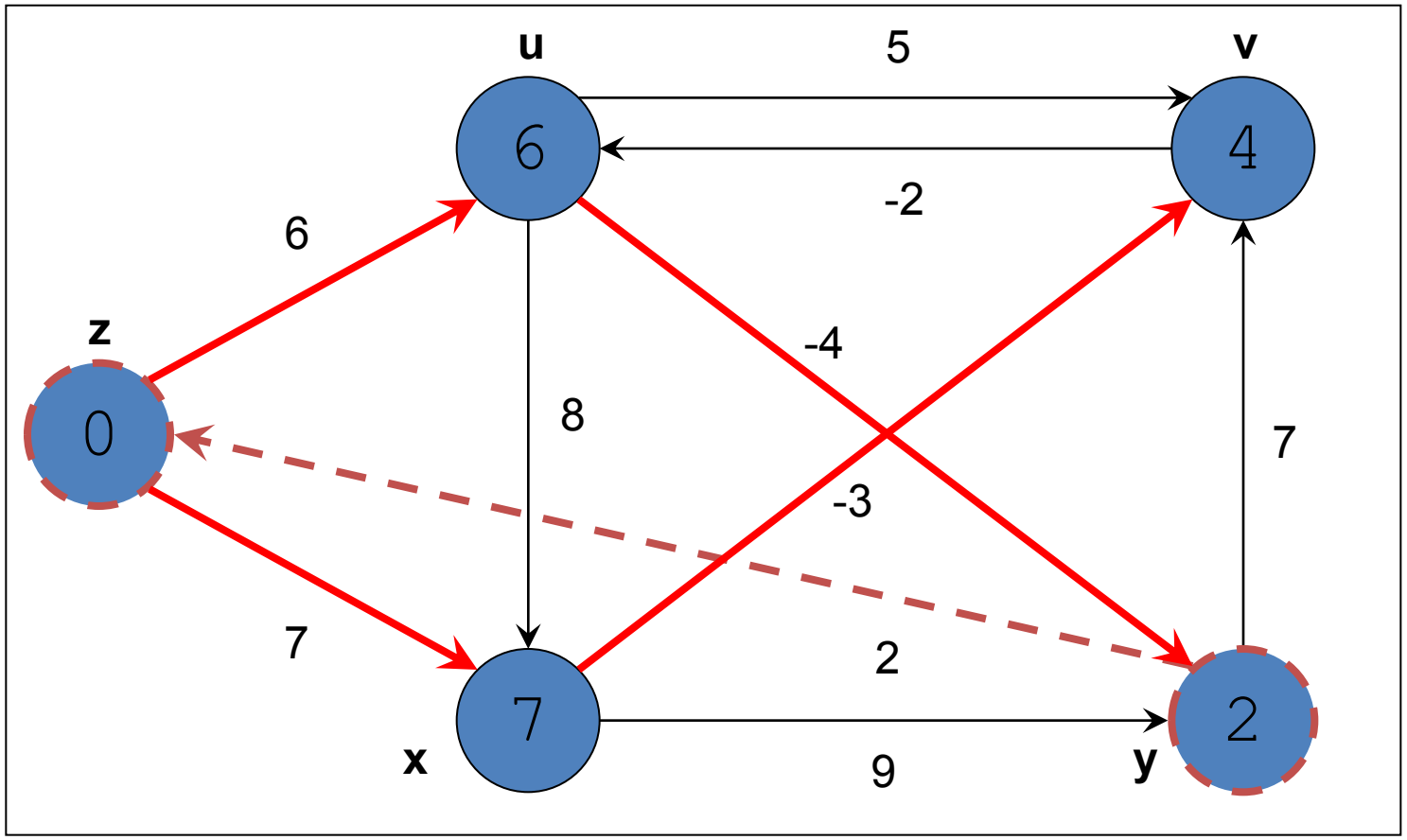
Paso 2.7

Aplicar Relax al Arco (y,v)

Pregunta: **¿ d[v] > d[y] + w(y , v) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z) ←
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**x**	**z**	**u**		**}**

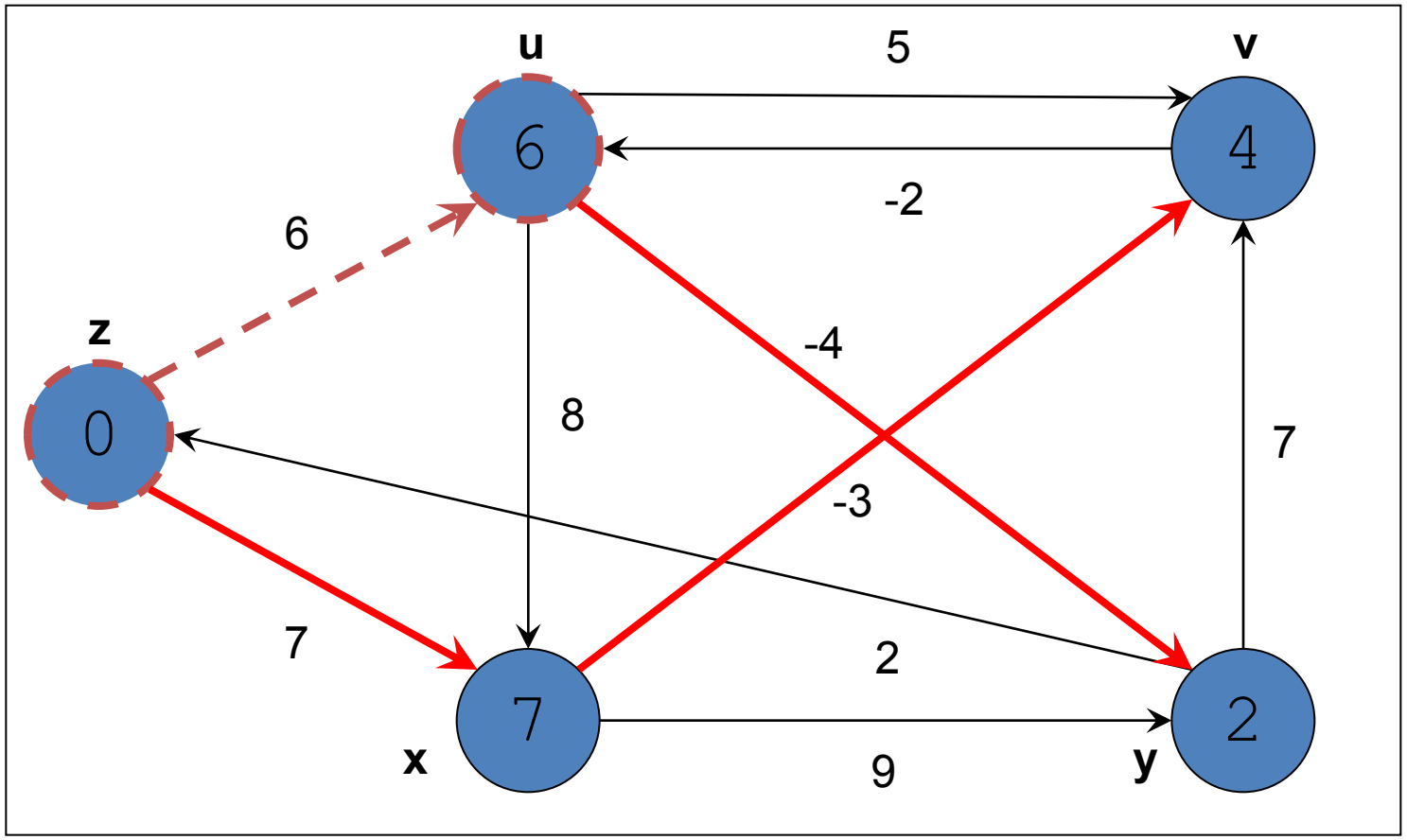
Paso 2.8

Aplicar Relax al Arco (y,z)

Pregunta: **¿ d[z] > d[y] + w(y , z) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u) ←
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**x**	**z**	**u**		**}**

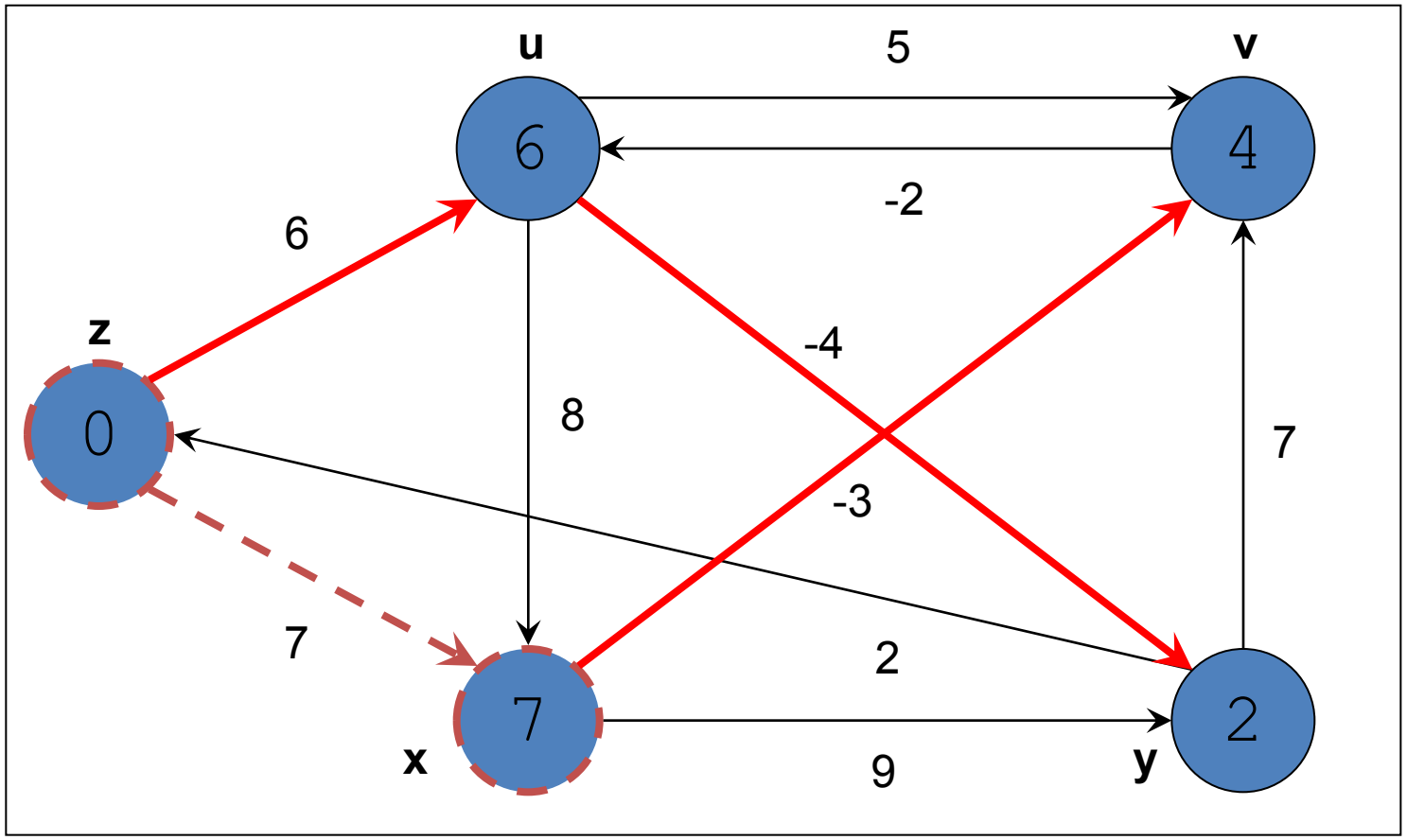
Paso 2.9

Aplicar Relax al Arco (z,u)

Pregunta: **¿ d[u] > d[z] + w(z , u) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x) ←

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**x**	**z**	**u**		**}**

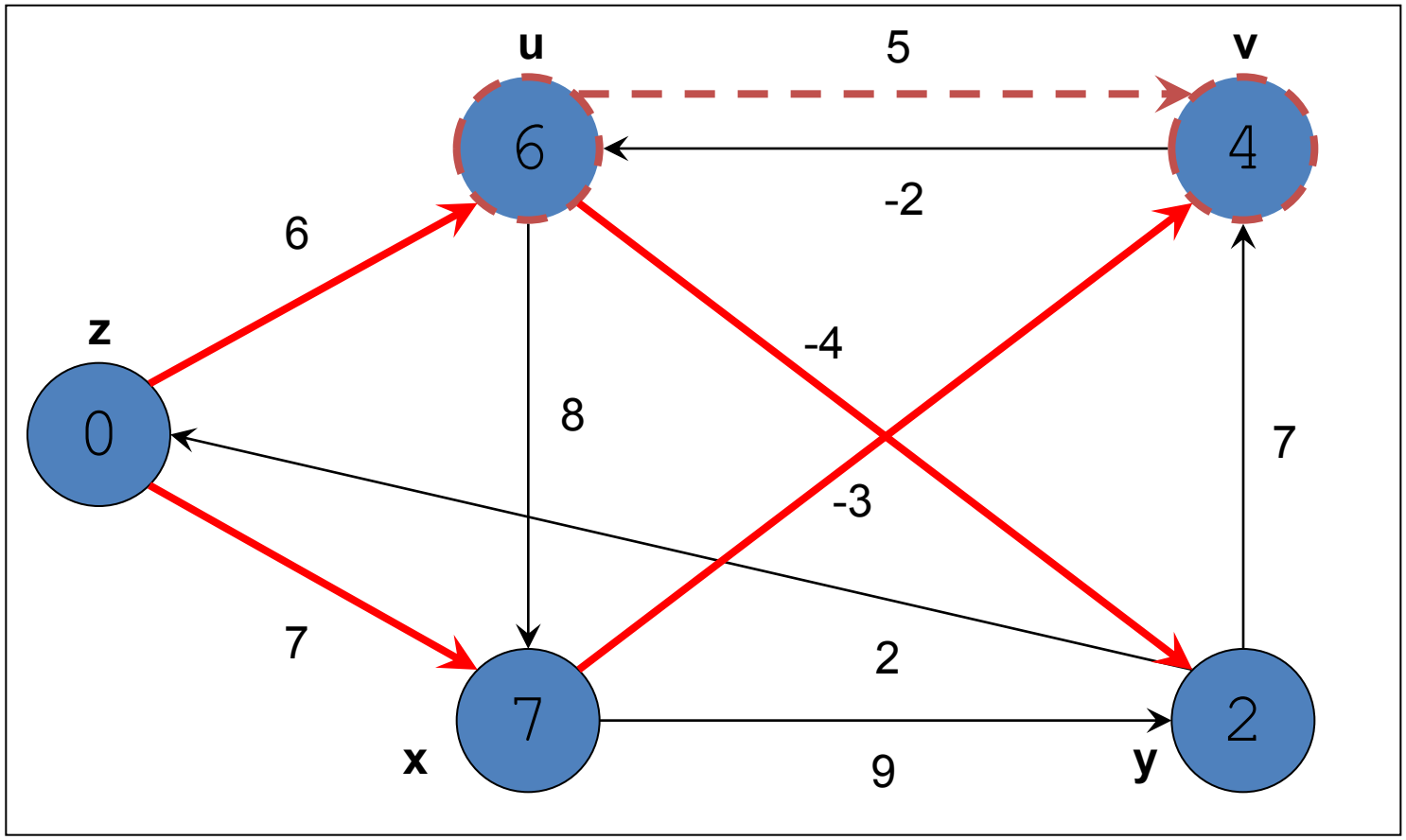
Paso 2.10

Aplicar Relax al Arco (z,x)

Pregunta: ¿ d[x] > d[z] + w(z , x) ?

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v) ←
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**x**	**z**	**u**		**}**

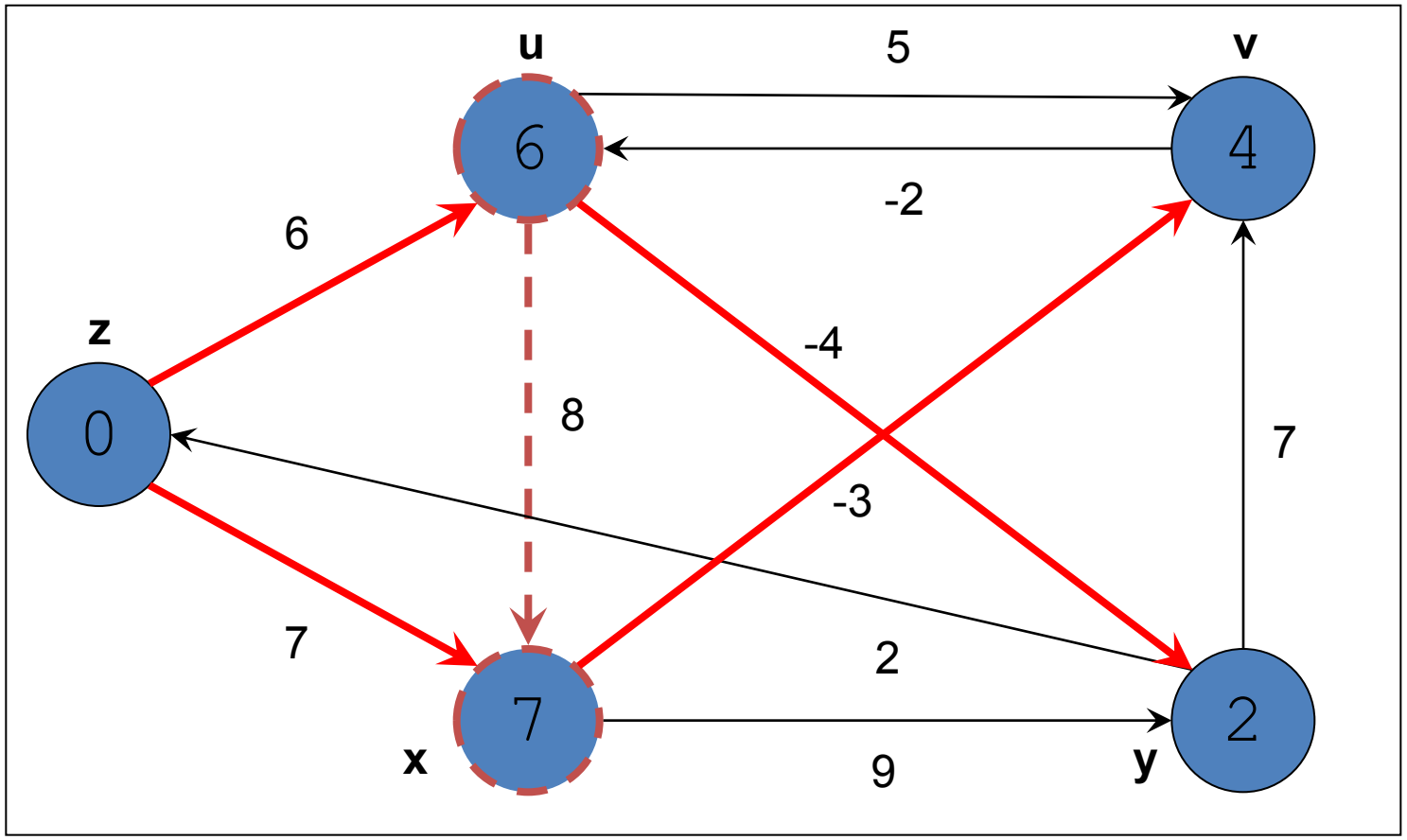
Paso 3.1

Aplicar Relax al Arco (u,v)

Pregunta: **¿ d[v] > d[u] + w(u , v) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x) ←
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**x**	**z**	**u**		**}**

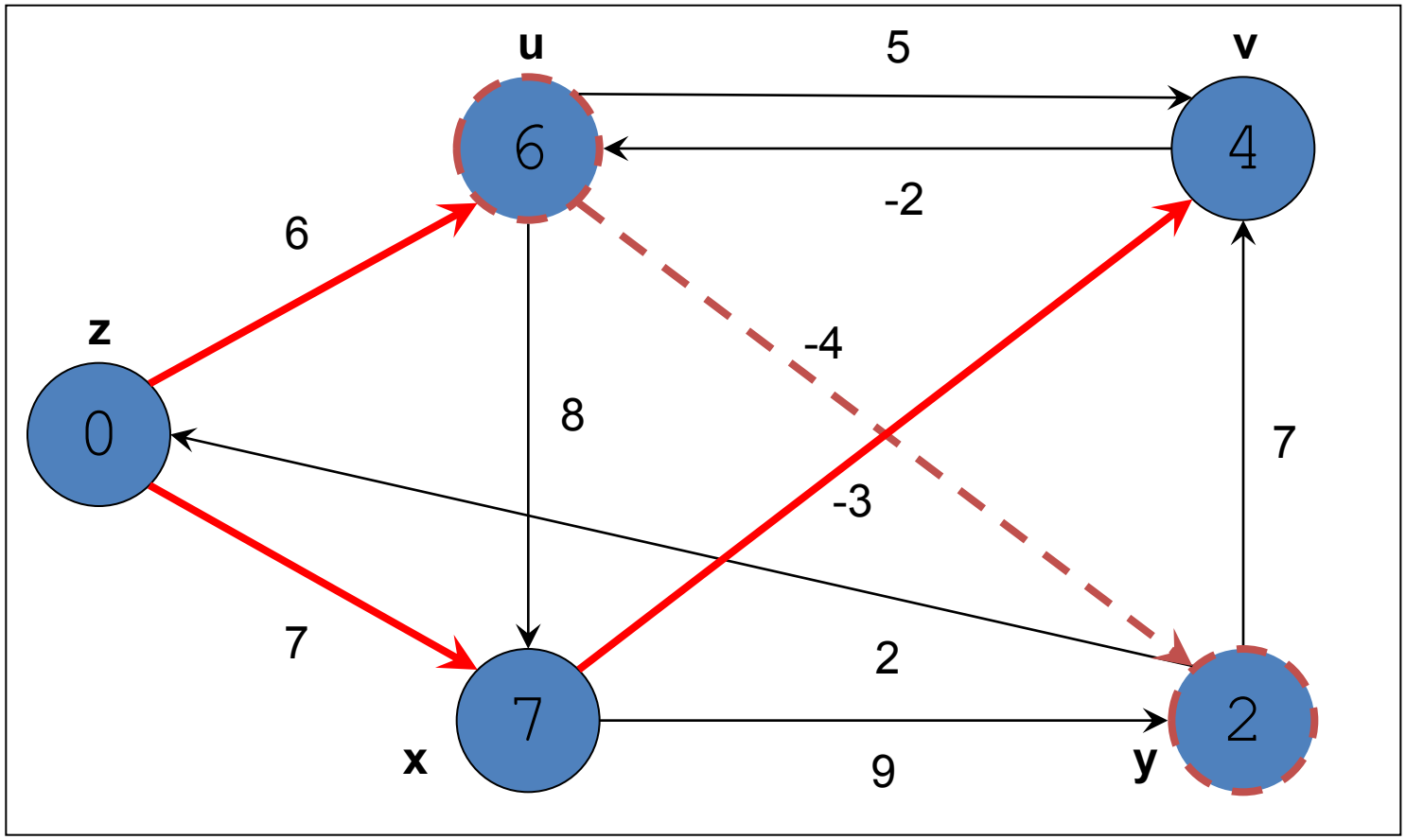
Paso 3.2

Aplicar Relax al Arco (u,x)

Pregunta: **¿ d[x] > d[u] + w(u , x) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y) ←
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**x**	**z**	**u**		**}**

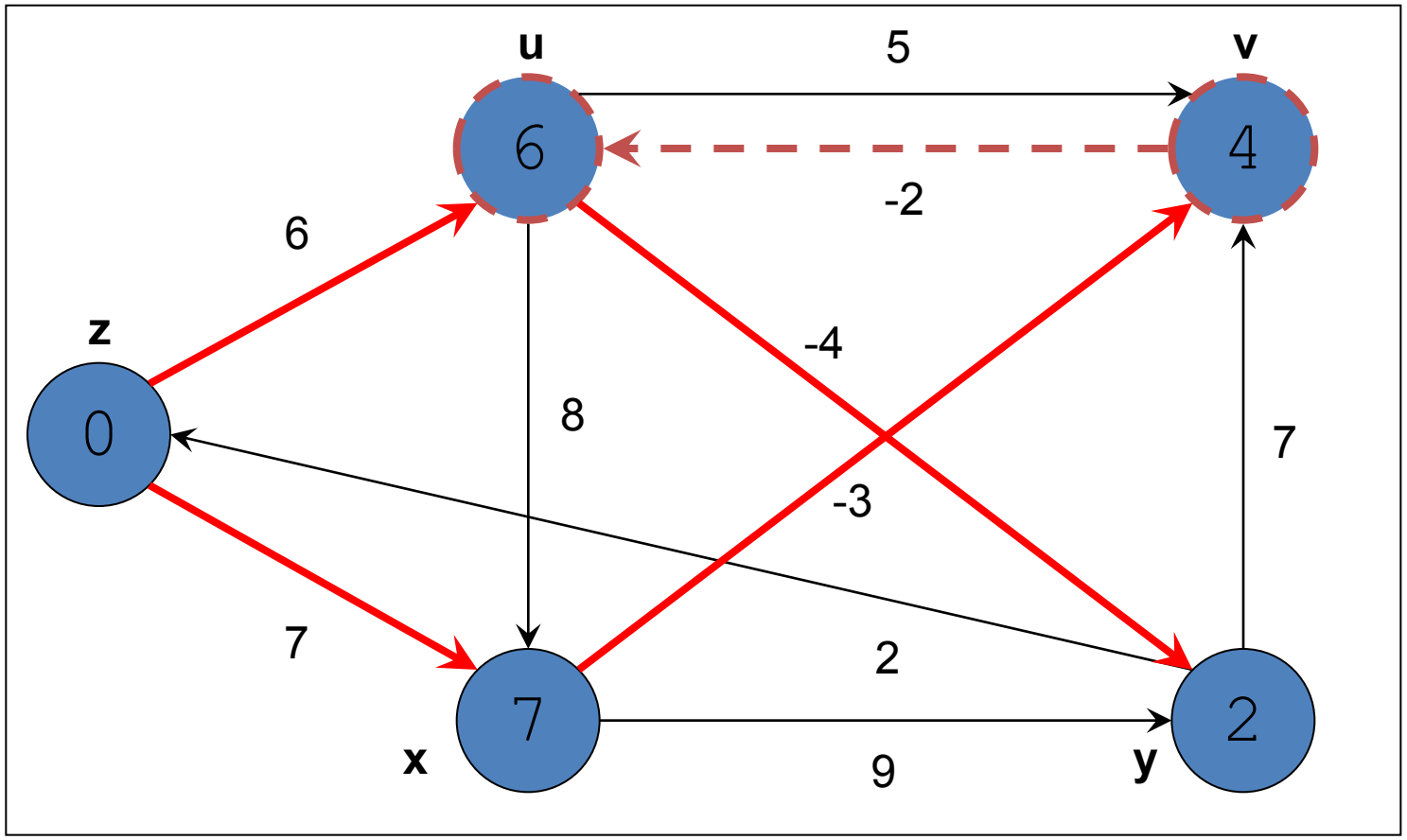
Paso 3.3

Aplicar Relax al Arco (u,y)

Pregunta: **¿ d[y] > d[u] + w(u , y) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u) ←
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	6	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**z**	**x**	**z**	**u**		**}**

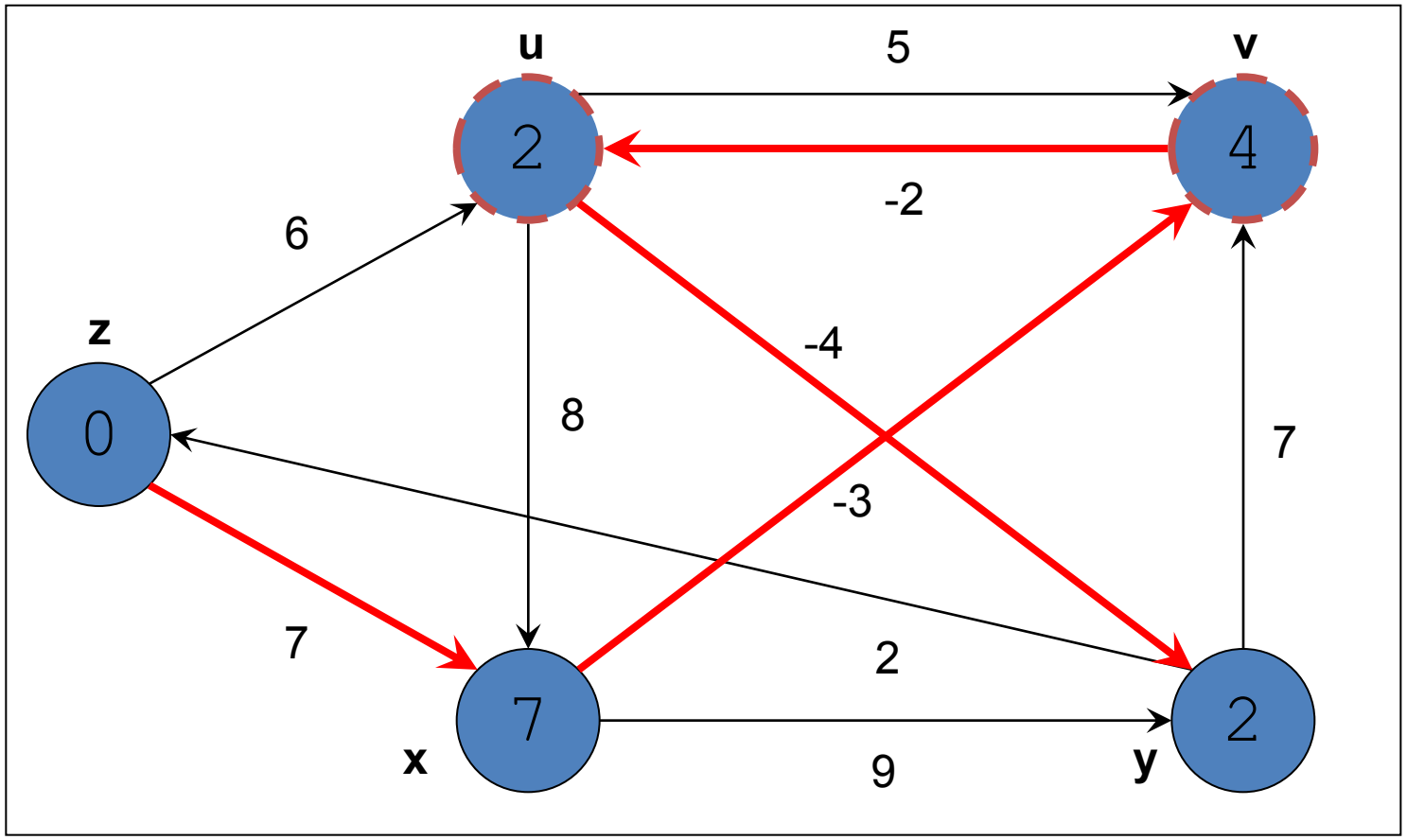
Paso 3.4

Aplicar Relax al Arco (v, u)

Pregunta: ¿ d[u] > d[v] + w(v , u) ?

Respuesta: SI

Proceso: d[u] = d[v] + w(v, u) y Π[u] = v



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u) ←
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

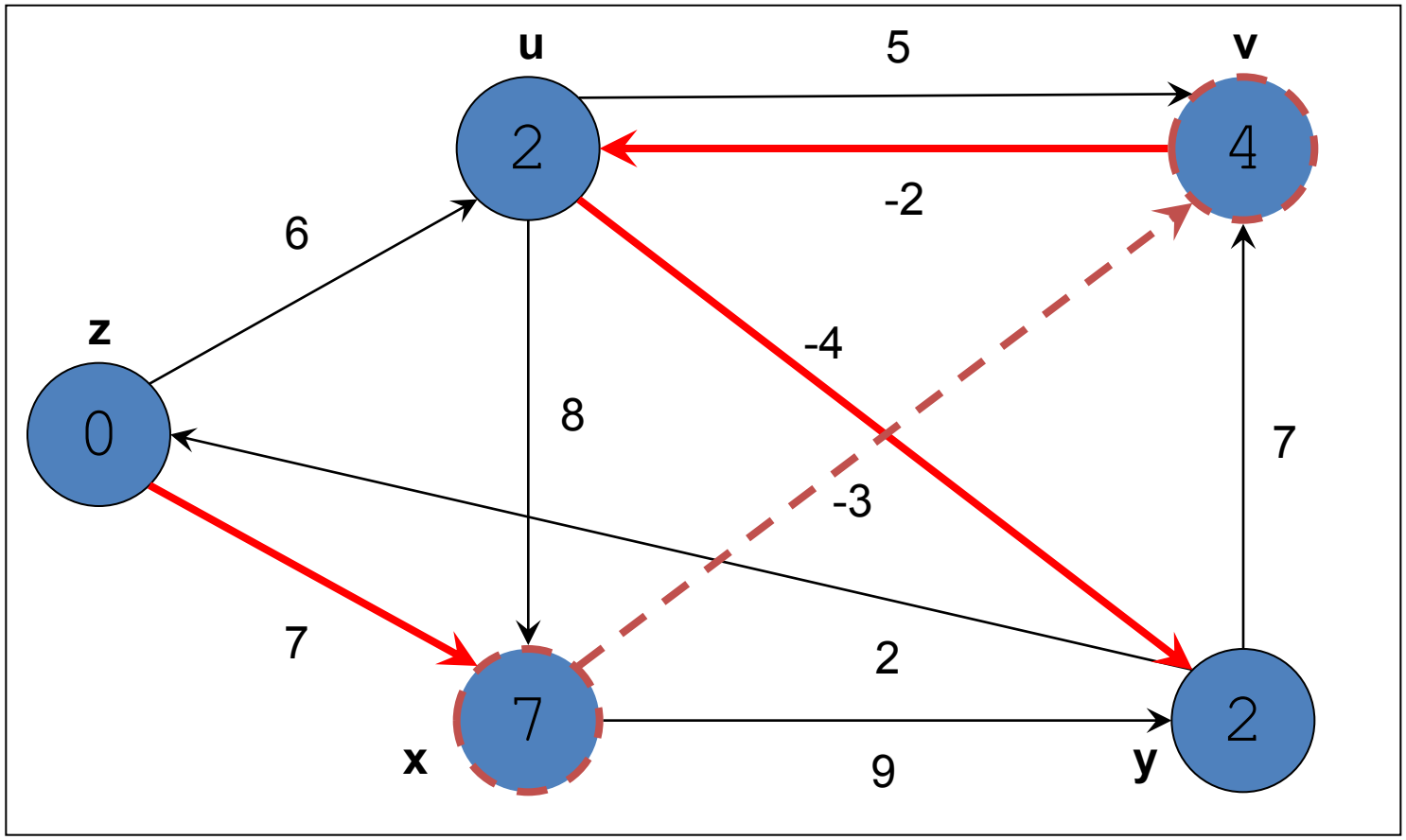
Paso 3.4

Aplicar Relax al Arco (v, u)

Pregunta: ¿ d[u] > d[v] + w(v , u) ?

Respuesta: SI

Proceso: d[u] = d[v] + w(v, u) y Π[u] = v



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v) ←
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

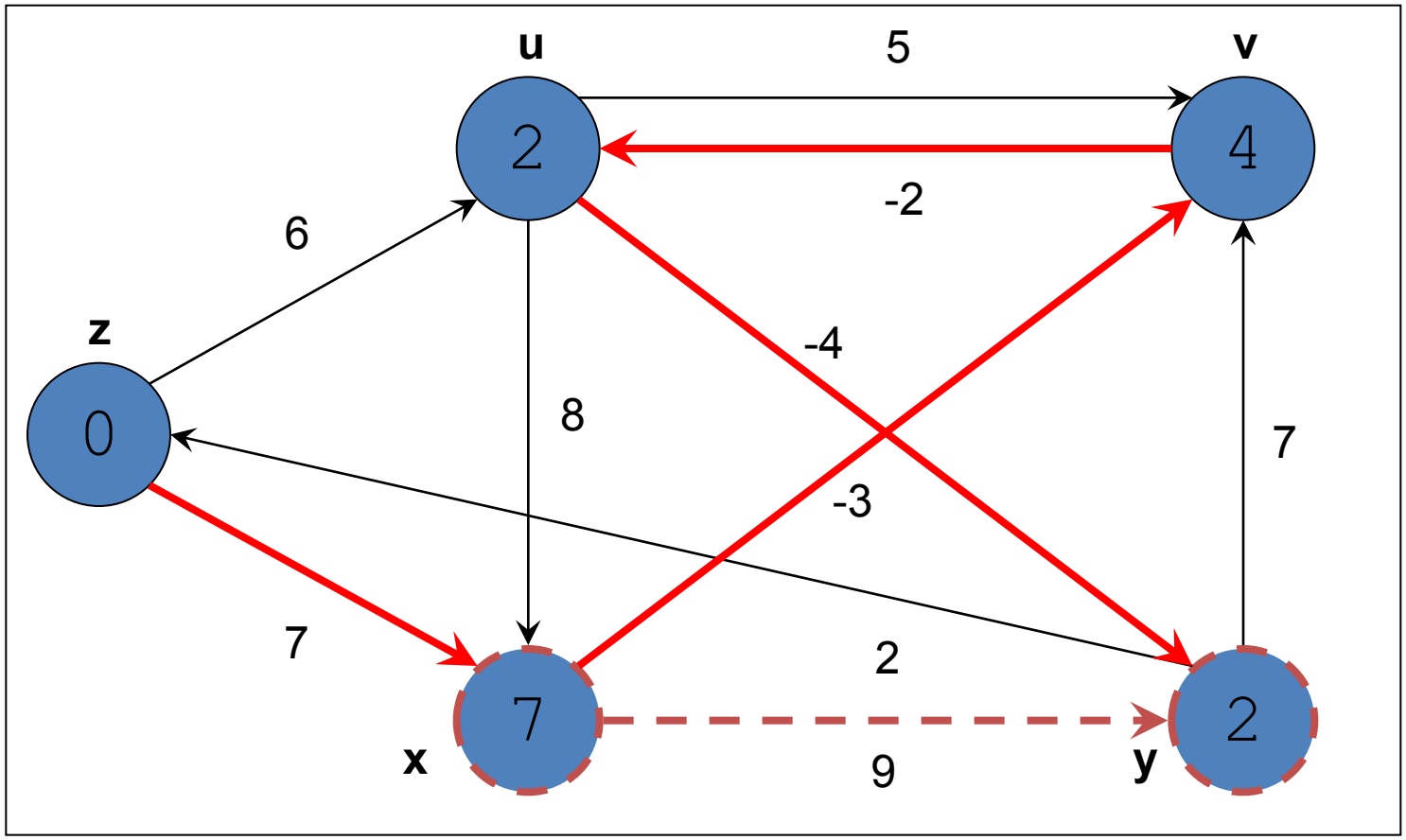
Paso 3.5

Aplicar Relax al Arco (x, v)

Pregunta: **¿ d[v] > d[x] + w(x , v) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y) ←
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

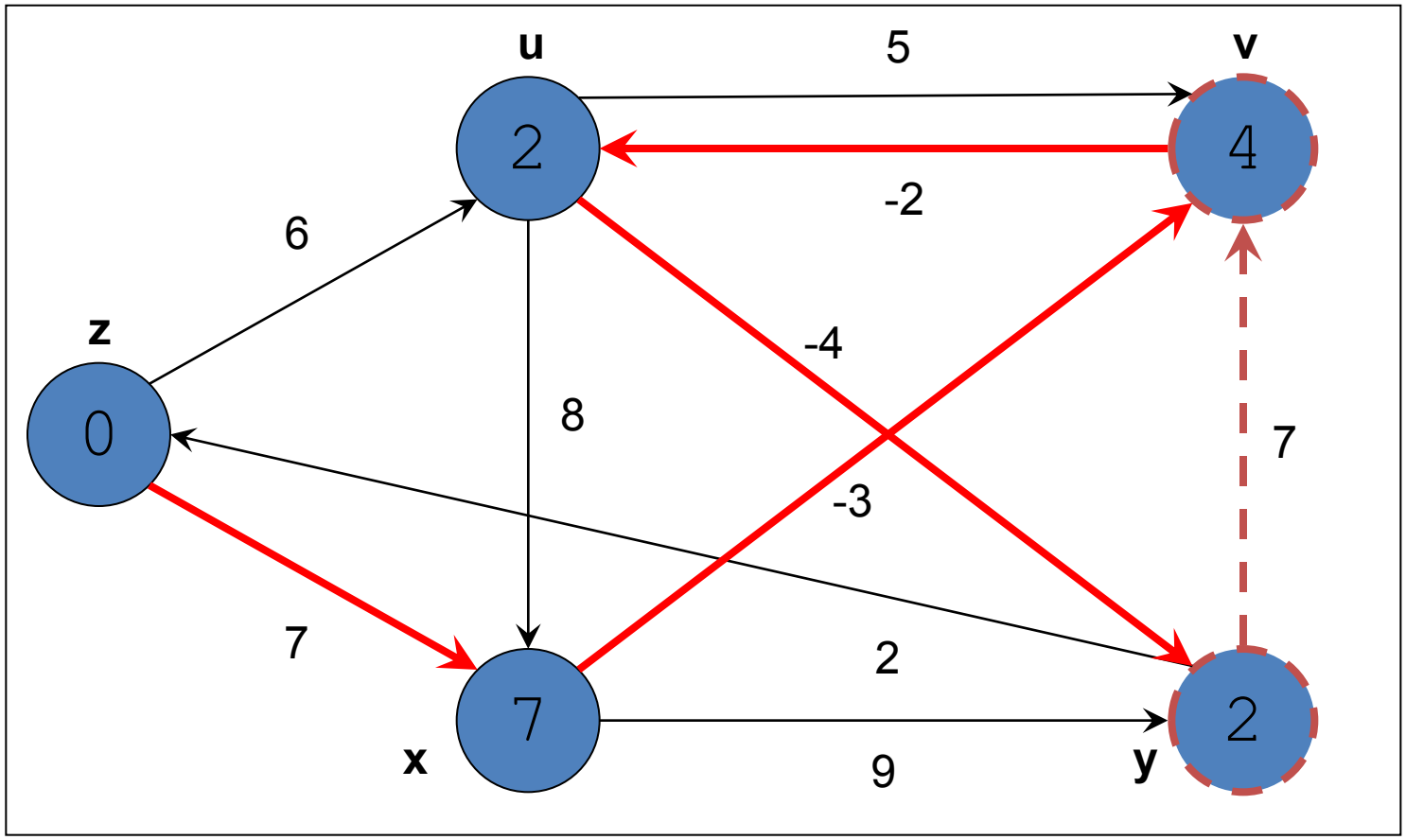
Paso 3.6

Aplicar Relax al Arco (x, y)

Pregunta: **¿ d[y] > d[x] + w(x , y) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v) ←
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

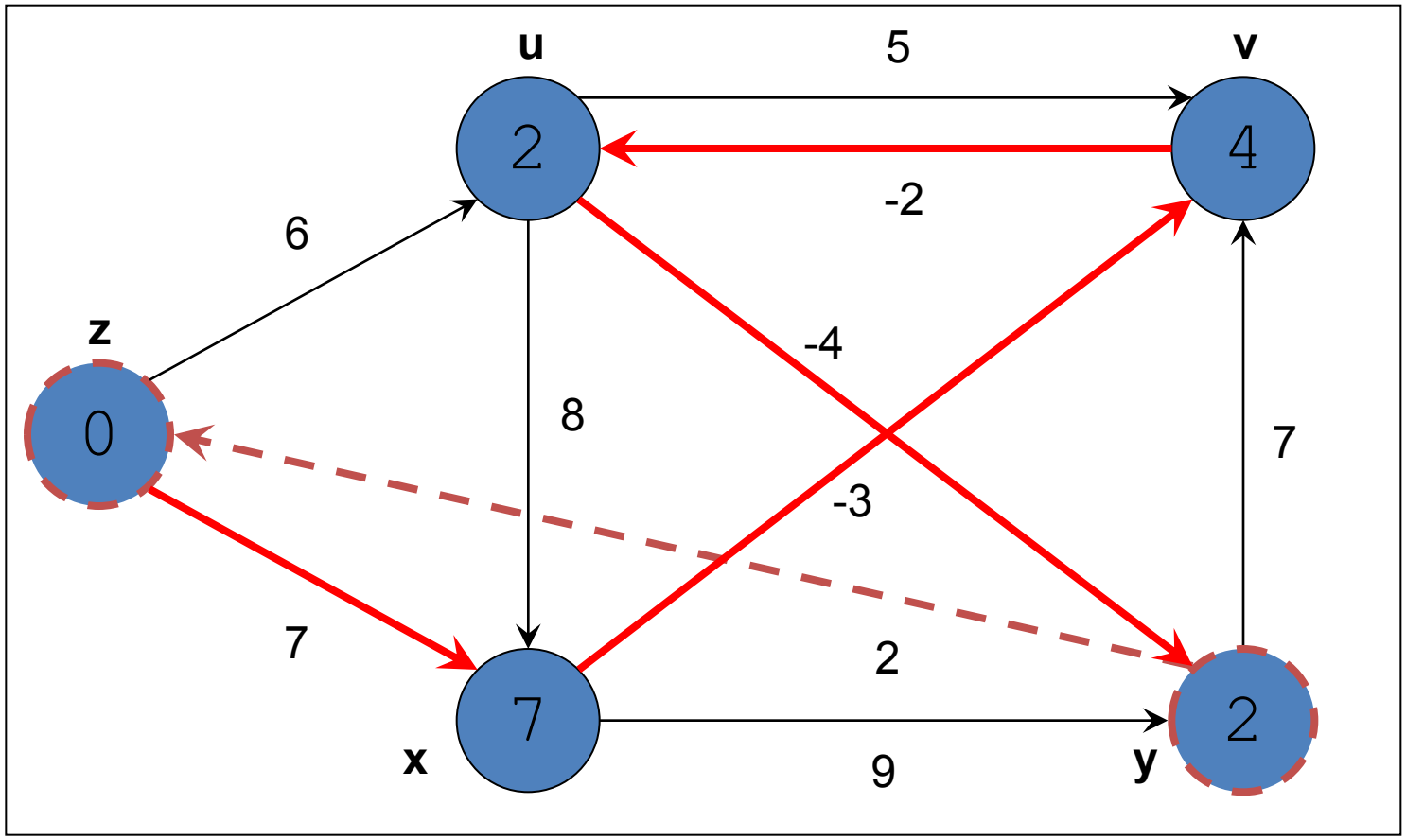
Paso 3.7

Aplicar Relax al Arco (y, v)

Pregunta: **¿ d[v] > d[y] + w(y , v) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z) ←
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

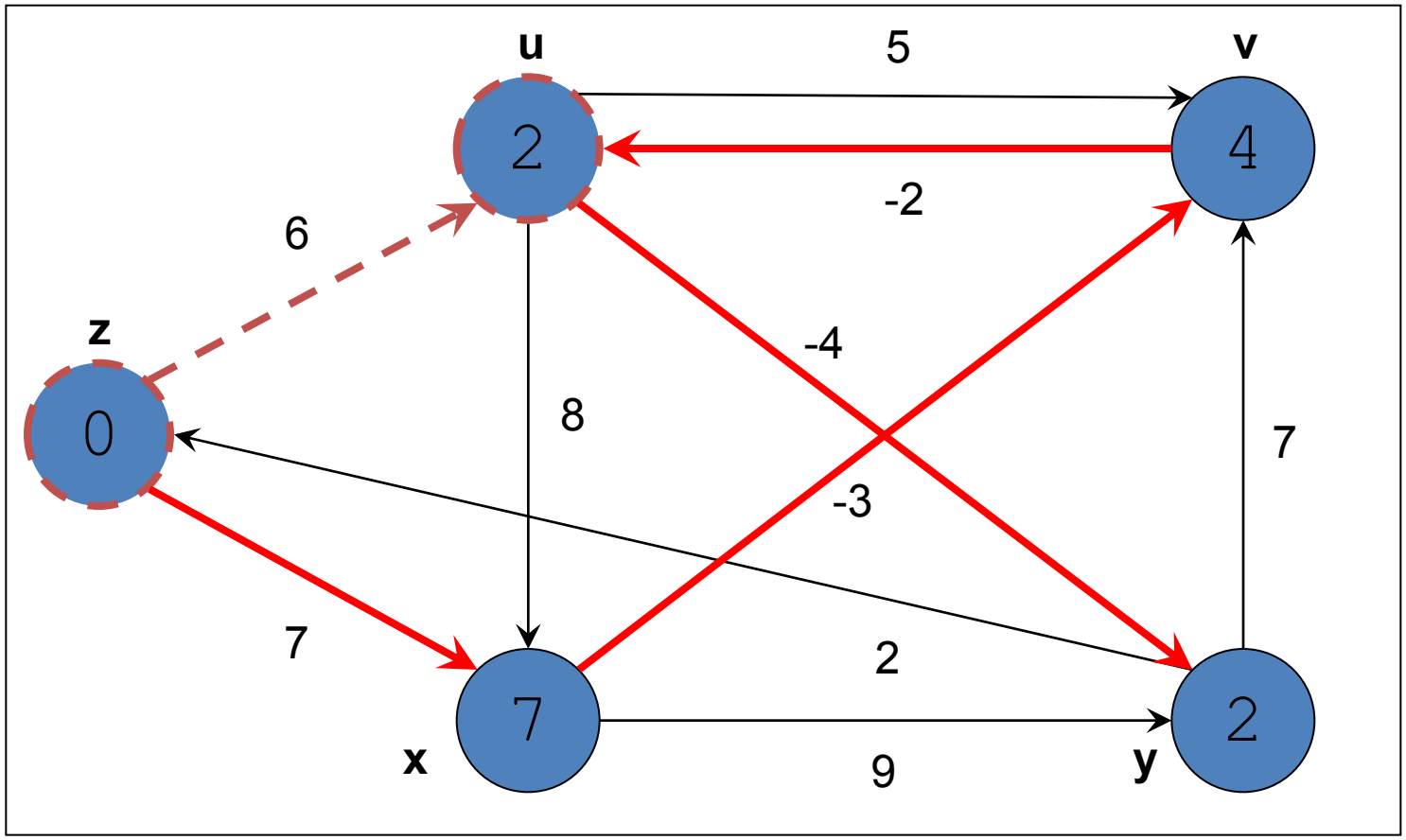
Paso 3.8

Aplicar Relax al Arco (y, z)

Pregunta: **¿ d[z] > d[y] + w(y , z) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u) ←
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

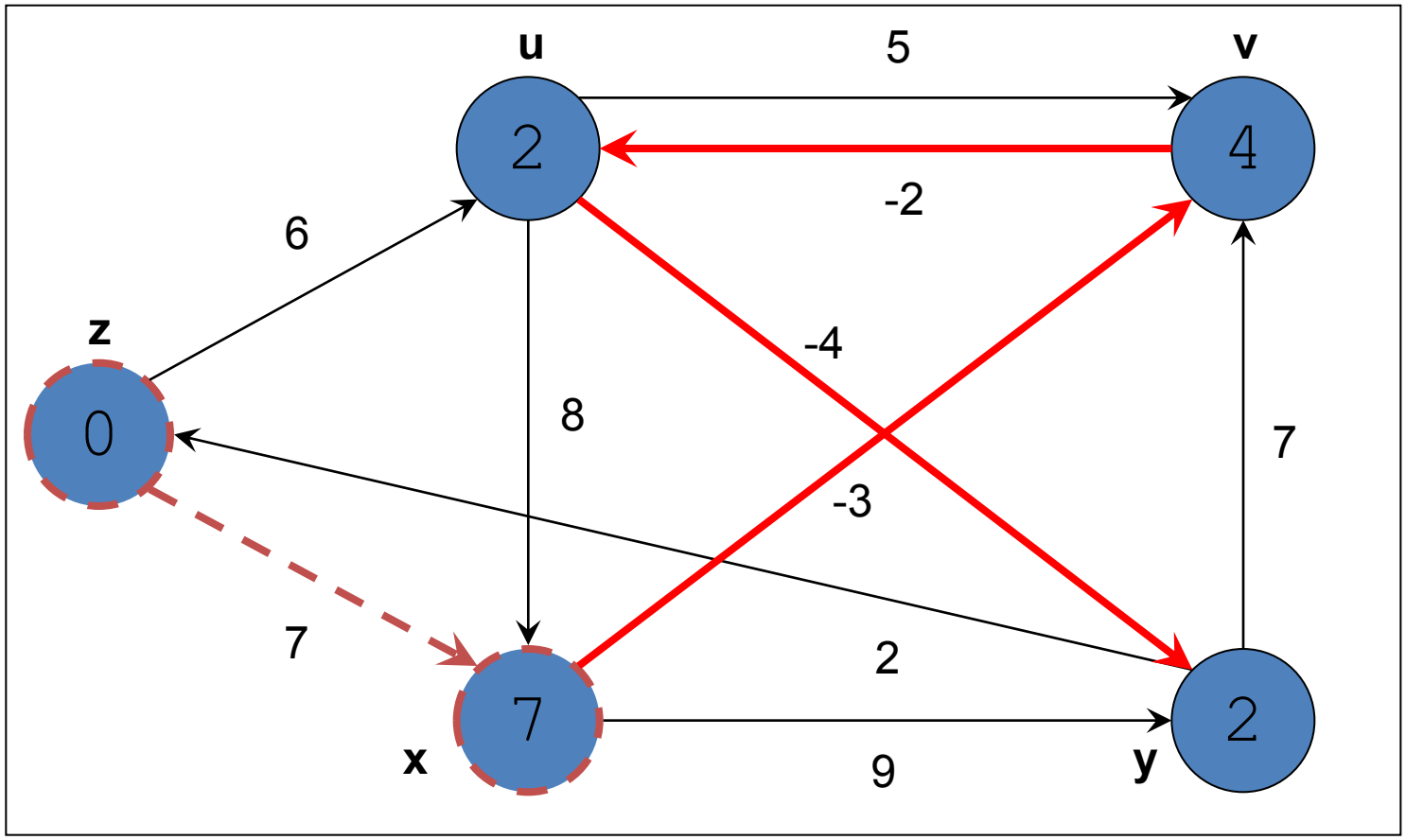
Paso 3.9

Aplicar Relax al Arco (z, u)

Pregunta: **¿ d[u] > d[z] + w(z , u) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x) ←

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

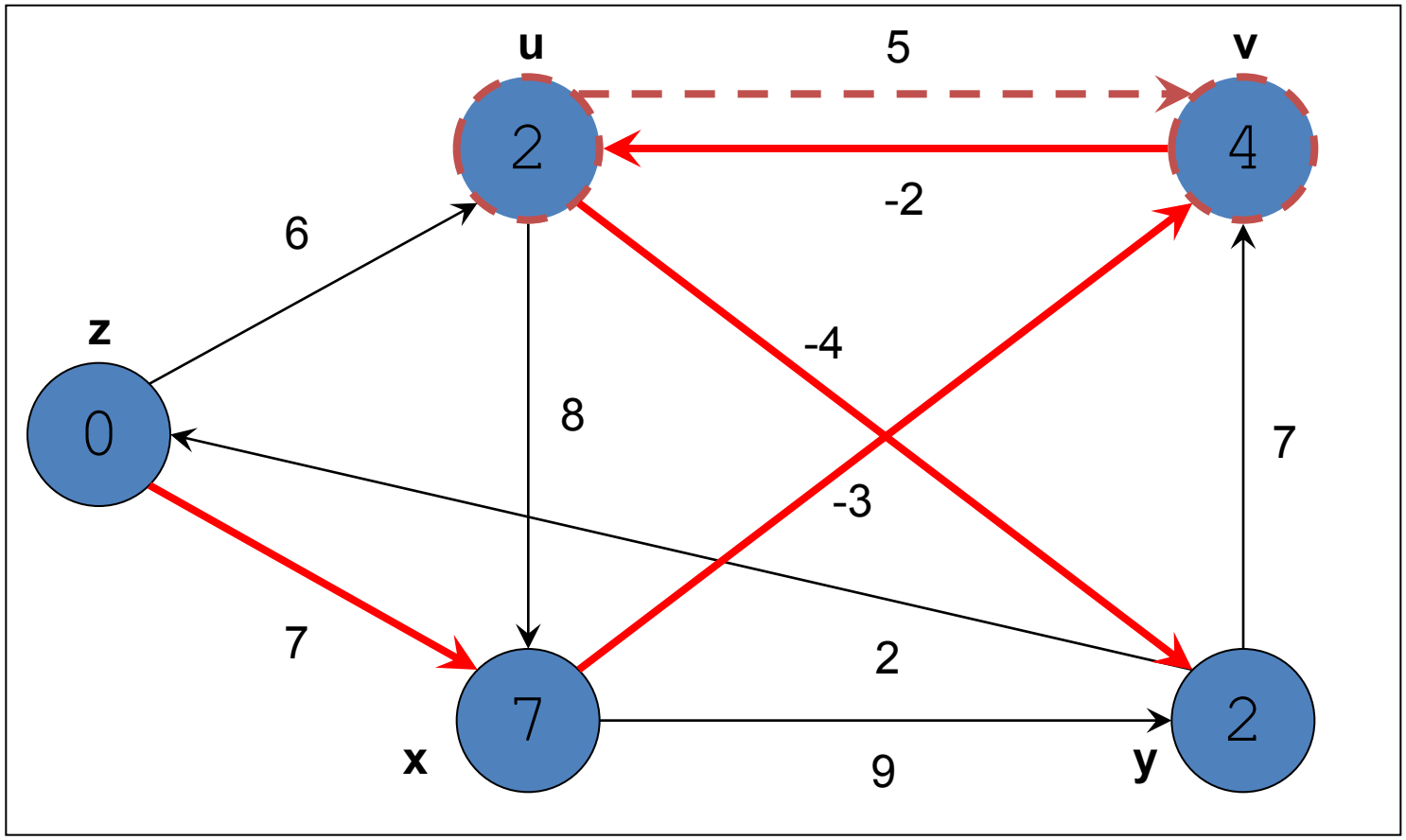
Paso 3.10

Aplicar Relax al Arco (z, x)

Pregunta: **¿ d[x] > d[z] + w(z , x) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v) ←
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

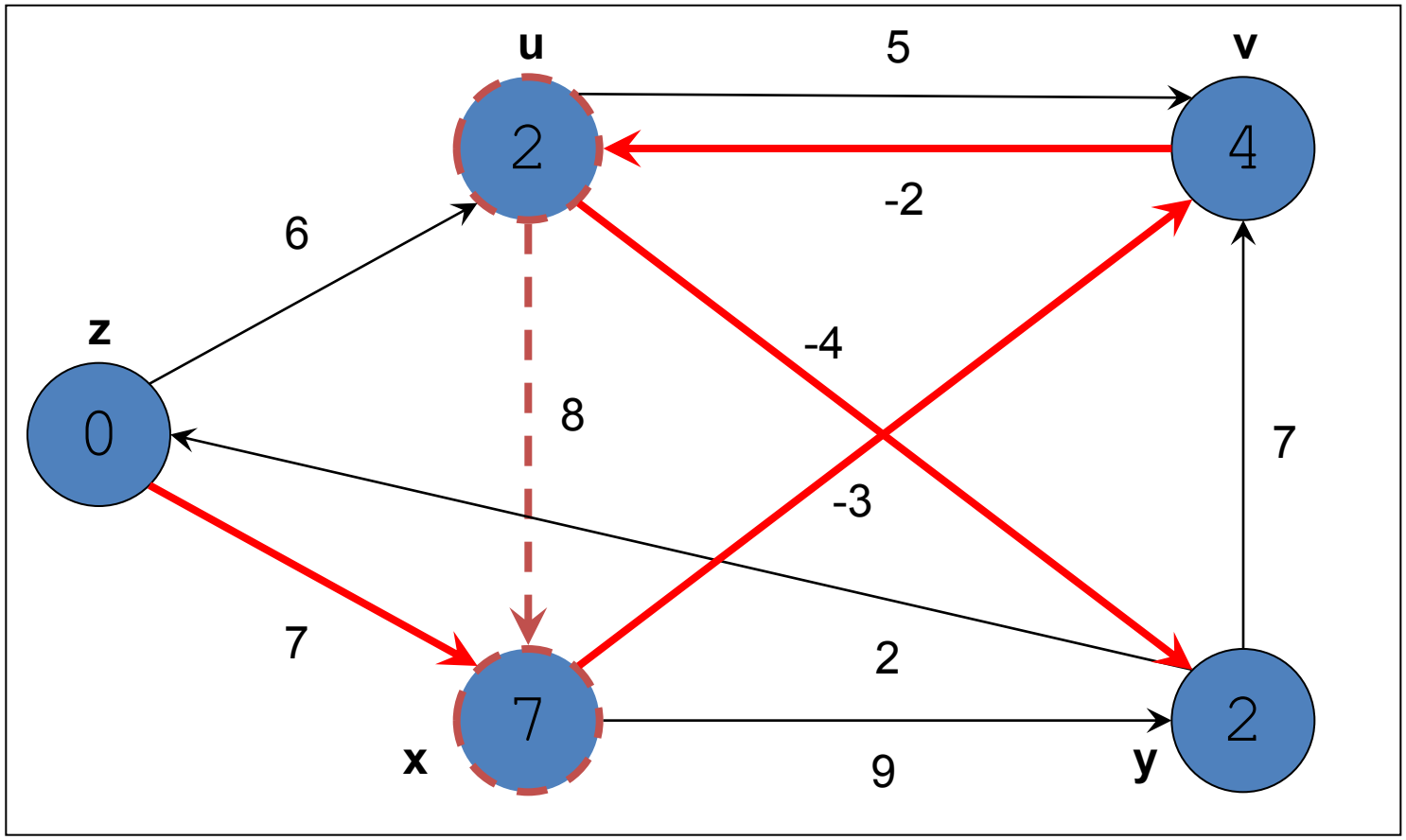
Paso 4.1

Aplicar Relax al Arco (u, v)

Pregunta: **¿ d[v] > d[u] + w(u , v) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x) ←
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

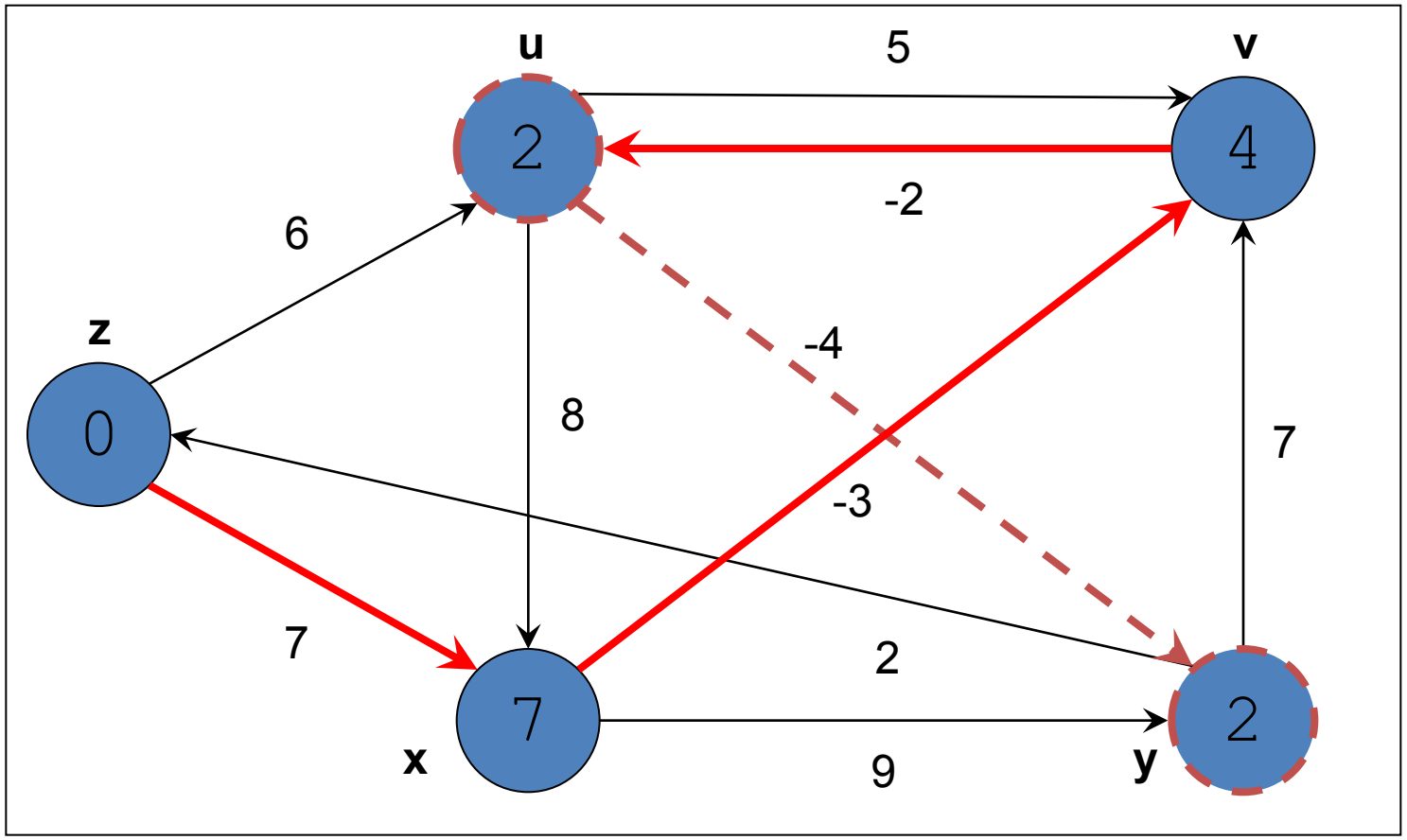
Paso 4.2

Aplicar Relax al Arco (u, x)

Pregunta: **¿ d[x] > d[u] + w(u , x) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y) ←
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

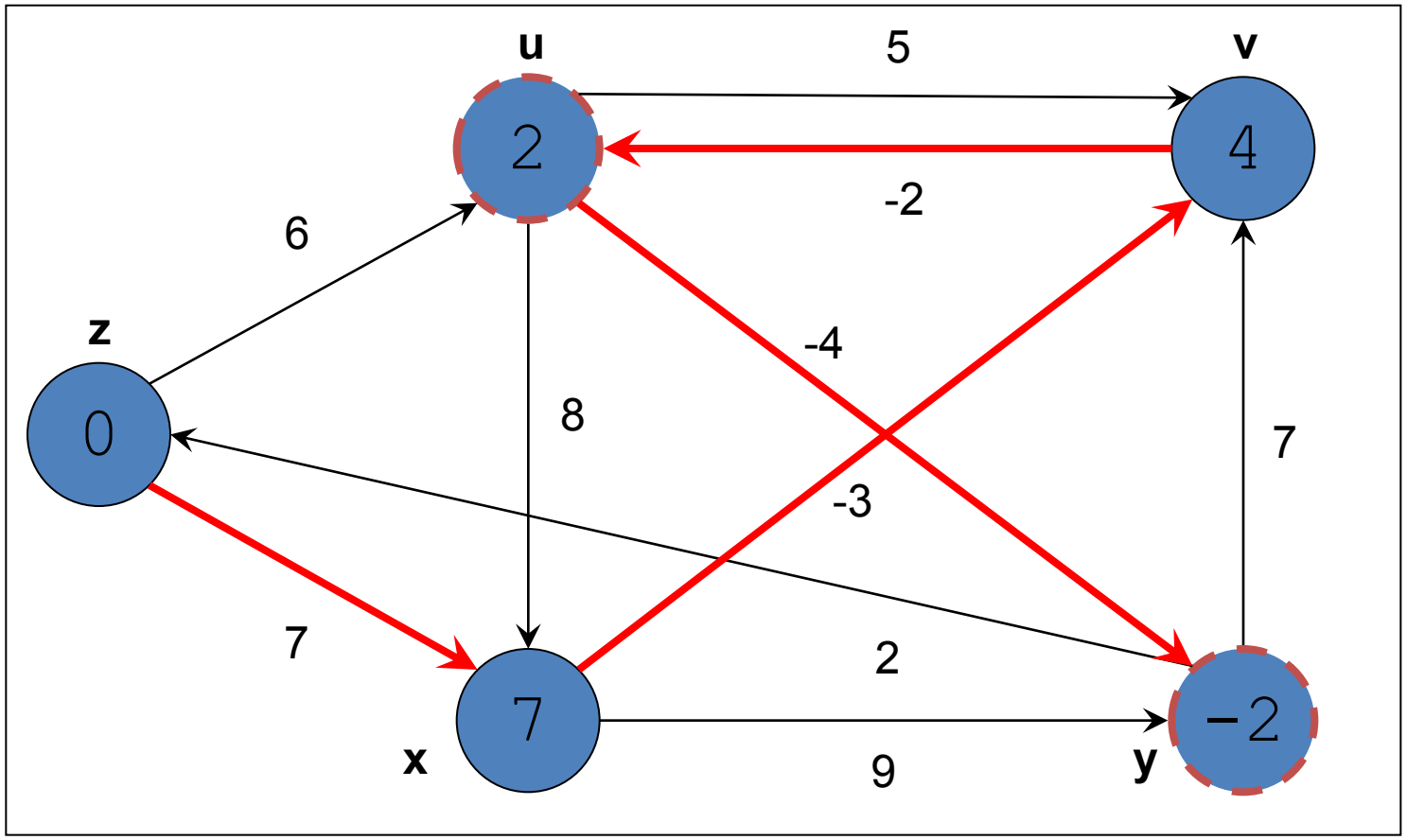
Paso 4.3

Aplicar Relax al Arco (u, y)

Pregunta: ¿ d[y] > d[u] + w(u , y) ?

Respuesta: SI

Proceso: d[y] = d[u] + w(u, y) y Π[y] = u



Lista de Arcos

(u,v)
(u,x)
(u,y) ←
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	–2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

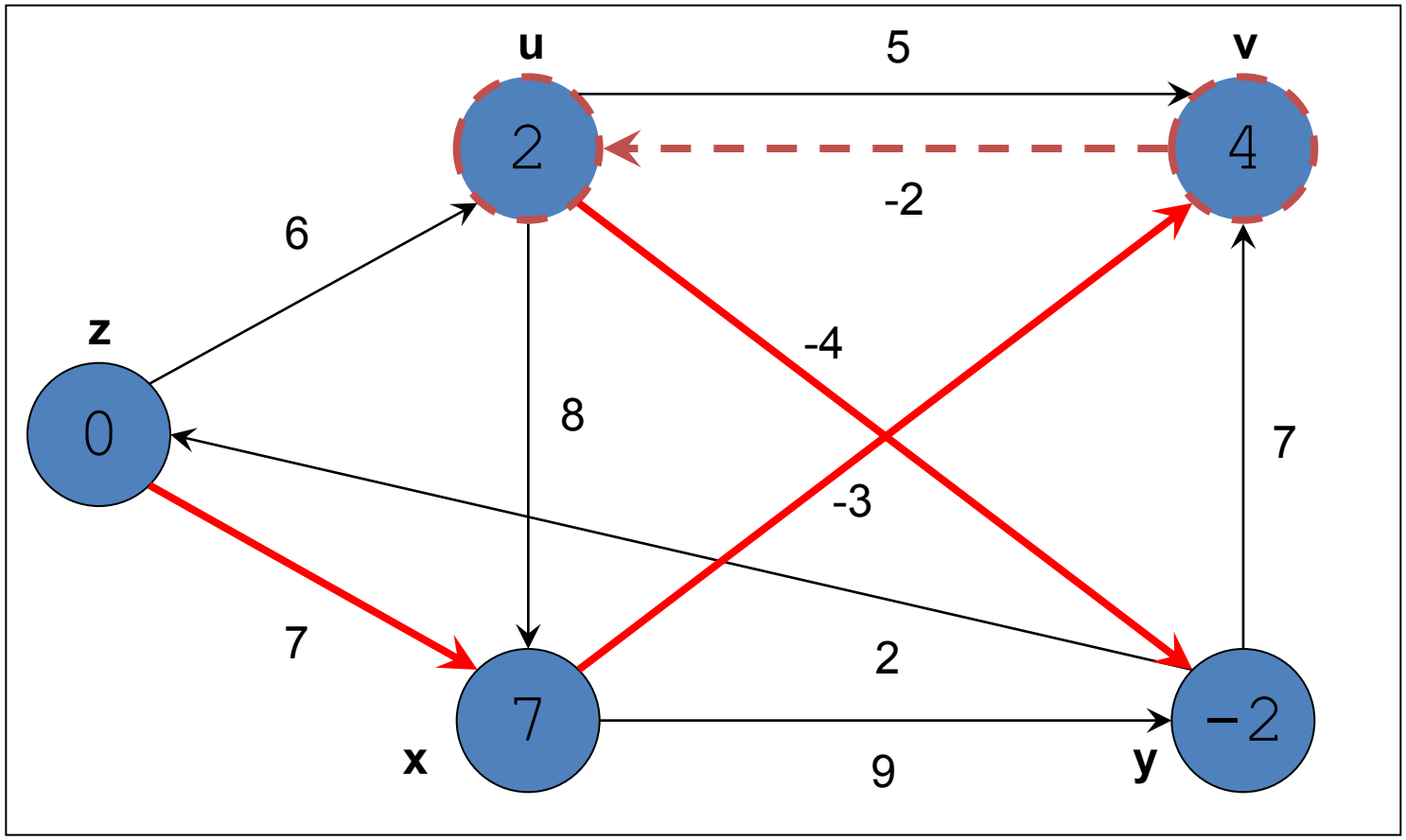
Paso 4.3

Aplicar Relax al Arco (u, y)

Pregunta: ¿ d[y] > d[u] + w(u , y) ?

Respuesta: SI

Proceso: d[y] = d[u] + w(u, y) y Π[y] = u



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u) ←
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	–2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

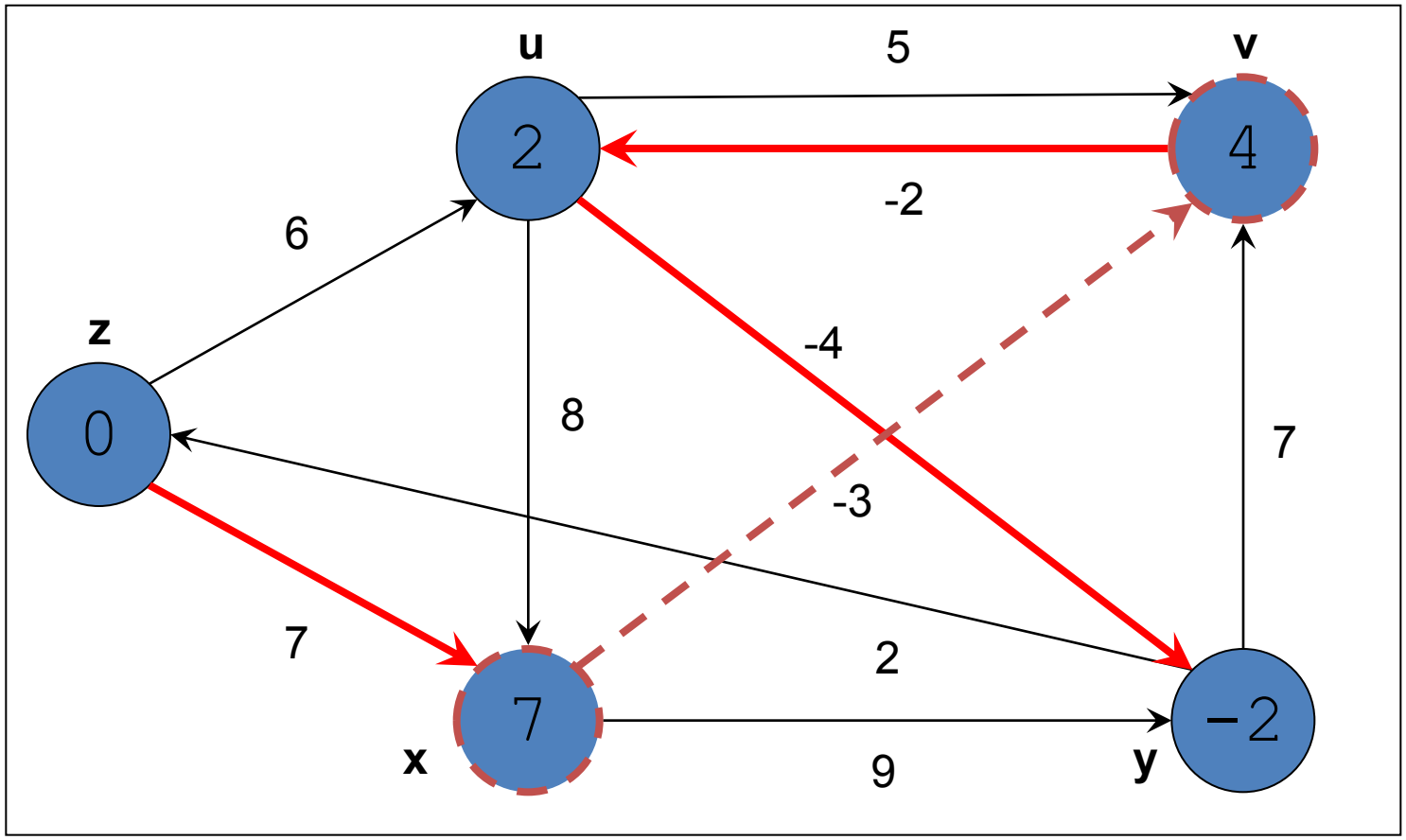
Paso 4.4

Aplicar Relax al Arco (v, u)

Pregunta: **¿ d[u] > d[v] + w(v , u) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v) ←
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	–2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

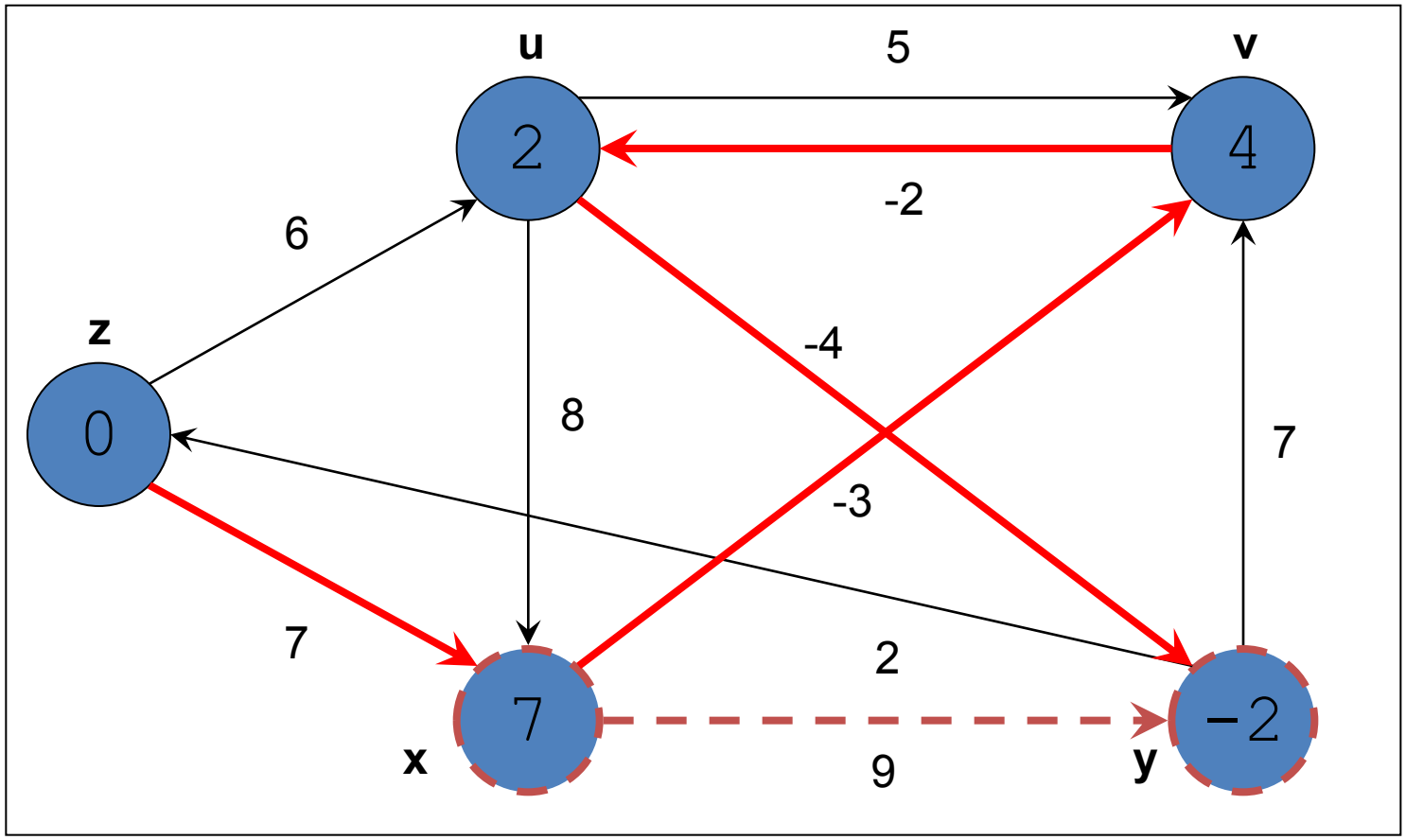
Paso 4.5

Aplicar Relax al Arco (x, v)

Pregunta: **¿ d[v] > d[x] + w(x , v) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y) ←
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	–2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

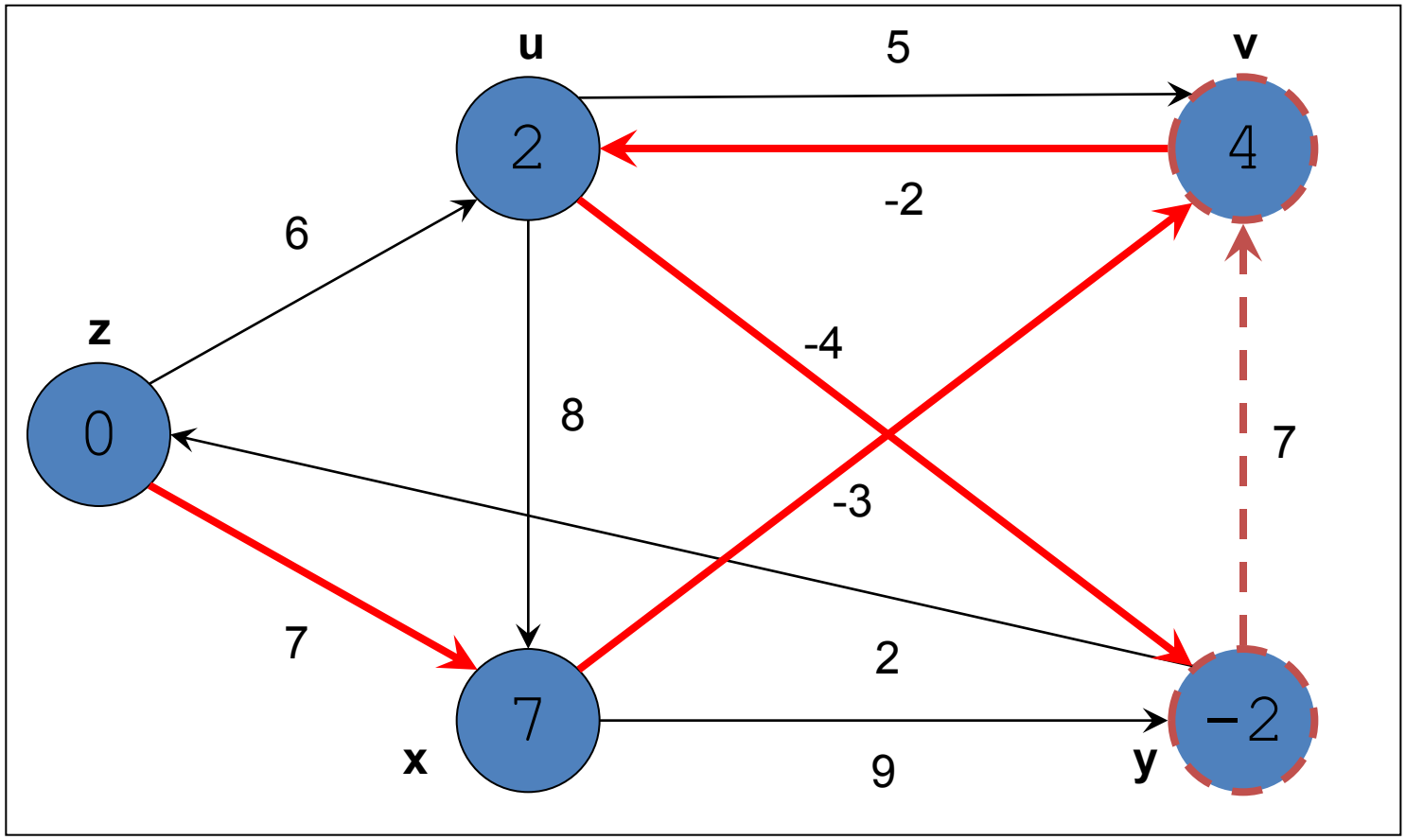
Paso 4.6

Aplicar Relax al Arco (x, y)

Pregunta: ¿ d[y] > d[x] + w(x , y) ?

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v) ←
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	–2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

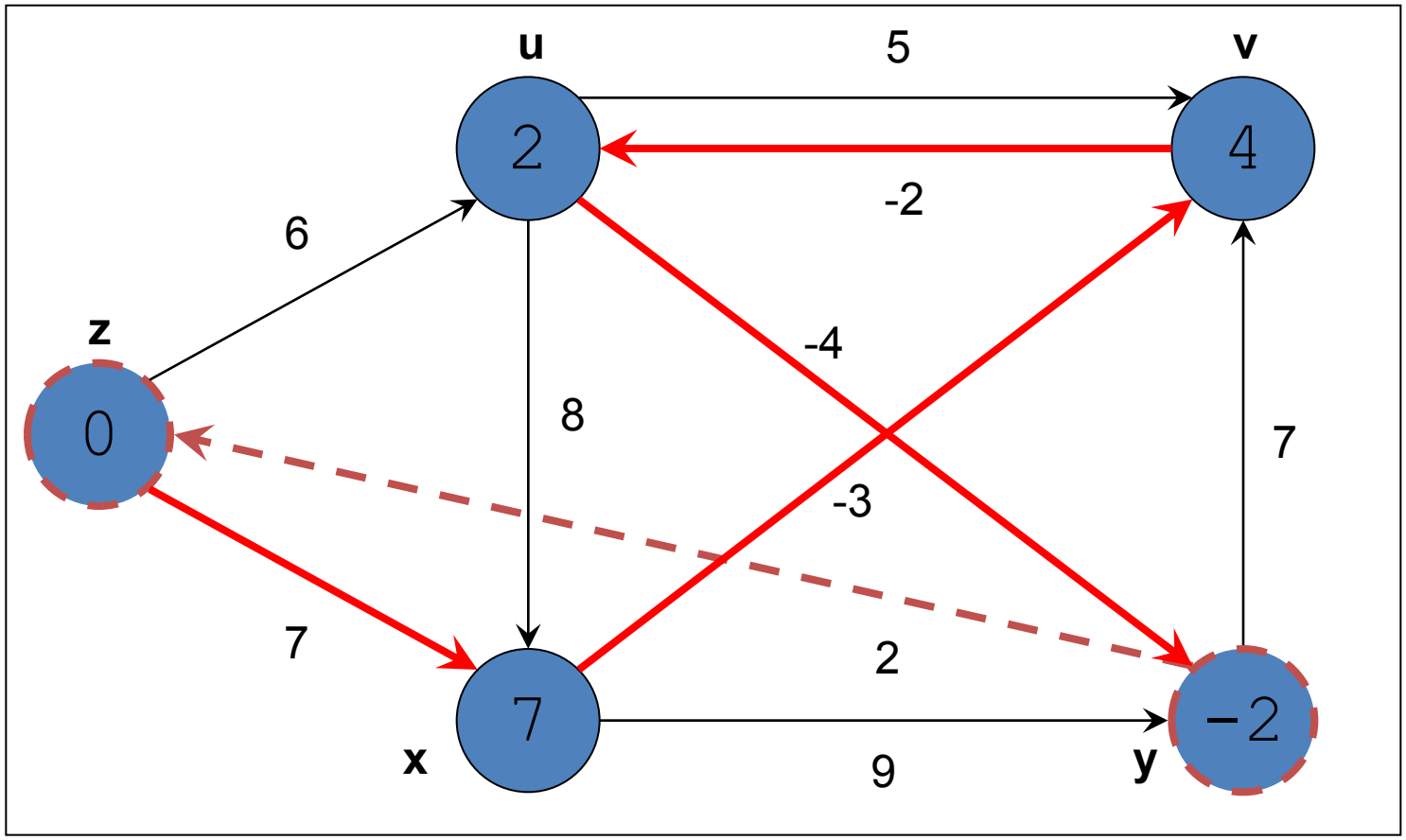
Paso 4.7

Aplicar Relax al Arco (y, v)

Pregunta: **¿ d[v] > d[y] + w(y , v) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z) ←
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	–2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

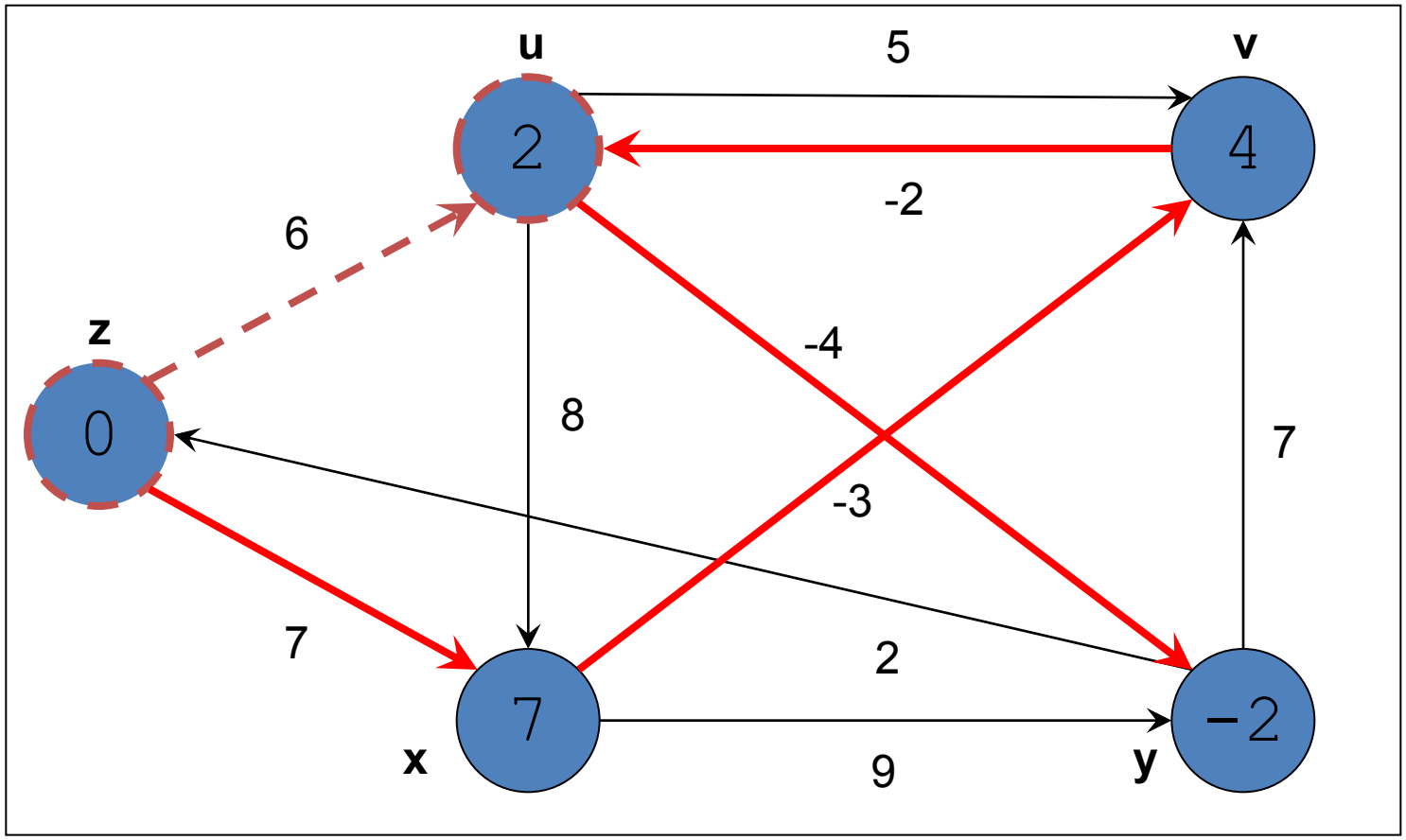
Paso 4.8

Aplicar Relax al Arco (y, z)

Pregunta: **¿ d[z] > d[y] + w(y , z) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u) ←
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	–2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

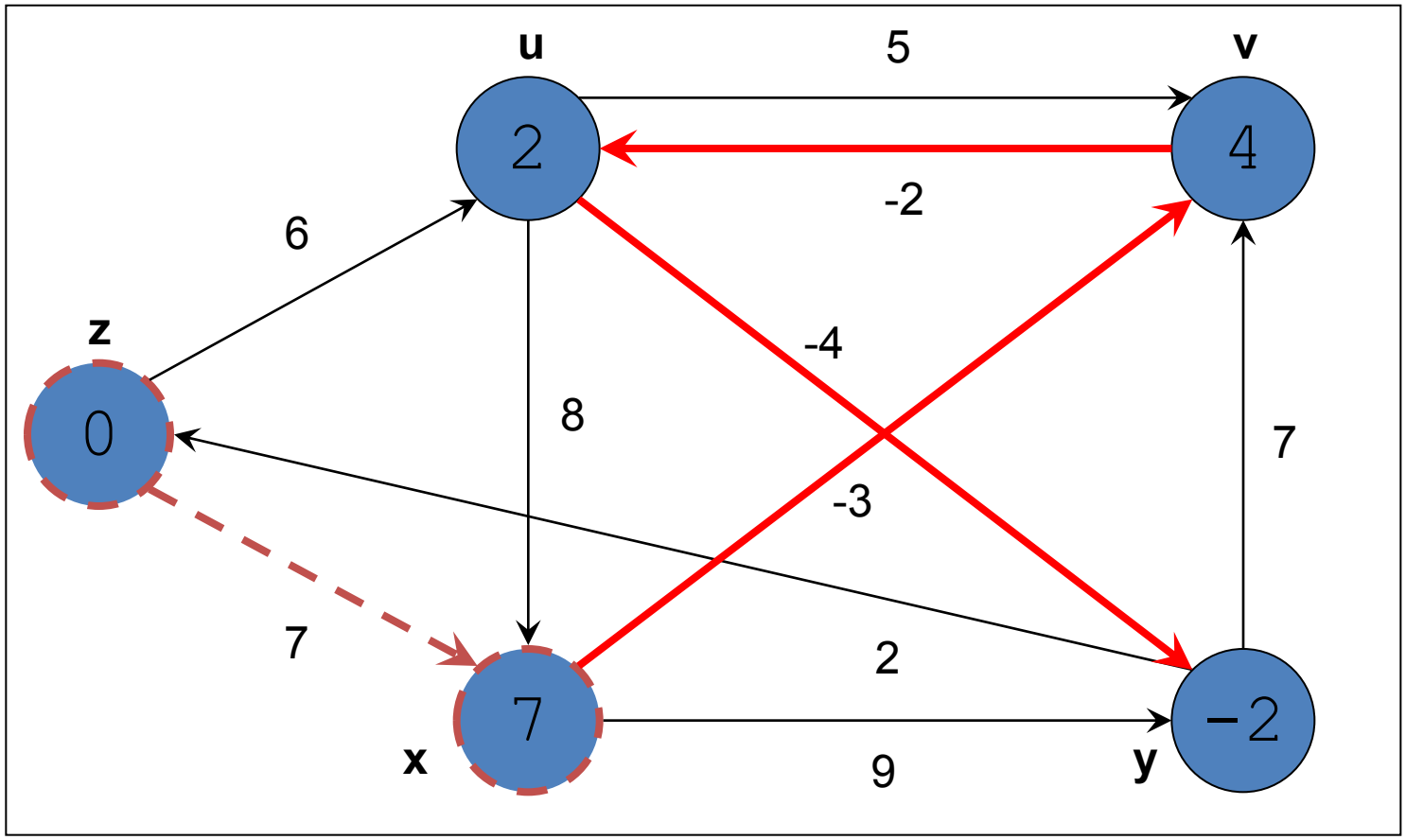
Paso 4.9

Aplicar Relax al Arco (z, u)

Pregunta: **¿ d[u] > d[z] + w(z , u) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x) ←

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	–2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

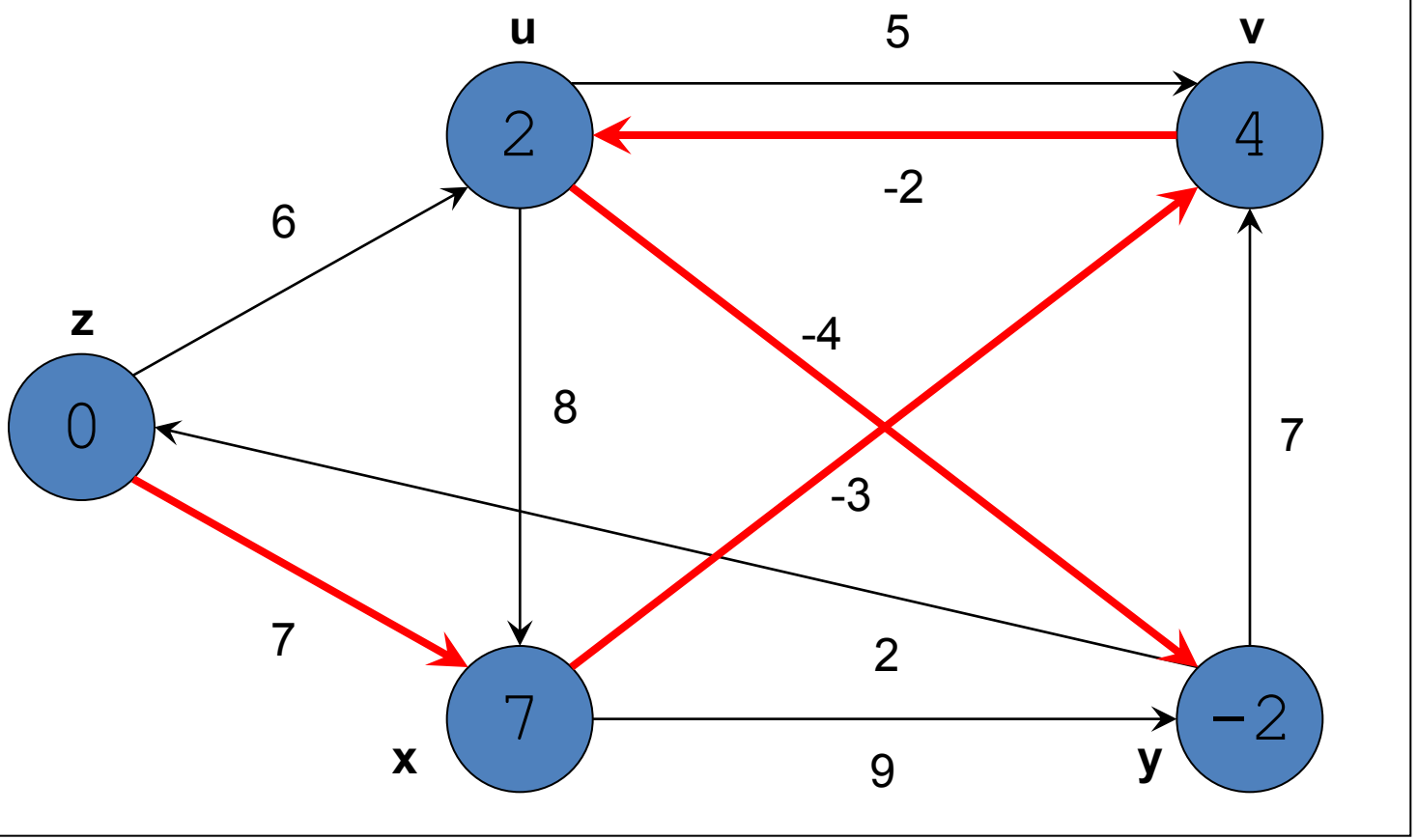
Paso 4.10

Aplicar Relax al Arco (z, x)

Pregunta: **¿ d[x] > d[z] + w(z , x) ?**

Respuesta: NO

Proceso: **No se hace nada.**



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	–2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

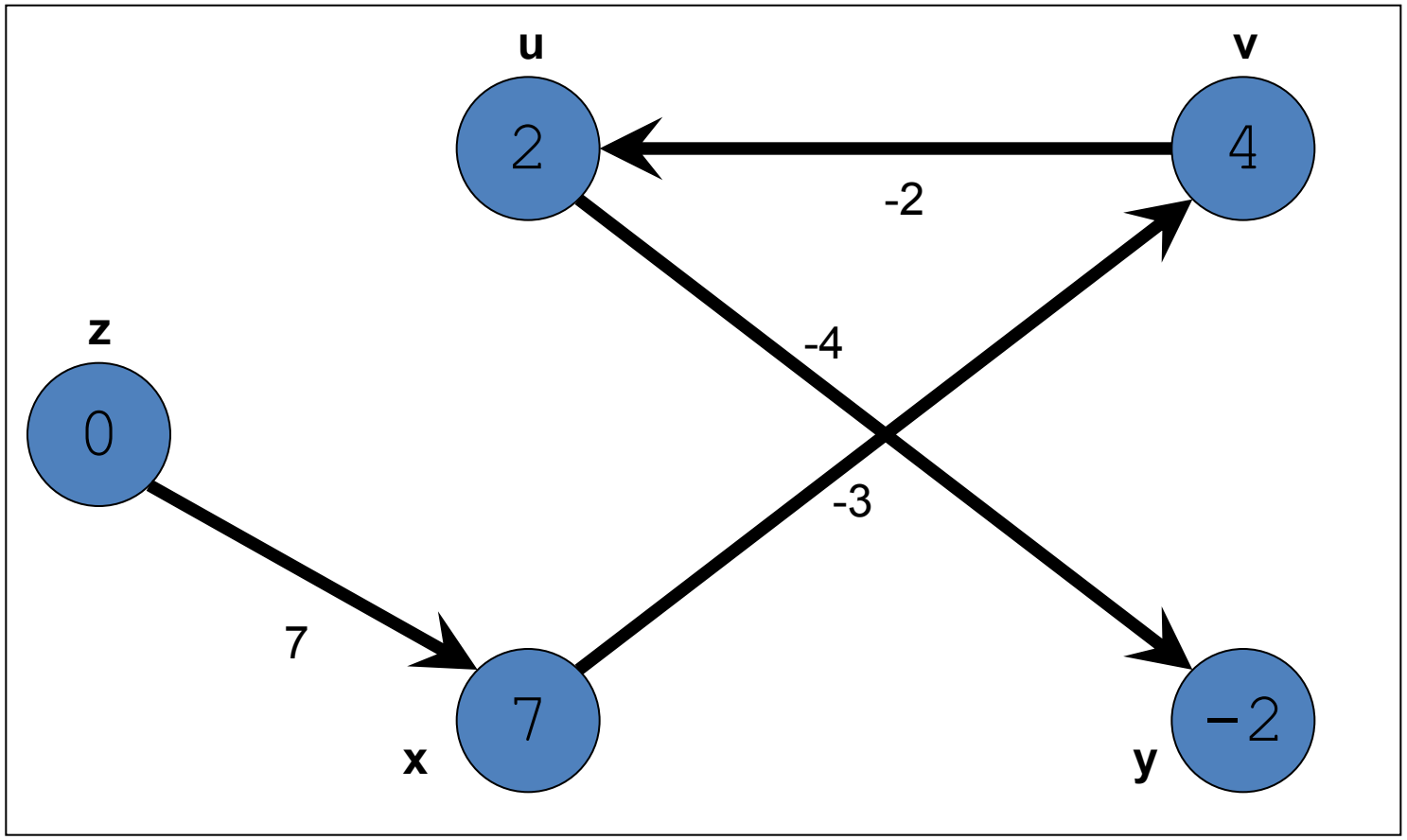
Paso 5.0

Verificar en cada arco que se cumple la condición:

d[Vf] <= d[Vi] + w(Vi , Vf)

Si no se cumple:

=> NO EXISTE SOLUCIÓN.



Lista de Arcos

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

V	**[**	**]**	**=**	**{**	**u**	**v**	**x**	**y**	**z**	**}**
d	**[**	**]**	**=**	**{**	2	4	7	–2	0	**}**
Π	**[**	**]**	**=**	**{**	**v**	**x**	**z**	**u**		**}**

SOLUCIÓN

Prueba de Funcionamiento

Lema:

- Sea G(V,E) un grafo con peso w: E$\rightarrow$R, dirigido.
- Sea s$\subset$V el vertice origen y sea s~>u$\rightarrow$v un camino corto en G para algun par de vertices u, v $\subset$ V.
- Suponga que G se inicia con INITIALIZE(G,s) y luego existe una secuencia de pasos de relajación RELAX(u,v,w).

Si d[u]=δ(s,u) en cualquier momento previo de la llamada con (u,v), entonces d[v]= δ(s,v) todo el tiempo despues de la llamada.

Prueba:

Sabiendo que d[u]= δ(s,u) en algún momento antes de relajar (u,v), y que esto se mantiene al dejar el arco (u,v):

$$d[v] \leq d[u] + w(u,v)$$
$$= \delta(s,u) + w(u,v)$$
$$= \delta(s,v)$$

Prueba de Funcionamiento

Lema:

Al terminar BELLMAN-FORD, se tiene d[v]= δ(s,v) $\forall$ v$\subset$V y s~>v.

Prueba:

- Sea v un nodo alcanzable desde s, y p=<v_0, v_1, …, v_k> un camino corto desde s a v, donde v_0=s y v_k=v.
- Como hay |V|-1 pasadas es suficiente probar por inducción.

  Base: d[v_0]= δ(s,v_0)=0, y se mantiene…

  Paso inductivo: d[v_{i-1}]= δ(s,v_{i-1}) luego de la (i-1)esima pasada, el arco (v_{i-1},v_i) se relaja en la i-esima pasada.

Por el lema anterior se concluye que d[v_i]= δ(s,v_i) y se mantiene…

APLICACIONES DEL ALGORITMO

- Una versión del Algoritmo del Bellman-Ford distribuido se usa para **diseñar protocolos de encaminamiento basados en vector de distancia**s (redes de comunicación).

- Ejemplo el protocolo de encaminamiento de información (**RIP**).

  http://neo.lcc.uma.es/evirtual/cdd/applets/BellmanFord/Example3.html

Algoritmo de Johnson para grafos esparcidos

Algoritmo de Johnson para grafos esparcidos

- Es mejor asintóticamente que el algoritmo de Floyd-Warshall para grafos esparcidos.

- Utiliza como subprogramas el algoritmo de Dijkstra y el de Bellman-Ford (combina).

- **Utiliza la técnica de reasignar pesos,** que consiste en:
  - Si todos los pesos de los arcos de G son positivos, entonces se usa Dijkstra y se corre para cada nodo de G
  - Si G tiene pesos negativos, se calcula un nuevo conjunto de pesos no negativos w' que permitan usar el mismo método.

Algoritmo de Johnson para grafos esparcidos

Los nuevos pesos positivos calculados deben cumplir:

- ∀ par de nodos u, v ∈ N, el C+C de u a v usando w es también el C+C de u a v usando w'
- ∀ arcos (u, v), el nuevo peso w'(u, v) es no negativo

Lema: La reasignación de pesos no cambia los C+C

- Dado un digrafo etiquetado G con $w : A \rightarrow \Re$ y sea $h:N \rightarrow \Re$, para cada arco (u, v) ∈ A se define
$$w'(u, v) = w(u, v) + h(u) - h(v) \geq 0$$
- Sea p = ⟨n0, n1, ..., nk⟩ el camino de n0 a nk, entonces w(p) = δ(n0, nk) si y solo si w'(p) = δ'(n0, nk).
- G tiene ciclos con peso negativos usando w si y solo si G los tiene también usando w'.

Algoritmo de Johnson para grafos esparcidos

Se calcula G' con los pesos reasignados de G según

$$w'(u, v) = w(u, v) + h(u) - h(v) \geq 0.$$

El algoritmo de Johnson regresa

- D con los pesos de los C+C o
- despliega la imposibilidad de su cálculo por tener ciclos de peso negativo

Se asumen los nodos numerados de 1 a n

Para crear G’, se crea un nodo ficticio 0 y un arco desde él hasta el resto de los nodos de G, con peso inicial 0.

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Algoritmo de Johnson para grafos esparcidos

Versión 1.0

C+CJohnson():Arreglo(n x n)De [Real]

{pre: n > 0 } {pos: n > 0 }

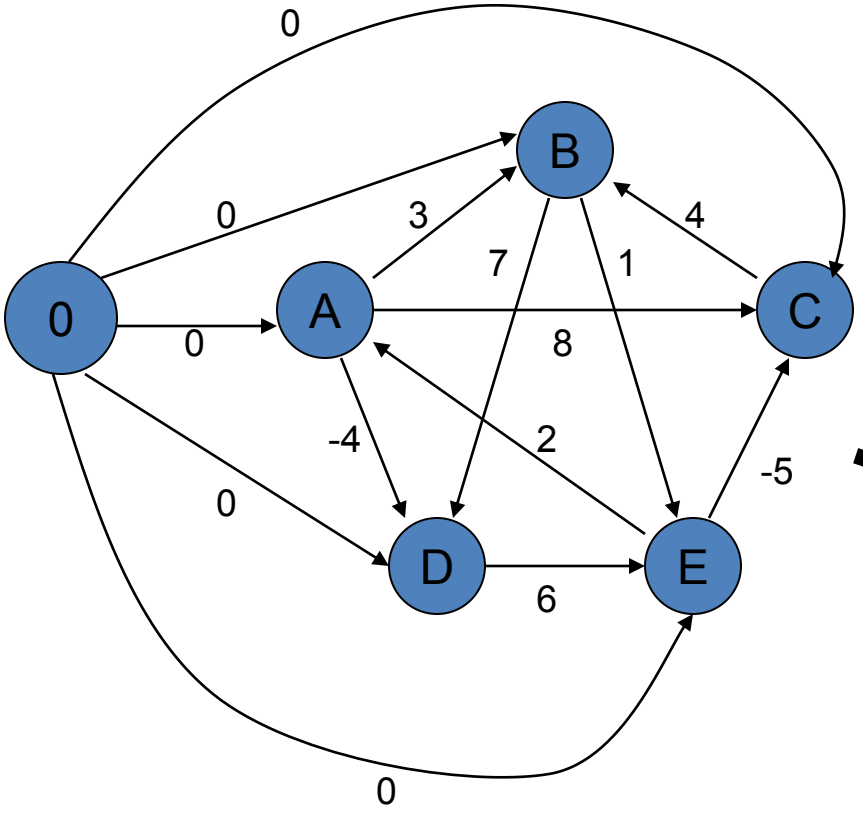
1	Calcular G' con N' = N U { s } y A' = A U {(s, v): v∈N}	-u, v: Entero. Indican nodos del grafo y son los subíndices de os lazos. -D: Arreglo(n x n)De [Real]. Matriz con los pesos de los C+C actuales. -G', N', A': Grafo auxiliar con un nodo adicional (s). -h: Arreglo(n)De [Real]. contiene los valores para reasignar los pesos.
2	Si (¬G'.C+CBellmanFord(s)) entonces despliegue "G contiene ciclos de peso negativo" sino [h(v) = δ(s, v) calculado por Bellman-Ford] v ∈ N [w'(u,v) = w(u,v)+h(u)-h(v)] (u, v) ∈ A [C+CDijkstra(u) para calcular δ'(u,v) [D(u, v) = δ'(u, v) + h(v) - h(u)] v ∈ N] u ∈ N fsi	
3	regrese D	

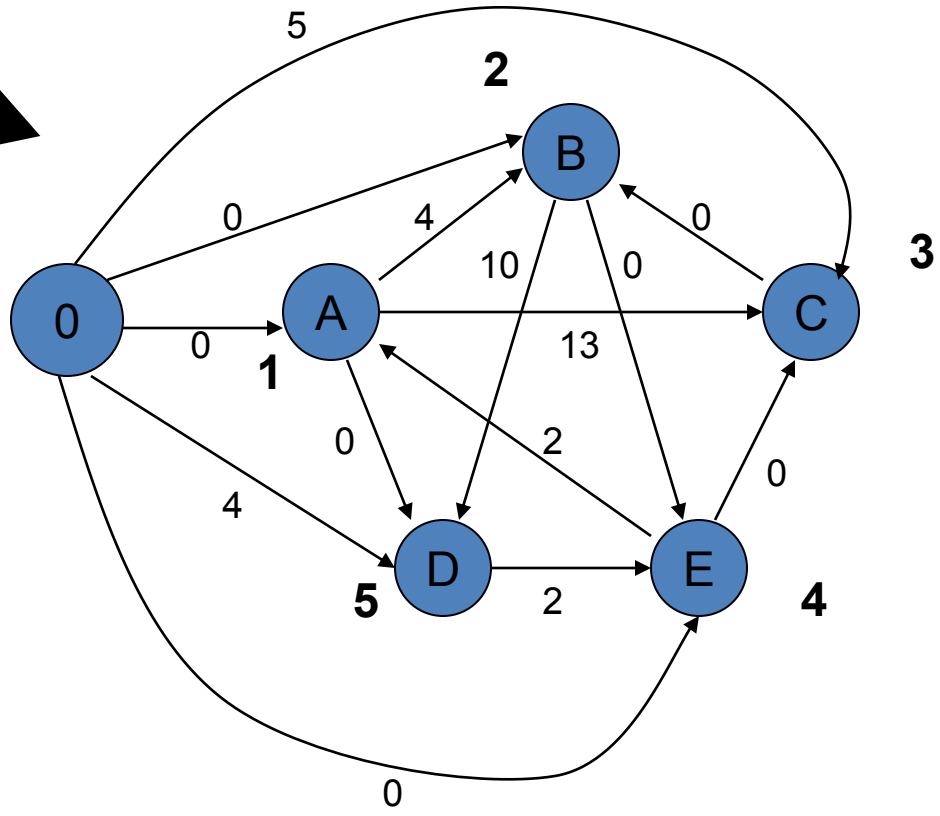
T(n) = O(N2 lg N + N A) si el algoritmo de Dijkstra está implantado con montículos de Fibonacci.

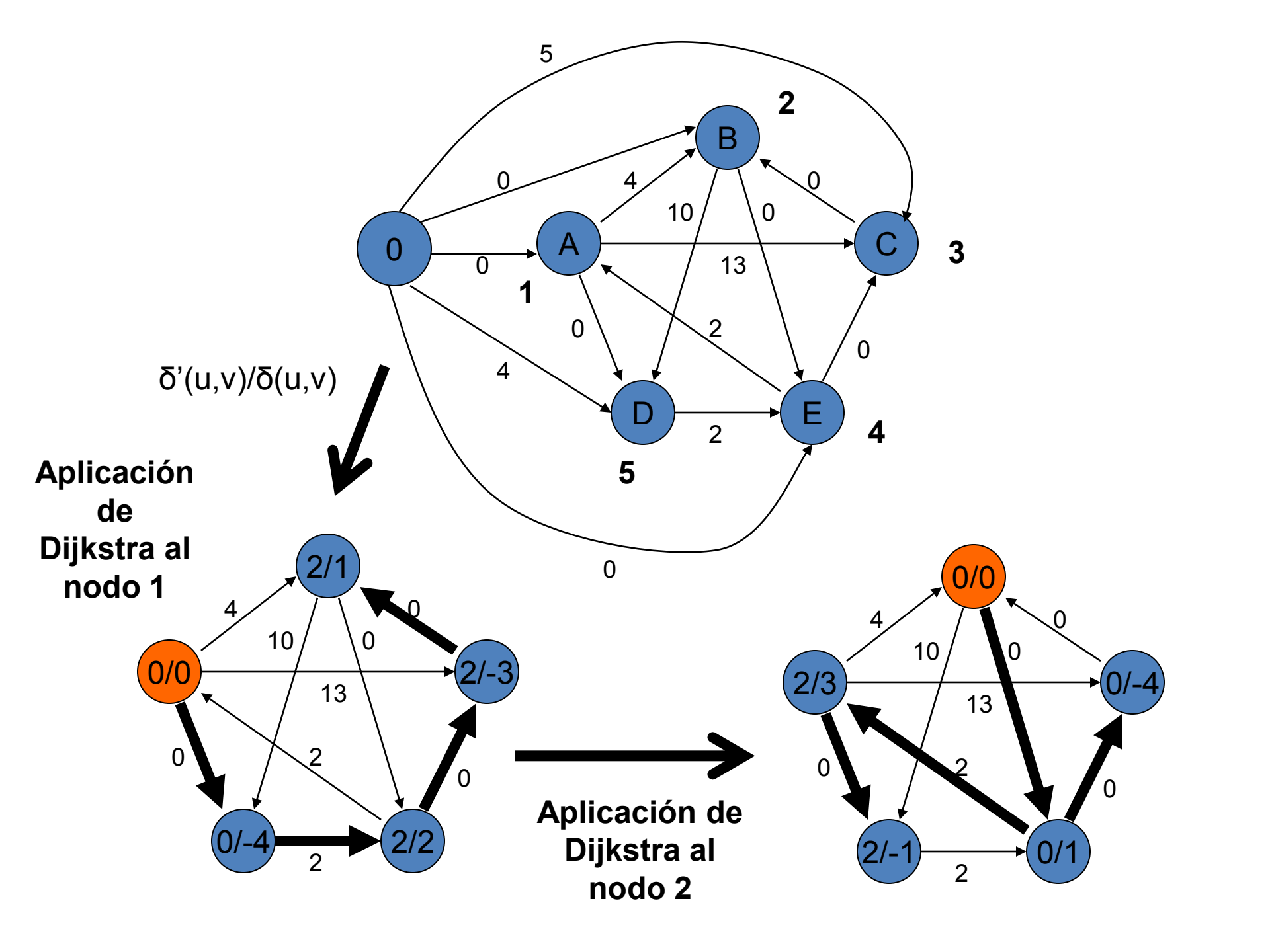
T(n) = O(N A lg N) si está implantado con montículos binarios.

Aún así es asintóticamente más rápido que el algoritmo de Floyd-Warshall

G' luego de aplicar BellmanFord que calcula los δ(s,v), de asignar h(v) y de calcular w'






5
2
B
0
4
0
10
0
A
C
3
0
0
13
1
0
2
4
0
D
E
2
4
5
0
δ'(u,v)/δ(u,v)
Aplicación de Dijkstra al nodo 1
2/1
4
0
10
0
0/0
2/-3
13
0
2
0
0/-4
2/2
2
Aplicación de Dijkstra al nodo 2
0/0
4
0
10
0
2/3
0/-4
13
0
2
0
2/-1
0/1
2

Algunas Propiedades de los caminos mínimos

- Un C+C entre dos nodos contiene otros C+Cs dentro.

- Sea G =(N, A) un digrafo etiquetado con w: A $\rightarrow \Re$ y p = ⟨n0, n1, ..., nk⟩, el C+C de n1 a nk, para cualquier i, j / 1 ≤ i ≤ j ≤ k pi j = ⟨ni, ni+1, ..., nj⟩ un subcamino de p de ni a nj ⇒ pij es el C+C entre ni y nj

.

- Suponga que p de s a v puede ser descompuesto en p' de s a u → v para algún u y p', entonces el peso del C+C de s a v

$$\delta(s, v) = \delta(s, u) + w(u, v).$$