

CAMINOS MÍNIMOS

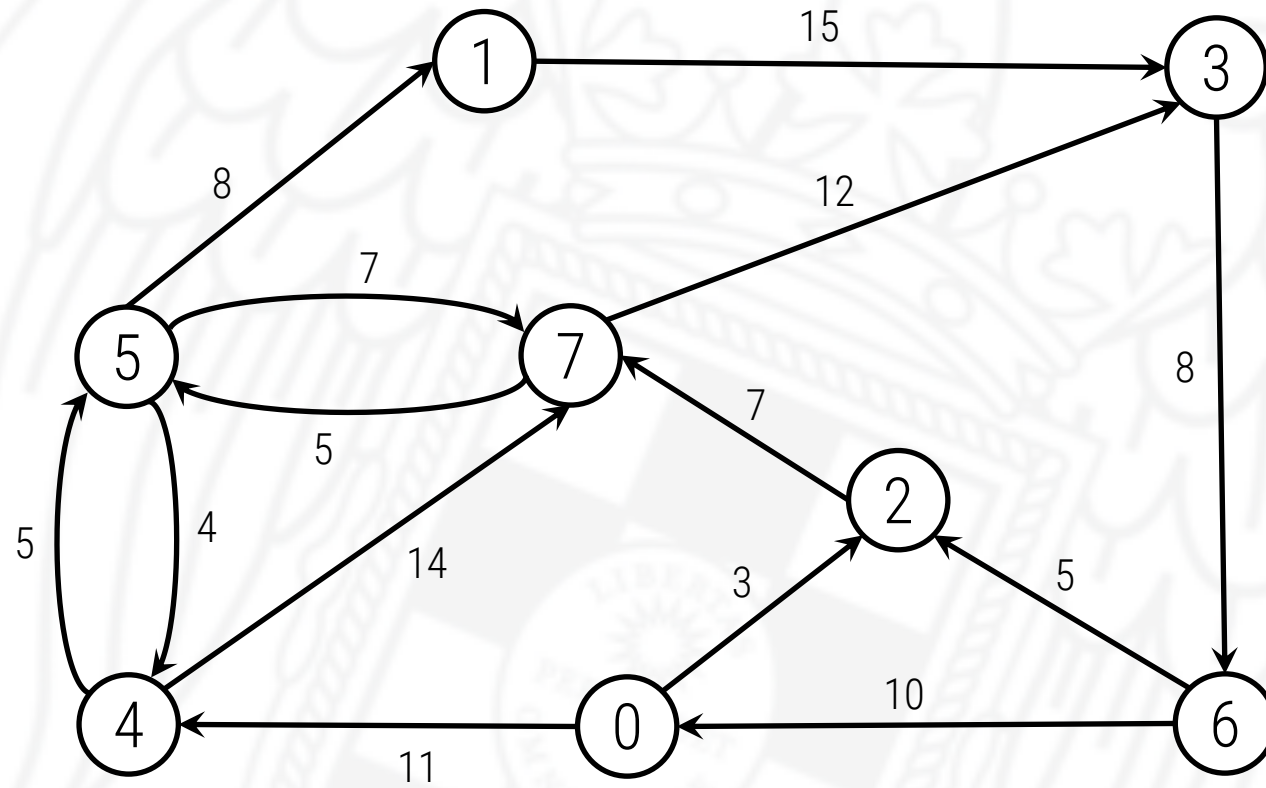
ALBERTO VERDEJO



U N I V E R S I D A D
COMPLUTENSE
M A D R I D

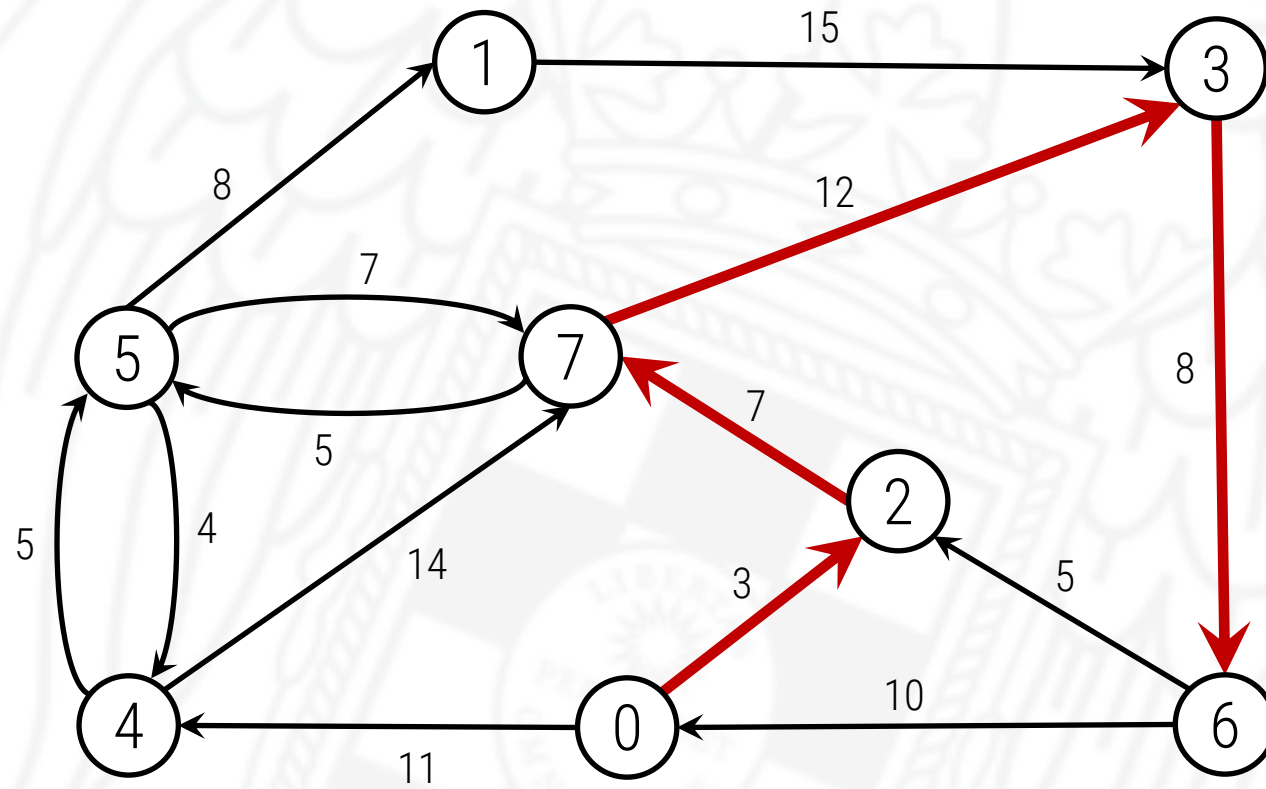
Caminos mínimos en un digrafo

- Dado un digrafo valorado, encontrar el camino mínimo de s a t .

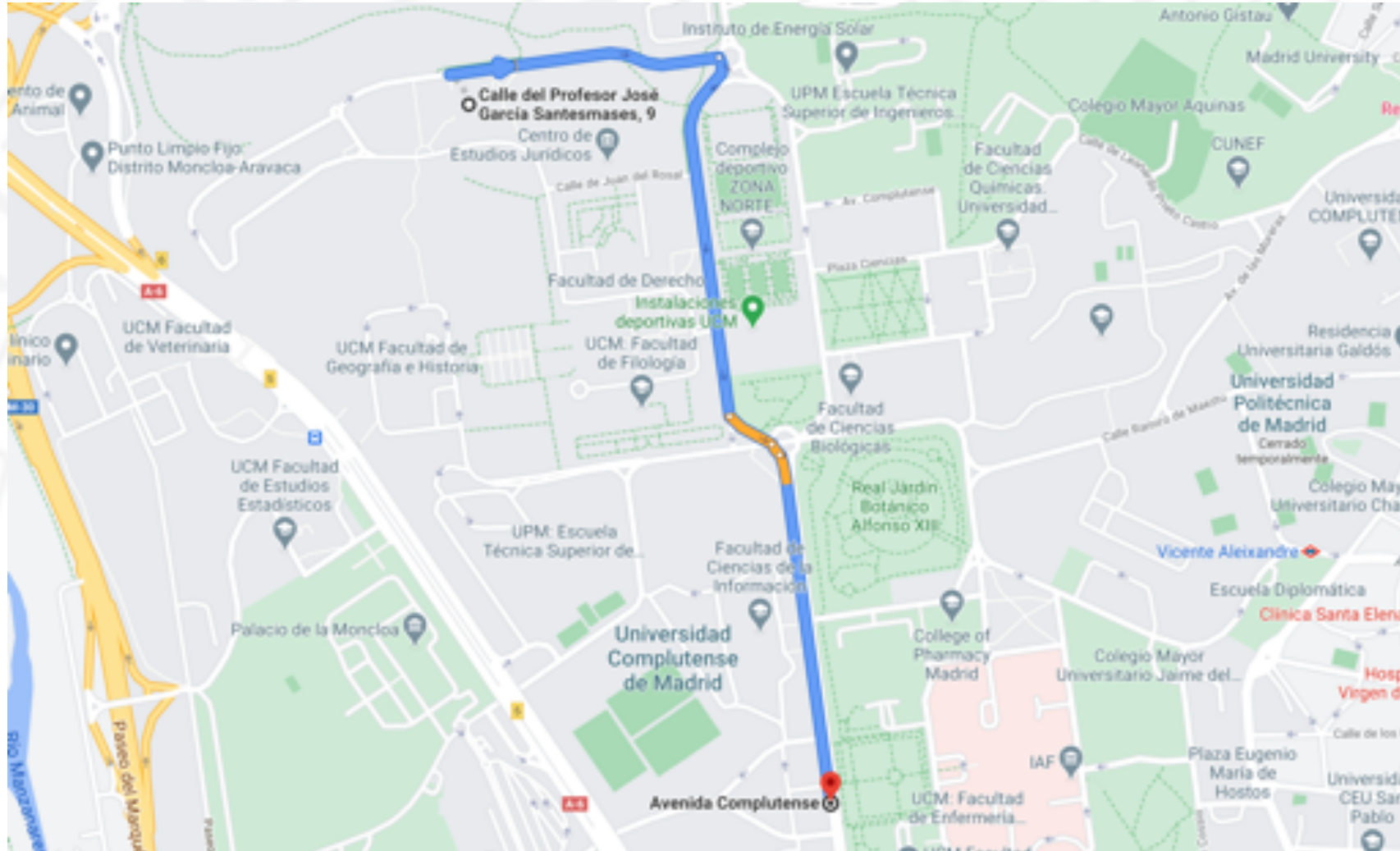


Caminos mínimos en un digrafo

- Dado un digrafo valorado, encontrar el camino mínimo de s a t .



Google Maps



Variantes del problema

¿Entre qué vértices?

- ▶ Origen único: desde un vértice s a todos los demás
- ▶ Destino único: de cualquier vértice a un vértice t
- ▶ De punto a punto: de un vértice s a otro t
- ▶ Entre cualquier par de vértices

Restricciones sobre los pesos

- ▶ Pesos no negativos
- ▶ Pesos euclídeos
- ▶ Pesos arbitrarios

Presencia de ciclos

- ▶ Con ciclos dirigidos
- ▶ Sin ciclos dirigidos
- ▶ Sin ciclos de coste negativo

Caminos mínimos desde un origen único

- Caminos mínimos desde un vértice **origen** a todos los demás.

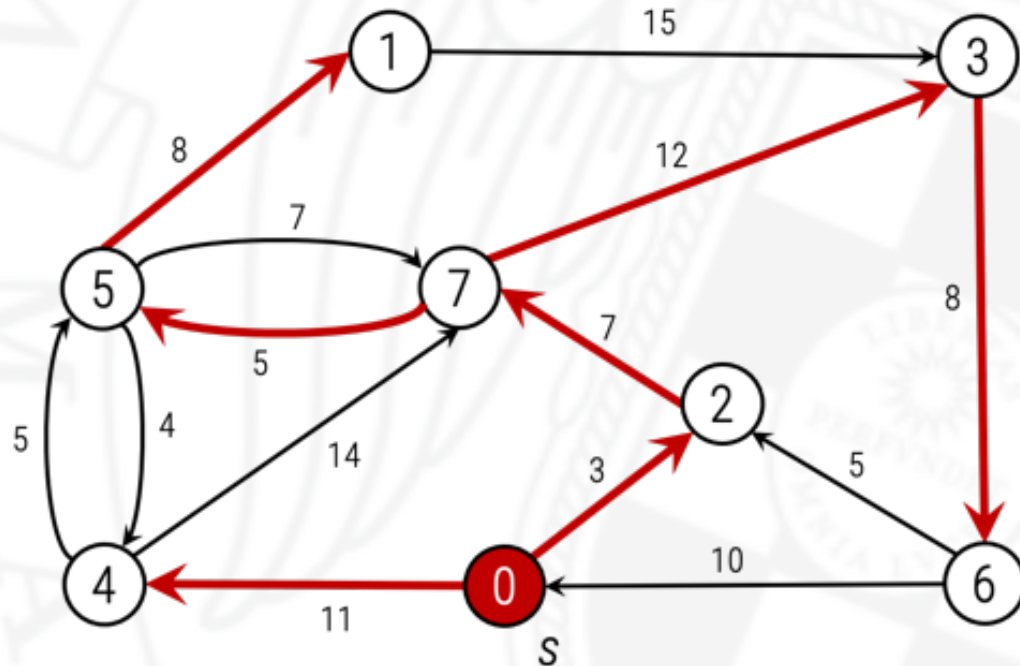
```
template <typename Valor>
class CaminosMinimos {
public:
    CaminosMinimos(DigrafoValorado<Valor> const& g, int origen);

    Valor distancia(int v) const;

    Camino<Valor> camino(int v) const;
};
```

Caminos mínimos desde un origen único

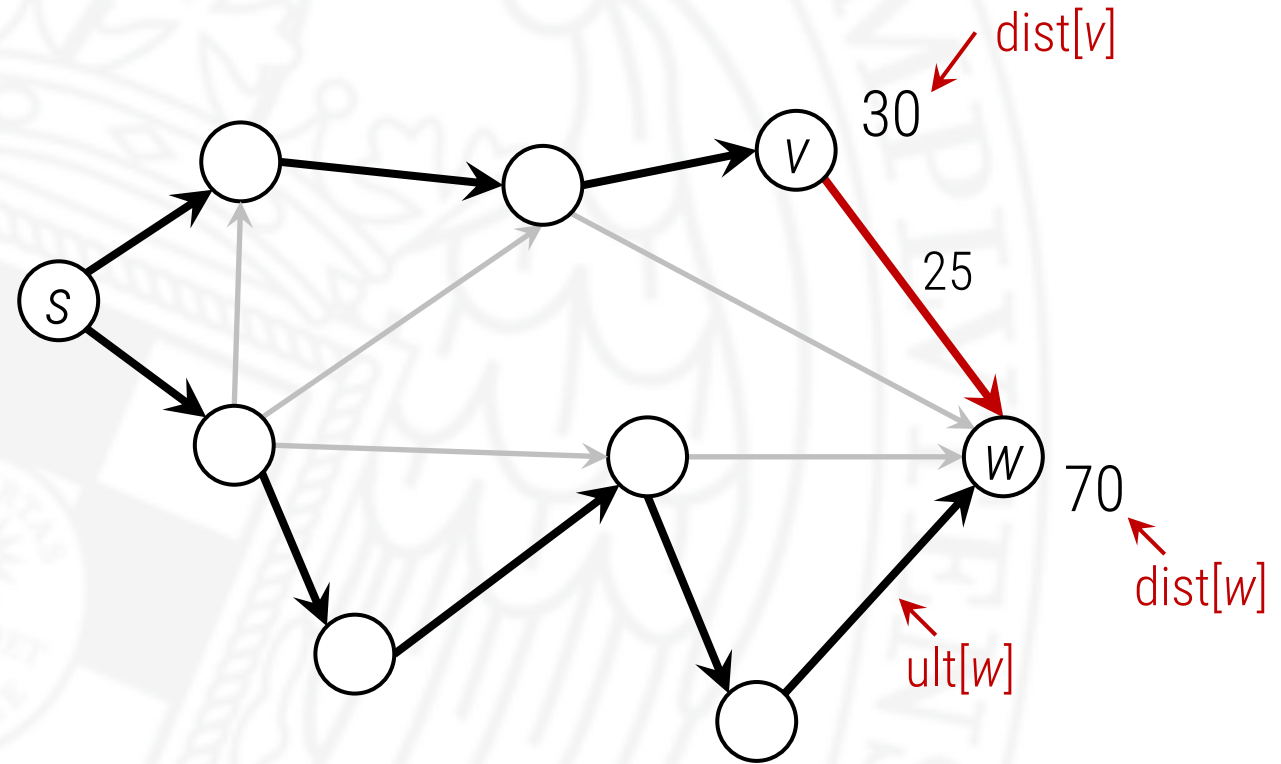
- ▶ Los caminos mínimos forman un **árbol de caminos mínimos**.
- ▶ Se pueden representar todos los caminos con dos vectores:
 - `dist[v]` es la longitud del camino más corto desde el origen a v
 - `ult[v]` es la última arista del camino más corto desde el origen a v



v	<code>dist[]</code>	<code>ult[]</code>
0	0	-
1	23	5 → 1
2	3	0 → 2
3	22	7 → 3
4	11	0 → 4
5	15	7 → 5
6	30	3 → 6
7	10	2 → 7

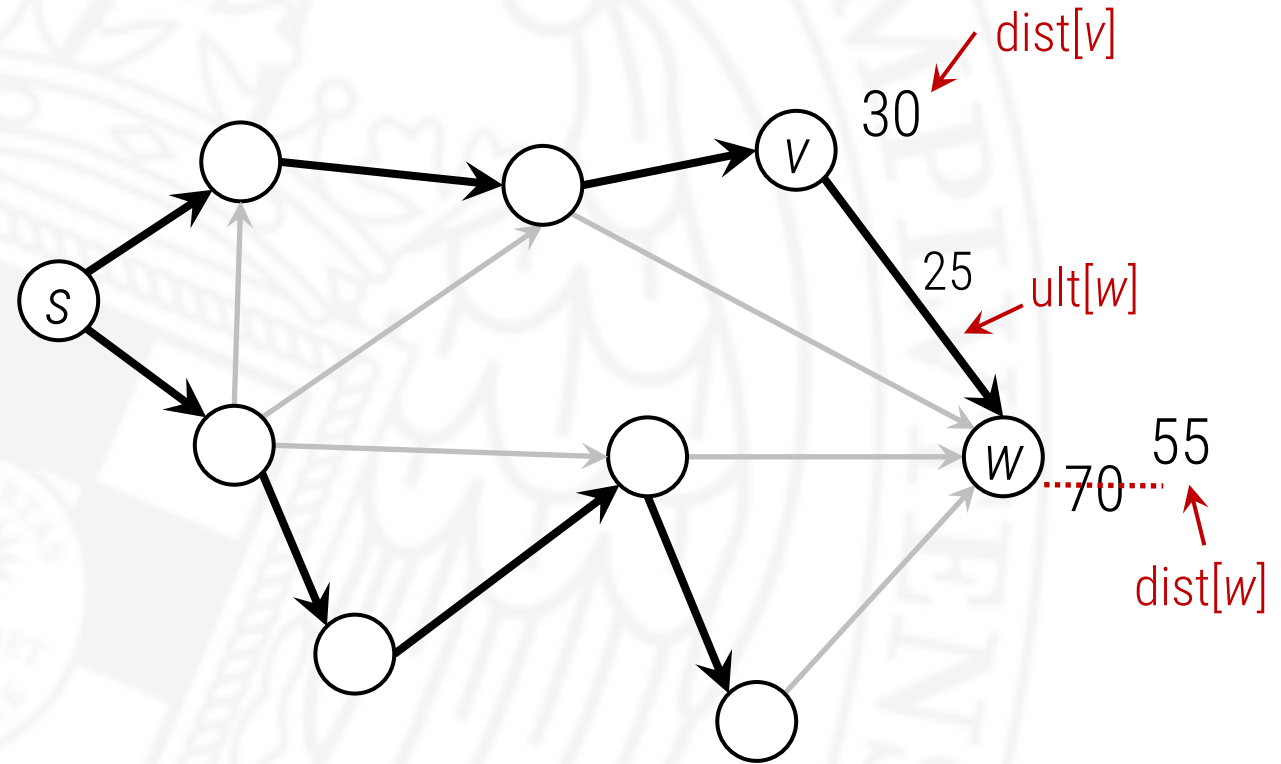
Relajación de aristas

- ▶ $\text{dist}[v]$ es la longitud del camino más corto *conocido* de s a v
- ▶ $\text{dist}[w]$ es la longitud del camino más corto *conocido* de s a w
- ▶ $\text{ult}[w]$ es la última arista del camino más corto *conocido* de s a w
- ▶ Si la arista $v \rightarrow w$ proporciona un camino más corto hasta w a través de v , se actualizan tanto $\text{dist}[w]$ como $\text{ult}[w]$



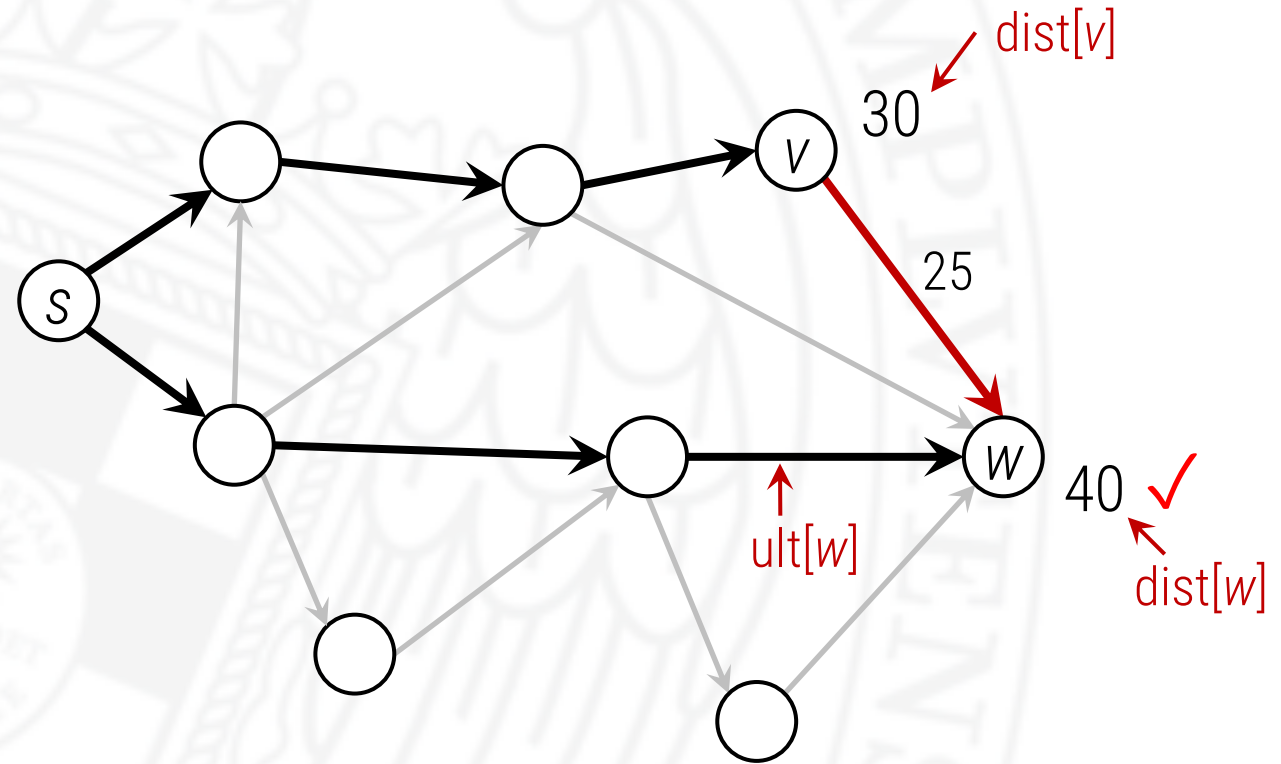
Relajación de aristas

- ▶ $\text{dist}[v]$ es la longitud del camino más corto *conocido* de s a v
- ▶ $\text{dist}[w]$ es la longitud del camino más corto *conocido* de s a w
- ▶ $\text{ult}[w]$ es la última arista del camino más corto *conocido* de s a w
- ▶ Si la arista $v \rightarrow w$ proporciona un camino más corto hasta w a través de v , se actualizan tanto $\text{dist}[w]$ como $\text{ult}[w]$



Relajación de aristas

- ▶ $\text{dist}[v]$ es la longitud del camino más corto *conocido* de s a v
- ▶ $\text{dist}[w]$ es la longitud del camino más corto *conocido* de s a w
- ▶ $\text{ult}[w]$ es la última arista del camino más corto *conocido* de s a w
- ▶ Si la arista $v \rightarrow w$ proporciona un camino más corto hasta w a través de v , se actualizan tanto $\text{dist}[w]$ como $\text{ult}[w]$



Relajación de aristas

- ▶ `dist[v]` es la longitud del camino más corto *conocido* de `s` a `v`
- ▶ `dist[w]` es la longitud del camino más corto *conocido* de `s` a `w`
- ▶ `ult[w]` es la última arista del camino más corto *conocido* de `s` a `w`

- ▶ Si la arista $v \rightarrow w$ proporciona un camino más corto hasta `w` a través de `v`, se actualizan tanto `dist[w]` como `ult[w]`

```
void relajar(AristaDirigida<Valor> a) {  
    int v = a.desde(), w = a.hasta();  
    if (dist[w] > dist[v] + a.valor()) {  
        dist[w] = dist[v] + a.valor();  
        ult[w] = a;  
    }  
}
```

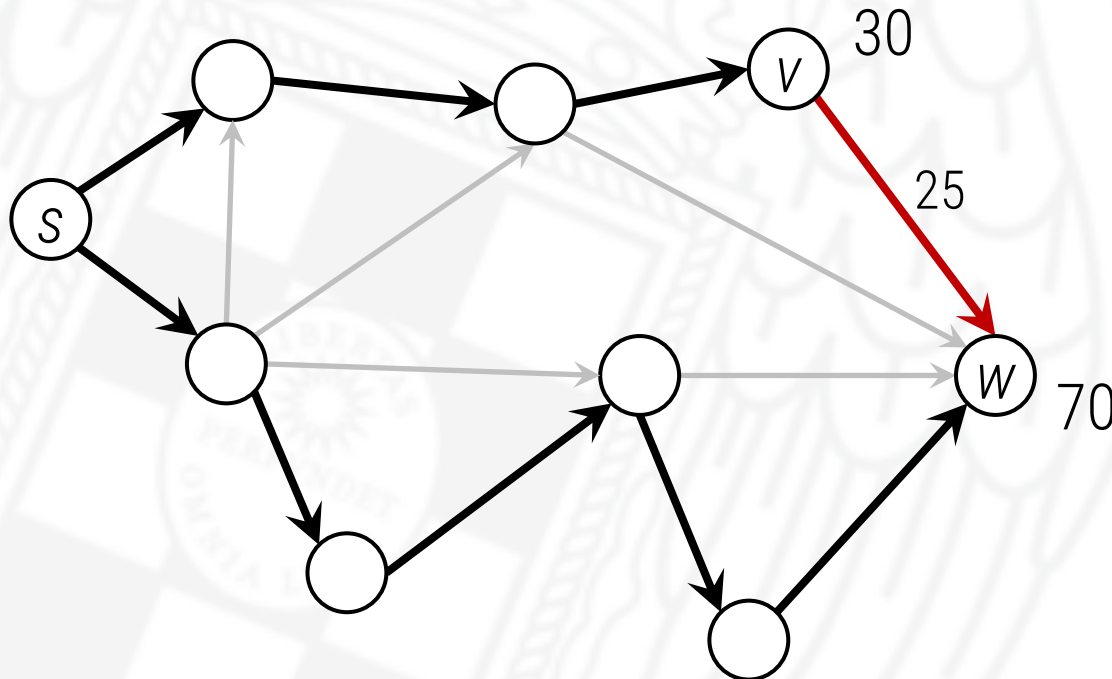
Condiciones de optimalidad

- ▶ Sea G un digrafo valorado. `dist[]` contiene las distancias de los caminos más cortos desde s al resto de vértices **si y solo si**
 - ▶ `dist[s] = 0`
 - ▶ Para todo v , `dist[v]` es la longitud de algún camino de s a v
 - ▶ Para toda arista $v \xrightarrow{a} w$, `dist[w] ≤ dist[v] + a.valor()`

Condiciones de optimalidad. Demostración

⇒ (necesarias)

- ▶ Supongamos $\text{dist}[w] > \text{dist}[v] + a.\text{valor}()$ para alguna arista $v \xrightarrow{a} w$
- ▶ Entonces a nos daría un camino de s a w (a través de v) de longitud menor que $\text{dist}[w]$



Condiciones de optimalidad. Demostración

⇐ (suficientes)

► Supongamos que $s \xrightarrow{a_1} v_1 \xrightarrow{a_2} v_2 \rightarrow \dots \rightarrow v_{k-1} \xrightarrow{a_k} w$ es un camino más corto de s a w

► Entonces, $\text{dist}[v_1] \leq \text{dist}[s] + a_1.\text{valor}()$

$$\text{dist}[v_2] \leq \text{dist}[v_1] + a_2.\text{valor}()$$

...

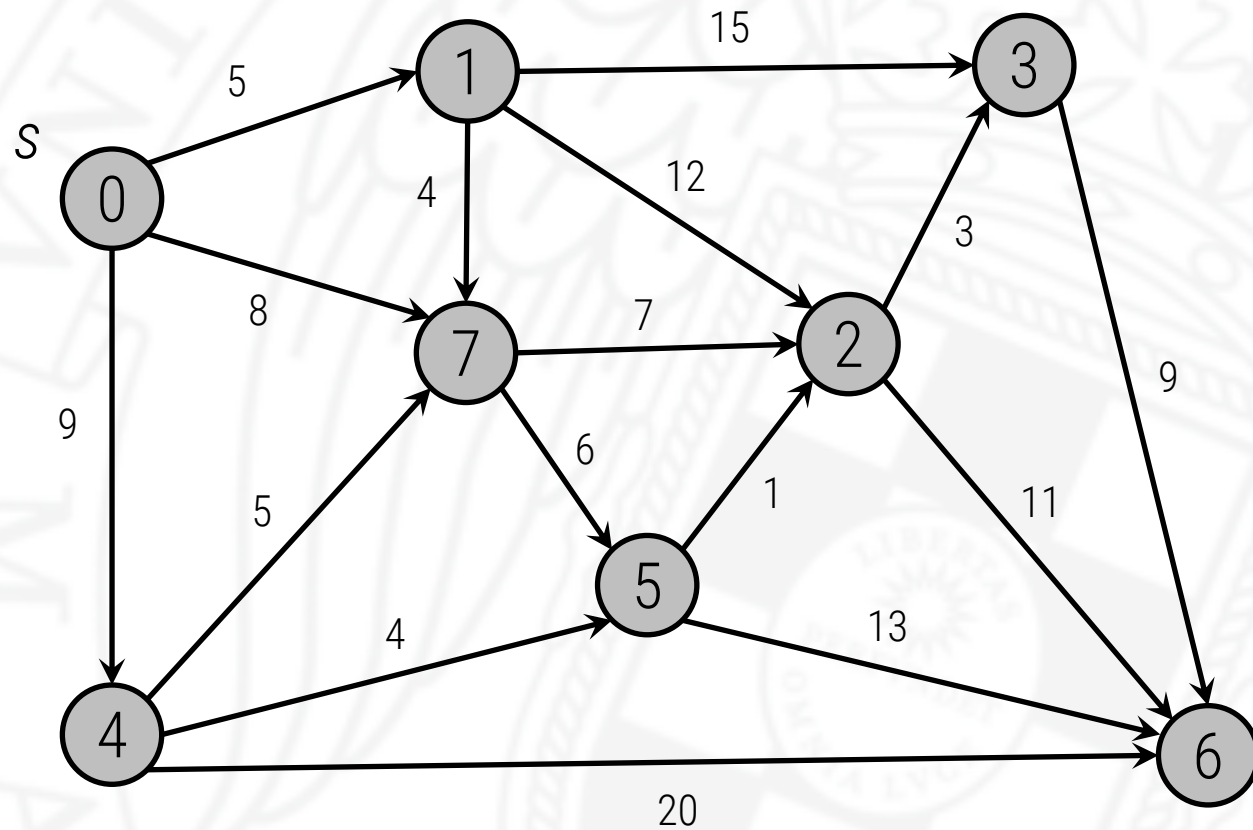
$$\text{dist}[w] \leq \text{dist}[v_{k-1}] + a_k.\text{valor}()$$

$$\text{dist}[w] \leq \underbrace{a_1.\text{valor}() + a_2.\text{valor}() + \dots + a_k.\text{valor}()}_{}_{}$$

longitud del camino más corto de s a w

Algoritmo de Dijkstra

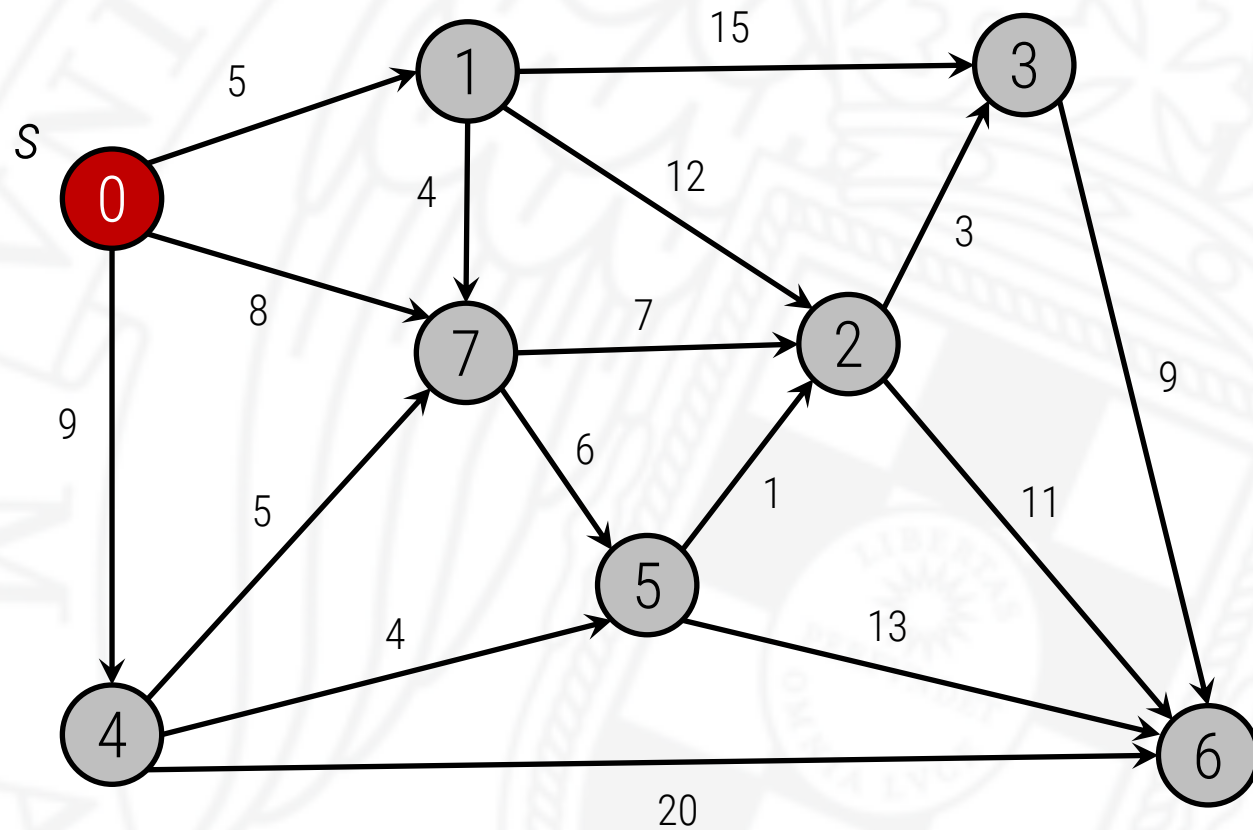
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0		
1		
2		
3		
4		
5		
6		
7		

Algoritmo de Dijkstra

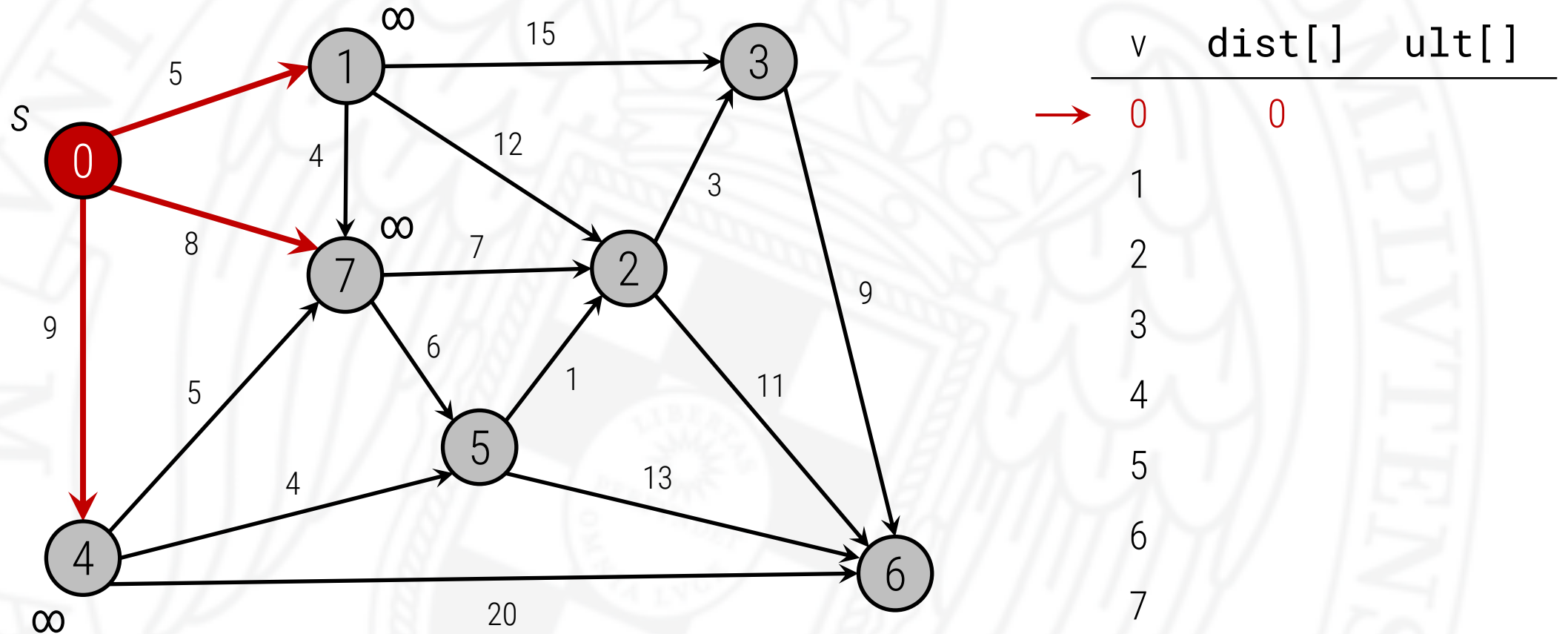
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1		
2		
3		
4		
5		
6		
7		

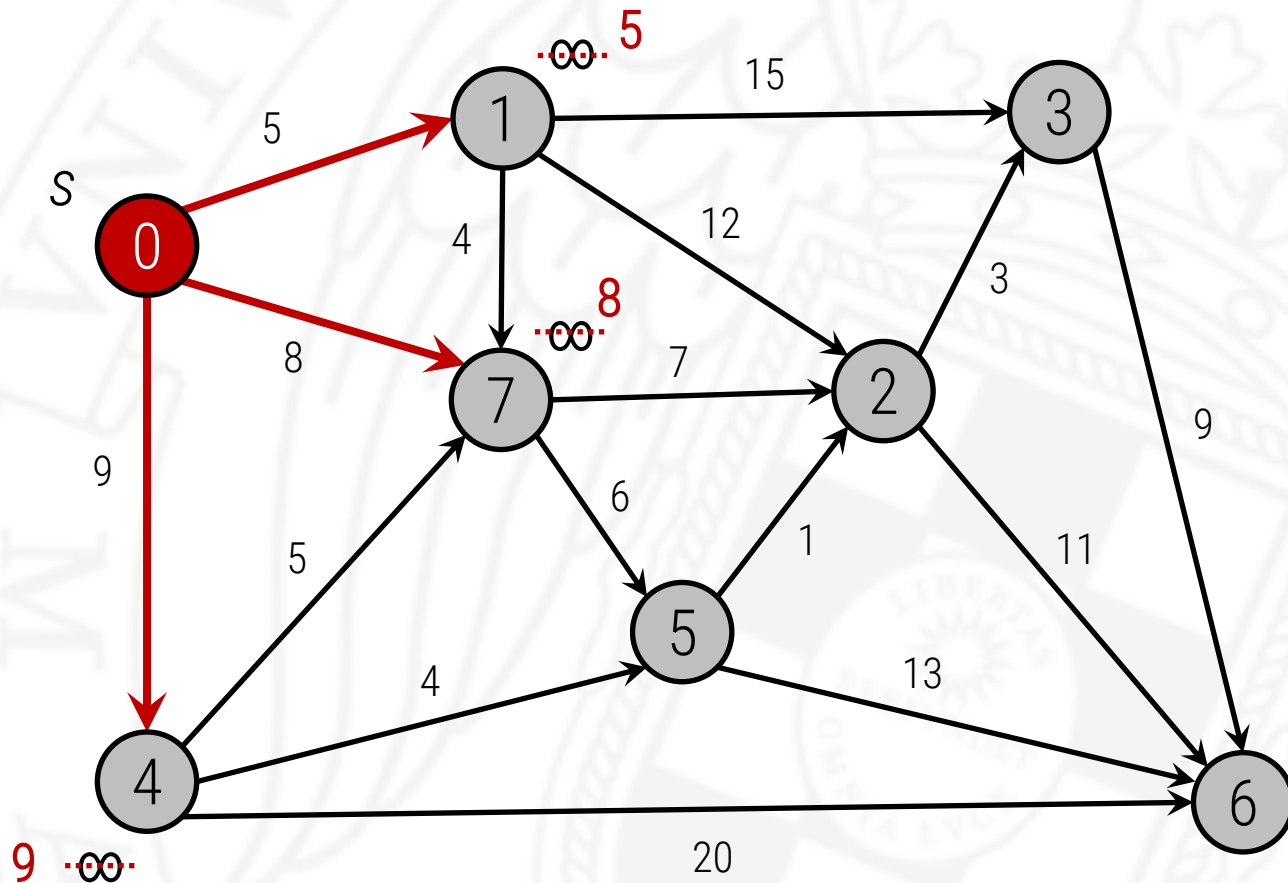
Algoritmo de Dijkstra

- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



Algoritmo de Dijkstra

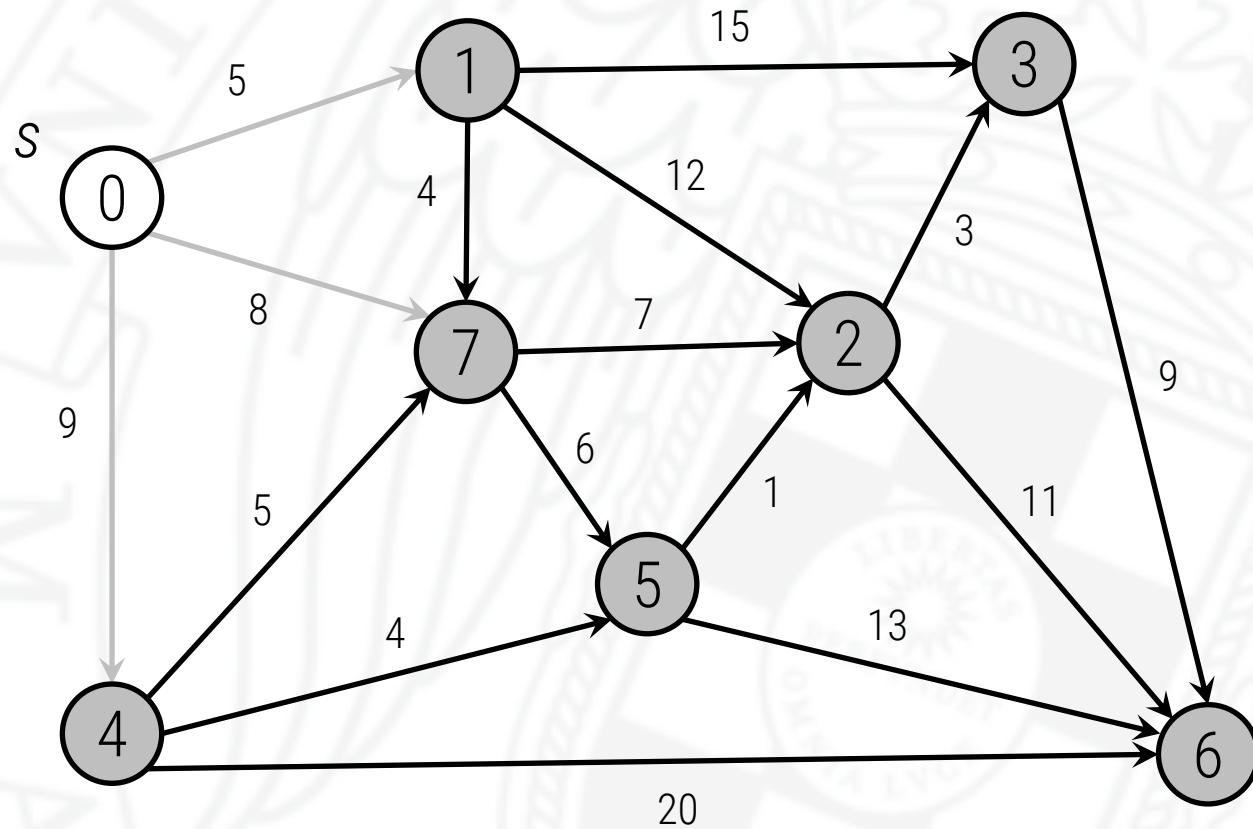
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2		
3		
4	9	0 → 4
5		
6		
7	8	0 → 7

Algoritmo de Dijkstra

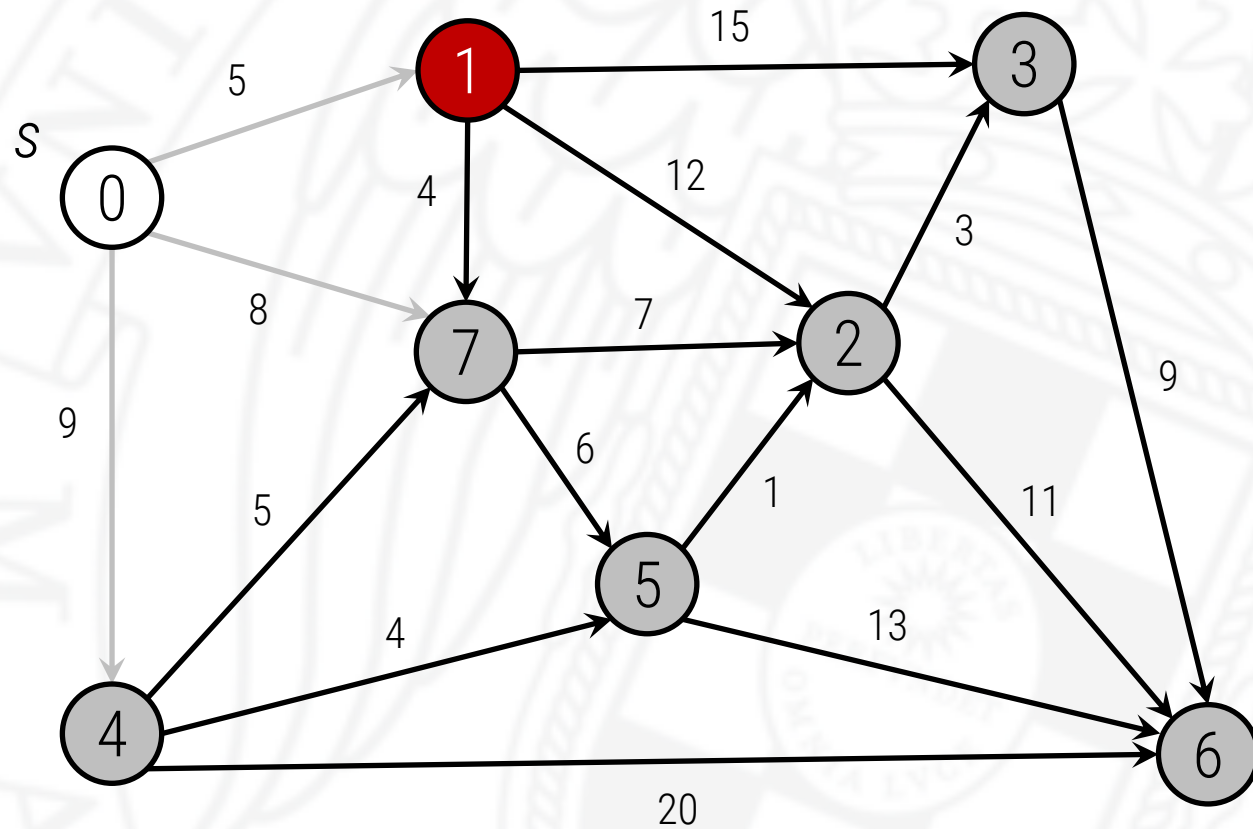
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2		
3		
4	9	0 → 4
5		
6		
7	8	0 → 7

Algoritmo de Dijkstra

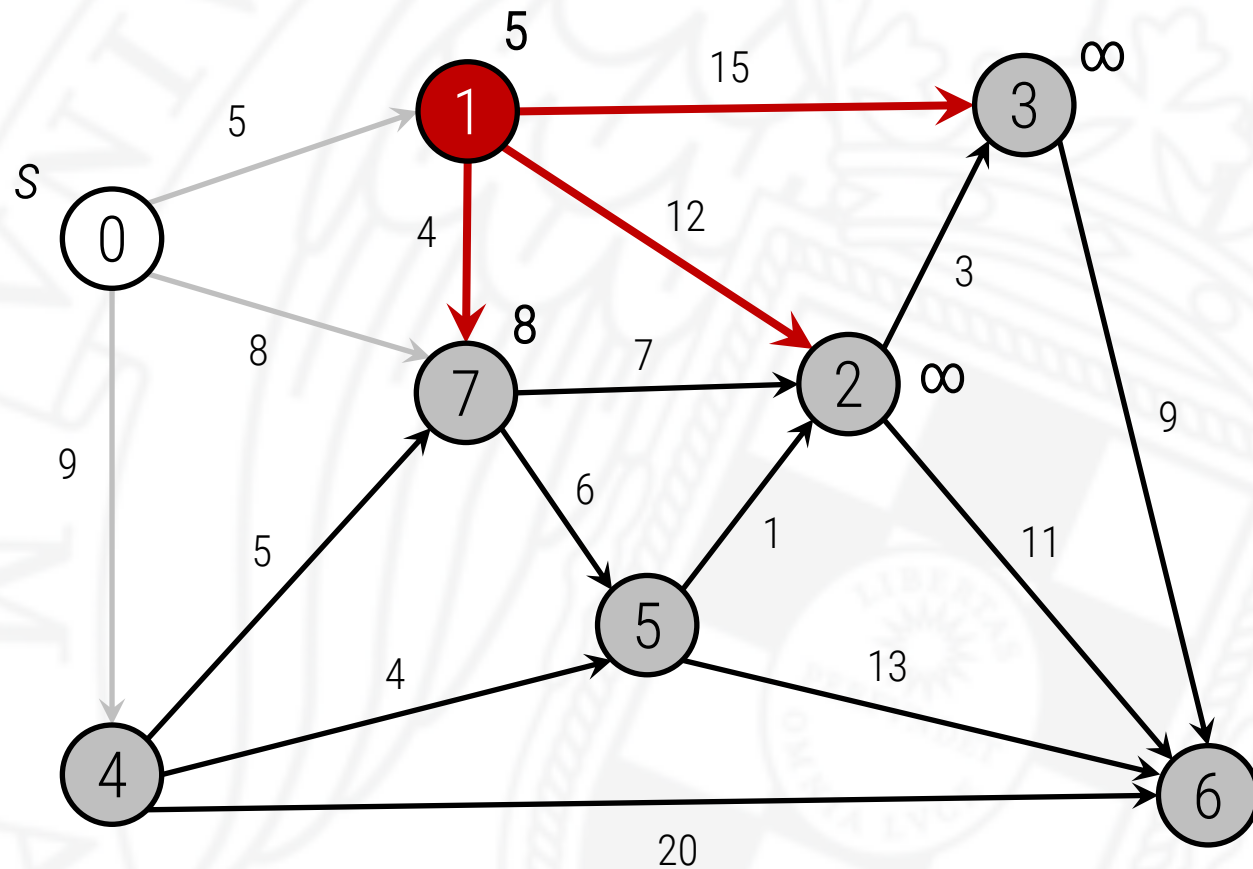
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
→ 1	5	0 → 1
2		
3		
4	9	0 → 4
5		
6		
7	8	0 → 7

Algoritmo de Dijkstra

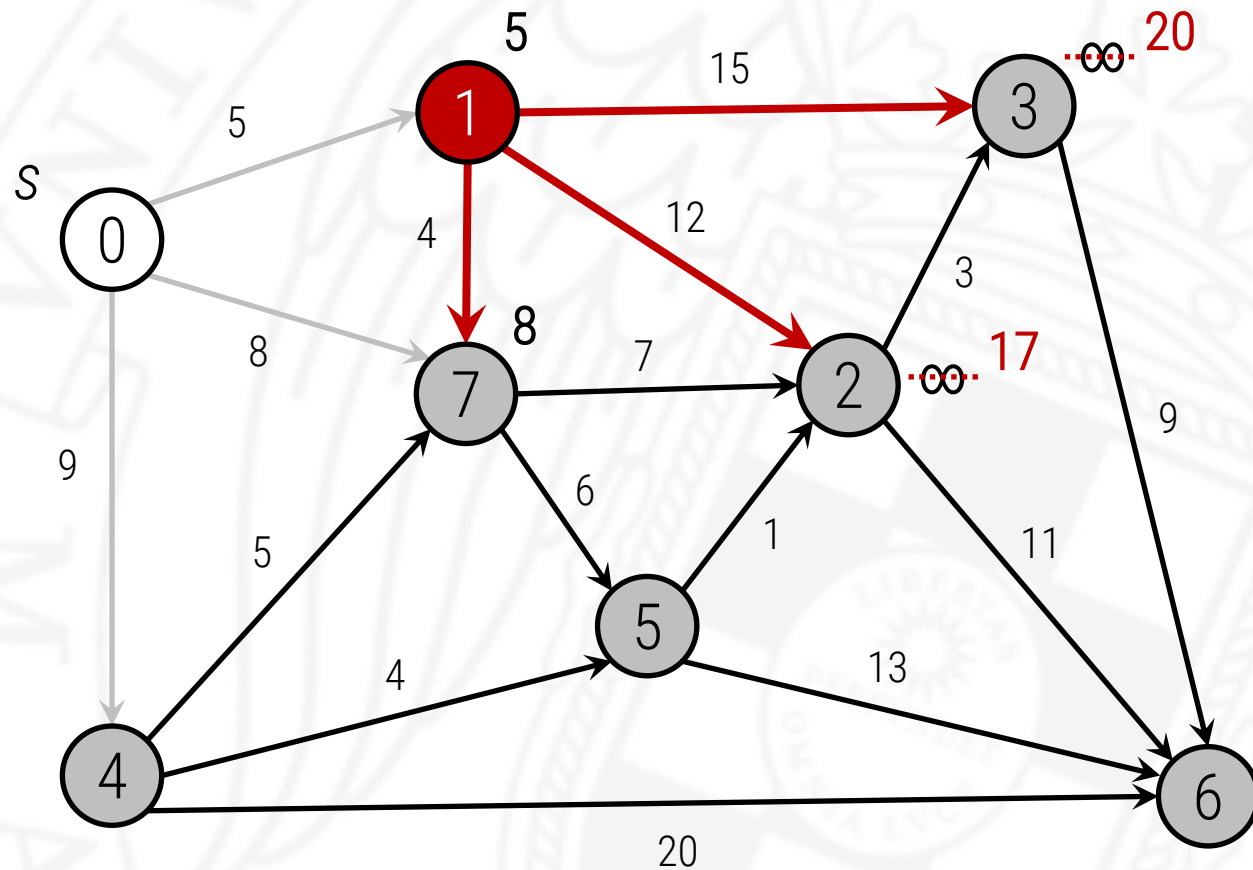
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	17	1 → 2
3	20	1 → 3
4	9	0 → 4
5		
6		
7	8	0 → 7

Algoritmo de Dijkstra

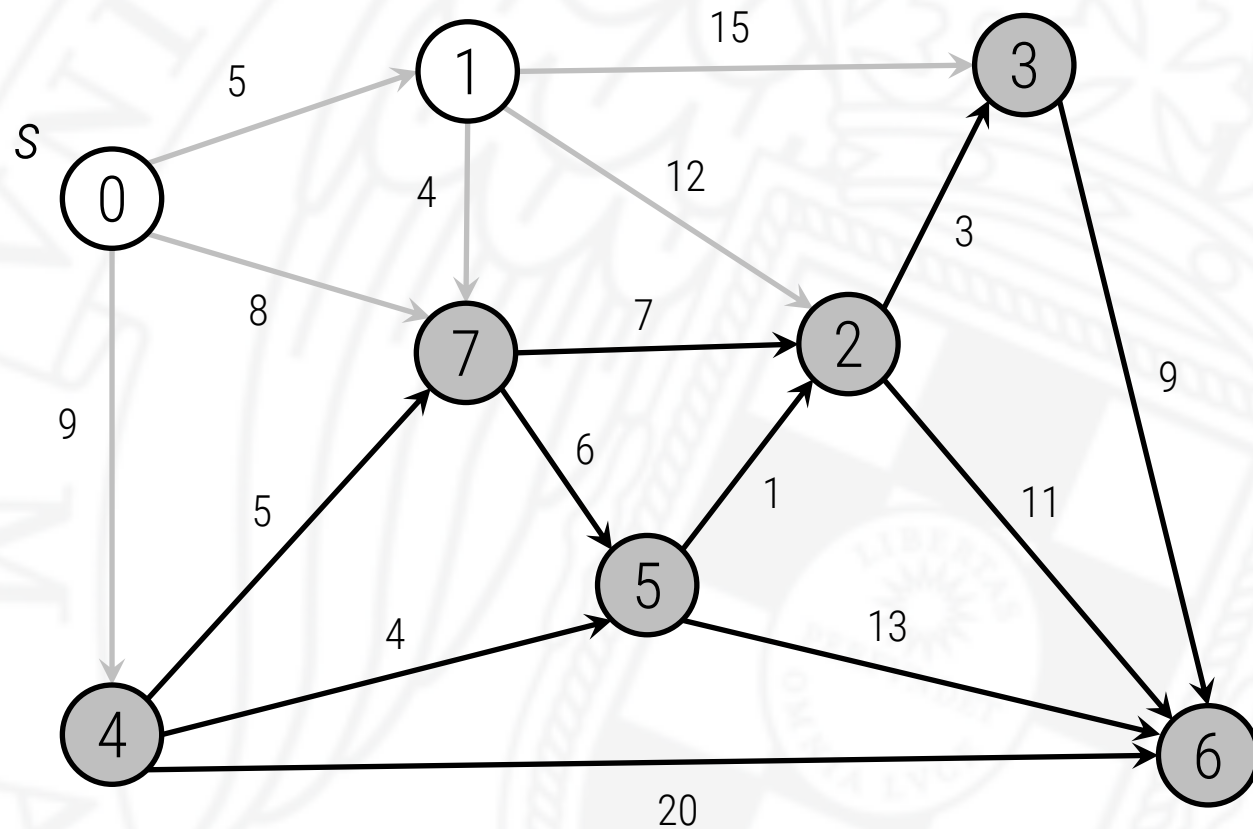
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	17	1 → 2
3	20	1 → 3
4	9	0 → 4
5		
6		
7	8	0 → 7

Algoritmo de Dijkstra

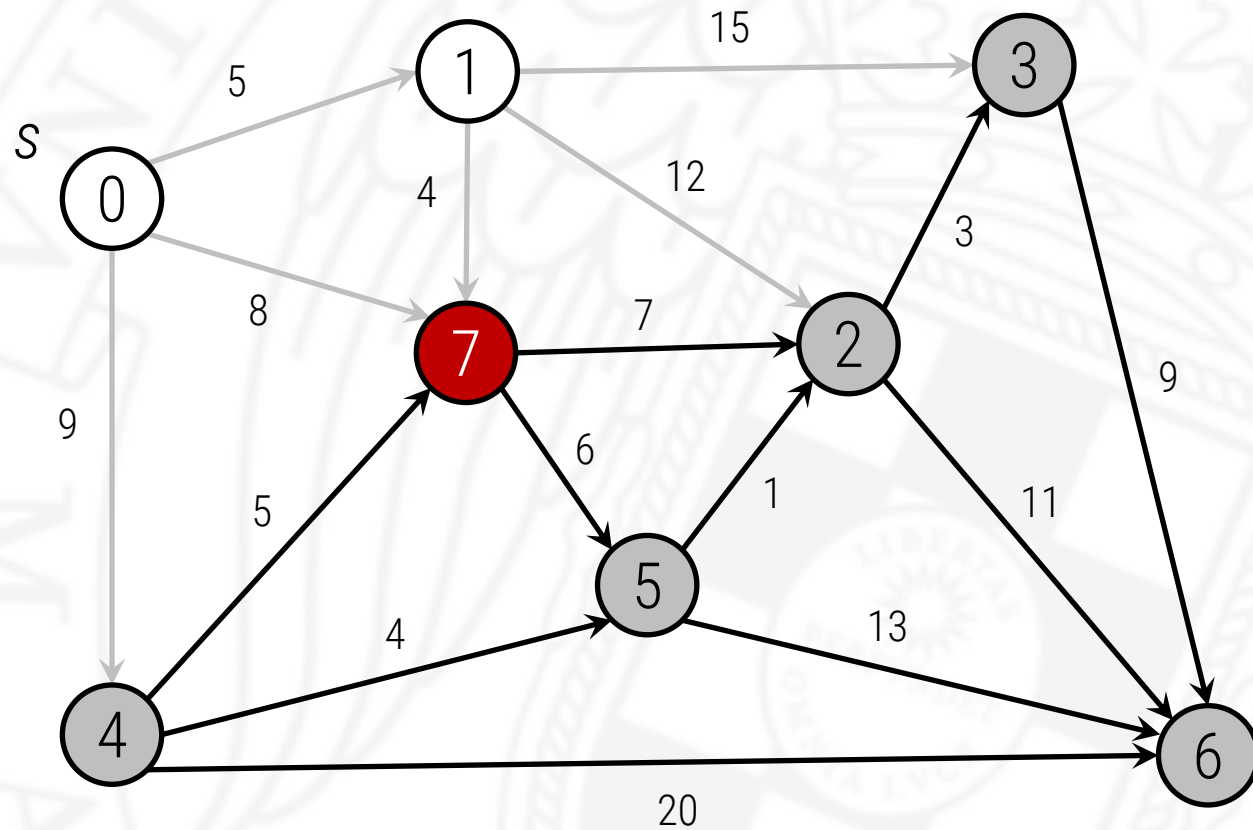
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	17	1 → 2
3	20	1 → 3
4	9	0 → 4
5		
6		
7	8	0 → 7

Algoritmo de Dijkstra

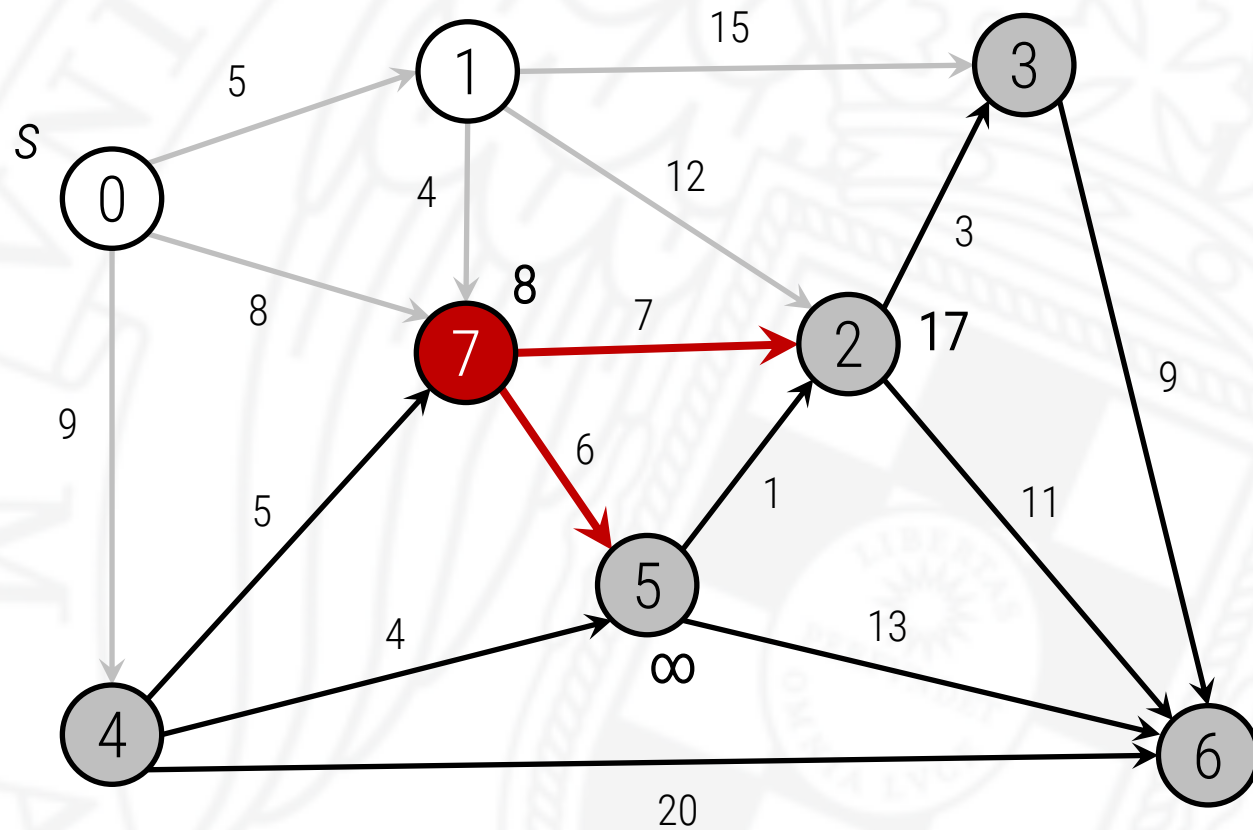
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	17	1 → 2
3	20	1 → 3
4	9	0 → 4
5		
6		
7	8	0 → 7

Algoritmo de Dijkstra

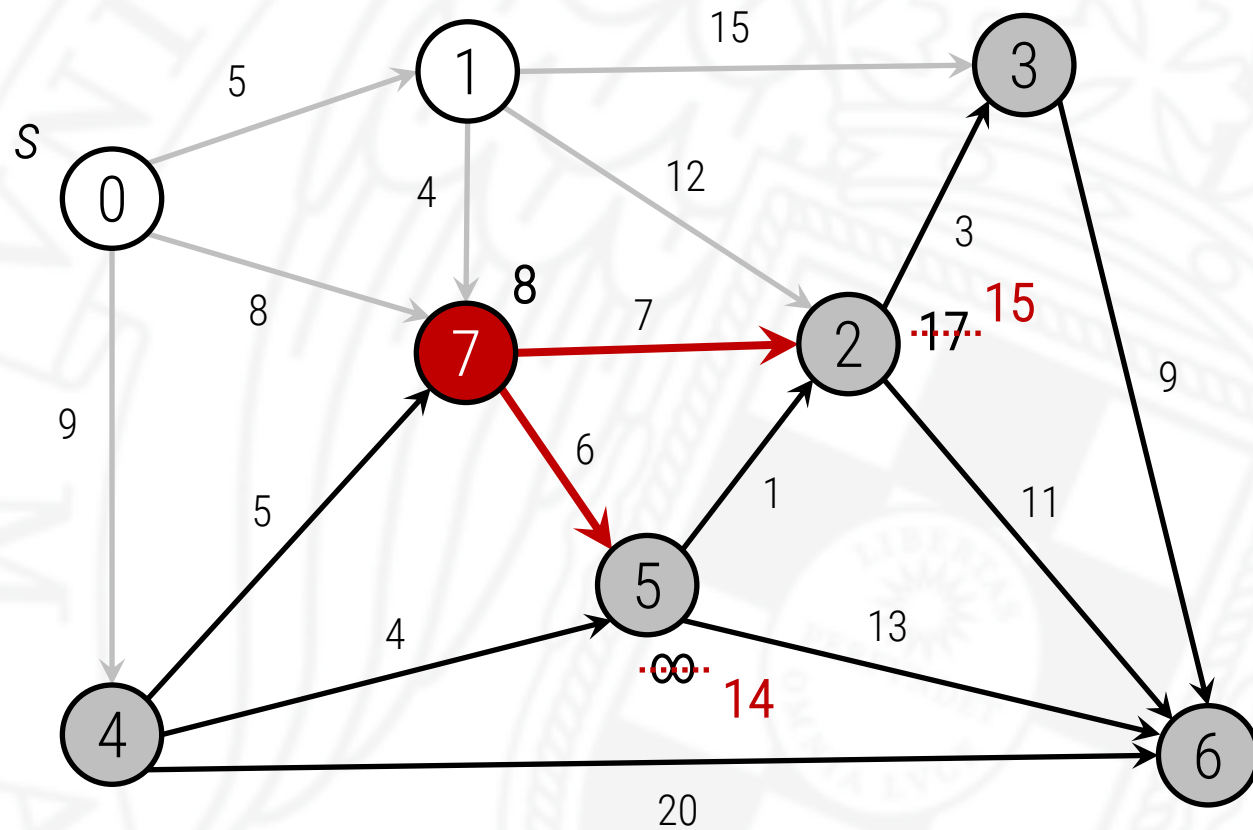
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	17	1 → 2
3	20	1 → 3
4	9	0 → 4
5		
6		
7	8	0 → 7

Algoritmo de Dijkstra

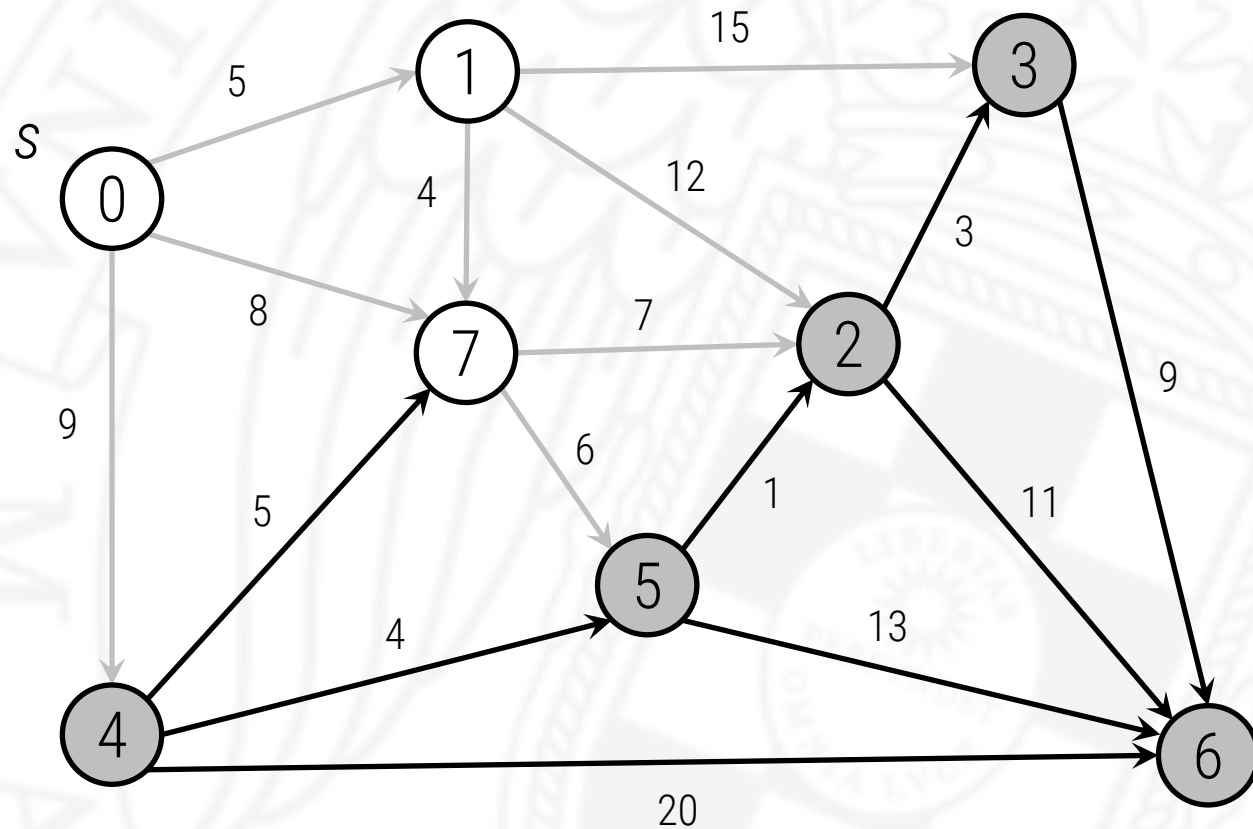
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	15	7 → 2
3	20	1 → 3
4	9	0 → 4
5	14	7 → 5
6		
7	8	0 → 7

Algoritmo de Dijkstra

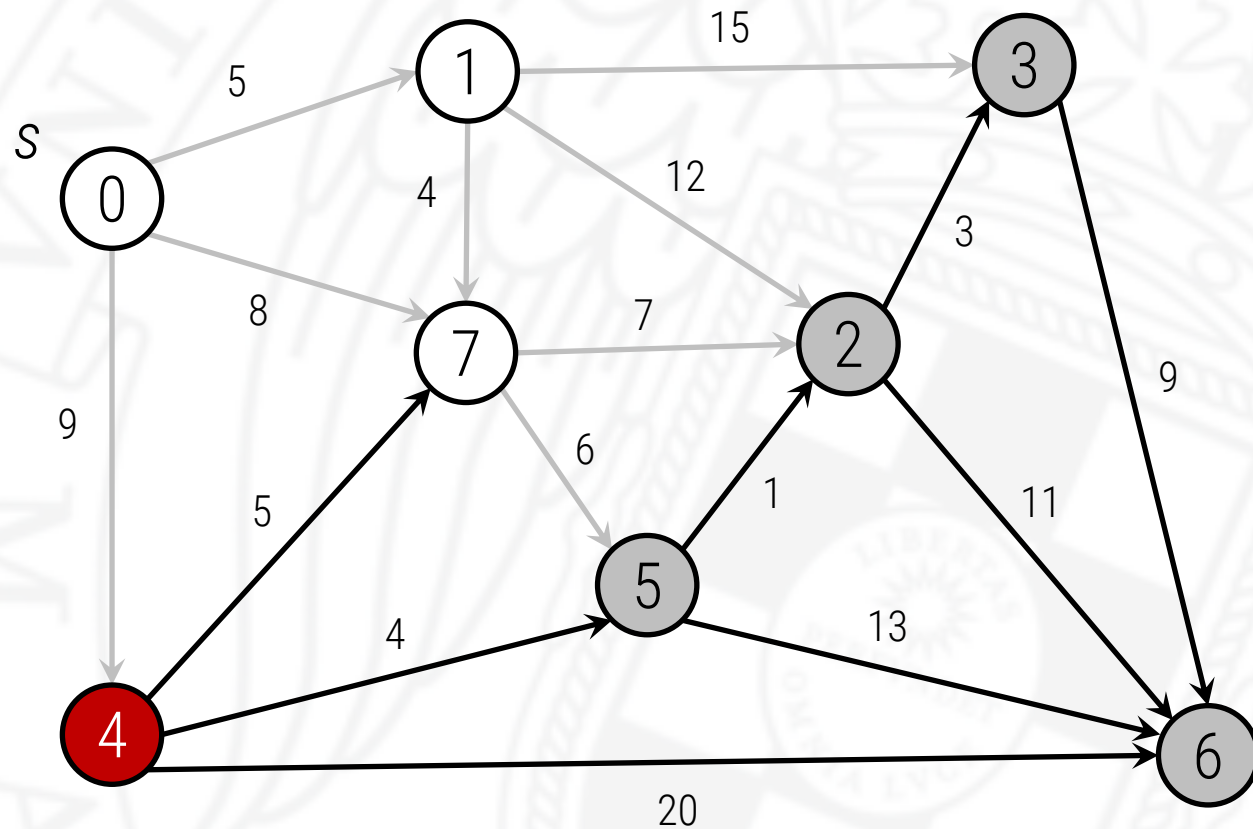
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	15	7 → 2
3	20	1 → 3
4	9	0 → 4
5	14	7 → 5
6		
7	8	0 → 7

Algoritmo de Dijkstra

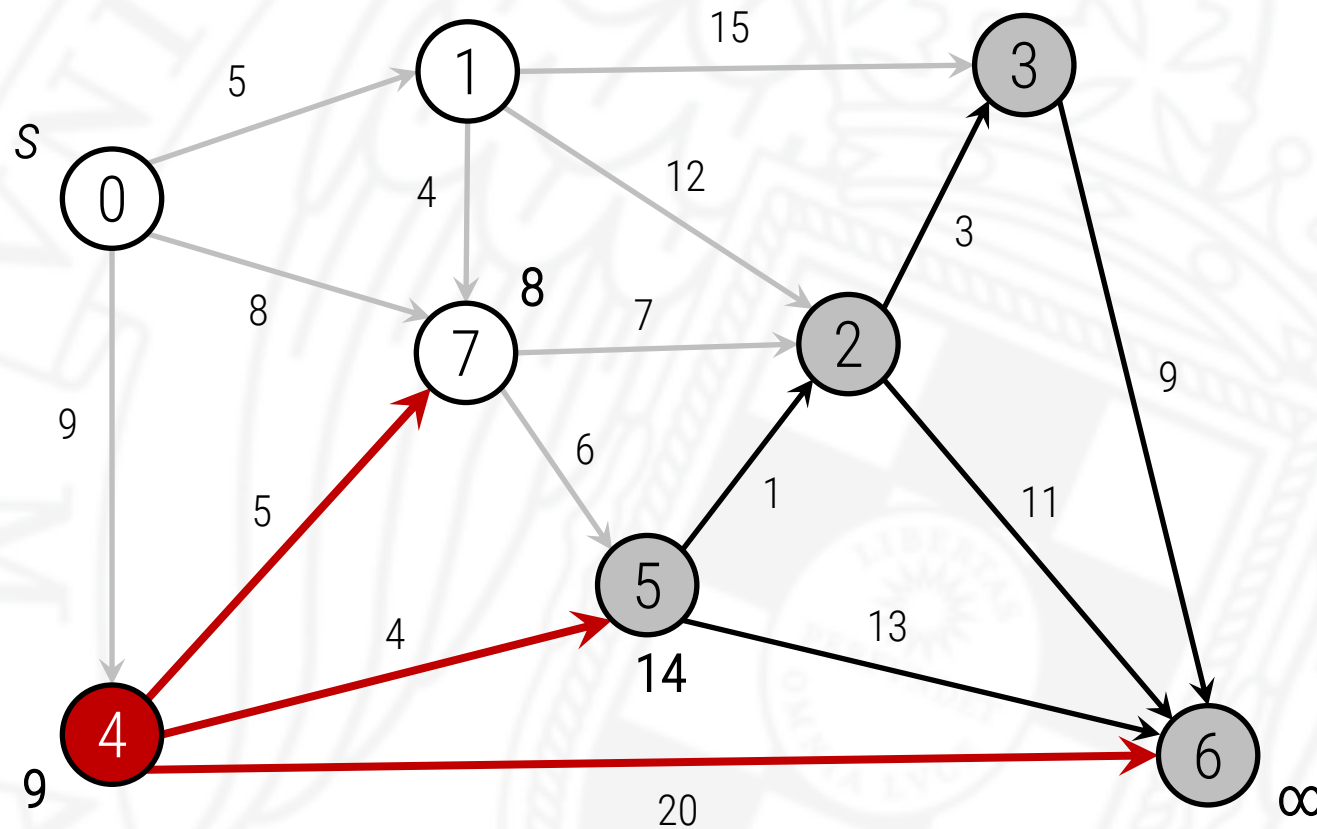
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	15	7 → 2
3	20	1 → 3
→ 4	9	0 → 4
5	14	7 → 5
6		
7	8	0 → 7

Algoritmo de Dijkstra

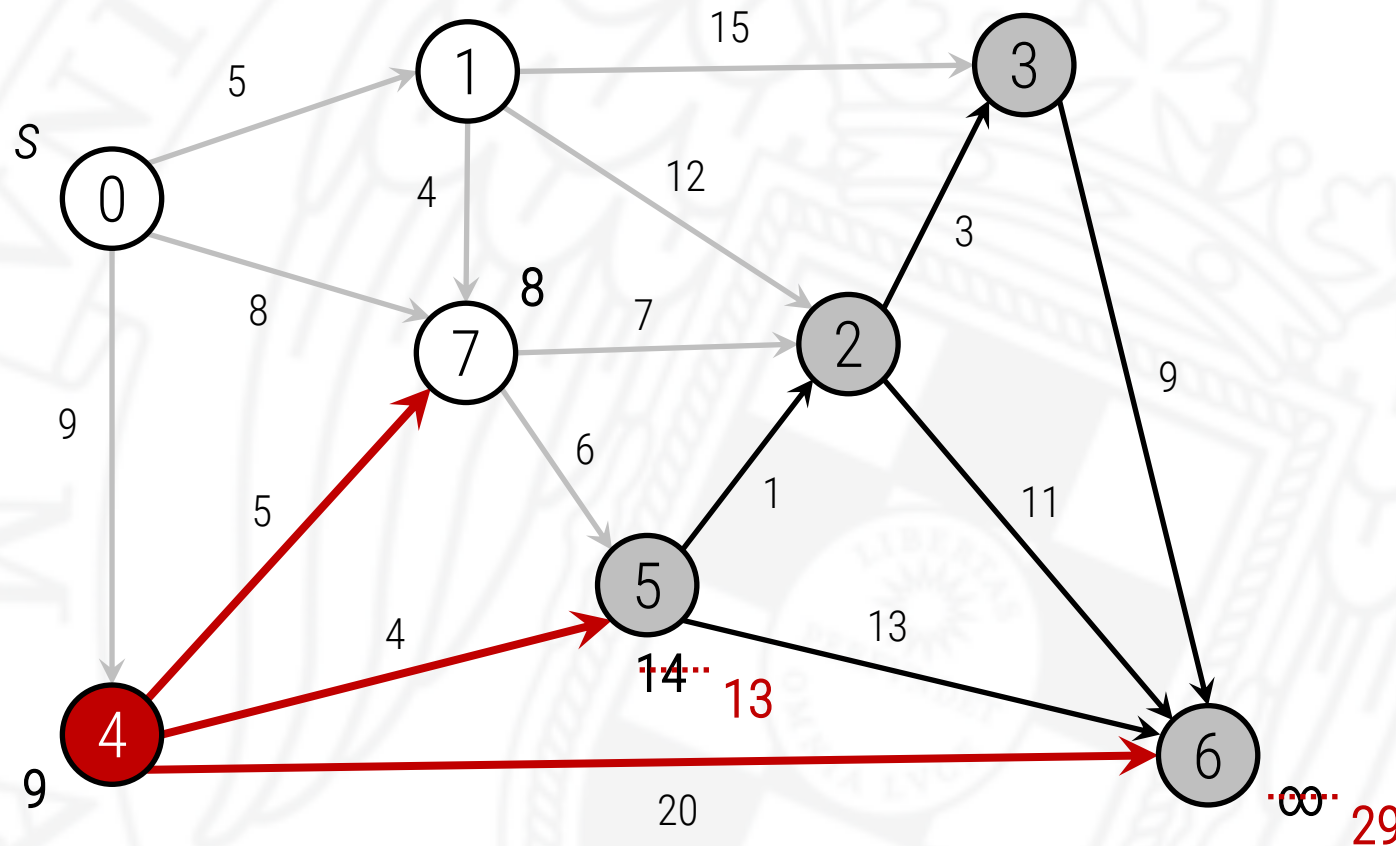
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	15	7 → 2
3	20	1 → 3
→ 4	9	0 → 4
5	14	7 → 5
6		
7	8	0 → 7

Algoritmo de Dijkstra

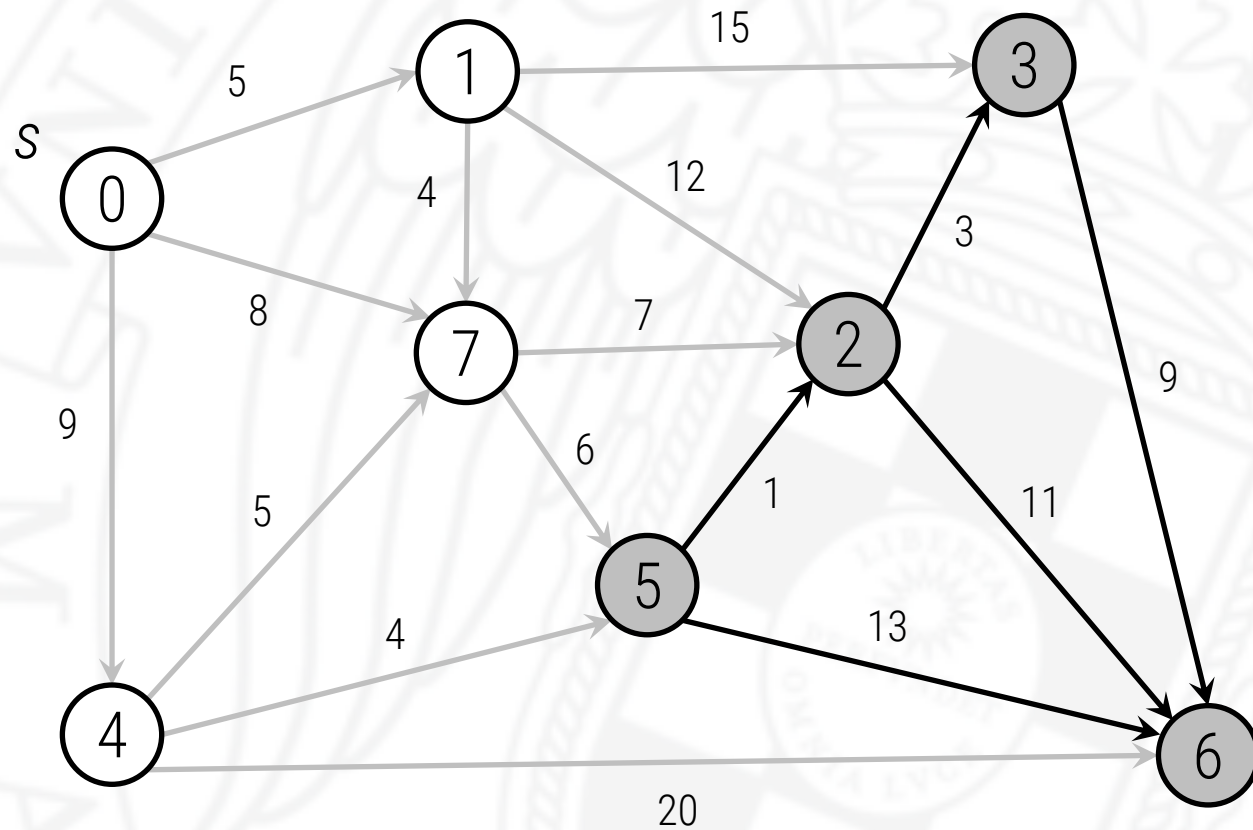
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



	v	dist[]	ult[]
	0	0	
	1	5	0 \rightarrow 1
	2	15	7 \rightarrow 2
	3	20	1 \rightarrow 3
\rightarrow	4	9	0 \rightarrow 4
	5	13	4 \rightarrow 5
	6	29	4 \rightarrow 6
	7	8	0 \rightarrow 7

Algoritmo de Dijkstra

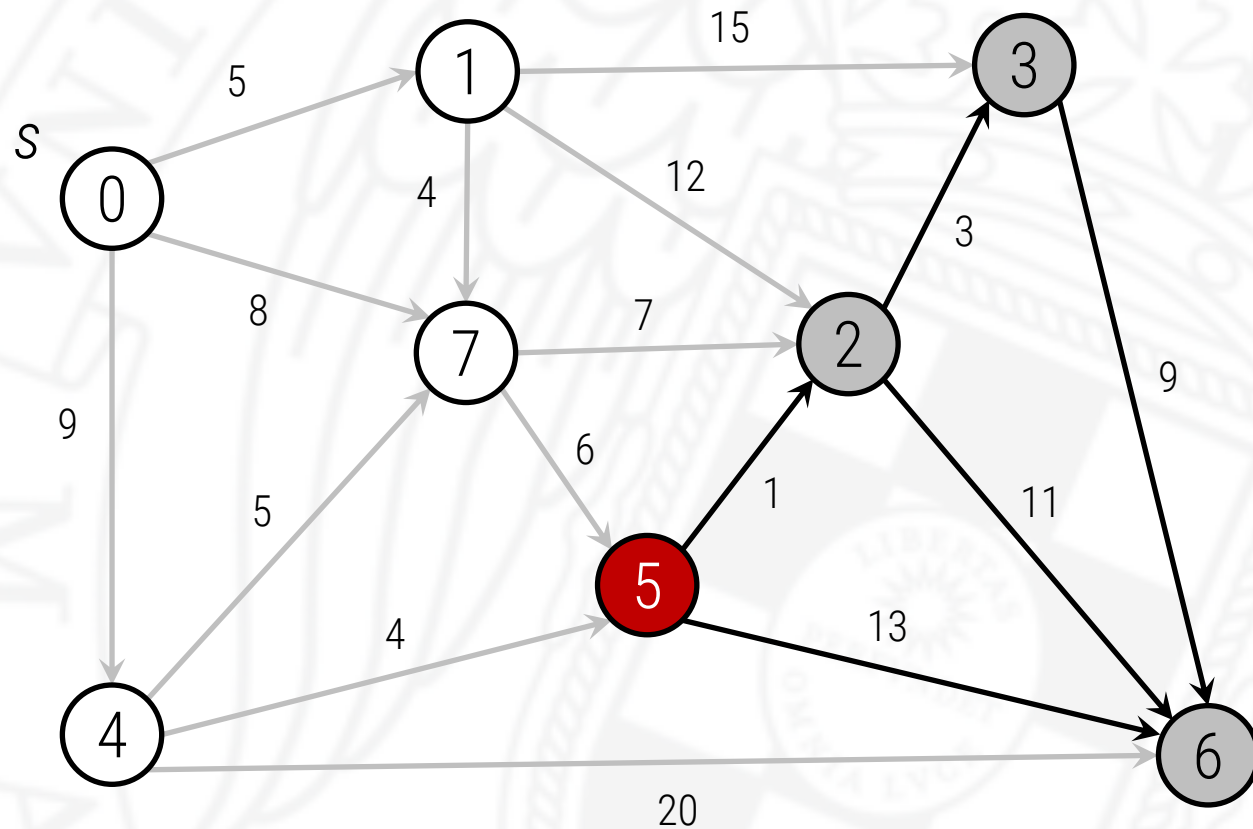
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
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

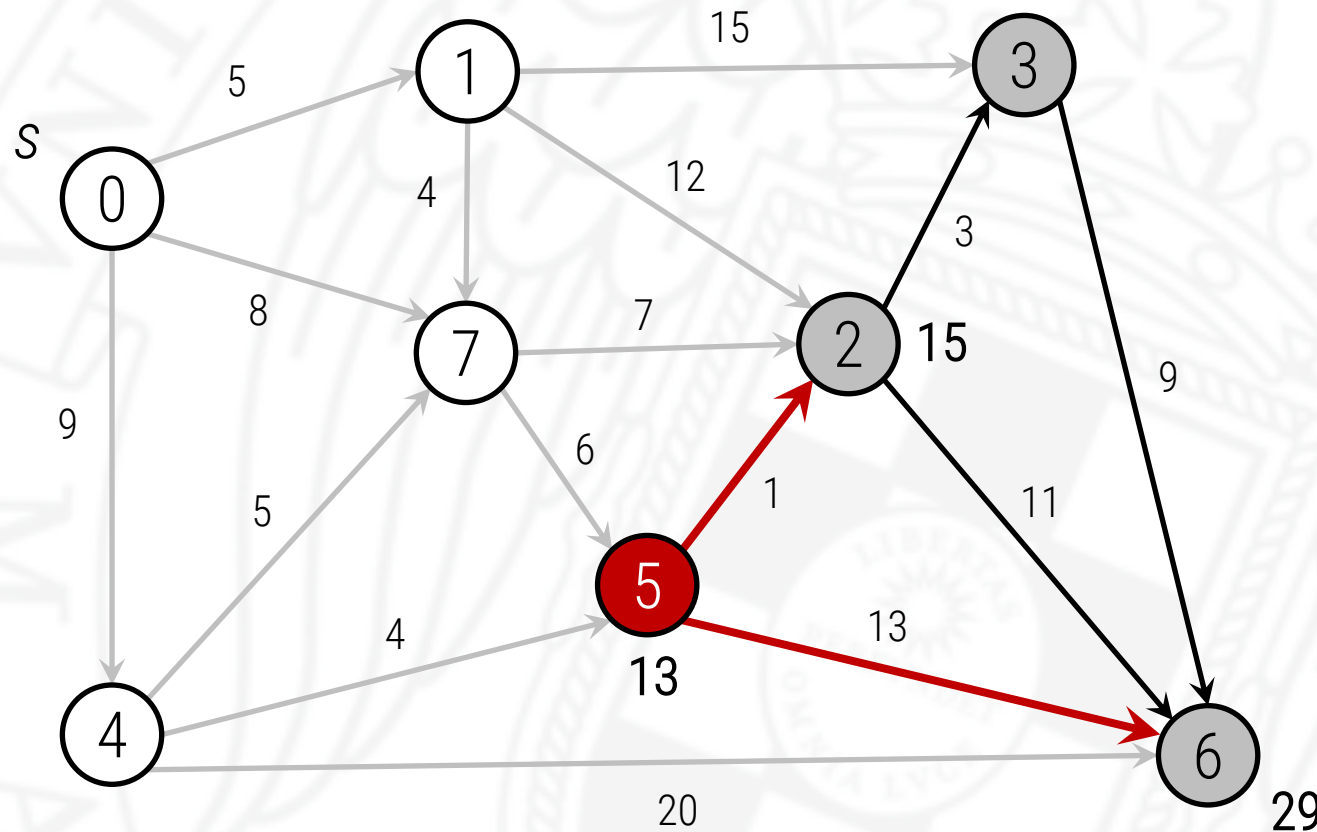
- Considera los vértices en orden creciente de distancia desde el origen.
- Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
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

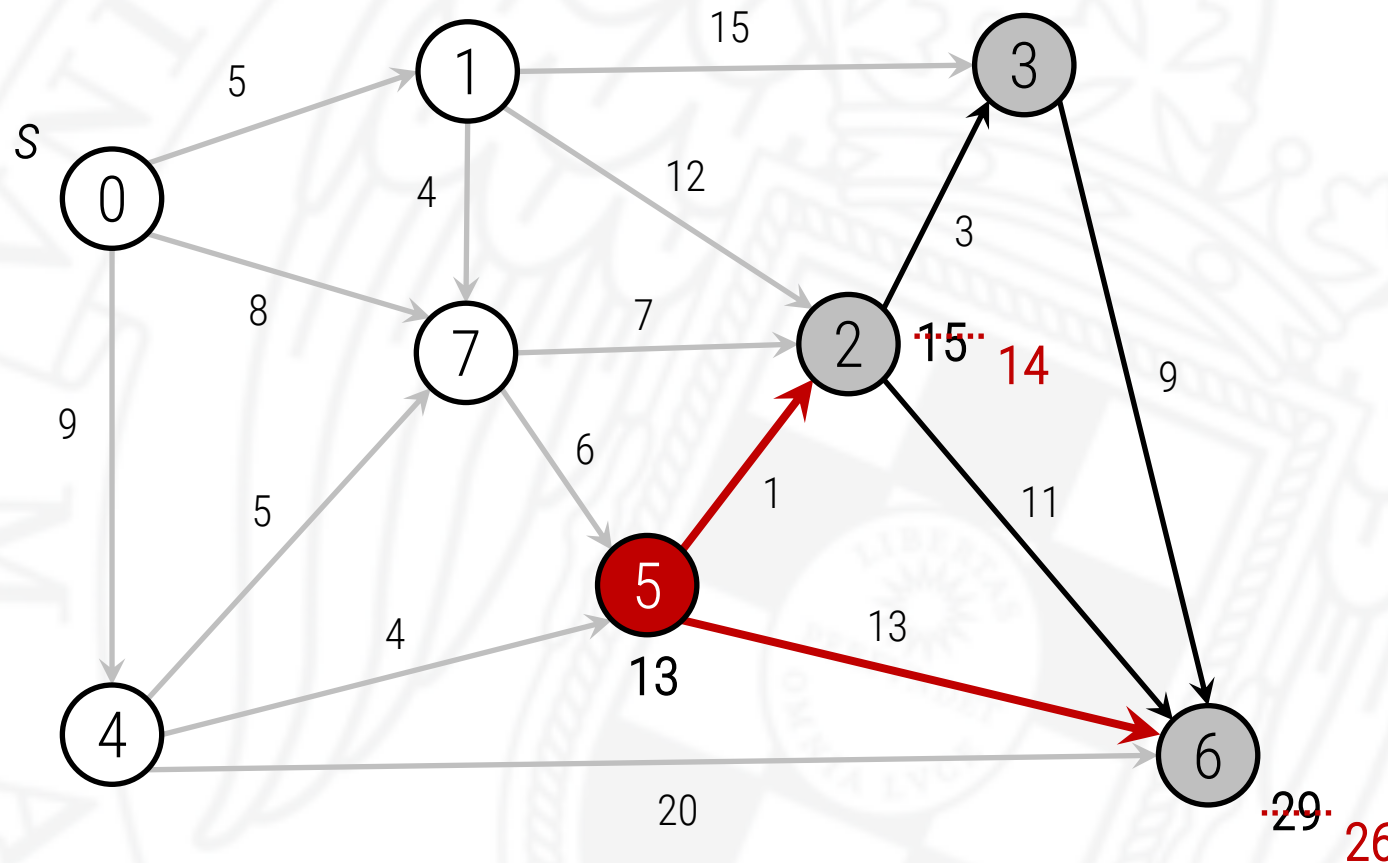
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
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

Algoritmo de Dijkstra

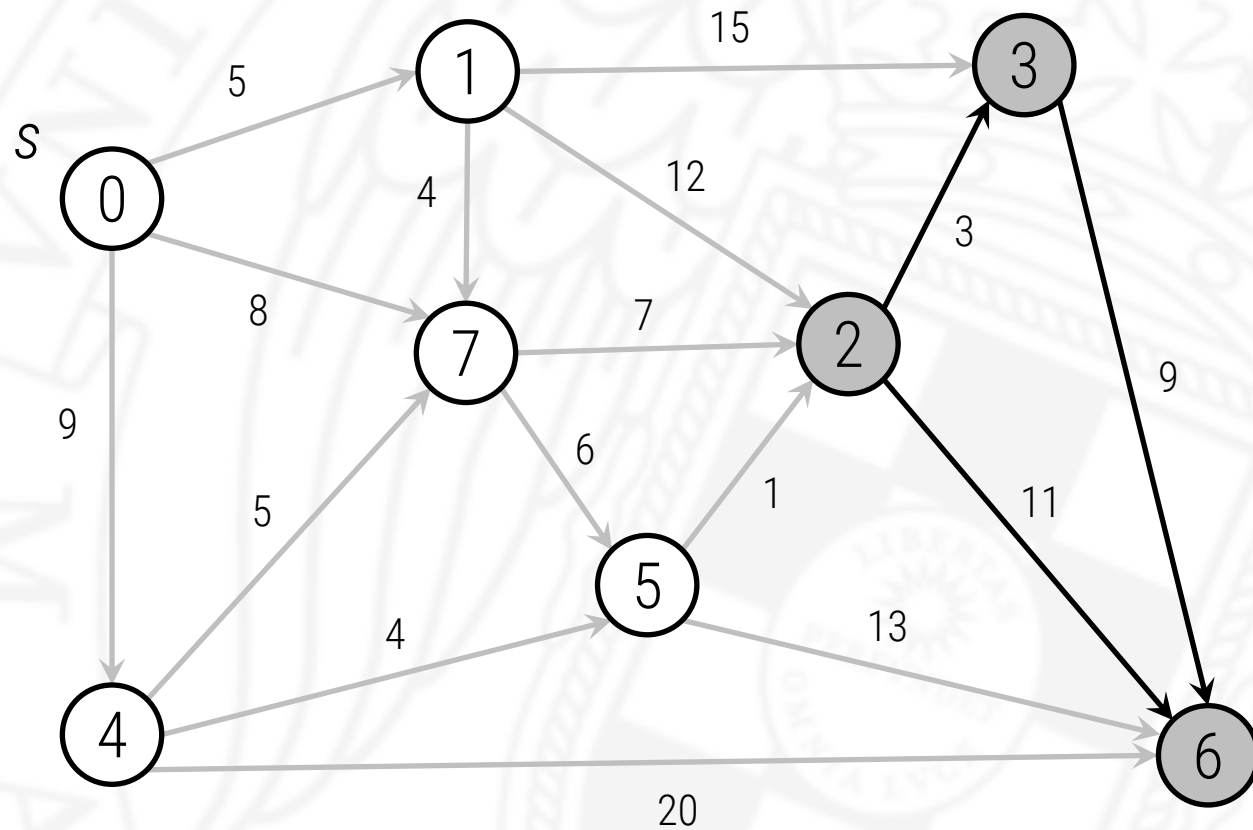
- Considera los vértices en orden creciente de distancia desde el origen.
- Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
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

Algoritmo de Dijkstra

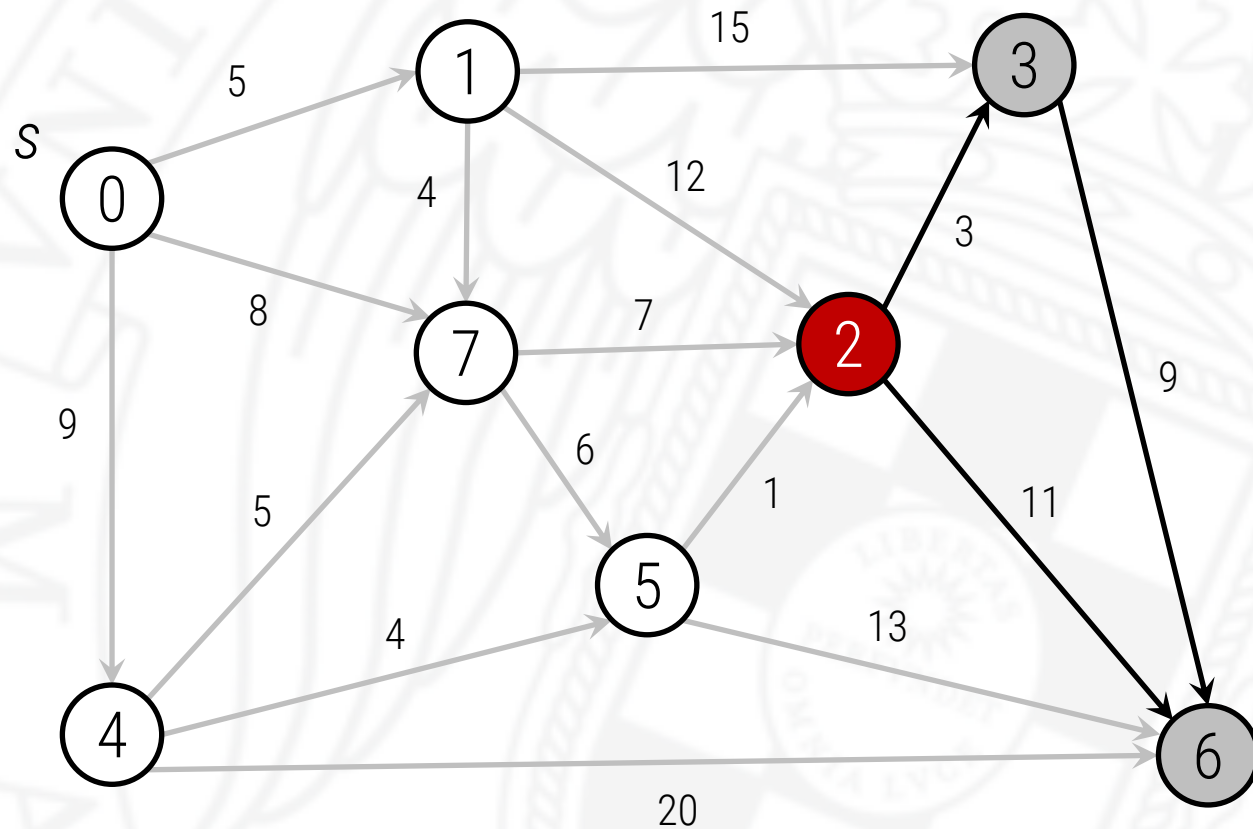
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
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

Algoritmo de Dijkstra

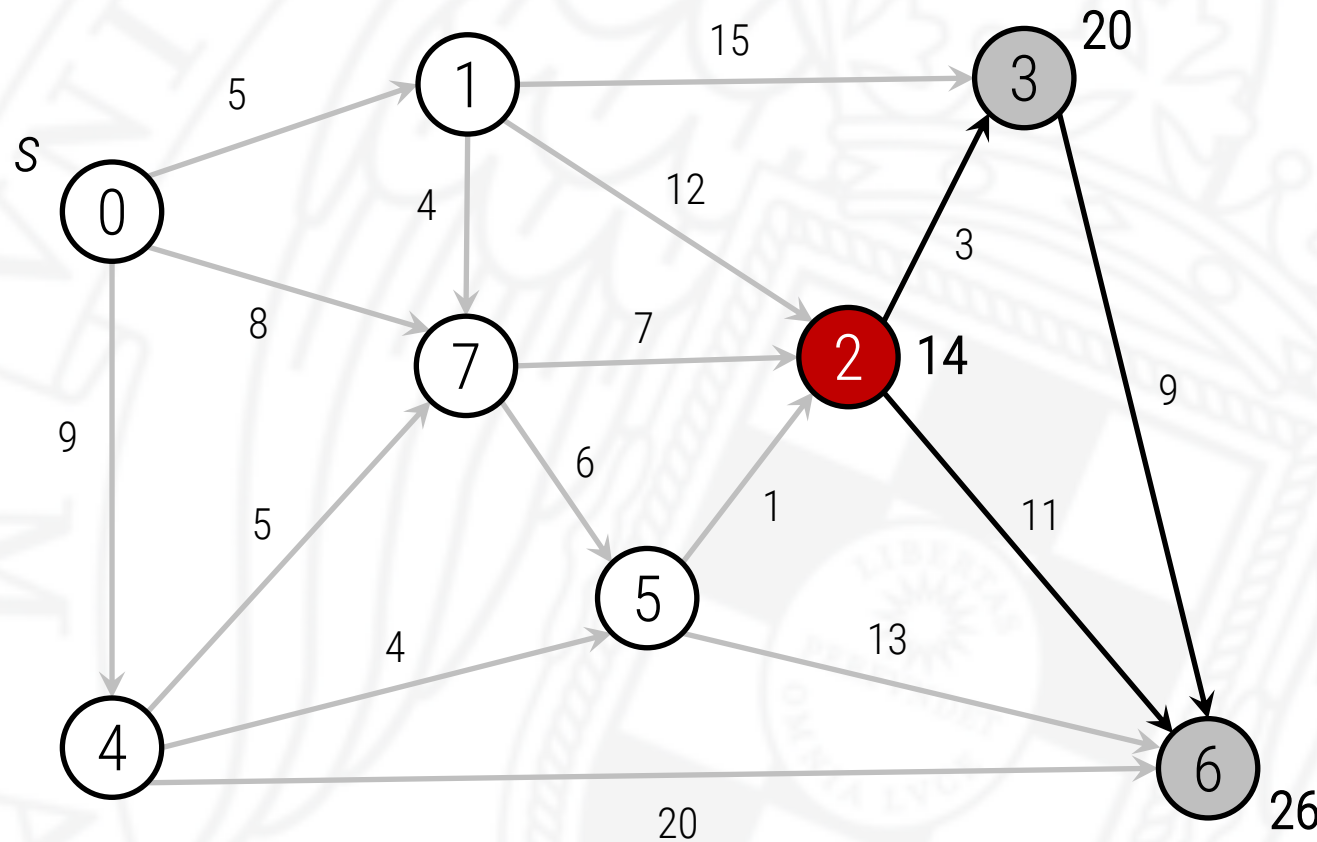
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
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

Algoritmo de Dijkstra

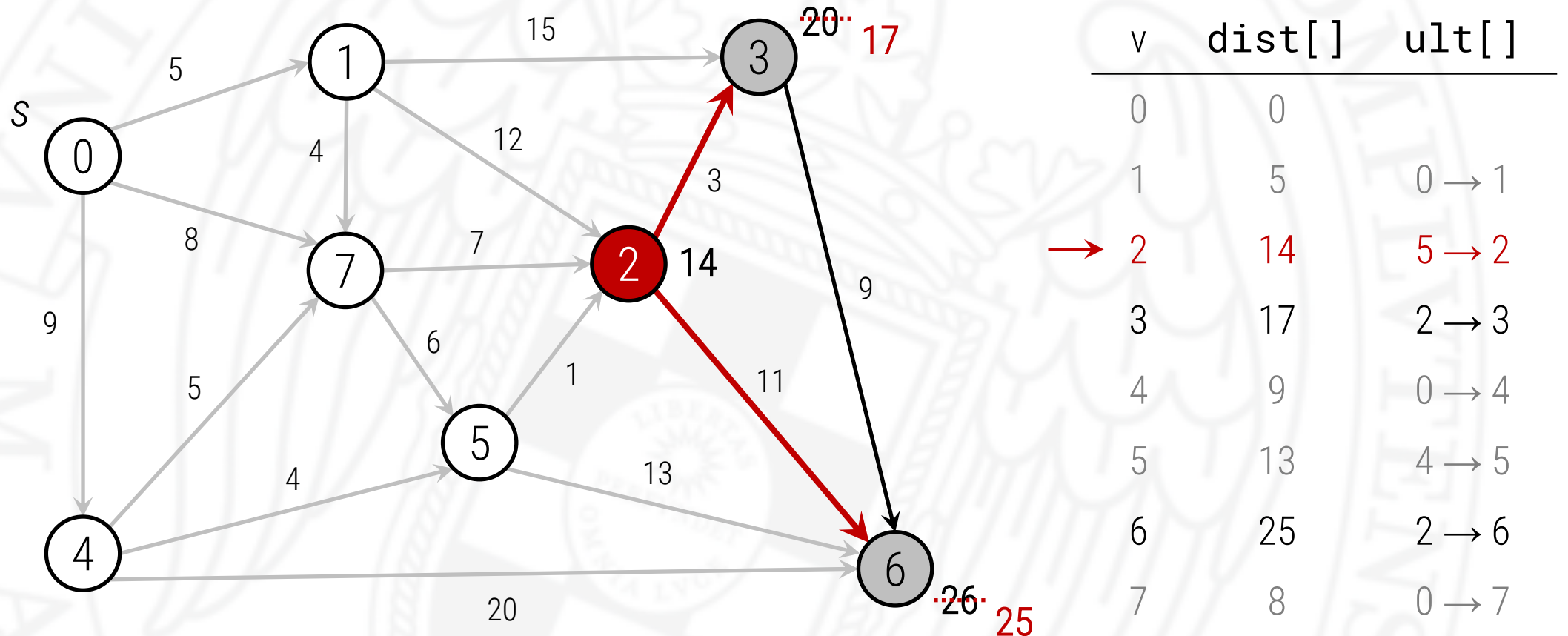
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
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

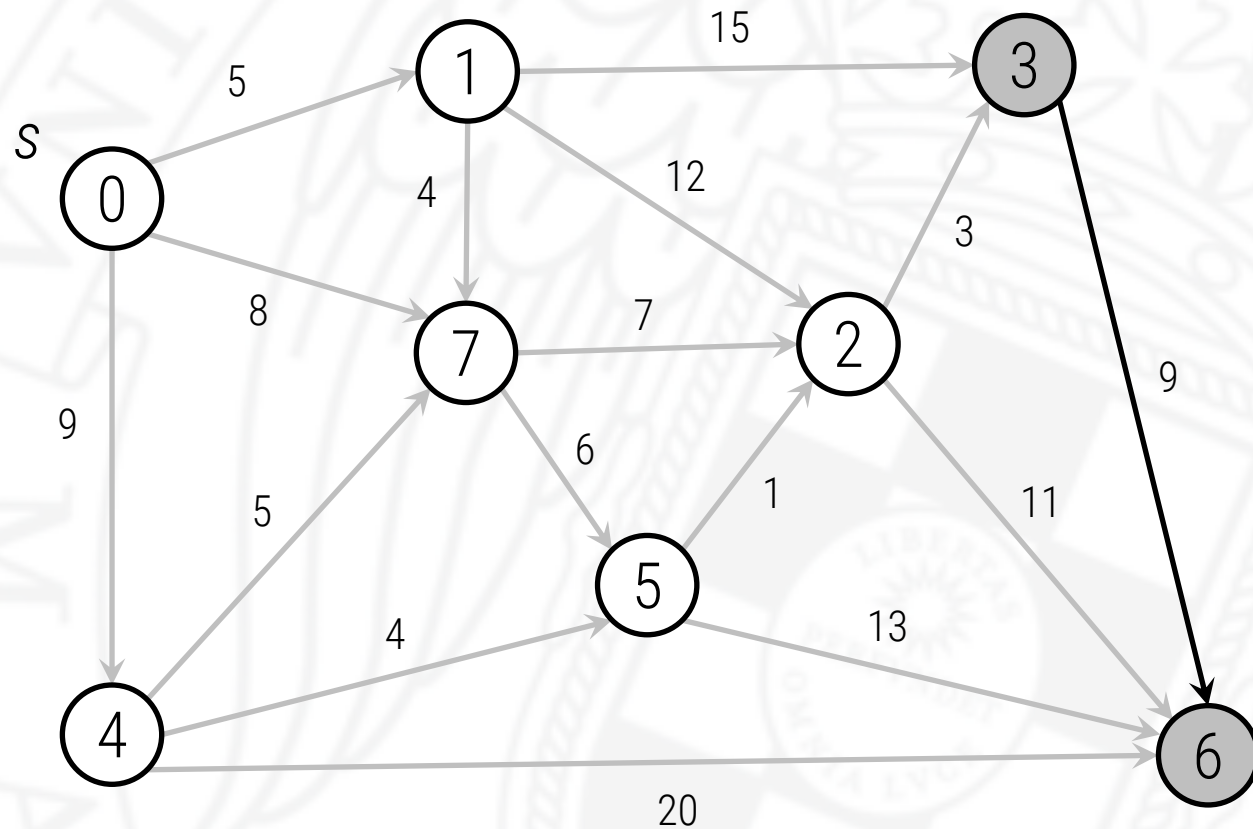
Algoritmo de Dijkstra

- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



Algoritmo de Dijkstra

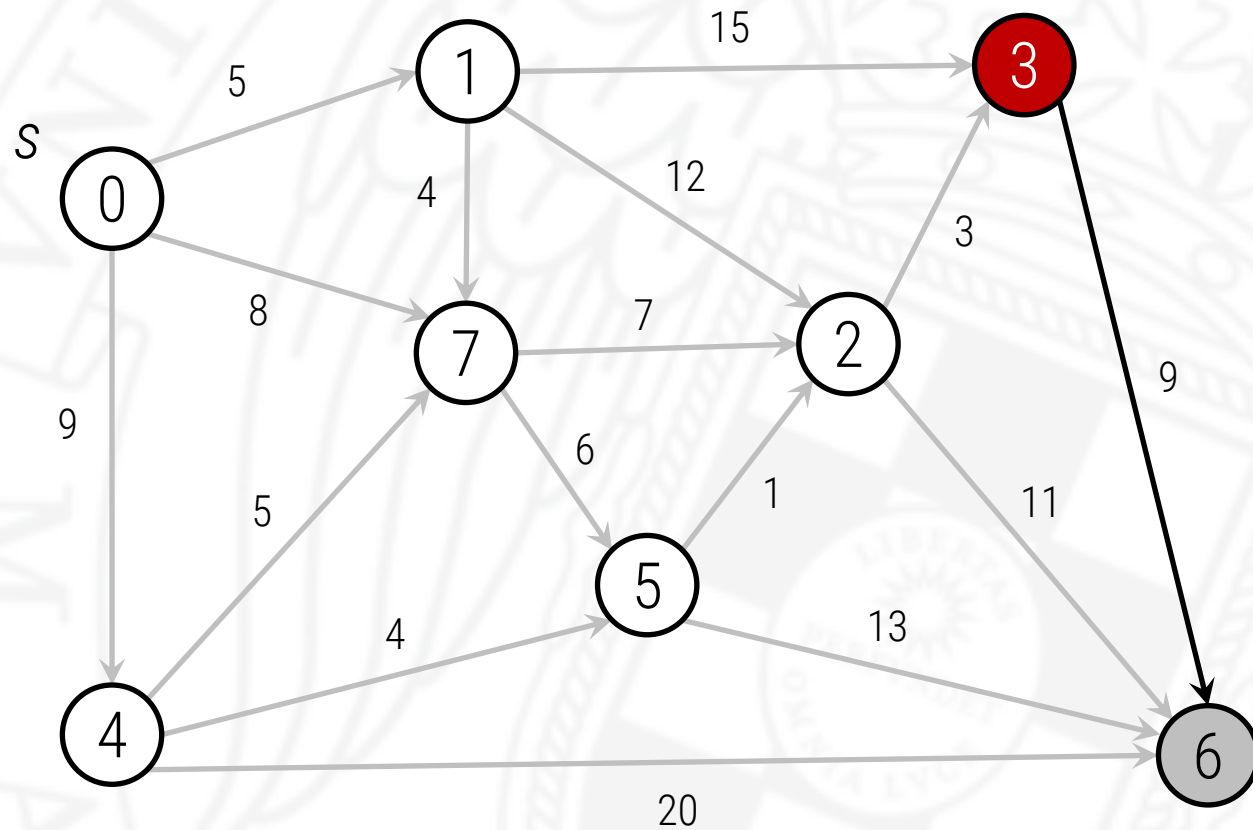
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	14	5 → 2
3	17	2 → 3
4	9	0 → 4
5	13	4 → 5
6	25	2 → 6
7	8	0 → 7

Algoritmo de Dijkstra

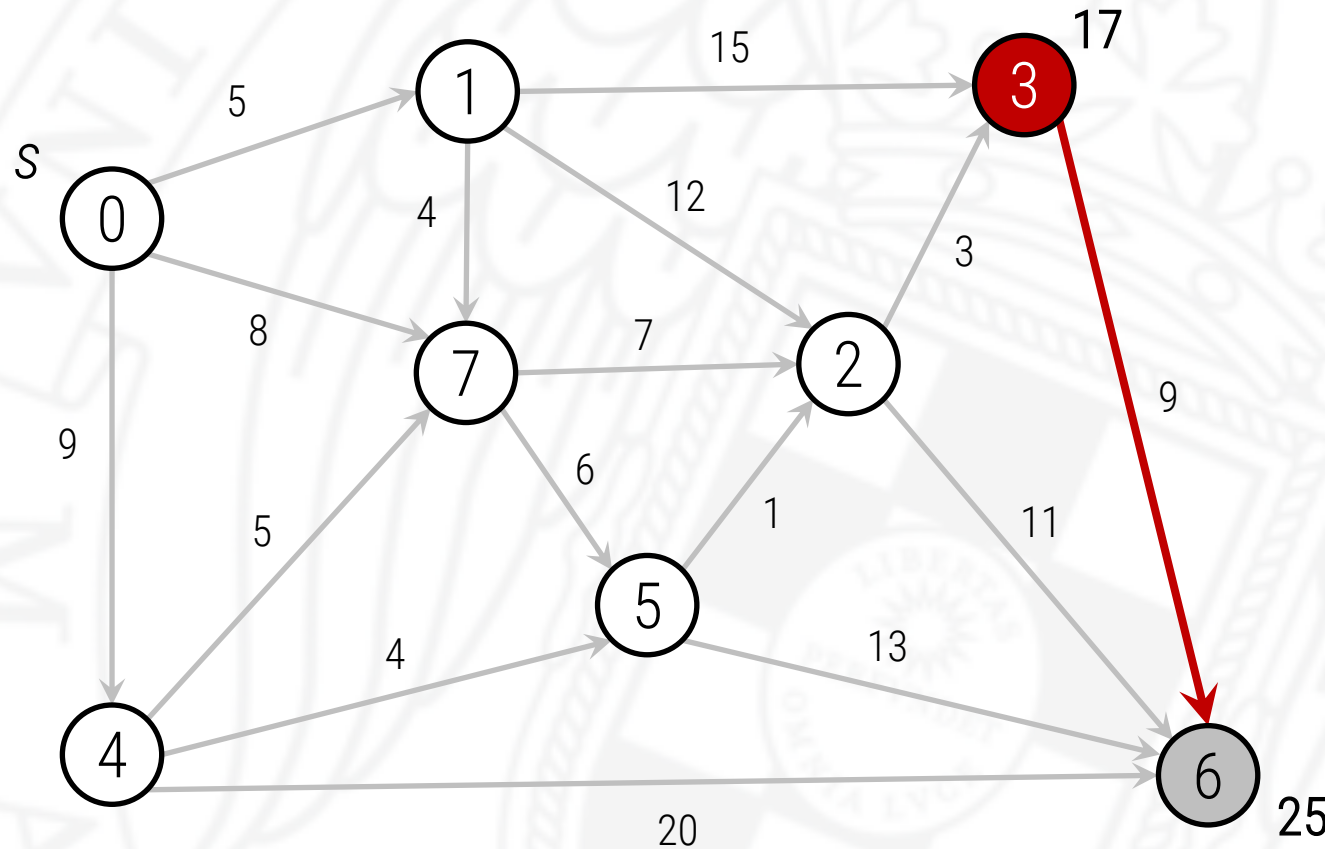
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	14	5 → 2
→ 3	17	2 → 3
4	9	0 → 4
5	13	4 → 5
6	25	2 → 6
7	8	0 → 7

Algoritmo de Dijkstra

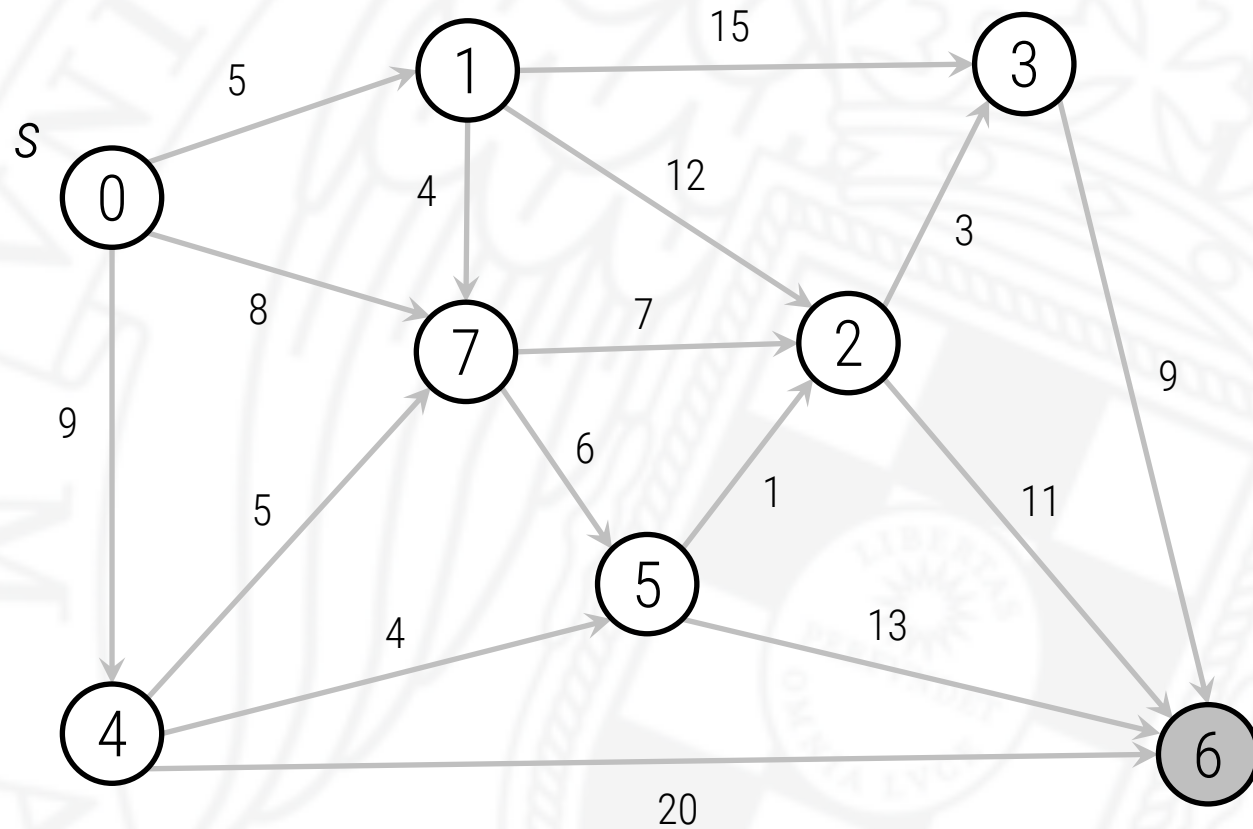
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	14	5 → 2
3	17	2 → 3
4	9	0 → 4
5	13	4 → 5
6	25	2 → 6
7	8	0 → 7

Algoritmo de Dijkstra

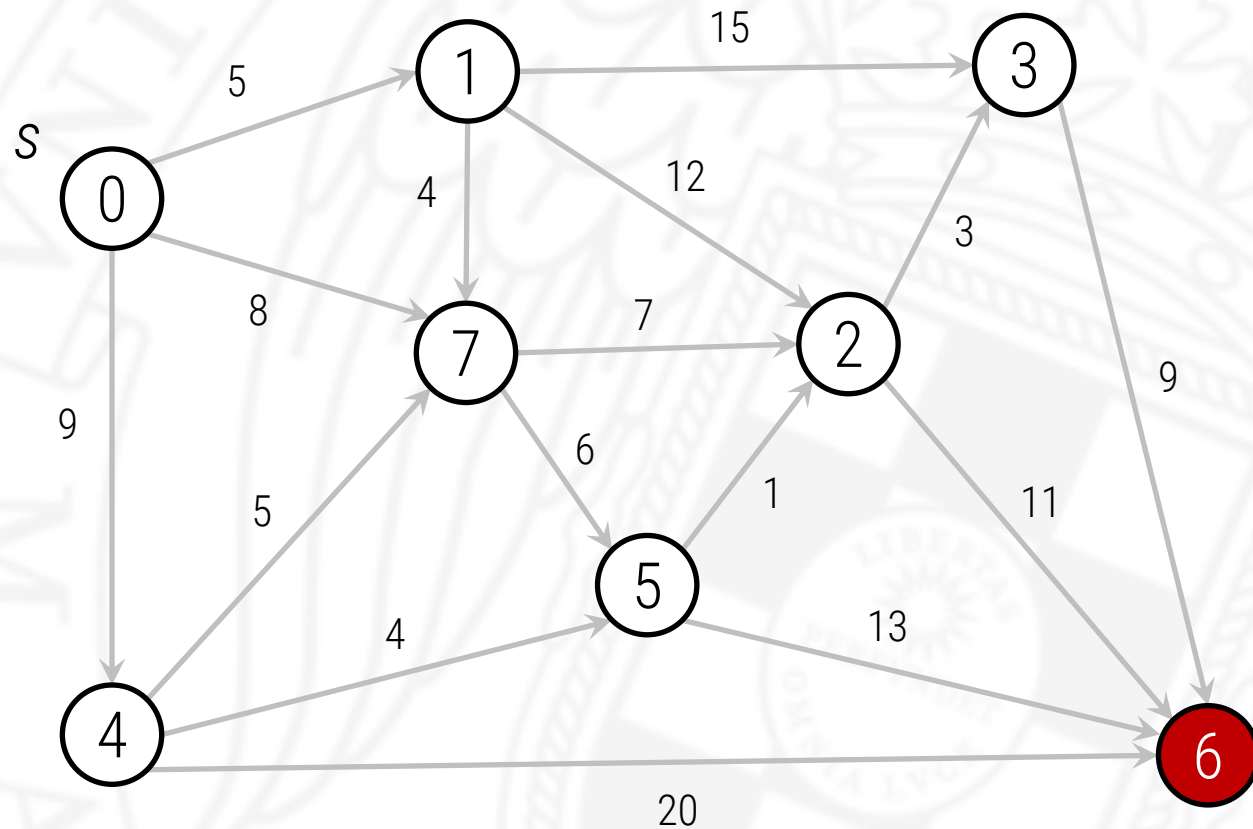
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	14	5 → 2
3	17	2 → 3
4	9	0 → 4
5	13	4 → 5
6	25	2 → 6
7	8	0 → 7

Algoritmo de Dijkstra

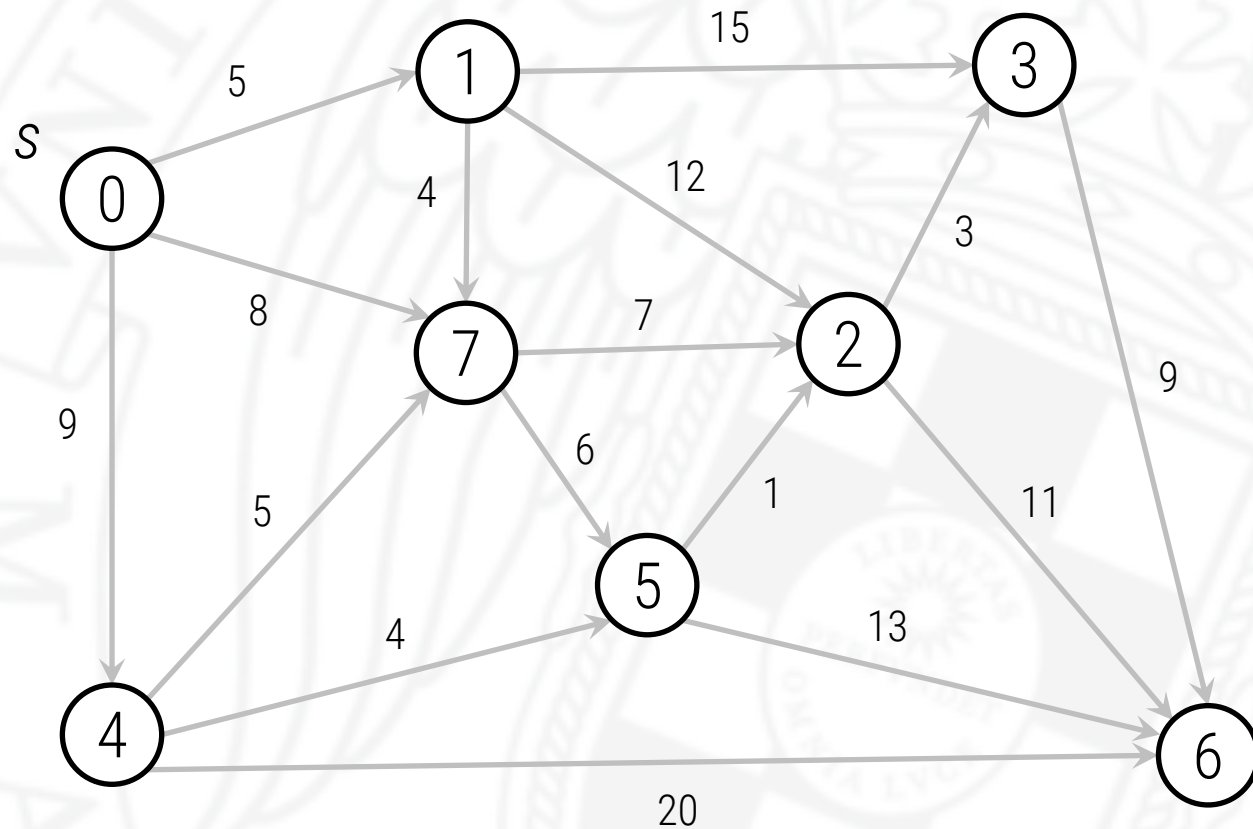
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	14	5 → 2
3	17	2 → 3
4	9	0 → 4
5	13	4 → 5
→ 6	25	2 → 6
7	8	0 → 7

Algoritmo de Dijkstra

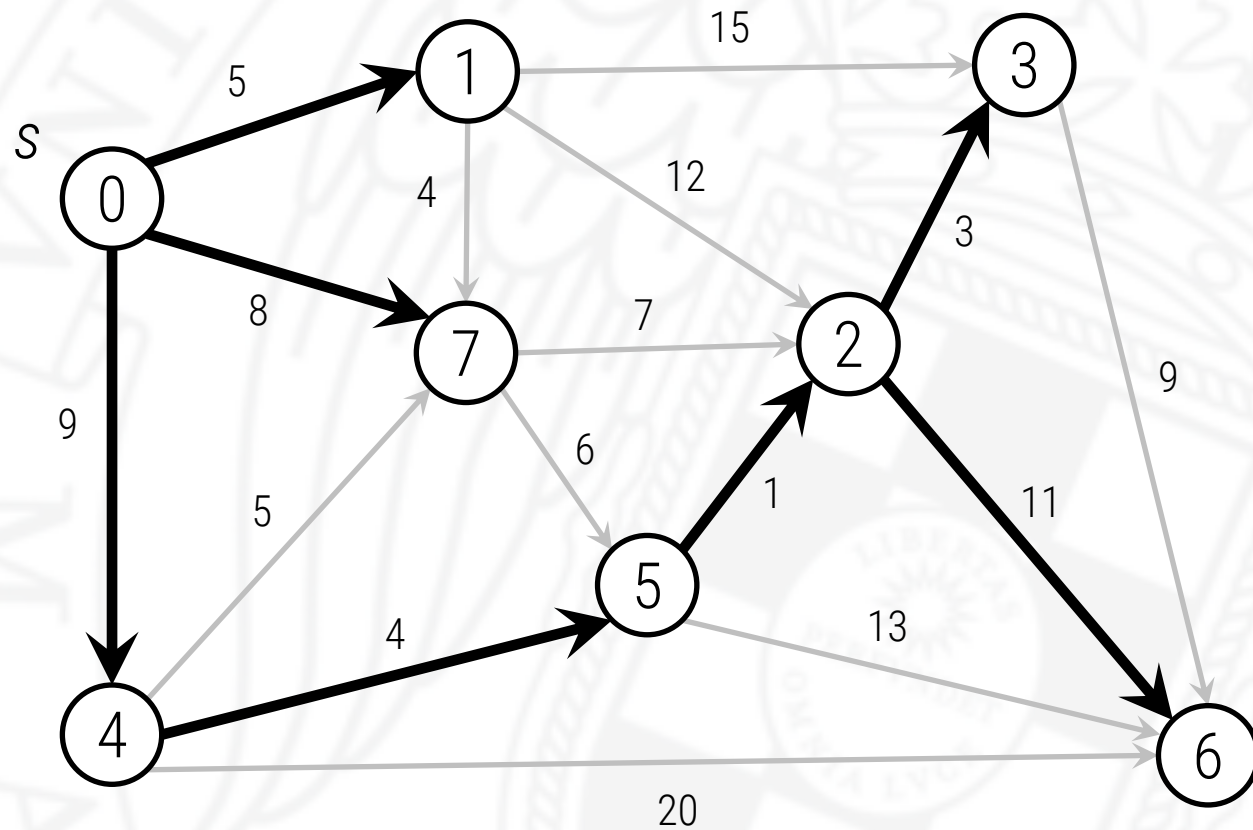
- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	14	5 → 2
3	17	2 → 3
4	9	0 → 4
5	13	4 → 5
6	25	2 → 6
7	8	0 → 7

Algoritmo de Dijkstra

- ▶ Considera los vértices en orden creciente de distancia desde el origen.
- ▶ Añade el vértice al árbol y relaja todas las aristas que salen de él.



v	dist[]	ult[]
0	0	
1	5	0 → 1
2	14	5 → 2
3	17	2 → 3
4	9	0 → 4
5	13	4 → 5
6	25	2 → 6
7	8	0 → 7

Algoritmo de Dijkstra, implementación

```
template <typename Valor>
class Dijkstra {
public:
    Dijkstra(DigrafoValorado<Valor> const& g, int orig) : origen(orig),
        dist(g.V(), INF), ult(g.V()), pq(g.V()) {
        dist[origen] = 0;
        pq.push(origen, 0);
        while (!pq.empty()) {
            int v = pq.top().elem; pq.pop();
            for (auto a : g.ady(v))
                relajar(a);
        }
    }
}
```

Algoritmo de Dijkstra, implementación

```
bool hayCamino(int v) const { return dist[v] != INF; }

Valor distancia(int v) const { return dist[v]; }

Camino<Valor> camino(int v) const {
    Camino<Valor> cam;
    // recuperamos el camino retrocediendo
    AristaDirigida<Valor> a;
    for (a = ult[v]; a.desde() != origen; a = ult[a.desde()])
        cam.push_front(a);
    cam.push_front(a);
    return cam;
}
```

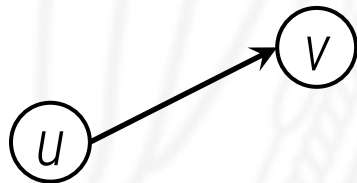
Algoritmo de Dijkstra, implementación

```
private:
    const Valor INF = std::numeric_limits<Valor>::max();
    int origen;
    std::vector<Valor> dist;
    std::vector<AristaDirigida<Valor>> ult;
    IndexPQ<Valor> pq;

    void relajar(AristaDirigida<Valor> a) {
        int v = a.desde(), w = a.hasta();
        if (dist[w] > dist[v] + a.valor()) {
            dist[w] = dist[v] + a.valor();  ult[w] = a;
            pq.update(w, dist[w]);
        }
    }
};
```


Algoritmo de Dijkstra, corrección

- ▶ Dado un digrafo valorado con aristas de costes no negativos, el algoritmo de Dijkstra calcula un árbol de caminos mínimos desde un origen.
- ▶ Cada arista $v \xrightarrow{a} w$ se relaja exactamente una vez (cuando se relaja v), haciendo que se cumpla **$\text{dist}[w] \leq \text{dist}[v] + a.\text{valor}()$**
- ▶ La desigualdad se mantiene hasta que termina el algoritmo, porque
 - **$\text{dist}[w]$** no puede aumentar
 - **$\text{dist}[v]$** no cambiará



Algoritmo de Dijkstra, análisis del coste

- El algoritmo de Dijkstra, aplicado a un grafo con V vértices y A aristas, calcula caminos mínimos desde el origen al resto de vértices en un tiempo en $O(A \log V)$ y con un espacio adicional en $O(V)$.

Operación	Frecuencia	Coste por operación
inicializar los vectores	1	V
construir cola prioridad	1	V
pop	V	$\log V$
update	A	$\log V$