GRAFOS III

Algoritmos en grafos

Definiciones

Definición 1

Un **árbol generador o recubridor** *de un grafo conexo* G *es un árbol que contiene todos los vértices de* G *y es subgrafo de* G.

Definición 2

Un **grafo ponderado** $G = (V, E, \omega)$ *es un grafo en el que a cada arista* $e \in E$ *se le asocia un peso* $\omega(e) \in \mathbb{R}$.

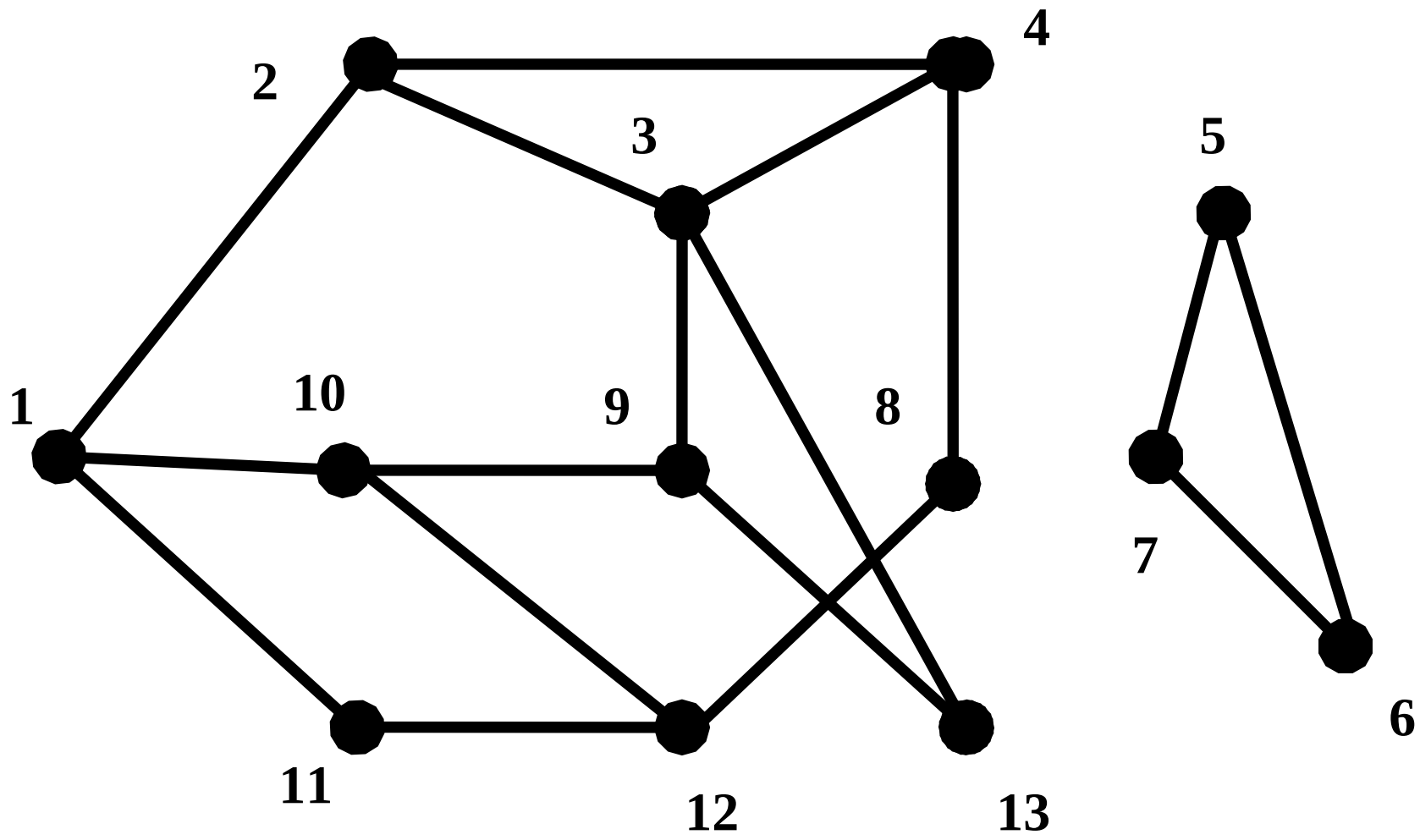
Definición 3

Un **árbol generador de peso mínimo** *de un grafo conexo ponderado es un árbol generador tal que la suma de los pesos de sus aristas es la más pequeña posible.*

Definición 4

Un **algoritmo voraz** *es áquel que en cada paso toma la elección óptima.*

Algoritmos de búsqueda en grafos simples y conexión



Algoritmo de búsqueda BA (BFS)

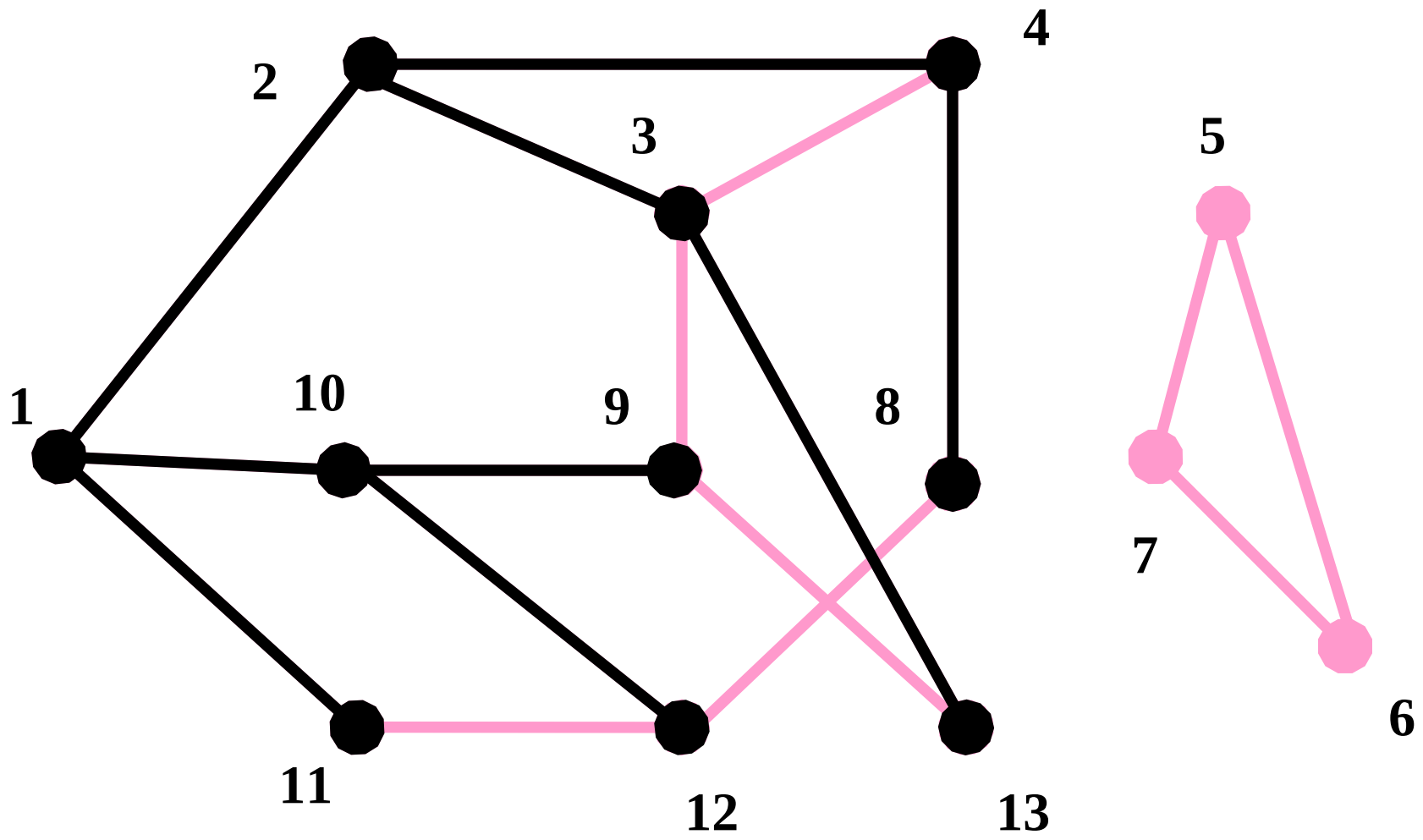
Sea G = (V , A) un grafo simple. |V| = n

- Se marca (elige) un vértices v_1 (inicial).
- Se localizan sus vecinos y se etiquetan
- Se incorporan las aristas correspondientes (que no formen ciclo).
- Se elige el «primero» de una numeración arbitraria.
- Se repite el proceso hasta que no se pueden incorporar más vértices.

Complejidad algorítmica $\boldsymbol{O}(|V| + |A|)$ ó bien $\boldsymbol{O}(n^2)$

Si G es conexo, el árbol será generador con raíz. Si no lo es, se detendrá al completar la componente conexa donde esté el vértice inicial.

Algoritmo de búsqueda BA (BFS)



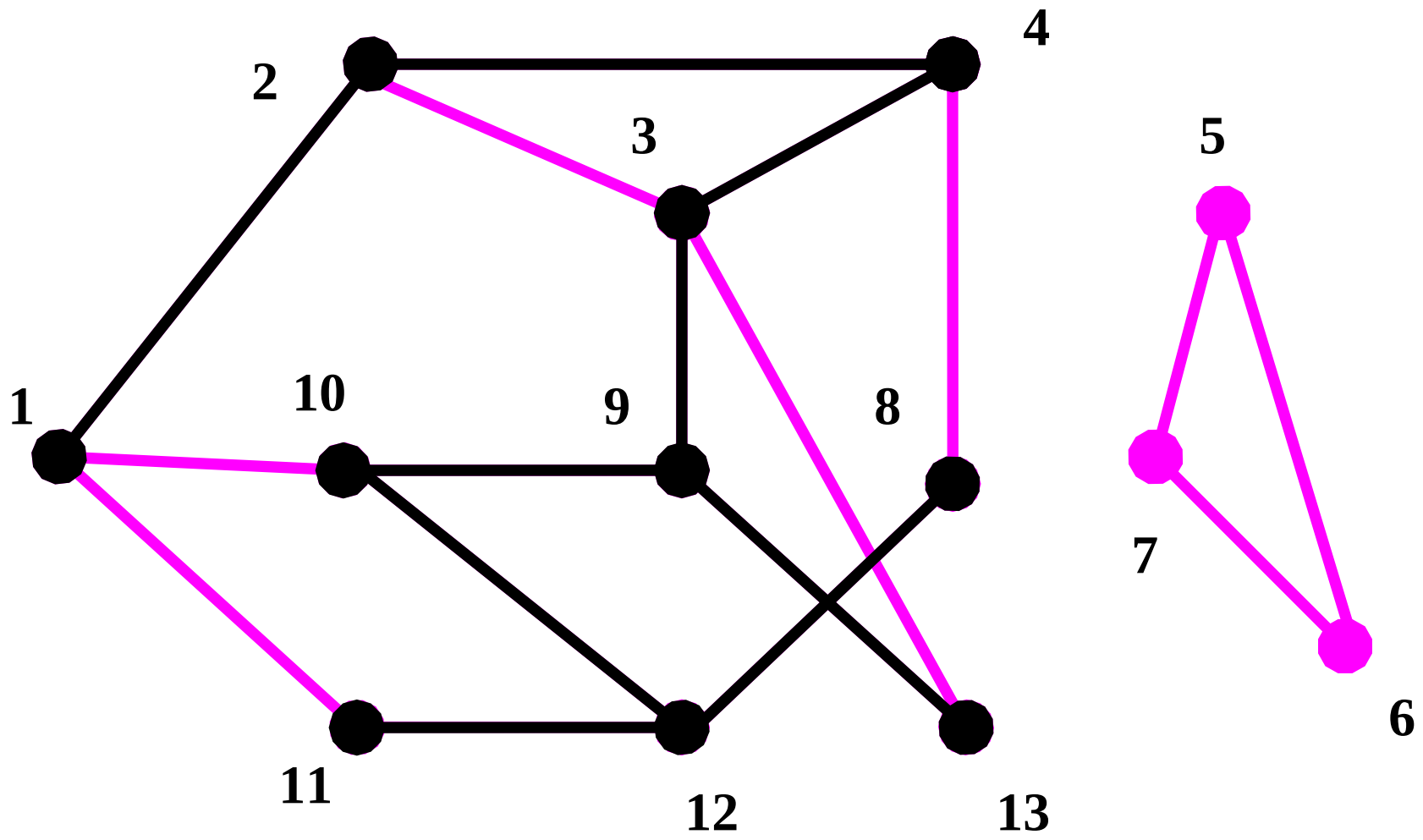
Algoritmo de búsqueda BP (DFS)

Sea G = (V , A) un grafo simple. |V| = n

- Se marca (elige) un vértice v_1 (inicial).
- Se localizan sus vecinos. Se toma uno de ellos, v_2, y se incorpora la arista $\{v_1 , v_2\}$, que no forme ciclo.
- Se avanza a v_2 y se buscan vecinos no visitados.
- Se repite el proceso.
- Cuando se llega a un vértice que no tenga vecinos nuevos que añadir (v_k), se retrocede al vértice anterior (v_{k-1})
- Se repite el proceso

  Coste computacional $\boldsymbol{O}(|V| + |A|)$ ó bien $\boldsymbol{O}(n^2)$

Algoritmo de búsqueda BP (DFS)



PRIM Y KRUSKAL

(minimum spanning tree- MST)

PRIM Y KRUSKAL

(minimum spanning tree- MST)

ALGORITMO DE PRIM

(Árbol generador de peso mínimo)

Sea G = (V , A) un grafo conexo ponderado. |V| = n

1. Se marca (elige) un vértice **u** (inicial) de G y se considera el árbol **T = {u}**.
2. Se considera la arista **a** de mínimo peso que une un vértice de T y un vértice que no es de T, y se hace .
3. Si el número de aristas de T es n – 1 el algoritmo termina. En caso contrario se vuelve al paso 2.

ALGORITMO DE PRIM

procedure *Prim(G: grafo ponderado conexo con n vértices)*

$T_1 = (V_1, E_1)$ *donde* $E_1 = \{e_1\}$, $e_1 = \{x_0, x_1\}$ *es la arista con peso mínimo* ω_{min} *y* $V_1 = \{x_0, x_1\}$.

for $i = 1$ **to** $n - 2$

begin

$e_{i+1} = \{x_j, x_{i+1}\}$ *arista de peso mínimo incidente con un vértice* x_j *de* $T_i = (V_i, E_i)$ *y que no forme un ciclo si se le añade a* T_i

$T_{i+1} = (V_i \cup \{x_{i+1}\}, E_i \cup \{e_{i+1}\}) = (V_{i+1}, E_{i+1})$

end

Notas:

- La arista e_i $(i = 1, \ldots, n-1)$ puede no ser única.
- El árbol generador de peso mínimo puede no ser único.
- El resultado es un árbol con n vértices, luego tiene $n - 1$ aristas.

Coste computacional $\boldsymbol{O}(n^2)$, ó bien $\boldsymbol{O}(|V|\log|V| + |A|)$

PRIM

ALGORITMO DE KRUSKAL

(Árbol generador de peso mínimo)

Sea G = (V , A) un grafo conexo ponderado. |V| = n

1. Se elige la arista **a** de mínimo peso, y se considera .
2. Sea **b** la arista de mínimo peso que no pertenece a **T,** y que al unir a T no presenta ciclos y se hace
3. Si **T** tiene n – 1 aristas el algoritmo termina. En caso contrario se vuelve al paso 2.

ALGORITMO DE KRUSKAL

procedure *Kruskal(G: grafo ponderado conexo con* n *vértices)*

$T_0 = (V, E_0)$ *con* $E_0 = \emptyset$

for $i = 1$ **to** $n - 1$

begin

e_i = *arista de peso mínimo que no forme un ciclo si se le añade a* $T_{i-1} = (V, E_{i-1})$

$T_i = (V, E_{i-1} \cup \{e_i\}) = (V, E_i)$

end

Notas:

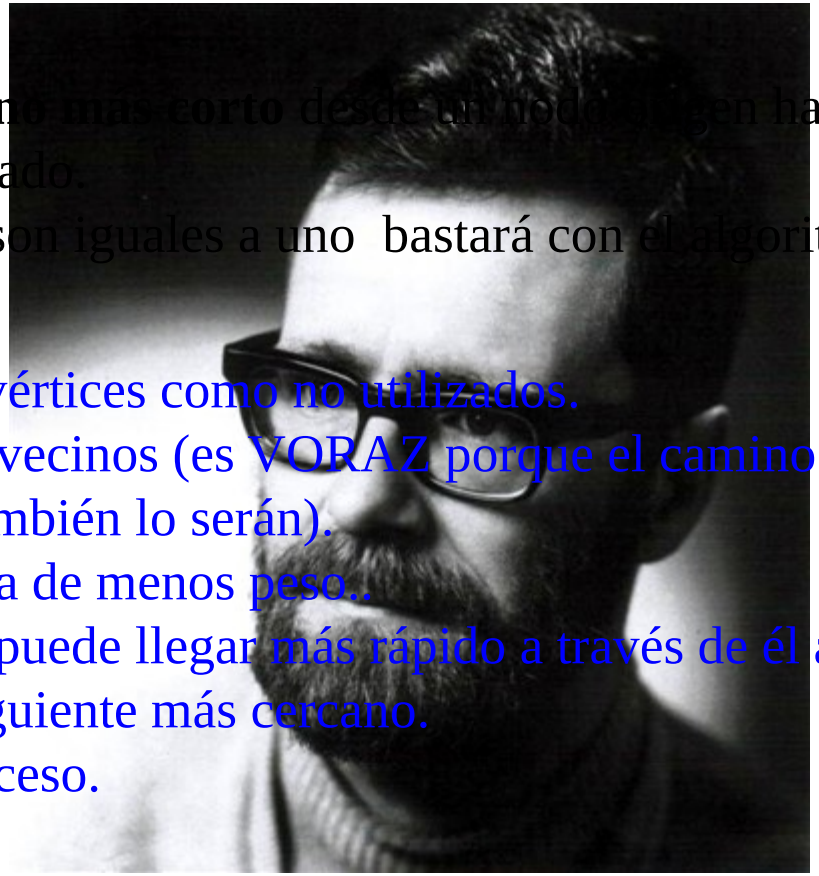
- La arista e_i $(i = 1, \ldots, n-1)$ puede no ser única.
- La arista e_i puede no ser incidente con ningún vértice en T_{i-1}.

Coste computacional: ***O***(|A|log(|V|)

KRUSKAL

ALGORITMO DE DIJKSTRA

(Problema del camino mínimo 1959)



Determina el **camino** más corto desde un nodo origen hacia los demás nodos en un grafo ponderado.
Si todos los pesos son iguales a uno bastará con el algoritmo BA.

¿Cómo trabaja?:

- Se marcan los vértices como no utilizados.
- Se evalúan sus vecinos (es VORAZ porque el camino mínimo contendrá caminos que también lo serán).
- Se elige la arista de menos peso..
- Se evalúa si se puede llegar más rápido a través de él a los demás.
- Se escoge el siguiente más cercano.
- Se repite el proceso.

ALGORITMO DE DIJKSTRA

(Problema del camino mínimo 1959)

Sea G = (V, A, W) un grafo conexo ponderado con $\Omega = (\omega_{ij})$, su matriz de pesos.

Este algoritmo construye, en cada paso, un **camino de mínimo peso** desde un vértice inicial v_p a otro vértice.

Se usa:

- Una lista, L, que contendrá los vértices para los que se ha construido el camino.
- Un vector de pesos, D, que tendrá, al final, los pesos mínimos.
- Inicio: $L = \{v_p\}$; $D = \Omega(p,:)$, p-ésima fila de pesos.

ALGORITMO DE DIJKSTRA

(Problema del camino mínimo 1959)

Pseudocódigo:

Inicio: Ω ; v_p ; $L = \{v_p\}$; $D = \Omega(p , :)$

Mientras que $V \setminus L \neq \varnothing$

tomar $v_k \in V \setminus L$ con $D(k)$ mínimo

hacer $L = L \cup \{v_k\}$

para cada v_j de $V \setminus L$

si $D(j) > D(k) + \Omega(k,j)$

hacer $D(j) = D(k) + \Omega(k,j)$

fin

fin

fin

ALGORITMO DE DIJKSTRA

(1) Paso inicial: Marcamos el origen s con la etiqueta permanente $\boxed{(0, s)}$.

El resto de los vértices $j \in V$ $(j \neq s)$ se marcan con etiquetas temporales:

- Si $\{j, s\} \in E$, se le asigna la etiqueta $(\omega_{s,j}, s)$.
- Si $\{j, s\} \notin E$, se le asigna la etiqueta $(\infty, -)$.

(2) Sea $v \in V$ el último vértice que se ha vuelto permanente. Examinamos cada vértice temporal j comparando δ_j con el valor de $\delta_v + \omega_{v,j}$:

- Si $\delta_v + \omega_{v,j} < \delta_j$, cambiamos (δ_j, P_j) por $(\delta_v + \omega_{v,j}, v)$.
- Si $\delta_v + \omega_{v,j} \geq \delta_j$, no hacemos nada.

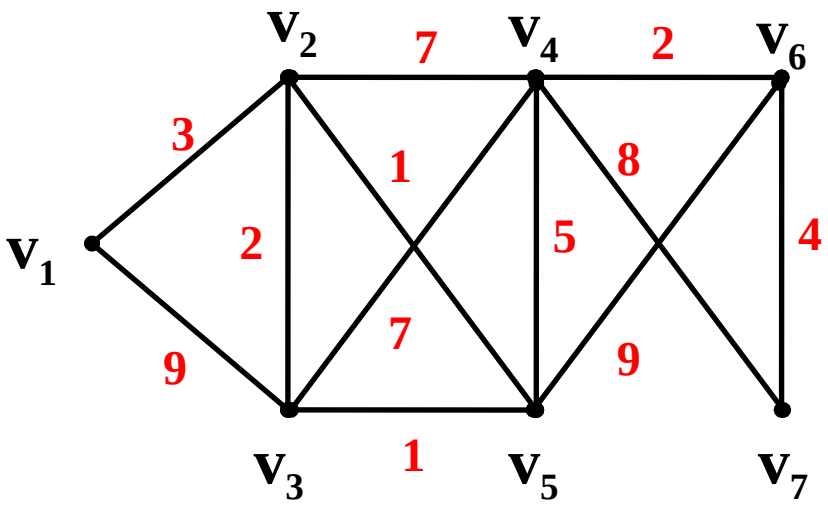
(3) De entre todos los vértices temporales j elegimos el que tenga el estimador δ_j más pequeño $(= \delta_{\text{min}})$.

- Si $\delta_{\text{min}} = \infty$, el algoritmo termina: no hay camino entre s y t.
- Si $\delta_{\text{min}} < \infty$, marcamos dicho vértice con la etiqueta permanente $\boxed{(\delta_{\text{min}}, P_j)}$

(4) Si t es el vértice cuya etiqueta $\boxed{(\delta_t, P_t)}$ se ha hecho permanente, el algoritmo termina. La longitud del camino más corto entre s y t es δ_t y dicho camino se obtiene siguiendo las etiquetas permanentes en sentido contrario $t \to P_t \to \cdots \to s$. Si no es t, volver al paso (2).

Algoritmo de Dijkstra (ejemplo)

Sea G, y Ω su matriz de pesos

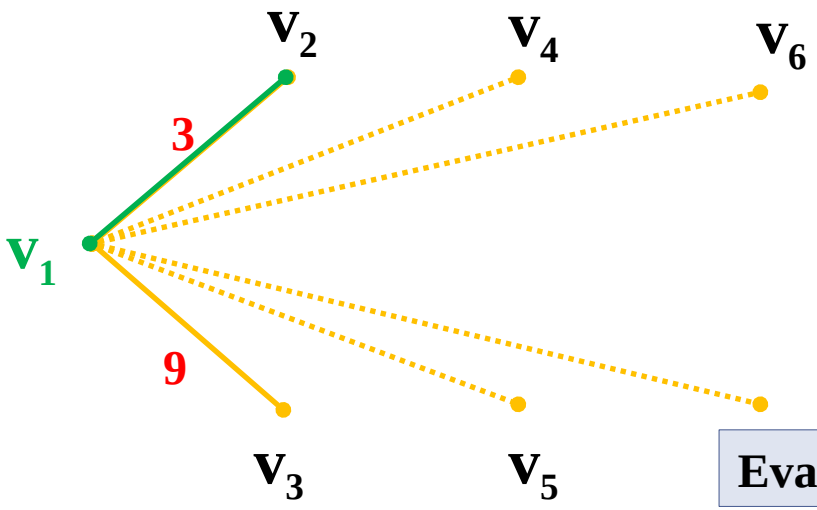


Etiquetado inicial permanente

Ω	Paso 1						
v_1	(0 , v1)						
v_2	(3 , v1)						
v_3	(9 , v1)						
v_4	∞						
v_5	∞						
v_6	∞						
v_7	∞						

Algoritmo de Dijstra (ejemplo)

Sea G, y Ω su matriz de pesos

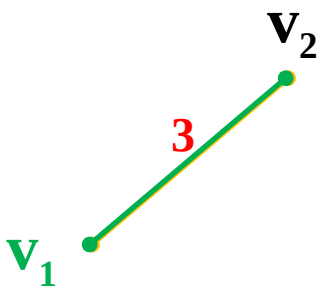


Evaluación: mín{P(2), P(3), P(4), P(5), P(6), P(7)}

Ω	Paso 1	Paso 2					
v_1	(0 , v1)						
v_2	(3 , v1)						
v_3	(9 , v1)						
v_4	∞						
v_5	∞						
v_6	∞						
v_7	∞						

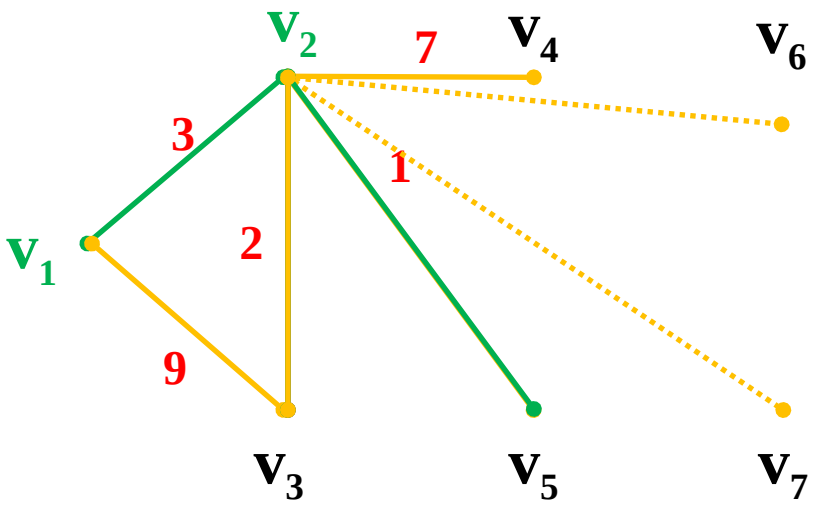
Algoritmo de Dijstra (ejemplo)

Sea G, y Ω su matriz de pesos



Ω	Paso 1	Paso 2					
v_1	(0 , v1)	*					
v_2	(3 , v1)	(3 , v1)					
v_3	(9 , v1)						
v_4	∞						
v_5	∞						
v_6	∞						
v_7	∞						

Sea G, y Ω su matriz de pesos



mín{P(3), P(4), P(5), P(6), P(7)}

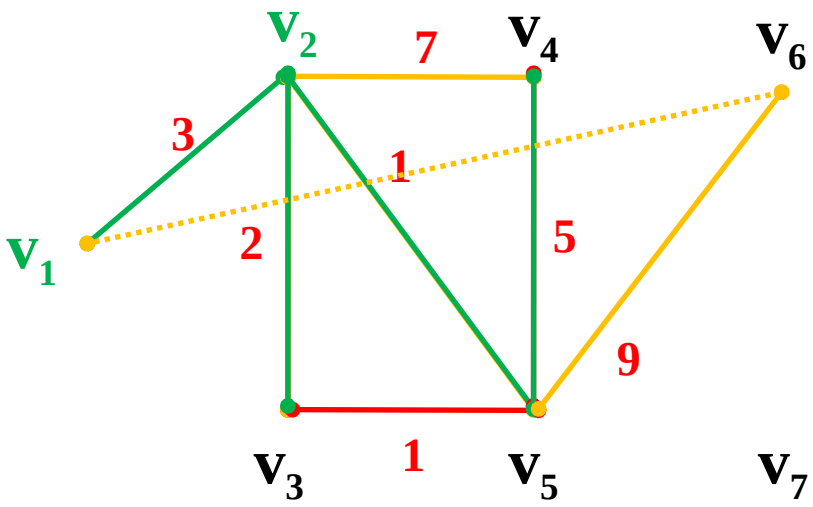
Ω	Paso 1	Paso 2				
v_1	(0 , v1)	*				
v_2	(3 , v1)	(3 , v1)				
v_3	(9 , v1)	(5 , v2)				
v_4	∞	(10 , v2)				
v_5	∞	(4 , v2)				
v_6	∞	∞				
v_7	∞	∞				

¿P(3) < P(2) + Ω(2, 3) ?

¿9 < 3 + 2?

¿∞ < 3 + 1?

Sea G, y Ω su matriz de pesos



Ω	Paso 1	Paso 2	Paso 3			
v_1	(0 , v1)	*	*			
v_2	(3 , v1)	(3 , v1)	*			
v_3	(9 , v1)	(5 , v2)	(5 , v2)			
v_4	∞	(10 , v2)	(9, v5)			
v_5	∞	(4 , v2)	(4 , v2)			
v_6	∞	∞	(13, v5)			
v_7	∞	∞	∞			

mín{P(3), P(4), P(6), P(7)}

¿P(3) < P(5) + Ω(3, 5) ?

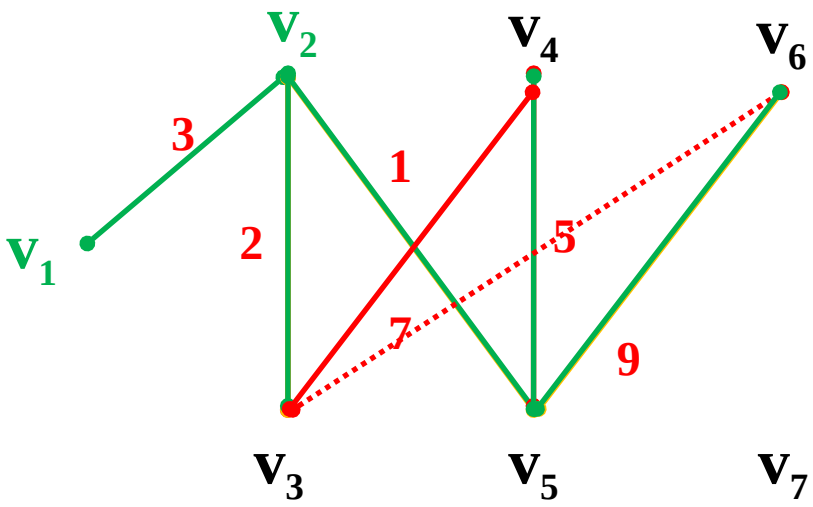
¿P(4) < P(5) + Ω(4,5) ?

¿P(6) < P(5) + Ω(5, 6) ?

¿P(7) < P(5) + Ω(5,7) ?

¿∞ < 4 + ∞ ?

Sea G, y Ω su matriz de pesos



Ω	Paso 1	Paso 2	Paso 3	Paso 4		
v_1	(0 , v1)	*	*	*		
v_2	(3 , v1)	(3 , v1)	*	*		
v_3	(9 , v1)	(5 , v2)	(5 , v2)	(5 , v2)		
v_4	∞	(10 , v2)	(9, v5)	(9, v5)		
v_5	∞	(4 , v2)	(4 , v2)	*		
v_6	∞	∞	(13, v5)	(13, v5)		
v_7	∞	∞	∞	∞		

mín{P(4), P(6), P(7)}

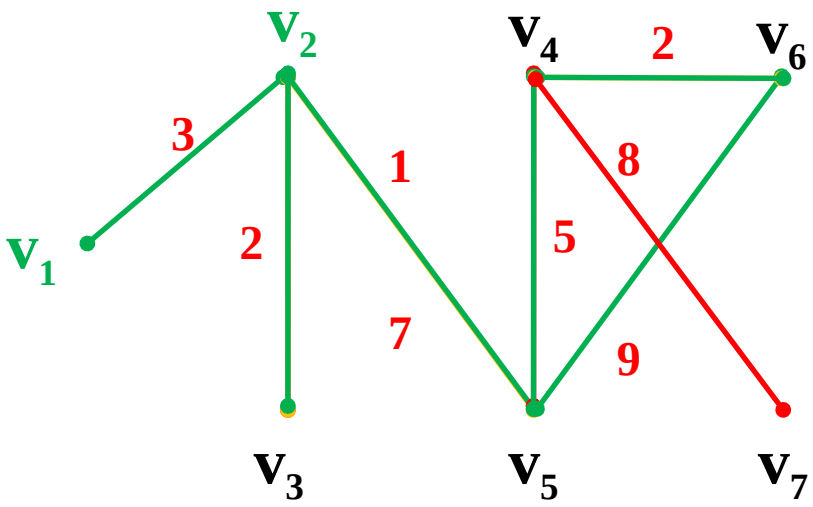
¿P(4) < P(3) + Ω(3,4) ?

¿P(6) < P(3) + Ω(3,6) ?

¿P(7) < P(3) + Ω(3,7) ?

¿∞ < 5 + ∞?

Sea G, y Ω su matriz de pesos



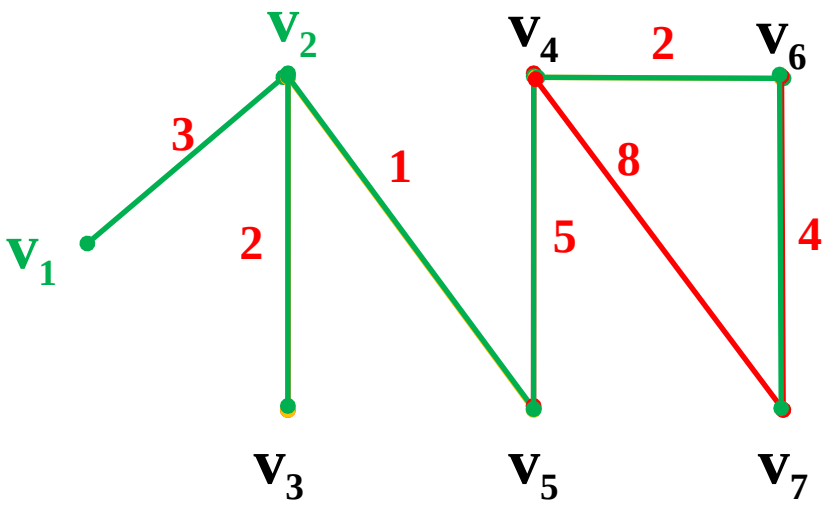
Ω	Paso 1	Paso 2	Paso 3	Paso 4	Paso 5	
v_1	(0 , v1)	*	*	*	*	
v_2	(3 , v1)	(3 , v1)	*	*	*	
v_3	(9 , v1)	(5 , v2)	(5 , v2)	(5 , v2)	*	
v_4	∞	(10 , v2)	(9, v5)	(9, v5)	(9, v5)	
v_5	∞	(4 , v2)	(4 , v2)	*	*	
v_6	∞	∞	(13, v5)	(13, v5)	(11, v4)	
v_7	∞	∞	∞	∞	(17, v4)	

mín{P(6), P(7)}

¿P(6) < P(4) + Ω(4,6) ?
¿13 < 9 + 2?

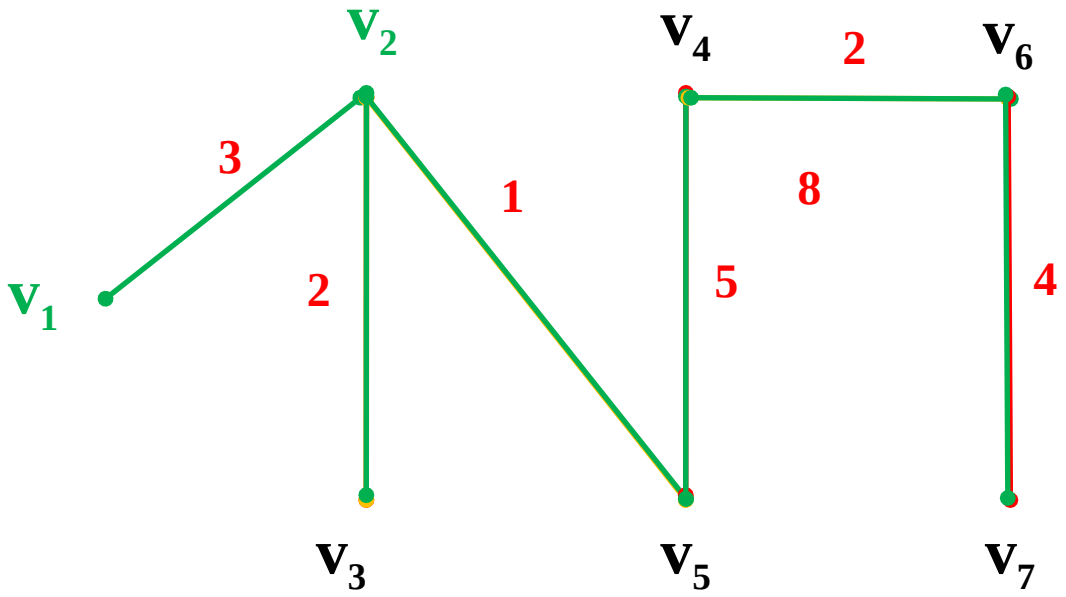
¿P(7) < P(4) + Ω(4,7) ?
∞ < 9 + 8

Sea G, y Ω su matriz de pesos



Ω	Paso 1	Paso 2	Paso 3	Paso 4	Paso 5	Paso 6
v_1	(0 , v1)	*	*	*	*	*
v_2	(3 , v1)	(3 , v1)	*	*	*	*
v_3	(9 , v1)	(5 , v2)	(5 , v2)	(5 , v2)	*	*
v_4	∞	(10 , v2)	(9, v5)	(9, v5)	(9, v5)	*
v_5	∞	(4 , v2)	(4 , v2)	*	*	*
v_6	∞	∞	(13, v5)	(13, v5)	(11, v4)	(11, v4)
v_7	∞	∞	∞	∞	(17, v4)	(15, v6)

¿P(7) < P(6) + Ω(6,7) ?
17 < 11 + 4



Ω	Paso 1	Paso 2	Paso 3	Paso 4	Paso 5	Paso 6	Paso 7
v_1	(0 , v1)	*	*	*	*	*	*
v_2	(3 , v1)	(3 , v1)	*	*	*	*	*
v_3	(9 , v1)	(5 , v2)	(5 , v2)	(5 , v2)	*	*	*
v_4	∞	(10 , v2)	(9, v5)	(9, v5)	(9, v5)	*	*
v_5	∞	(4 , v2)	(4 , v2)	*	*	*	*
v_6	∞	∞	(13, v5)	(13, v5)	(11, v4)	(11, v4)	*
v_7	∞	∞	∞	∞	(17, v4)	(15, v6)	(15, v6)