INTRODUCCIÓN A LA TEORÍA DE GRAFOS: EL PROBLEMA DE LOS CAMINOS MÁS CORTOS

Antonio Hervás Jorge

2017

OBJETIVOS

1

- Vamos a ver un problema interesante.

2

- Aparecen pesos de las aristas de nuevo, y esto caracteriza el problema.

3

- Nos quedaremos con ganas de ver más casos..

DEPARTAMENTO DE MATEMÁTICA APLICADA
UNIVERSIDAD POLITÉCNICA DE VALENCIA

Problema de encontrar el/los camino/s más cortos

desde un vértice X a otro y/o al resto de los vértices del grafo.

entre todos y cada uno de los vértices del grafo.

• El problema de los caminos más cortos con un sólo origen

- **G = (V,E)** un **grafo dirigido**.
- **C = [C(p,q)]$_{n \times n}$** la **matriz de pesos** del grafo G.
- **s** $\in$ **V** un **vértice origen.**

Determinar el **coste del camino más corto** del vértice origen **S** **al resto** de los vértices de **V**.

(El **coste total del camino** es la **suma de los pesos** de los arcos del camino)

Pesos de las aristas del grafo

POSITIVAS

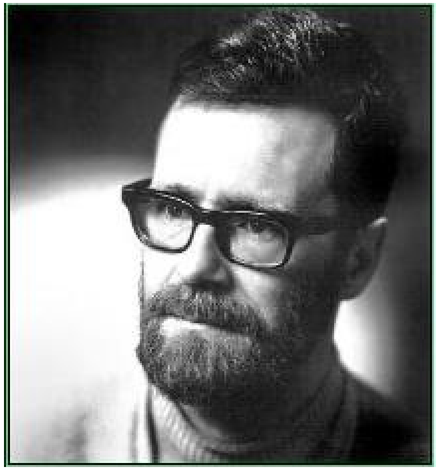
Algoritmo de DIJKSTRA.

NEGATIVAS

Algoritmo de BELLMAN-FORD.

Algoritmo de DIJKSTRA.

- Los pesos **C(p,q)** de todas las aristas **deben ser positivos**.



Edger W. Dijkstra.
1930/2002

Algoritmo de DIJKSTRA.

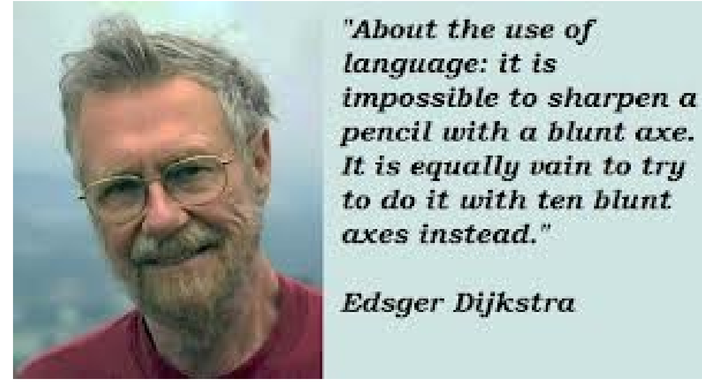
- Los pesos **C(p,q)** de todas las aristas **deben ser positivos**.
- Si una **arista no está en el grafo** le asignamos un **peso** $+\infty$.

Algoritmo de DIJKSTRA.

- Los pesos **C(p,q)** de todas las aristas **deben ser positivos**.
- Si una **arista no está en el grafo** le asignamos un **peso** $+\infty$
- **A cada vértice** se le asignará una **etiqueta** $l(x_i)$.

Algoritmo de DIJKSTRA.

- Los pesos **C(p,q)** de todas las aristas **deben ser positivos**.
- Si una **arista no está en el grafo** le asignamos un **peso + ∞**.
- **A cada vértice** se le asignará una **etiqueta $l(x_i)$**.
  - Representará una **cota superior de la longitud del camino más corto del vértice de partida al vértice x_i**.



Algoritmo de DIJKSTRA.

- Los pesos **C(p,q)** de todas las aristas **deben ser positivos**.

- Si una **arista no está en el grafo** le asignamos un **peso** $+\infty$.

- **A cada vértice** se le asignará una **etiqueta** $\mathbf{l(x_i)}$.
  - Representará una **cota superior de la longitud del camino más corto del vértice de partida al vértice** $\mathbf{x_i}$

.

  - Será variable en principio, pero **en cada iteración se fijará una.**

Algoritmo de DIJKSTRA.

Los pesos **C(p,q)** de todas las aristas **deben ser positivos**.

Si una **arista no está en el grafo** le asignamos un **peso** $+\infty$.

A cada vértice se le asignará una **etiqueta** $l(x_i)$.

- Representará una **cota superior de la longitud del camino más corto del vértice de partida al vértice** x_i.
- Será variable en principio, pero **en cada iteración se fijará una.**

En **cada iteración disminuyen las etiquetas** de los vértices.
(A medida que se alcancen los vértices desde el vértice de partida)

Algoritmo de DIJKSTRA.

- Los pesos **C(p,q)** de todas las aristas **deben ser positivos**.

- Si una **arista no está en el grafo** le asignamos un **peso** $+\infty$.

- **A cada vértice** se le asignará una **etiqueta** $\mathbf{l(x_i)}$.
  - Representará una **cota superior de la longitud del camino más corto del vértice de partida al vértice** x_i.
  - Será variable en principio, pero **en cada iteración se fijará una.**

- En **cada iteración disminuyen las etiquetas** de los vértices.
  (A medida que se alcancen los vértices desde el vértice de partida)

- El **algoritmo acaba cuando se fije la etiqueta del vértice buscado** (o todas las etiquetas sean fijadas)

Algoritmo de DIJKSTRA.

[Paso1] Sea **s** el vértice origen, asignarle una etiqueta que será fija $l(s) = 0$.

$l(x_i) = +\infty \quad \forall\, x_i \in V$ /* variable */

Sea $P = \mathbf{s}$.

[Paso 2] Para todo $x_i \in \Gamma(P)$ con etiqueta variable, actualizar las etiquetas:

$l(x_i) = \min [\, l(x_i) \,,\, l(P) + C(P, x_i) \,]$

[Paso 3] Sea $x_i = \min [\, l(x_j) \,]$, x_j con etiqueta variable.

[Paso 4] Marcar la etiqueta de x_i como fija y hacer $P = x_i$.

[Paso 5]

(1) **Si sólo se desea el camino de s a t.**

Si P = t

entonces

l(P) es la longitud del camino más corto buscado

STOP.

sino ir al PASO2.

(2) **Si se desean los caminos más cortos de s al resto de los vértices.**

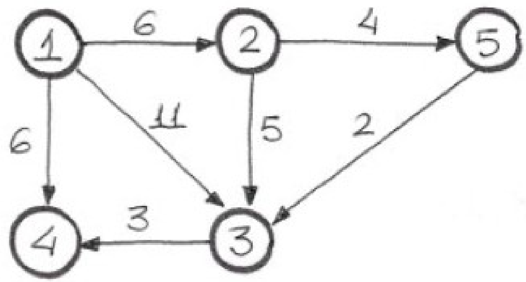
Si todos los vértices tienen etiqueta fija

entonces estas indican las longitudes de los caminos más cortos. STOP.

sino ir al PASO 2.

TEOREMA.

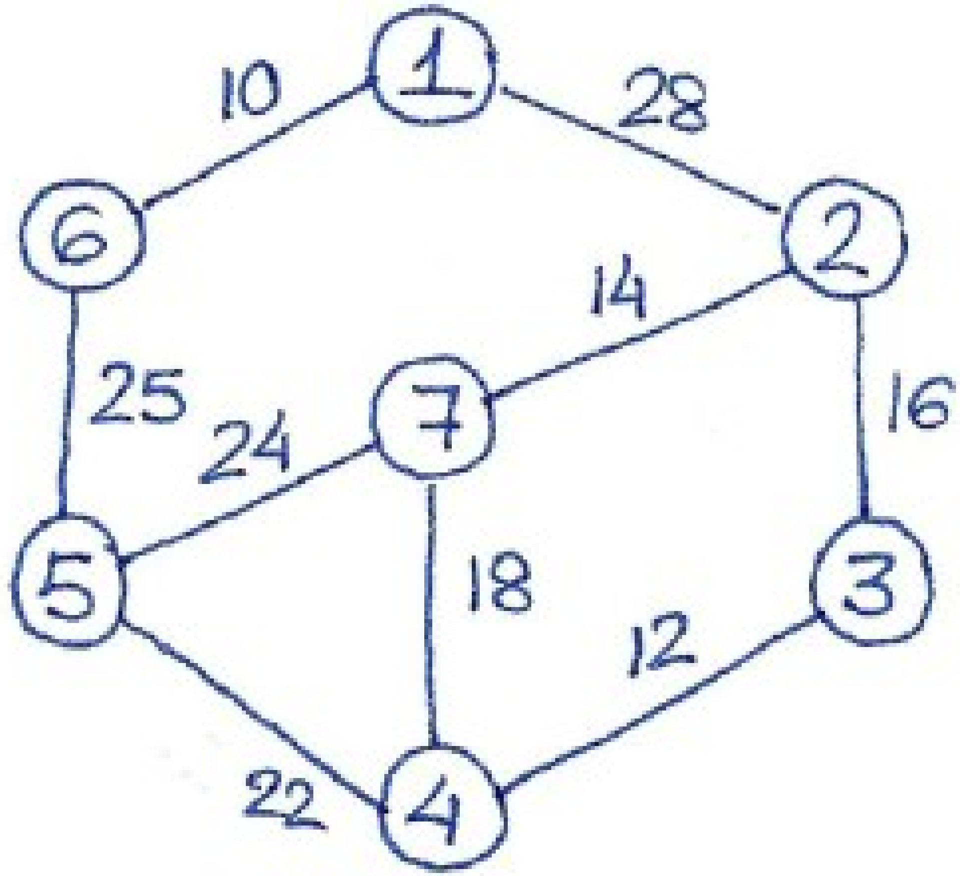
El algoritmo de **DIJKSTRA** suministra los **caminos más cortos de un vértice v al resto de vértices** en un grafo conexo con una matriz de **pesos positivos**.



Caminos más cortos de 1 al resto de vértices.

	ℓ^1	ℓ^2	ℓ^3	ℓ^4	ℓ^5	
1	$\boxed{0}$	—	—	—	—	—
2	∞	6	$\boxed{6}$	—	—	—
3	∞	11	11	11	$\boxed{11}$	—
4	∞	$\boxed{6}$	—	—	—	—
5	∞	∞	∞	$\boxed{10}$	—	—

si tenemos dos iguales elegimos el que más nos convenga.


1
2
3
4
5
6
7
10
28
14
16
25
24
18
12
22

Algoritmo de BELLMAN-FORD.

- Proporciona el **camino más corto entre dos o más vértices** de un **grafo conexo**.

Richard E. Bellman Richard E. Bellman
(26 de agosto 1920 – 19 marzo de 1984)

Lester Randolph Ford Jr.

Algoritmo de BELLMAN-FORD.

- Proporciona el **camino más corto entre dos o más vértices** de un **grafo conexo**.

- Ponderado con **matriz de pesos general** (**positivos o negativos**).

Algoritmo de BELLMAN-FORD.

porciona el **camino más corto entre dos o más vértices**
grafo conexo.

derado con **matriz de pesos general** (**positivos o negativos**).

o deben existir ciclos de peso total negativ

Algoritmo de BELLMAN-FORD.

Proporciona el **camino más corto entre dos o más vértices**
e un **grafo conexo**.

Ponderado con **matriz de pesos general** (**positivos o negativos**).

No deben existir ciclos de peso total negativos.

Se asignan **etiquetas $l^k(x)$** a los vértices.

Algoritmo de BELLMAN-FORD.

- Proporciona el **camino más corto entre dos o más vértices** de un **grafo conexo**.
- Ponderado con **matriz de pesos general** (**positivos o negativos**).
- **No deben existir ciclos de peso total negativos**.

- Se asignan **etiquetas $l^k(x)$** a los vértices.
  - Representa la longitud del **camino más corto del vértice s al vértice x** que contenga **k o menos aristas**.
  - Permanecerán **variables hasta la última iteración**.

Algoritmo de BELLMAN-FORD.

Proporciona el **camino más corto entre dos o más vértices**
e un **grafo conexo**.
Ponderado con **matriz de pesos general** (**positivos o negativos**).
No deben existir ciclos de peso total negativos.
Se asignan **etiquetas $l^k(x)$** a los vértices.
- Representa la longitud del **camino más corto del vértice s al vértice x** que contenga **k o menos aristas**.
- Permanecerán **variables hasta la última iteración**.

- Al **final** de la **iteración k calcularemos la etiqueta k + 1**.

Algoritmo de BELLMAN-FORD.

roporciona el **camino más corto entre dos o más vértices**
un **grafo conexo**.
onderado con **matriz de pesos general** (**positivos o negativos**).
o deben existir ciclos de peso total negativos.
e asignan **etiquetas $l^k(x)$** a los vértices.
- Representa la longitud del **camino más corto del vértice s al vértice x** que contenga **k o menos aristas**.
- Permanecerán **variables hasta la última iteración**.
- Al **final** de la **iteración k calcularemos la etiqueta k + 1**.

l **algoritmo acabará** cuando calcule los **caminos de longitud n - 1**
o los más largos si son de longitud menor).

Algoritmo de BELLMAN-FORD.

[Paso 1] Inicialización.

$S = \Gamma(s)$;

$k = 1$;

Y las etiquetas:

$l^1(s) = 0$;

para los $x_i \in \Gamma(s)$: $l^k(x_i) = C(s, x_i)$

para el resto de los vértices $l^k(x_i) = \infty$

[Paso 2] Para todo $x_i \Gamma \in (S)$ actualizar las etiquetas:

$l^{k+1}(x_i) = \min [\, l^k(x_i) , \min_{x_j \in T_i} \{ l^k(x_j) + C(x_j,x_i) \} \,]$

$T_i = \Gamma^{-1}(x_i) \cap S$

$l^{k+1}(x_i) = l^k(x_i)$ para $x_i \in \Gamma(S)^1$

[Paso 3] Test de finalización.

a) Si $\mathbf{k \leq n\text{-}1}$ y $\forall \mathbf{x_i}$, $\mathbf{l^{k+1}(x_i) = l^k(x_i)}$ $\Rightarrow$ STOP.

Se han obtenido las longitudes de los caminos más cortos y vienen dadas por las etiquetas actuales.

b) Si $\mathbf{k < n\text{-}1}$ y hay **algún** $\mathbf{x_i}$, $\mathbf{l^{k+1}(x_i) \neq l^k(x_i)}$ $\Rightarrow$ PASO 4.

c) Si $\mathbf{k = n\text{-}1}$ y hay **algún** $\mathbf{x_i}$, $\mathbf{l^{k+1} \neq l^k(x_i)}$ $\Rightarrow$

NO HAY SOLUCIÓN.

STOP.

[Paso 4] $S = \{ x_i \ / \ l^{k+1} (x_i) \neq l^k (x_i) \}$

S contiene los vértices cuyo camino más corto es de cardinalidad k+1.

[Paso 5] $k = k+1$; ir al PASO 2.

Camino mínimo entre **todos** los vértices del grafo.

Algoritmo de FLOYD-WARSHALL.

- Actúa **directamente sobre la matriz de pesos**, **sin asignar etiquetas a los vértices**.

Robert W. Floyd
1936/2001

Stephen Warshall.
1935/2006

DEPARTAMENTO DE MATEMÁTICA APLICADA
UNIVERSIDAD POLITÉCNICA DE VALENCIA

Algoritmo de FLOYD-WARSHALL.

- Actúa **directamente sobre la matriz de pesos**, **sin asignar etiquetas a los vértices**.

- Calcula la **longitud de los caminos más cortos entre todos los vértices del grafo**.

Algoritmo de FLOYD-WARSHALL.

• Actúa **directamente sobre la matriz de pesos**, **sin asignar etiquetas a los vértices**.

• Calcula la **longitud de los caminos más cortos entre todos los vértices del grafo**.

• **Detecta los ciclos de peso negativo** (en el case de que existan).

Algoritmo de FLOYD-WARSHALL.

$C = (c_{ij})$ →

- **Matriz de pesos** del grafo.
- Se **actualiza en cada iteración**.

DEPARTAMENTO DE MATEMÁTICA APLICADA
UNIVERSIDAD POLITÉCNICA DE VALENCIA

Algoritmo de FLOYD-WARSHALL.

$C = (c_{ij})$ →

- **Matriz de pesos** del grafo.
- Se **actualiza en cada iteración**.

C^k →

- **Matriz de pesos** de un grafo **donde**:
  - los **vértices** son los **del grafo original**
  - las **aristas** son los **caminos más cortos con k o menos aristas en el grafo de partida**.

Algoritmo de FLOYD-WARSHALL.

- **Al principio**, **la diagonal de la matriz C sólo hay ceros**.
- Si **algún valor de la diagonal se hace negativo**

$\Downarrow$

Ciclo de peso total negativo.

$\Downarrow$

No hay solución.

- En **otro caso** tras **n iteraciones** obtendremos la **matriz**

con las longitudes de los caminos más cortos.

Algoritmo de FLOYD-WARSHALL.

Paso 1] $\mathbf{k = 0}$

Paso 2] $\mathbf{k = k+1}$

Paso3] $\forall\, \mathbf{i} \neq \mathbf{k} \;/\; \mathbf{c_{ik}} \neq \infty$

$\forall\, \mathbf{j} \neq \mathbf{k} \;/\; \mathbf{c_{kj}} \neq \infty$

$$\left.\begin{array}{l} \forall\, \mathbf{i} \neq \mathbf{k} \;/\; \mathbf{c_{ik}} \neq \infty \\ \forall\, \mathbf{j} \neq \mathbf{k} \;/\; \mathbf{c_{kj}} \neq \infty \end{array}\right\} \quad \mathbf{c_{ij} = \min \{ c_{ij},\, c_{ik} + c_{kj} \}}$$

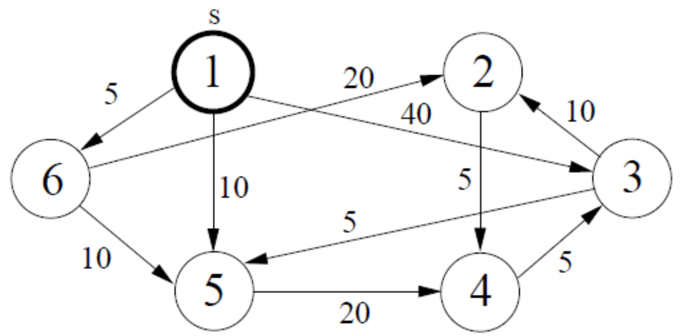
Paso4] a) Si $\exists\, \mathbf{c_{ii} < 0} \Rightarrow$ STOP,circuito de pesos negativos.

b)Si $\forall\, \mathbf{i},\, \mathbf{c_{ii}} \geq \mathbf{0} \wedge \mathbf{k = n} \Rightarrow$ STOP.

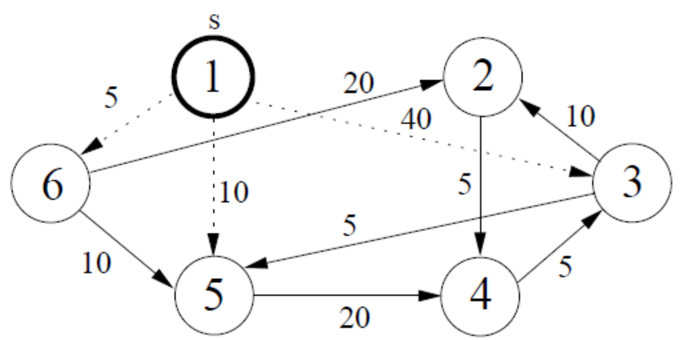
$[c_{ij}]_{n\times n}$ representa las longitudes de los caminos más cortos de x_i a

c) $c_{ii} > 0$, $\forall$ $\mathbf{i, k < n}$ $\Rightarrow$ ir al PASO 2.

Aplicando DIJKSTRA

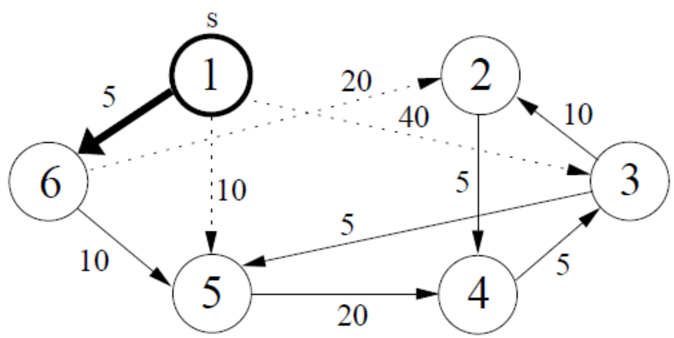


S	Q	u		1	2	3	4	5	6
$\{\}$	$\{1,2,3,4,5,6\}$	$-$	D	0	∞	∞	∞	∞	∞
			P	$NULO$	$NULO$	$NULO$	$NULO$	$NULO$	$NULO$

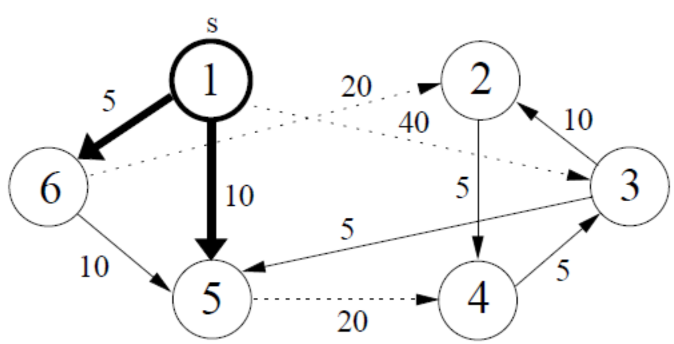


S	Q	u		1	2	3	4	5	6
$\{1\}$	$\{2,3,4,5,6\}$	1	D	0	∞	40	∞	10	5
			P	$NULO$	$NULO$	1	$NULO$	1	1

Aplicando DIJKSTRA

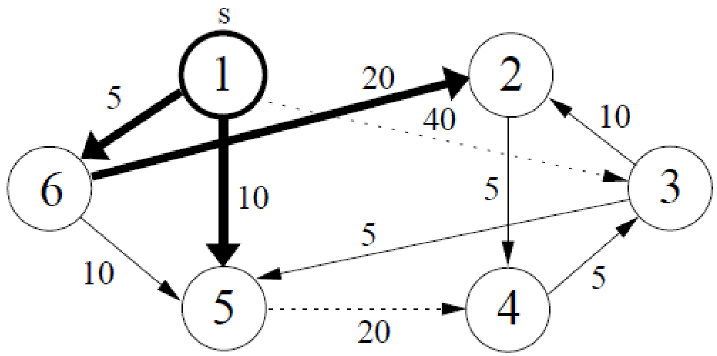


S	Q	u		1	2	3	4	5	6
{1,6}	{2,3,4,5}	6	D	0	25	40	∞	10	5
			P	$NULO$	6	1	$NULO$	1	1

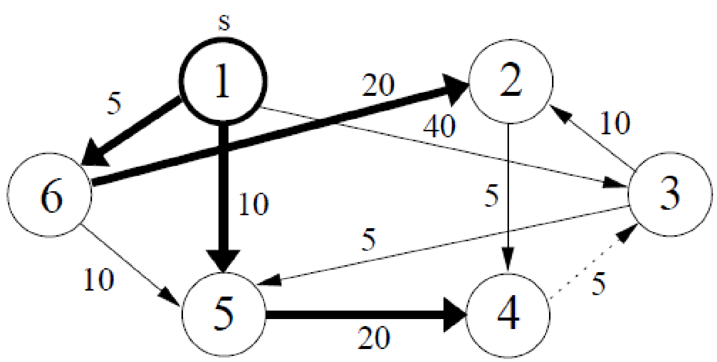


S	Q	u		1	2	3	4	5	6
{1,6,5}	{2,3,4}	5	D	0	25	40	30	10	5
			P	$NULO$	6	1	5	1	1

Aplicando DIJKSTRA

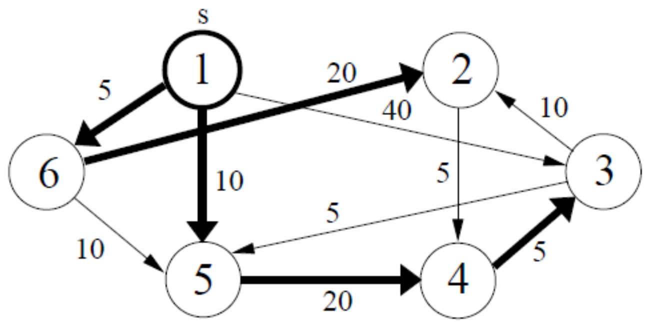


S	Q	u		1	2	3	4	5	6
$\{1,6,5,2\}$	$\{3,4\}$	2	D	0	25	40	30	10	5
			P	$NULO$	6	1	5	1	1



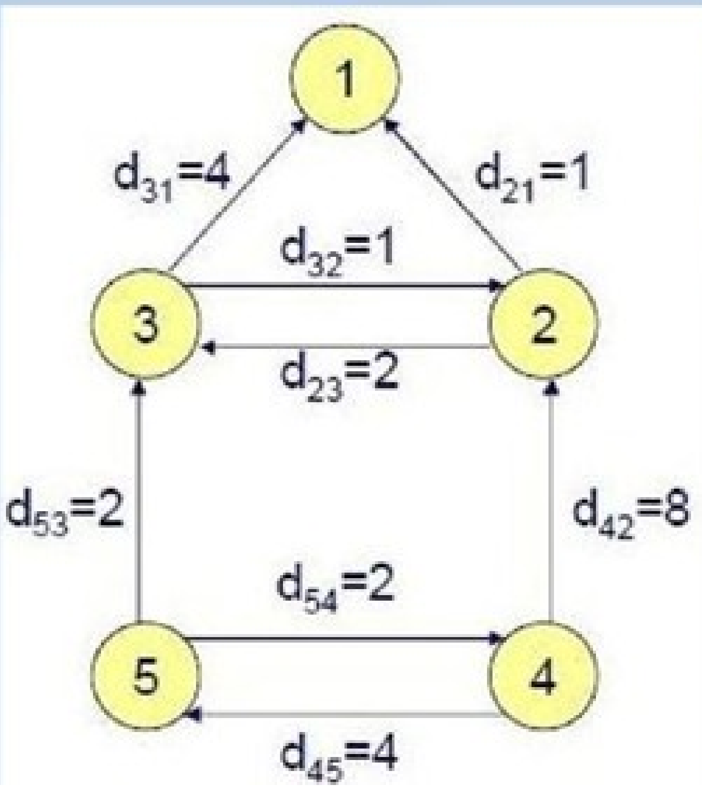
S	Q	u		1	2	3	4	5	6
$\{1,6,5,2,4\}$	$\{3\}$	4	D	0	25	35	30	10	5
			P	$NULO$	6	4	5	1	1

Aplicando DIJKSTRA



S	Q	u		1	2	3	4	5	6
{1,6,5,2,4,3}	{}	3	D	0	25	35	30	10	5
			P	$NULO$	6	4	5	1	1

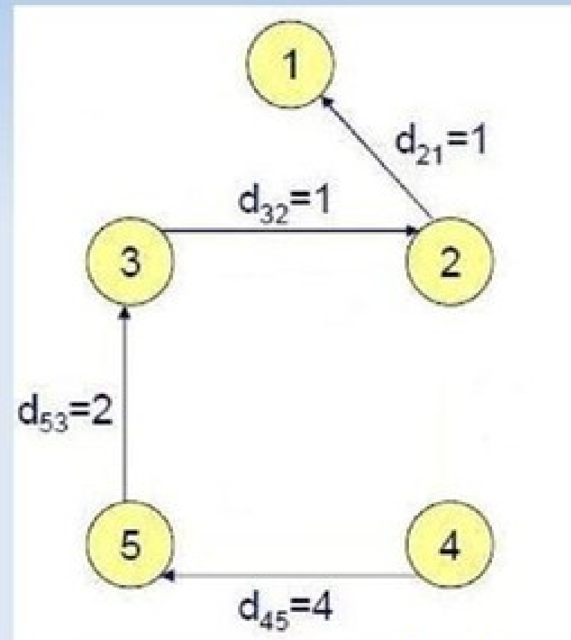
Aplicando BELLMAN-FORD



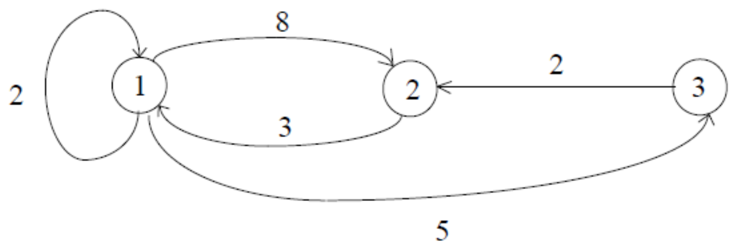
destino / *origen* \	1	2	3	4	5
1	0	∞	∞	∞	∞
2	1	0	2	∞	∞
3	4	1	0	∞	∞
4	∞	8	∞	0	4
5	∞	∞	2	2	0

Aplicando BELLMAN-FORD

	D_1	D_2		D_3		D_4		D_5	
n		$d_{21}+D_1^*$	$d_{23}+D_3^*$	$d_{31}+D_1^*$	$d_{32}+D_2^*$	$d_{41}+D_1^*$	$d_{43}+D_3^*$	$d_{51}+D_1^*$	$d_{54}+D_4^*$
0	0	∞		∞		∞		∞	
1	0	1+0 1	2+∞ ∞	4+0 4	1+ ∞ ∞	4+ ∞ ∞	8+ ∞ ∞	2+ ∞ ∞	2+ ∞ ∞
2	0	1+0 1	2+4 6	4+0 4	1+1 2	4+ ∞ ∞	8+1 9	2+4 6	2+ ∞ ∞
3	0	1+0 1	2+2 4	4+0 4	1+1 2	4+6 10	8+1 9	2+2 4	2+9 11
4	0	1+0 1	2+2 4	4+0 4	1+1 2	4+4 8	8+1 9	2+2 4	2+9 11
5	0	1+0 1	2+2 4	4+0 4	1+1 2	4+4 8	8+1 9	2+2 4	2+8 10



Algoritmo de Floyd: Ejemplo



A_0	1	2	3
1	0	8	5
2	3	0	α
3	α	2	0

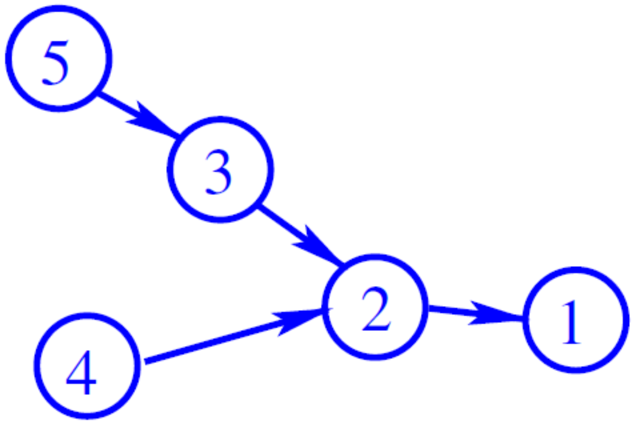
A_1	1	2	3
1	0	8	5
2	3	0	8
3	α	2	0

A_2	1	2	3
1	0	8	5
2	3	0	8
3	5	2	0

A_3	1	2	3
1	0	7	5
2	3	0	8
3	5	2	0

Aplicar Warshall a los dos grafos anteriores.

Aplicando ORDEN TOPOLOGICO



mR

5 3 4 2 1

mL

5 4 2 3 1