Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

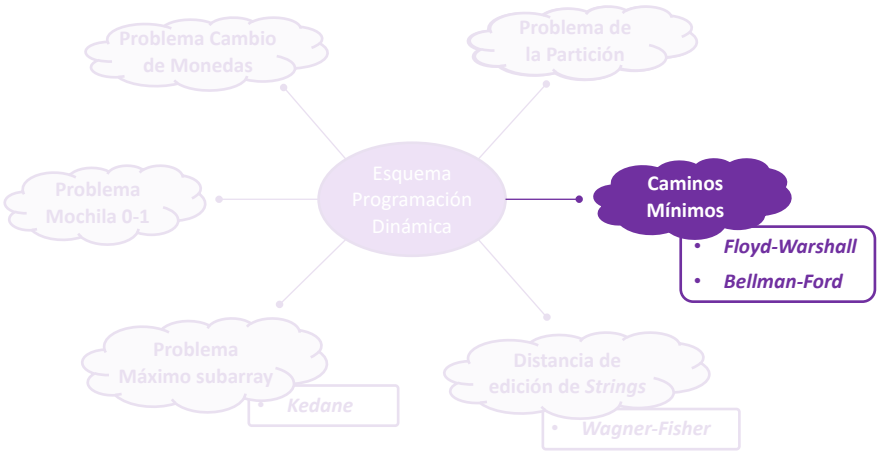
Tema 16. Programación Dinámica en Grafos

Algorítmica y Complejidad

1

Introducción

Algunos problemas planteados



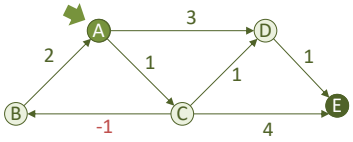
2

Introducción

Camino más corto en un Grafo

Problema *Encontrar el camino más corto entre dos vértices.*

Camino más corto entre **A** y **E**



1. Introducción

1 2 3

3

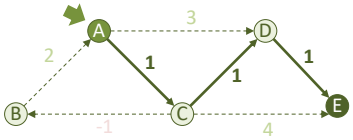
Introducción

Camino más corto en un Grafo

Problema *Encontrar el camino más corto entre dos vértices.*

Camino más corto entre **A** y **E**

ACDE



1. Introducción

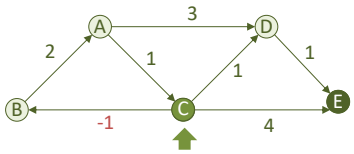
1 2 3

4

Introducción

Camino más corto en un Grafo

Problema *Encontrar el camino más corto entre dos vértices.*



Camino más corto entre **A** y **E**

ACDE

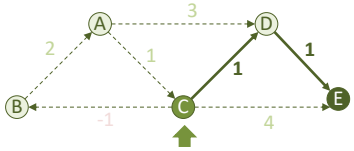
Camino más corto entre **C** y **E**

5

Introducción

Camino más corto en un Grafo

Problema *Encontrar el camino más corto entre dos vértices.*



Camino más corto entre **A** y **E**

ACDE

Camino más corto entre **C** y **E**

CDE

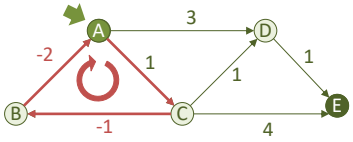
El problema exhibe **subestructura óptima**

6

Introducción

Camino más corto en un Grafo

Problema — *Encontrar el camino más corto entre dos vértices.*



Ciclo Negativo

-2-1+1 = -2

Camino más corto entre **A** y **E**

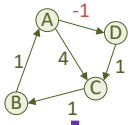
ACB ACB ACB …. ACB ACDE

No existe el camino mínimo

7

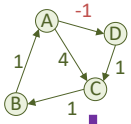
Introducción

Camino más corto en un Grafo



Camino Mínimo — *Algoritmo de Floyd-Warshall*

Origen	Destino	Camino
A	B	ADCB
A	C	ADC
…	…	…
D	C	DC



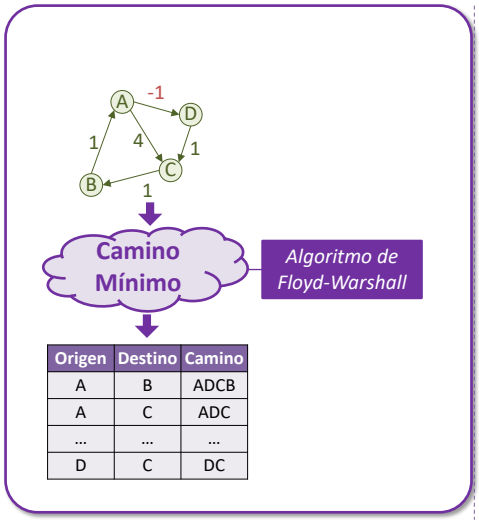
Camino Mínimo — *Algoritmo de Bellman-Ford*

Destino	Camino
B	ADCB
C	ADC
D	AD

8

Algoritmo Floyd-Warshall

Camino más corto en un Grafo

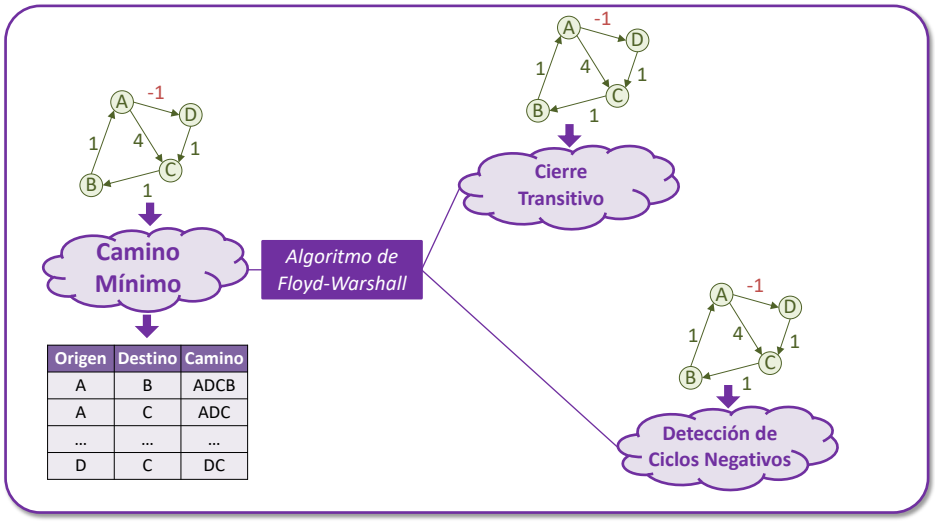


Origen	Destino	Camino
A	B	ADCB
A	C	ADC
...	...	...
D	C	DC

9

Algoritmo Floyd-Warshall

Camino más corto en un Grafo

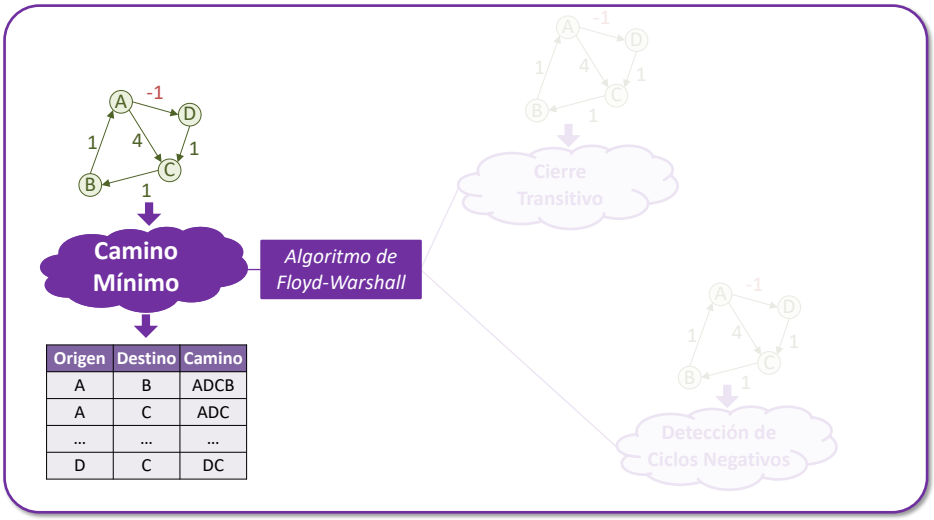


Origen	Destino	Camino
A	B	ADCB
A	C	ADC
...	...	...
D	C	DC

10

Algoritmo Floyd-Warshall

Camino más corto en un Grafo

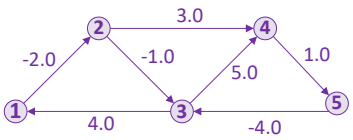


Origen	Destino	Camino
A	B	ADCB
A	C	ADC
...	...	...
D	C	DC

11

Algoritmo Floyd-Warshall

Ecuación de Recurrencia

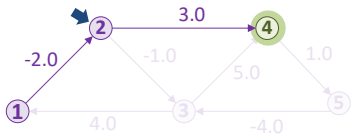


d(i,j,k): Distancia mínima del vértice **i** al vértice **j** con vértices intermedios en el conjunto {**1,…,k**}

12

Algoritmo Floyd-Warshall

Ecuación de Recurrencia

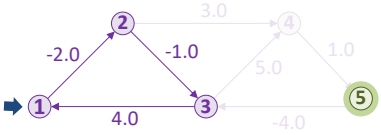


d(2,4,2): Distancia mínima del vértice **2** al vértice **4** con vértices intermedios en el conjunto {**1,2**}

13

Algoritmo Floyd-Warshall

Ecuación de Recurrencia

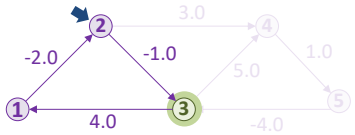


d(1,5,3): Distancia mínima del vértice **1** al vértice **5** con vértices intermedios en el conjunto {**1,2,3**}

14

Algoritmo Floyd-Warshall

Ecuación de Recurrencia

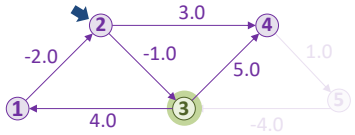


d(2,3,2): Distancia mínima del vértice **2** al vértice **3** con vértices intermedios en el conjunto {**1,2**}

15

Algoritmo Floyd-Warshall

Ecuación de Recurrencia

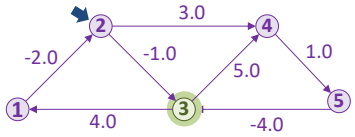


d(2,3,4): Distancia mínima del vértice **2** al vértice **3** con vértices intermedios en el conjunto {**1,2,3,4**}

16

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



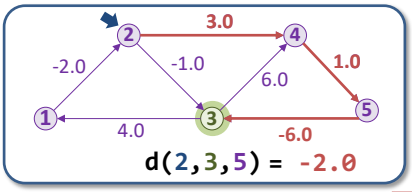
d(2,3,5): Distancia mínima del vértice **2** al vértice **3** con vértices intermedios en el conjunto {**1,2,3,4,5**}

Problema de Caminos Mínimos Original!!

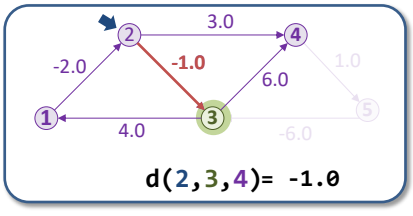
17

Algoritmo Floyd-Warshall

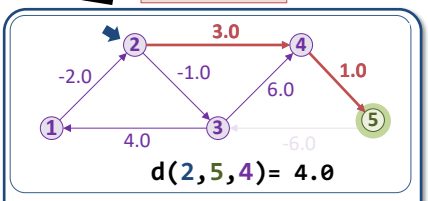
Ecuación de Recurrencia

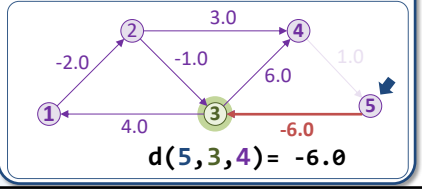


-1.0 NO pasa por **5**



SÍ pasa por **5** **-2.0**

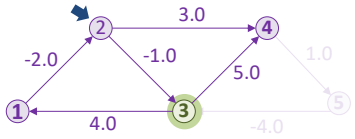




18

Algoritmo Floyd-Warshall

Ecuación de Recurrencia. Caso Recursivo



d(i,j,k): Distancia mínima del vértice **i** al vértice **j** con vértices intermedios en el conjunto {**1,…,k**}

Caso Recursivo: *k>0*

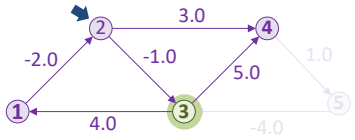
$$d(i,j,k) = min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\}$$

El camino mínimo **no** pasa por el vértice ***k***

19

Algoritmo Floyd-Warshall

Ecuación de Recurrencia. Caso Recursivo



d(i,j,k): Distancia mínima del vértice **i** al vértice **j** con vértices intermedios en el conjunto {**1,…,k**}

Caso Recursivo: *k>0*

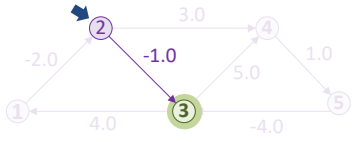
$$d(i,j,k) = min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\}$$

El camino mínimo **sí** pasa por el vértice ***k***

20

Algoritmo Floyd-Warshall

Ecuación de Recurrencia. Caso Base



`d(2,3,0)=-1.0`

`d(i,j,0)`: Distancia mínima del vértice `i` al vértice `j` sin usar vértices intermedios.

Caso Base: k=0

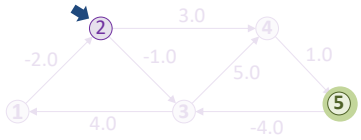
$$d(i,j,0) = w_{ij}$$

peso de la arista que va del vértice `i` al vértice `j`

21

Algoritmo Floyd-Warshall

Ecuación de Recurrencia. Caso Base



`d(2,5,0)=∞`

`d(i,j,0)`: Distancia mínima del vértice `i` al vértice `j` sin usar vértices intermedios.

Caso Base: k=0

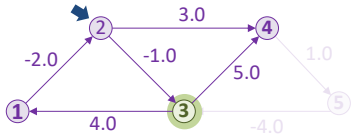
$$d(i,j,0) = w_{ij}$$

peso de la arista que va del vértice `i` al vértice `j`

22

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



d(i,j,k): Distancia mínima del vértice **i** al vértice **j** con vértices intermedios en el conjunto {**1,…,k**}

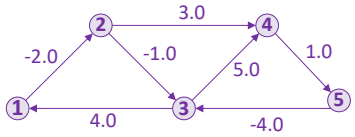
$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

Solo necesitamos recordar $d(i,j,k-1)$ para conseguir $d(i,j,k)$

23

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



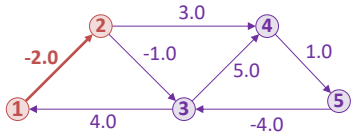
$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,0)	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	∞	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

24

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



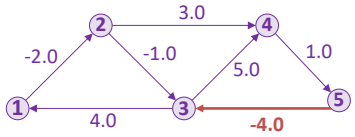
$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,0)	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	∞	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

25

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



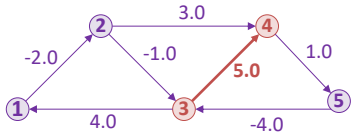
$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,0)	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	∞	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

26

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



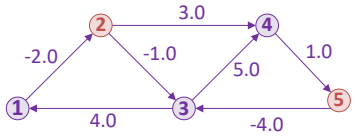
$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,0)	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	∞	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

27

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

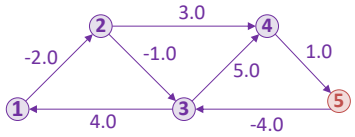
d(i,j,0)	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	∞	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

→ No hay arista

28

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

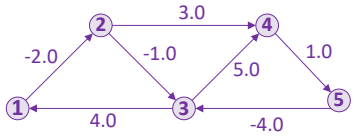
d(i,j,0)	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	∞	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

→ El mismo vértice

29

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,0)	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	∞	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

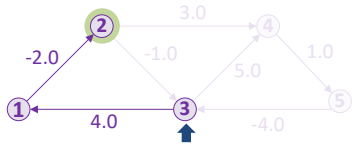
➡

d(i,j,1)	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	**2.0**	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

30

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,0)

	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	∞	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

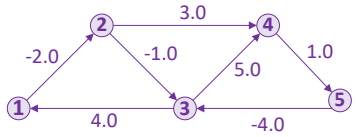
d(i,j,1)

	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

31

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,2)

	1	2	3	4	5
1	0	-2.0	**-3.0**	**1.0**	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

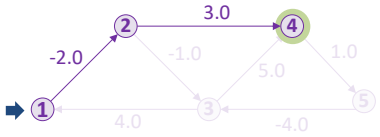
d(i,j,1)

	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

32

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,2)

	1	2	3	4	5
1	0	-2.0	**-3.0**	**1.0**	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

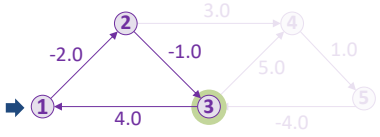
d(i,j,1)

	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

33

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,2)

	1	2	3	4	5
1	0	-2.0	**-3.0**	**1.0**	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

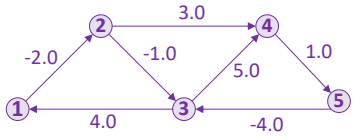
d(i,j,1)

	1	2	3	4	5
1	0	-2.0	∞	∞	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

34

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,2)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

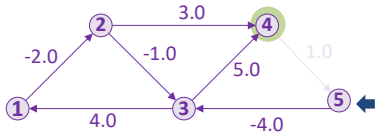
d(i,j,3)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	**3.0**	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	**0.0**	**2.0**	-4.0	**1.0**	0

35

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,2)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	**5.0**	∞
4	∞	∞	∞	0	1.0
5	∞	∞	**-4.0**	∞	0

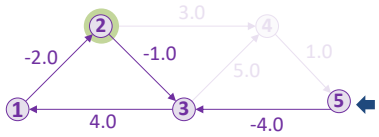
d(i,j,3)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	**3.0**	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	**0.0**	**-2.0**	-4.0	**1.0**	0

36

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,2)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

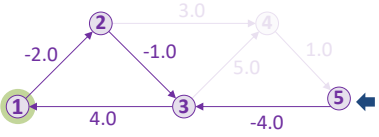
d(i,j,3)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	**3.0**	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	**0.0**	**-2.0**	-4.0	**1.0**	0

37

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,2)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

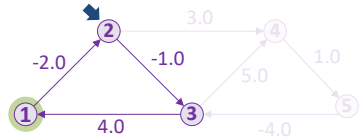
d(i,j,3)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	**3.0**	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	**0.0**	**-2.0**	-4.0	**1.0**	0

38

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,2)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

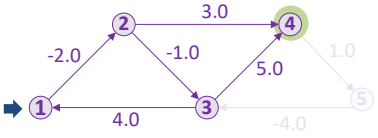
d(i,j,3)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	3.0	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	**0.0**	-2.0	-4.0	**1.0**	0

39

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,2)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	∞	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	∞	∞	-4.0	∞	0

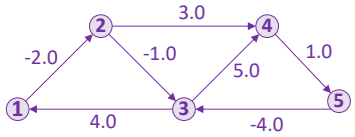
d(i,j,3)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	**3.0**	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	**0.0**	**-2.0**	-4.0	**1.0**	0

40

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,4)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	**2.0**
2	3.0	0	-1.0	3.0	**4.0**
3	4.0	2.0	0	5.0	**6.0**
4	∞	∞	∞	0	1.0
5	0.0	-2.0	-4.0	1.0	0

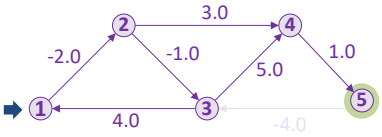
d(i,j,3)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	3.0	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	0.0	-2.0	-4.0	1.0	0

41

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,4)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	**2.0**
2	3.0	0	-1.0	3.0	**4.0**
3	4.0	2.0	0	5.0	**6.0**
4	∞	∞	∞	0	1.0
5	0.0	-2.0	-4.0	1.0	0

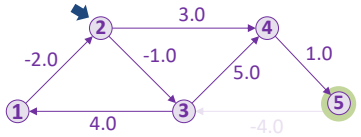
d(i,j,3)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	3.0	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	0.0	-2.0	-4.0	1.0	0

42

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,4)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	**2.0**
2	3.0	0	-1.0	3.0	**4.0**
3	4.0	2.0	0	5.0	**6.0**
4	∞	∞	∞	0	1.0
5	0.0	-2.0	-4.0	1.0	0

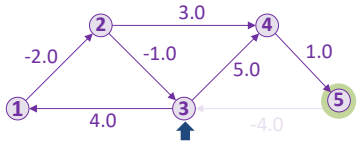
d(i,j,3)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	3.0	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	0.0	-2.0	-4.0	1.0	0

43

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,4)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	**2.0**
2	3.0	0	-1.0	3.0	**4.0**
3	4.0	2.0	0	5.0	**6.0**
4	∞	∞	∞	0	1.0
5	0.0	-2.0	-4.0	1.0	0

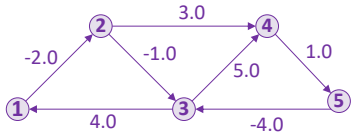
d(i,j,3)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	∞
2	3.0	0	-1.0	3.0	∞
3	4.0	2.0	0	5.0	∞
4	∞	∞	∞	0	1.0
5	0.0	-2.0	-4.0	1.0	0

44

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,4)	1	2	3	4	5
1	0	-2.0	-3.0	1.0	2.0
2	3.0	0	-1.0	3.0	4.0
3	4.0	2.0	0	5.0	6.0
4	∞	∞	∞	0	1.0
5	0.0	-2.0	-4.0	1.0	0

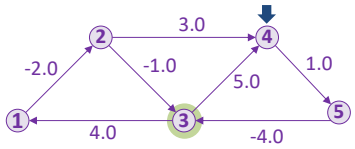
➡

d(i,j,5)	1	2	3	4	5
1	0	-2.0	-3.0	1.0	2.0
2	3.0	0	-1.0	3.0	4.0
3	4.0	2.0	0	5.0	6.0
4	**1.0**	**-1.0**	**-3.0**	0	1.0
5	0.0	-2.0	-4.0	1.0	0

45

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,4)	1	2	3	4	5
1	0	-2.0	-3.0	1.0	2.0
2	3.0	0	-1.0	3.0	4.0
3	4.0	2.0	0	5.0	6.0
4	∞	∞	∞	0	1.0
5	0.0	-2.0	-4.0	1.0	0

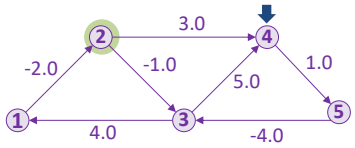
➡

d(i,j,5)	1	2	3	4	5
1	0	-2.0	-3.0	1.0	2.0
2	3.0	0	-1.0	3.0	4.0
3	4.0	2.0	0	5.0	6.0
4	**1.0**	**-1.0**	**-3.0**	0	1.0
5	0.0	-2.0	-4.0	1.0	0

46

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,4)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	2.0
2	3.0	0	-1.0	3.0	4.0
3	4.0	2.0	0	5.0	6.0
4	∞	∞	∞	0	1.0
5	0.0	-2.0	-4.0	1.0	0

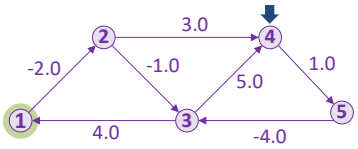
d(i,j,5)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	2.0
2	3.0	0	-1.0	3.0	4.0
3	4.0	2.0	0	5.0	6.0
4	**1.0**	**-1.0**	**-3.0**	0	1.0
5	0.0	-2.0	-4.0	1.0	0

47

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,4)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	2.0
2	3.0	0	-1.0	3.0	4.0
3	4.0	2.0	0	5.0	6.0
4	∞	∞	∞	0	1.0
5	**0.0**	-2.0	-4.0	1.0	0

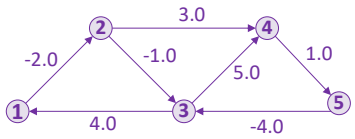
d(i,j,5)

	1	2	3	4	5
1	0	-2.0	-3.0	1.0	2.0
2	3.0	0	-1.0	3.0	4.0
3	4.0	2.0	0	5.0	6.0
4	**1.0**	**-1.0**	**-3.0**	0	1.0
5	0.0	-2.0	-4.0	1.0	0

48

Algoritmo Floyd-Warshall

Ecuación de Recurrencia



	1	2	3	4	5
1	0	-2.0	-3.0	1.0	2.0
2	3.0	0	-1.0	3.0	4.0
3	4.0	2.0	0	5.0	6.0
4	1.0	-1.0	-3.0	0	1.0
5	0.0	-2.0	-4.0	1.0	0

49

Algoritmo Floyd-Warshall

Relación de recurrencia

Actividad 16.1. Implementa el algoritmo (de Floyd –Warshall) que determina la distancia mínima entre cada par de vértices de un grafo.

```
float[][] distanciasMinimas(float[][] grafo){
 float[][] d = new float[grafo.length][grafo.length];
  for (int i=0; i<grafo.length; i++)
    for (int j=0; j<grafo.length; j++)
       if (i!=j) d[i][j]=grafo[i][j];
       else d[i][j]=0;
  for (int k=0; k<grafo.length; k++)
    for (int i=0; i<grafo.length; i++)
      for (int j=0; j<grafo.length; j++)
        if (d[i][k] + d[k][j]<d[i][j])
            d[i][j] = d[i][k] + d[k][j];

  return d;
}
```


50

Algoritmo Floyd-Warshall

Relación de recurrencia

Actividad 16.1. Implementa el algoritmo (de Floyd –Warshall) que determina la distancia mínima entre cada par de vértices de un grafo.

```
float[][] distanciasMinimas(float[][] grafo){
 float[][] d = new float[grafo.length][grafo.length];
  for (int i=0; i<grafo.length; i++)
    for (int j=0; j<grafo.length; j++)
       if (i!=j) d[i][j]=grafo[i][j];
       else d[i][j]=0;
  for (int k=0; k<grafo.length; k++)
    for (int i=0; i<grafo.length; i++)
      for (int j=0; j<grafo.length; j++)
        if (d[i][k] + d[k][j]<d[i][j])
            d[i][j] = d[i][k] + d[k][j];

  return d;
}
```

Caso Base:

$$d(i,j,0) = w_{ij}$$

51

Algoritmo Floyd-Warshall

Relación de recurrencia

Actividad 16.1. Implementa el algoritmo (de Floyd –Warshall) que determina la distancia mínima entre cada par de vértices de un grafo.

```
float[][] distanciasMinimas(float[][] grafo){
 float[][] d = new float[grafo.length][grafo.length];
  for (int i=0; i<grafo.length; i++)
    for (int j=0; j<grafo.length; j++)
       if (i!=j) d[i][j]=grafo[i][j];
       else d[i][j]=0;
  for (int k=0; k<grafo.length; k++)
    for (int i=0; i<grafo.length; i++)
      for (int j=0; j<grafo.length; j++)
        if (d[i][k] + d[k][j]<d[i][j])
            d[i][j] = d[i][k] + d[k][j];

  return d;
}
```

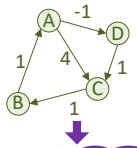
Caso Recursivo:

$$d(i,j,k) = min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\}$$

52

Algoritmo Floyd-Warshall

Camino más corto en un Grafo



Camino Mínimo

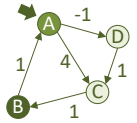
Algoritmo de Floyd-Warshall

d(*i*,*j*)	A	B	C	D
A	0	1.0	0	-1.0
B	1.0	0	1.0	0.0
C	2.0	1.0	0	1.0
D	3.0	2.0	1.0	0

53

Algoritmo Floyd-Warshall

Camino más corto en un Grafo



Camino Mínimo

Algoritmo de Floyd-Warshall

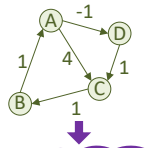
¿Cuál es el camino mínimo?

d(*i*,*j*)	A	B	C	D
A	0	1.0	0	-1.0
B	1.0	0	1.0	0.0
C	2.0	1.0	0	1.0
D	3.0	2.0	1.0	0

54

Algoritmo Floyd-Warshall

Camino más corto en un Grafo



Camino Mínimo — *Algoritmo de Floyd-Warshall*

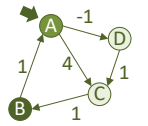
d(*i*,*j*)	A	B	C	D
A	0	1.0	0	-1.0
B	1.0	0	1.0	0.0
C	2.0	1.0	0	1.0
D	3.0	2.0	1.0	0

p(*i*,*j*)	A	B	C	D
A	-	C	D	A
B	B	-	D	A
C	B	C	-	A
D	B	C	D	-

55

Algoritmo Floyd-Warshall

Camino más corto en un Grafo



Camino Mínimo — *Algoritmo de Floyd-Warshall*

¿Cuál es el camino mínimo?

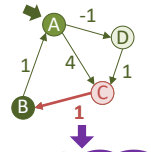
d(*i*,*j*)	A	B	C	D
A	0	1.0	0	-1.0
B	1.0	0	1.0	0.0
C	2.0	1.0	0	1.0
D	3.0	2.0	1.0	0

p(*i*,*j*)	A	B	C	D
A	-	C	D	A
B	B	-	D	A
C	B	C	-	A
D	B	C	D	-

56

Algoritmo Floyd-Warshall

Camino más corto en un Grafo



Camino Mínimo — *Algoritmo de Floyd-Warshall*

d(*i*,*j*)	A	B	C	D
A	0	1.0	0	-1.0
B	1.0	0	1.0	0.0
C	2.0	1.0	0	1.0
D	3.0	2.0	1.0	0

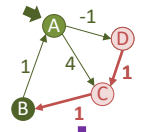
p(*i*,*j*)	A	B	C	D
A	-	C	D	A
B	B	-	D	A
C	B	C	-	A
D	B	C	D	-

Camino Mínimo: ...CB

57

Algoritmo Floyd-Warshall

Camino más corto en un Grafo



Camino Mínimo — *Algoritmo de Floyd-Warshall*

d(*i*,*j*)	A	B	C	D
A	0	1.0	0	-1.0
B	1.0	0	1.0	0.0
C	2.0	1.0	0	1.0
D	3.0	2.0	1.0	0

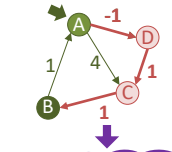
p(*i*,*j*)	A	B	C	D
A	-	C	D	A
B	B	-	D	A
C	B	C	-	A
D	B	C	D	-

Camino Mínimo: ...DCB

58

Algoritmo Floyd-Warshall

Camino más corto en un Grafo



Camino Mínimo — *Algoritmo de Floyd-Warshall*

d(i,j)	A	B	C	D
A	0	1.0	0	-1.0
B	1.0	0	1.0	0.0
C	2.0	1.0	0	1.0
D	3.0	2.0	1.0	0

p(i,j)	A	B	C	D
A	-	C	D	A
B	B	-	D	A
C	B	C	-	A
D	B	C	D	-

Camino Mínimo: ADCB

59

Algoritmo Floyd-Warshall

Camino más corto en un Grafo

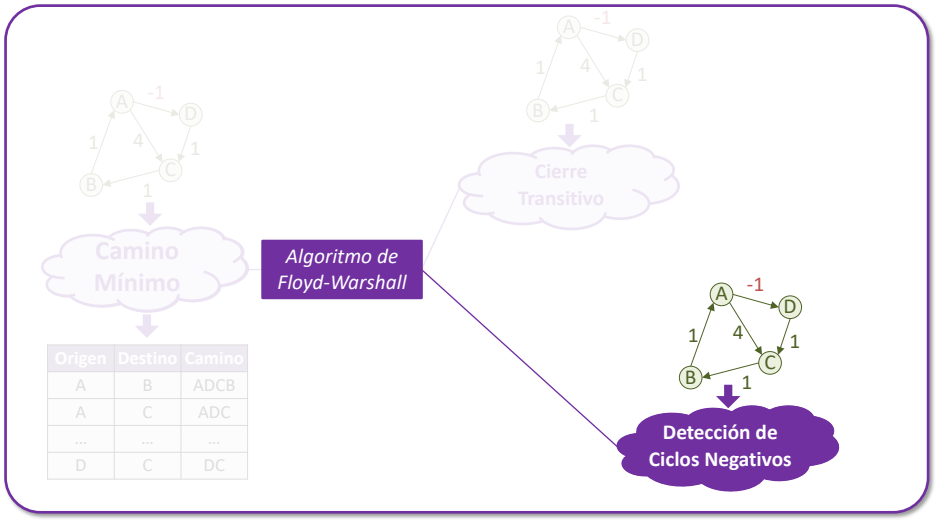
Actividad 16.2. Implementa el algoritmo (de Floyd –Warshall) que determina la distancia y el camino mínimo entre cada par de vértices de un grafo.

```
float[][] distanciasMinimas(float[][] grafo, int[][] predecesor){
 float[][] d = new float[grafo.length][grafo.length];
 for (int i=0; i<grafo.length; i++)
    for (int j=0; j<grafo.length; j++) {
       if (i!=j) d[i][j]=grafo[i][j];
       else d[i][j]=0;
       if (grafo[i][j] < Float.MAX_VALUE) predecesor[i][j]= i;
    }
  for (int k=0; k<grafo.length; k++)
    for (int i=0; i<grafo.length; i++)
      for (int j=0; j<grafo.length; j++)
        if (d[i][k] + d[k][j]<d[i][j]) {
            d[i][j] = d[i][k] + d[k][j];
            predecesor[i][j]= predecesor[k][j];
        }
  return d;
}
```


60

Algoritmo Floyd-Warshall

Ciclos Negativos en un Grafo

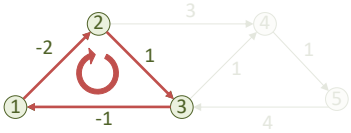


61

Algoritmo Floyd-Warshall

Ciclos Negativos en un Grafo

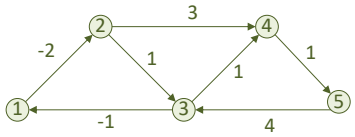
Ciclo Negativo
-2+1-1 = -2



62

Algoritmo Floyd-Warshall

Ciclos Negativos en un Grafo



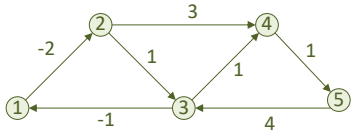
$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,0)	1	2	3	4	5
1	0.0	-2.0	∞	∞	∞
2	∞	0.0	1.0	3.0	∞
3	-1.0	∞	0.0	1.0	∞
4	∞	∞	∞	0.0	1.0
5	∞	∞	4.0	∞	0.0

63

Algoritmo Floyd-Warshall

Ciclos Negativos en un Grafo



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,0)	1	2	3	4	5
1	0.0	-2.0	∞	∞	∞
2	∞	0.0	1.0	3.0	∞
3	-1.0	∞	0.0	1.0	∞
4	∞	∞	∞	0.0	1.0
5	∞	∞	4.0	∞	0.0

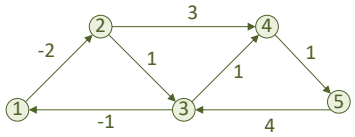
➡

d(i,j,1)	1	2	3	4	5
1	0.0	-2.0	∞	∞	∞
2	∞	0.0	1.0	3.0	∞
3	-1.0	**-3.0**	0.0	1.0	∞
4	∞	∞	∞	0.0	1.0
5	∞	∞	4.0	∞	0.0

64

Algoritmo Floyd-Warshall

Ciclos Negativos en un Grafo



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,2)

	1	2	3	4	5
1	0.0	-2.0	∞	∞	∞
2	∞	0.0	1.0	3.0	∞
3	-1.0	-3.0	**-2.0**	**0.0**	∞
4	∞	∞	∞	0.0	1.0
5	∞	∞	4.0	∞	0.0

d(i,j,1)

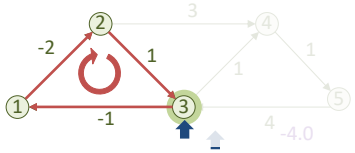
	1	2	3	4	5
1	0.0	-2.0	∞	∞	∞
2	∞	0.0	1.0	3.0	∞
3	-1.0	-3.0	0.0	1.0	∞
4	∞	∞	∞	0.0	1.0
5	∞	∞	4.0	∞	0.0

65

Algoritmo Floyd-Warshall

Ciclos Negativos en un Grafo

Ciclo Negativo
-2+1-1 = -2



$$d(i,j,k) = \begin{cases} min\{d(i,j,k-1), d(i,k,k-1) + d(k,j,k-1)\} & si\ k > 0 \\ w_{ij} & k = 0 \end{cases}$$

d(i,j,2)

	1	2	3	4	5
1	0.0	-2.0	∞	∞	∞
2	∞	0.0	1.0	3.0	∞
3	-1.0	-3.0	**-2.0**	**0.0**	∞
4	∞	∞	∞	0.0	1.0
5	∞	∞	4.0	∞	0.0

d(i,j,1)

	1	2	3	4	5
1	0.0	-2.0	∞	∞	∞
2	∞	0.0	**1.0**	3.0	∞
3	-1.0	**-3.0**	**0.0**	1.0	∞
4	∞	∞	∞	0.0	1.0
5	∞	∞	4.0	∞	0.0

66

Algoritmo Floyd-Warshall

Ciclos Negativos en un Grafo

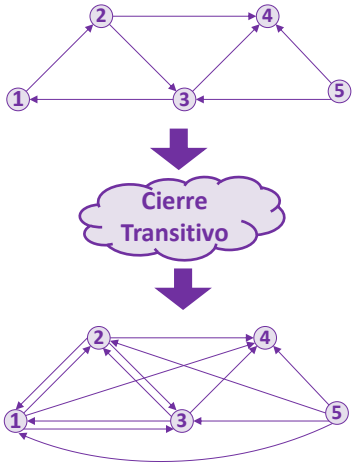
Actividad 16.3. **Utiliza el algoritmo de Floyd –Warshall para detectar ciclos negativos en un grafo.**

```
boolean hayCiclosNegativos(float[][] grafo){
 boolean hayCicloNegativo = false;
 float[][] d = new float[grafo.length][grafo.length];
 for (int i=0; i<grafo.length; i++)
    for (int j=0; j<grafo.length; j++)
       if (i!=j) d[i][j]=grafo[i][j];   else d[i][j]=0;
  int k=0;
  while (k<grafo.length && !hayCicloNegativo){
    for (int i=0; i<grafo.length; i++) {
      for (int j=0; j<grafo.length; j++)
        if (d[i][k] + d[k][j]<d[i][j])
            d[i][j] = d[i][k] + d[k][j];
     if (d[i][i]<0) hayCicloNegativo=true;
      }
     k++;
  }
 return hayCicloNegativo;
}
```


67

Algoritmo Floyd-Warshall

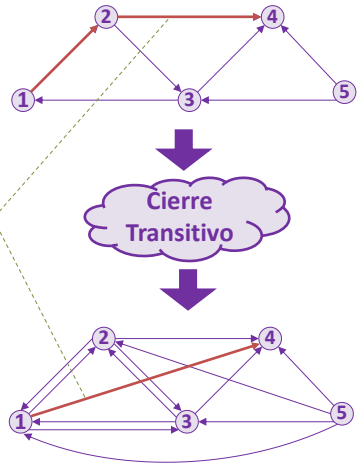
Cierre Transitivo de un Grafo: Enunciado



68

Algoritmo Floyd-Warshall

Cierre Transitivo de un Grafo: Enunciado

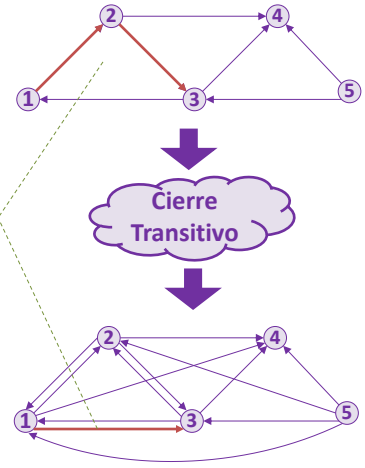


Hay un camino en el grafo de entrada del vértice **1** al vértice **4**

69

Algoritmo Floyd-Warshall

Cierre Transitivo de un Grafo: Enunciado



Hay un camino en el grafo de entrada del vértice **1** al vértice **3**

70

Algoritmo Floyd-Warshall

Cierre transitivo de un Grafo

Actividad 16.4. Utiliza el algoritmo de Floyd –Warshall para obtener el cierre transitivo de un grafo.

```
boolean[][] cierreTransitivo(boolean[][] grafo){
 boolean[][] d = new boolean[grafo.length][grafo.length];
 for (int i=0; i<grafo.length; i++)
   for (int j=0; j<grafo.length; j++)
      d[i][j]=grafo[i][j];
 for (int k=0; k<grafo.length; k++)
   for (int i=0; i<grafo.length; i++)
     for (int j=0; j<grafo.length; j++)
        d[i][j] = d[i][j] || (d[i][k] && d[k][j]);

 return d;
}
```

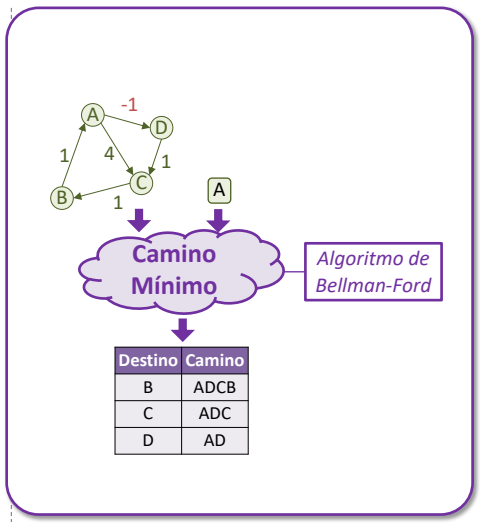
2. Algoritmo de Floyd-Warshall

1 2 3

71

Algoritmo de Bellman-Ford

Camino Mínimo



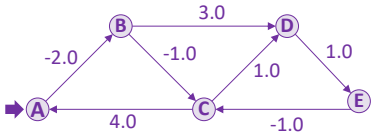
3. Algoritmo de Bellman-Ford

1 2 3

72

Algoritmo de Bellman-Ford

Ecuación de Recurrencia

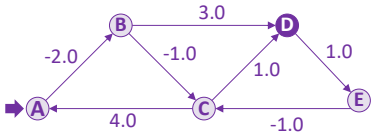


d(j,k): Distancia del camino más corto del vértice **A** al vértice **j** y que contenga **k** saltos como máximo.

73

Algoritmo de Bellman-Ford

Ecuación de Recurrencia



d(D,0): Distancia del camino más corto del vértice **A** al vértice **D** y que no contenga saltos.

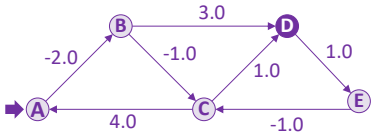
∞

No existe camino de **A** a **D** sin saltos

74

Algoritmo de Bellman-Ford

Ecuación de Recurrencia



d(D,1): Distancia del camino más corto del vértice **A** al vértice **D** y que contenga **1** salto como máximo.

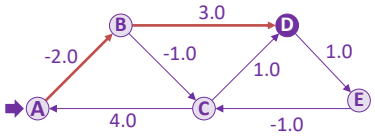
∞

No existe camino de **A** a **D** con **1** salto

75

Algoritmo de Bellman-Ford

Ecuación de Recurrencia



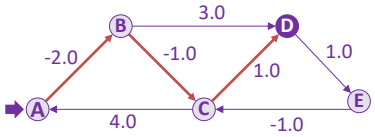
d(D,2): Distancia del camino más corto del vértice **A** al vértice **D** y que contenga **2** saltos como máximo.

d(D,2) = 1.0	Camino: **A-B-D**

76

Algoritmo de Bellman-Ford

Ecuación de Recurrencia



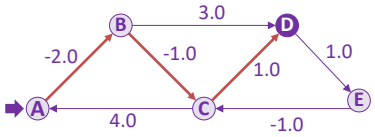
d(D,3): Distancia del camino más corto del vértice **A** al vértice **D** y que contenga **3** saltos como máximo.

d(D,3)= -2.0	Camino: **A-B-C-D**

77

Algoritmo de Bellman-Ford

Ecuación de Recurrencia



d(D,4): Distancia del camino más corto del vértice **A** al vértice **D** y que contenga **4** saltos como máximo.

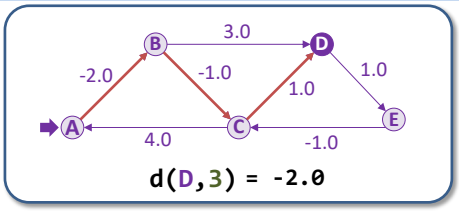
d(D,4)= -2.0	Camino: **A-B-C-D**

Problema de Caminos Mínimos Original!!

78

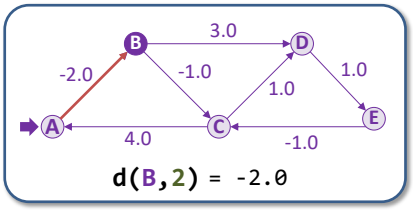
Algoritmo de Bellman-Ford

Ecuación de Recurrencia



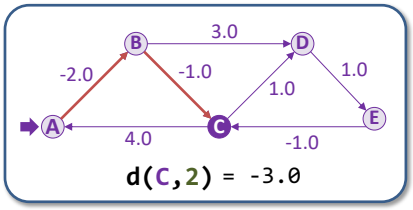
Pasa por **B** como máximo en **2** pasos

-2.0 + 3.0 = 1.0



Pasa por **C** como máximo en **2** pasos

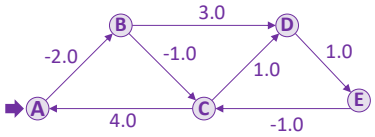
-3.0 + 1.0 = -2.0



79

Algoritmo de Bellman-Ford

Ecuación de Recurrencia. Caso Recursivo



d(j,k): Distancia del camino más corto del vértice **A** al vértice **j** y que contenga **k** saltos como máximo.

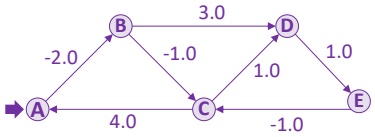
Caso Recursivo: $k > 0$

$$d(j,k) = min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\}$$

80

Algoritmo de Bellman-Ford

Ecuación de Recurrencia



d(D,0): Distancia del camino más corto del vértice **A** al vértice **D** y que no contenga saltos.

Caso Base:

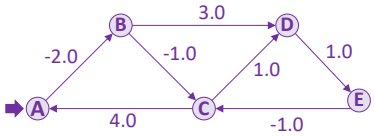
$$d(A, 0) = 0$$

$$Si\ j \neq A\ entonces\ d(j, 0) = \infty$$

81

Algoritmo de Bellman-Ford

Ecuación de Recurrencia



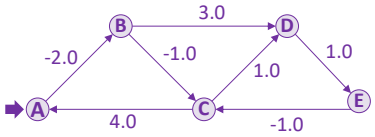
d(j,k): Distancia del camino más corto del vértice **A** al vértice **j** y que contenga **k** saltos como máximo.

$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ \min_{(i,j)\in E} \{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

82

Algoritmo de Bellman-Ford

Ecuación de Recurrencia



d(j,k): Distancia del camino más corto del vértice **A** al vértice **j** y que contenga **k** saltos como máximo.

$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

```
float[] d = new float[grafo.NumVertices];
```

```
if (d[i]+peso<d[j])
    d[j]=d[i]+peso;
```


83

Algoritmo de Bellman-Ford

Implementación

```
float[] distanciasBellmanFord(Graph grafo, int origen) {
   float[] d = new float[grafo.NumVertices];
   for (int i=0; i<grafo.NumVertices; i++) {d[i] = Float.MAX_VALUE;}
   d[origen] = 0.0;

   for (int k=1; k<grafo.NumVertices; k++)
      for (int e=0; e<grafo.numAristas; e++) {
        int i = grafo.arista[e].origen; int j = grafo.arista[e].destino;
        int peso = grafo.arista[e].peso;
        if (d[i]!=Float.MAX_VALUE && d[i]+peso<d[j])
            d[j]=d[i]+peso;
      }
   return d;
}
```


84

Algoritmo de Bellman-Ford

Implementación

```
float[] distanciasBellmanFord(Graph grafo, int origen) {
   float[] d = new float[grafo.NumVertices];
   for (int i=0; i<grafo.NumVertices; i++) {d[i] = Float.MAX_VALUE;}
   d[origen] = 0.0;

   for (int k=1; k<grafo.NumVertices; k++)
      for (int e=0; e<grafo.numAristas; e++) {
        int i = grafo.arista[e].origen; int j = grafo.arista[e].destino;
        int peso = grafo.arista[e].peso;
        if (d[i]!=Float.MAX_VALUE && d[i]+peso<d[j])
            d[j]=d[i]+peso;
      }
   return d;
}
```

Caso Base:

$$d(A, k) = 0$$

$$Si\ j \neq A\ entonces\ d(j, 0) = \infty$$

85

Algoritmo de Bellman-Ford

Implementación

```
float[] distanciasBellmanFord(Graph grafo, int origen) {
   float[] d = new float[grafo.NumVertices];
   for (int i=0; i<grafo.NumVertices; i++) {d[i] = Float.MAX_VALUE;}
   d[origen] = 0.0;

   for (int k=1; k<grafo.NumVertices; k++)
      for (int e=0; e<grafo.numAristas; e++) {
        int i = grafo.arista[e].origen; int j = grafo.arista[e].destino;
        int peso = grafo.arista[e].peso;
        if (d[i]!=Float.MAX_VALUE && d[i]+peso<d[j])
            d[j]=d[i]+peso;
      }
   return d;
}
```

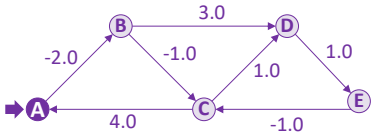
Caso Recursivo:

$$d(j, k) = min_{(i,j)\in E}\{d(i, k-1) + w_{i,j}\}$$

86

Algoritmo de Bellman-Ford

Ecuación de Recurrencia



$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

A	B	C	D	E
0.0				

3. Algoritmo de Bellman-Ford

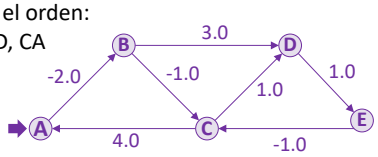
1 2 3

87

Algoritmo de Bellman-Ford

Ecuación de Recurrencia

Se procesan las aristas en el orden:
AB, BC, BD, DE, EC, CD, CA



$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

A	B	C	D	E
0.0	∞	∞	∞	∞

3. Algoritmo de Bellman-Ford

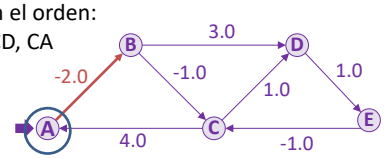
1 2 3

88

Algoritmo de Bellman-Ford

Ecuación de Recurrencia

Se procesan las aristas en el orden:
AB, BC, BD, DE, EC, CD, CA



$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

A	B	C	D	E
0.0	-2.0	∞	∞	∞

0.0-2.0

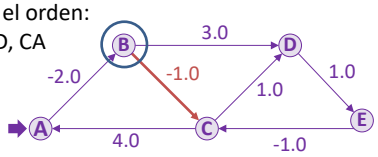
Se procesan las aristas en el orden:
AB, BC, BD, CD, DE, EC, CA

89

Algoritmo de Bellman-Ford

Ecuación de Recurrencia

Se procesan las aristas en el orden:
AB, BC, BD, DE, EC, CD, CA



$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

A	B	C	D	E
0.0	-2.0	-3.0	∞	∞

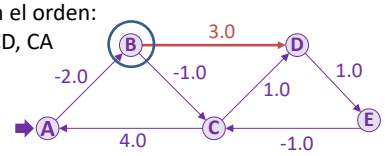
-2.0-1.0

90

Algoritmo de Bellman-Ford

Ecuación de Recurrencia

Se procesan las aristas en el orden:
AB, BC, **BD**, DE, EC, CD, CA



$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

A	B	C	D	E
0.0	-2.0	-3.0	1.0	∞

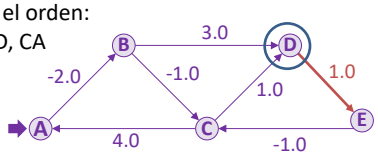
-2.0+3.0

91

Algoritmo de Bellman-Ford

Ecuación de Recurrencia

Se procesan las aristas en el orden:
AB, BC, BD, **DE**, EC, CD, CA



$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

A	B	C	D	E
0.0	-2.0	-3.0	1.0	2.0

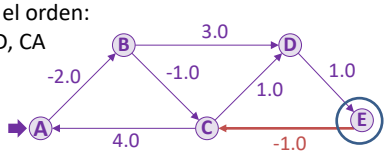
1.0+1.0

92

Algoritmo de Bellman-Ford

Ecuación de Recurrencia

Se procesan las aristas en el orden:
AB, BC, BD, DE, **EC**, CD, CA



$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

A	B	C	D	E
0.0	-2.0	**-3.0**	1.0	**2.0**

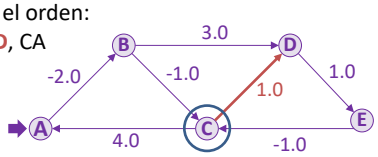
2.0-1.0
No se actualiza

93

Algoritmo de Bellman-Ford

Ecuación de Recurrencia

Se procesan las aristas en el orden:
AB, BC, BD, DE, EC, **CD**, CA



$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

A	B	C	D	E
0.0	-2.0	**-3.0**	**-2.0**	2.0

-3.0+1.0

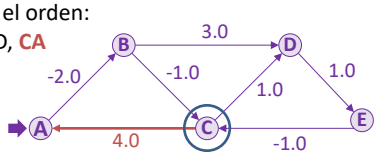
Se procesan las aristas en el orden:
AB, BC, BD, DE, EC, CA, CD

94

Algoritmo de Bellman-Ford

Ecuación de Recurrencia

Se procesan las aristas en el orden:
AB, BC, BD, DE, EC, CD, **CA**



$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

A	B	C	D	E
0.0	-2.0	**-3.0**	-2.0	2.0

-3.0+4.0

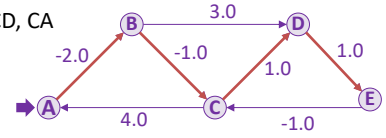
No se actualiza

95

Algoritmo de Bellman-Ford

Ecuación de Recurrencia

Se procesan las aristas en el orden:
AB, BC, BD, DE, EC, CD, CA



$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

Número de iteraciones por las aristas: **1**

A	B	C	D	E
0.0	-2.0	-3.0	-2.0	2.0

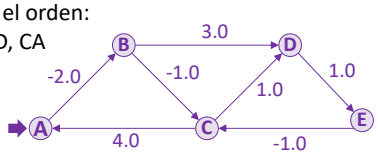
No es la distancia del camino mínimo

96

Algoritmo de Bellman-Ford

Ecuación de Recurrencia

Se procesan las aristas en el orden:
AB, BC, BD, DE, EC, CD, CA



$$d(j,k) = \begin{cases} 0 & Si\ j = A \\ \infty & Si\ j \neq A\ y\ k = 0 \\ min_{(i,j)\in E}\{d(i,k-1) + w_{i,j}\} & k > 0 \end{cases}$$

Número de iteraciones por las aristas: 4

A	B	C	D	E
0.0	-2.0	-3.0	-2.0	-1.0

97

Algoritmo de Bellman-Ford

Implementación

Actividad 16.5. Calcula la complejidad en tiempo del algoritmo de Bellman-Ford.

Θ(V·E)

```
float[] distanciasBellmanFord(Graph grafo, int origen) {
   float[] d = new float[grafo.NumVertices];
   for (int i=0; i<grafo.NumVertices; i++) {d[i] = Float.MAX_VALUE;}
   d[origen] = 0.0;

   for (int k=1; k<grafo.NumVertices; k++)
      for (int e=0; e<grafo.numAristas; e++) {
        int i = grafo.arista[e].origen; int j = grafo.arista[e].destino;
        int peso = grafo.arista[e].peso;
        if (d[i]!=Float.MAX_VALUE && d[i]+peso<d[j])
            d[j]=d[i]+peso;
      }
   return d;
}
```


98

Algoritmo de Bellman-Ford

Implementación

Actividad 16.6. Modifica el algoritmo para que se puedan obtener cuáles son los caminos

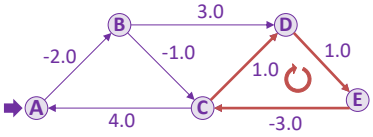
```
float[] distanciasBellmanFord(Graph grafo, int origen, int[] predecesor) {
   float[] d = new float[grafo.NumVertices];
   for (int i=0; i<grafo.NumVertices; i++) {
      d[i] = Float.MAX_VALUE; predecesor[i]=i;}
   d[origen] = 0.0;

   for (int k=1; k<grafo.NumVertices; k++)
      for (int e=0; e<grafo.numAristas; e++) {
        int i = grafo.arista[e].origen; int j = grafo.arista[e].destino;
        int peso = grafo.arista[e].peso;
        if (d[i]!=Float.MAX_VALUE && d[i]+peso<d[j]){
            d[j]=d[i]+peso;
            predecesor[j]=i; }
      }
   return d;
}
```


99

Algoritmo de Bellman-Ford

Ciclos Negativos

¿Qué ocurre si hay ciclos Negativos?



Número de iteraciones por las aristas: **1**

A	B	C	D	E
0.0	-2.0	-4.0	-2.0	-1.0

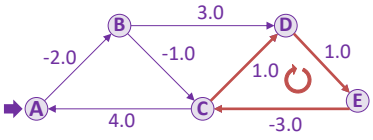
Se procesan las aristas en el orden:
AB, BD, BC, CD, DE, EC, CA

100

Algoritmo de Bellman-Ford

Ciclos Negativos

¿Qué ocurre si hay ciclos Negativos?



Número de iteraciones por las aristas: **2**

A	B	C	D	E
0.0	-2.0	-5.0	-3.0	-2.0

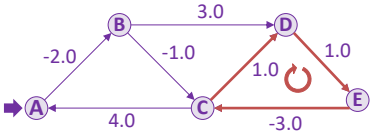
Se procesan las aristas en el orden:
AB, BD, BC, CD, DE, EC, CA

101

Algoritmo de Bellman-Ford

Ciclos Negativos

¿Qué ocurre si hay ciclos Negativos?



Número de iteraciones por las aristas: **3**

A	B	C	D	E
0.0	-2.0	-6.0	-4.0	-3.0

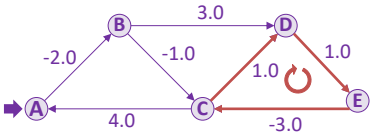
Se procesan las aristas en el orden:
AB, BD, BC, CD, DE, EC, CA

102

Algoritmo de Bellman-Ford

Ciclos Negativos

¿Qué ocurre si hay ciclos Negativos?



Número de iteraciones por las aristas: **4**

A	B	C	D	E
0.0	-2.0	-7.0	-5.0	-4.0

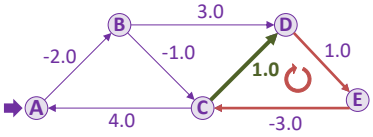
Se procesan las aristas en el orden:
AB, BD, BC, CD, DE, EC, CA

103

Algoritmo de Bellman-Ford

Ciclos Negativos

¿Qué ocurre si hay ciclos Negativos?



Número de iteraciones por las aristas: **4**

A	B	C	D	E
0.0	-2.0	**-7.0**	**-5.0**	-4.0

-7.0 + 1.0 < -5.0

Se procesan las aristas en el orden:
AB, BD, BC, CD, DE, EC, CA

104

Algoritmo de Bellman-Ford

Implementación

```
boolean distanciasBellmanFord(Graph grafo, int origen, int[] p, float[] d) {
   for (int i=0; i<grafo.NumVertices; i++) {
      d[i] = Float.MAX_VALUE; p[i]=i;}
   d[origen] = 0.0;
   for (int k=1; k<grafo.NumVertices; k++)
      for (int e=0; e<grafo.numAristas; e++) {
        int i = grafo.arista[e].origen; int j = grafo.arista[e].destino;
        int peso = grafo.arista[e].peso;
        if (d[i]!=Float.MAX_VALUE && d[i]+peso<d[j])
           { d[j]=d[i]+peso;             p[j]=i;}
        }
   int e=0; boolean cicloNegativo= false;
   while (e<grafo.numAristas && !cicloNegativo) {
     int i = grafo.arista[e].origen; int j = grafo.arista[e].destino;
     int peso = grafo.arista[e].peso;
     if (d[i]!=Float.MAX_VALUE && d[i]+peso<d[j]) cicloNegativo=true;
     e++;
    }
   return cicloNegativo;
}
```

3. Algoritmo de Bellman-Ford

1 2 3

105