

ALGORITMO DE FLOYD-WARSHALL

Curso: Algoritmos y Estructura de Datos

Prof. Mamani Rodriguez, Zoraida Emperatriz

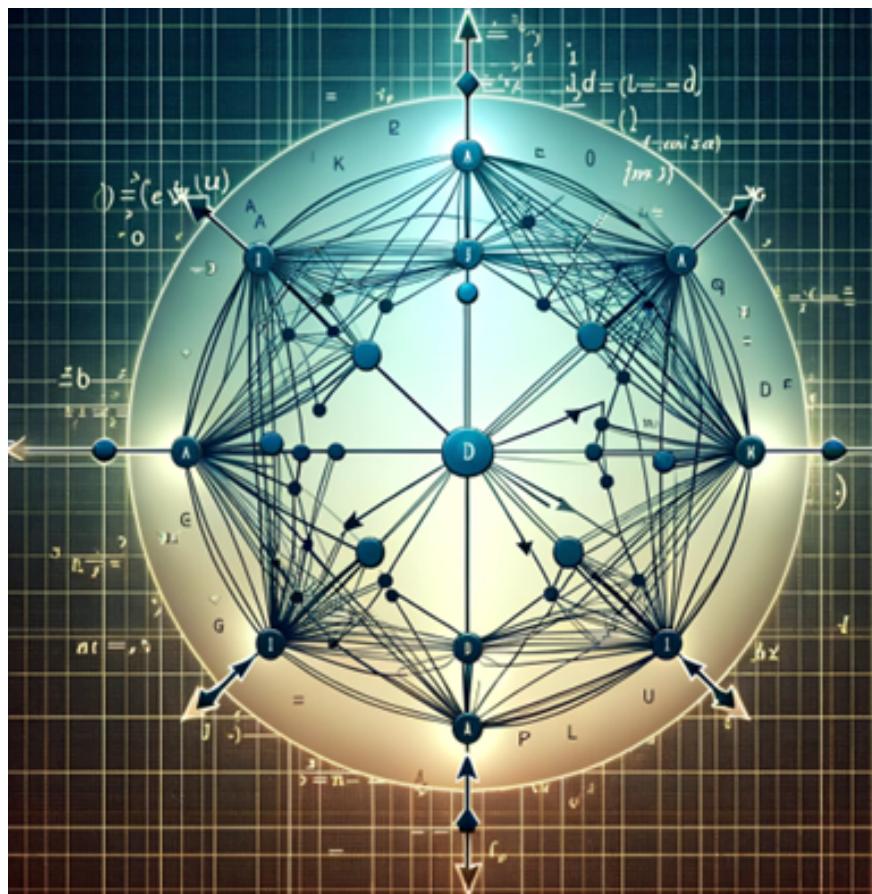
Integrantes:

- Osorio Tello, Jesus Diego
- Llanos Rosadio, José Ismael
- Olivera Calderón, Renato Steven



Origen

- Stephen Warshall, publica su algoritmo original en 1962.,

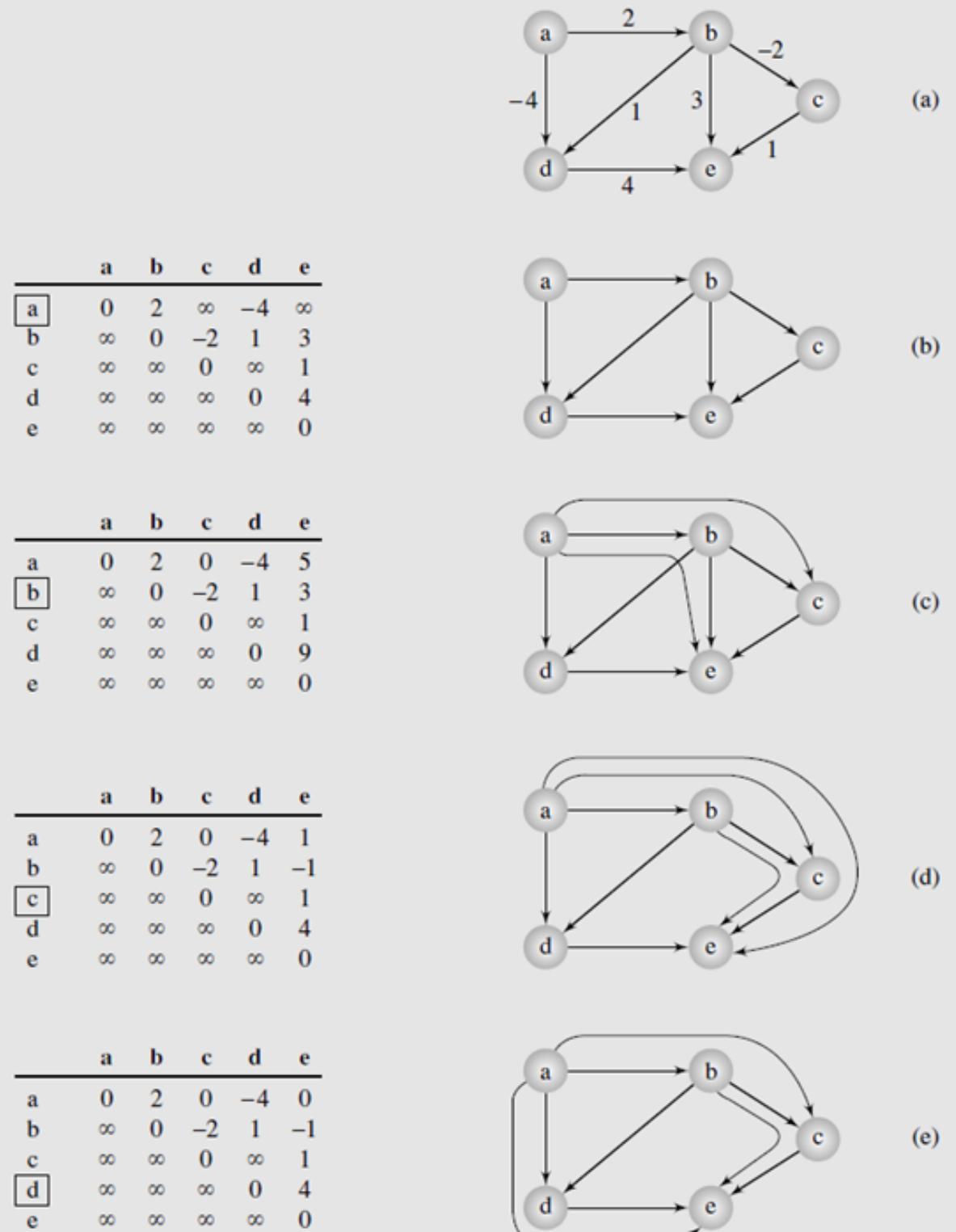


Grafo dirigido, que demuestra el concepto de cierre transitivo con caminos entre varios nodos.

- Robert W. Floyd, adaptó y generalizó el algoritmo de Warshall, usa la metodología de Programación Dinámica para resolver el problema

Algoritmo de Floyd - Warshall

- El algoritmo de Floyd-Warshall es un método fundamental en teoría de grafos para encontrar los caminos más cortos entre todos los pares de vértices en un grafo ponderado. Es especialmente útil en análisis de redes y situaciones donde se necesita conocer todas las distancias más cortas dentro de una red.
- Funciona en grafos dirigidos y no dirigidos, puede manejar pesos negativos, pero no ciclos negativos y su complejidad es $O(V^3)$ donde V es el número de vértices o nodos



Algoritmo de Floyd - Warshall: Modo de uso

- El algoritmo trabaja con programación dinámica, lo que garantiza que la solución por éste algoritmo es óptima, además que entrega todos los caminos más cortos para ir desde un nodo i a un nodo j cualquiera y el recorrido necesario para completar dicho recorrido.
- Representa una red de n nodos como una matriz cuadrada de orden n, la llamaremos matriz C. De esta forma, el valor C_{ij} representa el coste de ir desde el nodo i al nodo j.

- Definiremos otra matriz D, cuadrada de orden n, cuyos elementos van a ser los nodos predecesores den el camino hacia el nodo origen, es decir, el valor D_{ij} representará el nodo predecesor a j en el camino mínimo desde i hasta j. Inicialmente se comienza en caminos de longitud , por lo que $D_{ij} = i$.
- Las diagonales de ambas matrices representan el coste y el nodo predecesor para ir de un nodo a si mismo, por lo que no sirven para nada, estarán boqueadas.

- Fórmula:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

Algoritmo de Floyd -Warshall: Pseudocódigo

Algorithm Floyd-Warshall (A, n):

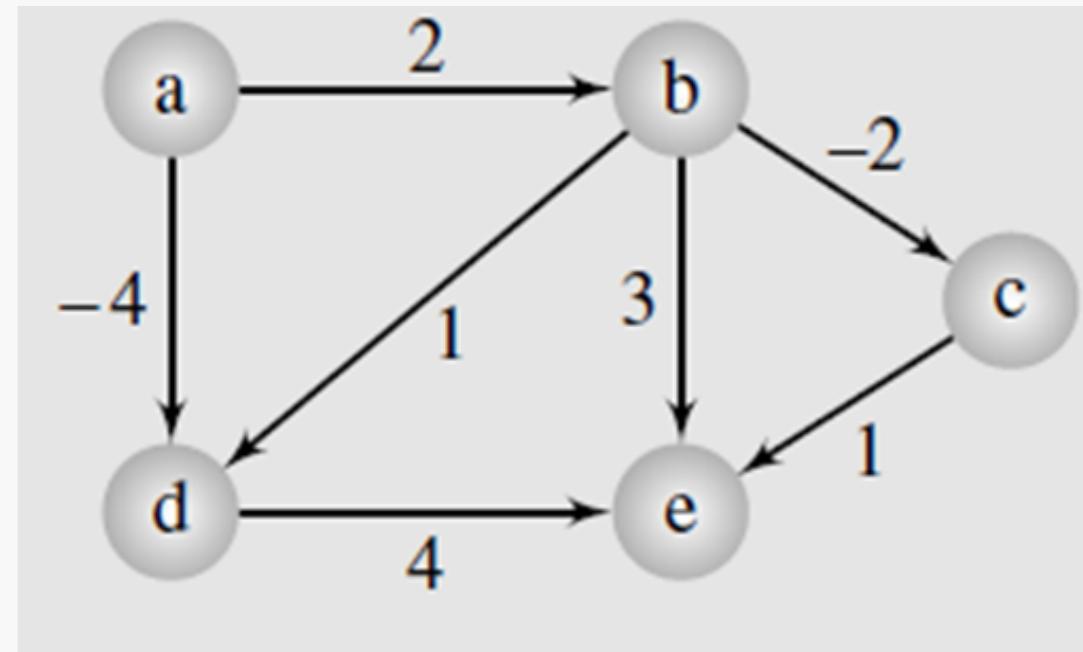
```
for  $k \leftarrow 1$  to  $n$  do
    for  $i \leftarrow 1$  to  $n$  do
        for  $j \leftarrow 1$  to  $n$  do
             $A[i][j] \leftarrow \min(A[i][j], A[i][k] + A[k][j])$ 
```

Pseudocódigo del algoritmo Floyd-Warshall

Complejidad y Aplicaciones:

- El algoritmo es capaz de hacer esto con solo n^2 comparaciones. Lo hace mejorando paulatinamente una estimación del camino más corto entre dos vértices, hasta que se sabe que la optimización es óptima.
- Camino mínimo en grafos dirigidos.
- Inversión de matrices de números reales (algoritmo de Gauss - Jordan)
- Comprobar si un grafo no dirigido es bipartido.

Ejemplo



The purple wavy line graphic is located on the right side of the two matrices, extending from the bottom to the top of the slide.

Matriz de distancias					
	a	b	c	d	e
a	0	2	∞	-4	∞
b	∞	0	-2	1	3
c	∞	∞	0	∞	1
d	∞	∞	∞	0	4
e	∞	∞	∞	∞	0

Matriz de recorridos					
	a	b	c	d	e
a	-	b	c	d	e
b	a	-	c	d	e
c	a	b	-	d	e
d	a	b	c	-	e
e	a	b	c	d	-

	a	b	c	d	e	
a	0	2	∞	-4	∞	
b	∞	0	-2	1	3	
c	∞	∞	0	∞	1	
d	∞	∞	∞	0	4	
e	∞	∞	∞	∞	0	

	a	b	c	d	e	
a	-	b	c	d	e	
b	a	-	c	d	e	
c	a	b	-	d	e	
d	a	b	c	-	e	
e	a	b	c	d	-	

	a	b	c	d	e
a	0	2	∞	-4	∞
b	∞	0	-2	1	3
c	∞	∞	0	∞	1
d	∞	∞	∞	0	4
e	∞	∞	∞	∞	0

	a	b	c	d	e
a	0	2	0	-4	5
b	∞	0	-2	1	3
c	∞	∞	0	∞	1
d	∞	∞	∞	0	4
e	∞	∞	∞	∞	0

	a	b	c	d	e
a	-	b	b	d	b
b	a	-	c	d	e
c	a	b	-	d	e
d	a	b	c	-	e
e	a	b	c	d	-

	a	b	c	d	e
a	0	2	0	-4	5
b	∞	0	-2	1	3
c	∞	∞	0	∞	1
d	∞	∞	∞	0	4
e	∞	∞	∞	∞	0

	a	b	c	d	e
a	0	2	0	-4	1
b	∞	0	-2	1	-1
c	∞	∞	0	∞	1
d	∞	∞	∞	0	4
e	∞	∞	∞	∞	0

	a	b	c	d	e
a	-	b	b	d	c
b	a	-	c	d	c
c	a	b	-	d	e
d	a	b	c	-	e
e	a	b	c	d	-

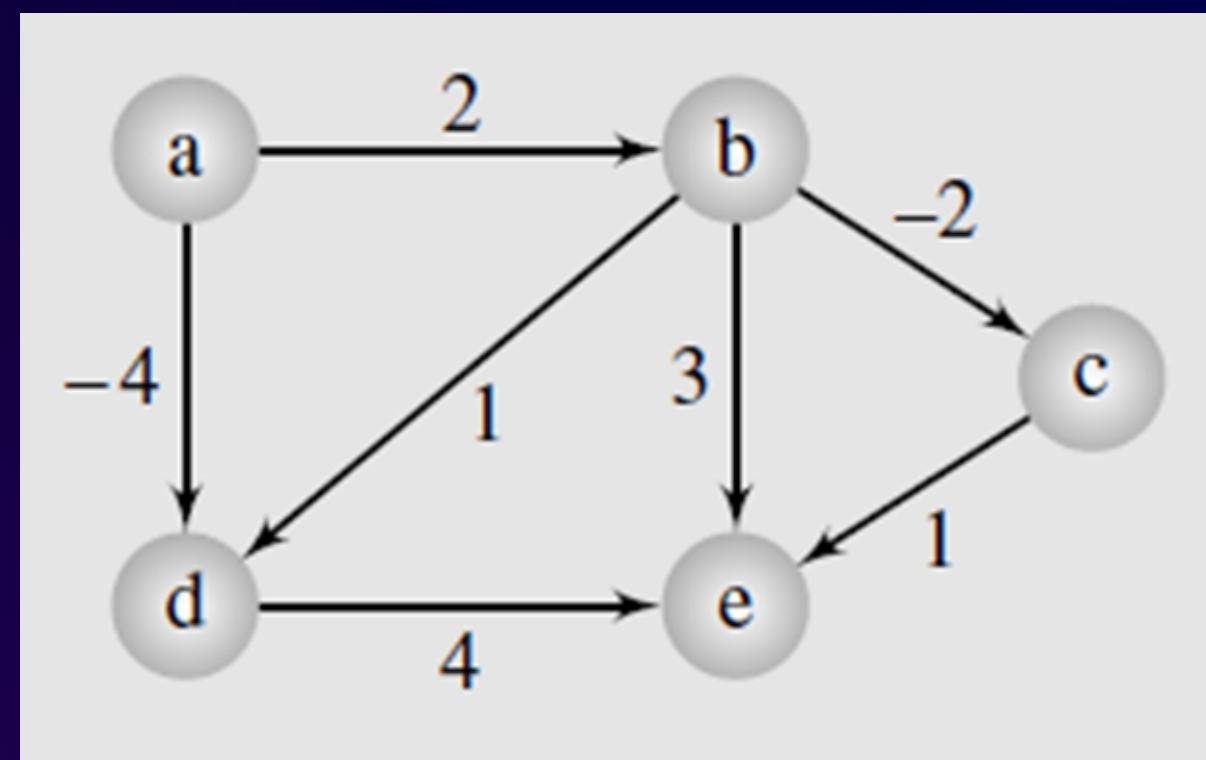
	a	b	c	d	e
a	0	2	0	-4	1
b	∞	0	-2	1	-1
c	∞	∞	0	∞	1
d	∞	∞	∞	0	4
e	∞	∞	∞	∞	0

	a	b	c	d	e
a	0	2	0	-4	0
b	∞	0	-2	1	-1
c	∞	∞	0	∞	1
d	∞	∞	∞	0	4
e	∞	∞	∞	∞	0

	a	b	c	d	e
a	-	b	b	d	d
b	a	-	c	d	c
c	a	b	-	d	e
d	a	b	c	-	e
e	a	b	c	d	-

	a	b	c	d	e
a	0	2	0	-4	0
b	∞	0	-2	1	-1
c	∞	∞	0	∞	1
d	∞	∞	∞	0	4
e	∞	∞	∞	∞	0

	a	b	c	d	e
a	-	b	b	d	d
b	a	-	c	d	c
c	a	b	-	d	e
d	a	b	c	-	e
e	a	b	c	d	-



Código del Algoritmo Floyd-Warshall



```
1 package uni.aed.Algoritmo_Floyd_Warshall;
2
3
4 public class Pc5_FloydWarshall {
5
6     // Constante para representar un valor infinito. Se usa para distancias iniciales entre nodos no conectados.
7     public static final int INF = Integer.MAX_VALUE;
8
9     // Método que implementa el algoritmo de Floyd-Warshall
10    public void floydWarshall(int[][] graph) {
11        int n = graph.length; //número de nodos en el grafo
12        int[][] dist = new int[n][n]; // Matriz que almacenará las distancias más cortas entre los nodos
13
14        // Inicialización de la matriz de distancias con los valores iniciales del grafo
15        inicializarMatrizDistancias(graph, dist, n);
16
17        // Aplicación del algoritmo de Floyd-Warshall para calcular las distancias más cortas
18        aplicarFloydWarshall(dist, n);
19
20        // Impresión de la matriz de distancias resultante
21        imprimirDistancias(dist, n);
22    }
23
24
25    private void inicializarMatrizDistancias(int[][] graph, int[][] dist, int n) {
26        for (int i = 0; i < n; i++) {
27            // Copia la fila i del grafo a la fila i de la matriz de distancias
28            System.arraycopy(graph[i], srcPos:0, dist[i], destPos: 0, length:n);
29        }
30    }
}
```

```
33     private void aplicarFloydWarshall(int[][][] dist, int n) {  
34         // Iteración sobre todos los nodos intermedios k  
35         for (int k = 0; k < n; k++) {  
36             // Iteración sobre todos los nodos i  
37             for (int i = 0; i < n; i++) {  
38                 // Iteración sobre todos los nodos j  
39                 for (int j = 0; j < n; j++) {  
40                     // Comprobación y actualización de la distancia si se encuentra un camino más corto  
41                     if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {  
42                         dist[i][j] = dist[i][k] + dist[k][j];  
43                     }  
44                 }  
45             }  
46         }  
47     }  
48  
49     private void imprimirDistancias(int[][][] dist, int n) {  
50         System.out.println("Matriz de distancias mínimas:");  
51         for (int i = 0; i < n; i++) {  
52             for (int j = 0; j < n; j++) {  
53                 // Si la distancia es infinita, imprime "INF". De lo contrario, imprime la distancia real.  
54                 if (dist[i][j] == INF) {  
55                     System.out.print("INF\t");  
56                 } else {  
57                     System.out.print(dist[i][j] + "\t");  
58                 }  
59             }  
60             System.out.println();  
61         }  
62     }
```

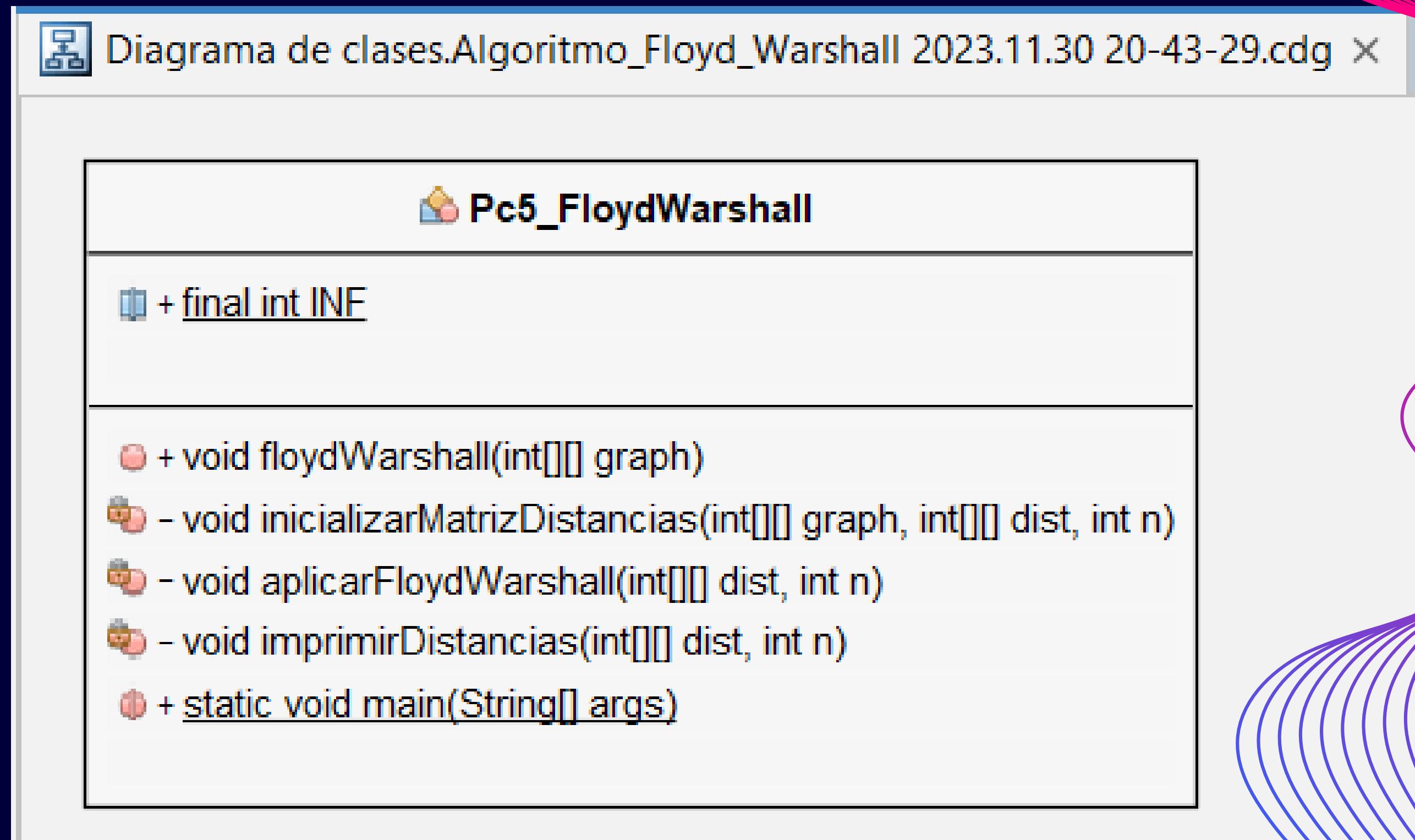
```
64     // Método principal para ejecutar el programa
65     public static void main(String[] args) {
66         // Matriz de adyacencia que representa el grafo
67         int[][] graph = {
68             { 0, 2, INF, -4, INF },
69             { INF, 0, -2, 1, 3 },
70             { INF, INF, 0, INF, 1 },
71             { INF, INF, INF, 0, 4 },
72             { INF, INF, INF, INF, 0 }
73         };
74
75         // Creación de una instancia de la clase y ejecución del algoritmo de Floyd-Warshall
76         Pcs_FloydWarshall fw = new Pcs_FloydWarshall();
77         fw.floydWarshall(graph);
78     }
79 }
```

Resultado del ejemplo

Output - Run (Pc5_FloydWarshall)

```
from pom.xml
[ jar ]
--- resources:3.3.1:resources (default-resources) @ 232CC232Project ---
Copying 5 resources from src\main\resources to target\classes
--- compiler:3.11.0:compile (default-compile) @ 232CC232Project ---
Nothing to compile - all classes are up to date
--- exec:3.1.0:exec (default-cli) @ 232CC232Project ---
Matriz de distancias mínimas:
0      2      0     -4      0
INF    0     -2      1     -1
INF    INF     0     INF      1
INF    INF     INF      0      4
INF    INF     INF     INF      0
BUILD SUCCESS
```

Diagrama de la clase



Caso de aplicación

Este caso tratará sobre un sistema de transporte con cinco estaciones. Cada estación será representada por un nodo en el grafo, y las rutas entre las estaciones serán las aristas con pesos que representan el tiempo de viaje en minutos.. Se tiene 5 estaciones:

- Estación A
- Estación B
- Estación C
- Estación D
- Estación E

Matriz de adyacencia

	A	B	C	D	E
A	0	10	INF	30	INF
B	10	0	20	INF	INF
C	INF	20	0	10	15
D	30	INF	10	0	5
E	INF	INF	15	5	0

Matriz de Recorrido

	A	B	C	D	E
A	-	B	C	D	E
B	A	-	C	D	E
C	A	B	-	D	E
D	A	B	C	-	E
E	A	B	C	D	-

Luego de aplicar el **Algoritmo Floyd-Warshall**

Matriz de adyacencia

	A	B	C	D	E
A	0	10	30	30	35
B	10	0	20	30	35
C	30	20	0	10	15
D	30	30	10	0	5
E	35	35	15	5	0

Matriz de Recorrido

	A	B	C	D	E
A	-	A	B	A	D
B	B	-	B	C	C
C	B	C	-	C	C
D	D	C	D	-	D
E	D	C	E	E	-

Muchas Gracias

