



# Paralelización de Algoritmos Matriciales Masivos con CUDA: Warshall Lógico (Fase 1)

Reporte de Diseño y Análisis Secuencial

---

**Asignatura:** Algoritmos Paralelos y  
Distribuidos

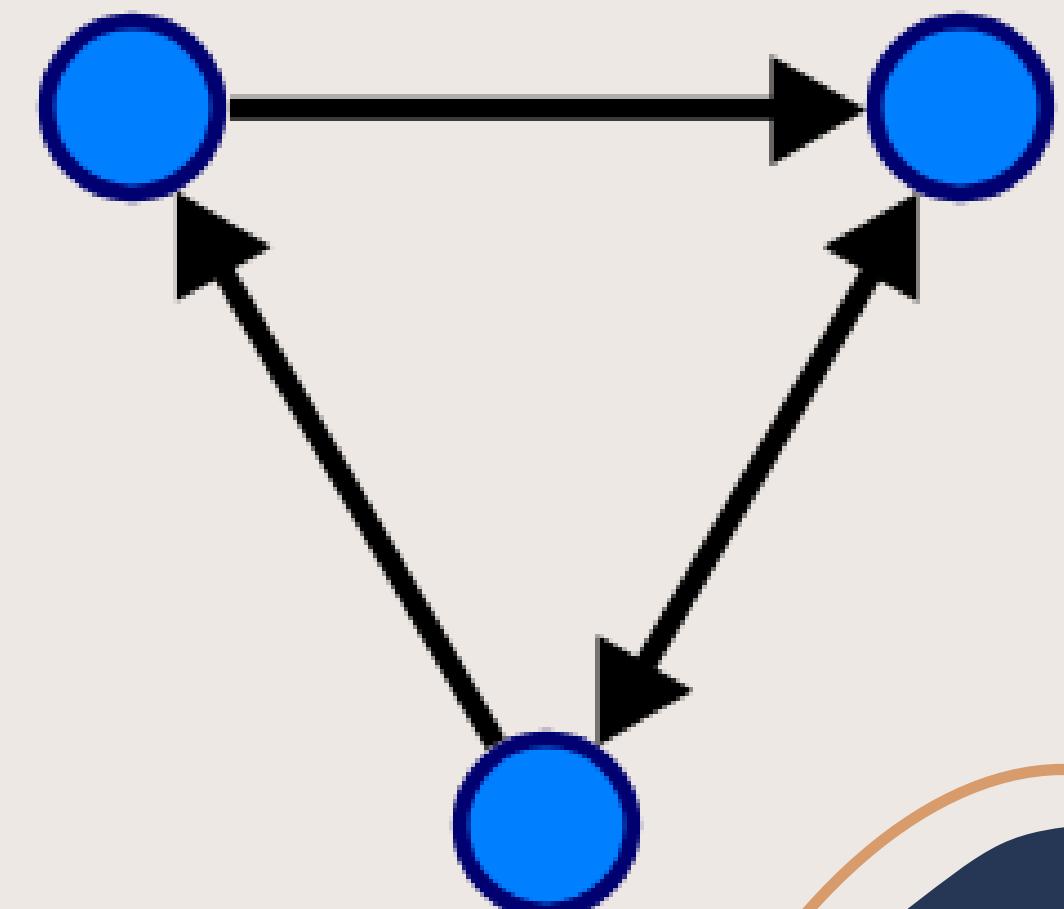
**Docente:**  
Mgt. Ray Dueñas Jiménez

**Autores:**  
Castro Pari, Rayneld Fidel  
Mayhuire Chacon, Brenda Lucia  
Mendoza Quispe, Jose Daniel  
Perez Cahuana, Gabriel  
Zevallos Yanqui, Andy Jefferson

# GRAFO DIRIGIDO

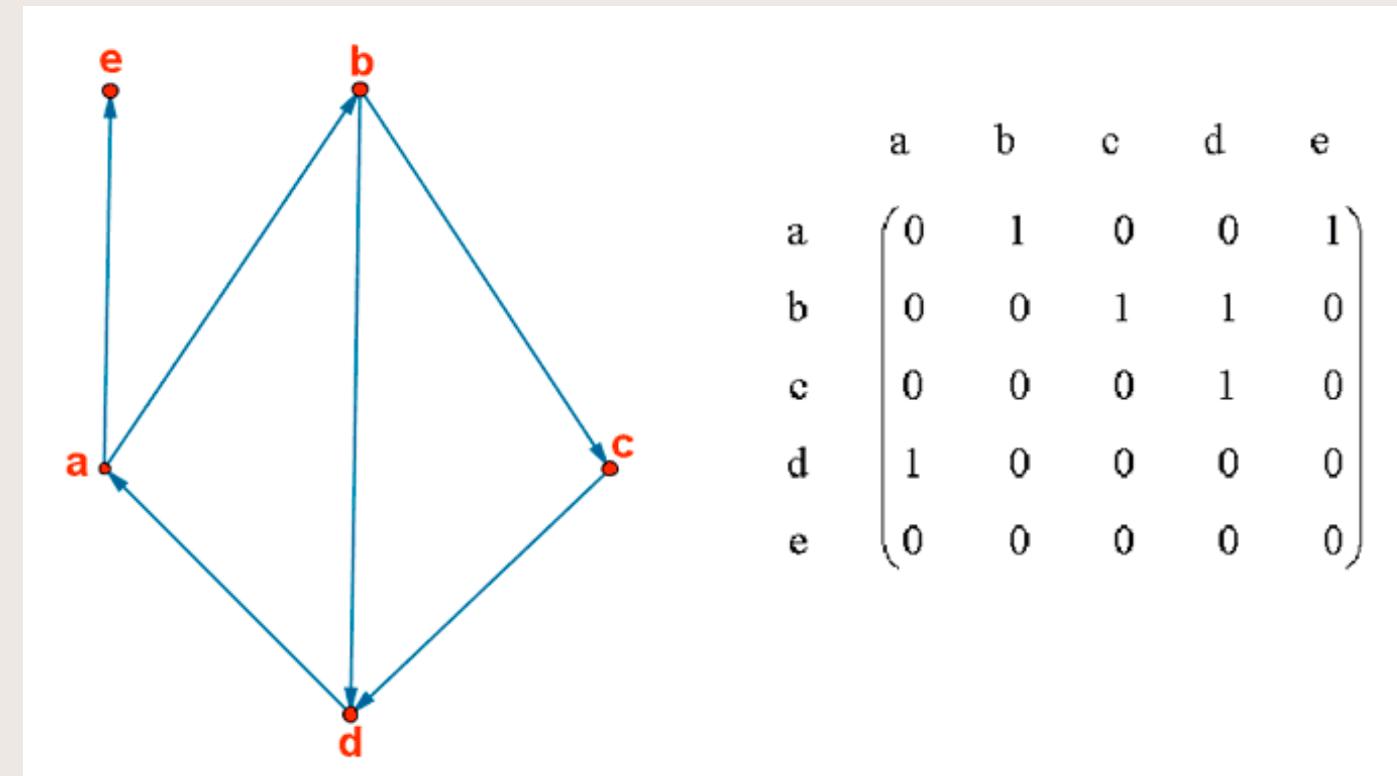
¿Qué es un Grafo Dirigido?

- Un grafo dirigido  $G = (V, E)$  está compuesto por:
  - **V:** un conjunto de vértices  $(V_1, V_2, \dots, V_N)$
  - **E:** un conjunto de aristas (o arcos), que son pares ordenados de vértices  $(i, j)$ , donde hay una relación de dirección entre los vértices  $i$  y  $j$ .



# REPRESENTACION

- La matriz de adyacencia A es una matriz booleana  $N \times N$  que describe las conexiones entre vértices:
  - $A_{ij}=1$  si existe un arco directo de  $i$  a  $j$ .
  - $A_{ij}=0$  si no existe un arco directo de  $i$  a  $j$ .



# ALCANZABILIDAD Y CERRADURA TRANSITIVA

## ¿QUÉ ES LA ALCANZABILIDAD?

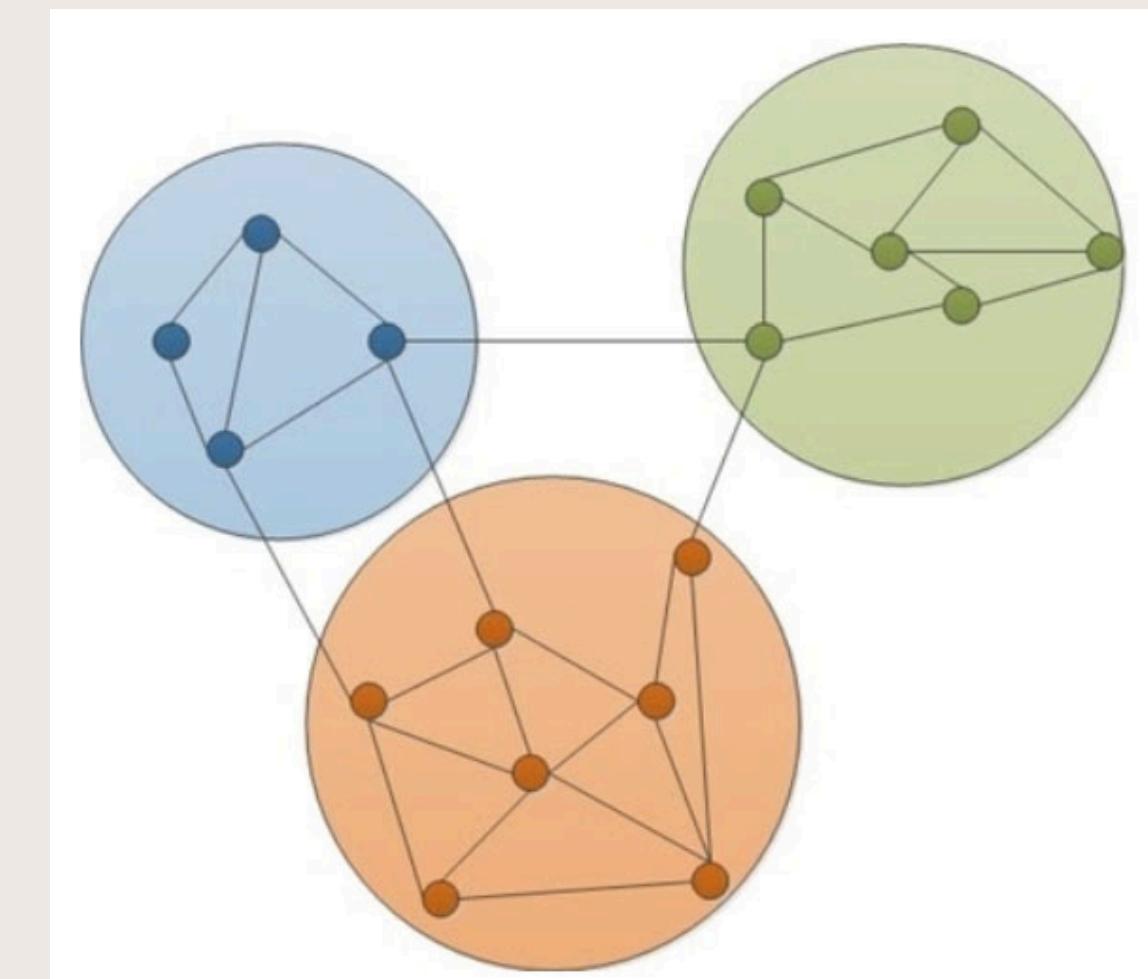
¿Existe un camino de un vértice  $i$  a un vértice  $j$ ?

El camino puede tener longitud mayor o igual a 1.

No solo se consideran arcos directos, sino también caminos con vértices intermedios.

## CERRADURA TRANSITIVA

- La cerradura transitiva de un grafo dirigido resume toda su conectividad.
- Se representa mediante una matriz booleana  $T$ , donde:
- $T_{ij}=1 \Leftrightarrow$  existe algún camino de  $i$  a  $j$ .
- $T_{ij}=0$  en caso contrario.



# EXPRESIÓN EN FORMA MATRICIAL

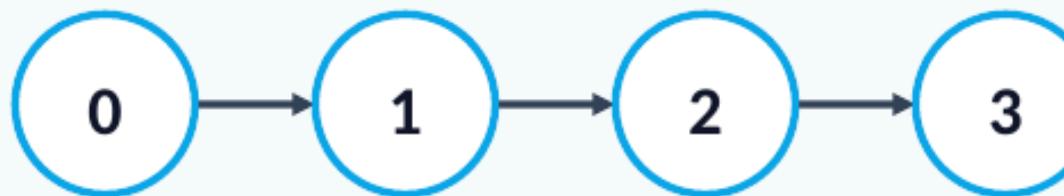
$$A_{ij} = A_{ij} \vee (A_{ik} \wedge A_{kj})$$

- $A_{ij}$ : alcanzabilidad conocida hasta el momento.
- $A_{ik} \wedge A_{kj}$ : nuevo camino pasando por k.
- El OR acumula la información de alcanzabilidad.

# Warshall lógico: grafo → matriz → cerradura transitiva

Ejemplo con 4 vértices (0,1,2,3). Resumen visual del proceso y el resultado.

## 1) Grafo dirigido de ejemplo



Arcos:  $0 \rightarrow 1$ ,  $1 \rightarrow 2$ ,  $2 \rightarrow 3$

Objetivo: calcular si existe camino  $i \Rightarrow j$  (uno o más saltos).

Warshall lógico construye la cerradura transitiva  $T$  a partir de la matriz de adyacencia  $A$ , permitiendo vértices intermedios  $k = 0..n-1$ .

## 2) Matriz de adyacencia $A$ (conexión directa)

	0	1	2	3
0	0	1	0	0
1	0	0	1	0
2	0	0	0	1
3	0	0	0	0

## 6) Resultado: cerradura transitiva $T$

	0	1	2	3
0	0	1	1	1
1	0	0	1	1
2	0	0	0	1
3	0	0	0	0

= conexiones descubiertas por caminos indirectos

## 4) Regla clave (Warshall lógico)

$$A[i,j] \leftarrow A[i,j] \vee (A[i,k] \wedge A[k,j])$$

" $i$  llega a  $j$  si ya llegaba, o si llega a  $k$  y  $k$  llega a  $j$ "

## 5) Qué se descubre (pasos clave)

- $k=0$ : no cambia
- $k=1$ : aparece  $0 \rightarrow 2$
- $k=2$ : aparecen  $1 \rightarrow 3$  y  $0 \rightarrow 3$
- $k=3$ : no cambia

## 7) Interpretación

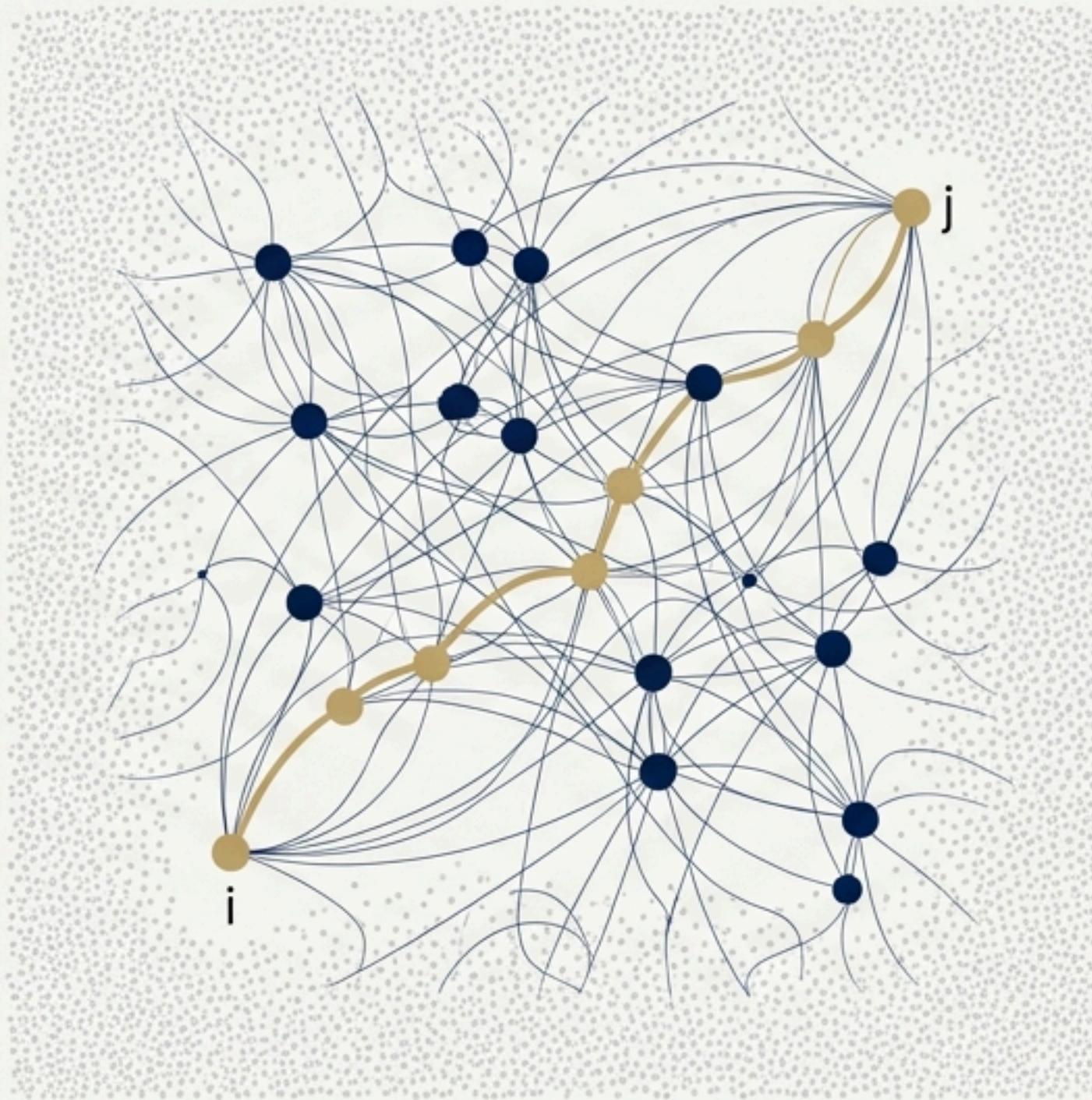
En la matriz final  $T$ :  $T[i,j]=1 \Leftrightarrow$  existe un camino  $i \Rightarrow j$ .

Ej.:  $T[0,3]=1$  ✓ ( $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$ ) |

$T[2,0]=0$  ✗

# El análisis de alcanzabilidad en grafos masivos es un problema fundamental con un costo computacional prohibitivo.

- El cómputo moderno opera sobre grafos masivos ( $N \geq 1024$  nodos) en dominios críticos: conectividad de redes, análisis de dependencias de software, grafos de estados y logística.
- Una pregunta clave es la **alcanzabilidad**: ¿existe un camino entre cualquier par de nodos ( $i, j$ )?
- El algoritmo de Warshall Lógico ofrece una solución clásica y robusta para este problema, calculando la **cerradura transitiva** de la matriz de adyacencia del grafo.



# Representación Matricial

De los datos de Entrada (matriz de entrada):

```
Ingrese la matriz de adyacencia (4x4) con 0/1.  
Formato permitido por fila: '0 1 0 1' o '0101...'
```

```
Fila 0: 0 1 0 1
```

```
Fila 1: 1111
```

```
Fila 2: 0000
```

```
Fila 3: 1000
```

```
repeats (>=1): 5
```

```
verify (1=si, 0=no): 1
```

```
print (1=si, 0=no): 1
```

```
Densidad p calculada desde la matriz: 0.438
```

# Representación Matricial

De los datos de Salida (matrices de salida):

```
==== MATRIZ DE ENTRADA (Grafo / Adyacencia) (N=4) ===
```

	0	1	2	3
0	0	1	0	1
1	1	1	1	1
2	0	0	0	0
3	1	0	0	0

```
==== MATRIZ DE VERIFICACIÓN (Referencia BFS) (N=4) ===
```

	0	1	2	3
0	1	1	1	1
1	1	1	1	1
2	0	0	0	0
3	1	1	1	1

```
==== MATRIZ DE SALIDA (Marshall lógico) (N=4) ===
```

	0	1	2	3
0	1	1	1	1
1	1	1	1	1
2	0	0	0	0
3	1	1	1	1

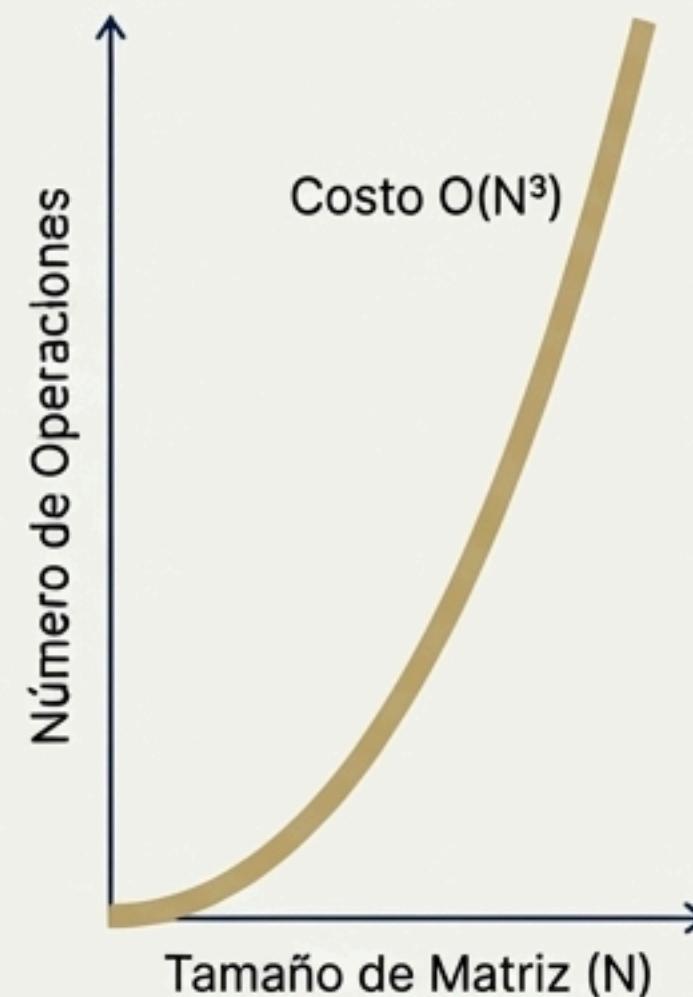
VALIDACIÓN (BFS) para N=4: OK

Tiempo del núcleo (marshall\_logical): 0.000000 s

# La complejidad temporal $O(N^3)$ del algoritmo de Warshall lo vuelve inviable para matrices masivas en CPU.

## La Teoría

- El algoritmo se basa en una recurrencia fundamental:  $A[i][j] \leftarrow A[i][j] \vee (A[i][k] \wedge A[k][j]).$
- Esto se traduce en una estructura de estructura de triple bucle anidado ( $k, i, j$ ), donde cada bucle itera  $N$  veces.
- El costo computacional total es directamente  $N^3$  operaciones lógicas.



## La Realidad

N	$N^3$ (Operaciones del núcleo)
1024	1,073,741,824 (~1.07 mil millones)
2048	8,589,934,592 (~8.6 mil millones)
4096	68,719,476,736 (~68.7 mil millones)

**Conclusión Clave:** Duplicar el tamaño de la matriz multiplica el tiempo de cómputo por ~8, un crecimiento insostenible.

# El código C implementa el triple bucle canónico con el atajo `A[i][k]` para mejorar la eficiencia.

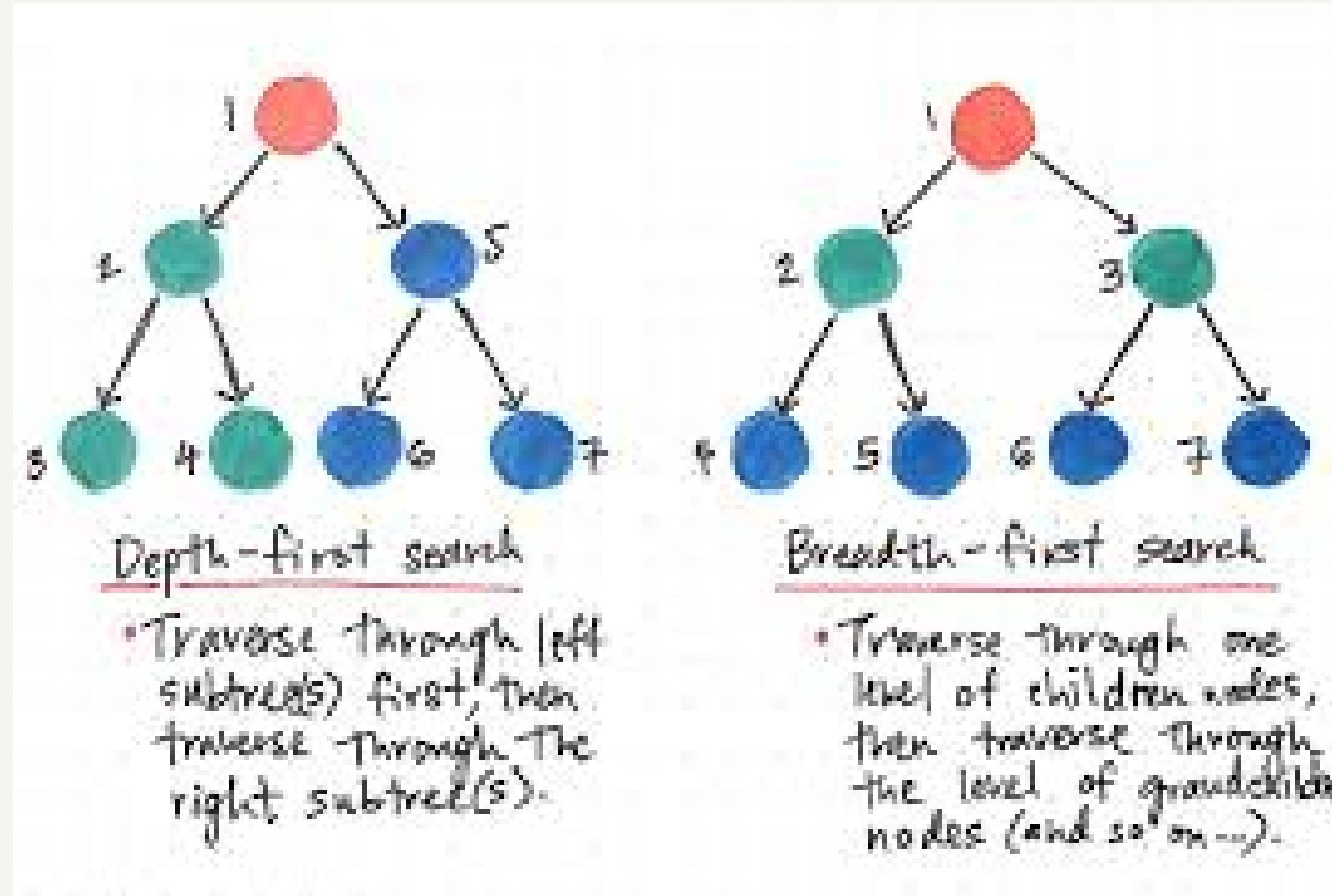
```
void warshall_logical(uint8_t* A, int N) {
    // A[i][j] = A[i][j] OR (A[i][k] AND A[k][j])
    // NO fuerza diagonal a 1.
    for (int k = 0; k < N; k++) {
        const uint8_t* row_k = &A[k * N];
        for (int i = 0; i < N; i++) {
            uint8_t aik = A[i * N + k];
            if (!aik) continue;
            uint8_t* row_i = &A[i * N];
            for (int j = 0; j < N; j++) {
                row_i[j] = (uint8_t)(row_i[j] | (aik & row_k[j]));
            }
        }
    }
}
```

$$T(N) = \sum_{k=0}^{N-1} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} O(1) = N \cdot N \cdot N \cdot O(1) = O(N^3).$$

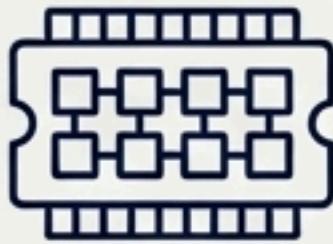
$$S(N) = O(N^2).$$

# Algoritmo de verificación implementado:

Cerradura transitiva por BFS (Breadth-First Search):



# Hemos desarrollado una implementación secuencial en C como base de referencia robusta y medible.



1. **Representación de Datos:** La matriz `A` se almacena como un bloque de memoria contiguo (`uint8_t*`) para optimizar el acceso a caché y facilitar la futura copia a la GPU.



2. **Reproducibilidad:** Se utiliza una semilla (`seed`) fija para generar las matrices iniciales, garantizando que las pruebas en CPU y GPU se realicen sobre los mismos datos. La densidad del grafo (`p`) es controlable.



3. **Medición Precisa:** El cronometraje (`clock_gettime`) se aísla estrictamente al núcleo computacional (el triple bucle), excluyendo la inicialización de datos y la E/S.



4. **Optimización Local:** Implementamos un "atajo" clave: si `A[i][k]` es 0, el bucle interno sobre `j` se omite, reduciendo significativamente el trabajo en matrices dispersas.

# Las mediciones del rendimiento secuencial confirman el cuello de botella cúbico y establecen nuestra línea base.

## Contexto de la Prueba

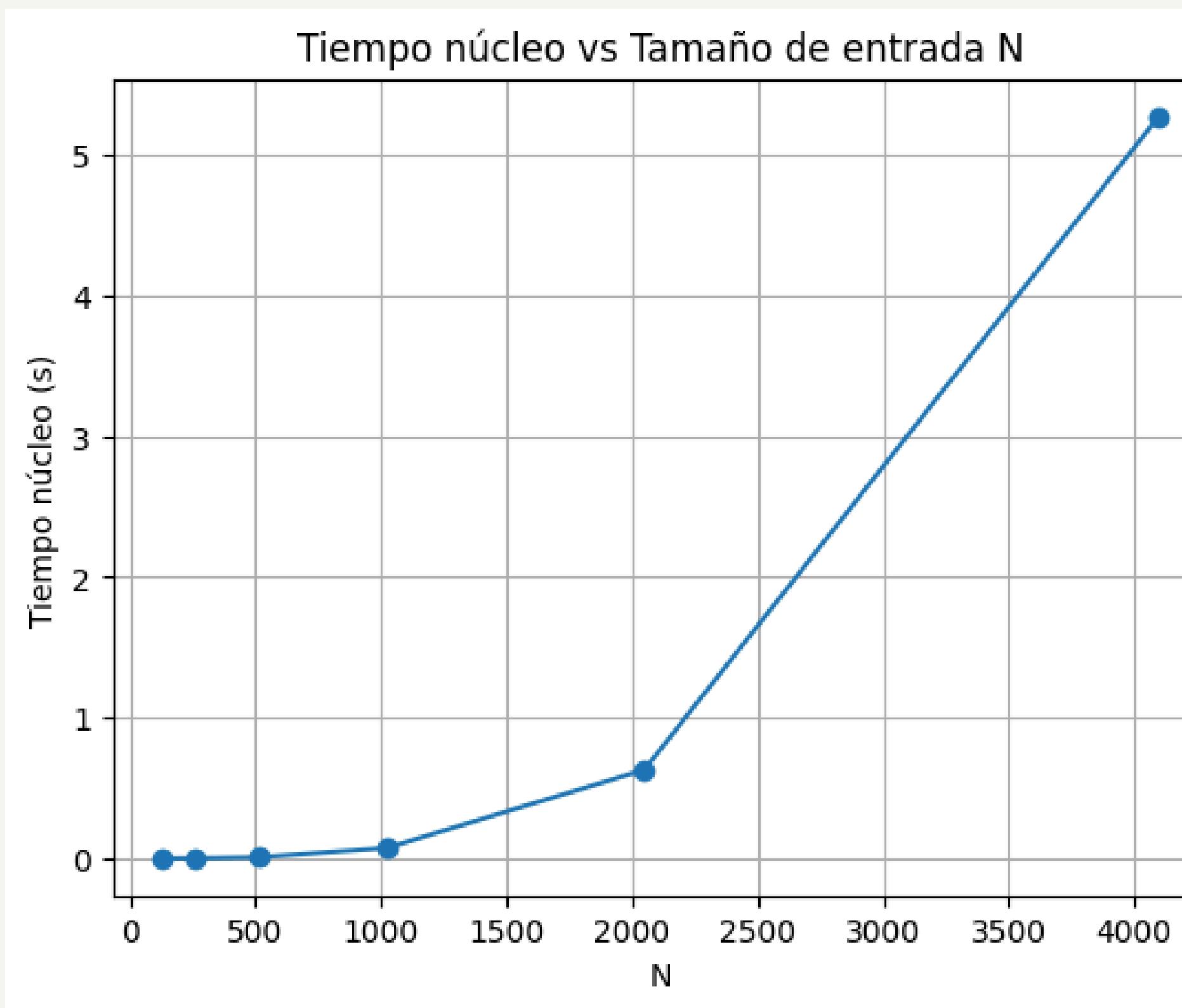
- **Compilador:** `gcc -O3 -std=c11`
- **Metodología:** Se ejecuta el programa con diferentes tamaños de N, midiendo únicamente el tiempo del núcleo computacional. `p` se ajusta para tamaños mayores para mantener tiempos de ejecución razonables.

N	N <sup>2</sup>	N <sup>3</sup>	p	Tiempo núcleo (s)
128	16,384	2,097,152	0.05	0.000062
256	65,536	16,777,216	0.05	0.000808
512	262,144	134,217,728	0.02	0.008243
1024	1,048,576	1,073,741,824	0.02	0.075686
2048	4,194,304	8,589,934,592	0.02	0.630069
4096	16,777,216	68,719,476,736	0.02	5.269098



**Observación Clave:** Se espera que el tiempo se multiplique por un factor cercano a 8 al duplicar N, validando el comportamiento O(N<sup>3</sup>).

# Gráfico de tiempo vs tamaño:

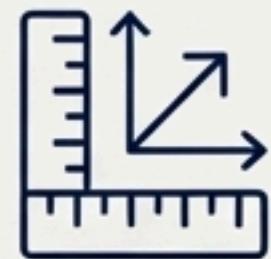


# Conclusión de la Fase 1: El análisis del algoritmo secuencial ha validado el desafío y ha dado como resultado un diseño de paralelización robusto para CUDA.



## Problema Cuantificado:

Se ha demostrado que el costo  $O(N^3)$  de Warshall es un cuello de botella real y medible para  $N \geq 1024$ .



## Línea Base Establecida:

Contamos con una implementación secuencial optimizada y verificada, que sirve como un punto de referencia de rendimiento justo y preciso.



## Plan de Acción Definido:

El diseño de paralelización para la Fase 2 está completo, especificando la estrategia de kernels, el mapeo de hilos y las optimizaciones clave de memoria.

## Siguiente Paso:

Final este Faso: Implementar y evaluar experimentalmente la solución CUDA diseñada, con el objetivo de lograr una aceleración sustancial sobre la base de referencia de la CPU.