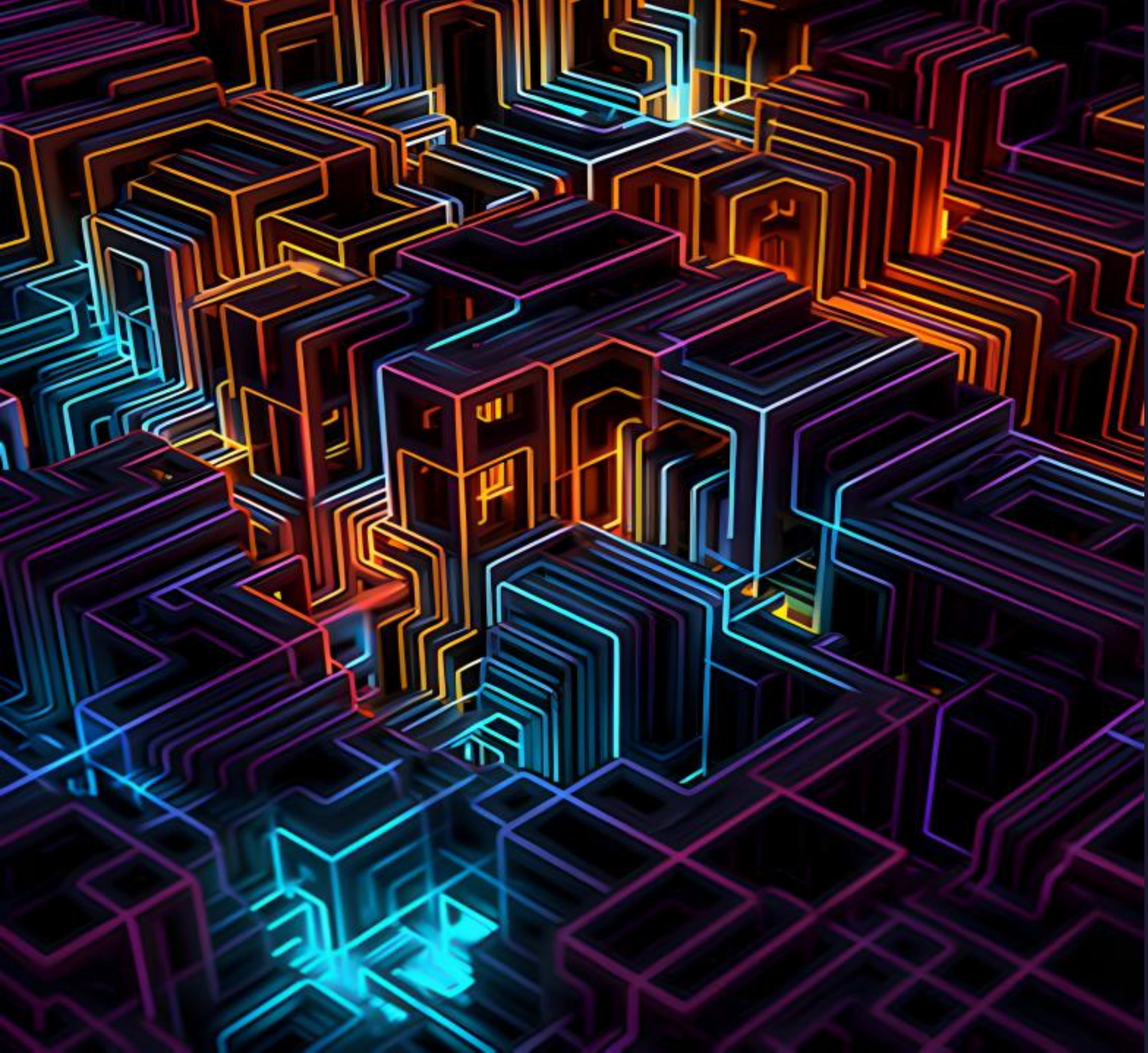


Algoritmos y problemas en grafos

Análisis y diseño de algoritmos avanzados

Dra. Valentina Narváez Terán

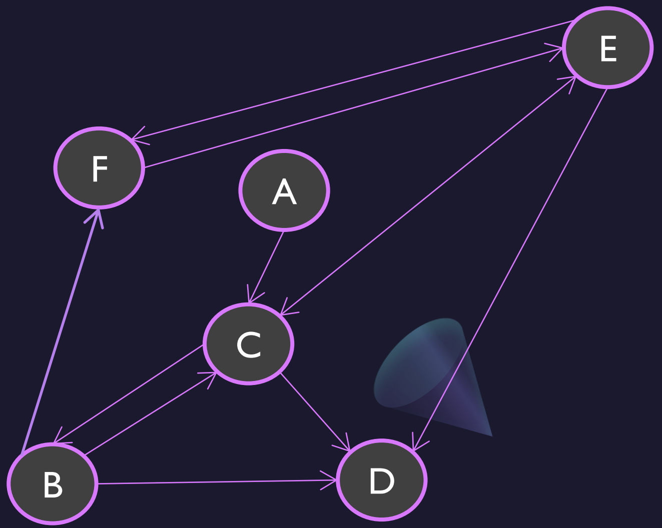
Tecnológico de Monterrey

Hay caminos y caminos más cortos

Caminos en grafos:
Dados un par de nodos, existe un camino entre ellos, si y solo si hay una secuencia de aristas que los conecte

Por ejemplo, B y D son conectados por { (B, F), (F, E), (E, D) }

	A	B	C	D	E	F
A			1			
B			1	1		1
C		1		1		
D						
E				1		1
F					1	



Caso trivial:
Es obvio que, si hay una arista dirigida de un nodo a otro, entonces existe un camino entre ellos, formado por esa única arista

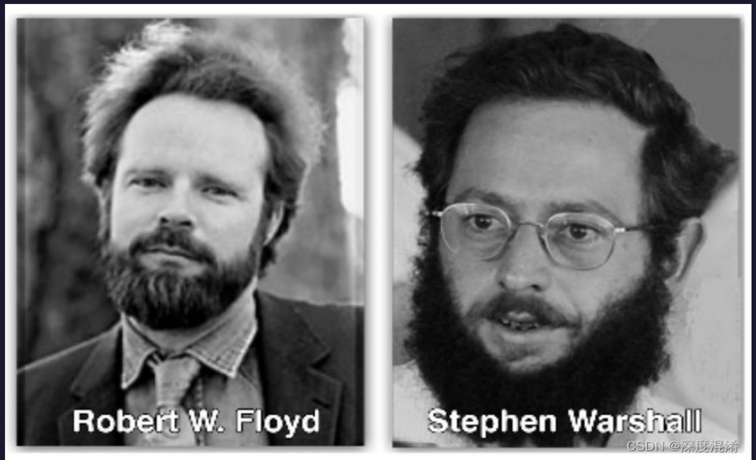
Pero ¿cómo saber si existe un camino entre cada par de vértices?
¿Puede calcularse a partir del caso trivial?

Hay caminos y caminos mas cortos

Warshall y Floyd crearon (mas o menos) el mismo algoritmo, mientras trataban de resolver problemas diferentes

Warshall: encontrar la cerradura transitiva (transitive closure) de un grafo dirigido

Floyd: encontrar los caminos mas cortos entre cada par de nodos (all-pairs shortest paths)



¿Transitive closure? Es una matriz binaria.
Si una celda $[i, j] = 1$ significa que existe un camino dirigido, de costo no-negativo, con origen en i y destino en j

¿Todos los caminos más cortos? ¿Cómo Dijkstra?
Casi. Dijkstra calcula el camino más corto desde un nodo fijo hacia a todos los demás

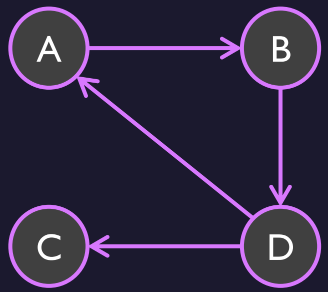
Para calcular el camino más corto entre cada par de nodos deberíamos aplicar Dijkstra n veces, una vez desde cada nodo

Algoritmo de Warshall

Matriz de adyacencia A

	A	B	C	D
A		1		
B				1
C				
D	1		1	

Grafo dirigido G



A partir de M se calculan otras matrices incrementalmente. Al final, la ultima matriz será la cerradura transitiva de G

Es programación dinámica, aunque Warshall y Floyd no lo llamaron así

El algoritmo se basa en la siguiente relación de recurrencia:

$$M_0 = A$$

$$M_k[i,j] = M_{k-1}[i,j] \text{ or } (M_{k-1}[i,k] \text{ and } M_{k-1}[k,j])$$

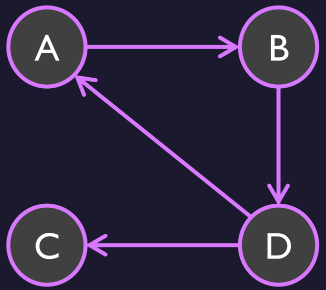
- M_0 es una copia de la matriz de adyacencia original A
- $M_1, M_2, \ldots, M_n$ serán otras matrices calculadas a partir de M_{k-1}
- La última matriz M_n será la cerradura transitiva
- En el código, podemos copiar A a M_0, y luego sobrescribir en M_0

Algoritmo de Warshall

Matriz de adyacencia A

	A	B	C	D
A		1		
B				1
C				
D	1		1	

Grafo dirigido G



	A	B	C	D
A		1		
B				1
C				
D	1		1	

M_0

	A	B	C	D
A		1		
B				1
C				
D	1	1	1	

Relación de recurrencia:

$$M_0 \; = A$$

$$M_k[i,j] = \underbrace{M_{k-1}\,[i,j]} \text{ or } \underbrace{(\, M_{k-1}[i,k] \text{ and } M_{k-1}[k,j] \,)}$$

M_1 tiene todos los 1's que tenga M_0

Todos esos $1's$ son caminos que ya conocíamos

Además, si en la matriz anterior (M_{k-1}) i se conecta con k, y k se conecta con j, entonces hay un camino de i a j

Hay un camino de D a A
Hay un camino de A a B
Entonces descubrimos que hay un camino de **D a B**

Algoritmo de Warshall

M = Matriz de adyacencia A

Para k = 0 hasta n
    Para $i = 0$ hasta n
        Para $j = 0$ hasta n
            $M[i,j] = M[i,j] \text{ or } (M[i,k] \text{ and } M[k,j])$

return la cerradura transitiva guardada en M

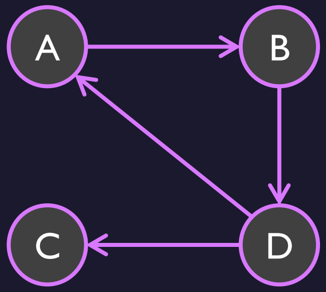
Complejidad? $O(n^3)$
Lo mismo que si aplicáramos n veces DFS
(DFS es $O(n^2)$)

Desventaja? La representación de matriz

Matriz de adyacencia A

	A	B	C	D
A		1		
B				1
C				
D	1		1	

Grafo dirigido G



Aplícalo a este ejemplo
¿Cómo resulta la matriz?

Algoritmo de Floyd

D = Matriz de adyacencia M, con 0's en la diagonal principal, e infinitos para nodos no-adyacentes

Para k = 0 hasta n
  Para $i = 0$ hasta n
    Para $j = 0$ hasta n
      $D[i,j] = \min(D[i,j], D[i,k] + D[k,j])$

return la matriz de costos de los caminos D

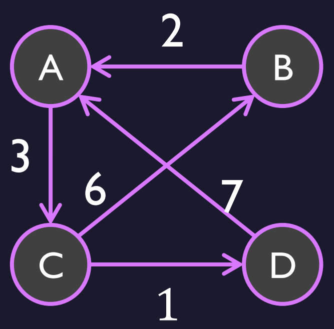
Complejidad?

La misma mecánica aplicada para saber si el camino existe se puede usar para acumular las longitudes del camino más corto entre cada par de vértices…

Matriz D

	A	B	C	D
A	0	∞	3	∞
B	2	0	∞	∞
C	0	6	0	1
D	7	∞	∞	0

Grafo dirigido G



Aplícalo a este ejemplo
¿Cómo resulta la matriz?

Algoritmo de Floyd

D = Matriz de adyacencia M, con 0's en la diagonal principal, e infinitos para nodos no-adyacentes

Para k = 0 hasta n
    Para $i = 0$ hasta n
        Para $j = 0$ hasta n
            $D[i,j] = \min(D[i,j], D[i,k] + D[k,j])$

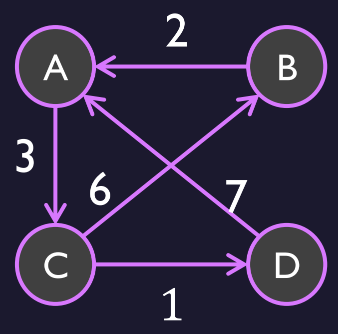
return la matriz de costos de los caminos D

La misma mecánica aplicada para saber si el camino existe se puede usar para acumular las longitudes del camino mas corto entre cada par de vértices…

Matriz D

	A	B	C	D
A	0	∞	3	∞
B	2	0	∞	∞
C	0	6	0	1
D	7	∞	∞	0

Grafo dirigido G

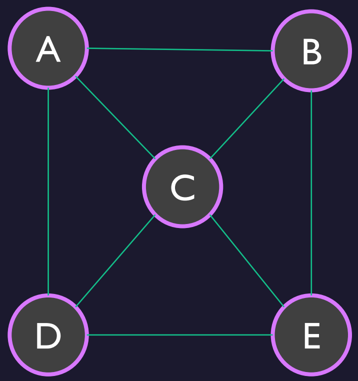


Complejidad? $O(n^3)$
Lo mismo que si aplicáramos n veces Dijkstra
(Dijkstra es $O(n^2)$)

Aplícalo a este ejemplo
¿Cómo resulta la matriz?

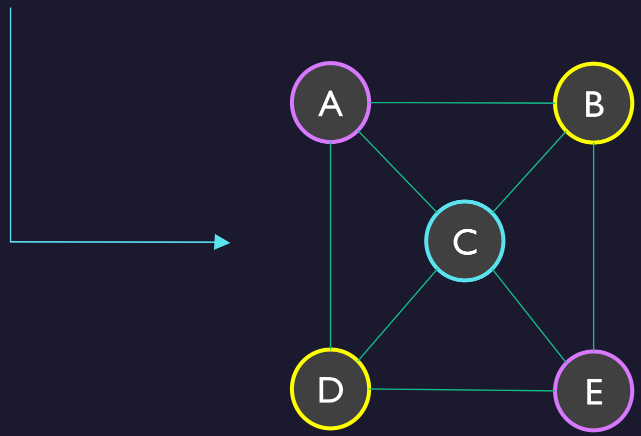
Otro tema: coloreado de grafos

Grafo G



Matriz M

	A	B	C	D	E
A		1	1	1	
B	1		1		1
C	1	1		1	1
D	1		1		1
E		1	1	1	



El problema: Dado un grafo no-dirigido ¿Cuál es la cantidad mínima k de colores que se puede usar para que cada par de nodos adyacentes tenga distinto color?

Es un problema de minimización en optimización combinatoria

Su versión en problema de decisión es:
¿Se puede conseguir colorear los nodos con k colores?

Es NP-Complete

¿Para que sirve?
Visualizaciones, coloreado de mapas, problemas de infraestructura…

Coloreado de grafos: método greedy básico

Crea una paleta de n colores distintos (no importa cuales)
Numera los colores de 0 a $n-1$
$k = 0$

Por cada nodo u:

  Asigna el color con el número mas pequeño, que no se haya usado para ningún vecino de u

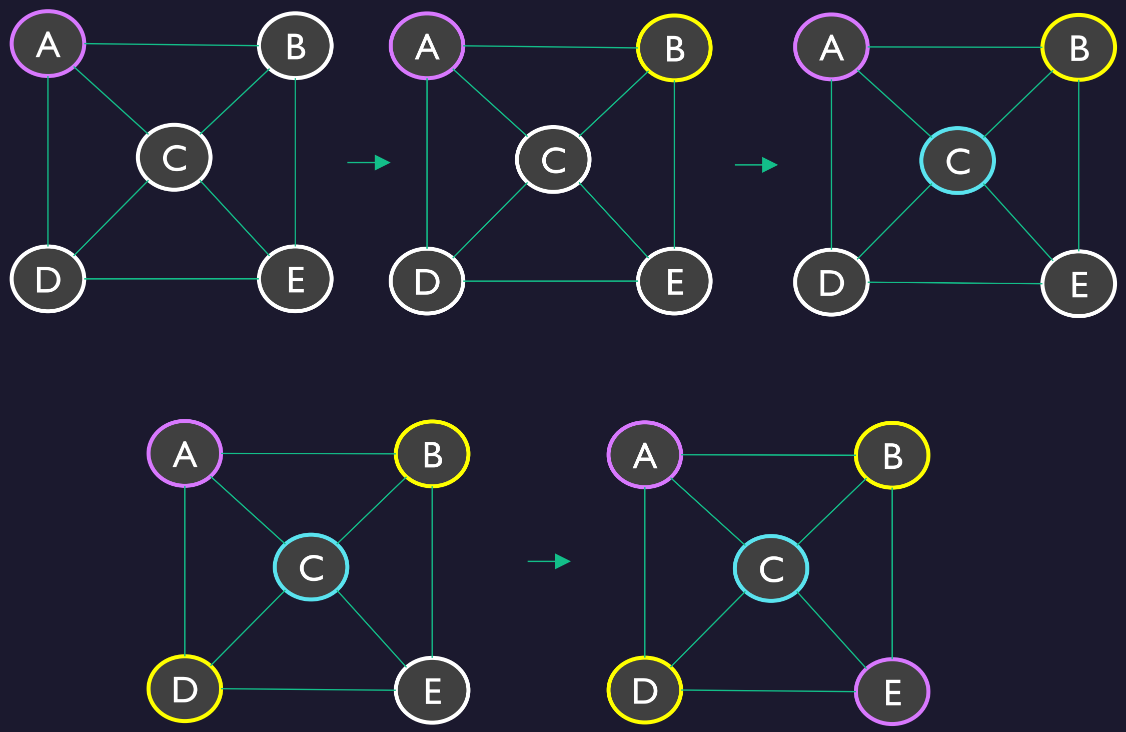
  Si es un color que no se había usado antes

    $k++$

Devuelve k

$$paleta = [0\!: morado, 1\!: amarillo, 2\!: azul, 3\!: verde, 4\!: naranja]$$

Grafo G, inicialmente, ningún nodo tiene color.
Si procesamos los nodos en orden alfabético:



Para este ejemplo, $k = 3$

Greedy mejorado: Algoritmo Welsh-Powell

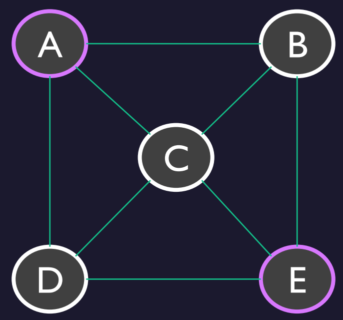
Ordena los nodos según su grado (número de vecinos)

k = 0
Por cada nodo, según el orden
    u = el nodo no-procesado con grado más pequeño
    Asigna a u el color k y marca u como procesado

    Para cada nodo j que no es vecino de u
        Asigna a j el mismo color que a u
    $k++$

$paleta = [0\colon morado, 1\colon amarillo,$
$2\colon azul, 3\colon verde, 4\colon naranja]$

$u = A$



A y todos sus no-vecinos (E) tendrán el color 0

nodo	grado	procesado	color
A	**3**	**1**	**0**
B	3		
D	3		
E	3		0
C	4		

Greedy mejorado: Algoritmo Welsh-Powell

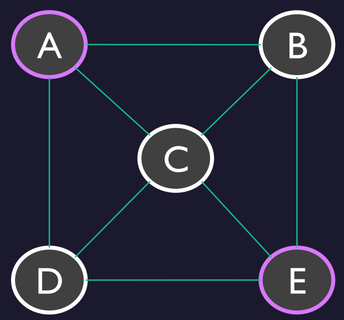
Ordena los nodos según su grado (numero de vecinos)

k = 0
Por cada nodo, según el orden
    u = el nodo no-procesado con grado mas pequeño
    Asigna a u el color k

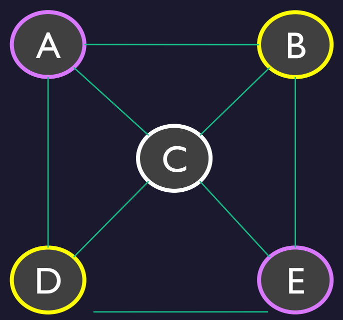
    Para cada nodo j que no es vecino de u
        Asigna a j el mismo color que a u
    $k++$

$$paleta = [0{:}\ morado, 1{:}\ amarillo, 2{:}\ azul, 3{:}\ verde, 4{:}\ naranja]$$

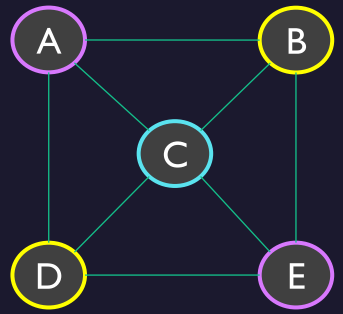
$u = A$



$u = B$



$u = C$



nodo	grado	procesado	color
A	3	1	0
B	3	1	1
D	3	1	1
E	3	1	0
C	4	1	2