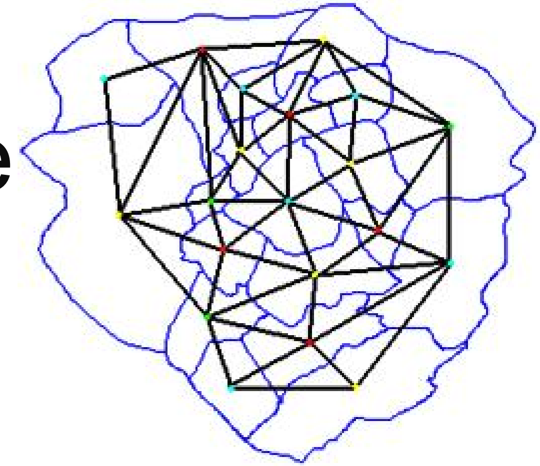
TEMA:

Problema de los caminos de costo mínimo entre <u>cualquier</u> par de vértices:

Algoritmos de *Floyd* y *Warshall*

Ejemplo que ilustra el tipo de problema:



Supónga que se tiene un grafo dirigido ponderado, que da el tiempo de vuelo para ciertas rutas entre ciudades, y se desea construir una tabla que brinde el menor tiempo requerido para volar entre dos ciudades cualesquiera.

Planteamiento del problema

Grafo **dirigido** y **ponderado** ***G = (V, A)*** con función de costo definida sobre sus arcos

Los **arcos pueden tener costos negativos**, pero en el grafo **no deben existir ciclos de costo negativo**

NOTA: A partir de pequeñas transformaciones (ejercicio de CP) los algoritmos que se propondrán pueden detectar la existencia de ciclos de costo negativo en el Grafo

PROBLEMA: encontrar el **camino de costo mínimo** entre ***v*** y ***w***, para cada par ordenado de vértices (***v***, ***w***)

Una solución trivial al problema:

Usar el algoritmo de Dijkstra (<u>si todos los costos son no negativos</u>):

- Considerar cada vértice de V, como el vértice origen

- Con esta variante, el orden del algoritmo es $O(|V|^3)$, si se usa la versión de Dijkstra que tiene orden $O(|V|^2)$. Si se usa la versión de Dijkstra que usa Heap, entonces el costo sería $O(|V|*|E|*\log(V))$.

Una forma más directa de solución para este problema:

El algoritmo de R. W. Floyd

•Se supone que los vértices de V están numerados de la siguiente forma: $V = \{1,2,\ldots,n\}$

•Se usa una matriz A, de orden $n \times n$, sobre la que se calculan los costos de los caminos de costo mínimo

Inicialización:

•$A[i, j] = c[i, j] \ \forall\, i \neq j,\ 1 \leq i,j \leq |V|,$

•Si no existe el arco (i, j), se asume $c[i, j] = \infty$

• $\forall i,\ 1 \leq i \leq |V|,\ A[i, i] = 0$ (elementos de la diagonal)

- Se hacen *n* iteraciones sobre la matriz A

- Al final de la *k*–ésima iteración, el valor de A[i, j] es el costo de un camino de costo mínimo de *i* a *j* que <u>no pasa por vértices</u> de número mayor que *k*

- NOTA: los vértices extremos del camino, *i* e *j*, pueden ser cualesquiera, pero todo vértice intermedio en el camino, debe ser $\leq k$

Procedimiento recursivo:

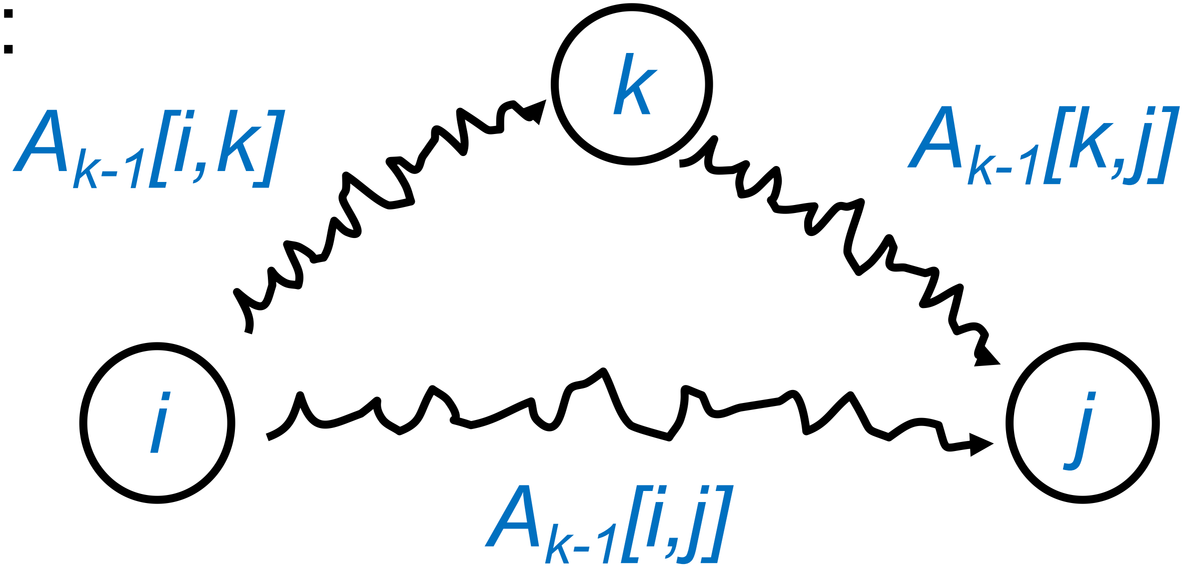
En la k-ésima iteración se aplica la siguiente fórmula para calcular *A*

$$A_k[i,j] = mín \begin{cases} A_{k-1}[i,j] \\ A_{k-1}[i,k] + A_{k-1}[k,j] \end{cases}$$

El subíndice *k*, denota el valor de la matriz *A* después de la *k*-ésima iteración.

Nota: Esto NO indica la existencia de n matrices distintas, o sea, NO indica: $M_{1} \ldots M_{k} \ldots M_{n}$

La siguiente figura ilustra lo que expresa la fórmula:



Para obtener $A_k[i,j]$:

comparar :

•$A_{k-1}[i,j]$ *(el costo de ir de i a j sin pasar por k o cualquier otro vértice con numeración mayor)*

con

•$A_{k-1}[i,k] + A_{k-1}[k,j]$ *(el costo de ir primero de i a k y después de k a j, sin pasar a través de un vértice con número mayor que k)*

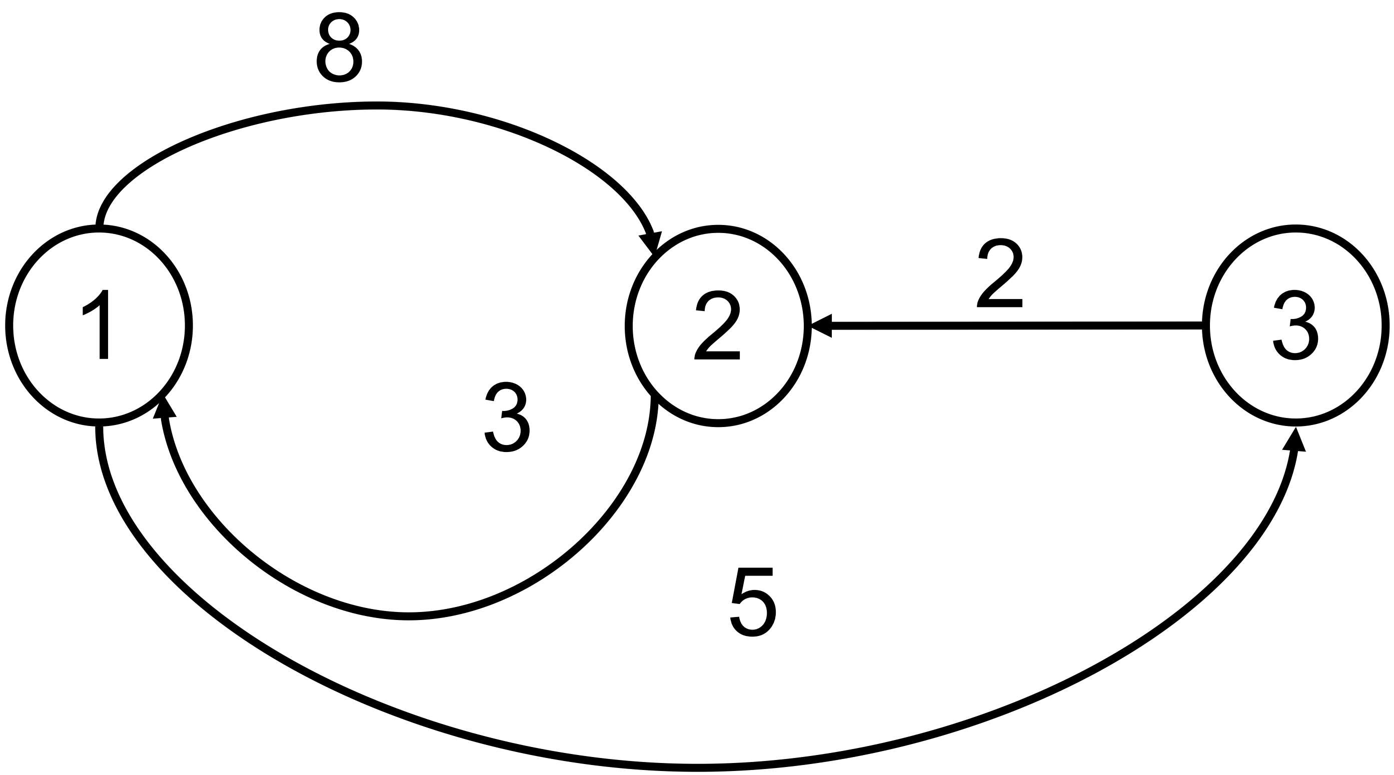
La interpretación de la fórmula es la siguiente:

Verificar si el costo del camino se mejora pasando por el vértice *k*.

En tal caso, se actualiza con el valor de los dos costos en los que se involucra a *k*

Ejemplo:

Sea el siguiente grafo, dirigido y ponderado



Las siguientes figuras muestran los valores iniciales de la matriz *A* y los que adquiere después de tres iteraciones

$$\begin{array}{c|ccc|} & 1 & 2 & 3 \\ \hline 1 & 0 & 8 & 5 \\ 2 & 3 & 0 & \infty \\ 3 & \infty & 2 & 0 \\ \hline \end{array} \qquad \begin{array}{c|ccc|} & 1 & 2 & 3 \\ \hline 1 & 0 & 8 & 5 \\ 2 & 3 & 0 & 8 \\ 3 & \infty & 2 & 0 \\ \hline \end{array} \qquad \begin{array}{c|ccc|} & 1 & 2 & 3 \\ \hline 1 & 0 & 8 & 5 \\ 2 & 3 & 0 & 8 \\ 3 & 5 & 2 & 0 \\ \hline \end{array} \qquad \begin{array}{c|ccc|} & 1 & 2 & 3 \\ \hline 1 & 0 & 7 & 5 \\ 2 & 3 & 0 & 8 \\ 3 & 5 & 2 & 0 \\ \hline \end{array}$$

$$A_0[i,j] \qquad\qquad A_1[i,j] \qquad\qquad A_2[i,j] \qquad\qquad A_3[i,j]$$

Note que: En la *k-ésima* iteración

$A_k[i,k] = A_{k-1}[i,k]$ y $A_k[k,j] = A_{k-1}[k,j]$,

Por tanto, ninguna entrada con cualquiera de sus subíndices (*el* *i* *o el* *j*) igual a *k*, cambia durante la *k-ésima* iteración.

Por tanto, TODA la ejecución del algoritmo se realiza con una sola copia de la matriz *A*

Justificación de lo que se acaba de expresar:

En cada caso en que este involucrado el índice *k* en algún *A[i,j]*, el valor del mismo no tiene la posibilidad de ser actualizado pues intervine en la suma algún *A[i, i]* que tiene valor cero y por tanto, <u>el mínimo queda entre dos valores iguales</u>

Ejemplo: En la 2da. Pasada (k=2) el valor del elemento

$$A_2[2, 1] = mín(A_1[2, 1], (A_1[2, 2] = 0 + A_1[2, 1]))$$
$$A_2[2, 1] = mín(A_1[2, 1], A_1[2, 1])$$
$$A_2[2, 1] = A_1[2, 1]$$

A continuación se muestra el Algoritmo de "Floyd", donde se realiza el cálculo con matrices de *n x n*

El algoritmo calcula la matriz *A*, de caminos de costo mínimo, sobre la matriz de costos *C*

Floyd (**variable A**: arreglo de n x n de enteros)
C: arreglo de n x n de reales

i, j, k: enteras;

para **i = 1, . . ., n** **hacer**
para **j = 1, … , n** **hacer**
C[i, j] → A[i, j] si existe arco (i,j)
∞ en caso contrario ;

para **i = 1, . . ., n** **hacer**
0 → A[i, i];
para **k = 1, . . ., n** **hacer**
para **i = 1, . . ., n** **hacer**
para **j = 1, . . ., n** **hacer**
si **(** A[i,k] + A[k,j] **)** < A[i,j] **entonces**
(A[i,k] + A[k,j] **)** → A[i,j]

`

El tiempo de ejecución de este algoritmo es *$O(n^3)$* debido al triple ciclo anidado

final

Algoritmo de Floy-Warshall (I to A)

FLOYD-WARSHALL(W)

$n \leftarrow rows[W]$

$D^{(0)} \leftarrow W$

for $k \leftarrow 1$ **to** n

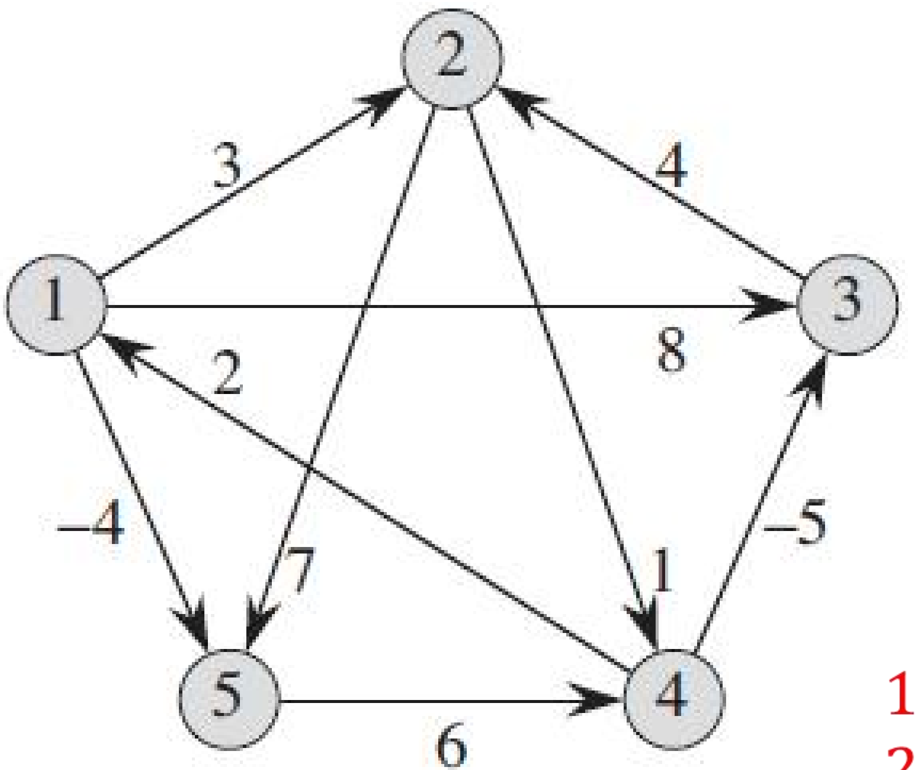
    do for $i \leftarrow 1$ **to** n

        do for $j \leftarrow 1$ **to** n

            do

return $D^{(n)}$

$$d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, \quad d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$$



$D^{(0)}$

	1	2	3	4	5
1	0	3	8	∞	−4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	−5	0	∞
5	∞	∞	∞	6	0

$\Pi^{(0)}$

	1	2	3	4	5
1	*null*	1	1	*null*	1
2	*null*	*null*	*null*	2	2
3	*null*	3	*null*	*null*	*null*
4	4	*null*	4	*null*	*null*
5	*null*	*null*	*null*	5	*null*

Comparación entre los algoritmos de Dijkstra y Floyd

- Para lograr lo mismo que hace Floyd, se aplica el algoritmo de Dijkstra (cuando los costos son no negativos) ***n*** veces, es decir, asumiendo en cada caso a un vértice diferente como origen. Tanto en Floyd como en Dijkstra el problema se resuelve en $O(n^3)$

- Se ha comprobado en la práctica que *Dijkstra* no es mejor que *Floyd* , pues hace más operaciones y por tanto, la constante es mayor

Correctitud del algoritmo de Floyd

Para verificar que este programa funciona, es fácil demostrar por inducción sobre *k* que después de *k* recorridos por el triple ciclo, *A[i, j]* contiene el costo de un camino de costo mínimo entre el vértice *i* y el vértice *j* que no pasa a través de un vértice con número mayor que *k*

Correctitud del algoritmo de Floyd

(*) Demostrar por inducción sobre *k* que después de *k* recorridos por el triple ciclo, $A_k[i, j]$ contiene el costo de un camino de costo mínimo entre *i* y *j* que no pasa a través de un vértice con número mayor que *k*

caso base, $k=0$:

$A_0[i, j]=A[i,j]$ es cierto que no hay ningún camino que une a *i* con *j*, con vértices intermedios de número mayor que *0*, es más, en estos momentos, los únicos caminos que serán asumidos en la matriz son sólo los que impliquen llegar de un vertice a otro a través de un arco, o sea, la única manera de llegar de *i* a *j*, hasta este momento, es a través del arco *(i, j)*

Pasada *k*: Supongamos que hasta la pasada *k*, $A_k[i, j]$ contiene el costo de un camino de costo mínimo entre *i* y *j* que no pasa a través de un vértice con número mayor que *k*

Pasada k+1:

Por construcción y teniendo en cuenta como se actualizaran los valores de la matriz en el paso k+1 a partir de los resultados obtenidos en el paso k, podemos asegurar que siempre se cumplirán I y II:

I. $A_{k+1}[i, j] \leq A_k[i, j]$

II. $A_{k+1}[i, j] \leq A_k[i, k+1] + A_k[k+1, j]$

por ser $A_{k+1}[i, j] = min \{A_k[i, j], A_k[i, k+1] + A_k[k+1, j]\}$

Negemos la tesis (*):

Al concluir la pasada k+1, supongamos que existe un camino $c=\{i \ldots v_p \ldots v_p \ldots j\}$ de costo menor que $A_{k+1}[i,j]$ con vértices intermedios de número menor o igual que k+1

$w(c) = w(i\ldots v_p) + w(v_p\ldots v_p) + w(v_p\ldots j) \geq w(i\ldots v_p) + 0 + w(v_p\ldots j) = w$

(camino simple obtenido)

Esto se puede hacer con todos los ciclos que existan en el Grafo

Justificación:

Como no existen ciclos de costo negativo, el costo del ciclo se puede acotar por 0, o sea, lo que se hace es "*cortar*" todos los ciclos para hacer los razonamientos a partir de caminos simples

Luego, podemos asumir que hay un camino simple cuyo costo es menor que $A_{k+1}[i, j]$ con vértices intermedios de número menor o igual que $k+1$

Sea este $(i, c_1\ldots c_q, j)$ con $c_s \leq k+1$ para todo $s \in \{1\ldots q\}$

Caso 1: NO HAY NINGUN Cs = k+1

$c_s < k+1 \ \forall s \in \{1...q\}$

$A_{k+1}[i, j] > w(i,c_1...c_q,j) \geq A_k[i,j]$ por hipótesis de inducción.

$\Rightarrow$ $\boxed{A_{k+1}[i,j] > A_k[i,j]}$
Contradicción con (I).

Caso 2:

$\exists\ r : c_r = k+1$ (este es único por ser camino simple), luego $c_s \leq k$ $\forall\ s \neq r,\ s \in \{1...q\}$.

$A_{k+1}[i,j] > w(i,c_1...k+1...c_q,j) = w(i...k+1) + w(k+1...j) \geq$

según lo supuesto

$\geq A_k[i,k+1] + A_k[k+1,j]$ por hipótesis de inducción.

$\Rightarrow$ $\boxed{A_{k+1}[i,j] > A_k[i,k+1] + A_k[k+1,j]}$
Contradicción con (II).

Entonces no existe un camino de costo menor que $A_{k+1}[i,j]$ con vértices intermedios de número menor o igual que k+1.

Hasta ahora hemos probado que costo mas chiquito que ese NO HAY

Faltaría por demostrar que existe ese camino cuyo costo es $A_{k+1}[i,j]$.

$A_{k+1}[i,j]$ guardará el costo del camino anterior, o sea el que habia sido calculado en el paso k para:

1. .... ir de i a j, si se cumple $A_k[i,j] \leq A_k[i,k+1]+A_k[k+1,i]$, o sea me quedo con el mismo que tenia y en tal caso, por hipotesis de induccion para el paso k, queda demostrada la existencia del camino.

2. .... ir de i a k+1 unido con el que teníamos de k+1 a j, en caso contrario, ambos calculados en el paso k y por la propia hipotesis de induccion supuesta para dicho paso, se cumplira tambien su existencia.

q.e.d.

Clausura transitiva - Algoritmo de Warshall

- En algunos problemas podría ser interesante saber solo si existe un camino de longitud ≥ 1 entre el vértice i y el vértice j.

- El algoritmo de Floyd puede especializarse para este problema y el algoritmo resultante, que **antecede** al de Floyd, se conoce como algoritmo de **Warshall**

Supóngase que la matriz de costo C es solo una *Matriz de Adyacencia* para el grafo dirigido dado

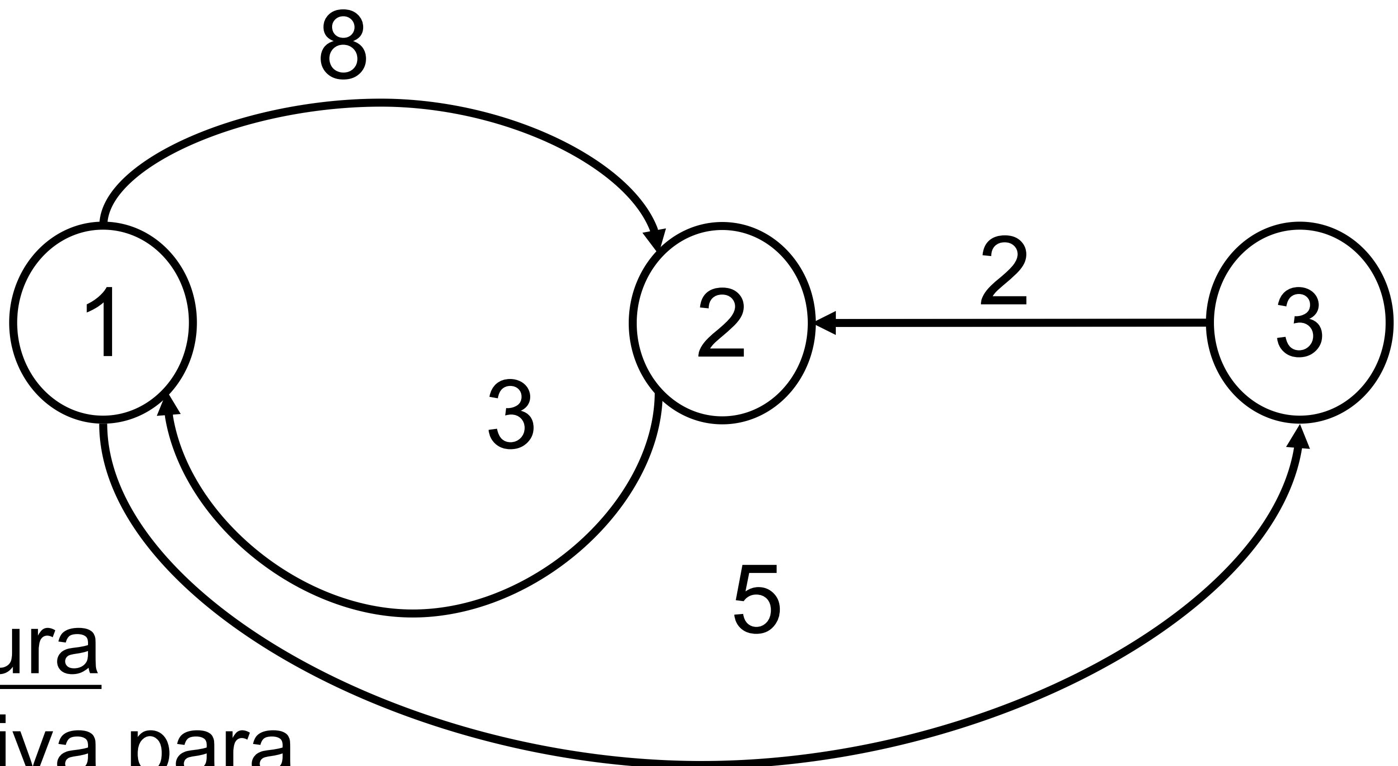
C[i, j] = 1 → si hay un arco de i a j

C[i, j] = 0 → si no hay un arco de i a j

Se desea obtener la matriz A,

A[i, j] = 1 → si hay un camino de i a j
A[i, j] = 0 → si no hay un camino de i a j

A se conoce a como cerradura o *Clausura Transitiva* de la *Matriz de Adyacencia*.



<u>Clausura transitiva</u> para la matriz de adyacencia del grafo

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

La clausura transitiva puede obtenerse con un procedimiento similar a Floyd aplicando la siguiente fórmula en el k-ésimo paso de la matriz booleana A

$A_k[i,j]$ = $A_{k-1}[i,j]$ **o (** $A_{k-1}[i,k]$ **y** $A_{k-1}[k,j]$ **)**

Esta fórmula establece que hay un camino de i a j con vértices intermedios en 1, 2, ….,k-1, si:

- ya existe un camino de i a j que no pasa por un vértice con número mayor que k-1

"O"

- hay un camino de i a k que no pasa por un vértice con número mayor que k-1

"Y"

un camino de k a j que no pasa por un vértice con número mayor que k-1

Igual que antes,

$A_k[i,k] = A_{k-1}[i,k]$ y $A_k[k,j] = A_{k-1}[k,j]$

así que se puede realizar el cálculo con solo una copia de la matriz A

```
Warshall ( variable A: arreglo de n x n de enteros )
        C: arreglo de n x n de “booleans”

        i, j, k: enteras;

        para    i = 1, . . ., n    hacer
                para    j = 1, … , n    hacer
                        C[i, j] → A[i, j];

        para    k = 1, . . ., n    hacer
                para    i = 1, . . ., n    hacer
                        para    j = 1, . . ., n    hacer
                                si A[i,j] = false entonces
                                        A[i,k] AND A[k,j] → A[i,j]
```