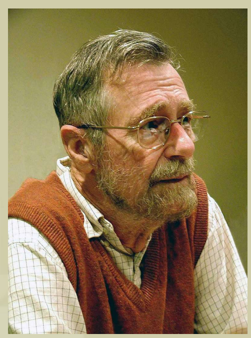
Tecnológico de Monterrey.

TC2038. Análisis y diseño de Algoritmos A

M.C. Ramona Fuentes Valdéz

rfuentes@tec.mx

Grafos



Algoritmo de Dijkstra

Edsger W. Dijkstra (1930-2002)

- ✓ Físico teórico. Programador en el Mathematisch Centrum.
- ✓ Profesor de la Cátedra Centenario de Schlumberger en Ciencias de la Computación, UT Austin.
- ✓ Premio Turing.

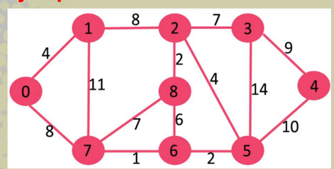
- ✓ Una de las figuras más influyentes de la generación fundadora de la ciencia de la computación.
- ✓ Sus contribuciones fundamentales cubren diversas áreas de la ciencia de la computación, incluida la construcción de compiladores, sistemas operativos, sistemas distribuidos, programación secuencial y concurrente, paradigma y metodología de programación, investigación de lenguajes de programación, diseño de programas, desarrollo de programas, verificación de programas, principios de ingeniería de software, algoritmos gráficos, y fundamentos filosóficos de la programación y la informática.
- ✓ Muchos de sus trabajos son fuente de nuevas áreas de investigación.
- ✓ Varios conceptos y problemas que ahora son un estándar en la informática fueron identificados por primera vez por Dijkstra o llevan nombres acuñados por él.

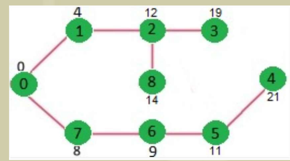
Grafos

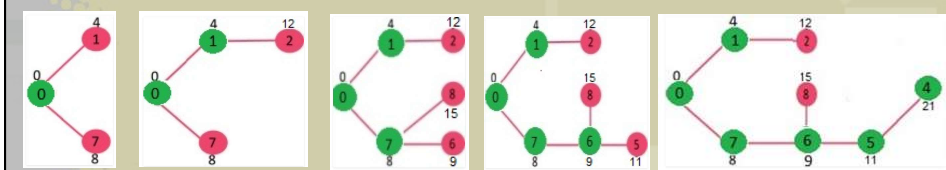
Algoritmo de Dijkstra

- Es útil cuando todos los pesos son positivos.
- Proporciona las rutas más cortas desde un vértice origen a todos los demás vértices del grafo.
  - Puede concluir una vez que se encuentra el camino más corto al nodo elegido.

Ejemplo:







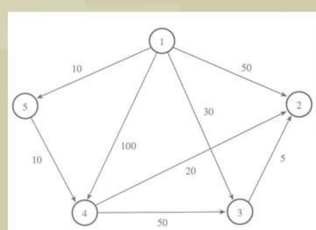
Grafos

Algoritmos

Algoritmo voraz de Dijkstra

```
función Dijkstra(L[1..m,1..n]):matriz[2..n]
  matriz D[2..n]
  {iniciación}
  C←{2,3,…,n} {S=N\C sólo existe implícitamente}
  {bucle voraz}
  para i←2 hasta n hacer
    D[i]=L[1,i]
  {bucle voraz}
  repetir n-2 veces
    v ← algún elemento de C que minimiza D[v]
    C ← C\{v} {e implícitamente S←S ∪ {v}}
    para cada w ∊ C hacer
       D[w] ← mín(D[w],D[v] + L[v,w])
  fin_repetir
  devolver D
fin_función
```

Ejemplo:



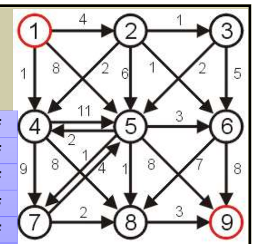
Paso	v	C	D
i	-	{2,3,4,5}	[50, 30, 100, 10]
1			[, , ,]
			[, , ,]
			[, , ,]
			[, , ,]

Grafos

Algoritmo de Dijkstra

Ejemplo:

Considera el siguiente grafo con pesos y ciclos positivos.

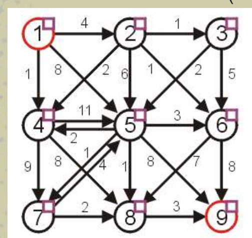


Análisis:

En un primer intento de resolver este problema, se podría requerir una matriz de valores booleanos, inicialmente todos falsos, que indiquen si hemos encontrado una ruta desde el origen.

1	F
2	F
3	F
4	F
5	F
6	F
7	F
8	F
9	F

Gráficamente, se incluirán casillas de verificación junto a cada uno de los vértices (*inicialmente sin marcar*)



Se trabajará de abajo hacia arriba.

- Tener en cuenta que si el vértice inicial tiene aristas adyacentes, habrá un vértice que sea la distancia más corta desde el vértice inicial. Este es el vértice más corto alcanzable del grafo.

Después, se intentará extender cualquier camino existente a nuevos vértices.

Inicialmente, se comenzará con el camino de longitud 0.

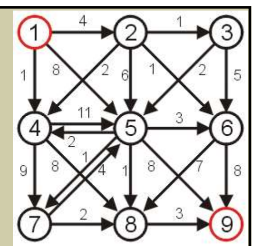
- este es el camino simple desde el vértice 1 a sí mismo.

Grafos

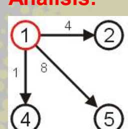
Algoritmo de Dijkstra

Ejemplo:

Considera el siguiente grafo con pesos y ciclos positivos.



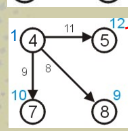
Análisis:



Si ahora ampliamos este camino, deberíamos considerar los caminos

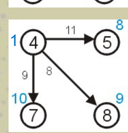
- (1, 2) longitud 4
- (1, 4) longitud 1
- (1, 5) longitud 8

El camino más corto hasta ahora es (1, 4) que tiene una longitud de 1.



Debemos recordar que la longitud de ese camino desde el nodo 1 al nodo 4 es 1.
Por lo tanto, necesitamos almacenar la longitud de una ruta que pasa por el nodo 4:

- 5 de longitud 12
- 7 de longitud 10
- 8 de longitud 9

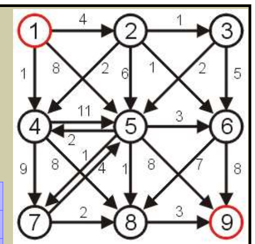


Ya hemos descubierto que hay un camino de longitud 8 al vértice 5 con el camino (1, 5).
Por lo tanto, se puede ignorar con seguridad el camino más largo.

Ahora sabemos que:
Existen caminos desde el vértice 1 hasta los vértices {2,4,5,7,8}.
Sabemos que el camino más corto desde el vértice 1 al vértice 4 es de longitud 1.
Sabemos que el camino más corto a los otros vértices {2,5,7,8} es como máximo la longitud indicada en la tabla de la derecha.

Vértice	Longitud
1	0
2	4
4	1
5	8
7	10
8	9

Grafos



Algoritmo de Dijkstra

Ejemplo:

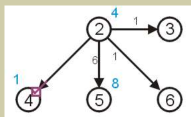
Considera el siguiente grafo con pesos y ciclos positivos.

Análisis:

No puede existir una ruta más corta a ninguno de los vértices 1 o 4, ya que las distancias solo pueden aumentar en cada iteración.
Consideramos estos vértices como **visitados**.

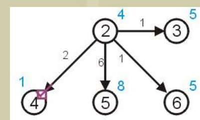
Vértice	Longitud
1	**0**
2	4
4	**1**
5	8
7	10
8	9

En el algoritmo de Dijkstra, siempre se toma el siguiente vértice no visitado que tiene el camino más corto actual desde el vértice inicial en la tabla.
→ Este es el vértice 2



Se puede intentar actualizar las rutas más cortas a los vértices 3 y 6 (*ambos de longitud 5*):

- Existe un camino de longitud 8 <10 al vértice 5 (10 = 4 + 6).
- Se sabe que el camino más corto a 4 es 1.



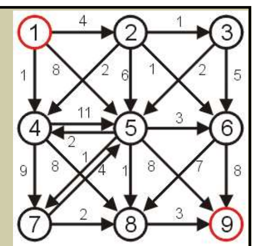
Para el seguimiento de los vértices a los que no ha llegado ningún camino, se les puede asignar una distancia inicial de:

- infinito (∞),
- un número mayor que cualquier camino posible
- un número negativo

Para este caso, elegiremos utilizar ∞

Además de encontrar la longitud del camino más corto, nos gustaría encontrar el camino más corto correspondiente.
Cada vez que actualizamos la distancia más corta a un vértice en particular, se realizará un seguimiento del predecesor utilizado para alcanzar este vértice en la ruta más corta

Grafos



Algoritmo de Dijkstra

Ejemplo:

Considera el siguiente grafo con pesos y ciclos positivos.

Análisis:

Además de encontrar la longitud del camino más corto, nos gustaría encontrar el camino más corto correspondiente.

Cada vez que actualizamos la distancia más corta a un vértice en particular, se realizará un seguimiento del predecesor utilizado para alcanzar este vértice en la ruta más corta.

Almacenaremos una tabla de apuntadores, cada uno inicialmente 0.
Esta tabla se actualizará cada vez que se actualice una distancia.

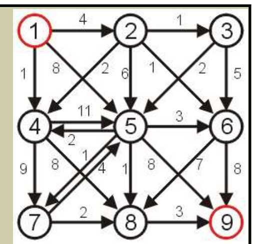
1	**0**
2	**0**
3	**0**
4	**0**
5	**0**
6	**0**
7	**0**
8	**0**
9	**0**

Gráficamente, mostraremos la referencia al vértice anterior mediante una flecha roja.

- si la distancia a un vértice es ∞, no habrá vértice precedente
- de lo contrario, habrá exactamente un vértice precedente

Para implementarse…

Grafos



Algoritmo de Dijkstra

Ejemplo:

Considera el siguiente grafo con pesos y ciclos positivos.

Implementación:

Inicialización

- se establece la distancia actual al vértice inicial como 0.
- para todos los demás vértices, se establece la distancia actual en ∞.
- todos los vértices se marcan inicialmente como no visitados.
- establecer el apuntador para todos los vértices en nulo.

Iteración

- encontrar un vértice no visitado que tenga la distancia más corta a él.
- marcarlo como visitado.
- para cada vértice no visitado que sea adyacente al vértice actual:
  - sumar la distancia al vértice actual al peso de la arista que lo conecta.
  - si es menor que la distancia actual a ese vértice, actualiza la distancia y establece el vértice padre del vértice adyacente para que sea el vértice actual.

Condición de detención

- nos detenemos con éxito cuando el vértice que estamos visitando es el vértice objetivo.
- si en algún punto, todos los vértices no visitados restantes tienen una distancia de ∞, entonces no hay camino desde el vértice inicial hasta el vértice final de salida.

Nota: No se detiene solo porque se haya actualizado la distancia al vértice final, se tiene que visitar el vértice objetivo.

Grafos

Algoritmo de Dijkstra

- El algoritmo de Dijkstra mantiene un conjunto **S** de vértices cuyos pesos finales de la ruta más corta del origen s ya se han determinado.
- El algoritmo selecciona repetidamente el vértice $u \in V - S$ con la estimación mínima del camino más corto, suma u a S y relaja todos las aristas dejando u.
- Se usa una cola Q de vértices de prioridad mínima, codificada por sus valores d.

Pseudocódigo

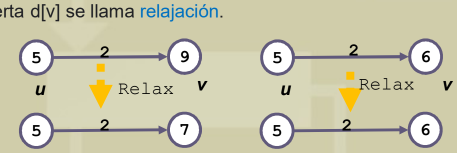
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V   (Inicializar Q con todos los vértices)
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

S: conjunto de vértices cuyos pesos finales de la ruta más corta del origen **s** ya se han determinado.
Q: cola de vértices de prioridad mínima.

Relajación

- Mantener la distancia más corta descubierta d[v] se llama relajación.

```
Relax(u,v,w) {
    if (d[v] > d[u]+w)
        then
            d[v]=d[u]+w;
}
```



Grafos

Floyd

El problema…

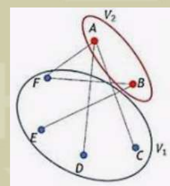
- Dado un grafo ponderado $G = (V, E, w)$, el problema de los caminos más cortos de todos los pares es encontrar precisamente los caminos más cortos entre todos los pares de vértices $v_i, v_j \in V$.
- Existen varios algoritmos para resolver este problema.

Algoritmo de Floyd-Warshall

- El algoritmo de Floyd-Warshall es un algoritmo para encontrar la ruta más corta entre todos los pares de vértices en un grafo ponderado.
- Este algoritmo funciona tanto para los grafos ponderados dirigidos como para los no dirigidos.
- No funciona para grafos con ciclos negativos (*donde la suma de los bordes en un ciclo es negativa*).
- El algoritmo de Floyd-Warshall también se le conoce como algoritmo de Floyd, algoritmo de Roy-Floyd, algoritmo de Roy-Warshall o algoritmo WFI.

Aplicaciones

- ✓ Encontrar el camino más corto en un grafo dirigido.
- ✓ Encontrar la cerradura transitiva de grafos dirigidos.
- ✓ Encontrar la inversa de matrices con números reales.
- ✓ Para probar si un grafo no dirigido es bipartito, es decir, que sus vértices se pueden separar en dos conjuntos disjuntos, de manera que las aristas no pueden relacionar vértices de un mismo conjunto.

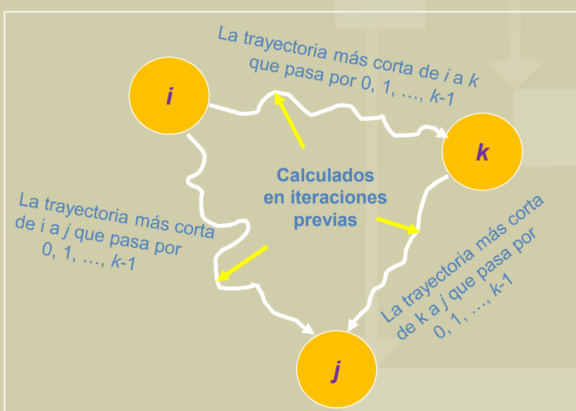


Grafos

Algoritmo de Floyd

Principio de optimalidad

Si ***k*** es un nodo del camino mínimo entre ***i*** y ***j***, entonces la parte del camino que va desde ***i*** hasta ***k*** y la que va de ***k*** hasta ***j*** es óptimo también.



Grafos

Algoritmo de Floyd

- Dada una ponderación, el grafo dirigido *G = (V, E)* con una función de ponderación *w: E → R* que asigna a las aristas pesos de valores reales.
- Se desea encontrar, para cada par de vértices *u, v* $\in$ *V*, la ruta más corta (*de menor peso*) de *u* a *v*, donde el peso de una ruta es la suma de los pesos de las aristas que lo forman.

Representación del grafo

- A diferencia de los algoritmos de origen único, que asumen una representación de lista de adyacencia del grafo, la mayoría de los algoritmos que revisaremos utilizarán una representación a través de una matriz de adyacencia.

Matriz de adyacencia

- Para facilidad de análisis, se asume que los vértices están numerados como: 1, 2, …, |V|, de modo que la entrada es una matriz W de *n x n* que representa los pesos de las aristas de grafo dirigido con n-vértices G = (V, E). Es decir: W = (w_{ij}) donde:

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \,, \\ \text{the weight of directed edge } (i, j) & \text{if } i \neq j \text{ and } (i, j) \in E \,, \\ \infty & \text{if } i \neq j \text{ and } (i, j) \notin E \,. \end{cases}$$

Estructura del camino más corto

- El algoritmo de Floyd considera los vértices intermedios de una ruta más corta, donde un vértice intermedio de una ruta simple p = $\langle v_1, v_2, \ldots, v_m \rangle$ es cualquier vértice de *p* distinto a v_1 o v_m, es decir, cualquier vértice en el conjunto $\{v_2, v_3, \ldots, v_{m-1}\}$.

Grafos

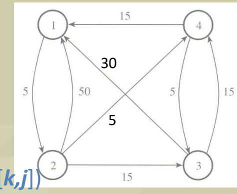
Algoritmo de Floyd

- Se basa en la construcción de una matriz ***D*** que da la longitud del camino más corto entre un par de nodos.
- El algoritmo asigna a ***D*** el valor inicial de ***L*** (distancias directas).
- Se efectúan ***n*** iteraciones. Después de la iteración ***k***, ***D*** da la longitud mínima de los caminos que utilizan solamente los nodos ***1..k*** como nodos intermedios.
- Al cabo de ***n*** iteraciones, ***D*** nos da la longitud de los caminos más cortos que utilicen algún nodo de ***N*** como nodo intermedio. Éste es el resultado deseado.
- En la iteración ***k***, el algoritmo debe comprobar para cada par de nodos *i,j* si existe o no un camino que vaya de *i* a *j* pasando por el nodo *k*, y que sea mejor que el camino óptimo actual y que pasa por los nodos {1,2,,***k-1***}, esto es:

$$D_k[\, i, j \,] = \min(D_{k-1}[i,j],\ D_{k-1}[i,k] + D_{k-1}[k,j])$$

Donde D_k representa a la matriz ***D*** después de la ***k****-ésima* iteración.

Grafos



D_0 = L=
{{ 0, 5, ∞, ∞},
{ 50, 0, 15, 5},
{ 30, ∞, 0, 15},
{ 15, ∞, 5, 0}};

Algoritmos

Algoritmo de Floyd

Ejemplo: Determinando D_k

$$D_k[i, j] = \min(D_{k-1}[i,j],\ D_{k-1}[i,k] + D_{k-1}[k,j])$$

k = 1, i = 1, 1 ≤ j ≤ n

$D_1[1,1] = \min(D_0[1,1],\ D_0[1,1] + D_0[1,1]) = 0$
$D_1[1,2] = \min(D_0[1,2],\ D_0[1,1] + D_0[1,2]) = 5$
$D_1[1,3] = \min(D_0[1,3],\ D_0[1,1] + D_0[1,3]) = \infty$
$D_1[1,4] = \min(D_0[1,4],\ D_0[1,1] + D_0[1,4]) = \infty$

k = 1, i = 2, 1≤ j ≤ n

$D_1[2,1] = \min(D_0[2,1],\ D_0[2,1] + D_0[1,1]) = 50$
$D_1[2,2] = \min(D_0[2,2],\ D_0[2,1] + D_0[1,2]) = 0$
$D_1[2,3] = \min(D_0[2,3],\ D_0[2,1] + D_0[1,3]) = 15$
$D_1[2,4] = \min(D_0[2,4],\ D_0[2,1] + D_0[1,4]) = 5$

k = 1, i = 3, 1 ≤ j ≤ n

$D_1[3,1] = \min(D_0[3,1],\ D_0[3,1] + D_0[1,1]) = 30$
$D_1[3,2] = \min(D_0[3,2],\ D_0[3,1] + D_0[1,2]) = 35$
$D_1[3,3] = \min(D_0[3,3],\ D_0[3,1] + D_0[1,3]) = 0$
$D_1[3,4] = \min(D_0[3,4],\ D_0[3,1] + D_0[1,4]) = 15$

k = 1, i = 4, 1 ≤ j ≤ n

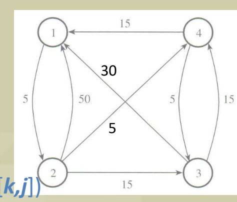
$D_1[4,1] = \min(D_0[4,1],\ D_0[4,1] + D_0[1,1]) = 15$
$D_1[4,2] = \min(D_0[4,2],\ D_0[4,1] + D_0[1,2]) = 20$
$D_1[4,3] = \min(D_0[4,3],\ D_0[4,1] + D_0[1,3]) = 5$
$D_1[4,4] = \min(D_0[4,4],\ D_0[4,1] + D_0[1,4]) = 0$

La matriz D_1 resultante es:

$$D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

Y se repite el proceso para *k = 2, 3 y 4 …*

Grafos



D_0 = L=
{{ 0, 5, ∞, ∞},
{ 50, 0, 15, 5},
{ 30, ∞, 0, 15},
{ 15, ∞, 5, 0}};

Algoritmos

Algoritmo de Floyd

Ejemplo: Determinando D_k

$$D_k[i, j] = \min(D_{k-1}[i,j],\ D_{k-1}[i,k] + D_{k-1}[k,j])$$

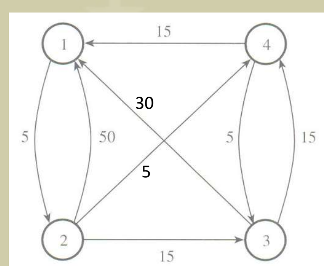
Y repitiendo el proceso para *k = 2, 3 y 4 …*

$$D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

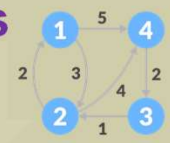
$$D_2 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

$$D_3 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 45 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

$$D_4 = \begin{pmatrix} 0 & 5 & 15 & 10 \\ 20 & 0 & 10 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$



Grafos



$$A^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 5 \\ 2 & 2 & 0 & \infty & 4 \\ 3 & \infty & 1 & 0 & \infty \\ 4 & \infty & \infty & 2 & 0 \end{array}$$

Algoritmos

Algoritmo de Floyd

Ejemplo: Determinando D_k

$$D_k[\,i, j\,] = \min(D_{k-1}[i,j],\ D_{k-1}[i,k] + D_{k-1}[k,j])$$

k* = 1, *i* = 1, 1 ≤ *j* ≤ *n

$D_1[1,1] = \min(D_0[1,1],\ D_0[1,1] + D_0[1,1]) = 0$

$D_1[1,2] = \min(D_0[1,2],\ D_0[1,1] + D_0[1,2]) = 3$

$D_1[1,3] = \min(D_0[1,3],\ D_0[1,1] + D_0[1,3]) = \infty$

$D_1[1,4] = \min(D_0[1,4],\ D_0[1,1] + D_0[1,4]) = 5$

k* = 1, *i* = 2, 1≤ *j* ≤ *n

$D_1[2,1] = \min(D_0[2,1],\ D_0[2,1] + D_0[1,1]) = 2$

$D_1[2,2] = \min(D_0[2,2],\ D_0[2,1] + D_0[1,2]) = 0$

$D_1[2,3] = \min(D_0[2,3],\ D_0[2,1] + D_0[1,3]) = \infty$

$D_1[2,4] = \min(D_0[2,4],\ D_0[2,1] + D_0[1,4]) = 4$

k* = 1, *i* = 3, 1 ≤ *j* ≤ *n

$D_1[3,1] = \min(D_0[3,1],\ D_0[3,1] + D_0[1,1]) = \infty$

$D_1[3,2] = \min(D_0[3,2],\ D_0[3,1] + D_0[1,2]) = 1$

$D_1[3,3] = \min(D_0[3,3],\ D_0[3,1] + D_0[1,3]) = 0$

$D_1[3,4] = \min(D_0[3,4],\ D_0[3,1] + D_0[1,4]) = \infty$

k* = 1, *i* = 4, 1 ≤ *j* ≤ *n

$D_1[4,1] = \min(D_0[4,1],\ D_0[4,1] + D_0[1,1]) = \infty$

$D_1[4,2] = \min(D_0[4,2],\ D_0[4,1] + D_0[1,2]) = \infty$

$D_1[4,3] = \min(D_0[4,3],\ D_0[4,1] + D_0[1,3]) = 2$

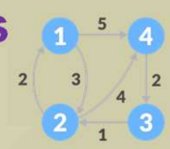
$D_1[4,4] = \min(D_0[4,4],\ D_0[4,1] + D_0[1,4]) = 0$

La matriz D_1 resultante es: ***k* = 1**

0	3	INF	5
2	0	INF	4
INF	1	0	INF
INF	INF	2	0

Y se repite el proceso para *k* = *2, 3 y 4 …*

Grafos



$$A^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 5 \\ 2 & 2 & 0 & \infty & 4 \\ 3 & \infty & 1 & 0 & \infty \\ 4 & \infty & \infty & 2 & 0 \end{array}$$

Algoritmos

Algoritmo de Floyd

Ejemplo: Determinando D_k

$$D_k[\,i, j\,] = \min(D_{k-1}[i,j],\ D_{k-1}[i,k] + D_{k-1}[k,j])$$

Y repitiendo el proceso para *k* = *2, 3 y 4 …*

***k* = 1**

0	3	INF	5
2	0	INF	4
INF	1	0	INF
INF	INF	2	0

***k* = 2**

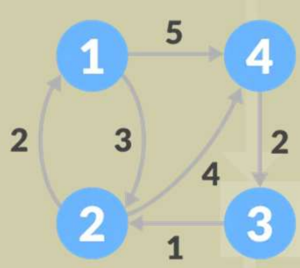
0	3	INF	5
2	0	INF	4
3	1	0	5
INF	INF	2	0

***k* = 3**

0	3	INF	5
2	0	INF	4
3	1	0	5
5	3	2	0

***k* = 4**

0	3	7	5
2	0	6	4
3	1	0	5
5	3	2	0



Grafos

Algoritmos

Algoritmo de Floyd

```
función Floyd(L[1..n,1..n]):matriz[1..n,1..n]
  matriz D[1..n,1..n]=L
  para k ← 1 hasta n hacer
    para i ← 1 hasta n hacer
      para j ← 1 hasta n hacer
        D[i,j] <- mín(D[i,j],D[i,k]+D[k,j])
  devolver D
fin_función
```

FLOYD-WARSHALL(W)

$n = W.rows$

$D^{(0)} = W$

for $k = 1$ **to** n

  let $D^{(k)} = \left(d_{ij}^{(k)}\right)$ be a new $n \times n$ matrix

  for $i = 1$ **to** n

    for $j = 1$ **to** n

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

return $D^{(n)}$

Sea d_{ij}^{k} el peso de un camino más corto desde el vértice *i* al vértice *j* para el cual todos los vértices intermedios están en el conjunto {*1, 2,…, k*}

Complejidad

La complejidad es: $O(n^3)$

- ¿Cuál sería la complejidad de implementar Dijkstra para el mismo problema?
  - Aunque tienen el mismo orden, en la práctica suele preferirse Floyd por su sencillez (*encierra una constante más pequeña*).