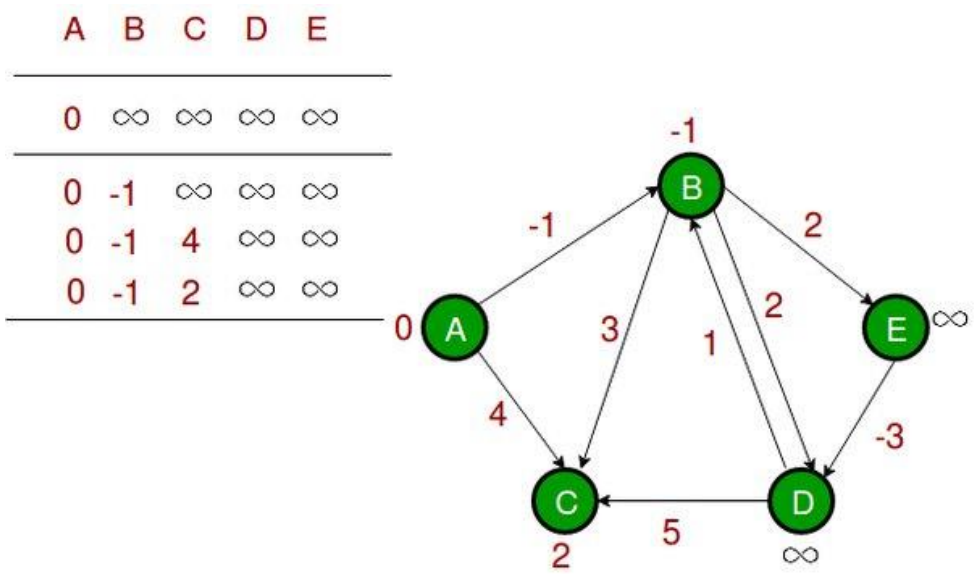


Complejidad Algorítmica

Unidad 2: Algoritmos voraces, programación dinámica y problemas P-NP

Módulo 13: Programación Dinámica en Grafos

Complejidad Algorítmica

Semana 13

MÓDULO 13: Programación Dinámica en Grafos

Contenido

1. Objetivo de la DP en Grafos
2. Principales Algoritmos de DP en grafos
   - Algoritmo Bellman Ford
   - Algoritmo Floyd-Warshall

Preguntas / Conclusiones

1. Objetivo de la DP en Grafos

¿Qué algoritmos de DP podemos aplicar en grafos?

El objetivo de esta sesión es:

> Manejar y desarrollar los principios del paradigma de **programación dinámica para encontrar caminos mínimos en grafos**.

Los siguientes algoritmos son ejemplos de Programación Dinámica:

- Algoritmo **Bellman-Ford**
- Algoritmo **Floy Warshall**

Aprendamos en que consisten…

2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

Objetivo: Encontrar el camino más corto desde un vértice a todos los demás vértices de un grafo ponderado.

Características

- Es similar al algoritmo de Dijkstra pero puede funcionar con gráficos en los que **los bordes pueden tener pesos negativos**.

- Los bordes de peso negativo pueden parecer inútiles al principio, pero pueden explicar muchos fenómenos, como:

  - El flujo de caja
  - El calor liberado (pesos positivos)/absorbido (pesos negativos) en una reacción química, etc.

Hay que tener cuidado con los bordes con pesos negativos…

¿Pesos negativos?
¿Por qué? ¿Para qué?

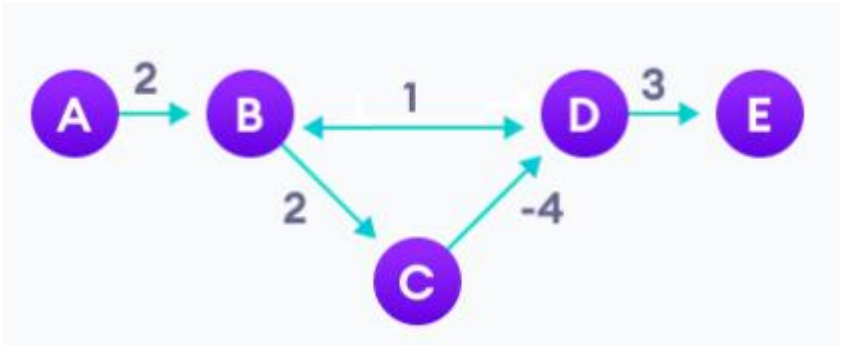
2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

¿Por qué debemos tener cuidado con los pesos negativos?

- Los bordes de peso negativo pueden **crear ciclos de peso negativo**, es decir, un ciclo que **reducirá la distancia total de la ruta** al regresar al mismo punto.

- Los ciclos de peso negativos pueden dar un resultado incorrecto al intentar encontrar el camino más corto. Ver el siguiente grafo:



- Los algoritmos de ruta más corta, como el algoritmo de Dijkstra, que no pueden detectar un ciclo de este tipo, pueden dar un resultado incorrecto porque pueden pasar por un ciclo de peso negativo y reducir la longitud de la ruta.

2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

- **Bellman Ford** es muy similar al Algoritmo de Dijkstra.
- A diferencia del algoritmo de Dijkstra, el algoritmo de **Bellman-Ford** puede funcionar en grafos con bordes de ponderación negativa.
- Esta capacidad hace que el algoritmo **Bellman-Ford** sea una opción popular.

¿Por qué es importante tener en cuenta los bordes negativos?

En la teoría de grafos, los bordes negativos son más importantes ya que pueden crear un ciclo negativo en un grafo dado.

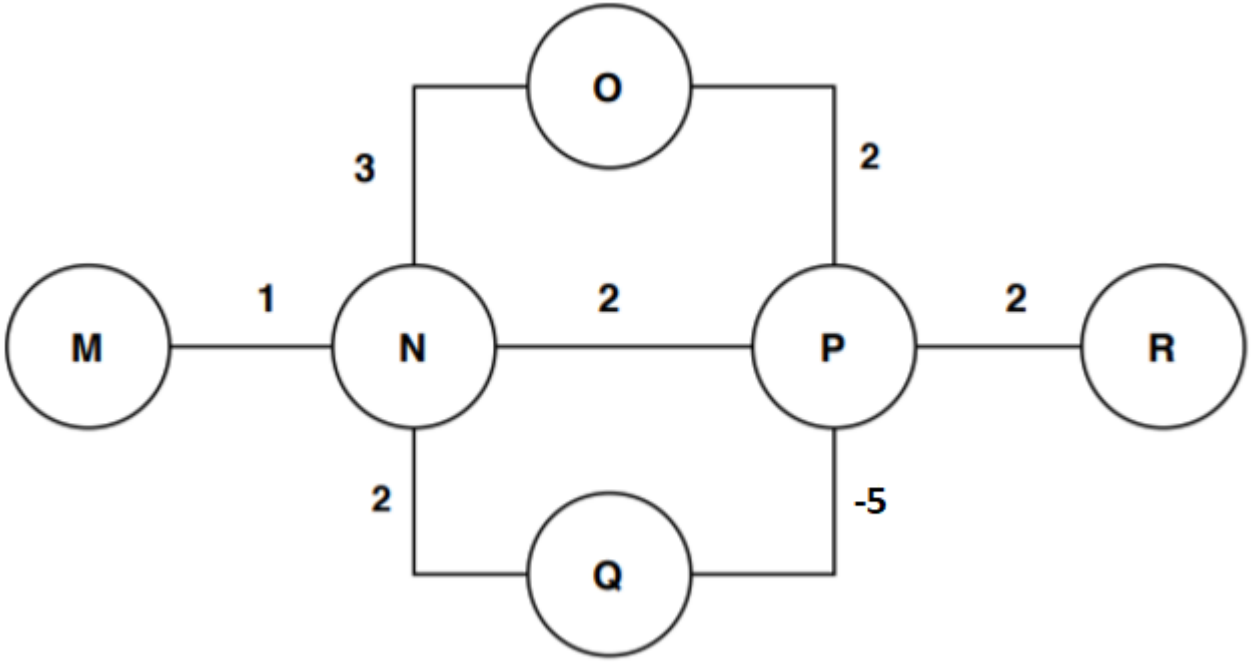
Veamos su funcionamiento en un ejemplo…

2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

EJEMPLO

Comencemos con un grafo ponderado simple con un ciclo negativo e intentemos encontrar la distancia más corta de un nodo a otro.

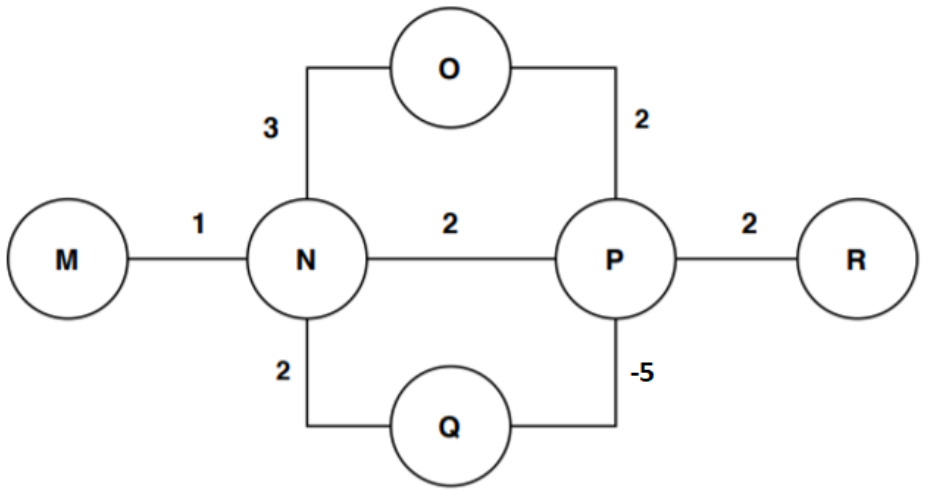


2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

EJEMPLO #1

Comencemos con un grafo ponderado simple con un ciclo negativo e intentemos encontrar la distancia más corta de un nodo a otro.



- Estamos considerando cada nodo como una ciudad. Queremos ir a la **ciudad R** desde la **ciudad M**.
- Hay tres caminos desde la ciudad M para llegar a la ciudad R: **MNPR**, **MNQPR**, **MNOPR**. Las longitudes de los caminos son 5, 0 y 8.
- Vemos que hay un ciclo negativo: **NQP**, que tiene una longitud de **-1**. Entonces, cada vez que cubrimos el camino **NQP**, la longitud de nuestro camino disminuirá.
- Esto hace que no podamos obtener una respuesta exacta sobre el camino más corto, ya que repetir infinitas veces el camino **NQP** sería, por definición, el camino menos costoso.

2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

Pasos del algoritmo Bellman-Ford

- Este algoritmo toma como entrada un **grafo ponderado dirigido** y **un vértice inicial**.
- Produce todos los caminos más cortos desde el vértice inicial hasta todos los demás vértices.

1. **El primer paso** es inicializar los vértices.
    - La variable D[] denota las distancias en este algoritmo.
    - La distancia entre el vértice inicial (S) y a sí mismo es 0.
    - El algoritmo inicializa con infinito ∞ la distancia desde el vértice inicial (S) hacia todos los demás vértices.

2. **El segundo paso**, el algoritmo comienza a calcular la distancia más corta desde el vértice inicial (S) hacia todos los demás vértices. Este paso corre (|V|-1) tiempos, igual a la cardinalidad del grafo -1.
    - El algoritmo intenta explorar diferentes caminos para llegar a otros vértices y calcular las distancias.
    - Si el algoritmo encuentra una distancia de un vértice que es más pequeña que el valor previamente almacenado, **relaja el borde** y **almacena el nuevo valor**.

Algorithm 1: Bellman-Ford Algorithm

```
Data: Given a directed graph G(V, E) , the starting vertex S, and
      the weight W of each edge
Result: Shortest path from S to all other vertices in G
D[S] = 0;
R = V - S;
C = cardinality(V);
for each vertex k ∈ R do
  | D[k] = ∞;
end
for each vertex i = 1 to (C - 1) do
  | for each edge (e1, e2) ∈ E do
  |   | Relax(e1, e2);
  | end
end
for each edge (e1, e2) ∈ E do
  | if D[e2] > D[e1] + W[e1, e2] then
  |   | Print("Graph contains negative weight cycle");
  | end
end
Procedure Relax (e1, e2)
for each edge (e1, e2) in E do
  | if D[e2] > D[e1] + W[e1, e2] then
  |   | D[e2] = D[e1] + W[e1, e2];
  | end
end
```

2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

Pasos del algoritmo Bellman-Ford (continuación)

3. En el **tercer paso**, después que el algoritmo itera el (|V|-1)tiempo y relaja todos los bordes requeridos, el algoritmo realiza una última verificación para averiguar si hay algún ciclo negativo en el gráfico.

   - Si existe un ciclo negativo, las distancias seguirán disminuyendo.
     - El algoritmo termina y da como resultado que el grafo contiene un ciclo negativo, por lo que el algoritmo no puede calcular la ruta más corta.

   - Si no se encuentra ningún ciclo negativo
     - El algoritmo devuelve las distancias más cortas.

¡Apliquemos estos pasos en un ejemplo!

Algorithm 1: Bellman-Ford Algorithm

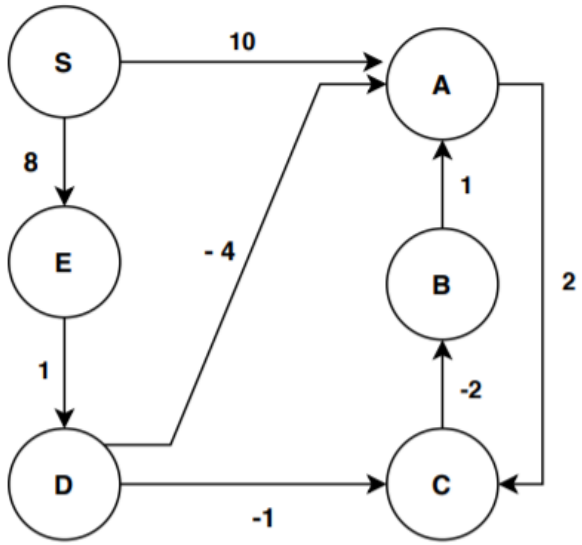
```
Data: Given a directed graph G(V, E) , the starting vertex S, and
      the weight W of each edge
Result: Shortest path from S to all other vertices in G
D[S] = 0;
R = V - S;
C = cardinality(V);
for each vertex k ∈ R do
|  D[k] = ∞;
end
for each vertex i = 1 to (C - 1) do
|  for each edge (e1, e2) ∈ E do
|  |  Relax(e1, e2);
|  end
end
for each edge (e1, e2) ∈ E do
|  if D[e2] > D[e1] + W[e1, e2] then
|  |  Print("Graph contains negative weight cycle");
|  end
end
Procedure Relax (e1, e2)
for each edge (e1, e2) in E do
|  if D[e2] > D[e1] + W[e1, e2] then
|  |  D[e2] = D[e1] + W[e1, e2];
|  end
end
```

2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

Ejemplo #1: Un grafo sin ciclo negativo

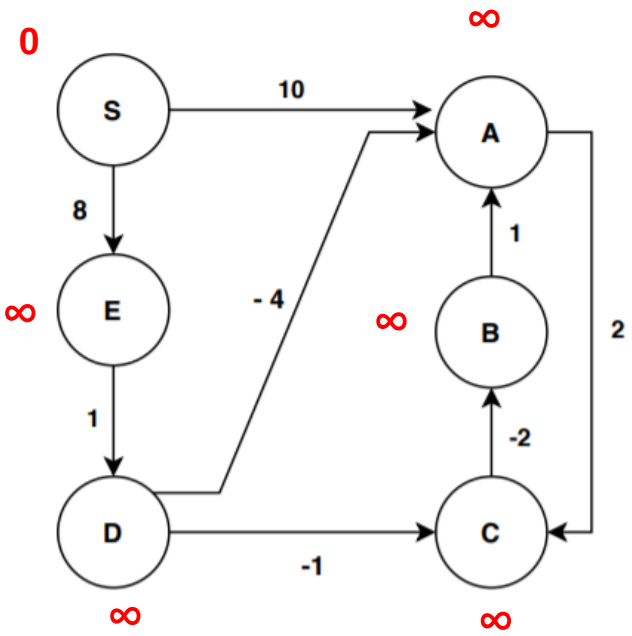
Suponemos que **S** es nuestro vértice inicial. Ahora estamos listos para comenzar con los pasos del algoritmo:



Paso #1: Inicialización

- Como comentamos, la distancia desde el nodo inicial (S) a sí mismo es 0.
- La distancia de todos los demás vértices es **infinita** para el paso de inicialización.
- Los valores en rojo denotan las distancias.
- Como tenemos seis vértices, el algoritmo ejecutará cinco iteraciones (|V|-1 veces) para encontrar el camino más corto y una iteración para encontrar un ciclo negativo (si existe alguno).

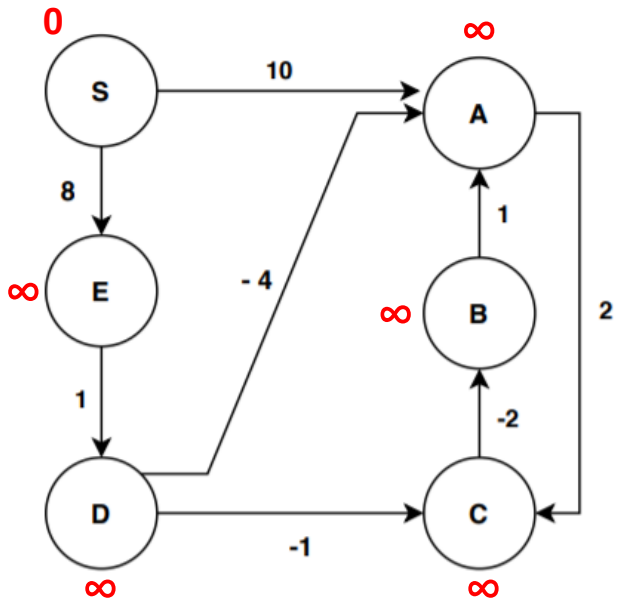
Iteración	S	S -> A	C -> B	A -> C	E -> D	S -> E	B -> A	D->A	D->C
	D(S)	D(A)	D(B)	D(C)	D(D)	D(E)	D(A)	D(A)	D(C)
0	0	∞	∞	∞	∞	∞	∞	∞	∞



2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

Ejemplo #1: Un grafo sin ciclo negativo



Paso #2: Iteración (|V|-1) veces

- Después de inicializar el grafo, ahora podemos proceder a la primera iteración:
- Actualizamos los valores de distancia de cada vértice:

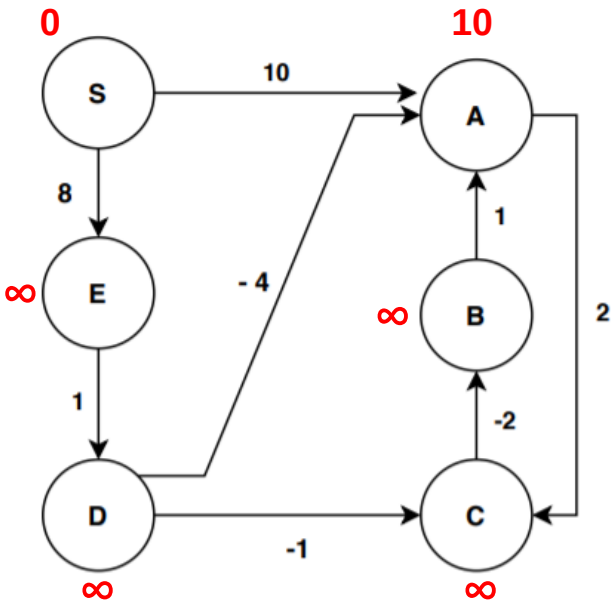
➢ **Iteración #1:** El algoritmo selecciona cada borde y lo pasa a la función Relax(). Primero, para el borde (S, A), veamos cómo funciona la función Relax(S, A). Primero verifica la condición:

$$D[A] > D[S] + W[S, A] \implies \infty > 0 + 10$$
$$\implies \infty > 10 \implies True$$

➢ La arista (S, A) satisface la condición de verificación, por lo tanto, el vértice A obtiene una nueva distancia:

$$D[A] = D[S] + W[S, A] \implies D[A] = 0 + 10 = 10$$

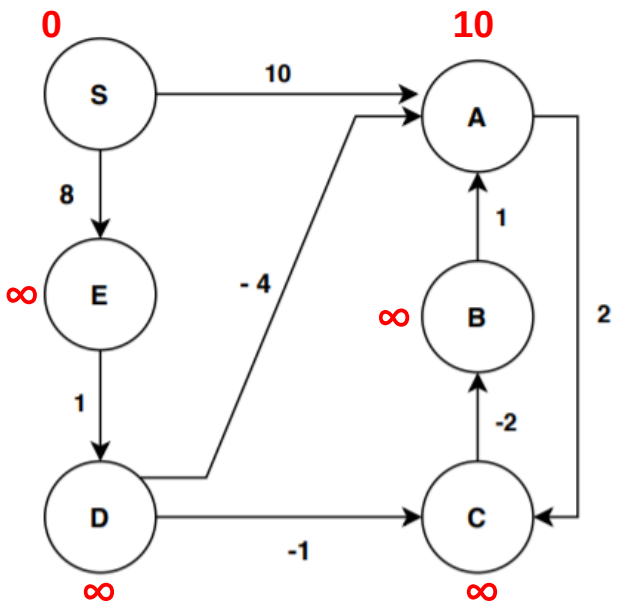
Ahora el nuevo valor de distancia del vértice A es 10



2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

Ejemplo #1: Un grafo sin ciclo negativo



Paso #2: Iteración (|V|-1) veces (continuación)

- Seguimos en la **iteración #1** por los siguientes bordes en este orden: (S, A) -> (A, C) -> (S, E) -> (C, B) -> (B, A) -> (E, D)

(A,C): **D[C]** > D[A] + W [A,C] = 10 + 2 => ∞ > 12 => **si** => **D[C] = 12**

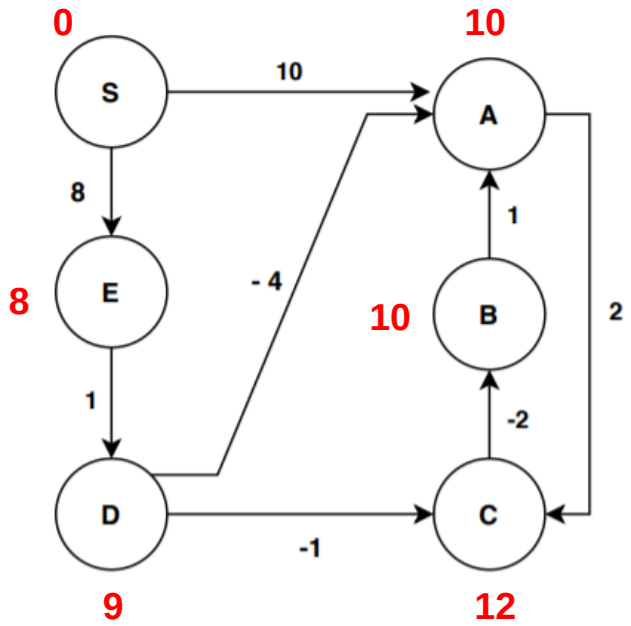
(S,E): **D[E]** > D[S] + W [S,E] = 0 + 8 => ∞ > 8 => **si** => **D[E] = 8**

(C,B): **D[B]** > D[C] + W [C,B] = 12 + (-2) => ∞ > 10 => **si** => **D[B] = 10**

(B,A): **D[A]** > D[B] + W [B,A] = 10 + 1 => **10** > 11 => no => **D[A] =10**

(E,D): **D[D]** > D[E] + W [E,D] = 8 + 1 = 9 => ∞ > 9 => **si** => **D[D] = 9**

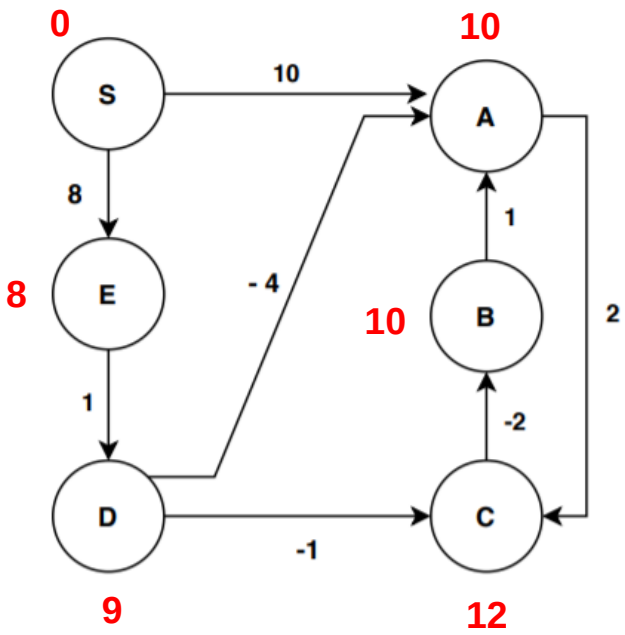
Iteración	S	S -> A	C -> B	A -> C	E -> D	S -> E	B -> A	D->A	D->C
	D(S)	D(A)	D(B)	D(C)	D(D)	D(E)	D(A)	D(A)	D(C)
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	10	10	12	9	8	10	10	12



2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

Ejemplo #1: Un grafo sin ciclo negativo



Paso #2: Iteración (|V|-1) veces (continuación)

- **Iteración #2:** Seguimos iterando por los siguientes bordes en este orden: (S, A) -> (S, E) -> (A, C) -> (B, A) -> (C, B) -> (D, C) -> (D, A)

(S,A) -> **D[A]** > D[S] + W [S,A] = 0 + 10 => **10** > 10 => no => **D[A] = 10**

(S,E) -> **D[E]** > D[S] + W [S,E] = 0 + 8 => **8** > 8 => no => **D[E] = 8**

(A,C) -> **D[C]** > D[A] + W [A,C] = 10 + 2 => **12** > 12 => no => **D[C] = 12**

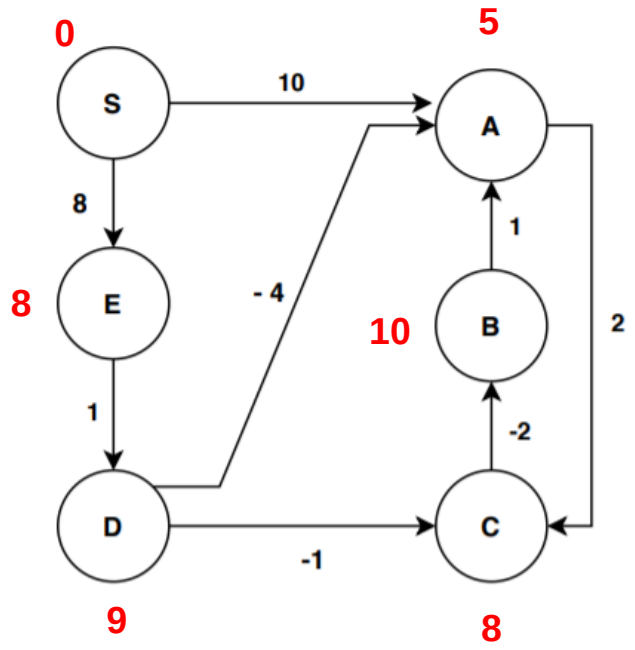
(B,A) -> **D[A]** > D[B] + W [B,A] = 10 + 1 => **10** > 11 => no => **D[A] =10**

(C,B) -> **D[B]** > D[C] + W [C,B] = 12 + (-2) => **10** > 10 => no => **D[B] = 10**

Si cumplen la condición:

(DC) -> **D[C]** > D[D] + W [D,C] = 9 + (-1) = 8 => **12** > 8 => SI => **D[C] = 8**

(DA) -> **D[A]** > D[D] + W [D,A] = 9 + (-4) = 5 => **10** > 5 => SI => **D[A] = 5**

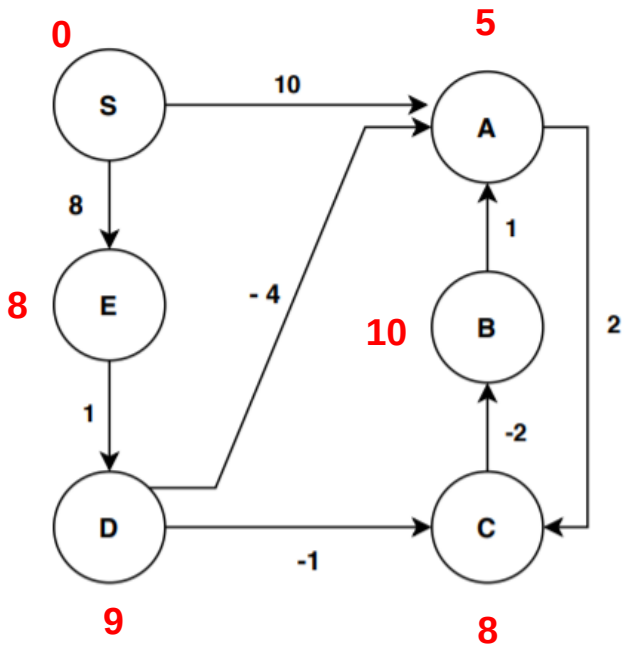


Iteración	S	S -> A	C -> B	A -> C	E -> D	S -> E	B -> A	D->A	D->C
	D(S)	D(A)	D(B)	D(C)	D(D)	D(E)	D(A)	D(A)	D(C)
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	10	10	12	9	8	10	10	12
2	0	5	10	8	9	8	10	5	8

2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

Ejemplo #1: Un grafo sin ciclo negativo



Paso #2: Iteración (|V|-1) veces (continuación)

- **Iteración #3:** Seguimos iterando por los siguientes bordes en este orden: (S, A) -> (S, E) -> (A, C) -> (B, A) -> (C, B) -> (D, C) -> (D, A)

(S,A) -> **D[A]** > D[S] + W [S,A] = 0 + 10 => **5** > 10 => no => **D[A] = 5**

(S,E) -> **D[E]** > D[S] + W [S,E] = 0 + 8 => **8** > 8 => no => **D[E] = 8**

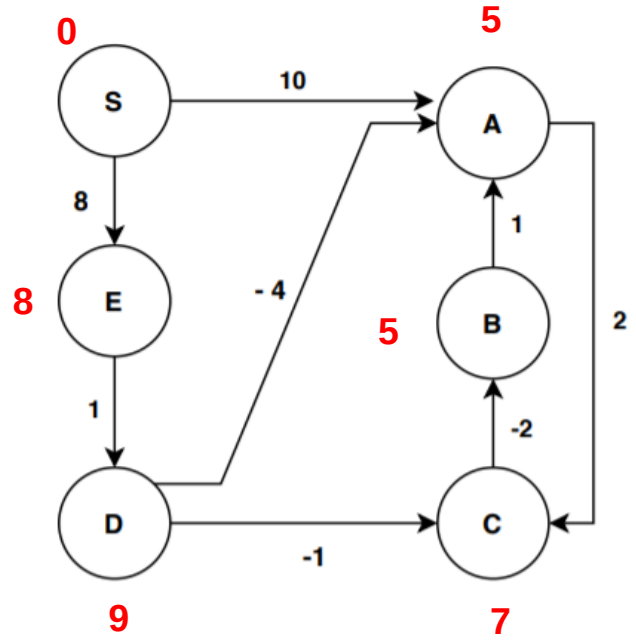
(A,C) -> **D[C]** > D[A] + W [A,C] = 5 + 2 => **8** > 7 => **si** => **D[C] = 7**

(B,A) -> **D[A]** > D[B] + W [B,A] = 10 + 1 => **5** > 11 => no => **D[A] =5**

(C,B) -> **D[B]** > D[C] + W [C,B] = 7 + (-2) => **10** > 5 => **si** => **D[B] = 5**

(DC) -> **D[C]** > D[D] + W [D,C] = 9 + (-1) = 8 => **7** > 8 => no => **D[C] = 7**

(DA) -> **D[A]** > D[D] + W [D,A] = 9 + (-4) = 5 => **5** > 5 => no => **D[A] = 5**

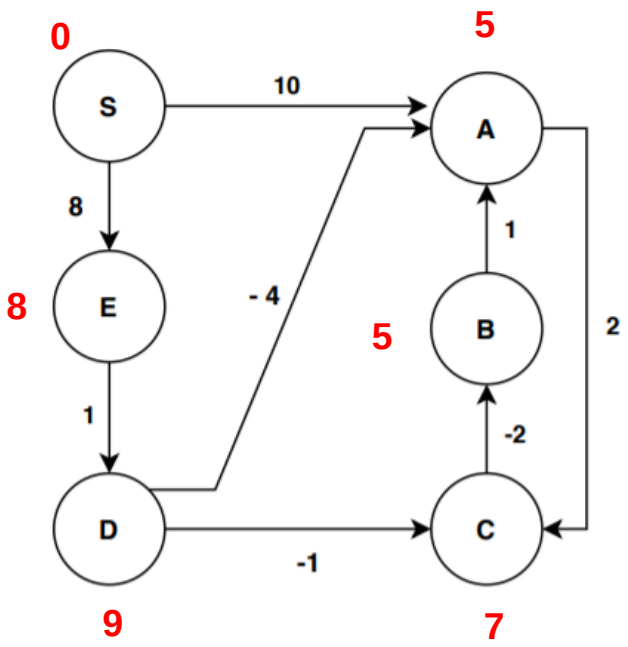


Iteración	S	S -> A	C -> B	A -> C	E -> D	S -> E	B -> A	D->A	D->C
	D(S)	D(A)	D(B)	D(C)	D(D)	D(E)	D(A)	D(A)	D(C)
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	10	10	12	9	8	∞	∞	∞
2	0	5	10	8	9	8	10	5	8
3	0	5	5	7	9	8	5	5	7

2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

Ejemplo #1: Un grafo sin ciclo negativo



Paso #2: Iteración (|V|-1) veces (continuación)

- **Iteración #4:** Seguimos iterando por los siguientes bordes en este orden: (S, A) -> (S, E) -> (A, C) -> (B, A) -> (C, B) -> (D, C) -> (D, A)

(S,A) -> **D[A]** > D[S] + W [S,A] = 0 + 10 => **5** > 10 => no => **D[A] = 5**

(S,E) -> **D[E]** > D[S] + W [S,E] = 0 + 8 => **8** > 8 => no => **D[E] = 8**

(A,C) -> **D[C]** > D[A] + W [A,C] = 5 + 2 => **7** > 7 => no => **D[C] = 7**

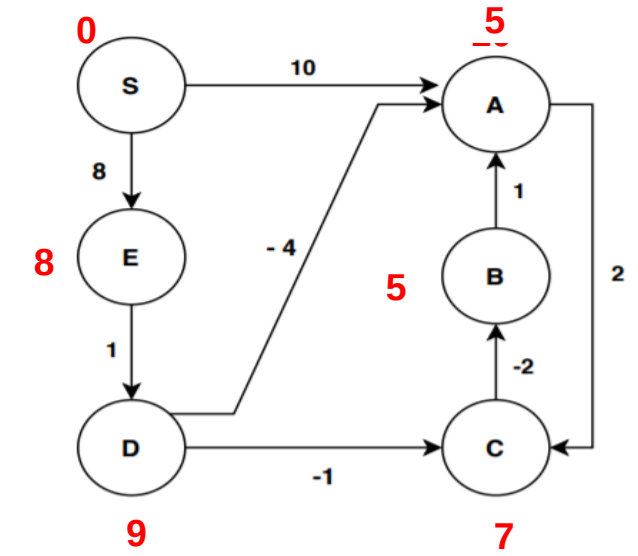
(B,A) -> **D[A]** > D[B] + W [B,A] = 5 + 1 => **5** > 6 => no => **D[A] =5**

(C,B) -> **D[B]** > D[C] + W [C,B] = 7 + (-2) => **5** > 5 => no => **D[B] = 5**

(DC) -> **D[C]** > D[D] + W [D,C] = 9 + (-1) = 8 => **7** > 8 => no => **D[C] = 7**

(DA) -> **D[A]** > D[D] + W [D,A] = 9 + (-4) = 5 => **5** > 5 => no => **D[A] = 5**

Ninguna actualización en la iteración #4

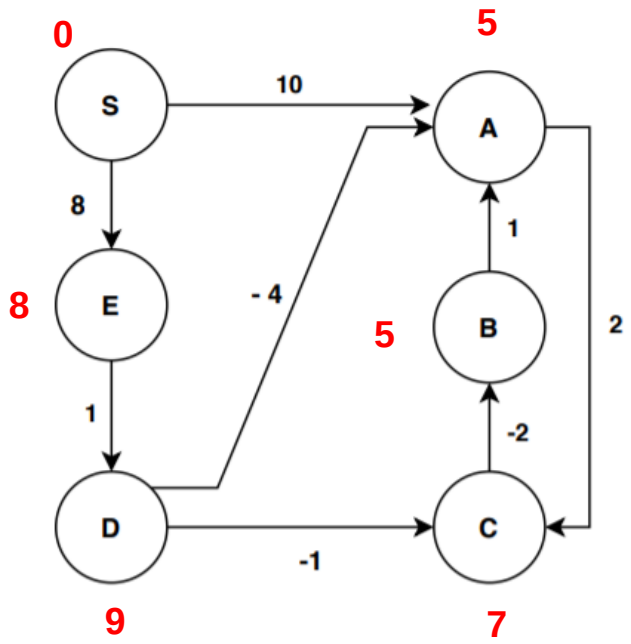


Iteración	S	S -> A	C -> B	A -> C	E -> D	S -> E	B -> A	D->A	D->C
	D(S)	D(A)	D(B)	D(C)	D(D)	D(E)	D(A)	D(A)	D(C)
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	10	10	12	9	8	10	10	12
2	0	5	10	8	9	8	10	5	8
3	0	5	5	7	9	8	5	5	7
4	0	5	5	7	9	8	5	5	7

2. Principales Algoritmos de DP en grafos

ALGORITMO BELLMAN FORD

Ejemplo #1: Un grafo sin ciclo negativo



- No hay actualizaciones en los valores de distancia de los vértices después de la cuarta iteración.
- Esto significa que el algoritmo ha producido el resultado final (a pesar que mencionamos que necesitamos ejecutar este algoritmo |V|-1 veces, es decir para 5 interacciones).
- En este caso, obtuvimos nuestro resultado después de 4 iteraciones.
- En general (|V| - 1) es el mayor número de iteraciones que necesitamos ejecutar en caso de que los valores de distancia de las iteraciones consecutivas no sean estables.
- En este caso, obtuvimos los mismos valores para dos iteraciones consecutivas, por lo que el algoritmo termina.

2. Principales Algoritmos de DP en grafos

ALGORITMO FLOYD-WARSHALL

Objetivo: Encontrar el camino más corto para cada par de vértices en un grafo dirigido ponderado.

Características

- En el problema del camino más corto de todos los pares, necesitamos encontrar todos los caminos más cortos desde cada vértice a todos los demás vértices en el grafo.

2. Principales Algoritmos de DP en grafos

ALGORITMO FLOYD-WARSHALL

Pasos del algoritmo FLOYD-WARSHALL

- Este algoritmo toma como entrada un **grafo ponderado dirigido G (V, E)**.
- Produce todos los caminos más cortos por cada par de vértices en G.

1. **El primer paso:**
   - El algoritmo construye una matriz cuadrática a partir del grafo inicial.
   - Inicializa la distancia con valores ∞ y sólo actualiza las celdas con los pesos de los bordes en el grafo.
   - Inserta **"0"** en las posiciones diagonales en la matriz.
2. **El segundo paso**, encontrar la distancia entre dos vértices
   - Al encontrar la distancia, también verifica si hay algún vértice intermedio entre dos vértices seleccionados.
   - Si existe un vértice intermedio, comprueba la distancia entre el par de vértices seleccionados que pasa por este vértice intermedio.
   - Si esta distancia al atravesar el vértice intermedio es menor que la distancia entre dos vértices seleccionados (sin pasar por el vértice intermedio), actualiza el valor de la distancia más corta en la matriz.

- El número de iteraciones es igual a la cardinalidad del conjunto de vértices.
- El algoritmo **devuelve la distancia más corta de cada vértice a otro** en el grafo inicial.

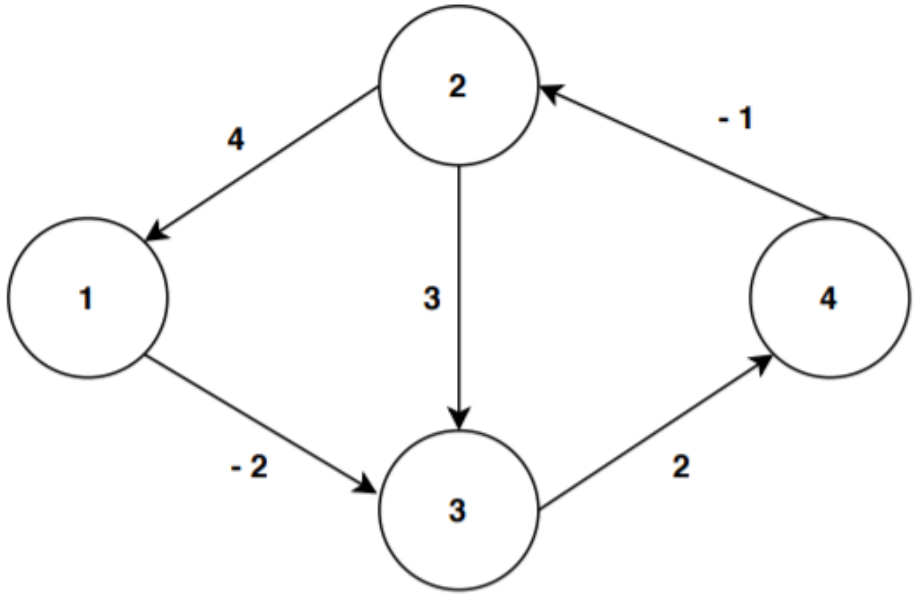
Algorithm 1: Pseudocode of Floyd-Warshall Algorithm

```
Data: A directed weighted graph G(V, E)
Result: Shortest path between each pair of vertices in G
for each d ∈ V do
    distance[d][d] ← 0;
end
for each edge (s, p) ∈ E do
    distance[s][p] ← weight(s, p);
end
n = cardinality(V);
for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            if distance[i][j] > distance[i][k] + distance[k][j] then
                distance[i][j] ← distance[i][k] + distance[k][j];
            end
        end
    end
end
```

2. Principales Algoritmos de DP en grafos

ALGORITMO FLOYD-WARSHALL

Ejemplo: Ejecutemos el algoritmo de **Floyd-Warshall** en un grafo dirigido ponderado:



Paso #1:

- Construimos una matriz cuadrática a partir del grafo de entrada G.

$$\begin{pmatrix} \infty & \infty & -2 & \infty \\ 4 & \infty & 3 & \infty \\ \infty & \infty & \infty & 2 \\ \infty & -1 & \infty & \infty \end{pmatrix}$$

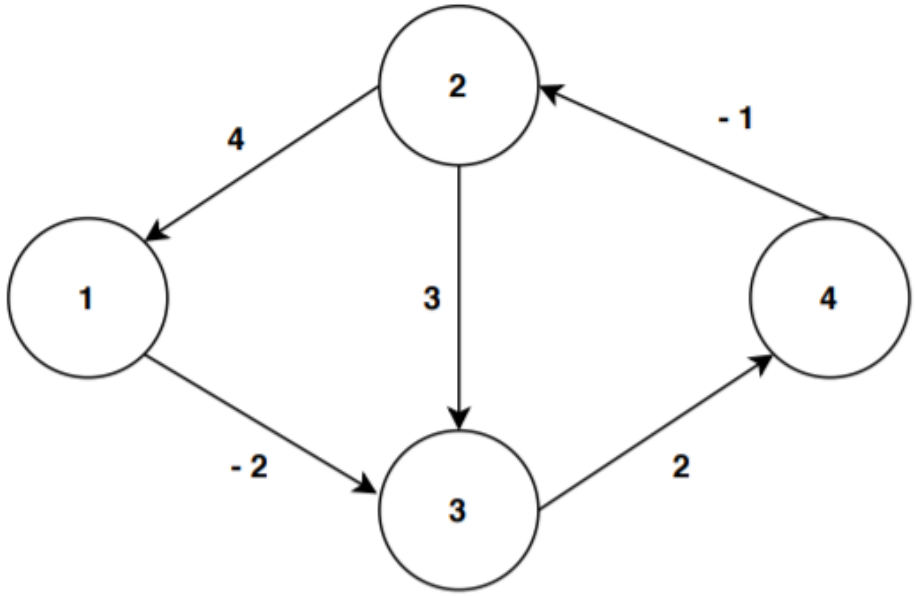
- Insertamos **"0"** en las posiciones diagonales en la matriz, y el resto de las posiciones se mantienen con los pesos de los bordes del grafo de entrada.

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 3 & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

2. Principales Algoritmos de DP en grafos

ALGORITMO FLOYD-WARSHALL

Ejemplo: Ejecutemos el algoritmo de **Floyd-Warshall** en un grafo dirigido ponderado:



Paso #2:

- Como la cardinalidad del conjunto de vértices es 4, tendremos 4 iteraciones.

Iteración #1: $k = 1, i = 1, j = 1$, verificaremos si debemos actualizar la matriz

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 3 & \infty \\ \infty & \propto & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

$$distance[i][j] > distance[i][k] + distance[k][j]$$

$$distance[1][1] > distance[1][1] + distance[1][1]$$

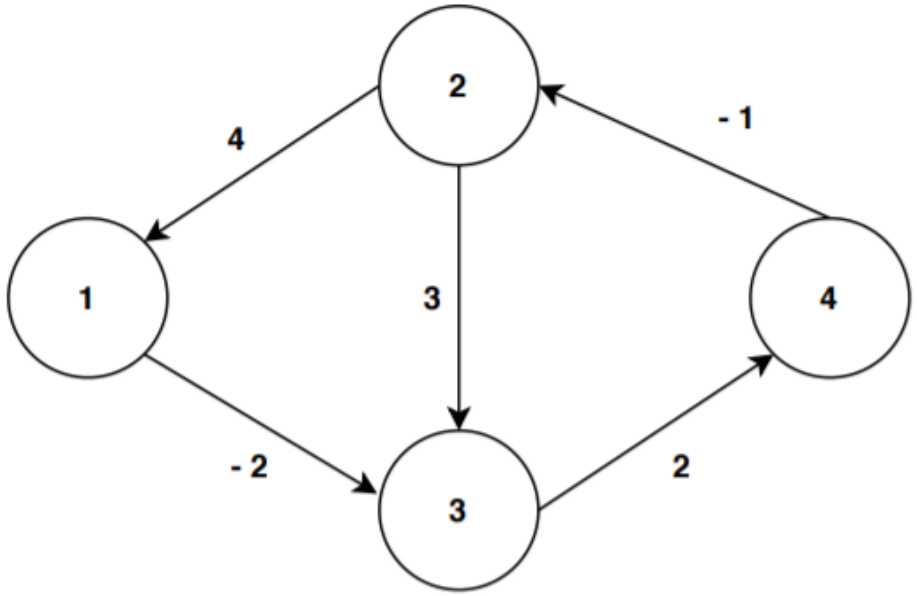
$$0 > 0 + 0 \implies \text{FALSE}$$

- Como ningún valor de la iteración cumple con la condición, no hay actualización en la matriz.

2. Principales Algoritmos de DP en grafos

ALGORITMO FLOYD-WARSHALL

Ejemplo: Ejecutemos el algoritmo de **Floyd-Warshall** en un grafo dirigido ponderado:



Paso #2:

Iteración #1: *k = 1, i = 1, j = 2 verificamos nuevamente en la matriz*

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 3 & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

$$distance[i][j] > distance[i][k] + distance[k][j]$$

$$distance[1][2] > distance[1][1] + distance[1][2]$$

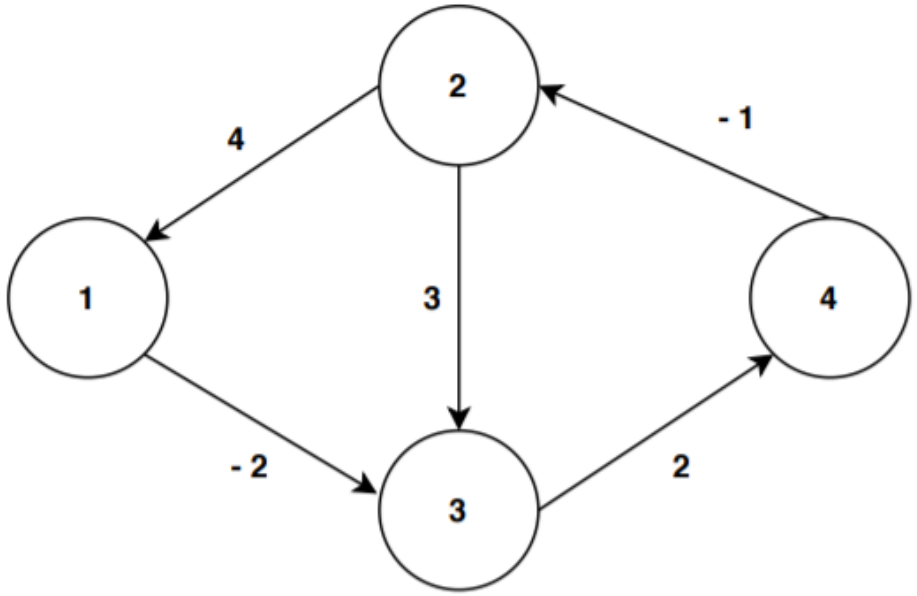
$$\infty > 0 + \infty \implies \text{FALSE}$$

- Por lo tanto, no habrá cambios en la matriz. De esta forma, continuaremos y comprobaremos todos los pares de vértices.

2. Principales Algoritmos de DP en grafos

ALGORITMO FLOYD-WARSHALL

Ejemplo: Ejecutemos el algoritmo de **Floyd-Warshall** en un grafo dirigido ponderado:



Paso #2:
Avancemos rápidamente a algunos valores que satisfagan la condición de distancia.

Iteración #1: Veremos que se cumple la condición para los valores del ciclo k =1, i=2, j= 3

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 2 & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

$$distance[i][j] > distance[i][k] + distance[k][j]$$

$$distance[2][3] > distance[2][1] + distance[1][3]$$

$$3 > 4 + -2$$

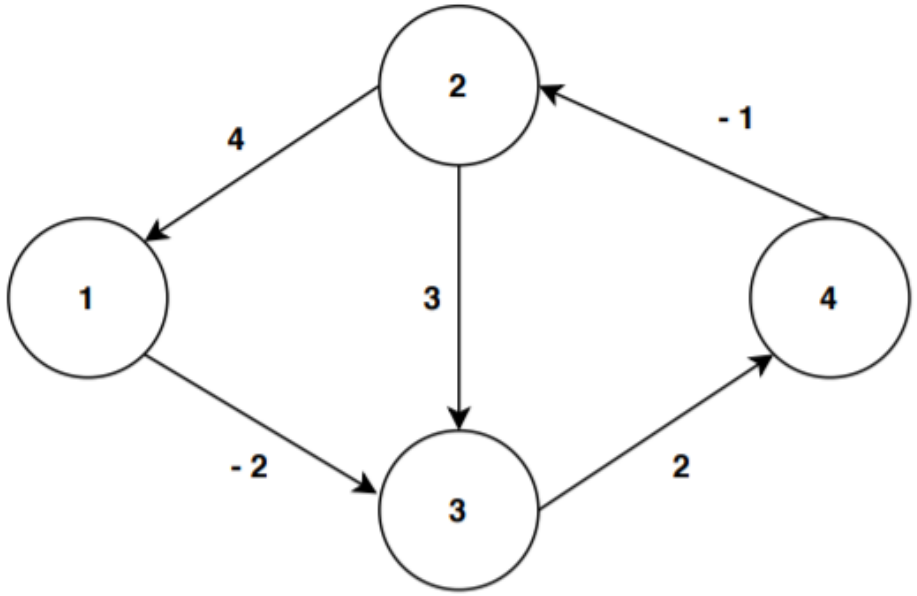
$$3 > 2 \implies \text{TRUE}$$

- Por lo tanto, la condición se cumple para el par de **vértices (2,3)**. Al principio, la distancia entre el vértice 2 a 3 **era 3**. Sin embargo, encontramos **una nueva distancia más corta 2**. Por eso, actualizamos la matriz con esta nueva distancia de ruta más corta.

2. Principales Algoritmos de DP en grafos

ALGORITMO FLOYD-WARSHALL

Ejemplo: Ejecutemos el algoritmo de **Floyd-Warshall** en un grafo dirigido ponderado:



Paso #2:
Avancemos rápidamente a algunos valores que satisfagan la condición de distancia.

Iteración #1: Veremos que se cumple la condición para los valores del ciclo k=2, i= 4, j= 1

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 2 & \infty \\ \infty & \infty & 0 & 2 \\ 3 & -1 & \infty & 0 \end{pmatrix}$$

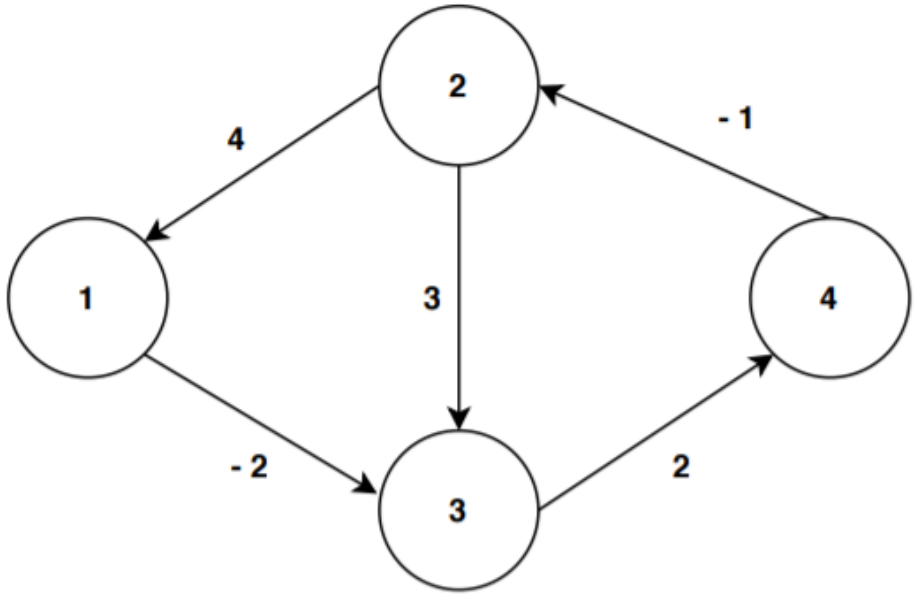
$$distance[i][j] > distance[i][k] + distance[k][j]$$
$$distance[4][1] > distance[4][2] + distance[2][1]$$
$$\infty > -1 + 4$$
$$\infty > 3 \implies \text{TRUE}$$

- Por lo tanto, la condición se cumple para el par de **vértices (4,1)**. Al principio, la distancia entre el vértice 4 a 1 era ∞. Sin embargo, encontramos **una nueva distancia más corta 3**. Por eso, actualizamos la matriz con esta nueva distancia de ruta más corta.

2. Principales Algoritmos de DP en grafos

ALGORITMO FLOYD-WARSHALL

Ejemplo: Ejecutemos el algoritmo de **Floyd-Warshall** en un grafo dirigido ponderado:



Paso #2:
Avancemos rápidamente a algunos valores que satisfagan la condición de distancia.

Iteración #n: continuamos y verificamos diferentes valores de bucle. Finalmente, después de que termine el algoritmo, obtendremos la matriz de salida que contiene las distancias más cortas de todos los pares:

$$\begin{pmatrix} 0 & -1 & -2 & 0 \\ 4 & 0 & 2 & 4 \\ 5 & 1 & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix}$$

- NO OLVIDAR que el número de iteraciones es igual a la cardinalidad del conjunto de vértices (en este ejemplo es 4).

Programación Dinámica en Grafos

CONCLUSIONES

1. El algoritmo **Bellman-Ford** es un ejemplo de programación dinámica.
2. Comienza con un vértice inicial y calcula las distancias de otros vértices que se pueden alcanzar por un borde. Luego continúa encontrando un camino con dos aristas y así sucesivamente. El algoritmo **Bellman-Ford** sigue el enfoque de abajo hacia arriba.
3. Complejidad de **Bellman-Ford**
   - Primero, el paso de inicialización toma O(V).
   - Luego, el algoritmo itera (|V| - 1)veces y cada iteración toma O(1)tiempo.
   - Después (|V| - 1)de las interacciones, el algoritmo elige todos los bordes y luego pasa los bordes a Relax() . Elegir todos los bordes lleva O(E)tiempo y la función Relax() lleva O(1)tiempo.
   - Por lo tanto , la complejidad de hacer todas las operaciones lleva O(VE)tiempo.
   - Dentro de la función Relax() , el algoritmo toma un par de aristas, realiza un paso de verificación y asigna el nuevo peso si está satisfecho. Todas estas operaciones toman O(E)tiempo.
   - Por lo tanto, el tiempo total del algoritmo Bellman-Ford es la suma del tiempo de inicialización, el tiempo de ciclo y el tiempo de la función Relax . En total, la complejidad temporal del algoritmo Bellman-Ford es O(VE).
4. El algoritmo **Floyd-Warshall** también es un ejemplo de programación dinámica.
   - Primero, insertamos los pesos de los bordes en la matriz. Hacemos esto usando un bucle **for** que visita todos los vértices del grafo. Esto se puede realizar en un tiempo O(n).
   - Luego, realizamos tres bucles anidados, cada uno de los cuales va desde uno hasta el número total de vértices del grafo. Por lo tanto, la complejidad temporal total de este algoritmo es O (n^3)

PREGUNTAS

Dudas y opiniones