

UNIVERSIDAD NACIONAL DE INGENIERÍA

Análisis y Diseño de Algoritmos

Caminos mínimos con programación dinámica

Bellman-Ford, ciclos negativos y Floyd-Warshall

Gerson Garrido

Semana 10

Hoy

Sesión A

1. Por qué hace falta otra idea
2. El subproblema: a lo sumo *i* aristas
3. Una sola fila, y su demostración
4. Detectar un ciclo negativo
5. Recuperar el ciclo
6. Los errores con “infinito”
7. Dónde se usa

Sesión B

8. Todos los pares
9. Floyd-Warshall
10. Por qué *k* va por fuera
11. La diagonal
12. Cuadro de decisión
13. Máscaras de bits y Held-Karp **[se puede cortar]**
14. Para el laboratorio: arbitraje

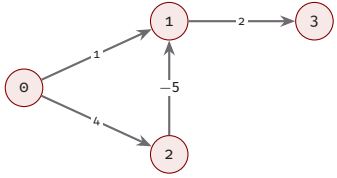
Lectura: apuntes de la semana 10. Código en `code/week-10`.

SECCIÓN 01

Por qué otra idea

01

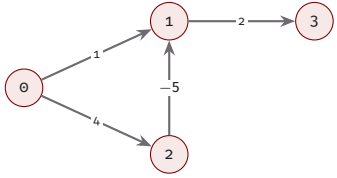
Lo que quedó pendiente de la semana 5



terminal

```
$ build/bf_demo
Dijkstra:        0   -1    4    3
```

Lo que quedó pendiente de la semana 5

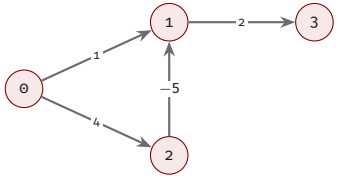


terminal

```
$ build/bf_demo
Dijkstra:        0   -1    4    3
```

- El nodo 3 debería valer 1:
  $0 \to 2 \to 1 \to 3$.

Lo que quedó pendiente de la semana 5

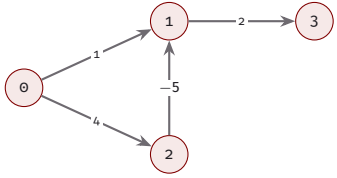


terminal

```
$ build/bf_demo
Dijkstra:        0   -1    4    3
```

- El nodo 3 debería valer 1:
  $0 \to 2 \to 1 \to 3$.
- Dijkstra fijó el 1 con distancia 1 y relajó sus aristas con ese valor.

Lo que quedó pendiente de la semana 5

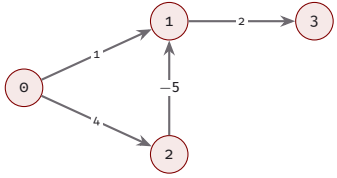



```
terminal

$ build/bf_demo
Dijkstra:        0   -1    4    3
```

- El nodo 3 debería valer 1:
  $0 \to 2 \to 1 \to 3$.
- Dijkstra fijó el 1 con distancia 1 y relajó sus aristas con ese valor.
- Cuando $d[1]$ baja a -1, nadie vuelve a leer sus aristas.

Lo que quedó pendiente de la semana 5



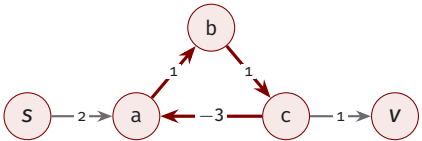

```
terminal

$ build/bf_demo
Dijkstra:        0   -1    4    3
```

- El nodo 3 debería valer 1:
  $0 \to 2 \to 1 \to 3$.
- Dijkstra fijó el 1 con distancia 1 y relajó sus aristas con ese valor.
- Cuando $d[1]$ baja a -1, nadie vuelve a leer sus aristas.

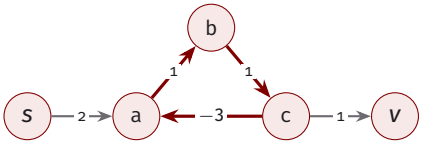
El problema: con un negativo, el futuro todavía puede mejorar lo ya fijado. Fijar nodos no sirve.

Y además: un ciclo negativo



Un **ciclo negativo** *(negative cycle)*: sus pesos suman menos que cero. Aquí, -1.

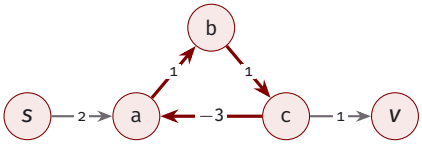
Y además: un ciclo negativo



Un **ciclo negativo** *(negative cycle)*: sus pesos suman menos que cero. Aquí, -1.

Cada vuelta acorta el camino de s a v. No hay mínimo: $\delta(v)$ **no existe**.

Y además: un ciclo negativo

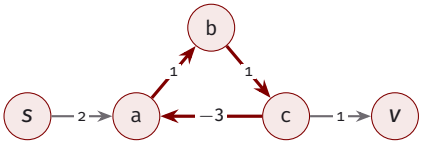


Un **ciclo negativo** *(negative cycle)*: sus pesos suman menos que cero. Aquí, -1.

Cada vuelta acorta el camino de s a v. No hay mínimo: $\delta(v)$ **no existe**.

Recordemos: un **camino** puede repetir nodos. Uno que no repite es un **camino simple**.

Y además: un ciclo negativo



Un **ciclo negativo** *(negative cycle)*: sus pesos suman menos que cero. Aquí, -1.

Cada vuelta acorta el camino de s a v. No hay mínimo: $\delta(v)$ **no existe**.

Recordemos: un **camino** puede repetir nodos. Uno que no repite es un **camino simple**.

Dos preguntas

¿Qué se calcula si no hay ciclos negativos? ¿Cómo se sabe que no los hay?

SECCIÓN 02

El subproblema

02

El parámetro que da un orden

Los nodos no tienen orden. Pero el **número de aristas** de un camino sí crece.

Subproblema

$d_i(v)$: el menor largo de un camino de s a v con **a lo sumo** i **aristas**, o ∞ si no hay ninguno.

El parámetro que da un orden

Los nodos no tienen orden. Pero el **número de aristas** de un camino sí crece.

Subproblema

$d_i(v)$: el menor largo de un camino de s a v con **a lo sumo** i **aristas**, o ∞ si no hay ninguno.

Con i aristas hay finitos caminos: el mínimo **siempre existe**, incluso con ciclos negativos.

El parámetro que da un orden

Los nodos no tienen orden. Pero el **número de aristas** de un camino sí crece.

Subproblema

$d_i(v)$: el menor largo de un camino de s a v con **a lo sumo** i **aristas**, o ∞ si no hay ninguno.

Con i aristas hay finitos caminos: el mínimo **siempre existe**, incluso con ciclos negativos.

La última decisión: ¿tiene a lo sumo $i-1$ aristas, o exactamente i, y entonces termina en alguna (u, v)?

El parámetro que da un orden

Los nodos no tienen orden. Pero el **número de aristas** de un camino sí crece.

Subproblema

$d_i(v)$: el menor largo de un camino de s a v con **a lo sumo** i **aristas**, o ∞ si no hay ninguno.

Con i aristas hay finitos caminos: el mínimo **siempre existe**, incluso con ciclos negativos.

La última decisión: ¿tiene a lo sumo $i-1$ aristas, o exactamente i, y entonces termina en alguna (u, v)?

$$d_i(v) = \min\Big(d_{i-1}(v),\ \min_{(u,v)} \big(d_{i-1}(u) + w(u,v)\big) \Big)$$

con $d_0(s) = 0$, $d_0(v) = \infty$ si $v \neq s$, y $\infty + w = \infty$.

Teorema: la recurrencia es correcta

Técnica: inducción sobre *i*, con las dos direcciones. Vale con cualquier signo.

Teorema: la recurrencia es correcta

Técnica: inducción sobre i, con las dos direcciones. Vale con cualquier signo.

- **Base.** Con 0 aristas, solo el camino vacío de s a s, de largo 0.

Teorema: la recurrencia es correcta

Técnica: inducción sobre i, con las dos direcciones. Vale con cualquier signo.

- **Base.** Con 0 aristas, solo el camino vacío de s a s, de largo 0.
- **Cada opción es un camino permitido.** $d_{i-1}(v)$ también tiene a lo sumo i aristas. Y un camino a u con $i-1$ aristas, más (u, v), tiene a lo sumo i.

Teorema: la recurrencia es correcta

Técnica: inducción sobre i, con las dos direcciones. Vale con cualquier signo.

- **Base.** Con 0 aristas, solo el camino vacío de s a s, de largo 0.
- **Cada opción es un camino permitido.** $d_{i-1}(v)$ también tiene a lo sumo i aristas. Y un camino a u con $i-1$ aristas, más (u, v), tiene a lo sumo i.
- **Todo camino permitido está cubierto.** Sea P un camino con a lo sumo i aristas:

Teorema: la recurrencia es correcta

Técnica: inducción sobre i, con las dos direcciones. Vale con cualquier signo.

- **Base.** Con 0 aristas, solo el camino vacío de s a s, de largo 0.
- **Cada opción es un camino permitido.** $d_{i-1}(v)$ también tiene a lo sumo i aristas. Y un camino a u con $i-1$ aristas, más (u, v), tiene a lo sumo i.
- **Todo camino permitido está cubierto.** Sea P un camino con a lo sumo i aristas:
  - si tiene a lo sumo $i-1$, mide al menos $d_{i-1}(v)$;

Teorema: la recurrencia es correcta

Técnica: inducción sobre i, con las dos direcciones. Vale con cualquier signo.

- **Base.** Con 0 aristas, solo el camino vacío de s a s, de largo 0.
- **Cada opción es un camino permitido.** $d_{i-1}(v)$ también tiene a lo sumo i aristas. Y un camino a u con $i-1$ aristas, más (u,v), tiene a lo sumo i.
- **Todo camino permitido está cubierto.** Sea P un camino con a lo sumo i aristas:
  - si tiene a lo sumo $i-1$, mide al menos $d_{i-1}(v)$;
  - si tiene i y termina en (u,v), mide al menos $d_{i-1}(u) + w(u,v)$.

Teorema: la recurrencia es correcta

Técnica: inducción sobre i, con las dos direcciones. Vale con cualquier signo.

- **Base.** Con 0 aristas, solo el camino vacío de s a s, de largo 0.
- **Cada opción es un camino permitido.** $d_{i-1}(v)$ también tiene a lo sumo i aristas. Y un camino a u con $i-1$ aristas, más (u, v), tiene a lo sumo i.
- **Todo camino permitido está cubierto.** Sea P un camino con a lo sumo i aristas:
  - si tiene a lo sumo $i-1$, mide al menos $d_{i-1}(v)$;
  - si tiene i y termina en (u, v), mide al menos $d_{i-1}(u) + w(u, v)$.
- Si no hay camino, ninguna opción es finita, y las dos cantidades son ∞.

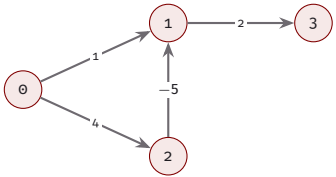
Teorema: la recurrencia es correcta

Técnica: inducción sobre i, con las dos direcciones. Vale con cualquier signo.

- **Base.** Con 0 aristas, solo el camino vacío de s a s, de largo 0.
- **Cada opción es un camino permitido.** $d_{i-1}(v)$ también tiene a lo sumo i aristas. Y un camino a u con $i-1$ aristas, más (u, v), tiene a lo sumo i.
- **Todo camino permitido está cubierto.** Sea P un camino con a lo sumo i aristas:
  - si tiene a lo sumo $i-1$, mide al menos $d_{i-1}(v)$;
  - si tiene i y termina en (u, v), mide al menos $d_{i-1}(u) + w(u, v)$.
- Si no hay camino, ninguna opción es finita, y las dos cantidades son ∞.

Nunca se dijo que el camino sea simple, ni que los pesos tengan un signo.

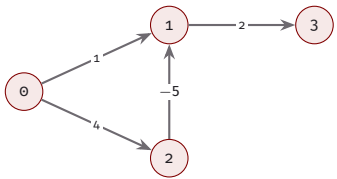
La tabla, fila por fila



i	0	1	2	3
0	0	∞	∞	∞
1	0	1	4	∞

La fila 0: solo el origen. La fila 1: las aristas que salen de 0.

La tabla, fila por fila

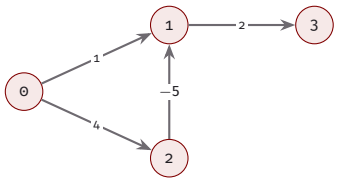


i	0	1	2	3
0	0	∞	∞	∞
1	0	1	4	∞
2	0	-1	4	3

$d_2(1) = \min(1,\ d_1(0) + 1,\ d_1(2) - 5) = -1.$

$d_2(3) = 3$: con dos aristas, lo mejor es $0 \to 1 \to 3$. No es un error: la fila responde otra pregunta.

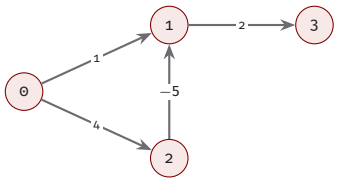
La tabla, fila por fila



i	0	1	2	3
0	0	∞	∞	∞
1	0	1	4	∞
2	0	-1	4	3
3	0	-1	4	1

$d_3(3) = d_2(1) + 2 = 1$. **Aquí aparece el camino que Dijkstra no vio.**

La tabla, fila por fila



i	0	1	2	3
0	0	∞	∞	∞
1	0	1	4	∞
2	0	−1	4	3
3	0	−1	4	1
4	0	−1	4	1

A partir de la fila 3, nada cambia.

Cada fila revisa E aristas. Con $V - 1$ filas: $O(VE)$, si todo nodo es alcanzable.

Falta justificar por qué bastan $V - 1$.

Sin ciclos negativos bastan $V - 1$ aristas

Lema

Si desde s no se alcanza un ciclo negativo, entonces para todo v alcanzable, $\delta(v)$ existe, es el largo de un camino simple, y $\delta(v) = d_{V-1}(v)$.

Técnica: quitar ciclos de un camino, sin alargarlo.

Sin ciclos negativos bastan $V - 1$ aristas

Lema

Si desde s no se alcanza un ciclo negativo, entonces para todo v alcanzable, $\delta(v)$ existe, es el largo de un camino simple, y $\delta(v) = d_{V-1}(v)$.

Técnica: quitar ciclos de un camino, sin alargarlo.

1. Si un camino repite un nodo, entre las dos apariciones hay un ciclo.

Sin ciclos negativos bastan $V - 1$ aristas

Lema

Si desde s no se alcanza un ciclo negativo, entonces para todo v alcanzable, $\delta(v)$ existe, es el largo de un camino simple, y $\delta(v) = d_{V-1}(v)$.

Técnica: quitar ciclos de un camino, sin alargarlo.

1. Si un camino repite un nodo, entre las dos apariciones hay un ciclo.
2. Ese ciclo se alcanza desde s, así que pesa 0 o más.

Sin ciclos negativos bastan $V - 1$ aristas

Lema

Si desde s no se alcanza un ciclo negativo, entonces para todo v alcanzable, $\delta(v)$ existe, es el largo de un camino simple, y $\delta(v) = d_{V-1}(v)$.

Técnica: quitar ciclos de un camino, sin alargarlo.

1. Si un camino repite un nodo, entre las dos apariciones hay un ciclo.
2. Ese ciclo se alcanza desde s, así que pesa 0 o más.
3. Quitarlo deja un camino de s a v, con menos aristas y no más largo.

Sin ciclos negativos bastan $V - 1$ aristas

Lema

Si desde s no se alcanza un ciclo negativo, entonces para todo v alcanzable, $\delta(v)$ existe, es el largo de un camino simple, y $\delta(v) = d_{V-1}(v)$.

Técnica: quitar ciclos de un camino, sin alargarlo.

1. Si un camino repite un nodo, entre las dos apariciones hay un ciclo.
2. Ese ciclo se alcanza desde s, así que pesa 0 o más.
3. Quitarlo deja un camino de s a v, con menos aristas y no más largo.
4. Queda un camino simple. Hay finitos, así que hay un mínimo: $\delta(v)$.

Sin ciclos negativos bastan $V - 1$ aristas

Lema

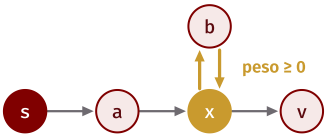
Si desde s no se alcanza un ciclo negativo, entonces para todo v alcanzable, $\delta(v)$ existe, es el largo de un camino simple, y $\delta(v) = d_{V-1}(v)$.

Técnica: quitar ciclos de un camino, sin alargarlo.

1. Si un camino repite un nodo, entre las dos apariciones hay un ciclo.
2. Ese ciclo se alcanza desde s, así que pesa 0 o más.
3. Quitarlo deja un camino de s a v, con menos aristas y no más largo.
4. Queda un camino simple. Hay finitos, así que hay un mínimo: $\delta(v)$.
5. Un camino simple visita a lo sumo V nodos: a lo sumo $V - 1$ aristas.

Un camino de V aristas repite un nodo

Un camino de V aristas, con V = 5 nodos

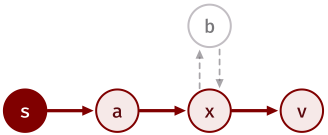


V aristas son V + 1 = 6 posiciones, y solo hay V = 5 nodos: alguno se repite. Aquí, x.



un ciclo que se alcanza desde s: pesa 0 o más

Sin el ciclo: menos aristas, y de largo menor o igual





Repitiendo esto queda un camino simple. Visita a lo sumo V nodos, y por lo tanto tiene a lo sumo V − 1 aristas.

Por eso bastan V − 1 filas.

SECCIÓN 03

Una sola fila

03

El programa de verdad: una sola fila

pseudocódigo

```
algoritmo bellman_ford(grafo, s)
    d[v] ← ∞ y pred[v] ← NULO para todo nodo v
    d[s] ← 0
    para vuelta desde 1 hasta V - 1 hacer
        para cada arista (u, v) hacer
            si d[u] ≠ ∞ y d[u] + w(u, v) < d[v] entonces
                d[v] ← d[u] + w(u, v)
                pred[v] ← u
            fin si
        fin para
    fin para
    devolver d, pred
fin algoritmo
```

El programa de verdad: una sola fila

pseudocódigo

```
algoritmo bellman_ford(grafo, s)
    d[v] ← ∞ y pred[v] ← NULO para todo nodo v
    d[s] ← 0
    para vuelta desde 1 hasta V - 1 hacer
        para cada arista (u, v) hacer
            si d[u] ≠ ∞ y d[u] + w(u, v) < d[v] entonces
                d[v] ← d[u] + w(u, v)
                pred[v] ← u
            fin si
        fin para
    fin para
    devolver d, pred
fin algoritmo
```

Dentro de una vuelta, $d[u]$ puede haber cambiado ya. En la mochila, eso resolvía otro problema.

$d[v]$ tras la vuelta i no es $d_i(v)$

las aristas en orden

vuelta	0	1	2	3
1	0	-1	4	1
2	0	-1	4	1

las aristas al revés

vuelta	0	1	2	3
1	0	1	4	inf
2	0	-1	4	3
3	0	-1	4	1
4	0	-1	4	1

Con el primer orden, tras una vuelta $d[3] = 1$, y $d_1(3) = \infty$.

$d[v]$ tras la vuelta i no es $d_i(v)$

las aristas en orden

vuelta	0	1	2	3
1	0	-1	4	1
2	0	-1	4	1

las aristas al revés

vuelta	0	1	2	3
1	0	1	4	inf
2	0	-1	4	3
3	0	-1	4	1
4	0	-1	4	1

Con el primer orden, tras una vuelta $d[3] = 1$, y $d_1(3) = \infty$.

Decir "$d[v] = d_i(v)$" es **falso**, y una demostración que lo afirme está mal.

$d[v]$ tras la vuelta i no es $d_i(v)$

las aristas en orden

vuelta	0	1	2	3
1	0	-1	4	1
2	0	-1	4	1

las aristas al revés

vuelta	0	1	2	3
1	0	1	4	inf
2	0	-1	4	3
3	0	-1	4	1
4	0	-1	4	1

Con el primer orden, tras una vuelta $d[3] = 1$, y $d_1(3) = \infty$.

Decir "$d[v] = d_i(v)$" es **falso**, y una demostración que lo afirme está mal.

Lo correcto es más débil, y alcanza.

Teorema: una fila basta

Durante toda la corrida

1. Si $d[v] \neq \infty$, $d[v]$ es el largo de algún camino de s a v.
2. Los valores de d nunca suben.
3. Después de la vuelta i: $d[v] \leq d_i(v)$.

Teorema: una fila basta

Durante toda la corrida

1. Si $d[v] \neq \infty$, $d[v]$ es el largo de algún camino de s a v.
2. Los valores de d nunca suben.
3. Después de la vuelta i: $d[v] \leq d_i(v)$.

- **(1) y (2).** Una relajación solo cambia $d[v]$ si lo baja. El valor nuevo es un camino a u más la arista (u, v).

Teorema: una fila basta

Durante toda la corrida

1. Si $d[v] \neq \infty$, $d[v]$ es el largo de algún camino de s a v.
2. Los valores de d nunca suben.
3. Después de la vuelta i: $d[v] \leq d_i(v)$.

- **(1) y (2).** Una relajación solo cambia $d[v]$ si lo baja. El valor nuevo es un camino a u más la arista (u, v).
- **(3), inducción sobre i.** $d_i(v)$ es $d_{i-1}(v)$ o $d_{i-1}(u) + w(u, v)$.

Teorema: una fila basta

Durante toda la corrida

1. Si $d[v] \neq \infty$, $d[v]$ es el largo de algún camino de s a v.
2. Los valores de d nunca suben.
3. Después de la vuelta i: $d[v] \leq d_i(v)$.

- **(1) y (2).** Una relajación solo cambia $d[v]$ si lo baja. El valor nuevo es un camino a u más la arista (u, v).
- **(3), inducción sobre i.** $d_i(v)$ es $d_{i-1}(v)$ o $d_{i-1}(u) + w(u, v)$.
- Primer caso: $d[v] \leq d_{i-1}(v)$, y no sube.

Teorema: una fila basta

Durante toda la corrida

1. Si $d[v] \neq \infty$, $d[v]$ es el largo de algún camino de s a v.
2. Los valores de d nunca suben.
3. Después de la vuelta i: $d[v] \leq d_i(v)$.

- **(1) y (2).** Una relajación solo cambia $d[v]$ si lo baja. El valor nuevo es un camino a u más la arista (u, v).
- **(3), inducción sobre i.** $d_i(v)$ es $d_{i-1}(v)$ o $d_{i-1}(u) + w(u, v)$.
- Primer caso: $d[v] \leq d_{i-1}(v)$, y no sube.
- Segundo caso: al mirar (u, v), se cumple $d[u] \leq d_{i-1}(u)$. **Que $d[u]$ haya cambiado en esta vuelta solo ayuda**: solo pudo bajar.

Teorema: una fila basta

Durante toda la corrida

1. Si $d[v] \neq \infty$, $d[v]$ es el largo de algún camino de s a v.
2. Los valores de d nunca suben.
3. Después de la vuelta i: $d[v] \leq d_i(v)$.

- **(1) y (2).** Una relajación solo cambia $d[v]$ si lo baja. El valor nuevo es un camino a u más la arista (u, v).
- **(3), inducción sobre i.** $d_i(v)$ es $d_{i-1}(v)$ o $d_{i-1}(u) + w(u, v)$.
- Primer caso: $d[v] \leq d_{i-1}(v)$, y no sube.
- Segundo caso: al mirar (u, v), se cumple $d[u] \leq d_{i-1}(u)$. **Que $d[u]$ haya cambiado en esta vuelta solo ayuda**: solo pudo bajar.
- Después de mirarla, $d[v] \leq d[u] + w(u, v) \leq d_i(v)$.

La conclusión: apretar entre dos cosas iguales

Sin ciclos negativos alcanzables, tras la vuelta $V - 1$:

$$\delta(v) \underset{(1)}{\leq} d[v] \underset{(3)}{\leq} d_{V-1}(v) \underset{\text{lema}}{=} \delta(v)$$

La conclusión: apretar entre dos cosas iguales

Sin ciclos negativos alcanzables, tras la vuelta $V - 1$:

$$\delta(v) \underset{(1)}{\leq} d[v] \underset{(3)}{\leq} d_{V-1}(v) \underset{\text{lema}}{=} \delta(v)$$

Nunca por encima de la tabla, y nunca por debajo de la distancia verdadera.

La conclusión: apretar entre dos cosas iguales

Sin ciclos negativos alcanzables, tras la vuelta $V - 1$:

$$\delta(v) \underset{(1)}{\leq} d[v] \underset{(3)}{\leq} d_{V-1}(v) \underset{\text{lema}}{=} \delta(v)$$

Nunca por encima de la tabla, y nunca por debajo de la distancia verdadera.

¿Por qué la mochila era distinta?

Allá, reusar un valor de esta fila era tomar un objeto dos veces: cambiaba el problema.
Aquí es usar una arista más de lo que la vuelta permite, y **tener más aristas nunca es ilegal**. La fila i era un andamio.

El orden importa para el tiempo, no para el resultado

terminal

```
Cuántas vueltas hacen falta, hasta una que no cambia nada, en una cadena de n nodos:
    n  aristas en orden  aristas al revés
   10                2                10
  100                2               100
 1000                2              1000
```

El orden importa para el tiempo, no para el resultado

terminal

```
Cuántas vueltas hacen falta, hasta una que no cambia nada, en una cadena de n nodos:
    n  aristas en orden  aristas al revés
   10                 2                10
  100                 2               100
 1000                 2              1000
```

- Parar si una vuelta no cambia nada es seguro: las siguientes tampoco cambian.

El orden importa para el tiempo, no para el resultado

terminal

```
Cuántas vueltas hacen falta, hasta una que no cambia nada, en una cadena de n nodos:
    n  aristas en orden  aristas al revés
   10                2                10
  100                2               100
 1000                2              1000
```

- Parar si una vuelta no cambia nada es seguro: las siguientes tampoco cambian.
- En orden, todo avanza en una vuelta. Al revés, una arista por vuelta.

El orden importa para el tiempo, no para el resultado

terminal

```
Cuántas vueltas hacen falta, hasta una que no cambia nada, en una cadena de n nodos:
    n  aristas en orden  aristas al revés
   10                2                10
  100                2               100
 1000                2              1000
```

- Parar si una vuelta no cambia nada es seguro: las siguientes tampoco cambian.
- En orden, todo avanza en una vuelta. Al revés, una arista por vuelta.
- Con parada temprana, el peor caso sigue siendo $\Theta(VE)$.

SECCIÓN 04

Ciclos negativos

04

Detectar: una vuelta más

Teorema

La vuelta V relaja alguna arista si y solo si desde s se alcanza un ciclo negativo.

Si no se alcanza ninguno, la vuelta V no relaja nada.

Detectar: una vuelta más

Teorema

La vuelta V relaja alguna arista si y solo si desde s se alcanza un ciclo negativo.

Si no se alcanza ninguno, la vuelta V no relaja nada.

- Tras la vuelta $V - 1$, $d[v] = \delta(v)$ en los alcanzables, y ∞ en los demás.

Detectar: una vuelta más

Teorema

La vuelta V relaja alguna arista si y solo si desde s se alcanza un ciclo negativo.

Si no se alcanza ninguno, la vuelta V no relaja nada.

- Tras la vuelta $V - 1$, $d[v] = \delta(v)$ en los alcanzables, y ∞ en los demás.
- Una relajación dejaría $d[v] < \delta(v)$.

Detectar: una vuelta más

Teorema
La vuelta V relaja alguna arista si y solo si desde s se alcanza un ciclo negativo.

Si no se alcanza ninguno, la vuelta V no relaja nada.

- Tras la vuelta $V - 1$, $d[v] = \delta(v)$ en los alcanzables, y ∞ en los demás.
- Una relajación dejaría $d[v] < \delta(v)$.
- Por (1), sería el largo de un camino más corto que el mínimo. Imposible.

Detectar: la otra dirección

Si la vuelta V no relaja nada, no se alcanza ningún ciclo negativo.

Detectar: la otra dirección

Si la vuelta V no relaja nada, no se alcanza ningún ciclo negativo.

- d no cambió en esa vuelta. Toda arista con $d[u] \neq \infty$ cumple $d[v] \leq d[u] + w(u, v)$.

Detectar: la otra dirección

Si la vuelta V no relaja nada, no se alcanza ningún ciclo negativo.

- d no cambió en esa vuelta. Toda arista con $d[u] \neq \infty$ cumple $d[v] \leq d[u] + w(u, v)$.
- Sea C un ciclo que se alcanza desde s. Cada nodo de C tiene un camino simple desde s, así que d es **finito** en él. **Aquí se usa que se alcanza.**

Detectar: la otra dirección

Si la vuelta V no relaja nada, no se alcanza ningún ciclo negativo.

- d no cambió en esa vuelta. Toda arista con $d[u] \neq \infty$ cumple $d[v] \leq d[u] + w(u, v)$.
- Sea C un ciclo que se alcanza desde s. Cada nodo de C tiene un camino simple desde s, así que d es **finito** en él. **Aquí se usa que se alcanza.**
- Sumando la desigualdad en las aristas de C:

$$\sum_{x \in C} d[x] \;\leq\; \sum_{x \in C} d[x] + \text{peso}(C)$$

Detectar: la otra dirección

Si la vuelta V no relaja nada, no se alcanza ningún ciclo negativo.

- d no cambió en esa vuelta. Toda arista con $d[u] \neq \infty$ cumple $d[v] \leq d[u] + w(u, v)$.
- Sea C un ciclo que se alcanza desde s. Cada nodo de C tiene un camino simple desde s, así que d es **finito** en él. **Aquí se usa que se alcanza.**
- Sumando la desigualdad en las aristas de C:

$$\sum_{x \in C} d[x] \;\leq\; \sum_{x \in C} d[x] + \text{peso}(C)$$

- Es la misma suma a cada lado, y es finita: se cancela. $0 \leq \text{peso}(C)$.

Detectar: la otra dirección

Si la vuelta V no relaja nada, no se alcanza ningún ciclo negativo.

- d no cambió en esa vuelta. Toda arista con $d[u] \neq \infty$ cumple $d[v] \leq d[u] + w(u, v)$.
- Sea C un ciclo que se alcanza desde s. Cada nodo de C tiene un camino simple desde s, así que d es **finito** en él. **Aquí se usa que se alcanza.**
- Sumando la desigualdad en las aristas de C:

$$\sum_{x \in C} d[x] \;\leq\; \sum_{x \in C} d[x] + \text{peso}(C)$$

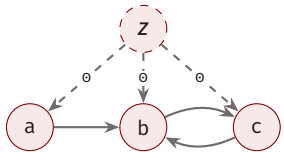
- Es la misma suma a cada lado, y es finita: se cancela. $0 \leq \text{peso}(C)$.

Error común

Bellman-Ford desde s **no ve** un ciclo negativo al que s no llega.

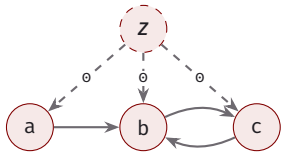
En cualquier parte del grafo: todos en 0

Si la pregunta es sobre todo el grafo: $d[v] \leftarrow 0$ y $\text{pred}[v] \leftarrow \texttt{NULO}$ para todos.



En cualquier parte del grafo: todos en 0

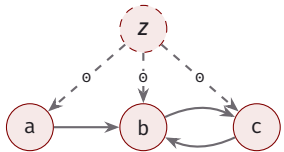
Si la pregunta es sobre todo el grafo: $d[v] \leftarrow 0$ y $\text{pred}[v] \leftarrow$ `NULO` para todos.



- Es como un origen nuevo que alcanza todo, y al que nadie entra.

En cualquier parte del grafo: todos en 0

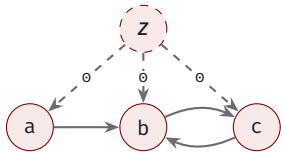
Si la pregunta es sobre todo el grafo: $d[v] \leftarrow 0$ y $\text{pred}[v] \leftarrow$ `NULO` para todos.



- Es como un origen nuevo que alcanza todo, y al que nadie entra.
- $d_i(v)$: el menor largo con a lo sumo i aristas **que termina en v y empieza donde sea**. $d_0(v) = 0$.

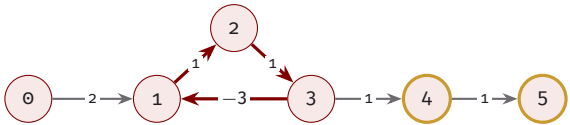
En cualquier parte del grafo: todos en 0

Si la pregunta es sobre todo el grafo: $d[v] \leftarrow 0$ y $\text{pred}[v] \leftarrow$ `NULO` para todos.



- Es como un origen nuevo que alcanza todo, y al que nadie entra.
- $d_i(v)$: el menor largo con a lo sumo i aristas **que termina en** v **y empieza donde sea**. $d_0(v) = 0$.
- Un camino simple sigue teniendo a lo sumo $V - 1$ aristas: V **vueltas**, y la última decide.

Recuperar el ciclo: la tentación falla



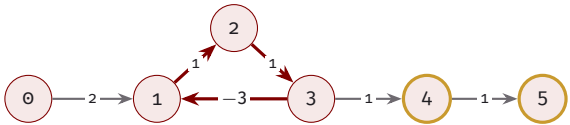
terminal

```
nodo relajado al final de la vuelta 6: 5
pred: -1 3 1 2 3 4

Leer directamente desde ese nodo, hasta volver a él (se corta a los 12 pasos):
  5 4 3 2 1 3 2 1 3 2 1 3 ...

Retroceder 6 veces y recién entonces leer:
  5 4 3 2 1 3 -> 2
ciclo: 3 1 2 -> 3, peso -1
```

Recuperar el ciclo: la tentación falla



terminal

```
nodo relajado al final de la vuelta 6: 5
pred: -1 3 1 2 3 4

Leer directamente desde ese nodo, hasta volver a él (se corta a los 12 pasos):
  5 4 3 2 1 3 2 1 3 2 1 3 ...

Retroceder 6 veces y recién entonces leer:
  5 4 3 2 1 3 -> 2
ciclo: 3 1 2 -> 3, peso -1
```

El nodo relajado cuelga del ciclo. Leer desde él: nunca se vuelve al 5.

Dos lemas sobre los predecesores

Lema 1

Si $\text{pred}[v] = u$, entonces $d[v] \geq d[u] + w(u, v)$.

Dos lemas sobre los predecesores

Lema 1

Si $\text{pred}[v] = u$, entonces $d[v] \geq d[u] + w(u, v)$.

Al relajar hay igualdad. Después, $d[u]$ solo baja. Si $d[v]$ cambia, es otra relajación.

Dos lemas sobre los predecesores

Lema 1

Si $\text{pred}[v] = u$, entonces $d[v] \geq d[u] + w(u, v)$.

Al relajar hay igualdad. Después, $d[u]$ solo baja. Si $d[v]$ cambia, es otra relajación.

Lema 2

Todo ciclo de predecesores tiene peso negativo.

Dos lemas sobre los predecesores

Lema 1

Si $\text{pred}[v] = u$, entonces $d[v] \geq d[u] + w(u, v)$.

Al relajar hay igualdad. Después, $d[u]$ solo baja. Si $d[v]$ cambia, es otra relajación.

Lema 2

Todo ciclo de predecesores tiene peso negativo.

- Miremos la relajación que lo cerró: $\text{pred}[v] \leftarrow u$. Lo demás ya forma un camino de v a u.

Dos lemas sobre los predecesores

Lema 1

Si $\text{pred}[v] = u$, entonces $d[v] \geq d[u] + w(u, v)$.

Al relajar hay igualdad. Después, $d[u]$ solo baja. Si $d[v]$ cambia, es otra relajación.

Lema 2

Todo ciclo de predecesores tiene peso negativo.

- Miremos la relajación que lo cerró: $\text{pred}[v] \leftarrow u$. Lo demás ya forma un camino de v a u.
- Lema 1, sumado por ese camino: $d[u] \geq d[v] +$ peso del camino.

Dos lemas sobre los predecesores

Lema 1

Si $\text{pred}[v] = u$, entonces $d[v] \geq d[u] + w(u, v)$.

Al relajar hay igualdad. Después, $d[u]$ solo baja. Si $d[v]$ cambia, es otra relajación.

Lema 2

Todo ciclo de predecesores tiene peso negativo.

- Miremos la relajación que lo cerró: $\text{pred}[v] \leftarrow u$. Lo demás ya forma un camino de v a u.
- Lema 1, sumado por ese camino: $d[u] \geq d[v] +$ peso del camino.
- La relajación es **estricta**: $d[u] + w(u, v) < d[v]$.

Dos lemas sobre los predecesores

Lema 1

Si $\text{pred}[v] = u$, entonces $d[v] \geq d[u] + w(u, v)$.

Al relajar hay igualdad. Después, $d[u]$ solo baja. Si $d[v]$ cambia, es otra relajación.

Lema 2

Todo ciclo de predecesores tiene peso negativo.

- Miremos la relajación que lo cerró: $\text{pred}[v] \leftarrow u$. Lo demás ya forma un camino de v a u.
- Lema 1, sumado por ese camino: $d[u] \geq d[v] +$ peso del camino.
- La relajación es **estricta**: $d[u] + w(u, v) < d[v]$.
- Juntas: peso del camino $+ w(u, v) < 0$. Es el peso del ciclo.

Teorema: retroceder V veces llega al ciclo

Técnica: si la cadena termina en vez de ciclarse, x no podría haberse relajado en la vuelta V.

Teorema: retroceder V veces llega al ciclo

Técnica: si la cadena termina en vez de ciclarse, x no podría haberse relajado en la vuelta V.

- Supongamos que los pred desde x llegan a un nodo sin predecesor. Ese nodo nunca se relajó: es s, con $d[s] = 0$.

Teorema: retroceder V veces llega al ciclo

Técnica: si la cadena termina en vez de ciclarse, x no podría haberse relajado en la vuelta V.

- Supongamos que los pred desde x llegan a un nodo sin predecesor. Ese nodo nunca se relajó: es s, con $d[s] = 0$.
- Hay un camino simple P de s a x. Lema 1 sumado por P: $d[x] \geq \text{peso}(P)$.

Teorema: retroceder V veces llega al ciclo

Técnica: si la cadena termina en vez de ciclarse, x no podría haberse relajado en la vuelta V.

- Supongamos que los pred desde x llegan a un nodo sin predecesor. Ese nodo nunca se relajó: es s, con $d[s] = 0$.
- Hay un camino simple P de s a x. Lema 1 sumado por P: $d[x] \geq \text{peso}(P)$.
- P tiene a lo sumo $V - 1$ aristas: $\text{peso}(P) \geq d_{V-1}(x)$.

Teorema: retroceder V veces llega al ciclo

Técnica: si la cadena termina en vez de ciclarse, x no podría haberse relajado en la vuelta V.

- Supongamos que los pred desde x llegan a un nodo sin predecesor. Ese nodo nunca se relajó: es s, con $d[s] = 0$.
- Hay un camino simple P de s a x. Lema 1 sumado por P: $d[x] \geq \text{peso}(P)$.
- P tiene a lo sumo $V - 1$ aristas: $\text{peso}(P) \geq d_{V-1}(x)$.
- Tras la vuelta $V - 1$, $d[x] \leq d_{V-1}(x)$. Y x se relajó en la vuelta V:

$$d[x] \;<\; d_{V-1}(x) \;\leq\; \text{peso}(P) \;\leq\; d[x]$$

Teorema: retroceder V veces llega al ciclo

Técnica: si la cadena termina en vez de ciclarse, x no podría haberse relajado en la vuelta V.

- Supongamos que los pred desde x llegan a un nodo sin predecesor. Ese nodo nunca se relajó: es s, con $d[s] = 0$.
- Hay un camino simple P de s a x. Lema 1 sumado por P: $d[x] \geq \text{peso}(P)$.
- P tiene a lo sumo $V - 1$ aristas: $\text{peso}(P) \geq d_{V-1}(x)$.
- Tras la vuelta $V - 1$, $d[x] \leq d_{V-1}(x)$. Y x se relajó en la vuelta V:

$$d[x] \;<\; d_{V-1}(x) \;\leq\; \text{peso}(P) \;\leq\; d[x]$$

- Contradicción. La cadena nunca termina: con V nodos, en a lo sumo $V - 1$ pasos entra a un ciclo.

Teorema: retroceder V veces llega al ciclo

Técnica: si la cadena termina en vez de ciclarse, x no podría haberse relajado en la vuelta V.

- Supongamos que los pred desde x llegan a un nodo sin predecesor. Ese nodo nunca se relajó: es s, con $d[s] = 0$.
- Hay un camino simple P de s a x. Lema 1 sumado por P: $d[x] \geq \text{peso}(P)$.
- P tiene a lo sumo $V - 1$ aristas: $\text{peso}(P) \geq d_{V-1}(x)$.
- Tras la vuelta $V - 1$, $d[x] \leq d_{V-1}(x)$. Y x se relajó en la vuelta V:

$$d[x] \;<\; d_{V-1}(x) \;\leq\; \text{peso}(P) \;\leq\; d[x]$$

- Contradicción. La cadena nunca termina: con V nodos, en a lo sumo $V - 1$ pasos entra a un ciclo.
- Tras V pasos está dentro. Por el lema 2, ese ciclo es negativo.

El algoritmo para recuperarlo

pseudocódigo

```
algoritmo recuperar_ciclo(pred, x, V)
    para paso desde 1 hasta V hacer
        x ← pred[x]
    fin para
    ciclo ← lista con x
    y ← pred[x]
    mientras y ≠ x hacer
        agregar y a ciclo
        y ← pred[y]
    fin mientras
    invertir ciclo
    devolver ciclo
fin algoritmo
```

Se invierte porque se leyó de fin a inicio.

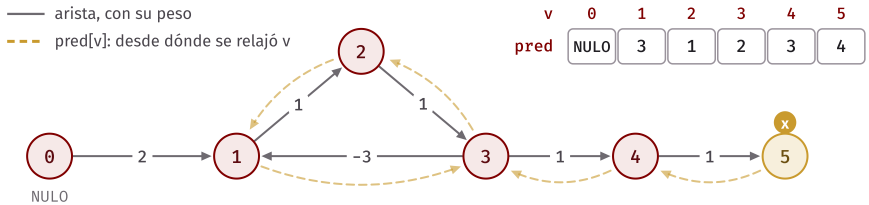
En el ejemplo: $3 \to 1 \to 2 \to 3$.

Sirve con cualquier nodo relajado en la vuelta V.

Error común

Tomar el nodo relajado como parte del ciclo. En grafos chicos suele estarlo, y el programa pasa las primeras pruebas.

Retroceder *V* veces, desde un nodo que cuelga



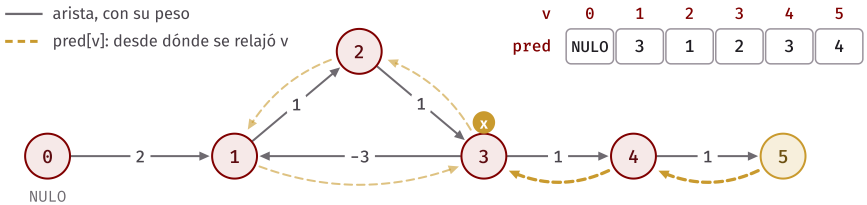
Sin retroceder: 5 4 3 2 1 3 2 1 3 2 1 3 ...

x tras cada paso:

0	1	2	3	4	5	6
5						

La tentación: seguir pred desde el 5, hasta volver a él.
Nunca se vuelve al 5: en el programa, es un ciclo infinito.

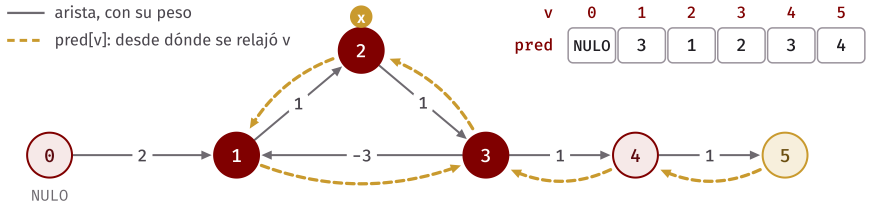
Retroceder *V* veces, desde un nodo que cuelga



Paso 2: x ← pred[4] = 3.

Ya está en el ciclo, pero el programa no lo sabe: no se detiene.

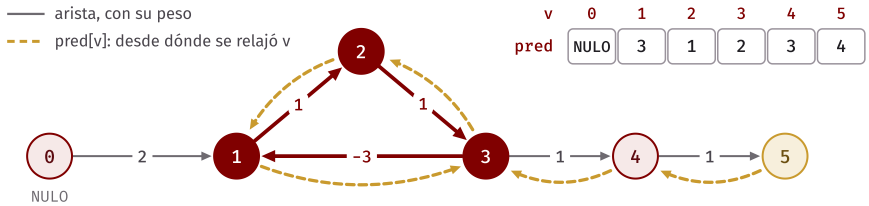
Retroceder *V* veces, desde un nodo que cuelga



Paso 6: x ← pred[3] = 2.

Tras V = 6 pasos, x = 2 está dentro del ciclo, y se puede leer.

Retroceder *V* veces, desde un nodo que cuelga



Sin retroceder: 5 4 3 2 1 3 2 1 3 2 1 3 ...

x tras cada paso:

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

se lee: 2 1 3

se invierte: 3 1 2 → 3

Se invierte: el ciclo 3 → 1 → 2 → 3, de peso −3 + 1 + 1 = −1.

Es el ciclo 1 → 2 → 3 → 1, escrito desde otro nodo.

SECCIÓN 05

Empezar en infinito

05

Primer error: `INT_MAX` como infinito

terminal

```
$ build/infinity_demo
infinity_demo.cpp:23:32: runtime error: signed integer overflow: 2147483647 + 5 cannot be represented in type 'int'
infinity_demo.cpp:24:47: runtime error: signed integer overflow: 2147483647 + 5 cannot be represented in type 'int'
Primer error: INT_MAX como infinito. Aristas 2->3 (5), 0->1 (7).
d con INT_MAX: 0 7 2147483647 -2147483644
correcto:      0 7 inf inf
```

Primer error: `INT_MAX` como infinito

terminal

```
$ build/infinity_demo
infinity_demo.cpp:23:32: runtime error: signed integer overflow: 2147483647 + 5 cannot be represented in type 'int'
infinity_demo.cpp:24:47: runtime error: signed integer overflow: 2147483647 + 5 cannot be represented in type 'int'
Primer error: INT_MAX como infinito. Aristas 2->3 (5), 0->1 (7).
d con INT_MAX: 0 7 2147483647 -2147483644
correcto:      0 7 inf inf
```

- `INT_MAX + 5` es overflow: **comportamiento indefinido**. El lenguaje no dice qué pasa.

Primer error: `INT_MAX` como infinito

terminal

```
$ build/infinity_demo
infinity_demo.cpp:23:32: runtime error: signed integer overflow: 2147483647 + 5 cannot be represented in type 'int'
infinity_demo.cpp:24:47: runtime error: signed integer overflow: 2147483647 + 5 cannot be represented in type 'int'
Primer error: INT_MAX como infinito. Aristas 2->3 (5), 0->1 (7).
d con INT_MAX: 0 7 2147483647 -2147483644
correcto:      0 7 inf inf
```

- `INT_MAX + 5` es overflow: **comportamiento indefinido**. El lenguaje no dice qué pasa.
- El sanitizer lo señala. En esta corrida dio la vuelta, y el nodo 3, que nadie alcanza, queda muy negativo.

Primer error: `INT_MAX` como infinito

terminal

```
$ build/infinity_demo
infinity_demo.cpp:23:32: runtime error: signed integer overflow: 2147483647 + 5 cannot be represented in type 'int'
infinity_demo.cpp:24:47: runtime error: signed integer overflow: 2147483647 + 5 cannot be represented in type 'int'
Primer error: INT_MAX como infinito. Aristas 2->3 (5), 0->1 (7).
d con INT_MAX: 0 7 2147483647 -2147483644
correcto:      0 7 inf inf
```

- `INT_MAX + 5` es overflow: **comportamiento indefinido**. El lenguaje no dice qué pasa.
- El sanitizer lo señala. En esta corrida dio la vuelta, y el nodo 3, que nadie alcanza, queda muy negativo.
- Otro compilador puede hacer otra cosa: no es un resultado que se pueda esperar.

Segundo error: infinito más un negativo

terminal

```
Segundo error: infinito + un negativo. Aristas 0->1 (7), 2->3 (-5).
sin revisar:   0 7 1000000000000 999999999995
revisando:     0 7 inf inf

Y un falso ciclo: 0->1 (7), y el ciclo 2->3 (-5), 3->2 (1), que el origen no alcanza.
sin revisar: la vuelta 4 cambió algo: sí
revisando:   la vuelta 4 cambió algo: no
```

Segundo error: infinito más un negativo

terminal

```
Segundo error: infinito + un negativo. Aristas 0->1 (7), 2->3 (-5).
sin revisar:   0 7 1000000000000 999999999995
revisando:     0 7 inf inf

Y un falso ciclo: 0->1 (7), y el ciclo 2->3 (-5), 3->2 (1), que el origen no alcanza.
sin revisar: la vuelta 4 cambió algo: sí
revisando:   la vuelta 4 cambió algo: no
```

- No hay overflow: $10^{12} - 5$ **sí es menor** que 10^{12}.

Segundo error: infinito más un negativo

terminal

```
Segundo error: infinito + un negativo. Aristas 0->1 (7), 2->3 (-5).
sin revisar:   0 7 1000000000000 999999999995
revisando:     0 7 inf inf

Y un falso ciclo: 0->1 (7), y el ciclo 2->3 (-5), 3->2 (1), que el origen no alcanza.
sin revisar: la vuelta 4 cambió algo: sí
revisando:   la vuelta 4 cambió algo: no
```

- No hay overflow: $10^{12} - 5$ **sí es menor** que 10^{12}.
- El nodo 3 “mejora” con un valor que no es ningún camino desde s. Se rompe (1), y todo lo que sigue.

Segundo error: infinito más un negativo

terminal

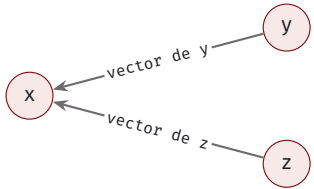
```
Segundo error: infinito + un negativo. Aristas 0->1 (7), 2->3 (-5).
sin revisar:   0 7 1000000000000 999999999995
revisando:     0 7 inf inf

Y un falso ciclo: 0->1 (7), y el ciclo 2->3 (-5), 3->2 (1), que el origen no alcanza.
sin revisar: la vuelta 4 cambió algo: sí
revisando:   la vuelta 4 cambió algo: no
```

- No hay overflow: $10^{12} - 5$ **sí es menor** que 10^{12}.
- El nodo 3 “mejora” con un valor que no es ningún camino desde s. Se rompe (1), y todo lo que sigue.
- **La regla:** de un nodo no alcanzado no se propaga nada. Un valor sin overflow, y la pregunta antes de sumar.

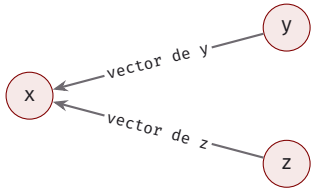
Dónde se usa: el vector de distancias

Para actualizar $d[v]$ hacen falta los valores de los vecinos y el peso de la arista. **Nada más.**



Dónde se usa: el vector de distancias

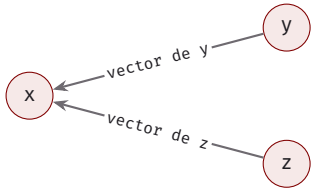
Para actualizar $d[v]$ hacen falta los valores de los vecinos y el peso de la arista. **Nada más.**



- Cada router guarda su estimación de la distancia a cada destino, y se la envía a sus vecinos.

Dónde se usa: el vector de distancias

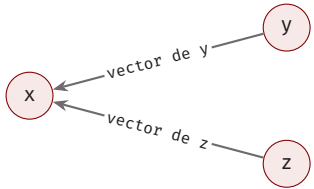
Para actualizar $d[v]$ hacen falta los valores de los vecinos y el peso de la arista. **Nada más.**



- Cada router guarda su estimación de la distancia a cada destino, y se la envía a sus vecinos.
- Al recibir un vector, relaja.

Dónde se usa: el vector de distancias

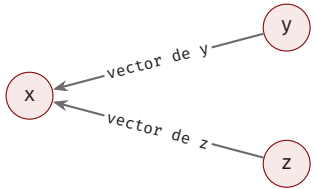
Para actualizar $d[v]$ hacen falta los valores de los vecinos y el peso de la arista. **Nada más.**



- Cada router guarda su estimación de la distancia a cada destino, y se la envía a sus vecinos.
- Al recibir un vector, relaja.
- RIP es un protocolo que usa esta idea.

Dónde se usa: el vector de distancias

Para actualizar $d[v]$ hacen falta los valores de los vecinos y el peso de la arista. **Nada más.**



- Cada router guarda su estimación de la distancia a cada destino, y se la envía a sus vecinos.
- Al recibir un vector, relaja.
- RIP es un protocolo que usa esta idea.

Los detalles los verán en Redes de Computadoras. Aquí importa que es la misma recurrencia, sin que nadie vea el mapa entero.

SECCIÓN 06

Todos los pares

06

V veces un algoritmo de un origen

Queremos $\delta(i, j)$ para todo par. La primera idea es correr V veces lo que ya tenemos:

Método	Costo	Condición
$V\times$ Dijkstra con heap	$O(V(V+E)\log V)$	Pesos no negativos
$V\times$ Bellman-Ford	$O(V^2E)$	Sin ciclos negativos

V veces un algoritmo de un origen

Queremos $\delta(i, j)$ para todo par. La primera idea es correr V veces lo que ya tenemos:

Método	Costo	Condición
$V\times$ Dijkstra con heap	$O(V(V+E)\log V)$	Pesos no negativos
$V\times$ Bellman-Ford	$O(V^2E)$	Sin ciclos negativos

En un grafo denso, $V\times$ Bellman-Ford es $O(V^4)$.

V veces un algoritmo de un origen

Queremos $\delta(i, j)$ para todo par. La primera idea es correr V veces lo que ya tenemos:

Método	Costo	Condición
$V\times$ Dijkstra con heap	$O(V(V+E)\log V)$	Pesos no negativos
$V\times$ Bellman-Ford	$O(V^2E)$	Sin ciclos negativos

En un grafo denso, $V\times$ Bellman-Ford es $O(V^4)$.

La idea de hoy

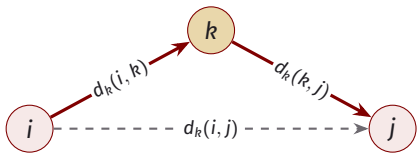
Otro parámetro: no cuántas aristas, sino **por dónde** puede pasar el camino. Baja a $\Theta(V^3)$.

Floyd-Warshall: el subproblema

Los **intermedios** de un camino son todos sus nodos, menos el primero y el último.

Subproblema, sin ciclos negativos

$d_k(i, j)$: el menor largo de un camino de i a j con intermedios entre los k **primeros nodos**: $0, \ldots, k-1$. La respuesta es $d_V(i, j)$.

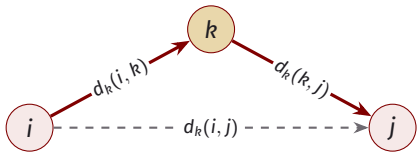


Floyd-Warshall: el subproblema

Los **intermedios** de un camino son todos sus nodos, menos el primero y el último.

Subproblema, sin ciclos negativos

$d_k(i,j)$: el menor largo de un camino de i a j con intermedios entre los k **primeros nodos**: $0, \ldots, k-1$. La respuesta es $d_V(i,j)$.



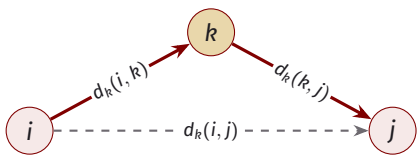
$$d_{k+1}(i,j) = \min\big(d_k(i,j),\ d_k(i,k) + d_k(k,j)\big)$$

Floyd-Warshall: el subproblema

Los **intermedios** de un camino son todos sus nodos, menos el primero y el último.

Subproblema, sin ciclos negativos

$d_k(i, j)$: el menor largo de un camino de i a j con intermedios entre los k **primeros nodos**: $0, \ldots, k-1$. La respuesta es $d_V(i, j)$.



$$d_{k+1}(i,j) = \min\big(d_k(i,j),\ d_k(i,k) + d_k(k,j)\big)$$

d_0 no permite ningún intermedio. d_{k+1} agrega al **nodo** k.

Teorema: la recurrencia de Floyd-Warshall

Técnica: inducción sobre k. La decisión es si el camino pasa por k.

Teorema: la recurrencia de Floyd-Warshall

Técnica: inducción sobre k. La decisión es si el camino pasa por k.

- **Base.** Sin intermedios: la arista directa o, con $i = j$, el camino vacío.

Teorema: la recurrencia de Floyd-Warshall

Técnica: inducción sobre k. La decisión es si el camino pasa por k.

- **Base.** Sin intermedios: la arista directa o, con $i = j$, el camino vacío.
- **Cada opción es un camino permitido.** Pegar un camino de i a k y uno de k a j, con intermedios entre los k primeros, da uno con intermedios entre los $k + 1$ primeros.

Teorema: la recurrencia de Floyd-Warshall

Técnica: inducción sobre k. La decisión es si el camino pasa por k.

- **Base.** Sin intermedios: la arista directa o, con $i = j$, el camino vacío.
- **Cada opción es un camino permitido.** Pegar un camino de i a k y uno de k a j, con intermedios entre los k primeros, da uno con intermedios entre los $k + 1$ primeros.
- **Todo camino permitido está cubierto.** Tomemos P. Sin ciclos negativos, quitarle ciclos no lo alarga: P es simple.

Teorema: la recurrencia de Floyd-Warshall

Técnica: inducción sobre k. La decisión es si el camino pasa por k.

- **Base.** Sin intermedios: la arista directa o, con $i = j$, el camino vacío.
- **Cada opción es un camino permitido.** Pegar un camino de i a k y uno de k a j, con intermedios entre los k primeros, da uno con intermedios entre los $k + 1$ primeros.
- **Todo camino permitido está cubierto.** Tomemos P. Sin ciclos negativos, quitarle ciclos no lo alarga: P es simple.
- Si k no es intermedio de P: cuenta en $d_k(i, j)$.

Teorema: la recurrencia de Floyd-Warshall

Técnica: inducción sobre k. La decisión es si el camino pasa por k.

- **Base.** Sin intermedios: la arista directa o, con $i = j$, el camino vacío.
- **Cada opción es un camino permitido.** Pegar un camino de i a k y uno de k a j, con intermedios entre los k primeros, da uno con intermedios entre los $k + 1$ primeros.
- **Todo camino permitido está cubierto.** Tomemos P. Sin ciclos negativos, quitarle ciclos no lo alarga: P es simple.
- Si k no es intermedio de P: cuenta en $d_k(i, j)$.
- Si lo es, **aparece una sola vez**. Se parte en dos tramos sin k como intermedio.

Teorema: la recurrencia de Floyd-Warshall

Técnica: inducción sobre k. La decisión es si el camino pasa por k.

- **Base.** Sin intermedios: la arista directa o, con $i = j$, el camino vacío.
- **Cada opción es un camino permitido.** Pegar un camino de i a k y uno de k a j, con intermedios entre los k primeros, da uno con intermedios entre los $k+1$ primeros.
- **Todo camino permitido está cubierto.** Tomemos P. Sin ciclos negativos, quitarle ciclos no lo alarga: P es simple.
- Si k no es intermedio de P: cuenta en $d_k(i, j)$.
- Si lo es, **aparece una sola vez**. Se parte en dos tramos sin k como intermedio.

El centro de la demostración

"Aparece una sola vez" es lo que permite partir en dos, y por eso se pide que no haya ciclos negativos.

El programa: tres ciclos, $\Theta(V^3)$

pseudocódigo

```
algoritmo floyd_warshall(grafo)
    d[i][j] ← d_0(i, j) para todo par (i, j)
    para k desde 0 hasta V - 1 hacer
        si d[k][k] < 0 entonces
            devolver "hay un ciclo negativo"
        fin si
        para i desde 0 hasta V - 1 hacer
            para j desde 0 hasta V - 1 hacer
                si d[i][k] ≠ ∞ y d[k][j] ≠ ∞ y d[i][k] + d[k][j] < d[i][j] entonces
                    d[i][j] ← d[i][k] + d[k][j]
                fin si
            fin para
        fin para
    fin para
    devolver d
fin algoritmo
```

Paso a paso: cada matriz responde otra pregunta

`d_0, sin nodos intermedios`

	0	1	2	3
0	0	4	∞	9
1	∞	0	-2	∞
2	3	∞	0	1
3	∞	5	∞	0

Aristas: $0\to1$ (4), $0\to3$ (9), $1\to2$ (-2), $2\to3$ (1), $3\to1$ (5), $2\to0$ (3).

Paso a paso: cada matriz responde otra pregunta

`d_1, permite al 0`

	0	1	2	3
0	0	4	∞	9
1	∞	0	-2	∞
2	3	7	0	1
3	∞	5	∞	0

$$2 \to 0 \to 1 = 3 + 4 = 7.$$

Paso a paso: cada matriz responde otra pregunta

`d_2, permite al 1`

	0	1	2	3
0	0	4	2	9
1	∞	0	-2	∞
2	3	7	0	1
3	∞	5	3	0

$0 \to 1 \to 2 = 2$, y $3 \to 1 \to 2 = 3$.

Paso a paso: cada matriz responde otra pregunta

`d_3, permite al 2`

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

$0 \to 3$ baja de 9 a $2 + 1 = 3$, y aparecen $1 \to 0$ y $1 \to 3$.

Paso a paso: cada matriz responde otra pregunta

`d_4, permite al 3`

	0	1	2	3
0	0	4	2	3
1	1	0	-2	-1
2	3	6	0	1
3	6	5	3	0

La casilla de 2 a 1 baja de 7 a $1 + 5 = 6$. Solo esta última matriz da las distancias.

Por qué se puede actualizar en el mismo lugar

El programa guarda una sola matriz. Cuando lee $d[i][k]$, ¿es de la capa k o de la $k+1$?

Lema

Sin ciclos negativos, en la vuelta de k no cambia ninguna casilla de la fila k ni de la columna k.

Por qué se puede actualizar en el mismo lugar

El programa guarda una sola matriz. Cuando lee $d[i][k]$, ¿es de la capa k o de la $k+1$?

Lema

Sin ciclos negativos, en la vuelta de k no cambia ninguna casilla de la fila k ni de la columna k.

- $d[k][k] = 0$: el camino vacío mide 0, y sin ciclos negativos ningún camino cerrado mide menos.

Por qué se puede actualizar en el mismo lugar

El programa guarda una sola matriz. Cuando lee $d[i][k]$, ¿es de la capa k o de la $k+1$?

Lema

Sin ciclos negativos, en la vuelta de k no cambia ninguna casilla de la fila k ni de la columna k.

- $d[k][k] = 0$: el camino vacío mide 0, y sin ciclos negativos ningún camino cerrado mide menos.
- Fila k: $d[k][k] + d[k][j] = d[k][j]$, que no es menor que $d[k][j]$.

Por qué se puede actualizar en el mismo lugar

El programa guarda una sola matriz. Cuando lee $d[i][k]$, ¿es de la capa k o de la $k+1$?

Lema

Sin ciclos negativos, en la vuelta de k no cambia ninguna casilla de la fila k ni de la columna k.

- $d[k][k] = 0$: el camino vacío mide 0, y sin ciclos negativos ningún camino cerrado mide menos.
- Fila k: $d[k][k] + d[k][j] = d[k][j]$, que no es menor que $d[k][j]$.
- Columna k: $d[i][k] + d[k][k] = d[i][k]$, igual.

Por qué se puede actualizar en el mismo lugar

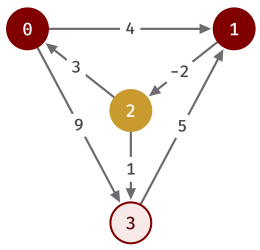
El programa guarda una sola matriz. Cuando lee $d[i][k]$, ¿es de la capa k o de la $k+1$?

Lema

Sin ciclos negativos, en la vuelta de k no cambia ninguna casilla de la fila k ni de la columna k.

- $d[k][k] = 0$: el camino vacío mide 0, y sin ciclos negativos ningún camino cerrado mide menos.
- Fila k: $d[k][k] + d[k][j] = d[k][j]$, que no es menor que $d[k][j]$.
- Columna k: $d[i][k] + d[k][k] = d[i][k]$, igual.
- Lo que se lee en la vuelta de k es exactamente $d_k(i,k)$ y $d_k(k,j)$, en cualquier orden de i y j.

La vuelta de $k = 2$: la fila y la columna 2 no cambian



d_2: intermedios en {0, 1}

	0	1	2	3
0	0	4	2	9
1	∞	0	-2	∞
2	3	7	0	1
3	∞	5	3	0

k = 2

d_3: intermedios en {0, 1, 2}

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

Lo que cambia, leyendo solo de la fila y la columna 2:

```
d[0][2] + d[2][3]   2 + 1 = 3      antes 9
d[1][2] + d[2][0]   -2 + 3 = 1     antes ∞
d[1][2] + d[2][3]   -2 + 1 = -1    antes ∞
d[3][2] + d[2][0]   3 + 3 = 6      antes ∞
```

d[2][2] = 0: en la fila 2, d[2][2] + d[2][j] = d[2][j]; en la columna 2, d[i][2] + d[2][2] = d[i][2].
Ninguna mejora, así que en el mismo lugar se lee lo mismo, en cualquier orden.

El error: k por dentro

terminal

```
El ciclo de k por dentro.
Aristas: 1->3 (5), 3->2 (5), 2->0 (9), 1->0 (30)
k por fuera:
    0  inf  inf  inf
   19    0   10    5
    9  inf    0  inf
   14  inf    5    0
k por dentro:
    0  inf  inf  inf
   30    0   10    5
    9  inf    0  inf
   14  inf    5    0
```

El bueno es $1 \to 3 \to 2 \to 0$: 19. Con k adentro, $d[1][0]$ prueba cada intermedio **una sola vez**:

k	$d[1][k] + d[k][0]$

El error: k por dentro

terminal

```
El ciclo de k por dentro.
Aristas: 1->3 (5), 3->2 (5), 2->0 (9), 1->0 (30)
k por fuera:
    0  inf  inf  inf
   19    0   10    5
    9  inf    0  inf
   14  inf    5    0
k por dentro:
    0  inf  inf  inf
   30    0   10    5
    9  inf    0  inf
   14  inf    5    0
```

El bueno es $1 \to 3 \to 2 \to 0$: 19. Con k adentro, $d[1][0]$ prueba cada intermedio **una sola vez**:

k	$d[1][k] + d[k][0]$
0, 1	$30 + 0$, $0 + 30$

El error: k por dentro

terminal

```
El ciclo de k por dentro.
Aristas: 1->3 (5), 3->2 (5), 2->0 (9), 1->0 (30)
k por fuera:
    0  inf  inf  inf
   19    0   10    5
    9  inf    0  inf
   14  inf    5    0
k por dentro:
    0  inf  inf  inf
   30    0   10    5
    9  inf    0  inf
   14  inf    5    0
```

El bueno es $1 \to 3 \to 2 \to 0$: 19. Con k adentro, $d[1][0]$ prueba cada intermedio **una sola vez**:

k	$d[1][k] + d[k][0]$
0, 1	$30 + 0$, $0 + 30$
2	$\infty + 9$: se llena después

El error: k por dentro

```
terminal

El ciclo de k por dentro.
Aristas: 1->3 (5), 3->2 (5), 2->0 (9), 1->0 (30)
k por fuera:
    0  inf  inf  inf
   19    0   10    5
    9  inf    0  inf
   14  inf    5    0
k por dentro:
    0  inf  inf  inf
   30    0   10    5
    9  inf    0  inf
   14  inf    5    0
```

El bueno es $1 \to 3 \to 2 \to 0$: 19. Con k adentro, $d[1][0]$ prueba cada intermedio **una sola vez**:

k	$d[1][k] + d[k][0]$
0, 1	$30 + 0$, $0 + 30$
2	$\infty + 9$: se llena después
3	$5 + \infty$: de una fila posterior

El error: k por dentro

terminal

```
El ciclo de k por dentro.
Aristas: 1->3 (5), 3->2 (5), 2->0 (9), 1->0 (30)
k por fuera:
    0  inf  inf  inf
   19    0   10    5
    9  inf    0  inf
   14  inf    5    0
k por dentro:
    0  inf  inf  inf
   30    0   10    5
    9  inf    0  inf
   14  inf    5    0
```

El bueno es $1 \to 3 \to 2 \to 0$: 19. Con k adentro, $d[1][0]$ prueba cada intermedio **una sola vez**:

k	$d[1][k] + d[k][0]$
0, 1	$30 + 0$, $0 + 30$
2	$\infty + 9$: se llena después
3	$5 + \infty$: de una fila posterior

Luego esas casillas se llenan, y **nadie vuelve a** $d[1][0]$.

El error: k por dentro

terminal

```
El ciclo de k por dentro.
Aristas: 1->3 (5), 3->2 (5), 2->0 (9), 1->0 (30)
k por fuera:
    0  inf  inf  inf
   19    0   10    5
    9  inf    0  inf
   14  inf    5    0
k por dentro:
    0  inf  inf  inf
   30    0   10    5
    9  inf    0  inf
   14  inf    5    0
```

El bueno es $1 \to 3 \to 2 \to 0$: 19. Con k adentro, $d[1][0]$ prueba cada intermedio **una sola vez**:

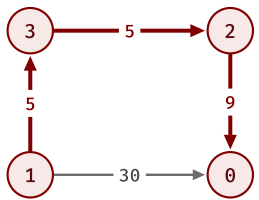
k	$d[1][k] + d[k][0]$
0, 1	$30 + 0$, $0 + 30$
2	$\infty + 9$: se llena después
3	$5 + \infty$: de una fila posterior

Luego esas casillas se llenan, y **nadie vuelve a** $d[1][0]$.

k **es el parámetro del subproblema: va por fuera.**

Por qué $d[1][0]$ se queda en 30

El camino bueno mide 19



5 + 5 + 9 = 19

directa: 30

Al calcular d[1][0], con k por dentro

i \ j	0	1	2	3
0	0	∞	∞	∞
1	30	0	∞	5
2	9	∞	0	∞
3	∞	∞	5	0

d[1][k] + d[k][0], una vez por k:

```
k = 0   30 + 0
k = 1   0 + 30
k = 2   ∞ + 9
k = 3   5 + ∞
```

Queda 30.

Punteadas: las que todavía no se calculan.

d[1][2]: se calcula después, en esta misma fila. Llegará a 10.

d[3][0]: de una fila posterior. Llegará a 14.

Esas casillas se vuelven finitas (10 y 14), pero nadie vuelve a d[1][0]: da 30, y con k por fuera, 19.

La capa k se llena completa antes de pasar a la k + 1, y por eso k va por fuera.

Con pruebas chiquitas no se nota

terminal

```
Grafos al azar en los que k por dentro se equivoca (20000 por fila, 35 % de aristas, pesos 1 a 9):
  n = 3: 0
  n = 4: 1534
  n = 5: 5272
  n = 8: 17060
```

Con pruebas chiquitas no se nota

terminal

```
Grafos al azar en los que k por dentro se equivoca (20000 por fila, 35 % de aristas, pesos 1 a 9):
  n = 3: 0
  n = 4: 1534
  n = 5: 5272
  n = 8: 17060
```

- Con 3 nodos, esta prueba no encontró ningún error. Con 4, menos de 1 de cada 10.

Con pruebas chiquitas no se nota

terminal

```
Grafos al azar en los que k por dentro se equivoca (20000 por fila, 35 % de aristas, pesos 1 a 9):
  n = 3: 0
  n = 4: 1534
  n = 5: 5272
  n = 8: 17060
```

- Con 3 nodos, esta prueba no encontró ningún error. Con 4, menos de 1 de cada 10.
- Compila, termina y da números creíbles.

Con pruebas chiquitas no se nota

terminal

```
Grafos al azar en los que k por dentro se equivoca (20000 por fila, 35 % de aristas, pesos 1 a 9):
  n = 3: 0
  n = 4: 1534
  n = 5: 5272
  n = 8: 17060
```

- Con 3 nodos, esta prueba no encontró ningún error. Con 4, menos de 1 de cada 10.
- Compila, termina y da números creíbles.

Error común

Poner el ciclo de k por dentro. No se ve mirando la salida: hay que recordar por qué va por fuera.

Detectar un ciclo negativo con la diagonal

Teorema

Hay un ciclo negativo si y solo si el programa ve $d[k][k] < 0$ al empezar la vuelta de k.

terminal

```
La diagonal. Aristas: 0->1 (1), 1->2 (-3), 2->1 (2), 2->3 (1)
  antes de k = 0, diagonal: 0 0 0 0
  antes de k = 1, diagonal: 0 0 0 0
  antes de k = 2, diagonal: 0 0 -1 0
  d[2][2] < 0: hay un ciclo negativo, y se detiene
```

Detectar un ciclo negativo con la diagonal

Teorema

Hay un ciclo negativo si y solo si el programa ve $d[k][k] < 0$ al empezar la vuelta de k.

terminal

```
La diagonal. Aristas: 0->1 (1), 1->2 (-3), 2->1 (2), 2->3 (1)
  antes de k = 0, diagonal: 0 0 0 0
  antes de k = 1, diagonal: 0 0 0 0
  antes de k = 2, diagonal: 0 0 -1 0
  d[2][2] < 0: hay un ciclo negativo, y se detiene
```

- **Si lo ve, hay uno.** Cada casilla es el largo de algún camino. Un camino cerrado negativo se descompone en ciclos simples, y alguno es negativo.

Si hay un ciclo negativo, el programa lo ve

Una cota que vale siempre, aunque haya ciclos negativos:

> **Tras procesar** $0, \ldots, k-1$
>
> $d[i][j]$, incluso con $i = j$, no supera el largo de ningún camino de i a j con intermedios distintos entre los k primeros.

Si hay un ciclo negativo, el programa lo ve

Una cota que vale siempre, aunque haya ciclos negativos:

> **Tras procesar** $0, \ldots, k-1$
> $d[i][j]$, incluso con $i = j$, no supera el largo de ningún camino de i a j con intermedios distintos entre los k primeros.

- Si el camino no usa a k: hipótesis, y los valores solo bajan.

Si hay un ciclo negativo, el programa lo ve

Una cota que vale siempre, aunque haya ciclos negativos:

> **Tras procesar** $0, \ldots, k-1$
> $d[i][j]$, incluso con $i = j$, no supera el largo de ningún camino de i a j con intermedios distintos entre los k primeros.

- Si el camino no usa a k: hipótesis, y los valores solo bajan.
- Si lo usa, lo usa una vez, y se parte en dos tramos.

Si hay un ciclo negativo, el programa lo ve

Una cota que vale siempre, aunque haya ciclos negativos:

> **Tras procesar** $0, \ldots, k-1$
> $d[i][j]$, incluso con $i = j$, no supera el largo de ningún camino de i a j con intermedios distintos entre los k primeros.

- Si el camino no usa a k: hipótesis, y los valores solo bajan.
- Si lo usa, lo usa una vez, y se parte en dos tramos.
- Sea C un ciclo negativo simple, y h **su nodo mayor**.

Si hay un ciclo negativo, el programa lo ve

Una cota que vale siempre, aunque haya ciclos negativos:

> **Tras procesar** $0, \ldots, k-1$
> $d[i][j]$, incluso con $i = j$, no supera el largo de ningún camino de i a j con intermedios distintos entre los k primeros.

- Si el camino no usa a k: hipótesis, y los valores solo bajan.
- Si lo usa, lo usa una vez, y se parte en dos tramos.
- Sea C un ciclo negativo simple, y h **su nodo mayor**.
- C es un camino de h a h, con intermedios distintos, todos menores que h.

Si hay un ciclo negativo, el programa lo ve

Una cota que vale siempre, aunque haya ciclos negativos:

> **Tras procesar** $0, \ldots, k-1$
> $d[i][j]$, incluso con $i = j$, no supera el largo de ningún camino de i a j con intermedios distintos entre los k primeros.

- Si el camino no usa a k: hipótesis, y los valores solo bajan.
- Si lo usa, lo usa una vez, y se parte en dos tramos.
- Sea C un ciclo negativo simple, y h **su nodo mayor**.
- C es un camino de h a h, con intermedios distintos, todos menores que h.
- Al empezar la vuelta de h, si no se detuvo antes: $d[h][h] \leq \text{peso}(C) < 0$.

Por qué se detiene: los valores crecen muchísimo

terminal

```
Sin detenerse: 40 nodos, todas las aristas de peso -1.
  tras la vuelta k =  0: menor valor -2.000e+00
  tras la vuelta k =  1: menor valor -1.200e+01
  tras la vuelta k =  2: menor valor -7.200e+01
  tras la vuelta k =  3: menor valor -4.320e+02
  tras la vuelta k =  4: menor valor -2.592e+03
  tras la vuelta k =  5: menor valor -1.555e+04
  tras la vuelta k = 24: menor valor -9.477e+18  (ya no cabe en long long)
```

Por qué se detiene: los valores crecen muchísimo

terminal

```
Sin detenerse: 40 nodos, todas las aristas de peso -1.
  tras la vuelta k =  0: menor valor -2.000e+00
  tras la vuelta k =  1: menor valor -1.200e+01
  tras la vuelta k =  2: menor valor -7.200e+01
  tras la vuelta k =  3: menor valor -4.320e+02
  tras la vuelta k =  4: menor valor -2.592e+03
  tras la vuelta k =  5: menor valor -1.555e+04
  tras la vuelta k = 24: menor valor -9.477e+18  (ya no cabe en long long)
```

- En este grafo, se multiplica por 6 en cada vuelta. Medido con `double`, para mostrarlo sin overflow.

Por qué se detiene: los valores crecen muchísimo

terminal

```
Sin detenerse: 40 nodos, todas las aristas de peso -1.
  tras la vuelta k =  0: menor valor -2.000e+00
  tras la vuelta k =  1: menor valor -1.200e+01
  tras la vuelta k =  2: menor valor -7.200e+01
  tras la vuelta k =  3: menor valor -4.320e+02
  tras la vuelta k =  4: menor valor -2.592e+03
  tras la vuelta k =  5: menor valor -1.555e+04
  tras la vuelta k = 24: menor valor -9.477e+18  (ya no cabe en long long)
```

- En este grafo, se multiplica por 6 en cada vuelta. Medido con `double`, para mostrarlo sin overflow.
- Con enteros sería overflow, y la diagonal dejaría de significar algo.

Por qué se detiene: los valores crecen muchísimo

terminal

```
Sin detenerse: 40 nodos, todas las aristas de peso -1.
  tras la vuelta k =  0: menor valor -2.000e+00
  tras la vuelta k =  1: menor valor -1.200e+01
  tras la vuelta k =  2: menor valor -7.200e+01
  tras la vuelta k =  3: menor valor -4.320e+02
  tras la vuelta k =  4: menor valor -2.592e+03
  tras la vuelta k =  5: menor valor -1.555e+04
  tras la vuelta k = 24: menor valor -9.477e+18  (ya no cabe en long long)
```

- En este grafo, se multiplica por 6 en cada vuelta. Medido con `double`, para mostrarlo sin overflow.
- Con enteros sería overflow, y la diagonal dejaría de significar algo.
- **La guardia:** mientras siga, no hay ciclos negativos entre los k primeros, y d_k guarda largos de caminos con intermedios distintos.

Las mediciones

terminal

```
$ build/all_pairs_bench
grafo                V        E V Dijkstra      V B-F        F-W
disperso, E=4V     100      399      0.002      0.001      0.001  iguales
disperso, E=4V     200      797      0.009      0.003      0.006  iguales
disperso, E=4V     400     1595      0.030      0.010      0.042  iguales
completo           100     9900      0.010      0.004      0.001  iguales
completo           200    39800      0.066      0.040      0.009  iguales
completo           400   159600      0.447      0.336      0.061  iguales
cadena al revés    200      199      0.001      0.004      0.003  iguales
cadena al revés    400      399      0.005      0.025      0.025  iguales
cadena al revés    800      799      0.018      0.193      0.194  iguales
```

Segundos, una corrida con `-O2`. Dependen de la máquina.

Las mediciones

terminal

```
$ build/all_pairs_bench
grafo                V        E V Dijkstra      V B-F        F-W
disperso, E=4V     100      399      0.002      0.001      0.001  iguales
disperso, E=4V     200      797      0.009      0.003      0.006  iguales
disperso, E=4V     400     1595      0.030      0.010      0.042  iguales
completo           100     9900      0.010      0.004      0.001  iguales
completo           200    39800      0.066      0.040      0.009  iguales
completo           400   159600      0.447      0.336      0.061  iguales
cadena al revés    200      199      0.001      0.004      0.003  iguales
cadena al revés    400      399      0.005      0.025      0.025  iguales
cadena al revés    800      799      0.018      0.193      0.194  iguales
```

Segundos, una corrida con `-O2`. Dependen de la máquina.

- Completo: gana Floyd-Warshall. Disperso de 400: es el más lento.

Las mediciones

terminal

```
$ build/all_pairs_bench
grafo                V        E V Dijkstra      V B-F        F-W
disperso, E=4V     100      399      0.002      0.001      0.001  iguales
disperso, E=4V     200      797      0.009      0.003      0.006  iguales
disperso, E=4V     400     1595      0.030      0.010      0.042  iguales
completo           100     9900      0.010      0.004      0.001  iguales
completo           200    39800      0.066      0.040      0.009  iguales
completo           400   159600      0.447      0.336      0.061  iguales
cadena al revés    200      199      0.001      0.004      0.003  iguales
cadena al revés    400      399      0.005      0.025      0.025  iguales
cadena al revés    800      799      0.018      0.193      0.194  iguales
```

Segundos, una corrida con `-O2`. Dependen de la máquina.

- Completo: gana Floyd-Warshall. Disperso de 400: es el más lento.
- $V\times$ Bellman-Ford tiene parada temprana. Contándolas aparte, en grafos generados igual con $V = 400$: 8 vueltas en promedio, 11 como máximo. Esas filas no son su peor caso.

Las mediciones

terminal

```
$ build/all_pairs_bench
grafo                V        E V Dijkstra      V B-F        F-W
disperso, E=4V     100      399      0.002      0.001      0.001  iguales
disperso, E=4V     200      797      0.009      0.003      0.006  iguales
disperso, E=4V     400     1595      0.030      0.010      0.042  iguales
completo           100     9900      0.010      0.004      0.001  iguales
completo           200    39800      0.066      0.040      0.009  iguales
completo           400   159600      0.447      0.336      0.061  iguales
cadena al revés    200      199      0.001      0.004      0.003  iguales
cadena al revés    400      399      0.005      0.025      0.025  iguales
cadena al revés    800      799      0.018      0.193      0.194  iguales
```

Segundos, una corrida con `-O2`. Dependen de la máquina.

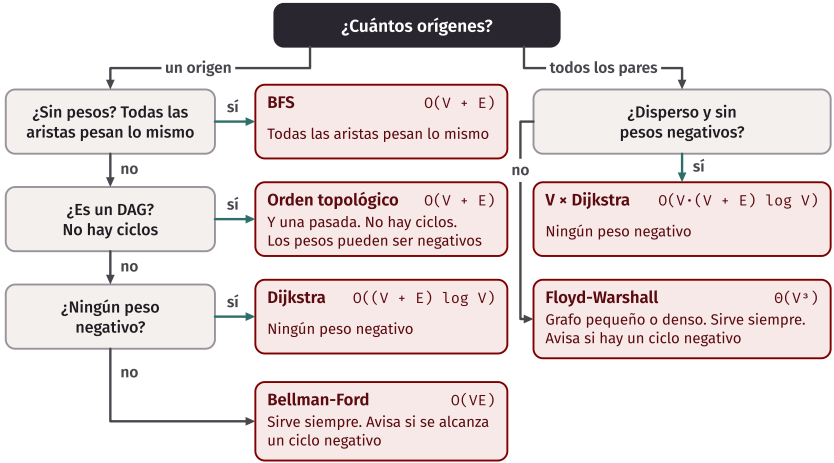
- Completo: gana Floyd-Warshall. Disperso de 400: es el más lento.
- $V\times$ Bellman-Ford tiene parada temprana. Contándolas aparte, en grafos generados igual con $V = 400$: 8 vueltas en promedio, 11 como máximo. Esas filas no son su peor caso.
- La cadena al revés sí lo acerca: duplicar V multiplicó su tiempo por 6 a 8.

SECCIÓN 07

Cuadro de decisión

07

Cuadro de decisión



Un DAG con negativos no necesita Bellman-Ford

El mismo grafo que rompió a Dijkstra es un DAG:

```
orden topológico: 0 -1 4 1
Bellman-Ford:     0 -1 4 1
Dijkstra:         0 -1 4 3
```

Un DAG con negativos no necesita Bellman-Ford

El mismo grafo que rompió a Dijkstra es un DAG:

```
orden topológico: 0 -1 4 1
Bellman-Ford:     0 -1 4 1
Dijkstra:         0 -1 4 3
```

Con ciclos negativos, ninguno da “el camino simple más corto”: es el camino más largo con los pesos cambiados de signo. Semana 14.

SECCIÓN 08

Máscaras de bits

08

[se puede cortar] Un subconjunto como un entero

Máscara de bits *(bitmask)*: el bit v vale 1 si v está en el subconjunto. $\{0, 2, 3\}$ es $1101_2 = 13$.

¿Está v?	`(mask & (1 << v)) != 0`
Agregar	`mask | (1 << v)`
Quitar	`mask & ~(1 << v)`
Todos	`(1 << n) - 1`

[se puede cortar] Un subconjunto como un entero

Máscara de bits *(bitmask)*: el bit v vale 1 si v está en el subconjunto. $\{0, 2, 3\}$ es $1101_2 = 13$.

¿Está v?	`(mask & (1 << v)) != 0`
Agregar	`mask | (1 << v)`
Quitar	`mask & ~(1 << v)`
Todos	`(1 << n) - 1`

Quitar un bit da un entero menor. Recorrer de 0 a $2^n - 1$ visita cada subconjunto después de los suyos.

[se puede cortar] Un subconjunto como un entero

Máscara de bits *(bitmask)*: el bit v vale 1 si v está en el subconjunto. $\{0, 2, 3\}$ es $1101_2 = 13$.

¿Está v?	`(mask & (1 << v)) != 0`
Agregar	`mask | (1 << v)`
Quitar	`mask & ~(1 << v)`
Todos	`(1 << n) - 1`

Quitar un bit da un entero menor. Recorrer de 0 a $2^n - 1$ visita cada subconjunto después de los suyos.

El orden de cálculo sale gratis.

[se puede cortar] Un subconjunto como un entero

Máscara de bits *(bitmask)*: el bit v vale 1 si v está en el subconjunto. $\{0, 2, 3\}$ es $1101_2 = 13$.

¿Está v?	`(mask & (1 << v)) != 0`
Agregar	`mask | (1 << v)`
Quitar	`mask & ~(1 << v)`
Todos	`(1 << n) - 1`

Quitar un bit da un entero menor. Recorrer de 0 a $2^n - 1$ visita cada subconjunto después de los suyos.

El orden de cálculo sale gratis.

El viajero *(traveling salesperson problem)*: salir de 0, pasar una vez por cada ciudad, volver. Con 20 ciudades, $19! \approx 1{,}2 \times 10^{17}$ recorridos.

[se puede cortar] Held-Karp

¿Qué necesita saber el futuro? Solo **cuáles se visitaron** y **dónde estamos**.

Subproblema

$C(S, v)$: el costo mínimo de un camino que sale de 0, visita exactamente los nodos de S, una vez cada uno, y termina en $v \in S$.

$$C(\{v\}, v) = c(0, v) \qquad C(S, v) = \min_{u \in S,\, u \neq v} C(S - \{v\}, u) + c(u, v)$$

[se puede cortar] Held-Karp

¿Qué necesita saber el futuro? Solo **cuáles se visitaron** y **dónde estamos**.

Subproblema

$C(S, v)$: el costo mínimo de un camino que sale de 0, visita exactamente los nodos de S, una vez cada uno, y termina en $v \in S$.

$$C(\{v\}, v) = c(0, v) \qquad C(S, v) = \min_{u \in S,\, u \neq v} C(S - \{v\}, u) + c(u, v)$$

- **Demostración:** inducción sobre $|S|$, mirando el penúltimo nodo. Como $v \notin S - \{v\}$, agregar (u, v) no repite ningún nodo.

[se puede cortar] Held-Karp

¿Qué necesita saber el futuro? Solo **cuáles se visitaron** y **dónde estamos**.

Subproblema

$C(S, v)$: el costo mínimo de un camino que sale de 0, visita exactamente los nodos de S, una vez cada uno, y termina en $v \in S$.

$$C(\{v\}, v) = c(0, v) \qquad C(S, v) = \min_{u \in S,\, u \neq v} C(S - \{v\}, u) + c(u, v)$$

- **Demostración:** inducción sobre $|S|$, mirando el penúltimo nodo. Como $v \notin S - \{v\}$, agregar (u, v) no repite ningún nodo.
- **Respuesta:** $\min_v C(\text{todos}, v) + c(v, 0)$.

[se puede cortar] Held-Karp

¿Qué necesita saber el futuro? Solo **cuáles se visitaron** y **dónde estamos**.

Subproblema

$C(S, v)$: el costo mínimo de un camino que sale de 0, visita exactamente los nodos de S, una vez cada uno, y termina en $v \in S$.

$$C(\{v\}, v) = c(0, v) \qquad C(S, v) = \min_{u \in S,\, u \neq v} C(S - \{v\}, u) + c(u, v)$$

- **Demostración:** inducción sobre $|S|$, mirando el penúltimo nodo. Como $v \notin S - \{v\}$, agregar (u, v) no repite ningún nodo.
- **Respuesta:** $\min_v C(\text{todos}, v) + c(v, 0)$.
- **Costo:** tiempo $O(n^2 2^n)$, memoria $O(n 2^n)$. Exponencial, pero mucho mejor que $n!$.

[se puede cortar] La tabla con 4 ciudades

$C(\{1,2\},2) = 10 + 9 = 19.$

terminal

```
máscara  subconjunto  termina en 1  en 2  en 3
      1          {1}            10     -     -
      2          {2}             -    15     -
      3        {1,2}            28    19     -
      4          {3}             -     -    20
      5        {1,3}            28     -    20
      6        {2,3}             -    29    27
      7      {1,2,3}            35    29    31

mejor recorrido, Held-Karp: 35
mejor recorrido, permutaciones: 35
```

[se puede cortar] La tabla con 4 ciudades

terminal

máscara	subconjunto	termina en 1	en 2	en 3
1	{1}	10	-	-
2	{2}	-	15	-
3	{1,2}	28	19	-
4	{3}	-	-	20
5	{1,3}	28	-	20
6	{2,3}	-	29	27
7	{1,2,3}	35	29	31

```
mejor recorrido, Held-Karp: 35
mejor recorrido, permutaciones: 35
```

$C(\{1,2\},2) = 10 + 9 = 19.$

$C(\{1,2,3\},2) = \min(28+9,\ 20+9) = 29.$

[se puede cortar] La tabla con 4 ciudades

terminal

máscara	subconjunto	termina en 1	en 2	en 3
1	{1}	10	-	-
2	{2}	-	15	-
3	{1,2}	28	19	-
4	{3}	-	-	20
5	{1,3}	28	-	20
6	{2,3}	-	29	27
7	{1,2,3}	35	29	31

```
mejor recorrido, Held-Karp: 35
mejor recorrido, permutaciones: 35
```

$C(\{1,2\},2) = 10 + 9 = 19.$

$C(\{1,2,3\},2) = \min(28+9,\ 20+9) = 29.$

La respuesta:
$\min(35+5,\ 29+6,\ 31+8) = 35,$
con $0 \to 1 \to 3 \to 2 \to 0.$

[se puede cortar] Permutaciones frente a Held-Karp

terminal

```
$ build/held_karp_bench
  n  permutaciones      Held-Karp memoria de tabla
 10        0.004 s        0.000 s           0.0 MB  iguales
 11        0.036 s        0.000 s           0.1 MB  iguales
 12        0.358 s        0.001 s           0.2 MB  iguales
 14              -        0.003 s           0.9 MB
 16              -        0.013 s           4.2 MB
 18              -        0.067 s          18.9 MB
 20              -        0.294 s          83.9 MB
```

- Una corrida con `-O2`, un grafo por fila. Los tiempos dependen de la máquina.
- La memoria no está medida: es la cuenta $n \cdot 2^{n-1} \cdot 8$ bytes.

[se puede cortar] Permutaciones frente a Held-Karp

terminal

```
$ build/held_karp_bench
  n  permutaciones      Held-Karp memoria de tabla
 10        0.004 s        0.000 s           0.0 MB  iguales
 11        0.036 s        0.000 s           0.1 MB  iguales
 12        0.358 s        0.001 s           0.2 MB  iguales
 14              -        0.003 s           0.9 MB
 16              -        0.013 s           4.2 MB
 18              -        0.067 s          18.9 MB
 20              -        0.294 s          83.9 MB
```

- Una corrida con `-O2`, un grafo por fila. Los tiempos dependen de la máquina.
- La memoria no está medida: es la cuenta $n \cdot 2^{n-1} \cdot 8$ bytes.
- Una extrapolación desde $n = 12$, no una medición: $19!$ recorridos tomarían más de 30 años.

SECCIÓN 09

Arbitraje

09

Del producto a la suma

Un **arbitraje**: una cadena de cambios que vuelve a la moneda inicial con más dinero.

Afirmación

Con peso $w = -\ln r$ en cada arista, las tasas de un ciclo dan un producto mayor que 1 si y solo si sus pesos dan una suma negativa.

Del producto a la suma

Un **arbitraje**: una cadena de cambios que vuelve a la moneda inicial con más dinero.

Afirmación

Con peso $w = -\ln r$ en cada arista, las tasas de un ciclo dan un producto mayor que 1 si y solo si sus pesos dan una suma negativa.

$$r_1 \cdots r_L > 1 \iff \ln r_1 + \cdots + \ln r_L > 0 \iff -\ln r_1 - \cdots - \ln r_L < 0$$

Del producto a la suma

Un **arbitraje**: una cadena de cambios que vuelve a la moneda inicial con más dinero.

Afirmación

Con peso $w = -\ln r$ en cada arista, las tasas de un ciclo dan un producto mayor que 1 si y solo si sus pesos dan una suma negativa.

$$r_1 \cdots r_L > 1 \iff \ln r_1 + \cdots + \ln r_L > 0 \iff -\ln r_1 - \cdots - \ln r_L < 0$$

Un arbitraje es un ciclo negativo.

Del producto a la suma

Un **arbitraje**: una cadena de cambios que vuelve a la moneda inicial con más dinero.

Afirmación

Con peso $w = -\ln r$ en cada arista, las tasas de un ciclo dan un producto mayor que 1 si y solo si sus pesos dan una suma negativa.

$$r_1 \cdots r_L > 1 \iff \ln r_1 + \cdots + \ln r_L > 0 \iff -\ln r_1 - \cdots - \ln r_L < 0$$

Un arbitraje es un ciclo negativo. Y en cualquier parte de la tabla: todos en 0.

El redondeo: productos de 1 con suma negativa

terminal

```
$ build/rates_demo
Producto real y suma de -ln(tasa) en double:
tasas                    producto         suma
2 y 0,5                  1 exacto    0.000e+00
1,25 y 0,8               1 exacto   -5.551e-17
1,6 y 0,625              1 exacto   -5.551e-17
2,5 y 0,4                1 exacto   -1.110e-16
1,6, 1,25 y 0,5          1 exacto   -1.110e-16
1,25, 1,25 y 0,64        1 exacto   -5.551e-17
1,1, 1,1 y 1/1,21 (*)   menor a 1    4.996e-16
(*) 0,826446280991735 es 1/1,21 truncado a 15 decimales.
```

El redondeo: productos de 1 con suma negativa

terminal

```
$ build/rates_demo
Producto real y suma de -ln(tasa) en double:
tasas                    producto         suma
2 y 0,5                  1 exacto    0.000e+00
1,25 y 0,8               1 exacto   -5.551e-17
1,6 y 0,625              1 exacto   -5.551e-17
2,5 y 0,4                1 exacto   -1.110e-16
1,6, 1,25 y 0,5          1 exacto   -1.110e-16
1,25, 1,25 y 0,64        1 exacto   -5.551e-17
1,1, 1,1 y 1/1,21 (*)   menor a 1    4.996e-16
(*) 0,826446280991735 es 1/1,21 truncado a 15 decimales.
```

- Con 1,25 y 0,8 no hay arbitraje, y **la suma sale negativa**.

El redondeo: productos de 1 con suma negativa

terminal

```
$ build/rates_demo
Producto real y suma de -ln(tasa) en double:
tasas                    producto         suma
2 y 0,5                  1 exacto    0.000e+00
1,25 y 0,8               1 exacto   -5.551e-17
1,6 y 0,625              1 exacto   -5.551e-17
2,5 y 0,4                1 exacto   -1.110e-16
1,6, 1,25 y 0,5          1 exacto   -1.110e-16
1,25, 1,25 y 0,64        1 exacto   -5.551e-17
1,1, 1,1 y 1/1,21 (*)   menor a 1    4.996e-16
(*) 0,826446280991735 es 1/1,21 truncado a 15 decimales.
```

- Con 1,25 y 0,8 no hay arbitraje, y **la suma sale negativa**.
- Aceptar cualquier ganancia, por pequeña que sea, es indistinguible del redondeo: hace falta un margen.

La promesa y la tolerancia

Promesa: el producto de todo ciclo simple es a lo sumo 1, o por lo menos $1 + g$.

La promesa y la tolerancia

Promesa: el producto de todo ciclo simple es a lo sumo 1, o por lo menos $1 + g$.

Tolerancia: sumar el mismo $\varepsilon > 0$ al peso de **cada arista**. Es otro grafo: las demostraciones valen tal cual.

La promesa y la tolerancia

Promesa: el producto de todo ciclo simple es a lo sumo 1, o por lo menos $1 + g$.

Tolerancia: sumar el mismo $\varepsilon > 0$ al peso de **cada arista**. Es otro grafo: las demostraciones valen tal cual.

Teorema (aritmética exacta)

Si $L\varepsilon < \ln(1 + g)$ para todo largo posible L, un ciclo es negativo con los pesos nuevos si y solo si tiene arbitraje.

La promesa y la tolerancia

Promesa: el producto de todo ciclo simple es a lo sumo 1, o por lo menos $1 + g$.

Tolerancia: sumar el mismo $\varepsilon > 0$ al peso de **cada arista**. Es otro grafo: las demostraciones valen tal cual.

Teorema (aritmética exacta)

Si $L\varepsilon < \ln(1 + g)$ para todo largo posible L, un ciclo es negativo con los pesos nuevos si y solo si tiene arbitraje.

- Si $P \geq 1 + g$: peso nuevo $= -\ln P + L\varepsilon \leq -\ln(1 + g) + L\varepsilon < 0$.

La promesa y la tolerancia

Promesa: el producto de todo ciclo simple es a lo sumo 1, o por lo menos $1 + g$.

Tolerancia: sumar el mismo $\varepsilon > 0$ al peso de **cada arista**. Es otro grafo: las demostraciones valen tal cual.

Teorema (aritmética exacta)

Si $L\varepsilon < \ln(1 + g)$ para todo largo posible L, un ciclo es negativo con los pesos nuevos si y solo si tiene arbitraje.

- Si $P \geq 1 + g$: peso nuevo $= -\ln P + L\varepsilon \leq -\ln(1 + g) + L\varepsilon < 0$.
- Si $P \leq 1$: peso nuevo $\geq L\varepsilon > 0$.

La promesa y la tolerancia

Promesa: el producto de todo ciclo simple es a lo sumo 1, o por lo menos $1 + g$.

Tolerancia: sumar el mismo $\varepsilon > 0$ al peso de **cada arista**. Es otro grafo: las demostraciones valen tal cual.

Teorema (aritmética exacta)

Si $L\varepsilon < \ln(1 + g)$ para todo largo posible L, un ciclo es negativo con los pesos nuevos si y solo si tiene arbitraje.

- Si $P \geq 1 + g$: peso nuevo $= -\ln P + L\varepsilon \leq -\ln(1 + g) + L\varepsilon < 0$.
- Si $P \leq 1$: peso nuevo $\geq L\varepsilon > 0$.
- Un ciclo recuperado es de verdad: peso nuevo negativo implica $P > 1$, y por la promesa, $P \geq 1 + g$.

La ventana de ε

$g = 10^{-6}$, $L \leq 400$, tasas entre 10^{-6} y 10^{6}.

terminal

```
epsilon tabla exacta (sin arbitraje)
     0e+00         sí (falso arbitraje)
     1e-15                           no
     1e-12                           no
     1e-09                           no
     1e-08                           no
     1e-06                           no
   epsilon       ciclo de 400, 1,000001
     0e+00                           sí
     1e-15                           sí
     1e-12                           sí
     1e-09                           sí
     1e-08           no (se le escapa)
     1e-06           no (se le escapa)
```

Error común: lo que debe quedar bajo la ganancia es $L\varepsilon$, no ε.

La ventana de ε

$g = 10^{-6}$, $L \leq 400$, tasas entre 10^{-6} y 10^{6}.

Por arriba: $\varepsilon < 10^{-6}/400 = 2{,}5 \times 10^{-9}$.

terminal

```
epsilon tabla exacta (sin arbitraje)
     0e+00         sí (falso arbitraje)
     1e-15                           no
     1e-12                           no
     1e-09                           no
     1e-08                           no
     1e-06                           no
   epsilon       ciclo de 400, 1,000001
     0e+00                           sí
     1e-15                           sí
     1e-12                           sí
     1e-09                           sí
     1e-08           no (se le escapa)
     1e-06           no (se le escapa)
```

Error común: lo que debe quedar bajo la ganancia es $L\varepsilon$, no ε.

La ventana de ε

$g = 10^{-6}$, $L \leq 400$, tasas entre 10^{-6} y 10^{6}.

Por arriba: $\varepsilon < 10^{-6}/400 = 2{,}5 \times 10^{-9}$.

Por abajo: sin arbitraje, $|d| < 400 \cdot \ln 10^{6} < 5526$. Una suma en `double` se equivoca en a lo sumo 2^{-53} de su resultado: unos 6×10^{-13}.

terminal

```
epsilon tabla exacta (sin arbitraje)
     0e+00         sí (falso arbitraje)
     1e-15                           no
     1e-12                           no
     1e-09                           no
     1e-08                           no
     1e-06                           no
   epsilon       ciclo de 400, 1,000001
     0e+00                           sí
     1e-15                           sí
     1e-12                           sí
     1e-09                           sí
     1e-08           no (se le escapa)
     1e-06           no (se le escapa)
```

Error común: lo que debe quedar bajo la ganancia es $L\varepsilon$, no ε.

La ventana de ε

$g = 10^{-6}$, $L \leq 400$, tasas entre 10^{-6} y 10^{6}.

Por arriba: $\varepsilon < 10^{-6}/400 = 2{,}5 \times 10^{-9}$.

Por abajo: sin arbitraje, $|d| < 400 \cdot \ln 10^{6} < 5526$. Una suma en `double` se equivoca en a lo sumo 2^{-53} de su resultado: unos 6×10^{-13}.

$\varepsilon = 10^{-9}$ queda lejos de las dos.

terminal

```
epsilon tabla exacta (sin arbitraje)
     0e+00         sí (falso arbitraje)
     1e-15                           no
     1e-12                           no
     1e-09                           no
     1e-08                           no
     1e-06                           no
   epsilon       ciclo de 400, 1,000001
     0e+00                           sí
     1e-15                           sí
     1e-12                           sí
     1e-09                           sí
     1e-08           no (se le escapa)
     1e-06           no (se le escapa)
```

Error común: lo que debe quedar bajo la ganancia es $L\varepsilon$, no ε.

La ventana de ε

$g = 10^{-6}$, $L \leq 400$, tasas entre 10^{-6} y 10^{6}.

Por arriba: $\varepsilon < 10^{-6}/400 = 2{,}5 \times 10^{-9}$.

Por abajo: sin arbitraje, $|d| < 400 \cdot \ln 10^{6} < 5526$. Una suma en `double` se equivoca en a lo sumo 2^{-53} de su resultado: unos 6×10^{-13}.

$\varepsilon = 10^{-9}$ queda lejos de las dos.

Con `double`: el teorema de recuperar el ciclo es en aritmética exacta. Revisen que no aparezca `NULO` al retroceder.

terminal

```
epsilon tabla exacta (sin arbitraje)
     0e+00         sí (falso arbitraje)
     1e-15                          no
     1e-12                          no
     1e-09                          no
     1e-08                          no
     1e-06                          no
   epsilon       ciclo de 400, 1,000001
     0e+00                          sí
     1e-15                          sí
     1e-12                          sí
     1e-09                          sí
     1e-08           no (se le escapa)
     1e-06           no (se le escapa)
```

Error común: lo que debe quedar bajo la ganancia es $L\varepsilon$, no ε.

Lo que hay que llevarse

- **Bellman-Ford:** $d_i(v)$ con a lo sumo i aristas. $V - 1$ bastan si no se alcanza un ciclo negativo. $O(VE)$.
- **Una fila:** $d[v]$ no es $d_i(v)$. Es el largo de un camino, solo baja, y queda bajo $d_i(v)$.
- **Detección:** la vuelta V relaja algo si y solo si se alcanza un ciclo negativo.
- **Recuperación:** retroceder V veces, y recién leer.
- **Infinito:** de un nodo no alcanzado no se propaga nada.
- **Floyd-Warshall:** d_k con intermedios entre los k primeros. $\Theta(V^3)$. La fila y la columna k no cambian en la vuelta k.
- k **por fuera:** es el parámetro del subproblema.
- **Diagonal:** $d[k][k] < 0$ al empezar la vuelta de k, y detenerse.
- **Elegir:** BFS, DAG, Dijkstra, Bellman-Ford, Floyd-Warshall, cada uno con su condición.
- **Arbitraje:** producto > 1 es suma de $-\ln r$ negativa. ε en cada peso, con $L\varepsilon$ bajo la ganancia y ε sobre el redondeo.

Para la próxima clase

- Leer los apuntes de la semana 10 y hacer sus ejercicios.
- **Laboratorio:** arbitraje con tipos de cambio, en `lab/arbitrage`.

Preguntas

Corran Bellman-Ford a mano sobre un grafo de 4 nodos con una arista negativa.
¿Por qué un camino mínimo sin ciclos negativos tiene a lo sumo $V - 1$ aristas?
¿Qué algoritmo usarían: grafo sin pesos; DAG con pesos negativos; red de carreteras; todos los pares en un grafo pequeño y denso?
Escriban el subproblema de Floyd-Warshall. ¿Por qué el orden de los ciclos importa?
¿Cómo se detecta un ciclo negativo con la matriz de Floyd-Warshall?