

# Algoritmo de Bellman-Ford

Clase 23

IIC 2133 - Sección 1

Prof. Sebastián Buggedo

# Sumario

Obertura

Algoritmo de Bellman-Ford

Epílogo

# Rutas en viajes 2.0

Consideremos el problema de planificar un viaje en auto desde  $A$  a  $B$

- Modelo de **grafo dirigido con costos**
- Ya sabemos resolver este problema...
- ... siempre que los costos sean **no negativos**

Algoritmo de Dijkstra resuelve el problema que ya conocemos

## ¿Por qué Dijkstra no sirve con costos negativos?

Dijkstra( $s$ ):

**for**  $u \in V - \{s\}$  :

$u.color \leftarrow \text{blanco}; d[u] \leftarrow \infty; \pi[u] \leftarrow \emptyset$

$s.color \leftarrow \text{gris}; d[s] \leftarrow 0; \pi[s] \leftarrow \emptyset$

$Q \leftarrow$  cola de **prioridades** vacía

Insert( $Q, s$ )

**while**  $Q$  no está vacía :

$u \leftarrow \text{Extract}(Q)$

**for**  $v \in \alpha[u]$  :

**if**  $v.color = \text{blanco} \vee v.color = \text{gris}$  :

**if**  $d[v] > d[u] + \text{cost}(u, v)$  :

$d[v] \leftarrow d[u] + \text{cost}(u, v); \pi[v] \leftarrow u$

**if**  $v.color = \text{blanco}$  :

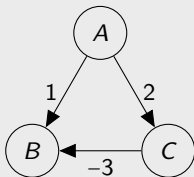
$v.color \leftarrow \text{gris}; Q.enqueue(v)$

$u.color \leftarrow \text{negro}$

# Dijkstra con costos negativos RIP

## Ejercicio

Tomando el siguiente grafo dirigido con costos negativos, compruebe que Dijkstra no entrega la ruta con el menor costo desde  $A$  hasta  $B$ . Proponga una explicación a este problema.



# Dijkstra con costos negativos RIP

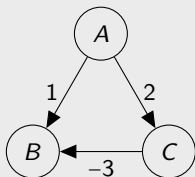
## Ejercicio

Llevaremos en una tabla el estado de  $d[X]$ ,  $\pi[X]$  y  $Q$  al **inicio** de la iteración  $t$  del loop. Recordemos que

$d[X] :=$  menor costo acumulado hasta  $X$

$\pi[X] :=$  ancestro de  $X$  en ese camino

Al inicio de la primera iteración, tenemos los valores iniciales y el nodo de partida como **más prioritario** en  $Q$  (además, es gris)



| $t$ | $(d[X], \pi[X])$ |                       |                       | $Q$                                 |                                    |
|-----|------------------|-----------------------|-----------------------|-------------------------------------|------------------------------------|
|     | $A$              | $B$                   | $C$                   |                                     |                                    |
| 1   | $(0, \emptyset)$ | $(\infty, \emptyset)$ | $(\infty, \emptyset)$ | <div><div>A</div><div>0</div></div> | <div><div></div><div>1</div></div> |

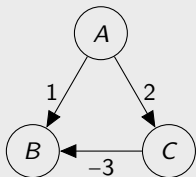
# Dijkstra con costos negativos RIP

## Ejercicio

Al revisar los vecinos de  $A$ , actualizamos sus costos y ancestro, pues tenemos un camino **de largo 1** que es mejor que el costo  $\infty$

Además, al descubrir  $B$  y  $C$ , se pintan grises y se agregan a la cola **respetando prioridad** dada por  $d[X]$

Como revisamos todas las conexiones desde  $A$ , lo terminamos

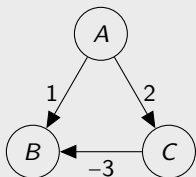


| $t$ | $(d[X], \pi[X])$ |                       |                       | $Q$                                 |                                     |
|-----|------------------|-----------------------|-----------------------|-------------------------------------|-------------------------------------|
|     | $A$              | $B$                   | $C$                   |                                     |                                     |
| 1   | $(0, \emptyset)$ | $(\infty, \emptyset)$ | $(\infty, \emptyset)$ | <div><div>A</div><div>0</div></div> | <div><div></div><div>1</div></div>  |
| 2   | $(0, \emptyset)$ | $(1, A)$              | $(2, A)$              | <div><div>B</div><div>0</div></div> | <div><div>C</div><div>1</div></div> |

# Dijkstra con costos negativos RIP

## Ejercicio

Por prioridad,  $Q$  entrega a  $B$  como siguiente elemento a revisar. Dado que no tiene vecinos, lo terminamos



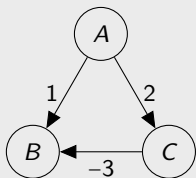
| $t$ | $(d[X], \pi[X])$ |          |          | $Q$                             |                                 |
|-----|------------------|----------|----------|---------------------------------|---------------------------------|
|     | $A$              | $B$      | $C$      |                                 |                                 |
| 2   | $(0, \emptyset)$ | $(1, A)$ | $(2, A)$ | <div><math>B</math><br/>0</div> | <div><math>C</math><br/>1</div> |
| 3   | $(0, \emptyset)$ | $(1, A)$ | $(2, A)$ | <div><math>C</math><br/>0</div> | <div></div>                     |



# Dijkstra con costos negativos RIP

## Ejercicio

Ahora extraemos de  $Q$  a  $C$ , y como su único vecino ya está terminado, terminamos a  $C$

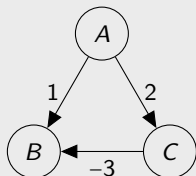


| $t$ | $(d[X], \pi[X])$ |          |          | $Q$                       |                          |
|-----|------------------|----------|----------|---------------------------|--------------------------|
|     | $A$              | $B$      | $C$      |                           |                          |
| 3   | $(0, \emptyset)$ | $(1, A)$ | $(2, A)$ | <div>C</div> <div>0</div> | <div></div> <div>1</div> |
| 4   | $(0, \emptyset)$ | $(1, A)$ | $(2, A)$ | <div></div> <div>0</div>  | <div></div> <div>1</div> |

# Dijkstra con costos negativos RIP

## Ejercicio

El estado final de los parámetros nos entrega la ruta encontrada por Dijkstra



| $t$ | $(d[X], \pi[X])$ |          |          | $Q$                                |                                    |
|-----|------------------|----------|----------|------------------------------------|------------------------------------|
|     | $A$              | $B$      | $C$      |                                    |                                    |
| 4   | $(0, \emptyset)$ | $(1, A)$ | $(2, A)$ | <div><div></div><div>0</div></div> | <div><div></div><div>1</div></div> |

La ruta de  $A$  hasta  $B$  entregada cumple

$$\text{cost}((A, B)) = 1 > -1 = \text{cost}((A, C, B))$$

de manera que Dijkstra no entrega la ruta con el costo mínimo en este ejemplo

# El problema de los costos negativos

No olvidemos que Dijkstra se basa en un recorrido BFS del grafo

- Asume que si ya llegamos en  $k$  pasos a un nodo  $u$  con costo  $d[u]$ ...
- ...no podemos llegar con un costo menor en  $k + 1$  pasos
- Esto simplifica el análisis y permite no revisar aristas cuando ya hemos terminado un nodo
- Recordar: si el nodo está terminado/negro, ya no cambia su  $d[u]$

Esto funciona cuando los costos son no negativos, pues una arista  $k + 1$  mantiene o incrementa el costo ya alcanzado en  $k$  pasos

Los costos negativos obligan a revisar caminos de distintos largos antes de dar por terminado un nodo

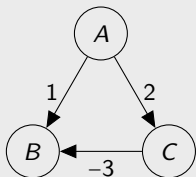
# El problema de los costos negativos

Ahora bien, no todo está perdido...

## Ejemplo

Para el grafo dado, si **forzamos** el largo de los caminos, esperamos obtener diferentes ancestros óptimos

Para ser más precisos, nos interesa el costo mínimo al usar **a lo más  $k$**  aristas



| $k$ | $(d[X], \pi[X])$ |           |           |
|-----|------------------|-----------|-----------|
|     | $A$              | $B$       | $C$       |
| 0   | $(0, \emptyset)$ | no aplica | no aplica |
| 1   | $(0, \emptyset)$ | $(1, A)$  | $(2, A)$  |
| 2   | $(0, \emptyset)$ | $(-1, C)$ | $(2, A)$  |

Notemos que para  $C$ , la ruta  $(A, C)$  es óptima con a lo más 1 y a lo más 2 aristas

Buscamos un algoritmo que permita resolver esta variante

# Playlist y Orquestas



Playlist: DatiWawos Tercer Acto

Además sigan en instagram:

@orquesta.ciudadana.santiago @orquesta\_tamen

# Sumario

Obertura

Algoritmo de Bellman-Ford

Epílogo

# Una idea de algoritmo

Para encontrar las rutas **más baratas** desde **una sola fuente** seguiremos la siguiente estrategia

- Buscamos rutas más cortas desde  $s$  con a lo más 1 arista
- Luego, con a lo más 2 aristas
- Luego, con a lo más 3 aristas
- ...

Esta estrategia aprovecha un enfoque de programación dinámica

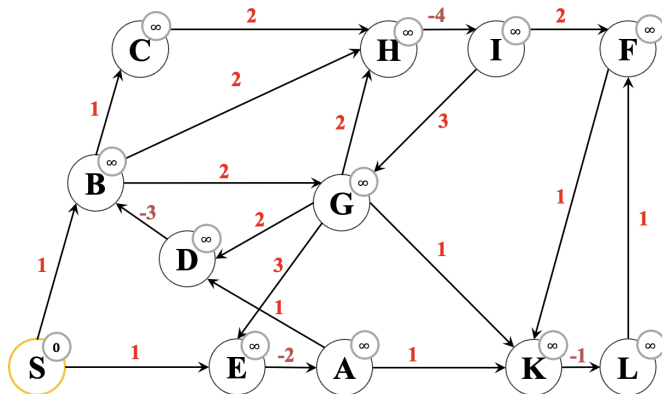
1. Buscamos las rutas más cortas con a lo más  $k$  aristas
2. Podemos utilizar las rutas más cortas con a lo más  $k - 1$  aristas

¿Hasta qué largo de  $k$  es necesario iterar?

# Una idea de algoritmo

Iniciamos los nodos con  $d[u] = \infty$  salvo la fuente  $S$

Marcaremos con **verde** los caminos más cortos asegurados **hasta el momento**

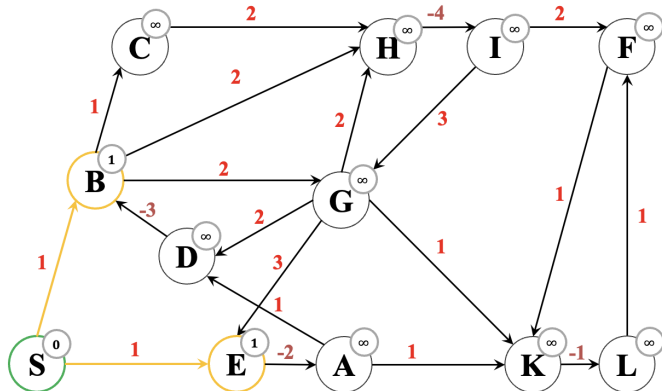




# Una idea de algoritmo

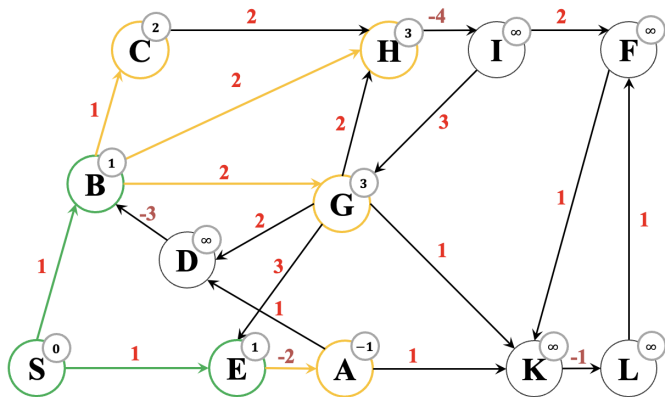
Logramos llegar a *B* y *E* con un costo acumulado menor al previo

Notemos que con caminos de largo  $\leq 1$ , llegamos directo desde *S*



# Una idea de algoritmo

Al extender a caminos de largo  $\leq 2$ , estamos usando las rutas ya calculadas

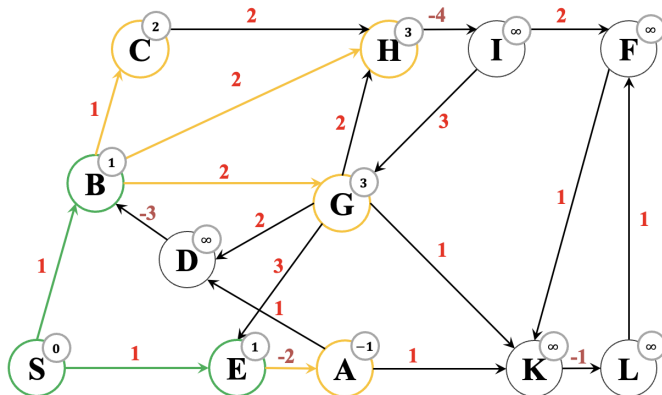


E.g. para  $G$ , como el ancestro óptimo es  $B$ , usamos la ruta más corta de largo  $\leq 1$  con la que se llega a  $B$

# Una idea de algoritmo

Ojo: ahora extenderemos el rango a caminos de largo  $\leq 3$

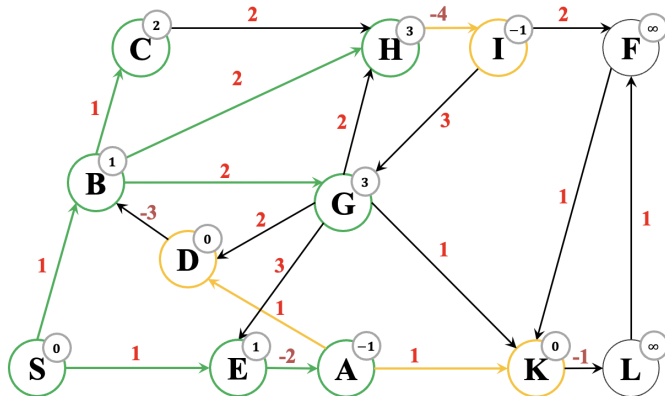
Llegaremos a nodos ya visitados a través de un camino alternativo



¿Qué hacemos en ese caso? ¿Los ignoramos como en Dijkstra?

# Una idea de algoritmo

En este caso, el camino  $(S, B, C, H)$  tiene costo mayor a  $(S, B, H)$ , de manera que no incluimos la arista  $(C, H)$  en dicho camino



# Subestructura óptima en rutas

## Definición

Dado un grafo dirigido  $G$  y nodos  $x, y \in V(G)$ , diremos que  $p : v_0, v_1, \dots, v_n$  es un camino dirigido de  $x$  hasta  $y$  si

- $(v_i, v_{i+1}) \in E(G)$  para cada  $0 \leq i \leq n-1$
- $v_0 = x$  y  $v_n = y$

También denotaremos a  $p$  como  $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_n$ . Si existe un camino dirigido  $p$  de  $x$  hasta  $y$  lo denotaremos por  $x \rightsquigarrow_p y$ . Además, el largo del camino dirigido  $p$  es  $|p| = n$

## Proposición

Si el camino dirigido  $u \rightsquigarrow_p x \rightarrow y$  es una ruta más corta de  $u$  a  $y$ , entonces  $p$  es una ruta más corta de  $u$  a  $x$

Un camino óptimo  $|p| \leq k$  no necesariamente contiene uno de largo  $|p'| = k-1$ . Siempre contiene uno de largo  $|p'| \leq k-1$

# Subestructura óptima en rutas

En nuestro algoritmo, consideremos la  $k$ -ésima iteración

- Estamos analizando las rutas más cortas con largo  $|p| \leq k$
- Sean  $s \rightsquigarrow u$  y  $s \rightsquigarrow v$  las rutas más cortas con  $|p| \leq k - 1$
- Para cada arista  $(u, v) \in E$  hay que comparar dos rutas
  - $s \rightsquigarrow u \rightarrow v$
  - $s \rightsquigarrow v$  (la misma ruta ya conocida, sin cambios)
- Tal comparación determina si la arista  $(u, v)$  se incluye en la ruta óptima  $s \rightsquigarrow v$  para obtener una de largo  $\leq k$

Notemos que esta comparación es esencialmente la misma que en Dijkstra

¿Cuál es el rango de  $k$  que debemos verificar?

# Largos posibles para rutas óptimas

Nos interesa conocer el largo máximo de una ruta más corta  $p$  desde  $s$  a cualquier nodo de  $G$

Para esto, definimos una herramienta útil

## Definición

Sea  $G$  un grafo no dirigido con costos. Un ciclo  $p: v_0, v_1, \dots, v_n$  de  $G$  es un **ciclo con costo negativo** si

$$\text{cost}(p) = \sum_{i=0}^{n-1} \text{cost}(v_i, v_{i+1}) < 0$$

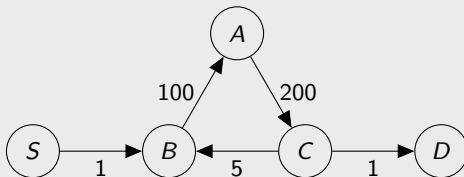
¿Sabemos algo sobre ciclos en rutas más cortas?

En especial, ¿sabemos algo sobre ciclos con costos negativos?

# Largos posibles para rutas óptimas

## Ejemplo

En el siguiente grafo, el ciclo  $B, A, C, B$  es de costo no negativo



Es claro que este ciclo no hace parte de ningún camino más corto, por muy barata que sea la arista  $(C, B)$

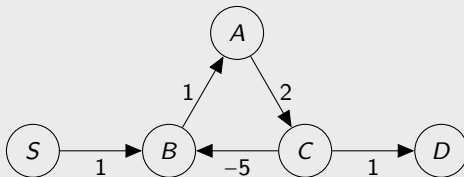
Se puede demostrar que toda ruta más corta  
**no tiene** ciclos de costo no negativo



# Largos posibles para rutas óptimas

## Ejemplo

En el siguiente grafo, el ciclo  $B, A, C, B$  es de costo negativo



¿Cuál es el camino dirigido más barato de  $S$  hasta  $D$ ?

- $cost(S, B, A, C, D) = 5$
- $cost(S, B, A, C, B, A, C, D) = 3$
- $cost(S, B, A, C, B, A, C, B, A, C, D) = 1$

Si existe un ciclo de costo negativo **alcanzable desde  $S$** ,  
no hay ruta más corta desde  $S$  hasta todo nodo

# Largos posibles para rutas óptimas

Dado un camino más corto  $p$  de  $s$  a  $u$

- Sabemos que  $p$  no tiene ciclos con costo negativo alcanzables desde  $s$  (impiden la existencia de rutas más cortas)
- $p$  tampoco tiene ciclos de costo no negativo

Luego, la ruta más corta más larga  $p$  no tiene ciclos

- Sabemos que no repite vértices ( $p$  es simple)
- La ruta más larga posible tendría  $|p| = |V|$

Los posibles valores de  $k$  están en el rango  $1 \dots |V| - 1$

# Algoritmo de Bellman-Ford

BellmanFord( $s$ ):

```
1  for  $u \in V$  :  
2       $d[u] \leftarrow \infty$ ;  $\pi[u] \leftarrow \emptyset$   
3   $d[s] \leftarrow 0$   
4  for  $k = 1 \dots |V| - 1$  :  
5      for  $(u, v) \in E$  :  
6          if  $d[v] > d[u] + \text{cost}(u, v)$  :  
7               $d[v] \leftarrow d[u] + \text{cost}(u, v)$   
8               $\pi[v] \leftarrow u$ 
```

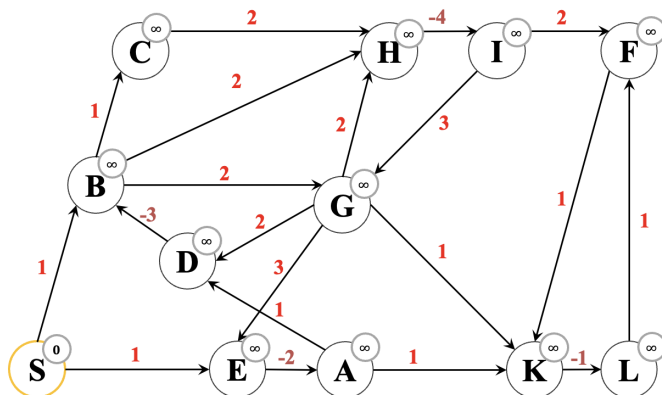
Al revisar  $|V| - 1$  veces **todas** las aristas, nos aseguramos de actualizar rutas en caso de encontrar atajos

# Algoritmo de Bellman-Ford

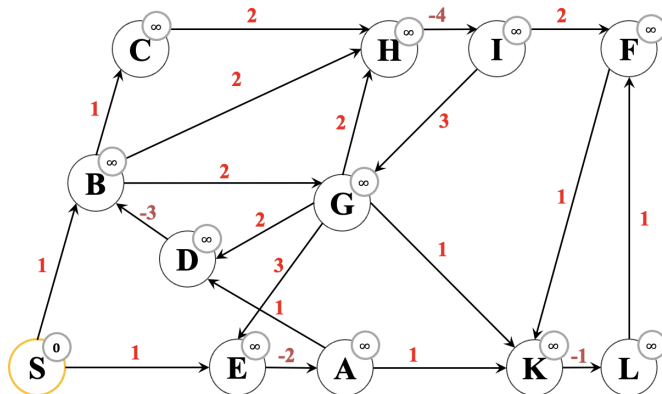
## Ejercicio

Determine los costos mínimos de las rutas más baratas para ir de S a los

demás nodos del siguiente grafo

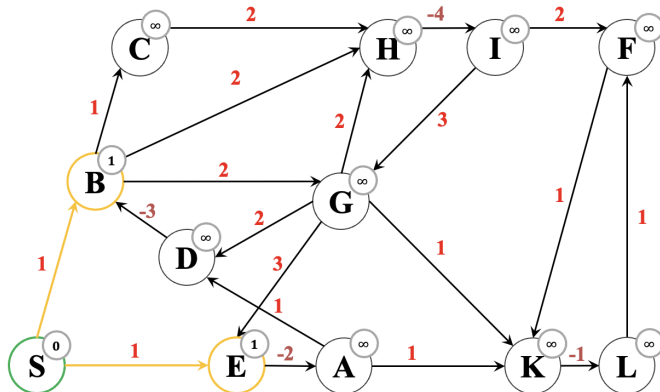


# Algoritmo de Bellman-Ford



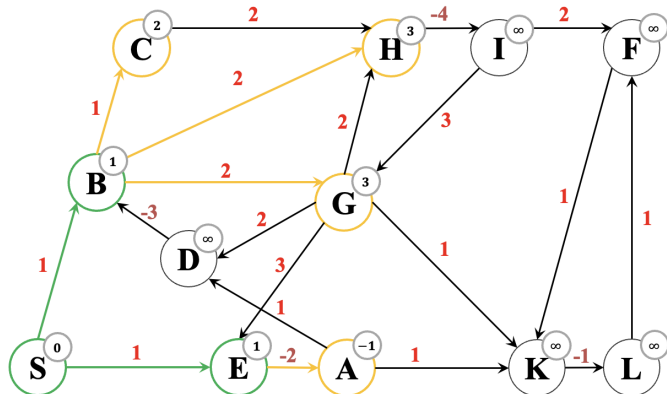
$k = 0$

# Algoritmo de Bellman-Ford



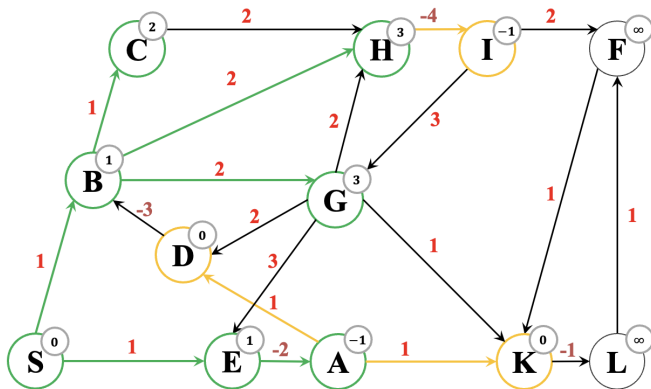
$k = 1$

# Algoritmo de Bellman-Ford



$k = 2$

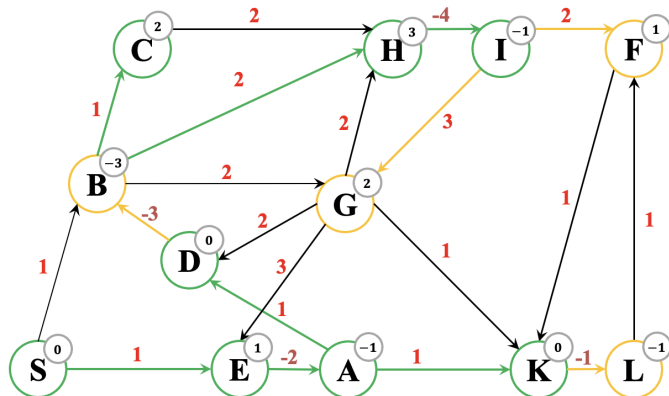
# Algoritmo de Bellman-Ford



$k = 3$

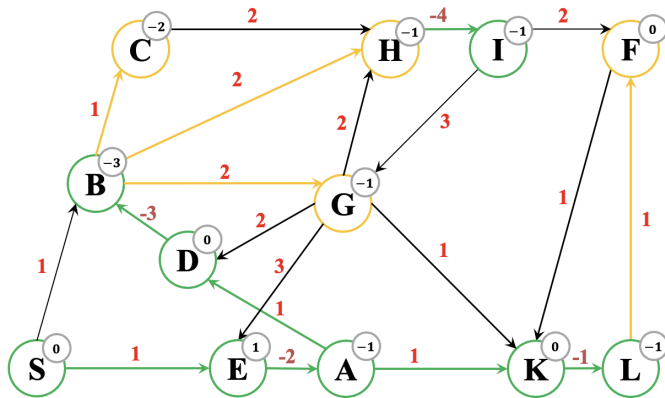


# Algoritmo de Bellman-Ford



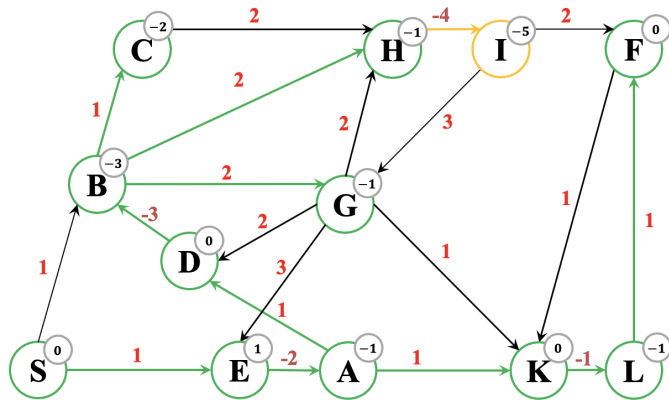
$k = 4$

# Algoritmo de Bellman-Ford



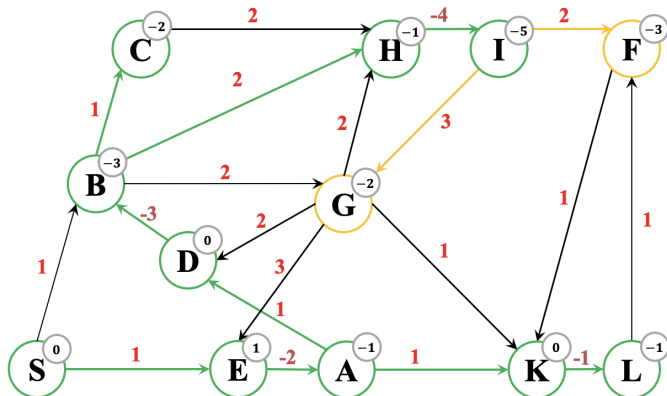
$k = 5$

# Algoritmo de Bellman-Ford



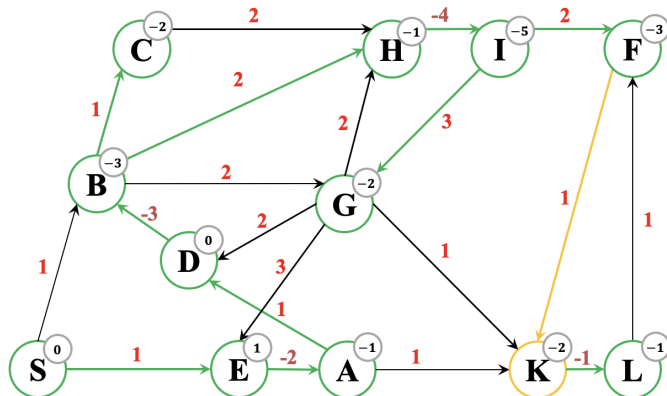
$k = 6$

# Algoritmo de Bellman-Ford



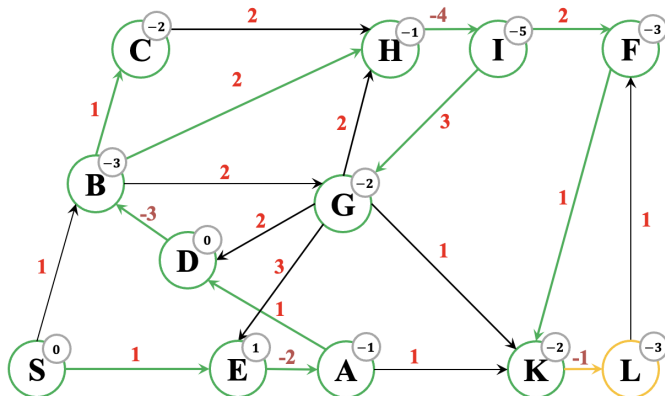
$k = 7$

# Algoritmo de Bellman-Ford



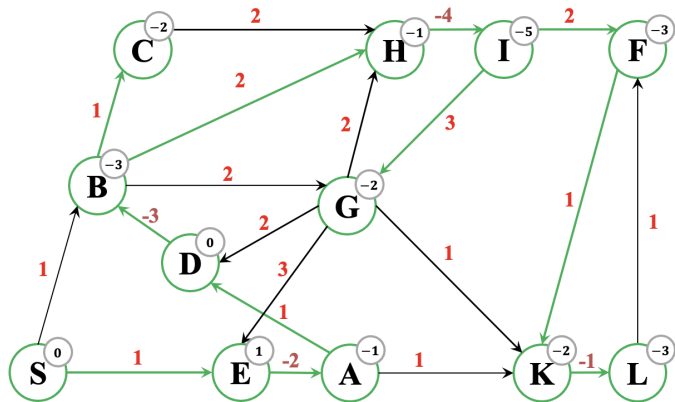
$k = 8$

# Algoritmo de Bellman-Ford



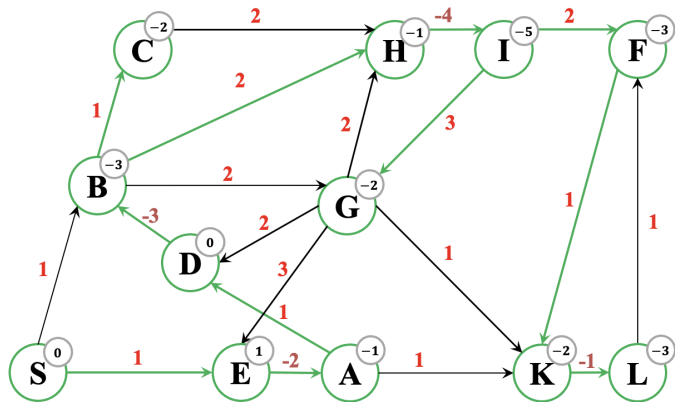
$k = 9$

# Algoritmo de Bellman-Ford



$k = 10$

# Algoritmo de Bellman-Ford



$k = 11$



# Ciclos con costo negativos

Detalle importante!

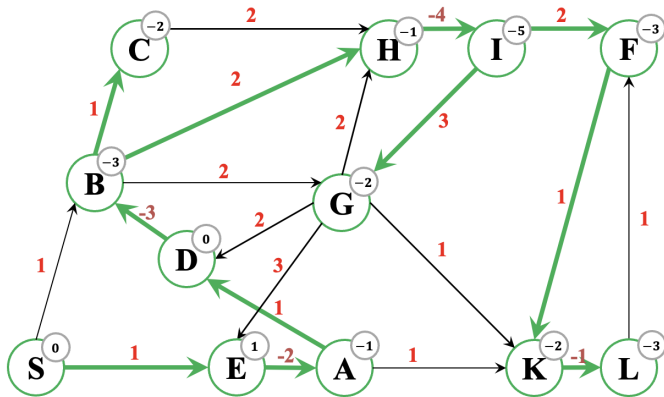
- Dijimos que  $|V| - 1$  iteraciones eran suficientes porque no habían ciclos

Lo que interesa no es que el **grafo sea acíclico**

Importa que **desde la fuente** no se llegue a ningún **ciclo con costo negativo** (pues es el único tipo de ciclo que puede aparecer)

¿Cómo chequeamos que no hayan ciclos de costo negativo?

## Ciclos con costo negativos



¿Qué pasa si al agregar una arista más, logramos mejorar un costo?

# Algoritmo de Bellman-Ford

**ValidBellmanFord( $s$ ):**

```
1  for  $u \in V$  :  
2       $d[u] \leftarrow \infty$ ;  $\pi[u] \leftarrow \emptyset$   
3   $d[s] \leftarrow 0$   
4  for  $k = 1 \dots |V| - 1$  :  
5      for  $(u, v) \in E$  :  
6          if  $d[v] > d[u] + \text{cost}(u, v)$  :  
7               $d[v] \leftarrow d[u] + \text{cost}(u, v)$   
8               $\pi[v] \leftarrow u$   
9  for  $(u, v) \in E$  :  
10     if  $d[v] > d[u] + \text{cost}(u, v)$  :  
11         return false  
12 return true
```

El resultado de este algoritmo indica si los costos en  $d$  son válidos como costos de rutas más cortas

# Complejidad

ValidBellmanFord( $s$ ):

```
1  for  $u \in V$  :  
2       $d[u] \leftarrow \infty$ ;  $\pi[u] \leftarrow \emptyset$   
3   $d[s] \leftarrow 0$   
4  for  $k = 1 \dots |V| - 1$  :  
5      for  $(u, v) \in E$  :  
6          if  $d[v] > d[u] + c(u, v)$  :  
7               $d[v] \leftarrow d[u] + c(u, v)$   
8               $\pi[v] \leftarrow u$   
9  for  $(u, v) \in E$  :  
10     if  $d[v] > d[u] + c(u, v)$  :  
11         return false  
12 return true
```

Actualización de costos

- $\mathcal{O}(V)$  iteraciones
- Cada una hace  $\mathcal{O}(E)$  comparaciones
- Total:  $\mathcal{O}(VE)$

Chequeo de ciclos

- $\mathcal{O}(E)$  comparaciones

Bellman-Ford es un algoritmo  $\mathcal{O}(VE)$  en el peor caso

# Sumario

Obertura

Algoritmo de Bellman-Ford

Epílogo

# Epílogo

Vea

**www.menti.com**

Introduce el código

**5337 5048**



O usa el código QR