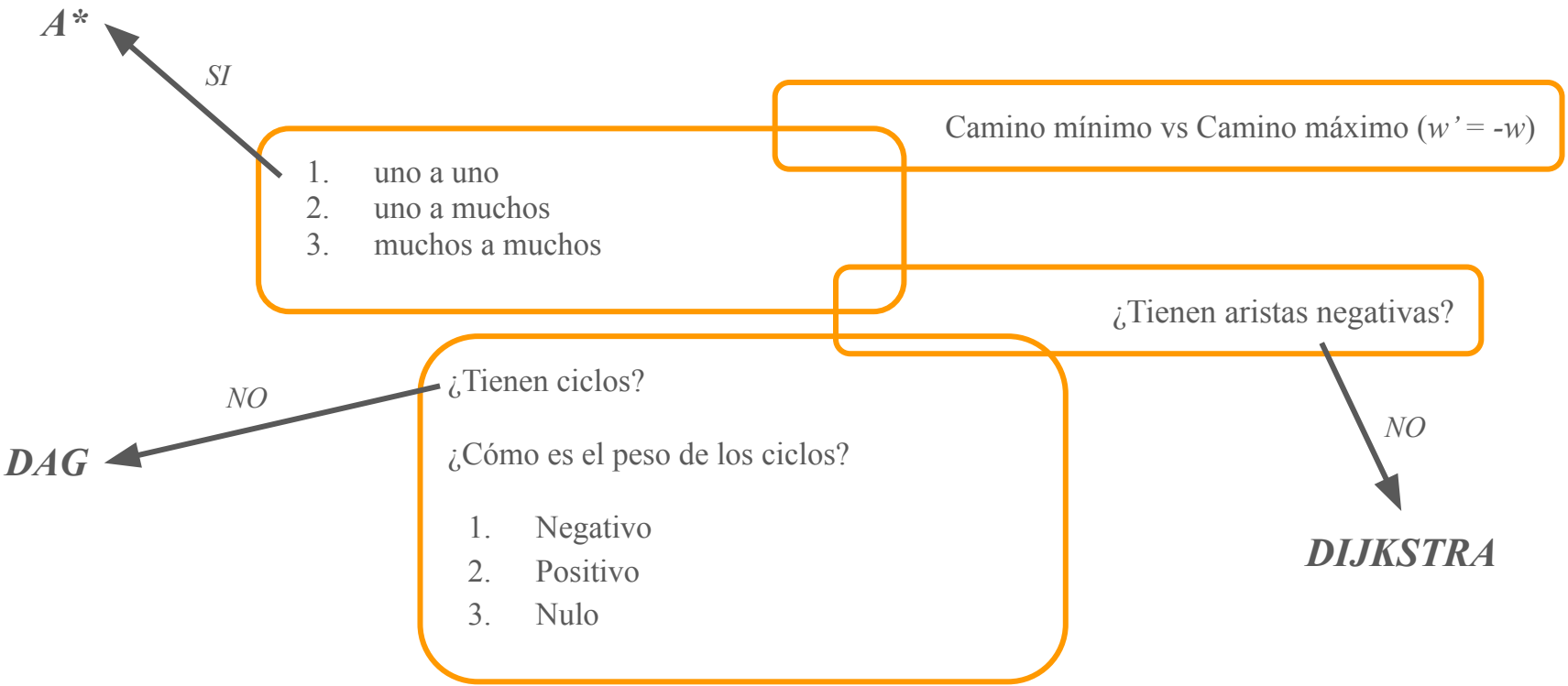
AED3 > Clase 8 >
Camino mínimo. Uno a todos.

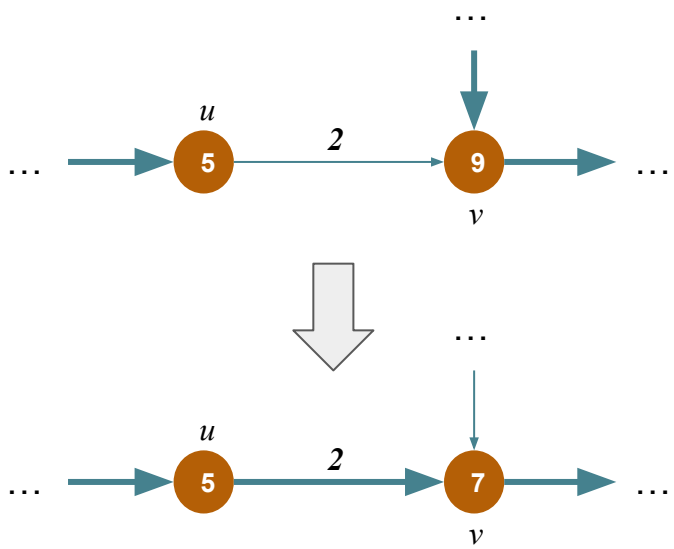
Repaso: Topología de problemas de camino mínimo



Repaso: Relajación

Propiedad de RELAJACIÓN

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```



Repaso: Relajación

$d(u, v)$ distancia estimada entre u y v, $\delta(u, v)$ distancia mínima (real) entre u y v

Desigualdad triangular Sea $(u , v) \in E$, $\delta(s, v) \leq \delta(s, u) + w(u, v)$

Límite superior Sea $v \in V$, $d(s, v) \geq \delta(s, v)$, y una vez que $d(s, v) = \delta(s, v)$ ya no cambia.

Nodos no alcanzables Si no hay camino de s a v, entonces $d(s, v) = \delta(s, v) = \infty$.

Convergencia Sea P = s→…→ u → v un camino mínimo de u a v y $d(s, u) = \delta(s, u)$ en un paso dado. Entonces, luego de relajar (u, v), ocurre que $d(s, v) = \delta(s, v)$.

Relajación Si $P = v_0, v_1, \ldots , v_k$ es un camino mínimo de $s = v_0$ a v_k, y se relajan los ejes en orden $(v_0, v_1), (v_1, v_2), \ldots , (v_{k-1}, v_k)$, entonces $d(s, v_k) = \delta(s, v_k)$. Esta propiedad se mantiene aunque se relajen otras aristas en el medio.

Subgrafo de predecesores Si se cumple que $d(s, v) = \delta(s, v) \ \forall \ v \in V$, entonces el subgrafo que forman los predecesores es un s-ACM.

Repaso: Relajación

$d(u, v)$ distancia estimada entre u y v, $\delta(u, v)$ distancia mínima (real) entre u y v

Relajación Si $P = v_0, v_1, \dots, v_k$ es un camino mínimo de $s = v_0$ a v_k, y se relajan los ejes en orden $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$, entonces $d(s, v_k) = \delta(s, v_k)$. Esta propiedad se mantiene aunque se relajen otras aristas en el medio.

Demo: (inducción)

Caso base: $P = v_0 \Rightarrow d(s = v_0, v_0) = \delta(v_0, v_0) = 0$. Por la prop. del **Límite superior** $d(v_0, v_0)$ ya no cambia.

Hipótesis inductiva: $P = v_0, v_1, \dots, v_{i-1}$ con $d(s, v_{i-1}) = \delta(s, v_{i-1})$

Paso inductivo: voy a relajar (v_{i-1}, v_i). Por la prop. de **Convergencia**, si se cumple la Hipótesis inductiva $\Rightarrow d(v_0, v_i) = \delta(v_0, v_i)$. Por la prop. del **Límite superior** $d(v_0, v_i)$ ya no cambia.

Repaso: DAGs

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

```
INIT ( G, s ) :
|     for v in V:
|     |     v.d = Inf
|     |     v.pred = None
s.d = 0
```

```
DAG ( G, s ) :
|     TOPOLOGICAL-SORT(G ):
|     INIT(G, s)
|     for u in V (en el orden de TOPOLOGICAL-SORT:
|     |     for v in Adj[u]:
|     |     |     RELAX(u, v)
```

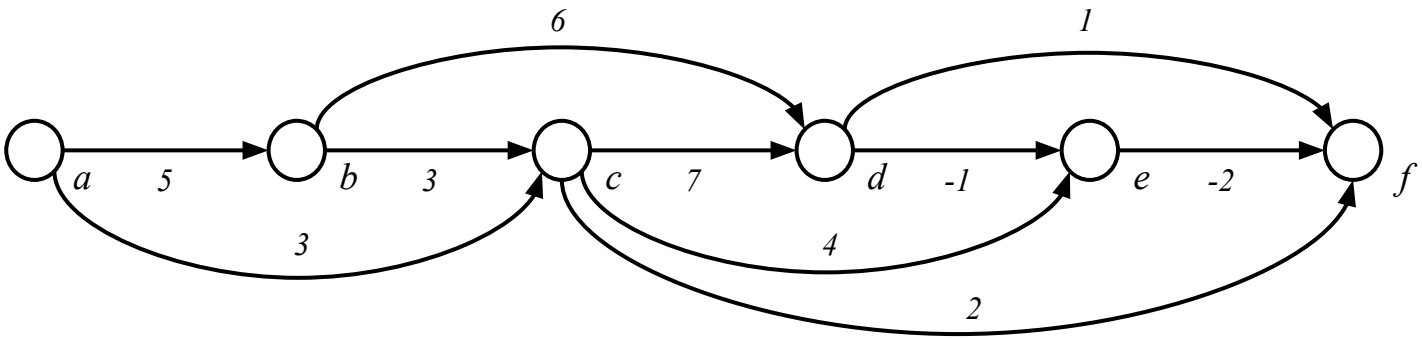
Repaso: DAGs

¿Cuánto tiempo me va a llevar la tarea?

¿Cuál es el cuello de botella?

```
DAG ( G, s ) :
|     TOPOLOGICAL-SORT(G ):
|     INIT(G, s)
|     for u in V (en el orden de TOPOLOGICAL-SORT:
|     |     for v in Adj[u]:
|     |     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

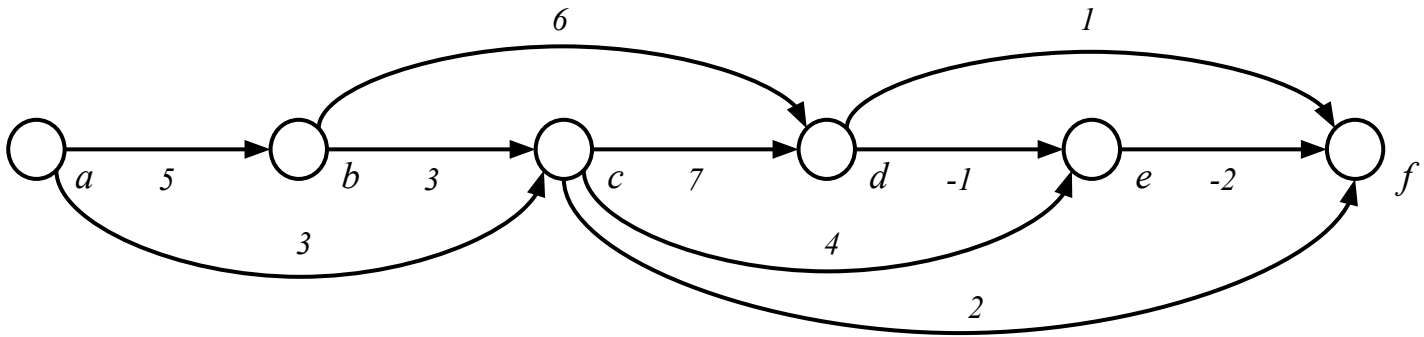


Repaso: DAGs

TOPOLOGICAL-SORT: a→b→c→d→e→f

```
DAG ( G, s ) :
|     TOPOLOGICAL-SORT(G ):
|     INIT(G, s)
|     for u in V (en el orden de TOPOLOGICAL-SORT:
|     |     for v in Adj[u]:
|     |     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

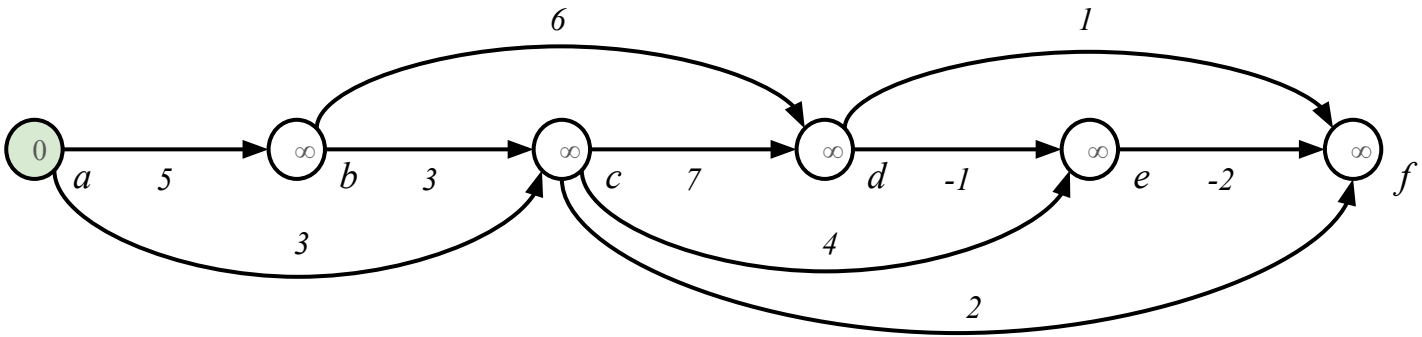


Repaso: DAGs

TOPOLOGICAL-SORT: a→b→c→d→e→f

```
DAG ( G, s ) :
|     TOPOLOGICAL-SORT(G ):
|     INIT(G, s)
|     for u in V (en el orden de TOPOLOGICAL-SORT:
|     |     for v in Adj[u]:
|     |     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

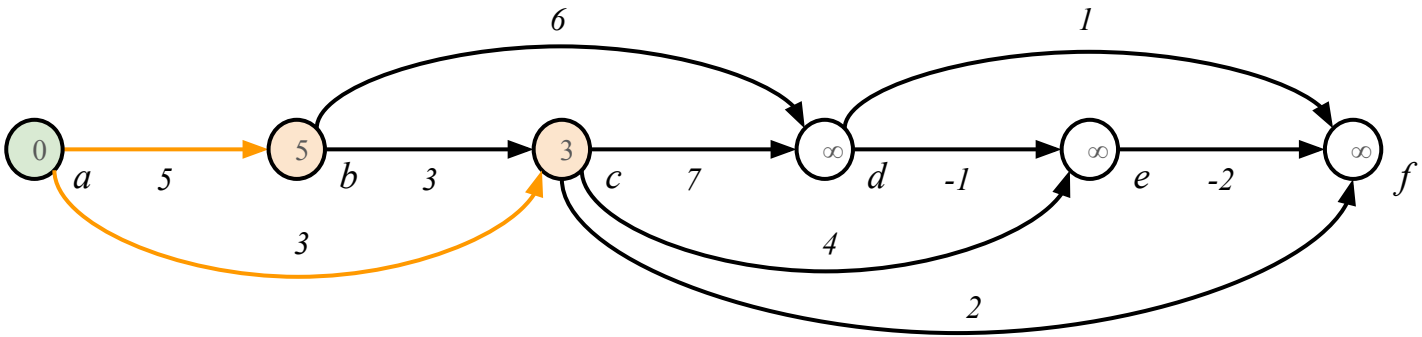


Repaso: DAGs

TOPOLOGICAL-SORT: a→b→c→d→e→f

```
DAG ( G, s ) :
|     TOPOLOGICAL-SORT(G ):
|     INIT(G, s)
|     for u in V (en el orden de TOPOLOGICAL-SORT:
|     |     for v in Adj[u]:
|     |     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

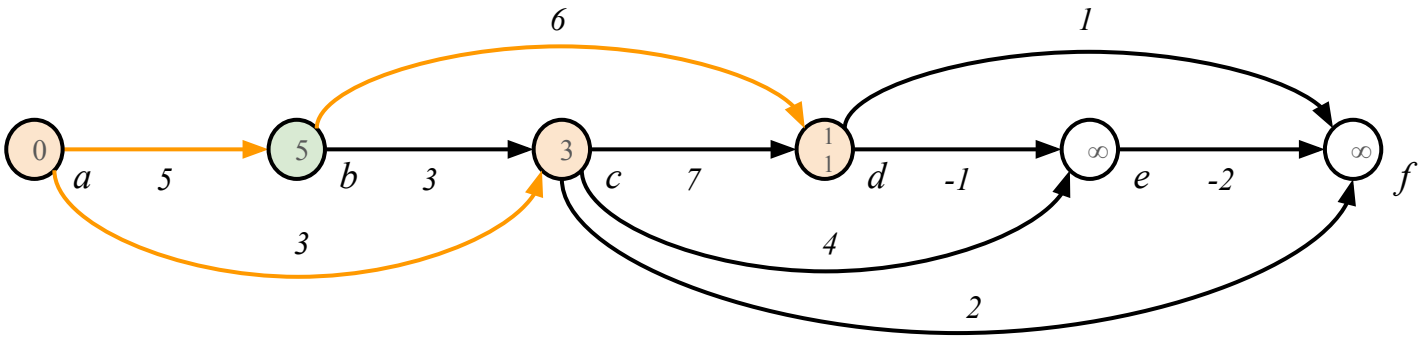


Repaso: DAGs

TOPOLOGICAL-SORT: a→b→c→d→e→f

```
DAG ( G, s ) :
|     TOPOLOGICAL-SORT(G ):
|     INIT(G, s)
|     for u in V (en el orden de TOPOLOGICAL-SORT:
|     |     for v in Adj[u]:
|     |     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

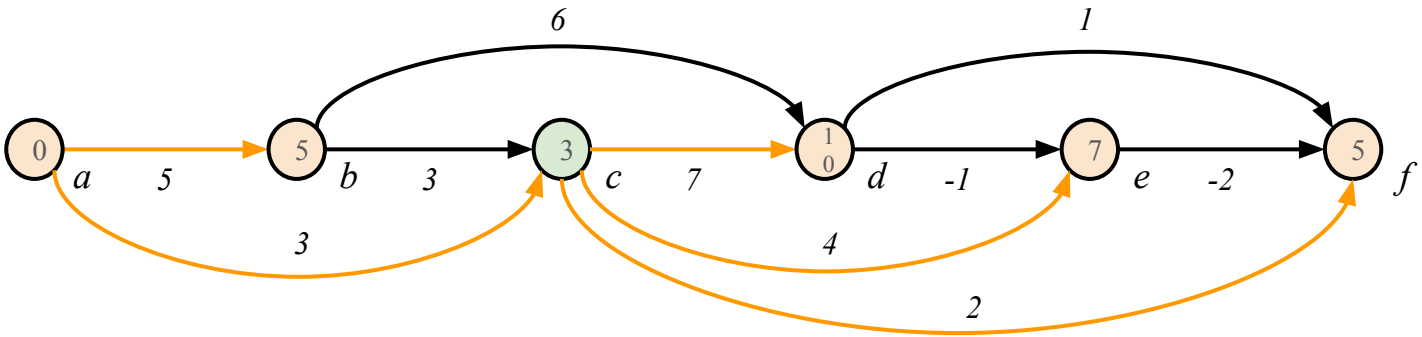


Repaso: DAGs

TOPOLOGICAL-SORT: a→b→c→d→e→f

```
DAG ( G, s ) :
|     TOPOLOGICAL-SORT(G ):
|     INIT(G, s)
|     for u in V (en el orden de TOPOLOGICAL-SORT:
|     |     for v in Adj[u]:
|     |     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

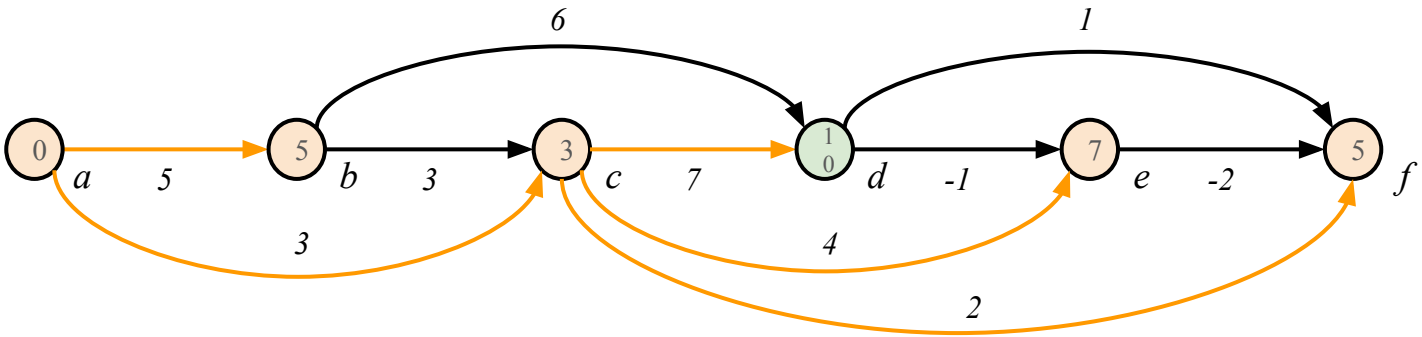


Repaso: DAGs

TOPOLOGICAL-SORT: a→b→c→d→e→f

```
DAG ( G, s ) :
|     TOPOLOGICAL-SORT(G ):
|     INIT(G, s)
|     for u in V (en el orden de TOPOLOGICAL-SORT:
|     |     for v in Adj[u]:
|     |     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

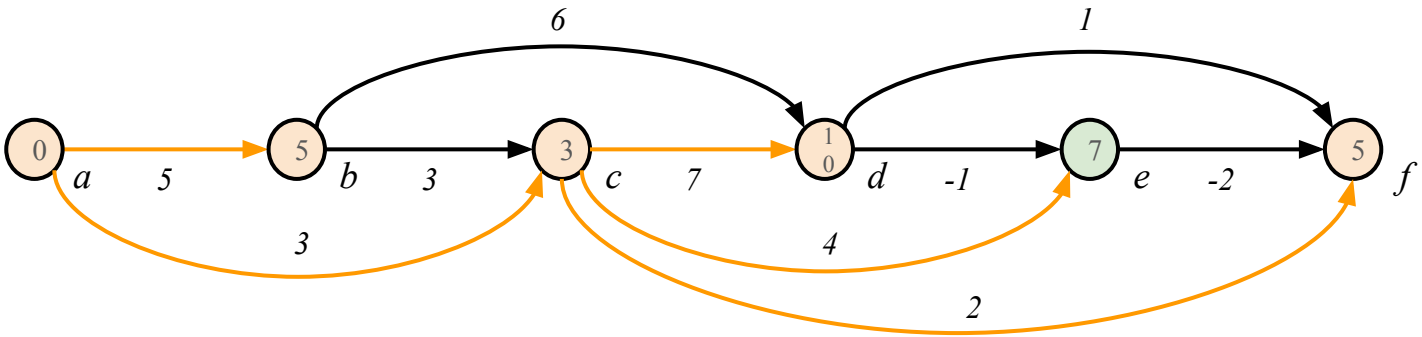


Repaso: DAGs

TOPOLOGICAL-SORT: a→b→c→d→e→f

```
DAG ( G, s ) :
|     TOPOLOGICAL-SORT(G ):
|     INIT(G, s)
|     for u in V (en el orden de TOPOLOGICAL-SORT:
|     |     for v in Adj[u]:
|     |     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

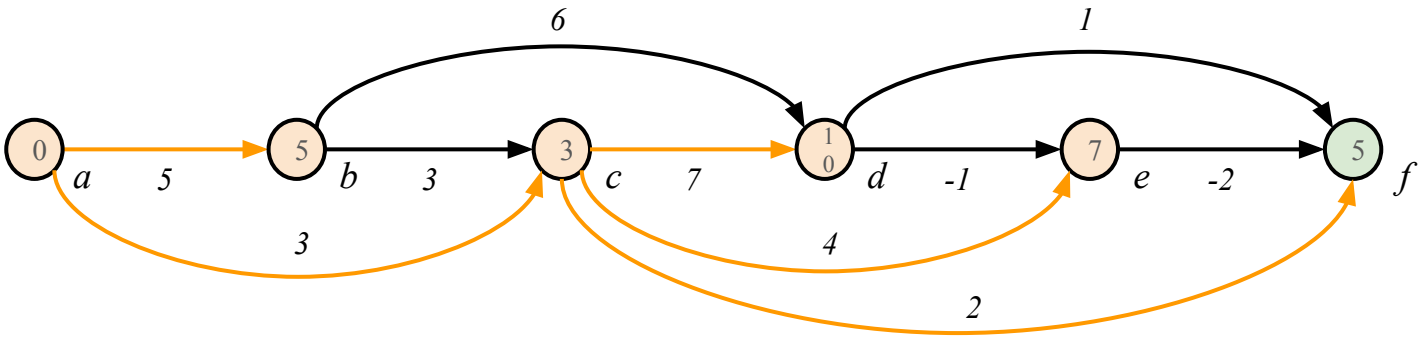


Repaso: DAGs

TOPOLOGICAL-SORT: a→b→c→d→e→f

```
DAG ( G, s ) :
|     TOPOLOGICAL-SORT(G ):
|     INIT(G, s)
|     for u in V (en el orden de TOPOLOGICAL-SORT:
|     |     for v in Adj[u]:
|     |     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```



Repaso: Dijkstra

Aristas no negativas: $w(u, v) \geq 0$

```
DIJKSTRA ( G, s ) :
|     INIT(G, s)
|     S = ∅
|     Q = ∅
|     for u in V:
|     |     INSERT(Q, u)
|     while Q:
|     |     u = EXTRACT-MIN()
|     |     S = S U {u}
|     |     for v in Adj[u]:
|     |     |     RELAX(u, v)
```

```
INIT ( G, s ) :
|     for v in V:
|     |     v.d = Inf
|     |     v.pred = None
s.d = 0
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
|     |     DECREASE-KEY(Q, v, v.d)
```

Repaso: Dijkstra

Aristas no negativas: $w(u, v) \geq 0$

```
DIJKSTRA ( G, s ) :
|     INIT(G, s)
|     S = ∅
|     Q = ∅
|     for u in V:
|     |     INSERT(Q, u)
|     while Q:
|     |     u = EXTRACT-MIN()
|     |     S = S U {u}
|     |     for v in Adj[u]:
|     |     |     RELAX(u, v)
|     |     |     if v.pred == u:
|     |     |     |     DECREASE-KEY(Q, v, v.d)
```

```
INIT ( G, s ) :
|     for v in V:
|     |     v.d = Inf
|     |     v.pred = None
s.d = 0
```

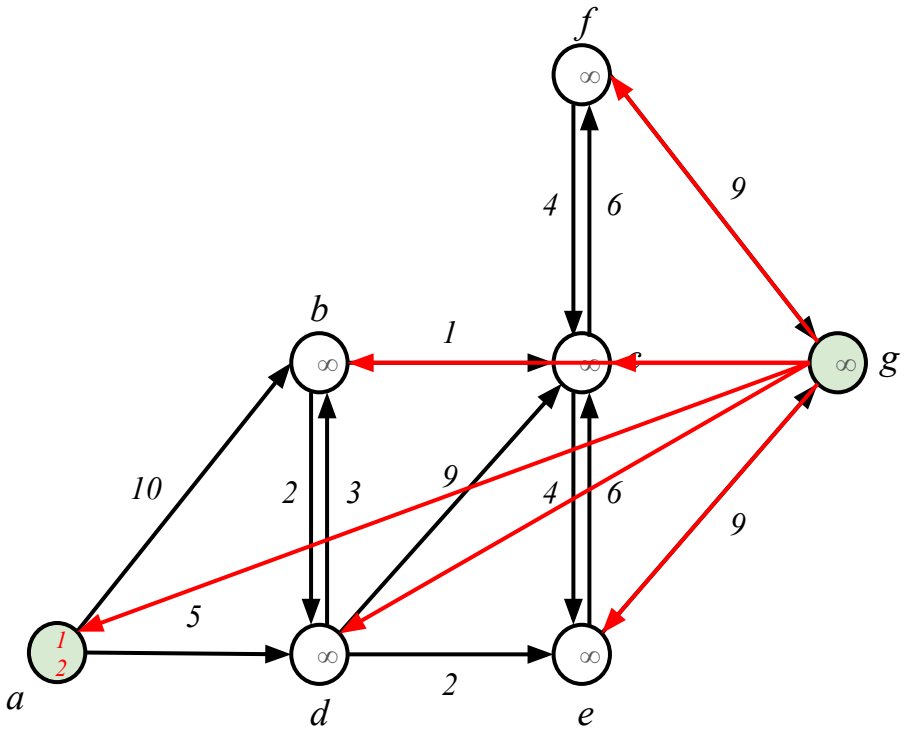
```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Repaso: A*

Inicializo los valores con una distancia estimada al objetivo (heurística). Es importante que subestime (admisible), y que sea consistente (que respete las distancias).

$h(g,x) \leq d(g,x)$ (admisible)

$|h(g,x) - h(g,y)| \leq d(x,y)$ (consistente)



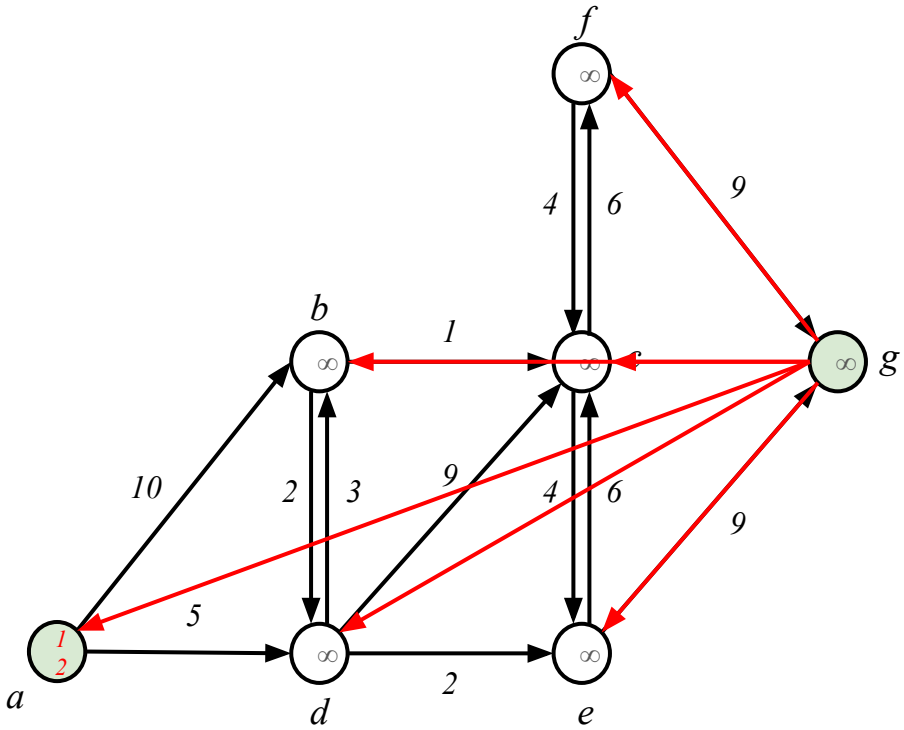
Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, *4*(2), 100-107.

Repaso: A*

Inicializo los valores con una distancia estimada al objetivo (heurística). Es importante que subestime (admisible), y que sea consistente (que respete las distancias).

$h(g,x) \leq d(g,x)$ (admisible)

$|h(g,x) - h(g,y)| \leq d(x,y)$ (consistente)



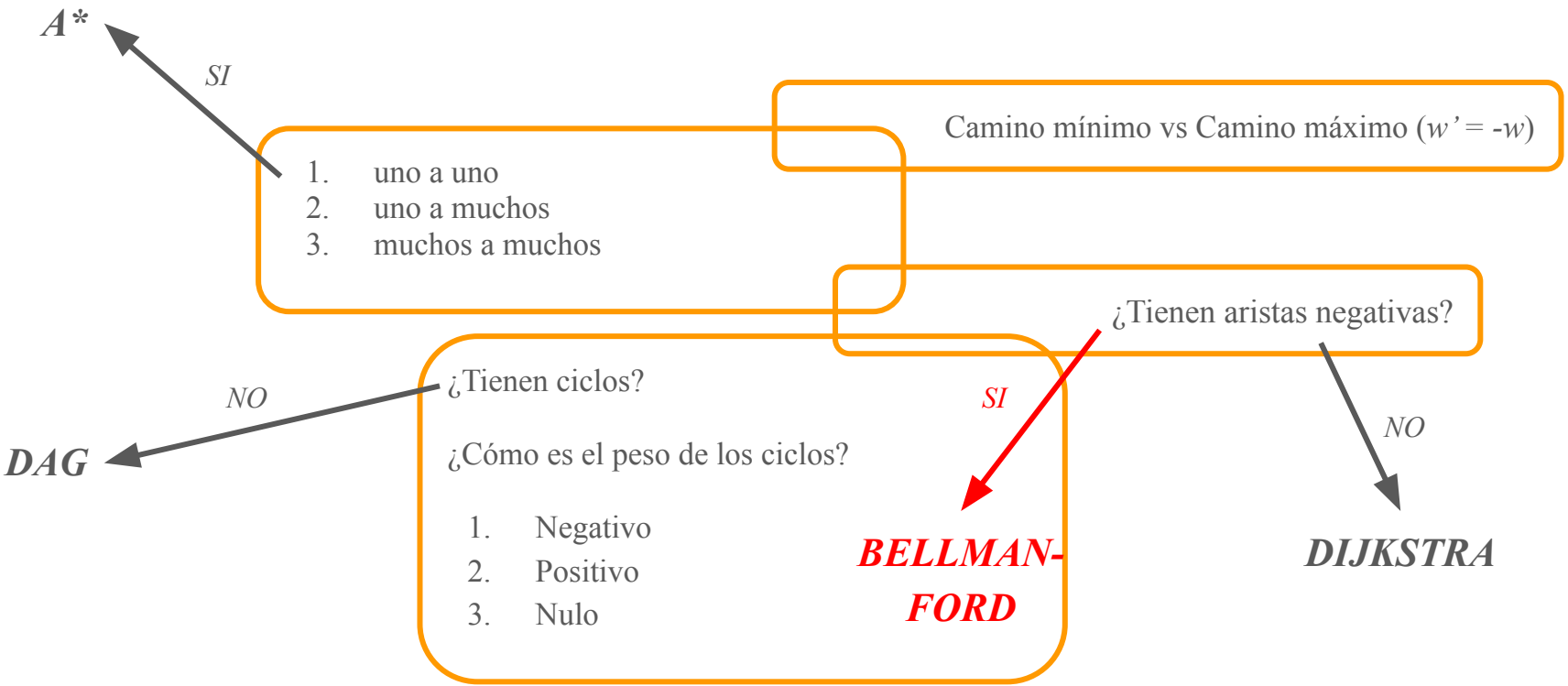
Repaso: A*

```
A-Star ( G, s ) :
|     INIT(G, s)
|     for v in V:
|     |     v.h = heuristica(g,h)
|     |     v.a = v.d + v.h
|     S = ∅
|     Q = [s]
|
|     while Q:
|     |     u = EXTRACT-MIN(Q)
|     |     S = S U {u}
|     |     if u == g:
|     |     |     return S # O le pido todo el grafo
|     |     |
|     |     for v in Adj[u]:
|     |     |     if v not in S:
|     |     |     |     if v not in Q:
|     |     |     |     |     Q = Q U {v}
|     |     |     |     |     v.d = u.d + w(u,v)
|     |     |     |     |     v.pred = u
|     |     |     |     |     v.a = v.d + v.h
|     |     |     |     |     DECREASE-KEY(Q, v, v.a)
```

```
INIT ( G, s ) :
|     for v in V:
|     |     v.d = Inf
|     |     v.pred = None
s.d = 0
```

AED3 > Clase 8 >
Camino mínimo. Uno a todos.

Repaso: Topología de problemas de camino mínimo



Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|     return True, v
```

```
INIT ( G, s ) :
|     for v in V:
|     |     v.d = Inf
|     |     v.pred = None
s.d = 0
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

¡Relajo TODAS las aristas a la vez!

¡n - 1 veces!

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Chequeo que no haya ciclos negativos:

Repito una vez más,

si todavía es posible seguir achicando distancias ⇒

¡Hay ciclos negativos!

Nota 1: No puedo decir a quienes afectan, sólo que son alcanzables desde “s”)

Nota 2: Querer decir más es NP-completo (*spoiler alert!!*)

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Si no corta ⇒ ¡NO hay ciclos negativos!

Devuelve lo mismo que antes.

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Lema (casi correctitud):

$G = (V, E, w)$ sin ciclos negativos, luego de $n-1$ iteraciones $\Rightarrow$

$v.d = \delta(s,v) \quad \forall\ v$ alcanzable desde s

Demo:

Bellman-Ford

```
for i = 1 to n-1:
|     for (u,v) in E:
|     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Lema (casi correctitud):

$G = (V, E, w)$ sin ciclos negativos, luego de $n-1$ iteraciones $\Rightarrow v.d = \delta(s,v) \ \forall\ v$ alcanzable desde s

Demo:

Sea $P = v_0, v_1, \ldots, v_{k-1}, v_k$ es un camino mínimo de $s = v_0$ a v_k,

como es camino simple $\Rightarrow k \leq n-1$

en cada iteración se relajan todas las aristas, en particular las de P

es decir que en algún momento de la primer iteración relajo $(v_0, v_1) \Rightarrow d(s, v_1) = \delta(s, v_1)$ y no cambia más! (x ***Límite superior***)

luego, en algún momento de la segunda iteración, relajo $(v_1, v_2) \Rightarrow d(s, v_2) = \delta(s, v_2)$ y no cambia más! (x ***Límite superior***, *y* ***propiedad de Relajación***, *lo hice en orden*)

Bellman-Ford

```
for i = 1 to n-1:
|     for (u,v) in E:
|     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Lema (casi correctitud):

$G = (V, E, w)$ sin ciclos negativos, luego de $n-1$ iteraciones $\Rightarrow v.d = \delta(s,v) \ \forall\ v$ alcanzable desde s

Demo:

Sea $P = v_0, \ldots, v_k$ es un camino mínimo de $s = v_0$ a v_k, como es camino simple $\Rightarrow k \leq n-1$

es decir que en algún momento de la primer iteración relajo $(v_0, v_1) \Rightarrow d(s, v_1) = \delta(s, v_1)$ y no cambia más! (x ***Límite superior***)

luego, en algún momento de la segunda iteración, relajo $(v_1, v_2) \Rightarrow d(s, v_2) = \delta(s, v_2)$ y no cambia más! (x ***Límite superior***, *y* ***propiedad de Relajación***, *lo hice en orden*)

luego, en algún momento de la tercera iteración, relajo $(v_2, v_3) \Rightarrow d(s, v_3) = \delta(s, v_3)$ y no cambia más! (x ***Límite superior***, *y* ***propiedad de Relajación***, *lo hice en orden*)

…, en algún momento de la k-ésima iteración, relajo $(v_{k-1}, v_k) \Rightarrow d(s, v_k) = \delta(s, v_k)$ y no cambia más! (x ***Límite superior***, *y* ***propiedad de Relajación***, *lo hice en orden*)

Bellman-Ford

```
for i = 1 to n-1:
|     for (u,v) in E:
|     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Lema (casi correctitud):

$G = (V, E, w)$ sin ciclos negativos, luego de $n-1$ iteraciones $\Rightarrow v.d = \delta(s,v) \ \forall\ v$ alcanzable desde s

Demo:

Sea $P = v_0, v_1, \ldots, v_{k-1}, v_k$ es un camino mínimo de $s = v_0$ a v_k,

como es camino simple $\Rightarrow k \leq n-1$

en cada iteración se relajan todas las aristas, en particular las de P

…, en algún momento de la k-ésima iteración, relajo $(v_{k-1}, v_k) \Rightarrow d(s, v_k) = \delta(s, v_k)$ y no cambia más! (x ***Límite superior***, *y* ***propiedad de Relajación***, *lo hice en orden*)

y k es a lo sumo $n-1$

$\Rightarrow v.d = d(s, v) = \delta(s, v) \ \forall\ v$ alcanzable desde s

Bellman-Ford

```
for i = 1 to n-1:
|     for (u,v) in E:
|     |     RELAX(u, v)
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Lema (casi correctitud):

G = (V, E, w) sin ciclos negativos, luego de *n-1* iteraciones $\Rightarrow$ $v.d = \delta(s,v) \ \forall \ v$ alcanzable desde *s*

Corolario:

Luego de ejecutar Bellman-Ford, si no tengo ciclos negativos, obtengo un camino simple entre *s* y *v*, y $v.d = \delta(s,v) \ \forall \ v$ alcanzable desde *s*

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Teorema (correctitud):

Si $G = (V, E, w)$ no contiene ciclos negativos alcanzable desde s, luego de ejecutar Bellman-Ford $\Rightarrow$ $v.d = \delta(s,v) \ \forall \ v$ alcanzable desde s y los predecesores ($v.pred$) forman un *s-ACM*

Si $G = (V, E, w)$ contiene ciclos negativos alcanzable desde s, luego de ejecutar Bellman-Ford $\Rightarrow$ Devuelve *False*

Demo:

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Teorema (correctitud):

Si $G = (V, E, w)$ no contiene ciclos negativos alcanzable desde s, luego de ejecutar Bellman-Ford $\Rightarrow$ $v.d = \delta(s,v)$ $\forall$ v alcanzable desde s y los predecesores ($v.pred$) forman un *s-ACM*

Si $G = (V, E, w)$ contiene ciclos negativos alcanzable desde s, luego de ejecutar Bellman-Ford $\Rightarrow$ Devuelve *False*

Demo:

Si $G = (V, E, w)$ no contiene ciclos negativos alcanzable desde s, luego de ejecutar Bellman-Ford $\Rightarrow$

(por ***Lema anterior*** + ***Límite superior***, *es decir que en el segundo for no se actuliza más*)

$v.d = \delta(s,v)$ $\forall$ v alcanzable desde s y los predecesores ($v.pred$) forman un *s-ACM*

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Teorema (correctitud):

…

Si $G = (V, E, w)$ contiene ciclos negativos alcanzable desde s, luego de ejecutar Bellman-Ford $\Rightarrow$ Devuelve *False*

Demo:

Si $G = (V, E, w)$ contiene ciclos negativos alcanzable desde s, luego de ejecutar Bellman-Ford $\Rightarrow$ Existe $C = v_0, \ldots v_k$, con $v_0 = v_k$ tq $\sum_i w(v_{i-1}, v_i) < 0$

(por el absurdo) Supongo que devuelve True $\Rightarrow$ No entra a ningún "if" $\Rightarrow$

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Demo: Si $G = (V, E, w)$ contiene ciclos negativos alcanzable desde s, luego de ejecutar Bellman-Ford $\Rightarrow$ Existe $C = v_0, \ldots v_k$, con $v_0 = v_k$ tq $\sum_i w(v_{i-1}, v_i) < 0$

(por el absurdo) Supongo que devuelve True $\Rightarrow$ No entra a ningún "if" $\Rightarrow$

$d(s,v_i) \leq d(s,v_{i-1}) + w(v_{i-1},v_i) \Rightarrow$

$d(s,v_i) - d(s,v_{i-1}) \leq w(v_{i-1},v_i) \Rightarrow$

$\sum_i (d(s,v_i) - d(s,v_{i-1})) \leq \sum_i w(v_{i-1},v_i) < 0$

$v_1 - v_0 + v_2 - v_1 + \ldots + v_{k-1} - v_{k-2} + v_k - v_{k-1} =$

$v_k - v_0 =$

$v_0 - v_0 =$

0

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

Demo: Si $G = (V, E, w)$ contiene ciclos negativos alcanzable desde s, luego de ejecutar Bellman-Ford $\Rightarrow$ Existe $C = v_0, \ldots v_k$, con $v_0 = v_k$ tq $\sum_i w(v_{i-1},v_i) < 0$

(por el absurdo) Supongo que devuelve True $\Rightarrow$ No entra a ningún "if" $\Rightarrow$

$d(s,v_i) \leq d(s,v_{i-1}) + w(v_{i-1},v_i) \Rightarrow$

$d(s,v_i) - d(s,v_{i-1}) \leq w(v_{i-1},v_i) \Rightarrow$

$\sum_i (d(s,v_i) - d(s,v_{i-1})) \leq \sum_i w(v_{i-1},v_i) < 0$

$v_1 - v_0 + v_2 - v_1 + \ldots + v_{k-1} - v_{k-2} + v_k - v_{k-1} = v_k - v_0 = v_0 - v_0 = 0$

$0 \leq \sum_i w(v_{i-1},v_i) < 0$

¡Absurdo!

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|     |
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|     |
|     return True, v
```

O(V)

O(1) } *O(E)* } *O(V*E)*

O(1) } *O(E)*

*O(V + V*E + E)*

denso (E~V²): *O(V³)*

ralo (E~V): *O(V²)*

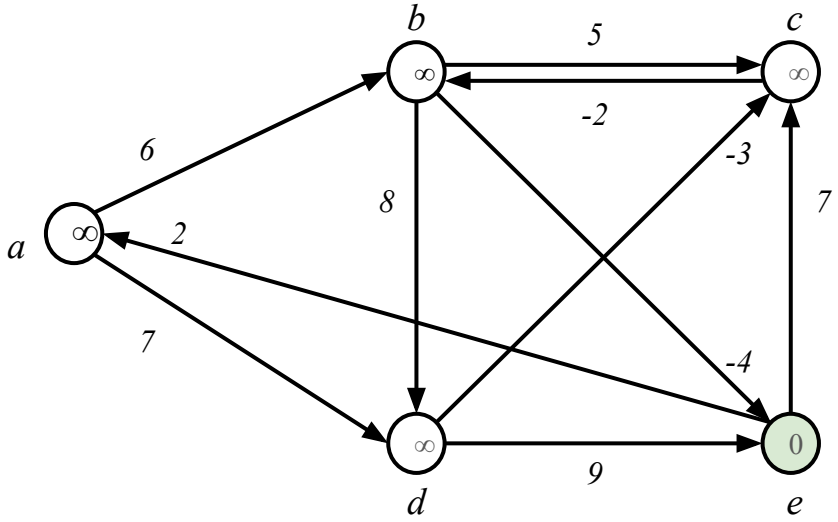
```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

O(1)

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```



(e,c), (c,b), (b,c), (a,b), (a,d), (d,e), (b,e), (d,c), (b,d), (b,c), (e,a)

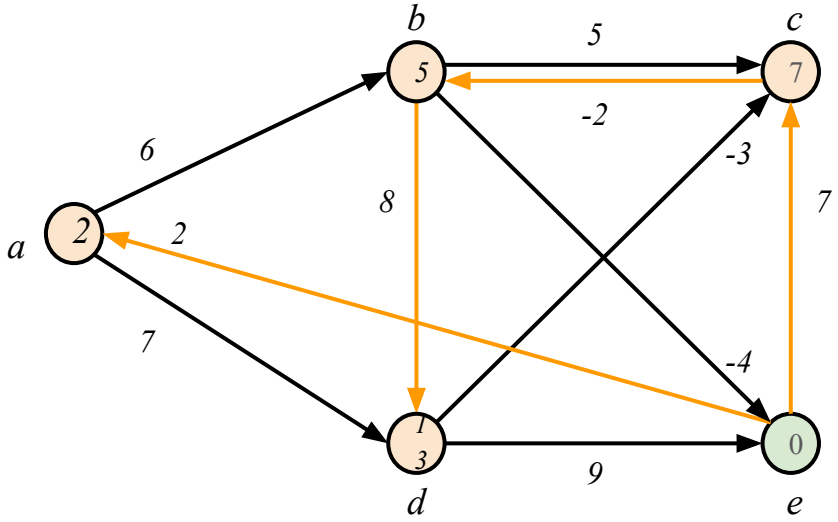
d = {a: ∞, b: ∞, c: ∞, d: ∞, e: 0}

pred = {a: None, b: None, c: None, d: None, e: None}

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```



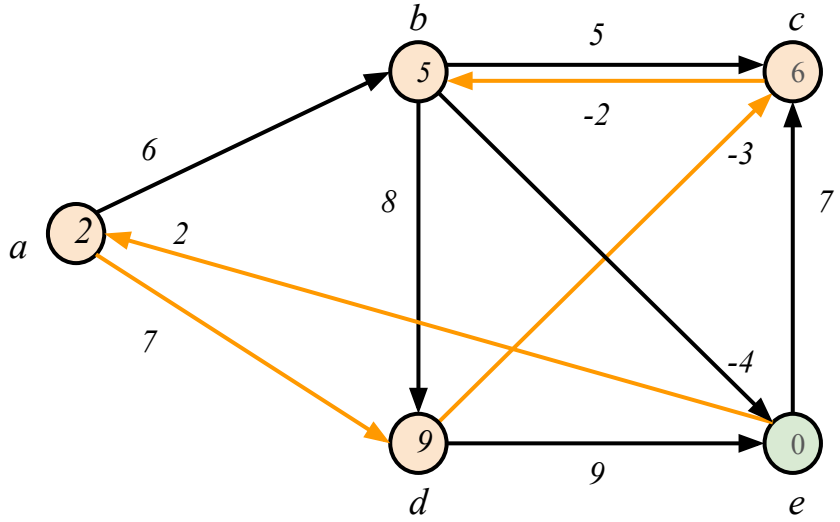
(e,c), (c,b), (b,c), (a,b), (a,d), (d,e), (b,e), (d,c), (b,d), (b,c), (e,a)

i = *1*
d = *{a: 2, b: 5, c: 7, d: 13, e: 0}*
pred = *{a: e, b: c, c: e, d: b, e: None}*

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```



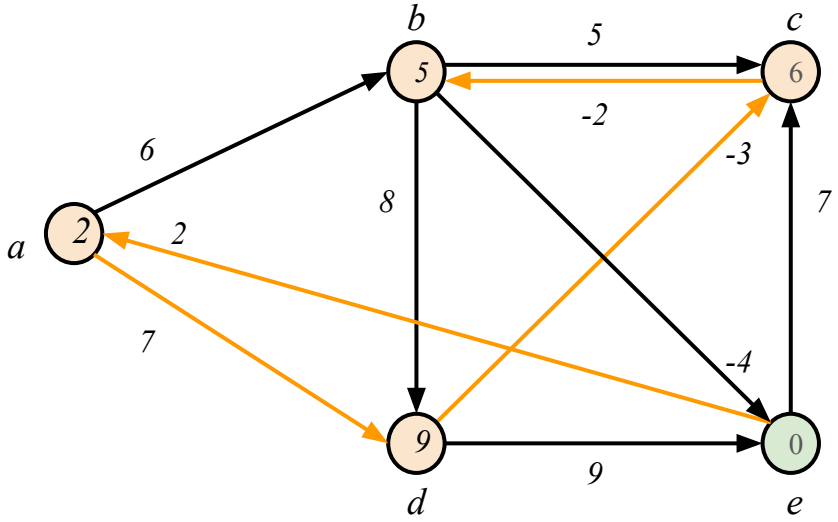
(e,c), (c,b), (b,c), (a,b), (a,d), (d,e), (b,e), (d,c), (b,d), (b,c), (e,a)

i = 2
d = {a: 2, b: 5, c: 6, d: 9, e: 0}
pred = {a: e, b: c, c: d, d: a, e: None}

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```



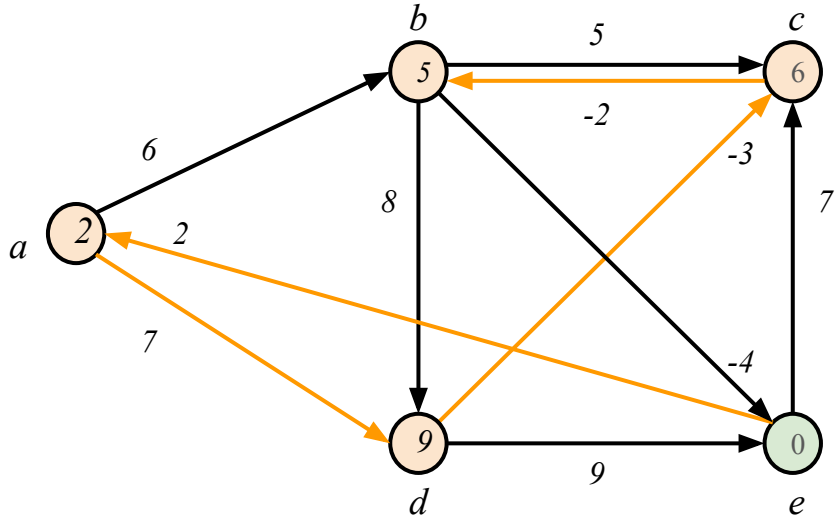
(e,c), (c,b), (b,c), (a,b), (a,d), (d,e), (b,e), (d,c), (b,d), (b,c), (e,a)

i = *3…*
d = *{a: 2, b: 5, c: 6, d: 9, e: 0}*
pred = *{a: e, b: c, c: d, d: a, e: None}*

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```



(e,c), (c,b), (b,c), (a,b), (a,d), (d,e), (b,e), (d,c), (b,d), (b,c), (e,a)

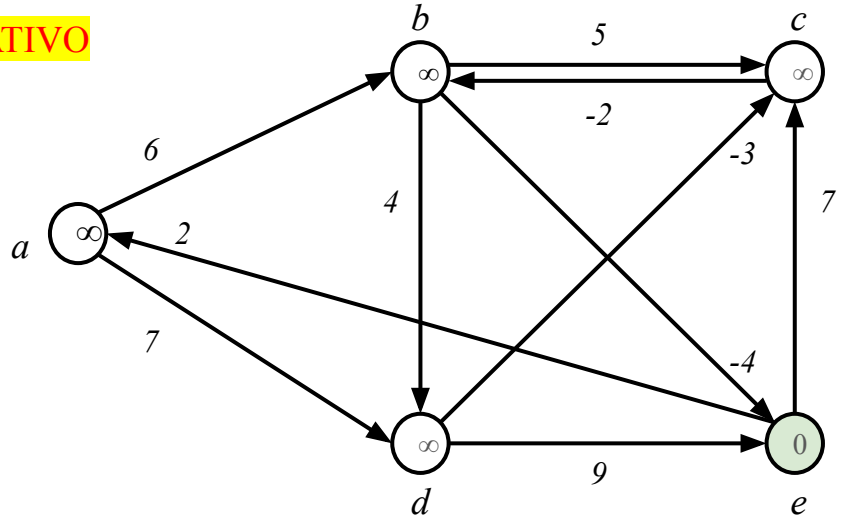
i = 3...
d = {a: 2, b: 5, c: 6, d: 9, e: 0}
pred = {a: e, b: c, c: d, d: a, e: None}

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

CON CICLO NEGATIVO



(e,c), (c,b), (b,c), (a,b), (a,d), (d,e), (b,e), (d,c), (b,d), (b,c), (e,a)

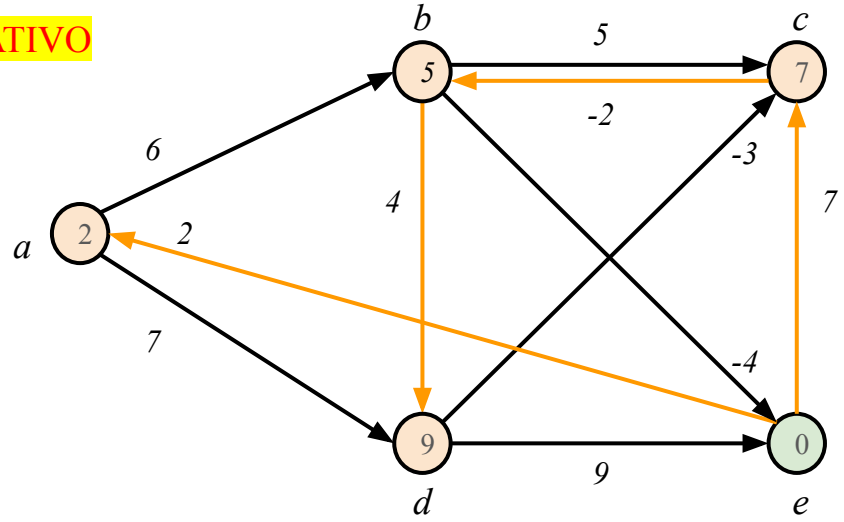
d = {a: ∞, b: ∞, c: ∞, d: ∞, e: 0}
pred = {a: None, b: None, c: None, d: None, e: None}

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

CON CICLO NEGATIVO



(e,c), (c,b), (b,c), (a,b), (a,d), (d,e), (b,e), (d,c), (b,d), (b,c), (e,a)

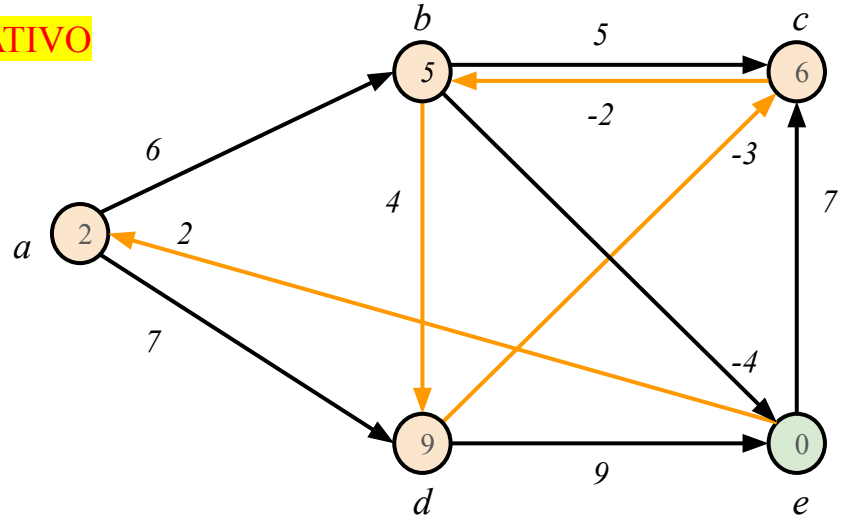
i = 1
d = {a: 2, b: 5, c: 7, d: 9, e: 0}
pred = {a: e, b: c, c: e, d: b, e: None}

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

CON CICLO NEGATIVO



(e,c), (c,b), (b,c), (a,b), (a,d), (d,e), (b,e), (d,c), (b,d), (b,c), (e,a)

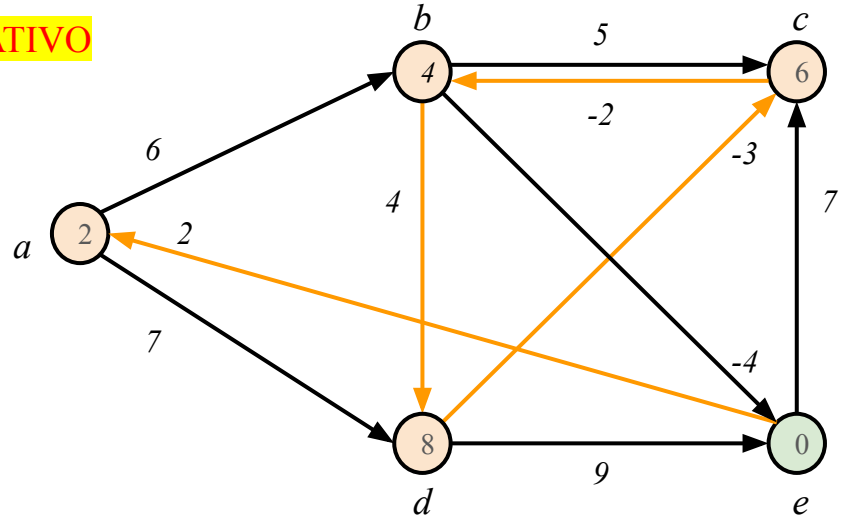
i = 2
d = {a: 2, b: 5, c: 6, d: 9, e: 0}
pred = {a: e, b: c, c: d, d: b, e: None}

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

CON CICLO NEGATIVO



(e,c), (c,b), (b,c), (a,b), (a,d), (d,e), (b,e), (d,c), (b,d), (b,c), (e,a)

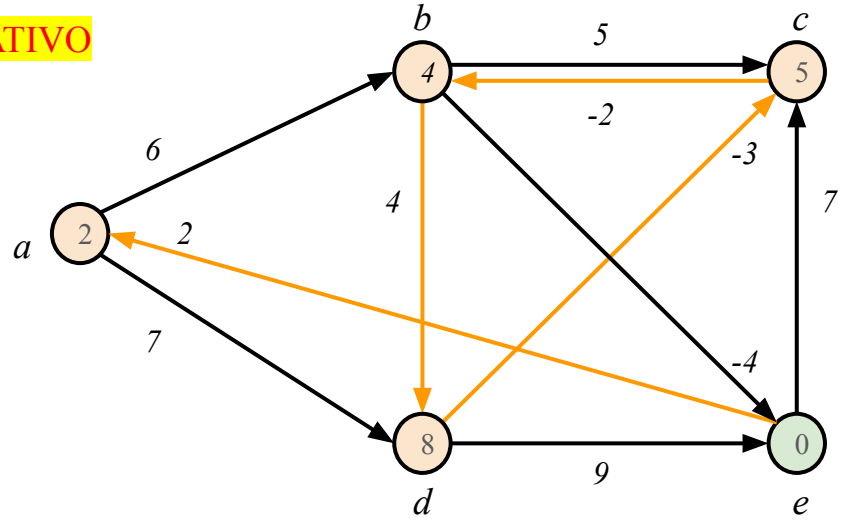
i = 3
d = {a: 2, b: 4, c: 6, d: 8, e: 0}
pred = {a: e, b: c, c: d, d: b, e: None}

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

CON CICLO NEGATIVO



(e,c), (c,b), (b,c), (a,b), (a,d), (d,e), (b,e), (d,c), (b,d), (b,c), (e,a)

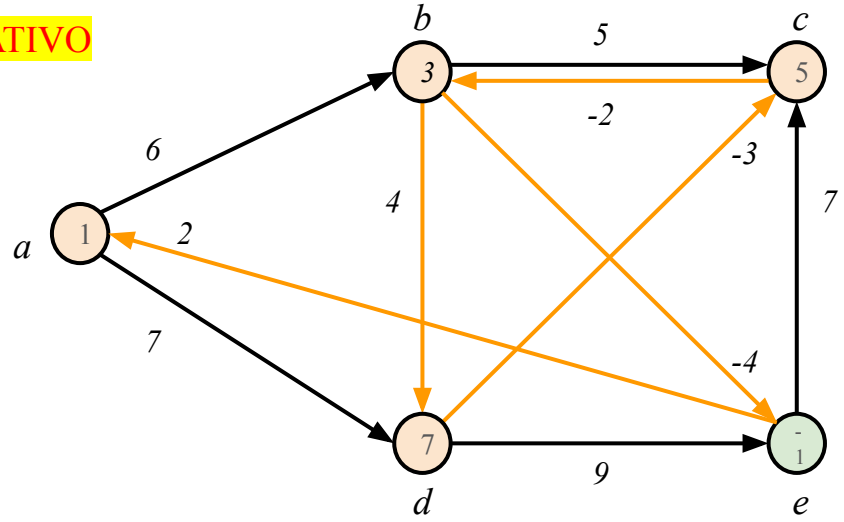
i = 4
d = {a: 2, b: 4, c: 5, d: 8, e: 0}
pred = {a: e, b: c, c: d, d: b, e: None}

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

CON CICLO NEGATIVO



(e,c), (c,b), (b,c), (a,b), (a,d), (d,e), (b,e), (d,c), (b,d), (b,c), (e,a)

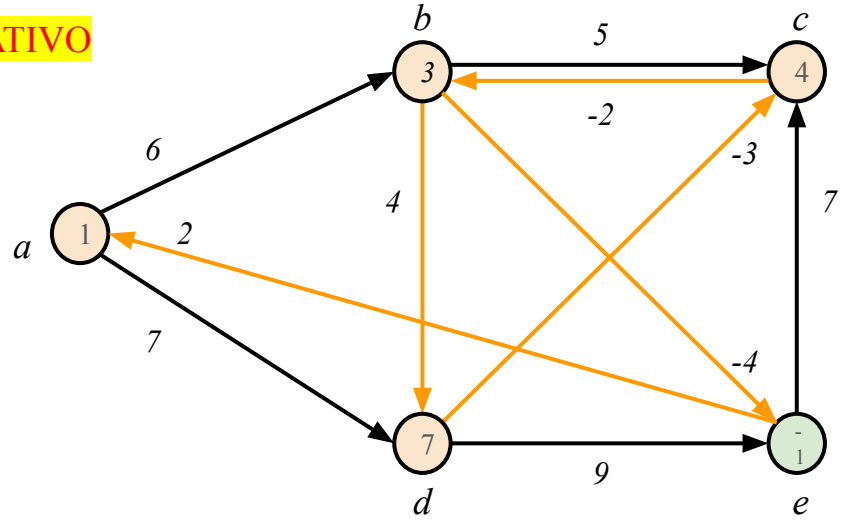
i = 5
d = {a: 1, b: 3, c: 5, d: 7, e: -1}
pred = {a: e, b: c, c: d, d: b, e: b}

Bellman-Ford

```
BELLMAN-FORD ( G, s ) :
|     INIT(G, s)
|
|     for i = 1 to n-1:
|     |     for (u,v) in E:
|     |     |     RELAX(u, v)
|
|     for (u,v) in E:
|     |     if v.d < u.d + w(u,v):
|     |     |     return False
|
|     return True, v
```

```
RELAX ( u , v, w ) :
|     if v.d > u.d + w(u,v):
|     |     v.d = u.d + w(u,v)
|     |     v.pred = u
```

CON CICLO NEGATIVO



(e,c), (c,b), (b,c), (a,b), (a,d), (d,e), (b,e), (d,c), (b,d), (b,c), (e,a)

d = {a: 1, b: 3, **c: 4**, d: 7, e: -1}
pred = {a: e, b: c, c: d, d: b, e: b}

Programación lineal

En general, dados: *A: m x m*, *b: n x m*, *c: n x 1*. Se quiere encontrar *x: n x 1* tq

MAXIMICE la FUNCIÓN OBJETIVO: $\Sigma_i c_i x_i$

con las RESTRICCIONES: $Ax \leq b$

Se conocen algoritmos polinomiales como SIMPLEX para resolverlo si se sabe que la solución existe

Programación lineal

En general, dados: *A: m x m*, *b: n x m*, *c: n x 1*. Se quiere encontrar *x: n x 1* tq

MAXIMICE la FUNCIÓN OBJETIVO: $\Sigma_i c_i x_i$

con las RESTRICCIONES: $Ax \leq b$

Se conocen algoritmos polinomiales como SIMPLEX para resolverlo si se sabe que la solución existe o para algunos casos particulares.

Programación lineal

En general, dados: *A: m x m*, *b: n x m*, *c: n x 1*. Se quiere encontrar *x: n x 1* tq

MAXIMICE la FUNCIÓN OBJETIVO: $\Sigma_i c_i x_i$

con las RESTRICCIONES: $Ax \leq b$

Se conocen algoritmos polinomiales como SIMPLEX para resolverlo si se sabe que la solución existe o para algunos casos particulares.

PERO si queremos saber si la solución existe

Programación lineal

En general, dados: *A: m x m*, *b: n x m*, *c: n x 1*. Se quiere encontrar *x: n x 1* tq

MAXIMICE la FUNCIÓN OBJETIVO: $\Sigma_i c_i x_i$

con las RESTRICCIONES: $Ax \leq b$

Se conocen algoritmos polinomiales como SIMPLEX para resolverlo si se sabe que la solución existe o para algunos casos particulares.

PERO si queremos saber si la solución existe, y no nos interesa la función objetivo

⇒ **BELLMAN - FORD**

Sistemas de diferencias

$$A \quad x \quad \leq \quad b$$

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \leq \begin{bmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 - x_2 \leq 0 \\ x_1 - x_5 \leq -1 \\ x_2 - x_5 \leq 1 \\ x_3 - x_1 \leq 5 \\ x_4 - x_1 \leq 4 \\ x_4 - x_3 \leq -1 \\ x_5 - x_3 \leq -3 \\ x_5 - x_4 \leq -3 \end{bmatrix}$$

Sistemas de diferencias

$$A \quad x \quad \leq \quad b$$

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \leq \begin{bmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{bmatrix}$$

sol.: x = (-5, -3, 0, -1, -4)

sol.: x' = (0, 2, 5, 4, 1)

Lema: Si x es solución de $Ax \leq b$

$\Rightarrow x+d$, $d = cte$ también es solución de $Ax \leq b$

Demo: $x_2 - x_1 \leq b$

$(x_2 + d) - (x_1 + d) \leq b$

$x_2 - x_1 \leq b$

Sistemas de diferencias

$$A \quad x \quad \leq \quad b$$

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \leq \begin{bmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{bmatrix}$$

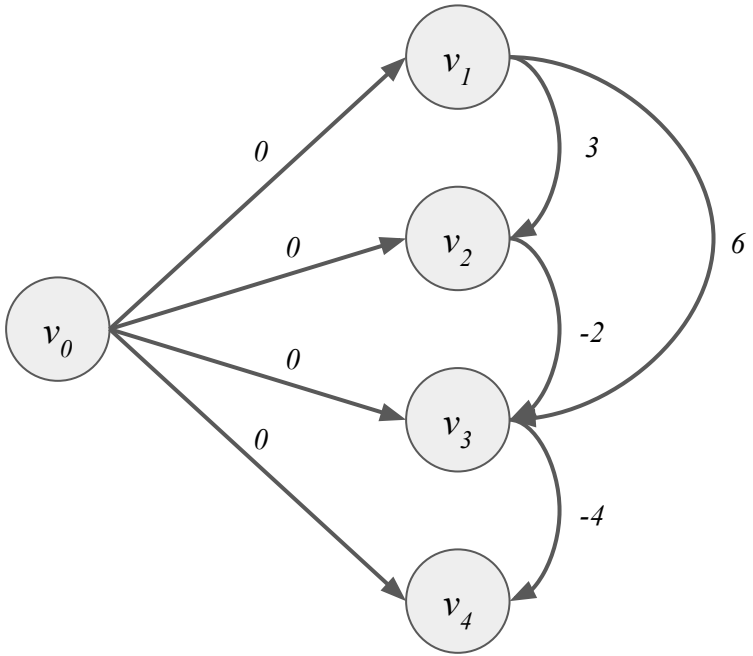
Apps:

Siguiendo con la organización de tareas… esto puede servir para agregarle restricciones de tiempo.

Sistemas de diferencias

$$A \quad x \quad \leq \quad b$$

$$\begin{bmatrix} x_2 - x_1 \leq 3 \\ x_3 - x_2 \leq -2 \\ x_3 - x_1 \leq 6 \\ x_4 - x_3 \leq -4 \end{bmatrix} \Rightarrow$$



$V = \{v_0, v_1, v_2, v_3, v_4\}$

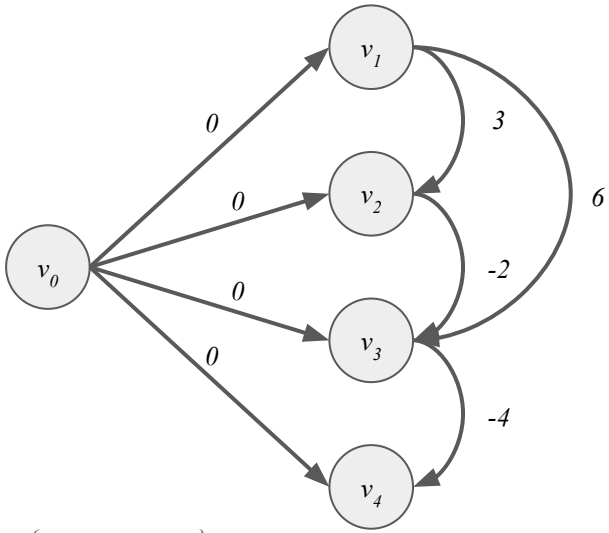
$E = \{(v_0, v_1), (v_0, v_2), (v_0, v_3), (v_0, v_4), (v_1, v_2), (v_2, v_3), (v_3, v_4), (v_1, v_3)\}$

$w(v_i, v_j) = b_k$

$w(v_0, v_i) = 0$

Sistemas de diferencias

$$A \quad x \quad \leq \quad b$$



$V = \{v_0, v_1, v_2, v_3, v_4\}$,

$E = \{(v_i, v_j): x_j - x_i \leq b_k\} \cup \{(v_0, v_1), \ldots, (v_0, v_k)\}$

$w(v_i, v_j) = b_k$, $w(v_0, v_i) = 0$

Teorema:

1) Si G no contiene ciclo negativos $\Rightarrow$

$x = (\delta(v_0,v_1), \delta(v_0,v_2), \ldots, \delta(v_0,v_k))$ es una sol. posible

2) Si G contiene ciclo negativos $\Rightarrow$

no existe solución

Demo:

1) (Desig. triang.) $\Rightarrow$

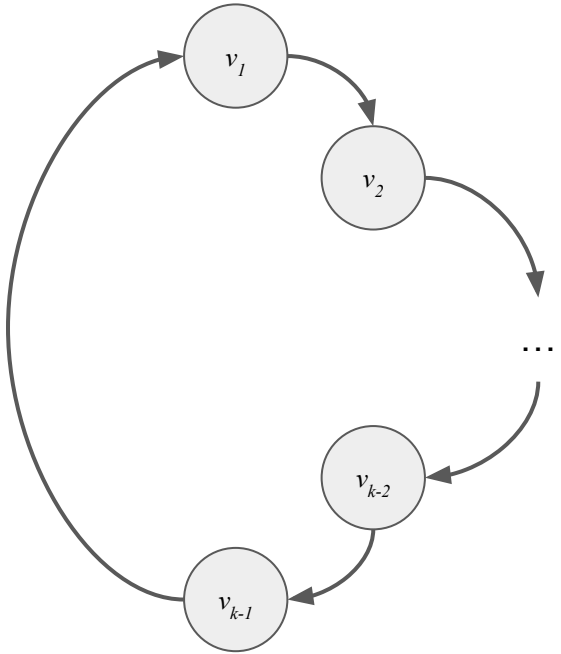
$$\delta(v_0,v_j) \leq \delta(v_0,v_i) + w(v_i,v_j)$$

$$\delta(v_0,v_j) - \delta(v_0,v_i) \leq w(v_i,v_j)$$

$$x_j - x_i \leq b_k$$

Sistemas de diferencias

$$A \quad x \quad \leq \quad b$$



Teorema: 2) Si G contiene ciclo negativos $\Rightarrow$ no existe solución

Demo:

2) (x Absurdo)

Supongo que existe un ciclo neg. $c = v_1, \ldots, v_{k-1}, v_k$ $(v_1 = v_k)$

$$\Rightarrow \quad x_2 - x_1 \leq w(v_2, v_1)$$

$$x_3 - x_2 \leq w(v_3, v_2)$$

$$\ldots$$

$$x_{k-1} - x_{k-2} \leq w(v_{k-1}, v_{k-2})$$

$$x_k - x_{k-1} \leq w(v_k, v_{k-1})$$

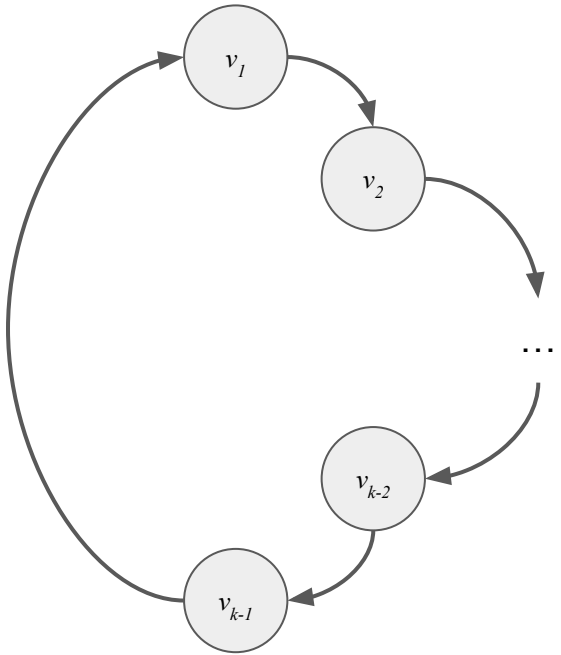
$$x_k - x_{k-1} + x_{k-1} - x_{k-2} + \ldots + x_3 - x_2 + x_2 - x_1 \leq \sum^{k-1} w(v_i, v_{i+1})$$

$$x_k - x_1 \leq \sum^{k-1} w(v_i, v_{i+1})$$

$$(x_k = x_1) \quad 0 \leq \sum^{k-1} w(v_i, v_{i+1}) = w(c)$$

Sistemas de diferencias

$$A \quad x \quad \leq \quad b$$



Teorema: 2) Si *G* contiene ciclo negativos ⇒ no existe solución

Demo:

2) (x Absurdo)

Supongo que existe un ciclo neg. $c = v_1, \ldots, v_{k-1}, v_k$ $(v_1 = v_k)$

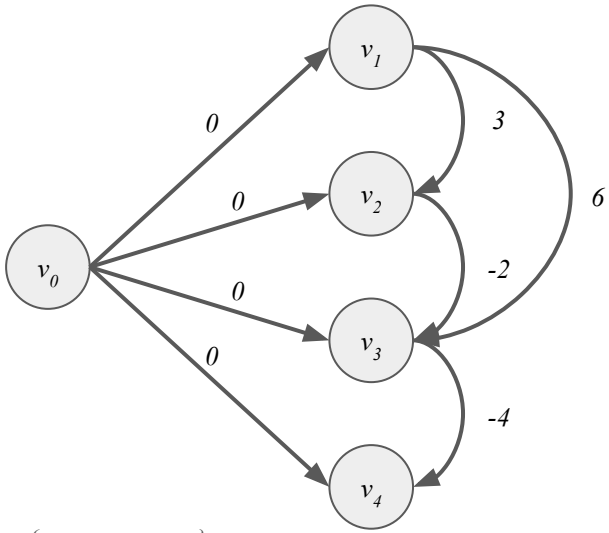
$\Rightarrow \quad 0 \leq \sum^{k-1} w(v_i, v_{i+1}) = w(c)$

$0 \leq w(c) < 0$ *(ciclo negativo)*

¡Absurdo!

Sistemas de diferencias

$$A \quad x \quad \leq \quad b$$



$V = \{v_0, v_1, v_2, v_3, v_4\}$,

$E = \{(v_i, v_j): x_j - x_i \leq b_k\} \cup \{(v_0, v_1), \ldots, (v_0, v_k)\}$

$w(v_i, v_j) = b_k$, $w(v_0, v_i) = 0$

En general, dadas *A: m x m*, *b: n x m*.

Se quiere encontrar *x: n x 1* tq cumpla las RESTRICCIONES Ax≤b

$O(V*E) =$

$O((n + 1)*(m + n))$

$O(n*m + n^2)$

denso ($m \sim n^2$): $O(n^3)$

ralo ($m \sim n$): $O(n^2)$