

Camino mínimo en grafos

Técnicas de Diseño de Algoritmos (Ex Algoritmos y Estructuras de Datos III)

Primer cuatrimestre 2024

Camino mínimo en grafos (orientados o no)

Sea $G = (V, X)$ un grafo y $l : X \rightarrow R$ una función de longitud/peso para las aristas de G .

Camino mínimo en grafos (orientados o no)

Sea $G = (V, X)$ un grafo y $l : X \rightarrow R$ una función de longitud/peso para las aristas de G .

Definiciones:

- La **longitud** de un recorrido R entre dos nodos v y u es la suma de las longitudes de las aristas del R :

$$l(R) = \sum_{e \in R} l(e)$$

Camino mínimo en grafos (orientados o no)

Sea $G = (V, X)$ un grafo y $l : X \rightarrow R$ una función de longitud/peso para las aristas de G .

Definiciones:

- ▶ La **longitud** de un recorrido R entre dos nodos v y u es la suma de las longitudes de las aristas del R :

$$l(R) = \sum_{e \in R} l(e)$$

- ▶ Un **recorrido mínimo** R^0 entre u y v es un recorrido entre u y v tal que $l(R^0) = \min\{l(R) \mid R \text{ es un recorrido entre } u \text{ y } v\}$.

Camino mínimo en grafos (orientados o no)

Sea $G = (V, X)$ un grafo y $l : X \rightarrow R$ una función de longitud/peso para las aristas de G .

Definiciones:

- ▶ La **longitud** de un recorrido R entre dos nodos v y u es la suma de las longitudes de las aristas del R :

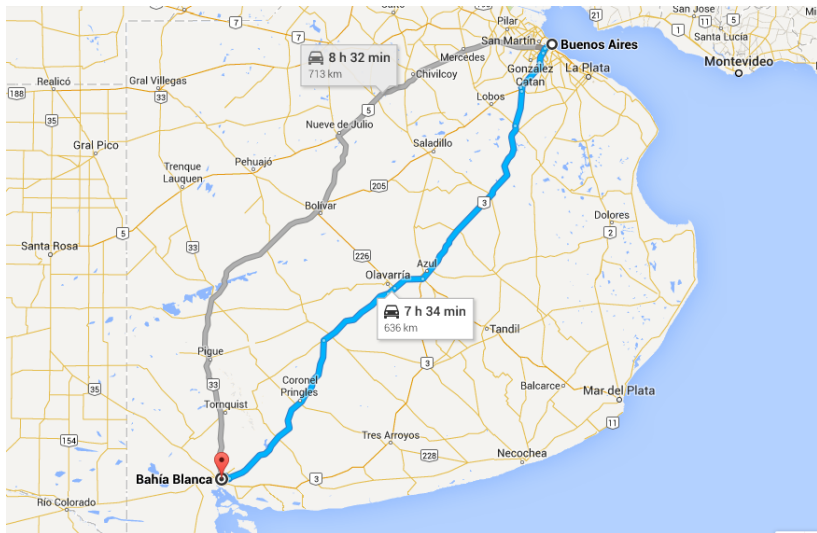
$$l(R) = \sum_{e \in R} l(e)$$

- ▶ Un **recorrido mínimo** R^0 entre u y v es un recorrido entre u y v tal que $l(R^0) = \min\{l(R) \mid R \text{ es un recorrido entre } u \text{ y } v\}$.
- ▶ Si un recorrido mínimo entre un par de nodos es un camino entonces se lo llama como **camino mínimo** entre ese par de nodos. La existencia de recorridos mínimos es equivalente a la existencia de caminos mínimos. Puede haber varios caminos mínimos.

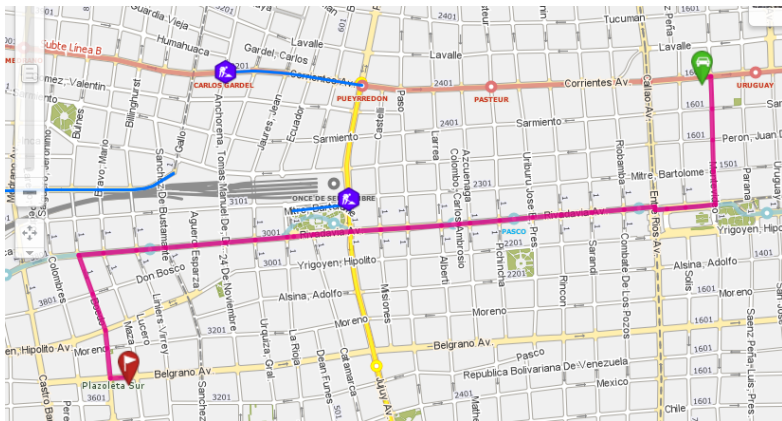
Camino mínimo en grafos (orientados o no)

- La **distancia** entre u y v , $dist(u, v)$, es la longitud de un camino mínimo entre u y v en caso de existir algún camino entre u y v y en caso contrario sería ∞ .

Camino mínimo en grafos (orientados o no)



Camino mínimo en grafos (orientados o no)



Camino mínimo en grafos (orientados o no)

Dado un grafo G , se pueden definir tres variantes de problemas sobre caminos mínimos:

Camino mínimo en grafos (orientados o no)

Dado un grafo G , se pueden definir tres variantes de problemas sobre caminos mínimos:

Único origen - único destino: Determinar un camino mínimo entre dos vértices específicos, v y u .

Camino mínimo en grafos (orientados o no)

Dado un grafo G , se pueden definir tres variantes de problemas sobre caminos mínimos:

Único origen - único destino: Determinar un camino mínimo entre dos vértices específicos, v y u .

Único origen - múltiples destinos: Determinar un camino mínimo desde un vértice específico v al resto de los vértices de G .

Camino mínimo en grafos (orientados o no)

Dado un grafo G , se pueden definir tres variantes de problemas sobre caminos mínimos:

Único origen - único destino: Determinar un camino mínimo entre dos vértices específicos, v y u .

Único origen - múltiples destinos: Determinar un camino mínimo desde un vértice específico v al resto de los vértices de G .

Múltiples orígenes - múltiples destinos: Determinar un camino mínimo entre todo par de vértices de G .

Camino mínimo en grafos (orientados o no)

Dado un grafo G , se pueden definir tres variantes de problemas sobre caminos mínimos:

Único origen - único destino: Determinar un camino mínimo entre dos vértices específicos, v y u .

Único origen - múltiples destinos: Determinar un camino mínimo desde un vértice específico v al resto de los vértices de G .

Múltiples orígenes - múltiples destinos: Determinar un camino mínimo entre todo par de vértices de G .

Todos estos conceptos se pueden adaptar cuando se trabaja con un grafo orientado.

Camino mínimo en grafos (orientados o no)

- ▶ **Aristas con peso negativo:** Si el grafo G no contiene ciclos de peso negativo o contiene alguno pero no es alcanzable desde v , entonces el problema sigue estando bien definido, aunque algunos caminos puedan tener longitud negativa. Sin embargo, si G tiene algún ciclo con peso negativo alcanzable desde v , el concepto de recorrido de peso mínimo deja de estar bien definido.

Camino mínimo en grafos (orientados o no)

- ▶ **Aristas con peso negativo:** Si el grafo G no contiene ciclos de peso negativo o contiene alguno pero no es alcanzable desde v , entonces el problema sigue estando bien definido, aunque algunos caminos puedan tener longitud negativa. Sin embargo, si G tiene algún ciclo con peso negativo alcanzable desde v , el concepto de recorrido de peso mínimo deja de estar bien definido.
- ▶ **Ciclos:** La existencia de un recorrido que minimiza entre los recorridos sin ciclos. Este problema sí está bien definido para cualquier circunstancia.

Camino mínimo en grafos (orientados o no)

- ▶ **Aristas con peso negativo:** Si el grafo G no contiene ciclos de peso negativo o contiene alguno pero no es alcanzable desde v , entonces el problema sigue estando bien definido, aunque algunos caminos puedan tener longitud negativa. Sin embargo, si G tiene algún ciclo con peso negativo alcanzable desde v , el concepto de recorrido de peso mínimo deja de estar bien definido.
- ▶ **Ciclos:** La existencia de un recorrido que minimiza entre los recorridos sin ciclos. Este problema sí está bien definido para cualquier circunstancia.
- ▶ **Propiedad de subestructura óptima de un camino mínimo:** Dado un digrafo $G = (V, X)$ con una función de peso $l : X \rightarrow R$, sea $P : v_1 \dots v_k$ un camino mínimo de v_1 a v_k . Entonces $\forall 1 \leq i \leq j \leq k$, $P_{v_i v_j}$ es un camino mínimo desde v_i a v_j .

Camino mínimo en grafos (orientados o no)

- ▶ **Aristas con peso negativo:** Si el grafo G no contiene ciclos de peso negativo o contiene alguno pero no es alcanzable desde v , entonces el problema sigue estando bien definido, aunque algunos caminos puedan tener longitud negativa. Sin embargo, si G tiene algún ciclo con peso negativo alcanzable desde v , el concepto de recorrido de peso mínimo deja de estar bien definido.
- ▶ **Ciclos:** La existencia de un recorrido que minimiza entre los recorridos sin ciclos. Este problema sí está bien definido para cualquier circunstancia.
- ▶ **Propiedad de subestructura óptima de un camino mínimo:** Dado un digrafo $G = (V, X)$ con una función de peso $l : X \rightarrow R$, sea $P : v_1 \dots v_k$ un camino mínimo de v_1 a v_k . Entonces $\forall 1 \leq i \leq j \leq k$, $P_{v_i v_j}$ es un camino mínimo desde v_i a v_j . ¿Cómo se puede probar esta propiedad?

Camino mínimo en grafos - Único origen-múltiples destinos

Problema: Dado $G = (V, X)$ un grafo y $l : X \rightarrow R$ una función que asigna a cada arco una cierta longitud y $v \in V$ un nodo del grafo, calcular los caminos mínimos de v al resto de los nodos.

Camino mínimo en grafos - Único origen-múltiples destinos

Problema: Dado $G = (V, X)$ un grafo y $l : X \rightarrow R$ una función que asigna a cada arco una cierta longitud y $v \in V$ un nodo del grafo, calcular los caminos mínimos de v al resto de los nodos.

Distintas situaciones:

- ▶ El grafo puede ser orientado o no.
- ▶ Todos los arcos tienen longitud no negativa.
- ▶ No hay un circuito orientado de longitud negativa.
- ▶ Hay circuitos orientados de longitud negativa.
- ▶ Queremos calcular los caminos mínimos entre todos los pares de nodos.

Algoritmo de Dijkstra (1959)



Edsger Dijkstra (1930–2002)

`www.cs.utexas.edu/users/EWD`

Algoritmo de Dijkstra (1959)

Asumimos que las longitudes de las aristas son no negativas. El grafo puede ser orientado o no orientado.

```
 $S := \{v\}, \pi(v) := 0$   
para todo  $u \in V$  hacer  
    si  $(v, u) \in X$  entonces  
         $\pi(u) := l(v, u)$   
    si no  
         $\pi(u) := \infty$   
    fin si  
fin para  
mientras  $S \neq V$  hacer  
    elegir  $w \in V \setminus S$  tal que  $\pi(w) = \min_{u \in V \setminus S} \pi(u)$   
     $S := S \cup \{w\}$   
    para todo  $u \in V \setminus S$  y  $(w, u) \in X$  hacer  
        si  $\pi(u) > \pi(w) + l(w, u)$  entonces  
             $\pi(u) := \pi(w) + l(w, u)$   
        fin si  
    fin para  
fin mientras  
retornar  $\pi$ 
```

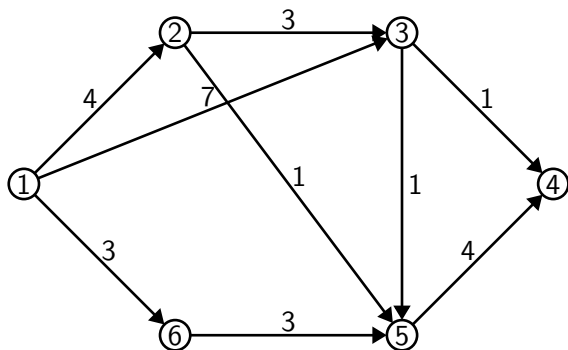
Algoritmo de Dijkstra (1959) - Determina camino mínimo

```
 $S := \{v\}, \pi(v) := 0, \text{pred}(v) := 0$   
para todo  $u \in V$  hacer  
    si  $(v, u) \in X$  entonces  
         $\pi(u) := l(v, u), \text{pred}(u) := v$   
    si no  
         $\pi(u) := \infty, \text{pred}(u) := \infty$   
    fin si  
fin para  
mientras  $S \neq V$  hacer  
    elegir  $w \in V \setminus S$  tal que  $\pi(w) = \min_{u \in V \setminus S} \pi(u)$   
     $S := S \cup w$   
    para todo  $u \in V \setminus S$  y  $(w, u) \in X$  hacer  
        si  $\pi(u) > \pi(w) + l(w, u)$  entonces  
             $\pi(u) := \pi(w) + l(w, u)$   
             $\text{pred}(u) := w$   
        fin si  
    fin para  
fin mientras  
retornar  $\pi, \text{pred}$ 
```

Algoritmo de Dijkstra (1959) - Determina camino mínimo

```
 $S := \{v\}, \pi(v) := 0, \text{pred}(v) := 0, w := v$   
para todo  $u \in V$  hacer  
    si  $(v, u) \in X$  entonces  
         $\pi(u) := l(v, u), \text{pred}(u) := v$   
    si no  
         $\pi(u) := \infty, \text{pred}(u) := \infty$   
    fin si  
fin para  
mientras  $S \neq V$  y  $\pi(w) < \infty$  hacer  
    elegir  $w \in V \setminus S$  tal que  $\pi(w) = \min_{u \in V \setminus S} \pi(u)$   
     $S := S \cup w$   
    para todo  $u \in V \setminus S$  y  $(w, u) \in X$  hacer  
        si  $\pi(u) > \pi(w) + l(w, u)$  entonces  
             $\pi(u) := \pi(w) + l(w, u)$   
             $\text{pred}(u) := w$   
        fin si  
    fin para  
fin mientras  
retornar  $\pi, \text{pred}$ 
```

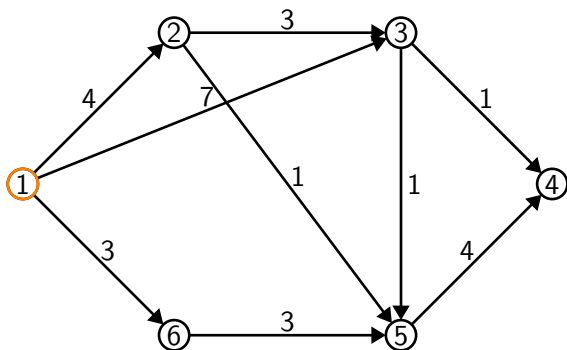
Algoritmo de Dijkstra - Ejemplo



Algoritmo de Dijkstra - Ejemplo

$$S = \{1\}$$

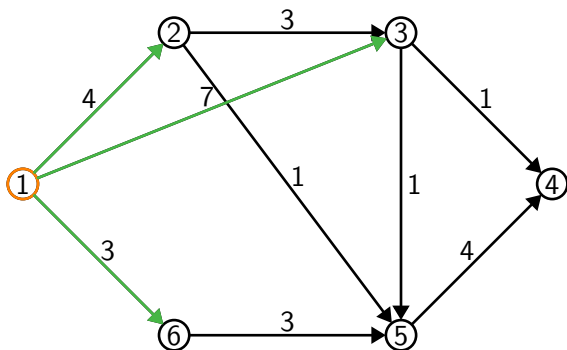
$$\pi = (0, ?, ?, ?, ?, ?)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1\}$$

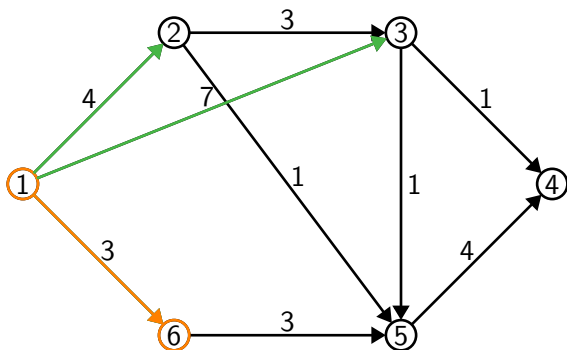
$$\pi = (0, 4, 7, \infty, \infty, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6\}$$

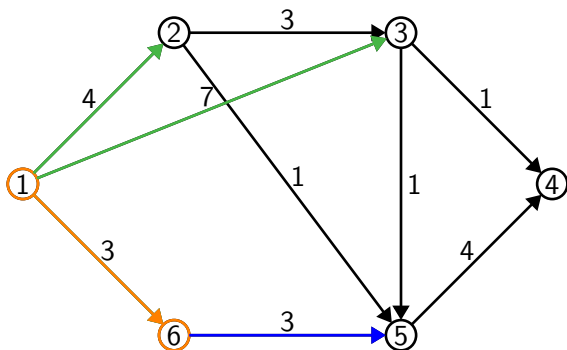
$$\pi = (0, 4, 7, \infty, \infty, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6\}$$

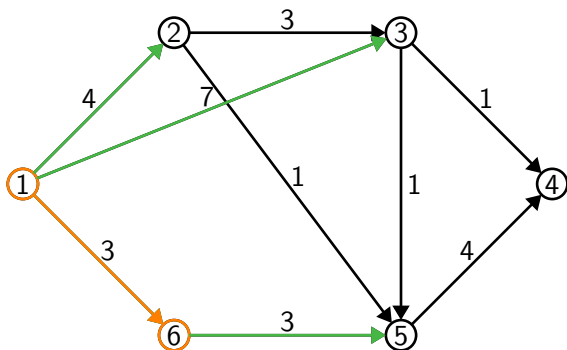
$$\pi = (0, 4, 7, \infty, \infty, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6\}$$

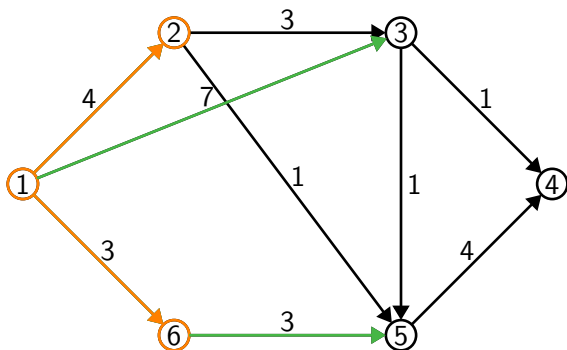
$$\pi = (0, 4, 7, \infty, 6, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6, 2\}$$

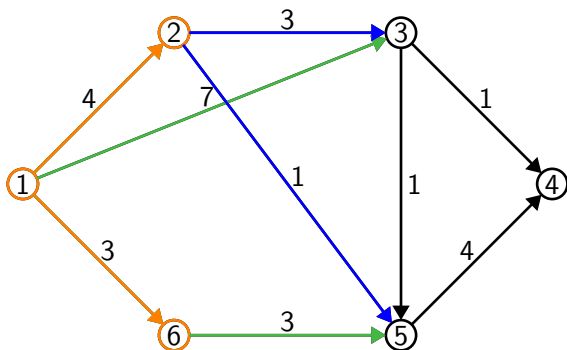
$$\pi = (0, 4, 7, \infty, 6, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6, 2\}$$

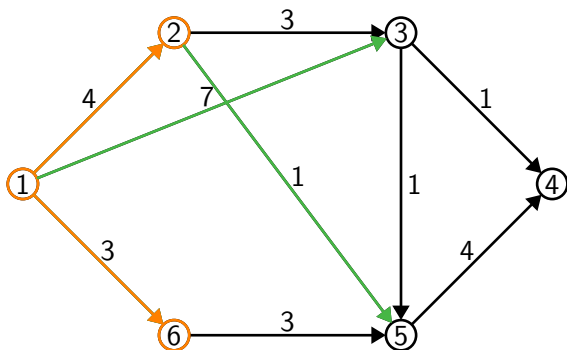
$$\pi = (0, 4, 7, \infty, 6, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6, 2\}$$

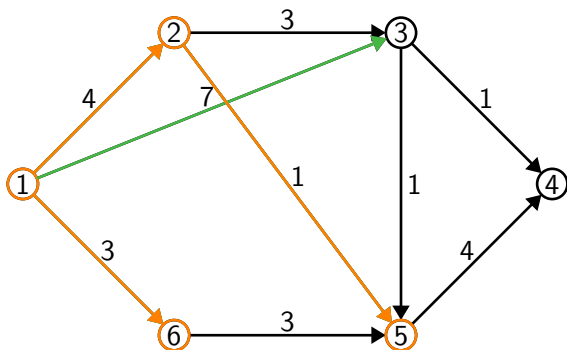
$$\pi = (0, 4, 7, \infty, 5, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6, 2, 5\}$$

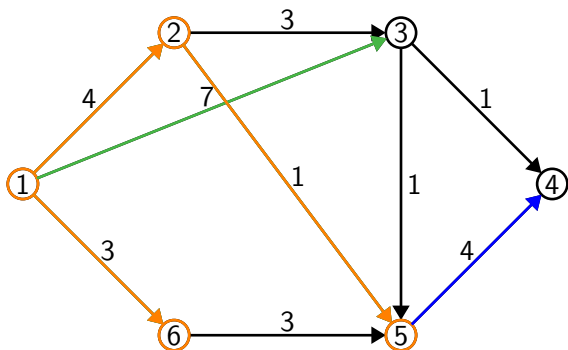
$$\pi = (0, 4, 7, \infty, 5, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6, 2, 5\}$$

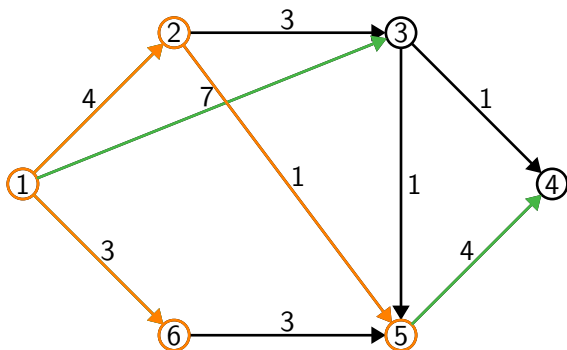
$$\pi = (0, 4, 7, \infty, 5, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6, 2, 5\}$$

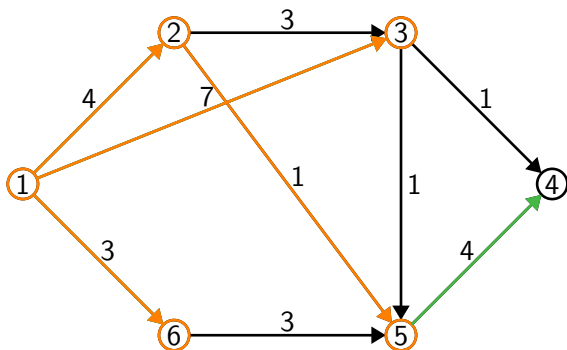
$$\pi = (0, 4, 7, 9, 5, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6, 2, 5, 3\}$$

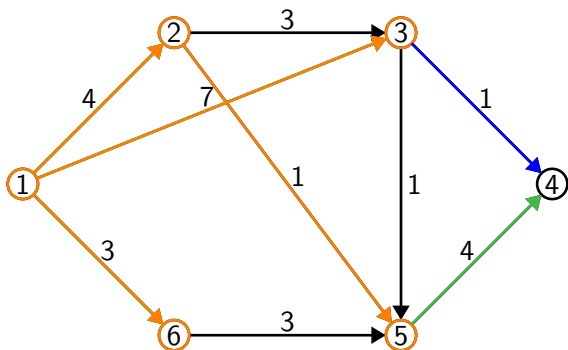
$$\pi = (0, 4, 7, 9, 5, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6, 2, 5, 3\}$$

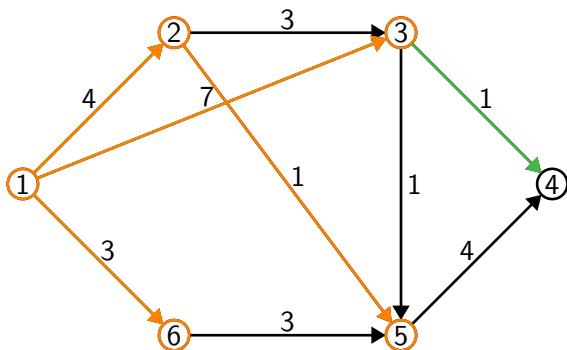
$$\pi = (0, 4, 7, 9, 5, 3)$$



Algoritmo de Dijkstra - Ejemplo

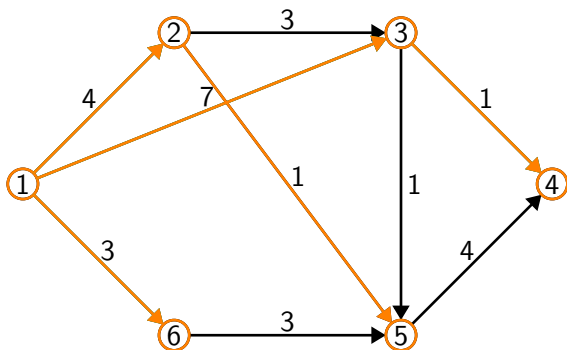
$$S = \{1, 6, 2, 5, 3\}$$

$$\pi = (0, 4, 7, 8, 5, 3)$$

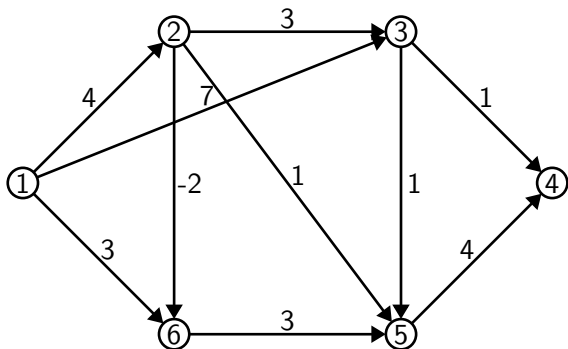


Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6, 2, 5, 3, 4\} \quad \pi = (0, 4, 7, 8, 5, 3)$$



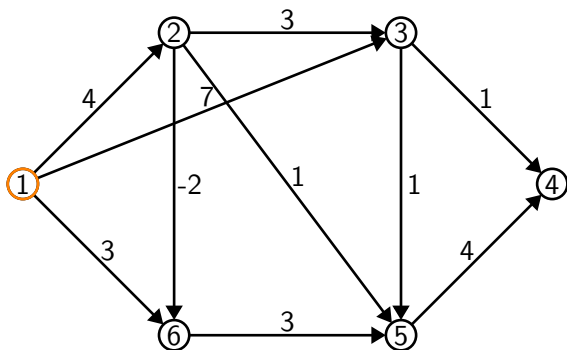
Algoritmo de Dijkstra - Ejemplo



Algoritmo de Dijkstra - Ejemplo

$$S = \{1\}$$

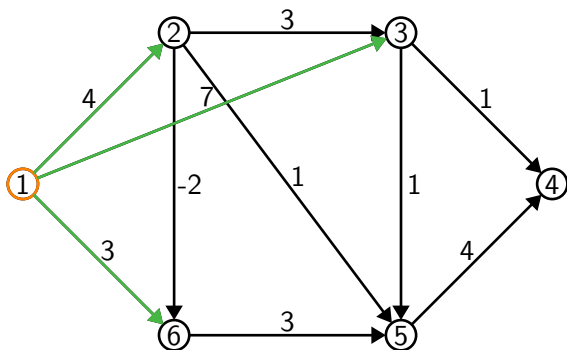
$$\pi = (0, ?, ?, ?, ?, ?)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1\}$$

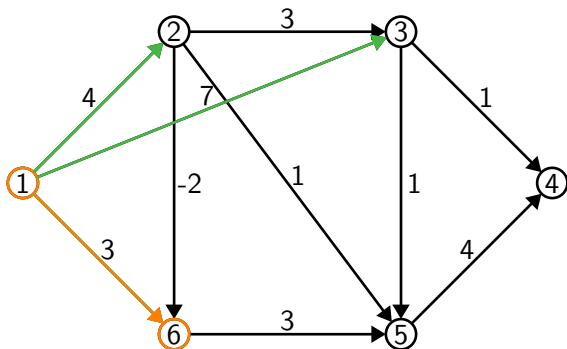
$$\pi = (0, 4, 7, \infty, \infty, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6\}$$

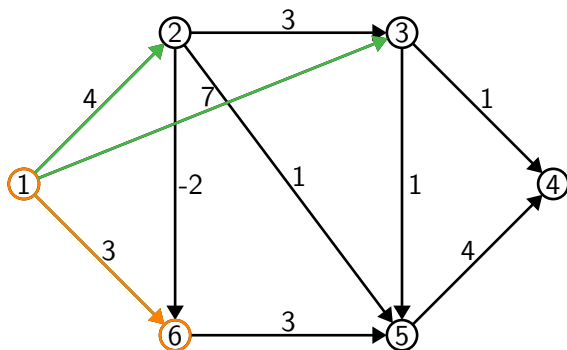
$$\pi = (0, 4, 7, \infty, \infty, 3)$$



Algoritmo de Dijkstra - Ejemplo

$$S = \{1, 6\}$$

$$\pi = (0, 4, 7, \infty, \infty, 3)$$



¡Ya no actualizará $\pi(6)$!

Algoritmo de Dijkstra

Lema: Dado un grafo orientado G con pesos no negativos en las aristas, al finalizar la iteración k el algoritmo de Dijkstra determina el camino mínimo entre el nodo v y los nodos de S_k (donde S_k conjunto S al finalizar la iteración k).

Algoritmo de Dijkstra

Lema: Dado un grafo orientado G con pesos no negativos en las aristas, al finalizar la iteración k el algoritmo de Dijkstra determina el camino mínimo entre el nodo v y los nodos de S_k (donde S_k conjunto S al finalizar la iteración k).

Teorema: Dado un grafo orientado G con pesos no negativos en las aristas, el algoritmo de Dijkstra determina el camino mínimo entre el nodo v y el resto de los nodos de G .

Prueba del lema

Llamamos w_k , el vértice agregado a S en la iteración $k \geq 1$, es decir que $S_k = S_{k-1} \cup \{w_k\}$, $v = w_0$ y $S_0 = \{w_0\}$.

Prueba del lema

Llamamos w_k , el vértice agregado a S en la iteración $k \geq 1$, es decir que $S_k = S_{k-1} \cup \{w_k\}$, $v = w_0$ y $S_0 = \{w_0\}$.

En primer lugar, vamos a probar que si $\pi(w_k) = \infty$ entonces $\text{dist}(v, w_k) = \infty$.

Prueba del lema

Llamamos w_k , el vértice agregado a S en la iteración $k \geq 1$, es decir que $S_k = S_{k-1} \cup \{w_k\}$, $v = w_0$ y $S_0 = \{w_0\}$.

En primer lugar, vamos a probar que si $\pi(w_k) = \infty$ entonces $\text{dist}(v, w_k) = \infty$. Sea $k' \leq k$, la primera iteración donde el nodo agregado de esa iteración, $w_{k'}$ con $\pi(w_{k'}) = \infty$.

Prueba del lema

Llamamos w_k , el vértice agregado a S en la iteración $k \geq 1$, es decir que $S_k = S_{k-1} \cup \{w_k\}$, $v = w_0$ y $S_0 = \{w_0\}$.

En primer lugar, vamos a probar que si $\pi(w_k) = \infty$ entonces $\text{dist}(v, w_k) = \infty$. Sea $k' \leq k$, la primera iteración donde el nodo agregado de esa iteración, $w_{k'}$ con $\pi(w_{k'}) = \infty$. Claramente, para cualquier vértice $u \notin S_{k'-1}$ en el momento de ejecutar la iteración k' , $\pi(u) = \infty$ ya que $\pi(w_{k'}) = \min_{u \in V \setminus S_{k'-1}} \pi(u) = \infty$.

Prueba del lema

Llamamos w_k , el vértice agregado a S en la iteración $k \geq 1$, es decir que $S_k = S_{k-1} \cup \{w_k\}$, $v = w_0$ y $S_0 = \{w_0\}$.

En primer lugar, vamos a probar que si $\pi(w_k) = \infty$ entonces $\text{dist}(v, w_k) = \infty$. Sea $k' \leq k$, la primera iteración donde el nodo agregado de esa iteración, $w_{k'}$ con $\pi(w_{k'}) = \infty$. Claramente, para cualquier vértice $u \notin S_{k'-1}$ en el momento de ejecutar la iteración k' , $\pi(u) = \infty$ ya que $\pi(w_{k'}) = \min_{u \in V \setminus S_{k'-1}} \pi(u) = \infty$. No existe arista que sale de un nodo $w_p \in S_{k'-1}$ con $p \leq k' - 1$ y llega a un nodo $u \notin S_{k'-1}$ pues si existiese (w_p, u) , $\pi(u) \leq \pi(w_p) + l(w_p, u) < \infty$, una vez que termina la iteración p .

Prueba del lema

Llamamos w_k , el vértice agregado a S en la iteración $k \geq 1$, es decir que $S_k = S_{k-1} \cup \{w_k\}$, $v = w_0$ y $S_0 = \{w_0\}$.

En primer lugar, vamos a probar que si $\pi(w_k) = \infty$ entonces $\text{dist}(v, w_k) = \infty$. Sea $k' \leq k$, la primera iteración donde el nodo agregado de esa iteración, $w_{k'}$ con $\pi(w_{k'}) = \infty$. Claramente, para cualquier vértice $u \notin S_{k'-1}$ en el momento de ejecutar la iteración k' , $\pi(u) = \infty$ ya que $\pi(w_{k'}) = \min_{u \in V \setminus S_{k'-1}} \pi(u) = \infty$. No existe arista que sale de un nodo $w_p \in S_{k'-1}$ con $p \leq k' - 1$ y llega a un nodo $u \notin S_{k'-1}$ pues si existiese (w_p, u) , $\pi(u) \leq \pi(w_p) + l(w_p, u) < \infty$, una vez que termina la iteración p . Lo cual es una contradicción, ya que no se puede aumentar el valor de $\pi(u)$ a lo largo de la ejecución del algoritmo.

Prueba del lema

Llamamos w_k , el vértice agregado a S en la iteración $k \geq 1$, es decir que $S_k = S_{k-1} \cup \{w_k\}$, $v = w_0$ y $S_0 = \{w_0\}$.

En primer lugar, vamos a probar que si $\pi(w_k) = \infty$ entonces $\text{dist}(v, w_k) = \infty$. Sea $k' \leq k$, la primera iteración donde el nodo agregado de esa iteración, $w_{k'}$ con $\pi(w_{k'}) = \infty$. Claramente, para cualquier vértice $u \notin S_{k'-1}$ en el momento de ejecutar la iteración k' , $\pi(u) = \infty$ ya que $\pi(w_{k'}) = \min_{u \in V \setminus S_{k'-1}} \pi(u) = \infty$. No existe arista que sale de un nodo $w_p \in S_{k'-1}$ con $p \leq k' - 1$ y llega a un nodo $u \notin S_{k'-1}$ pues si existiese (w_p, u) , $\pi(u) \leq \pi(w_p) + l(w_p, u) < \infty$, una vez que termina la iteración p . Lo cual es una contradicción, ya que no se puede aumentar el valor de $\pi(u)$ a lo largo de la ejecución del algoritmo. Por lo tanto, no hay camino que sale de un nodo de $S_{k'-1}$ y termina en un nodo fuera de $S_{k'-1}$ y particularmente para v y w_k .

Prueba del lema

Llamamos w_k , el vértice agregado a S en la iteración $k \geq 1$, es decir que $S_k = S_{k-1} \cup \{w_k\}$, $v = w_0$ y $S_0 = \{w_0\}$.

En primer lugar, vamos a probar que si $\pi(w_k) = \infty$ entonces $\text{dist}(v, w_k) = \infty$. Sea $k' \leq k$, la primera iteración donde el nodo agregado de esa iteración, $w_{k'}$ con $\pi(w_{k'}) = \infty$. Claramente, para cualquier vértice $u \notin S_{k'-1}$ en el momento de ejecutar la iteración k' , $\pi(u) = \infty$ ya que $\pi(w_{k'}) = \min_{u \in V \setminus S_{k'-1}} \pi(u) = \infty$. No existe arista que sale de un nodo $w_p \in S_{k'-1}$ con $p \leq k' - 1$ y llega a un nodo $u \notin S_{k'-1}$ pues si existiese (w_p, u) , $\pi(u) \leq \pi(w_p) + l(w_p, u) < \infty$, una vez que termina la iteración p . Lo cual es una contradicción, ya que no se puede aumentar el valor de $\pi(u)$ a lo largo de la ejecución del algoritmo. Por lo tanto, no hay camino que sale de un nodo de $S_{k'-1}$ y termina en un nodo fuera de $S_{k'-1}$ y particularmente para v y w_k . Consecuentemente, $\text{dist}(v, w_k) = \infty$.

Prueba del lema

Llamamos w_k , el vértice agregado a S en la iteración $k \geq 1$, es decir que $S_k = S_{k-1} \cup \{w_k\}$, $v = w_0$ y $S_0 = \{w_0\}$.

En primer lugar, vamos a probar que si $\pi(w_k) = \infty$ entonces $\text{dist}(v, w_k) = \infty$. Sea $k' \leq k$, la primera iteración donde el nodo agregado de esa iteración, $w_{k'}$ con $\pi(w_{k'}) = \infty$. Claramente, para cualquier vértice $u \notin S_{k'-1}$ en el momento de ejecutar la iteración k' , $\pi(u) = \infty$ ya que $\pi(w_{k'}) = \min_{u \in V \setminus S_{k'-1}} \pi(u) = \infty$. No existe arista que sale de un nodo $w_p \in S_{k'-1}$ con $p \leq k' - 1$ y llega a un nodo $u \notin S_{k'-1}$ pues si existiese (w_p, u) , $\pi(u) \leq \pi(w_p) + l(w_p, u) < \infty$, una vez que termina la iteración p . Lo cual es una contradicción, ya que no se puede aumentar el valor de $\pi(u)$ a lo largo de la ejecución del algoritmo. Por lo tanto, no hay camino que sale de un nodo de $S_{k'-1}$ y termina en un nodo fuera de $S_{k'-1}$ y particularmente para v y w_k . Consecuentemente, $\text{dist}(v, w_k) = \infty$. Este argumento también es el justificativo para interrumpir la ejecución cuando se haya elegido un nodo cuyo valor de π es ∞ para ser agregado a S .

Prueba del lema: Continuación

Probamos ahora por inducción en k si $\pi(w_k) < \infty$ entonces $\pi(w_k) = \text{dist}(v, w_k)$.

Prueba del lema: Continuación

Probamos ahora por inducción en k si $\pi(w_k) < \infty$ entonces $\pi(w_k) = \text{dist}(v, w_k)$.

Caso base con $k = 0$, $\pi(w_0 = v) = 0 < \infty$. Como $\text{dist}(v, w_0 = v) = 0$, así que se cumple $\pi(w_0) = \text{dist}(v, w_0)$.

Prueba del lema: Continuación

Probamos ahora por inducción en k si $\pi(w_k) < \infty$ entonces $\pi(w_k) = \text{dist}(v, w_k)$.

Caso base con $k = 0$, $\pi(w_0 = v) = 0 < \infty$. Como $\text{dist}(v, w_0 = v) = 0$, así que se cumple $\pi(w_0) = \text{dist}(v, w_0)$.

Supongamos que vale para cualquier $k' < k$ con $k \geq 1$ que si $\pi(w_{k'}) < \infty$ entonces $\pi(w_{k'}) = \text{dist}(v, w_{k'})$.

Prueba del lema: Continuación

Probamos ahora por inducción en k si $\pi(w_k) < \infty$ entonces $\pi(w_k) = \text{dist}(v, w_k)$.

Caso base con $k = 0$, $\pi(w_0 = v) = 0 < \infty$. Como $\text{dist}(v, w_0 = v) = 0$, así que se cumple $\pi(w_0) = \text{dist}(v, w_0)$.

Supongamos que vale para cualquier $k' < k$ con $k \geq 1$ que si $\pi(w_{k'}) < \infty$ entonces $\pi(w_{k'}) = \text{dist}(v, w_{k'})$. En caso que $\pi(w_k) < \infty$, tomemos C_{v, w_k} un camino mínimo que sale de v y termina en w_k , es decir que $l(C_{v, w_k}) = \text{dist}(v, w_k)$.

Prueba del lema: Continuación

Probamos ahora por inducción en k si $\pi(w_k) < \infty$ entonces $\pi(w_k) = \text{dist}(v, w_k)$.

Caso base con $k = 0$, $\pi(w_0 = v) = 0 < \infty$. Como $\text{dist}(v, w_0 = v) = 0$, así que se cumple $\pi(w_0) = \text{dist}(v, w_0)$.

Supongamos que vale para cualquier $k' < k$ con $k \geq 1$ que si $\pi(w_{k'}) < \infty$ entonces $\pi(w_{k'}) = \text{dist}(v, w_{k'})$. En caso que $\pi(w_k) < \infty$, tomemos C_{v, w_k} un camino mínimo que sale de v y termina en w_k , es decir que $l(C_{v, w_k}) = \text{dist}(v, w_k)$. Sabemos que el primer nodo de C_{v, w_k} , $v = w_0 \in S_{k-1}$ y su último nodo, $w_k \notin S_{k-1}$.

Prueba del lema: Continuación

Probamos ahora por inducción en k si $\pi(w_k) < \infty$ entonces $\pi(w_k) = \text{dist}(v, w_k)$.

Caso base con $k = 0$, $\pi(w_0 = v) = 0 < \infty$. Como $\text{dist}(v, w_0 = v) = 0$, así que se cumple $\pi(w_0) = \text{dist}(v, w_0)$.

Supongamos que vale para cualquier $k' < k$ con $k \geq 1$ que si $\pi(w_{k'}) < \infty$ entonces $\pi(w_{k'}) = \text{dist}(v, w_{k'})$. En caso que $\pi(w_k) < \infty$, tomemos C_{v, w_k} un camino mínimo que sale de v y termina en w_k , es decir que $l(C_{v, w_k}) = \text{dist}(v, w_k)$. Sabemos que el primer nodo de C_{v, w_k} , $v = w_0 \in S_{k-1}$ y su último nodo, $w_k \notin S_{k-1}$. Sea z , el primer nodo de C_{v, w_k} tal que $z \notin S_{k-1}$ y w_p el nodo anterior de z en C_{v, w_k} con $0 \leq p \leq k-1$ y C_{v, w_p} el subcamino de C_{v, w_k} entre v y w_p y $C_{v, z}$ el subcamino entre v y z .

Prueba del lema: Continuación

Probamos ahora por inducción en k si $\pi(w_k) < \infty$ entonces $\pi(w_k) = \text{dist}(v, w_k)$.

Caso base con $k = 0$, $\pi(w_0 = v) = 0 < \infty$. Como $\text{dist}(v, w_0 = v) = 0$, así que se cumple $\pi(w_0) = \text{dist}(v, w_0)$.

Supongamos que vale para cualquier $k' < k$ con $k \geq 1$ que si $\pi(w_{k'}) < \infty$ entonces $\pi(w_{k'}) = \text{dist}(v, w_{k'})$. En caso que $\pi(w_k) < \infty$, tomemos C_{v, w_k} un camino mínimo que sale de v y termina en w_k , es decir que $l(C_{v, w_k}) = \text{dist}(v, w_k)$. Sabemos que el primer nodo de C_{v, w_k} , $v = w_0 \in S_{k-1}$ y su último nodo, $w_k \notin S_{k-1}$. Sea z , el primer nodo de C_{v, w_k} tal que $z \notin S_{k-1}$ y w_p el nodo anterior de z en C_{v, w_k} con $0 \leq p \leq k-1$ y C_{v, w_p} el subcamino de C_{v, w_k} entre v y w_p y $C_{v, z}$ el subcamino entre v y z . Claramente, $l(C_{v, z}) = l(C_{v, w_p}) + l(w_p, z)$.

Prueba del lema: Continuación

Probamos ahora por inducción en k si $\pi(w_k) < \infty$ entonces $\pi(w_k) = \text{dist}(v, w_k)$.

Caso base con $k = 0$, $\pi(w_0 = v) = 0 < \infty$. Como $\text{dist}(v, w_0 = v) = 0$, así que se cumple $\pi(w_0) = \text{dist}(v, w_0)$.

Supongamos que vale para cualquier $k' < k$ con $k \geq 1$ que si $\pi(w_{k'}) < \infty$ entonces $\pi(w_{k'}) = \text{dist}(v, w_{k'})$. En caso que $\pi(w_k) < \infty$, tomemos C_{v, w_k} un camino mínimo que sale de v y termina en w_k , es decir que $l(C_{v, w_k}) = \text{dist}(v, w_k)$. Sabemos que el primer nodo de C_{v, w_k} , $v = w_0 \in S_{k-1}$ y su último nodo, $w_k \notin S_{k-1}$. Sea z , el primer nodo de C_{v, w_k} tal que $z \notin S_{k-1}$ y w_p el nodo anterior de z en C_{v, w_k} con $0 \leq p \leq k-1$ y C_{v, w_p} el subcamino de C_{v, w_k} entre v y w_p y $C_{v, z}$ el subcamino entre v y z . Claramente, $l(C_{v, z}) = l(C_{v, w_p}) + l(w_p, z)$. Entonces $\text{dist}(v, w_k) = l(C_{v, w_k}) \geq l(C_{v, z}) = l(C_{v, w_p}) + l(w_p, z) = \text{dist}(v, w_p) + l(w_p, z) = \pi(w_p) + l(w_p, z)$ porque no hay aristas de longitud negativa y por la propiedad de subestructura óptima de un camino mínimo e hipótesis inductiva aplicada para $p < k$.

Prueba del lema: Continuación 2

Es claro que vale $\text{dist}(v, w_k) \geq \pi(w_p) + I(w_p, z) \geq \pi(z)$ una vez que termina la iteración p y se mantiene la desigualdad $\text{dist}(v, w_k) \geq \pi(z)$ hasta que finaliza la ejecución del algoritmo.

Prueba del lema: Continuación 2

Es claro que vale $\text{dist}(v, w_k) \geq \pi(w_p) + I(w_p, z) \geq \pi(z)$ una vez que termina la iteración p y se mantiene la desigualdad $\text{dist}(v, w_k) \geq \pi(z)$ hasta que finaliza la ejecución del algoritmo. En particular, en la iteración k se eligió w_k para agregarlo a S (tanto w_k como z no están en S_{k-1}) lo cual implica que $\text{dist}(v, w_k) \geq \pi(z) \geq \pi(w_k)$ en ese momento.

Prueba del lema: Continuación 2

Es claro que vale $\text{dist}(v, w_k) \geq \pi(w_p) + l(w_p, z) \geq \pi(z)$ una vez que termina la iteración p y se mantiene la desigualdad $\text{dist}(v, w_k) \geq \pi(z)$ hasta que finaliza la ejecución del algoritmo. En particular, en la iteración k se eligió w_k para agregarlo a S (tanto w_k como z no están en S_{k-1}) lo cual implica que $\text{dist}(v, w_k) \geq \pi(z) \geq \pi(w_k)$ en ese momento.

Por otro lado, siempre vale $\text{dist}(v, w_k) \leq \pi(w_k)$ ya que $\pi(w_k)$ representa la longitud de un camino que sale de v y termina en w_k y $\text{dist}(v, w_k)$ es la longitud de un camino mínimo con estas características.

Prueba del lema: Continuación 2

Es claro que vale $\text{dist}(v, w_k) \geq \pi(w_p) + l(w_p, z) \geq \pi(z)$ una vez que termina la iteración p y se mantiene la desigualdad $\text{dist}(v, w_k) \geq \pi(z)$ hasta que finaliza la ejecución del algoritmo. En particular, en la iteración k se eligió w_k para agregarlo a S (tanto w_k como z no están en S_{k-1}) lo cual implica que $\text{dist}(v, w_k) \geq \pi(z) \geq \pi(w_k)$ en ese momento.

Por otro lado, siempre vale $\text{dist}(v, w_k) \leq \pi(w_k)$ ya que $\pi(w_k)$ representa la longitud de un camino que sale de v y termina en w_k y $\text{dist}(v, w_k)$ es la longitud de un camino mínimo con estas características. Entonces $\pi(w_k) = \text{dist}(v, w_k)$.

Complejidad del Algoritmo de Dijkstra

- ▶ $O(n^2)$, implementación trivial.

Complejidad del Algoritmo de Dijkstra

- ▶ $O(n^2)$, implementación trivial.
- ▶ $O((m + n) \log n)$, usando heap binario.

Complejidad del Algoritmo de Dijkstra

- ▶ $O(n^2)$, implementación trivial.
- ▶ $O((m + n) \log n)$, usando heap binario.
- ▶ $O(m + n \log n)$, usando heap Fibonacci.

Algoritmo de Ford (1956), más conocido como Bellman-Ford



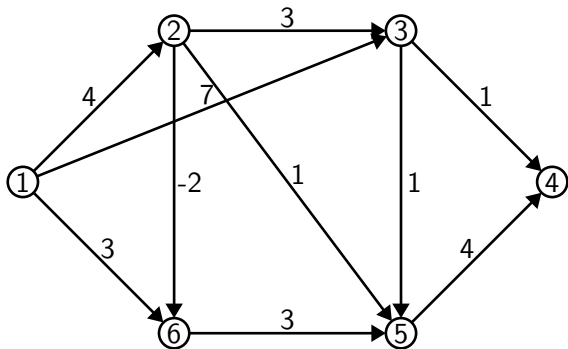
Lester Ford Jr. (1927–2017)

Algoritmo de Ford (1956)

Asumimos que el grafo es orientado y no tiene ciclos de longitud negativa alcanzables desde v .

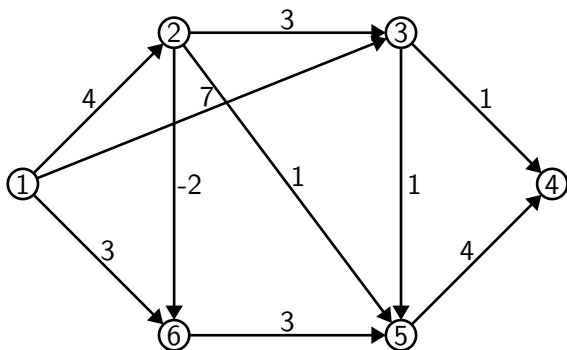
```
 $\pi(v) := 0$   
para todo  $u \in V \setminus \{v\}$  hacer  
     $\pi(u) := \infty$   
fin para  
mientras hay cambios en  $\pi$  hacer  
     $\pi' := \pi$   
    para todo  $u \in V \setminus \{v\}$  hacer  
         $\pi(u) := \min(\pi(u), \min_{(w,u) \in X} \pi'(w) + l(w, u))$   
    fin para  
fin mientras  
retornar  $\pi$ 
```


Algoritmo de Ford - Ejemplo



Algoritmo de Ford - Ejemplo

$$\pi = (0, \infty, \infty, \infty, \infty, \infty)$$

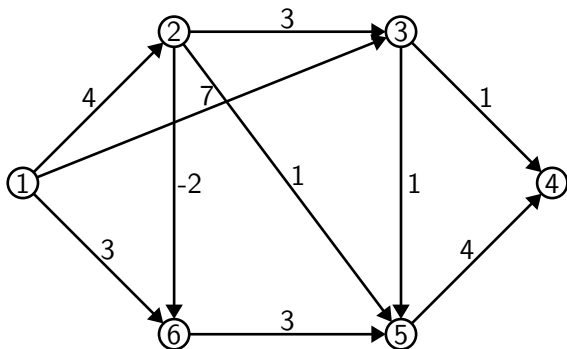


Algoritmo de Ford - Ejemplo

Iteración 1

$$\pi = (0, \infty, \infty, \infty, \infty, \infty)$$

$$\pi' = (0, \infty, \infty, \infty, \infty, \infty)$$

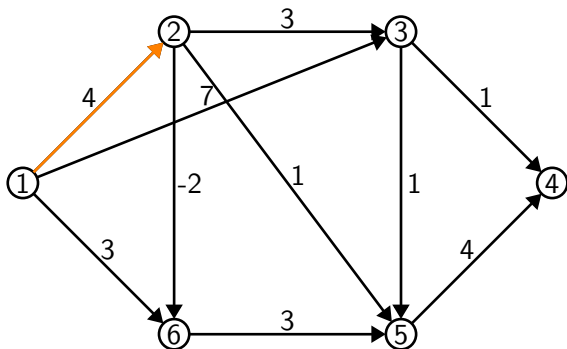


Algoritmo de Ford - Ejemplo

Iteración 1

$$\pi = (0, \infty, \infty, \infty, \infty, \infty)$$

$$\pi' = (0, \infty, \infty, \infty, \infty, \infty)$$

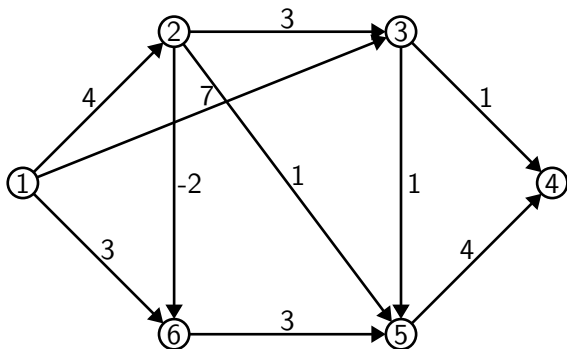


Algoritmo de Ford - Ejemplo

Iteración 1

$$\pi = (0, 4, \infty, \infty, \infty, \infty)$$

$$\pi' = (0, \infty, \infty, \infty, \infty, \infty)$$

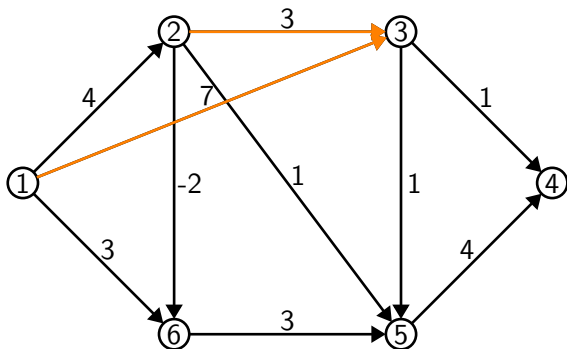


Algoritmo de Ford - Ejemplo

Iteración 1

$$\pi = (0, 4, \infty, \infty, \infty, \infty)$$

$$\pi' = (0, \infty, \infty, \infty, \infty, \infty)$$

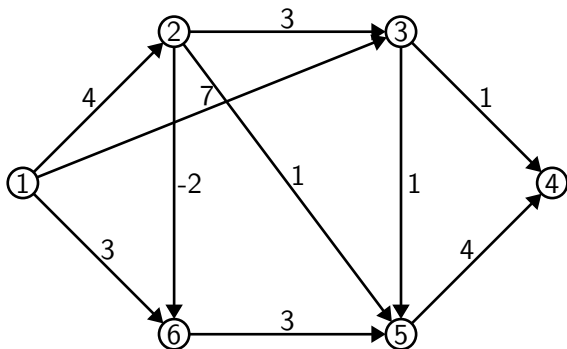


Algoritmo de Ford - Ejemplo

Iteración 1

$$\pi = (0, 4, 7, \infty, \infty, \infty)$$

$$\pi' = (0, \infty, \infty, \infty, \infty, \infty)$$

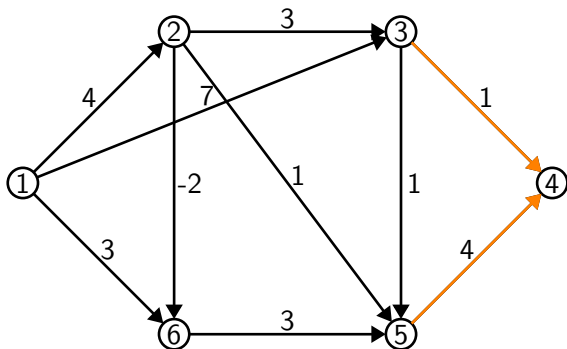


Algoritmo de Ford - Ejemplo

Iteración 1

$$\pi = (0, 4, 7, \infty, \infty, \infty)$$

$$\pi' = (0, \infty, \infty, \infty, \infty, \infty)$$

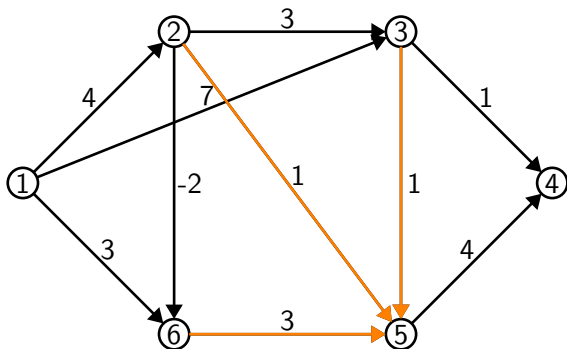


Algoritmo de Ford - Ejemplo

Iteración 1

$$\pi = (0, 4, 7, \infty, \textcolor{green}{\infty}, \infty)$$

$$\pi' = (0, \textcolor{orange}{\infty}, \textcolor{orange}{\infty}, \infty, \infty, \textcolor{orange}{\infty})$$

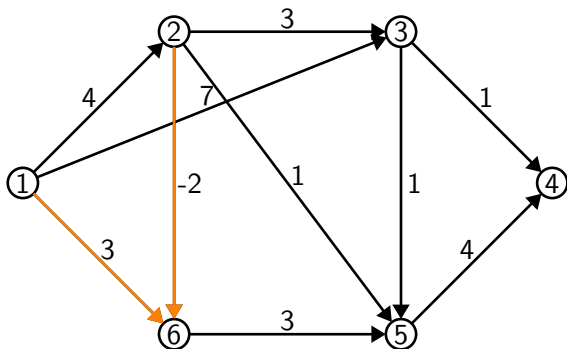


Algoritmo de Ford - Ejemplo

Iteración 1

$$\pi = (0, 4, 7, \infty, \infty, \infty)$$

$$\pi' = (0, \infty, \infty, \infty, \infty, \infty)$$

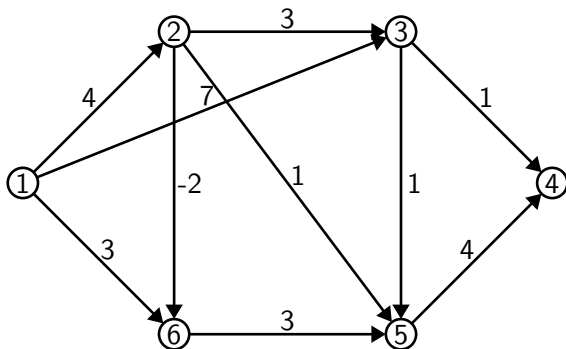


Algoritmo de Ford - Ejemplo

Iteración 1

$$\pi = (0, 4, 7, \infty, \infty, 3)$$

$$\pi' = (0, \infty, \infty, \infty, \infty, \infty)$$

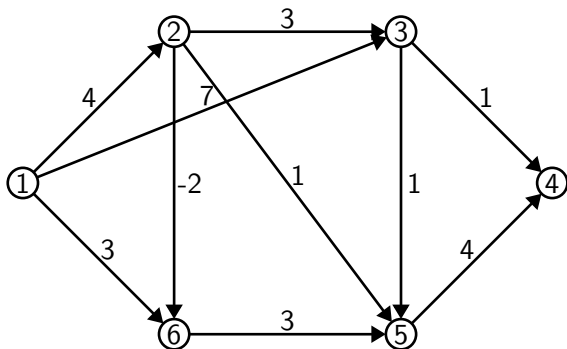


Algoritmo de Ford - Ejemplo

Iteración 2

$$\pi = (0, 4, 7, \infty, \infty, 3)$$

$$\pi' = (0, 4, 7, \infty, \infty, 3)$$

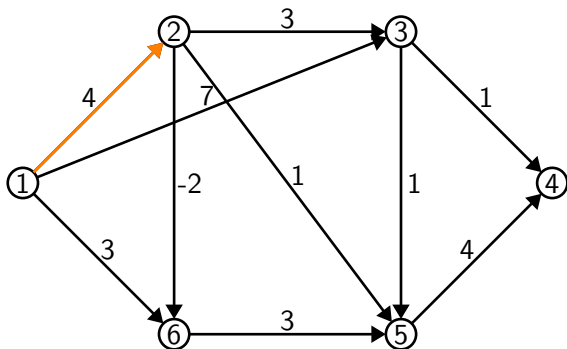


Algoritmo de Ford - Ejemplo

Iteración 2

$$\pi = (0, 4, 7, \infty, \infty, 3)$$

$$\pi' = (0, 4, 7, \infty, \infty, 3)$$

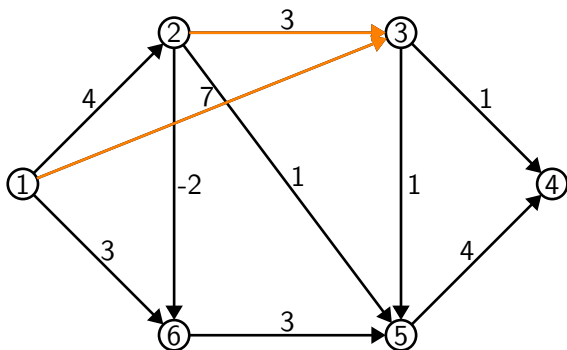


Algoritmo de Ford - Ejemplo

Iteración 2

$$\pi = (0, 4, 7, \infty, \infty, 3)$$

$$\pi' = (0, 4, 7, \infty, \infty, 3)$$

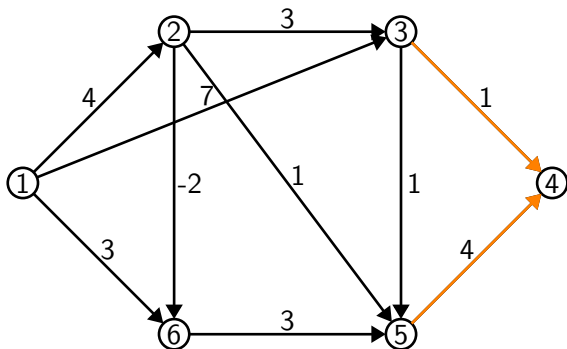


Algoritmo de Ford - Ejemplo

Iteración 2

$$\pi = (0, 4, 7, \infty, \infty, 3)$$

$$\pi' = (0, 4, 7, \infty, \infty, 3)$$

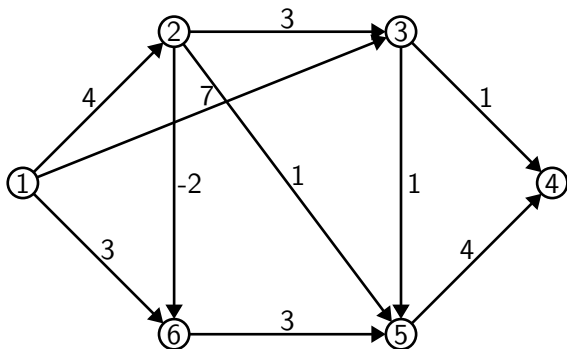


Algoritmo de Ford - Ejemplo

Iteración 2

$$\pi = (0, 4, 7, 8, \infty, 3)$$

$$\pi' = (0, 4, 7, \infty, \infty, 3)$$

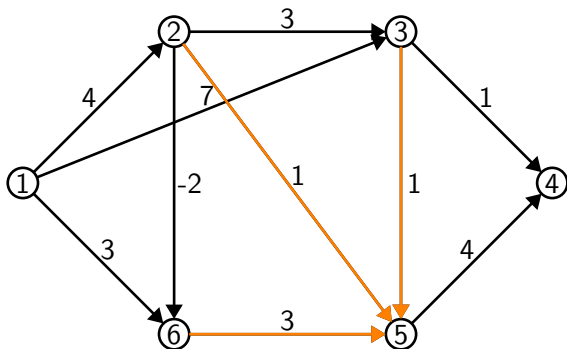


Algoritmo de Ford - Ejemplo

Iteración 2

$$\pi = (0, 4, 7, 8, \infty, 3)$$

$$\pi' = (0, 4, 7, \infty, \infty, 3)$$

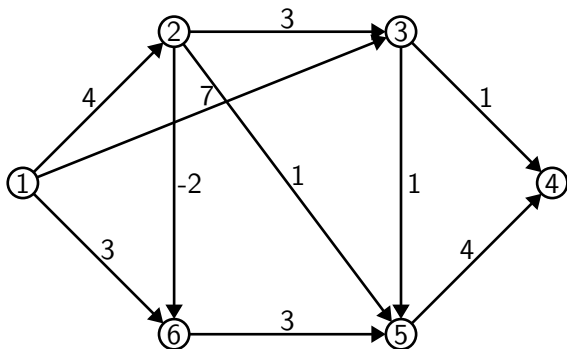


Algoritmo de Ford - Ejemplo

Iteración 2

$$\pi = (0, 4, 7, 8, 5, 3)$$

$$\pi' = (0, 4, 7, \infty, \infty, 3)$$

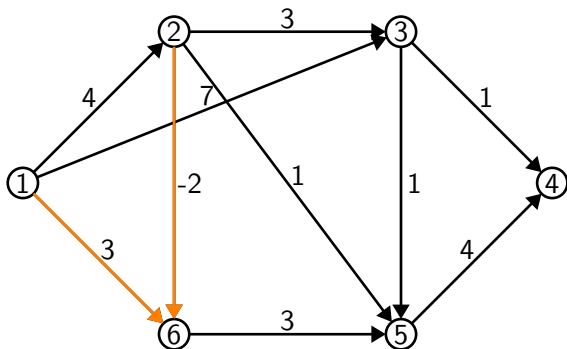


Algoritmo de Ford - Ejemplo

Iteración 2

$$\pi = (0, 4, 7, 8, 5, \mathbf{3})$$

$$\pi' = (\mathbf{0}, \mathbf{4}, 7, \infty, \infty, 3)$$

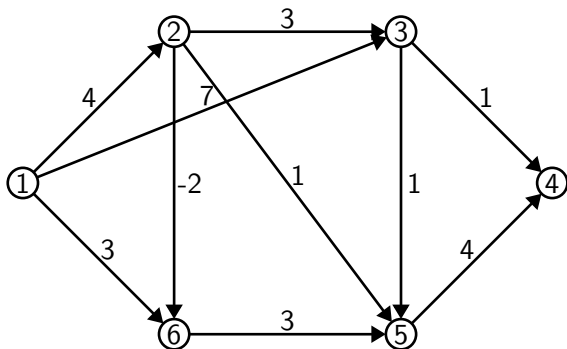


Algoritmo de Ford - Ejemplo

Iteración 2

$$\pi = (0, 4, 7, 8, 5, 2)$$

$$\pi' = (0, 4, 7, \infty, \infty, 3)$$



Algoritmo de Ford (1956)

Lema 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , en todo momento de la ejecución del algoritmo de **Ford**.

1. Si $\pi(w) < \infty$ para algún nodo w entonces existe un recorrido R que conecta v con w y $\pi(w) = l(R)$.
2. Si existe un camino mínimo entre v y w entonces $\pi(w) \geq dist(v, w)$.

Algoritmo de Ford (1956)

Lema 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , en todo momento de la ejecución del algoritmo de **Ford**.

1. Si $\pi(w) < \infty$ para algún nodo w entonces existe un recorrido R que conecta v con w y $\pi(w) = l(R)$.
2. Si existe un camino mínimo entre v y w entonces $\pi(w) \geq dist(v, w)$.

Lema 2: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v . Si C es un camino entre v y w con k aristas entonces al finalizar la iteración k (Loop de Mientras) del algoritmo de **Ford** (puede ocurrir antes), $\pi(w) \leq l(C)$.

Algoritmo de Ford (1956)

Lema 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , en todo momento de la ejecución del algoritmo de **Ford**.

1. Si $\pi(w) < \infty$ para algún nodo w entonces existe un recorrido R que conecta v con w y $\pi(w) = l(R)$.
2. Si existe un camino mínimo entre v y w entonces $\pi(w) \geq dist(v, w)$.

Lema 2: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v . Si C es un camino entre v y w con k aristas entonces al finalizar la iteración k (Loop de Mientras) del algoritmo de **Ford** (puede ocurrir antes), $\pi(w) \leq l(C)$.

Corolario 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , al finalizar la iteración k (Loop de Mientras) del algoritmo de **Ford** determina un camino mínimo entre v y w si existe un camino mínimo de v a w con a lo sumo k aristas.

Prueba de Lema 1

Probamos (1) por inducción en k que es el número de iteraciones transcurridas.

Prueba de Lema 1

Probamos (1) por inducción en k que es el número de iteraciones transcurridas.

Case base con $k = 0$. Únicamente $\pi(v) = 0 < \infty$ y el recorrido vacío que conecta v consigo mismo tiene longitud 0.

Prueba de Lema 1

Probamos (1) por inducción en k que es el número de iteraciones transcurridas.

Case base con $k = 0$. Únicamente $\pi(v) = 0 < \infty$ y el recorrido vacío que conecta v consigo mismo tiene longitud 0.

Supongamos que vale (1) al terminar la iteración $k - 1$ con $k \geq 1$.

Prueba de Lema 1

Probamos (1) por inducción en k que es el número de iteraciones transcurridas.

Case base con $k = 0$. Únicamente $\pi(v) = 0 < \infty$ y el recorrido vacío que conecta v consigo mismo tiene longitud 0.

Supongamos que vale (1) al terminar la iteración $k - 1$ con $k \geq 1$. Si $\pi(w) < \infty$ al terminar la iteración k , en el caso que $\pi(w)$ no se había modificado en la iteración k , por HI al finalizar la iteración $k - 1$ existe un recorrido R entre v y w con $l(R) = \pi(w)$.

Prueba de Lema 1

Probamos (1) por inducción en k que es el número de iteraciones transcurridas.

Case base con $k = 0$. Únicamente $\pi(v) = 0 < \infty$ y el recorrido vacío que conecta v consigo mismo tiene longitud 0.

Supongamos que vale (1) al terminar la iteración $k - 1$ con $k \geq 1$. Si $\pi(w) < \infty$ al terminar la iteración k , en el caso que $\pi(w)$ no se había modificado en la iteración k , por HI al finalizar la iteración $k - 1$ existe un recorrido R entre v y w con $l(R) = \pi(w)$. Caso contrario, $\pi(w) = \pi'(x) + l(x, w) < \infty$ para algún nodo x donde $\pi'(x) < \infty$ es el valor de $\pi(x)$ cuando termina la iteración $k - 1$ y por HI, existe un recorrido R' entre v y x tal que $l(R') = \pi'(x)$.

Prueba de Lema 1

Probamos (1) por inducción en k que es el número de iteraciones transcurridas.

Case base con $k = 0$. Únicamente $\pi(v) = 0 < \infty$ y el recorrido vacío que conecta v consigo mismo tiene longitud 0.

Supongamos que vale (1) al terminar la iteración $k - 1$ con $k \geq 1$.

Si $\pi(w) < \infty$ al terminar la iteración k , en el caso que $\pi(w)$ no se había modificado en la iteración k , por HI al finalizar la iteración $k - 1$ existe un recorrido R entre v y w con $l(R) = \pi(w)$. Caso contrario, $\pi(w) = \pi'(x) + l(x, w) < \infty$ para algún nodo x donde $\pi'(x) < \infty$ es el valor de $\pi(x)$ cuando termina la iteración $k - 1$ y por HI, existe un recorrido R' entre v y x tal que $l(R') = \pi'(x)$.

Agregamos la arista (x, w) a R' y obtenemos un recorrido R entre v y w y $l(R) = l(R') + l(x, w) = \pi'(x) + l(x, w) = \pi(w)$.

Prueba de Lema 1

Probamos (1) por inducción en k que es el número de iteraciones transcurridas.

Case base con $k = 0$. Únicamente $\pi(v) = 0 < \infty$ y el recorrido vacío que conecta v consigo mismo tiene longitud 0.

Supongamos que vale (1) al terminar la iteración $k - 1$ con $k \geq 1$.

Si $\pi(w) < \infty$ al terminar la iteración k , en el caso que $\pi(w)$ no se había modificado en la iteración k , por HI al finalizar la iteración $k - 1$ existe un recorrido R entre v y w con $l(R) = \pi(w)$. Caso contrario, $\pi(w) = \pi'(x) + l(x, w) < \infty$ para algún nodo x donde $\pi'(x) < \infty$ es el valor de $\pi(x)$ cuando termina la iteración $k - 1$ y por HI, existe un recorrido R' entre v y x tal que $l(R') = \pi'(x)$.

Agregamos la arista (x, w) a R' y obtenemos un recorrido R entre v y w y $l(R) = l(R') + l(x, w) = \pi'(x) + l(x, w) = \pi(w)$.

(2) es una consecuencia de (1) ya que $dist(v, w)$ es la longitud de cualquier camino mínimo entre v y w si existe al menos uno. Un camino mínimo es un recorrido mínimo.

Prueba de Lema 2

Probamos por inducción en k .

Prueba de Lema 2

Probamos por inducción en k .

Case base donde $k = 0$.

Prueba de Lema 2

Probamos por inducción en k .

Case base donde $k = 0$. Claramente, el único camino sin aristas (camino vacío) que sale de v es el que conecta v consigo mismo y tiene longitud 0.

Prueba de Lema 2

Probamos por inducción en k .

Case base donde $k = 0$. Claramente, el único camino sin aristas (camino vacío) que sale de v es el que conecta v consigo mismo y tiene longitud 0. Como se inicializa $\pi(v) = 0$ antes iterar, se cumple el lema.

Prueba de Lema 2

Probamos por inducción en k .

Case base donde $k = 0$. Claramente, el único camino sin aristas (camino vacío) que sale de v es el que conecta v consigo mismo y tiene longitud 0. Como se inicializa $\pi(v) = 0$ antes iterar, se cumple el lema.

Supongamos que vale el lema para caminos de $k - 1$ aristas que salen de v con $k \geq 1$.

Prueba de Lema 2

Probamos por inducción en k .

Case base donde $k = 0$. Claramente, el único camino sin aristas (camino vacío) que sale de v es el que conecta v consigo mismo y tiene longitud 0. Como se inicializa $\pi(v) = 0$ antes iterar, se cumple el lema.

Supongamos que vale el lema para caminos de $k - 1$ aristas que salen de v con $k \geq 1$. Ahora consideramos un camino C de k aristas que sale v y llega a w .

Prueba de Lema 2

Probamos por inducción en k .

Case base donde $k = 0$. Claramente, el único camino sin aristas (camino vacío) que sale de v es el que conecta v consigo mismo y tiene longitud 0. Como se inicializa $\pi(v) = 0$ antes iterar, se cumple el lema.

Supongamos que vale el lema para caminos de $k - 1$ aristas que salen de v con $k \geq 1$. Ahora consideramos un camino C de k aristas que sale v y llega a w . Sea u el nodo anterior de w en C y C' el subcamino obtenido de C quitando el nodo w y la arista (u, w) .

Prueba de Lema 2

Probamos por inducción en k .

Case base donde $k = 0$. Claramente, el único camino sin aristas (camino vacío) que sale de v es el que conecta v consigo mismo y tiene longitud 0. Como se inicializa $\pi(v) = 0$ antes iterar, se cumple el lema.

Supongamos que vale el lema para caminos de $k - 1$ aristas que salen de v con $k \geq 1$. Ahora consideramos un camino C de k aristas que sale v y llega a w . Sea u el nodo anterior de w en C y C' el subcamino obtenido de C quitando el nodo w y la arista (u, w) . C' conecta v con u y tiene $k - 1$ aristas.

Prueba de Lema 2

Probamos por inducción en k .

Case base donde $k = 0$. Claramente, el único camino sin aristas (camino vacío) que sale de v es el que conecta v consigo mismo y tiene longitud 0. Como se inicializa $\pi(v) = 0$ antes iterar, se cumple el lema.

Supongamos que vale el lema para caminos de $k - 1$ aristas que salen de v con $k \geq 1$. Ahora consideramos un camino C de k aristas que sale v y llega a w . Sea u el nodo anterior de w en C y C' el subcamino obtenido de C quitando el nodo w y la arista (u, w) . C' conecta v con u y tiene $k - 1$ aristas. Por *HI*, al terminar la iteración $k - 1$, $\pi(u) \leq l(C')$.

Prueba de Lema 2

Probamos por inducción en k .

Case base donde $k = 0$. Claramente, el único camino sin aristas (camino vacío) que sale de v es el que conecta v consigo mismo y tiene longitud 0. Como se inicializa $\pi(v) = 0$ antes iterar, se cumple el lema.

Supongamos que vale el lema para caminos de $k - 1$ aristas que salen de v con $k \geq 1$. Ahora consideramos un camino C de k aristas que sale v y llega a w . Sea u el nodo anterior de w en C y C' el subcamino obtenido de C quitando el nodo w y la arista (u, w) . C' conecta v con u y tiene $k - 1$ aristas. Por *HI*, al terminar la iteración $k - 1$, $\pi(u) \leq l(C')$. Al terminar la iteración k , $\pi(w) \leq \pi'(u) + l(u, w) \leq l(C') + l(u, w) = l(C)$ donde $\pi'(u)$ guarda el valor de $\pi(u)$ cuando terminó la iteración anterior.

Algoritmo de Ford (1956)

Teorema: Dado un grafo orientado G sin ciclos de longitud negativa alcanzables desde v , el algoritmo de **Ford** determina un camino mínimo entre el nodo v y cada nodo alcanzable desde v .

Algoritmo de Ford (1956)

Teorema: Dado un grafo orientado G sin ciclos de longitud negativa alcanzables desde v , el algoritmo de **Ford** determina un camino mínimo entre el nodo v y cada nodo alcanzable desde v .

Prueba: Sea w alcanzable desde v .

Algoritmo de Ford (1956)

Teorema: Dado un grafo orientado G sin ciclos de longitud negativa alcanzables desde v , el algoritmo de **Ford** determina un camino mínimo entre el nodo v y cada nodo alcanzable desde v .

Prueba: Sea w alcanzable desde v . Veamos que dado cualquier recorrido R entre v y w , en caso que R contiene un ciclo C y como C es alcanzable desde v , por hipótesis, C no es de longitud negativa y $R \setminus C$ es otro recorrido entre v y w , además $I(R \setminus C) \leq I(R)$.

Algoritmo de Ford (1956)

Teorema: Dado un grafo orientado G sin ciclos de longitud negativa alcanzables desde v , el algoritmo de **Ford** determina un camino mínimo entre el nodo v y cada nodo alcanzable desde v .

Prueba: Sea w alcanzable desde v . Veamos que dado cualquier recorrido R entre v y w , en caso que R contiene un ciclo C y como C es alcanzable desde v , por hipótesis, C no es de longitud negativa y $R \setminus C$ es otro recorrido entre v y w , además $I(R \setminus C) \leq I(R)$. Por lo tanto, basta considerar recorridos entre v y w que son caminos para buscar recorridos/caminos mínimos entre v y w .

Algoritmo de Ford (1956)

Teorema: Dado un grafo orientado G sin ciclos de longitud negativa alcanzables desde v , el algoritmo de **Ford** determina un camino mínimo entre el nodo v y cada nodo alcanzable desde v .

Prueba: Sea w alcanzable desde v . Veamos que dado cualquier recorrido R entre v y w , en caso que R contiene un ciclo C y como C es alcanzable desde v , por hipótesis, C no es de longitud negativa y $R \setminus C$ es otro recorrido entre v y w , además $I(R \setminus C) \leq I(R)$. Por lo tanto, basta considerar recorridos entre v y w que son caminos para buscar recorridos/caminos mínimos entre v y w . Como hay una cantidad finita de caminos posibles entre v y w , existe al menos un camino mínimo entre v y w y por Corolario 1, el algoritmo encuentra un camino mínimo entre v y w en las primeras $n - 1$ iteraciones ya que cualquier camino tiene a lo sumo $n - 1$ aristas.

Algoritmo de Ford (1956)

Corolario 2: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si hubo cambio de π hasta la iteración n inclusive en la ejecución entonces existe un ciclo de longitud negativa (c.l.n.) alcanzable desde v .

Algoritmo de Ford (1956)

Corolario 2: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si hubo cambio de π hasta la iteración n inclusive en la ejecución entonces existe un ciclo de longitud negativa (c.l.n.) alcanzable desde v .

Prueba: Supongamos que no existe un c.l.n. alcanzable desde v .

Algoritmo de Ford (1956)

Corolario 2: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si hubo cambio de π hasta la iteración n inclusive en la ejecución entonces existe un ciclo de longitud negativa (c.l.n.) alcanzable desde v .

Prueba: Supongamos que no existe un c.l.n. alcanzable desde v . De acuerdo a la prueba del último teorema, para cada nodo w alcanzable desde v se puede encontrar un camino mínimo entre v y w en las primeras $n - 1$ iteraciones, es decir que $\pi(w) = \text{dist}(v, w)$ y no se va modificar más (por Lema 1 parte (2)).

Algoritmo de Ford (1956)

Corolario 2: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si hubo cambio de π hasta la iteración n inclusive en la ejecución entonces existe un ciclo de longitud negativa (c.l.n.) alcanzable desde v .

Prueba: Supongamos que no existe un c.l.n. alcanzable desde v . De acuerdo a la prueba del último teorema, para cada nodo w alcanzable desde v se puede encontrar un camino mínimo entre v y w en las primeras $n - 1$ iteraciones, es decir que $\pi(w) = \text{dist}(v, w)$ y no se va modificar más (por Lema 1 parte (2)). Ahora consideramos un nodo w no alcanzable desde v , entonces no existe recorrido entre v y w y por Lema 1 parte (1), $\pi(w) = \infty$ en todo momento.

Algoritmo de Ford (1956)

Corolario 2: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si hubo cambio de π hasta la iteración n inclusive en la ejecución entonces existe un ciclo de longitud negativa (c.l.n.) alcanzable desde v .

Prueba: Supongamos que no existe un c.l.n. alcanzable desde v . De acuerdo a la prueba del último teorema, para cada nodo w alcanzable desde v se puede encontrar un camino mínimo entre v y w en las primeras $n - 1$ iteraciones, es decir que $\pi(w) = \text{dist}(v, w)$ y no se va modificar más (por Lema 1 parte (2)). Ahora consideramos un nodo w no alcanzable desde v , entonces no existe recorrido entre v y w y por Lema 1 parte (1), $\pi(w) = \infty$ en todo momento. Por lo tanto, en la iteración n no hubo cambio para ningún $\pi(w)$ (esto puede haber ocurrido desde alguna iteración anterior).

Algoritmo de Ford (1956)

Corolario 2: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si hubo cambio de π hasta la iteración n inclusive en la ejecución entonces existe un ciclo de longitud negativa (c.l.n.) alcanzable desde v .

Prueba: Supongamos que no existe un c.l.n. alcanzable desde v . De acuerdo a la prueba del último teorema, para cada nodo w alcanzable desde v se puede encontrar un camino mínimo entre v y w en las primeras $n - 1$ iteraciones, es decir que $\pi(w) = \text{dist}(v, w)$ y no se va modificar más (por Lema 1 parte (2)). Ahora consideramos un nodo w no alcanzable desde v , entonces no existe recorrido entre v y w y por Lema 1 parte (1), $\pi(w) = \infty$ en todo momento. Por lo tanto, en la iteración n no hubo cambio para ningún $\pi(w)$ (esto puede haber ocurrido desde alguna interacción anterior). Esto contradice que hubo cambio de π hasta la iteración n inclusive.

Algoritmo de Ford (1956)

Corolario 2: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si hubo cambio de π hasta la iteración n inclusive en la ejecución entonces existe un ciclo de longitud negativa (c.l.n.) alcanzable desde v .

Prueba: Supongamos que no existe un c.l.n. alcanzable desde v . De acuerdo a la prueba del último teorema, para cada nodo w alcanzable desde v se puede encontrar un camino mínimo entre v y w en las primeras $n - 1$ iteraciones, es decir que $\pi(w) = \text{dist}(v, w)$ y no se va modificar más (por Lema 1 parte (2)). Ahora consideramos un nodo w no alcanzable desde v , entonces no existe recorrido entre v y w y por Lema 1 parte (1), $\pi(w) = \infty$ en todo momento. Por lo tanto, en la iteración n no hubo cambio para ningún $\pi(w)$ (esto puede haber ocurrido desde alguna interacción anterior). Esto contradice que hubo cambio de π hasta la iteración n inclusive. Por lo cual, existe un c.l.n. alcanzable desde v .

Algoritmo de Ford (1956)

Proposición 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si existe un c.l.n. alcanzable desde v entonces hay cambio de π en toda iteración de la ejecución del algoritmo de **Ford**.

Algoritmo de Ford (1956)

Proposición 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si existe un c.l.n. alcanzable desde v entonces hay cambio de π en toda iteración de la ejecución del algoritmo de **Ford**.

Prueba: Sea C un c.l.n. alcanzable desde v y los nodos de C de acuerdo al orden de la orientación son w_1, \dots, w_q .

Algoritmo de Ford (1956)

Proposición 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si existe un c.l.n. alcanzable desde v entonces hay cambio de π en toda iteración de la ejecución del algoritmo de **Ford**.

Prueba: Sea C un c.l.n. alcanzable desde v y los nodos de C de acuerdo al orden de la orientación son w_1, \dots, w_q . Entonces cada nodo w_i es alcanzable desde v y por Lema 2, $\pi(w_i)$ pasa del valor ∞ a un valor acotado.

Algoritmo de Ford (1956)

Proposición 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si existe un c.l.n. alcanzable desde v entonces hay cambio de π en toda iteración de la ejecución del algoritmo de **Ford**.

Prueba: Sea C un c.l.n. alcanzable desde v y los nodos de C de acuerdo al orden de la orientación son w_1, \dots, w_q . Entonces cada nodo w_i es alcanzable desde v y por Lema 2, $\pi(w_i)$ pasa del valor ∞ a un valor acotado. Sea k la iteración a partir de la cual $\pi(w_i) < \infty$ para $1 \leq i \leq q$.

Algoritmo de Ford (1956)

Proposición 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si existe un c.l.n. alcanzable desde v entonces hay cambio de π en toda iteración de la ejecución del algoritmo de **Ford**.

Prueba: Sea C un c.l.n. alcanzable desde v y los nodos de C de acuerdo al orden de la orientación son w_1, \dots, w_q . Entonces cada nodo w_i es alcanzable desde v y por Lema 2, $\pi(w_i)$ pasa del valor ∞ a un valor acotado. Sea k la iteración a partir de la cual $\pi(w_i) < \infty$ para $1 \leq i \leq q$. Lo cual implica que hubo cambio en π en cada una de las primeras k iteraciones.

Algoritmo de Ford (1956)

Proposición 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si existe un c.l.n. alcanzable desde v entonces hay cambio de π en toda iteración de la ejecución del algoritmo de **Ford**.

Prueba: Sea C un c.l.n. alcanzable desde v y los nodos de C de acuerdo al orden de la orientación son w_1, \dots, w_q . Entonces cada nodo w_i es alcanzable desde v y por Lema 2, $\pi(w_i)$ pasa del valor ∞ a un valor acotado. Sea k la iteración a partir de la cual $\pi(w_i) < \infty$ para $1 \leq i \leq q$. Lo cual implica que hubo cambio en π en cada una de las primeras k iteraciones. Supongamos que existe una iteración $k' > k$, en la cual no hubo cambio en π .

Algoritmo de Ford (1956)

Proposición 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si existe un c.l.n. alcanzable desde v entonces hay cambio de π en toda iteración de la ejecución del algoritmo de **Ford**.

Prueba: Sea C un c.l.n. alcanzable desde v y los nodos de C de acuerdo al orden de la orientación son w_1, \dots, w_q . Entonces cada nodo w_i es alcanzable desde v y por Lema 2, $\pi(w_i)$ pasa del valor ∞ a un valor acotado. Sea k la iteración a partir de la cual $\pi(w_i) < \infty$ para $1 \leq i \leq q$. Lo cual implica que hubo cambio en π en cada una de las primeras k iteraciones. Supongamos que existe una iteración $k' > k$, en la cual no hubo cambio en π . Implicaría que no hubo cambio para $\pi(w_1), \dots, \pi(w_q)$.

Algoritmo de Ford (1956)

Proposición 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si existe un c.l.n. alcanzable desde v entonces hay cambio de π en toda iteración de la ejecución del algoritmo de **Ford**.

Prueba: Sea C un c.l.n. alcanzable desde v y los nodos de C de acuerdo al orden de la orientación son w_1, \dots, w_q . Entonces cada nodo w_i es alcanzable desde v y por Lema 2, $\pi(w_i)$ pasa del valor ∞ a un valor acotado. Sea k la iteración a partir de la cual $\pi(w_i) < \infty$ para $1 \leq i \leq q$. Lo cual implica que hubo cambio en π en cada una de las primeras k iteraciones. Supongamos que existe una iteración $k' > k$, en la cual no hubo cambio en π . Implicaría que no hubo cambio para $\pi(w_1), \dots, \pi(w_q)$. Entonces $\pi(w_1) \leq \pi(w_q) + l(w_q, w_1)$ y $\pi(w_{i+1}) \leq \pi(w_i) + l(w_i, w_{i+1})$ para $1 \leq i \leq q - 1$.

Algoritmo de Ford (1956)

Proposición 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si existe un c.l.n. alcanzable desde v entonces hay cambio de π en toda iteración de la ejecución del algoritmo de **Ford**.

Prueba: Sea C un c.l.n. alcanzable desde v y los nodos de C de acuerdo al orden de la orientación son w_1, \dots, w_q . Entonces cada nodo w_i es alcanzable desde v y por Lema 2, $\pi(w_i)$ pasa del valor ∞ a un valor acotado. Sea k la iteración a partir de la cual $\pi(w_i) < \infty$ para $1 \leq i \leq q$. Lo cual implica que hubo cambio en π en cada una de las primeras k iteraciones. Supongamos que existe una iteración $k' > k$, en la cual no hubo cambio en π . Implicaría que no hubo cambio para $\pi(w_1), \dots, \pi(w_q)$. Entonces $\pi(w_1) \leq \pi(w_q) + l(w_q, w_1)$ y $\pi(w_{i+1}) \leq \pi(w_i) + l(w_i, w_{i+1})$ para $1 \leq i \leq q - 1$. Si sumamos estas q desigualdades da
$$\sum_{i=1}^q \pi(w_i) \leq l(C) + \sum_{i=1}^q \pi(w_i).$$

Algoritmo de Ford (1956)

Proposición 1: Dado un grafo orientado G con una función de longitud l para sus aristas y un nodo origen v , si existe un c.l.n. alcanzable desde v entonces hay cambio de π en toda iteración de la ejecución del algoritmo de **Ford**.

Prueba: Sea C un c.l.n. alcanzable desde v y los nodos de C de acuerdo al orden de la orientación son w_1, \dots, w_q . Entonces cada nodo w_i es alcanzable desde v y por Lema 2, $\pi(w_i)$ pasa del valor ∞ a un valor acotado. Sea k la iteración a partir de la cual $\pi(w_i) < \infty$ para $1 \leq i \leq q$. Lo cual implica que hubo cambio en π en cada una de las primeras k iteraciones. Supongamos que existe una iteración $k' > k$, en la cual no hubo cambio en π . Implicaría que no hubo cambio para $\pi(w_1), \dots, \pi(w_q)$. Entonces $\pi(w_1) \leq \pi(w_q) + l(w_q, w_1)$ y $\pi(w_{i+1}) \leq \pi(w_i) + l(w_i, w_{i+1})$ para $1 \leq i \leq q - 1$. Si sumamos estas q desigualdades da $\sum_{i=1}^q \pi(w_i) \leq l(C) + \sum_{i=1}^q \pi(w_i)$. Absurdo porque $l(C) < 0$.

Algoritmo de Ford (1956)

- ▶ ¿Cuál es la complejidad del algoritmo de Ford?

Algoritmo de Ford (1956)

- ▶ ¿Cuál es la complejidad del algoritmo de Ford?
- ▶ ¿Qué pasa si aplicamos Ford a un grafo no orientado?

Algoritmo de Ford (1956)

- ▶ ¿Cuál es la complejidad del algoritmo de Ford?
- ▶ ¿Qué pasa si aplicamos Ford a un grafo no orientado?
- ▶ Mejora del cálculo de π

Algoritmo de Ford (1956)

- ▶ ¿Cuál es la complejidad del algoritmo de Ford?
- ▶ ¿Qué pasa si aplicamos Ford a un grafo no orientado?
- ▶ Mejora del cálculo de π
- ▶ ¿Cómo podemos modificar el algoritmo de Ford para detectar si hay ciclos de longitud negativa alcanzables desde v ?

Algoritmo de Ford (1956)

Asumimos que el grafo es orientado. Detecta si hay ciclos de longitud negativa alcanzables desde v .

$\pi(v) := 0$, $i := 0$

para todo $u \in V \setminus \{v\}$ **hacer**

$\pi(u) := \infty$

fin para

mientras hay cambios en π **e** $i < n$ **hacer**

$i := i + 1$

para todo $u \in V$ **hacer**

$\pi(u) := \min(\pi(u), \min_{(w,u) \in X} \pi(w) + l(w, u))$

fin para

fin mientras

si hay cambios en π **entonces**

retornar "Hay ciclos de longitud negativa alcanzable desde v ."

si no

retornar π

fin si

Algoritmo de Ford (1956)

1. ¿Qué modificaciones hay que hacer para que el algoritmo de Ford pueda generar explícitamente y eficientemente los caminos mínimos?

Algoritmo de Ford (1956)

1. ¿Qué modificaciones hay que hacer para que el algoritmo de Ford pueda generar explícitamente y eficientemente los caminos mínimos?
2. Idem al punto anterior pero para que pueda generar explícitamente y eficientemente un c.l.n. si hubiera alguno.

Algoritmo de Ford (1956)

1. Mantener un arreglo $pred$ como el algoritmo de Dijkstra, se actualiza $pred(u) := w$ cada vez que $\pi(u)$ es mejorado al valor de $\pi(w) + l(w, u)$.

Algoritmo de Ford (1956)

1. Mantener un arreglo $pred$ como el algoritmo de Dijkstra, se actualiza $pred(u) := w$ cada vez que $\pi(u)$ es mejorado al valor de $\pi(w) + l(w, u)$.
2. Implementar el punto (1) y en caso de detectar un c.l.n., considerar solamente las aristas $(pred(u), u)$ siempre que $pred(u)$ sea nodo del grafo (hay a lo sumo n aristas).

Algoritmo de Ford (1956)

1. Mantener un arreglo $pred$ como el algoritmo de Dijkstra, se actualiza $pred(u) := w$ cada vez que $\pi(u)$ es mejorado al valor de $\pi(w) + l(w, u)$.
2. Implementar el punto (1) y en caso de detectar un c.l.n., considerar solamente las aristas $(pred(u), u)$ siempre que $pred(u)$ sea nodo del grafo (hay a lo sumo n aristas). Aplicar DFS a partir de v , en caso de encontrar un back-edge, localizamos un ciclo y necesariamente es de longitud negativa.

Algoritmo de Ford (1956)

1. Mantener un arreglo $pred$ como el algoritmo de Dijkstra, se actualiza $pred(u) := w$ cada vez que $\pi(u)$ es mejorado al valor de $\pi(w) + l(w, u)$.
2. Implementar el punto (1) y en caso de detectar un c.l.n., considerar solamente las aristas $(pred(u), u)$ siempre que $pred(u)$ sea nodo del grafo (hay a lo sumo n aristas). Aplicar DFS a partir de v , en caso de encontrar un back-edge, localizamos un ciclo y necesariamente es de longitud negativa. Si no hay back-edge, significa que DFS devolvió un árbol con raíz v .

Algoritmo de Ford (1956)

1. Mantener un arreglo $pred$ como el algoritmo de Dijkstra, se actualiza $pred(u) := w$ cada vez que $\pi(u)$ es mejorado al valor de $\pi(w) + l(w, u)$.
2. Implementar el punto (1) y en caso de detectar un c.l.n., considerar solamente las aristas $(pred(u), u)$ siempre que $pred(u)$ sea nodo del grafo (hay a lo sumo n aristas). Aplicar DFS a partir de v , en caso de encontrar un back-edge, localizamos un ciclo y necesariamente es de longitud negativa. Si no hay back-edge, significa que DFS devolvió un árbol con raíz v . Entonces tomamos una arista $(pred(w), w)$ fuera de este árbol.

Algoritmo de Ford (1956)

1. Mantener un arreglo $pred$ como el algoritmo de Dijkstra, se actualiza $pred(u) := w$ cada vez que $\pi(u)$ es mejorado al valor de $\pi(w) + l(w, u)$.
2. Implementar el punto (1) y en caso de detectar un c.l.n., considerar solamente las aristas $(pred(u), u)$ siempre que $pred(u)$ sea nodo del grafo (hay a lo sumo n aristas). Aplicar DFS a partir de v , en caso de encontrar un back-edge, localizamos un ciclo y necesariamente es de longitud negativa. Si no hay back-edge, significa que DFS devolvió un árbol con raíz v . Entonces tomamos una arista $(pred(w), w)$ fuera de este árbol. Aplicamos DFS a partir de w pero invirtiendo las orientaciones de las aristas (en lugar de $(pred(u), u)$ es $(u, pred(u))$).

Algoritmo de Ford (1956)

1. Mantener un arreglo $pred$ como el algoritmo de Dijkstra, se actualiza $pred(u) := w$ cada vez que $\pi(u)$ es mejorado al valor de $\pi(w) + l(w, u)$.
2. Implementar el punto (1) y en caso de detectar un c.l.n., considerar solamente las aristas $(pred(u), u)$ siempre que $pred(u)$ sea nodo del grafo (hay a lo sumo n aristas). Aplicar DFS a partir de v , en caso de encontrar un back-edge, localizamos un ciclo y necesariamente es de longitud negativa. Si no hay back-edge, significa que DFS devolvió un árbol con raíz v . Entonces tomamos una arista $(pred(w), w)$ fuera de este árbol. Aplicamos DFS a partir de w pero invirtiendo las orientaciones de las aristas (en lugar de $(pred(u), u)$ es $(u, pred(u))$). Necesariamente hay un back-edge y localizamos un ciclo (volver a tomar las orientaciones originales y sigue siendo ciclo, este ciclo es de longitud negativa).

Algoritmo de Ford (1956)

Proposición 2: G tiene un c.l.n. alcanzable desde v sii el subgrafo G' de G con solamente aristas de $(pred(u), u)$ (siempre que $pred(u)$ sea nodo de G) tiene un ciclo con al menos 2 aristas.

Algoritmo de Ford (1956)

Proposición 2: G tiene un c.l.n. alcanzable desde v sii el subgrafo G' de G con solamente aristas de $(pred(u), u)$ (siempre que $pred(u)$ sea nodo de G) tiene un ciclo con al menos 2 aristas.

Prueba: Vamos a probar primero la vuelta.

Algoritmo de Ford (1956)

Proposición 2: G tiene un c.l.n. alcanzable desde v sii el subgrafo G' de G con solamente aristas de $(pred(u), u)$ (siempre que $pred(u)$ sea nodo de G) tiene un ciclo con al menos 2 aristas.

Prueba: Vamos a probar primero la vuelta. Sea C un ciclo en G' cuyos nodos son u_1, \dots, u_k , ($k \geq 2$) donde $u_i = pred(u_{i+1})$ para $1 \leq i \leq k-1$ y $u_k = pred(u_1)$.

Algoritmo de Ford (1956)

Proposición 2: G tiene un c.l.n. alcanzable desde v sii el subgrafo G' de G con solamente aristas de $(pred(u), u)$ (siempre que $pred(u)$ sea nodo de G) tiene un ciclo con al menos 2 aristas.

Prueba: Vamos a probar primero la vuelta. Sea C un ciclo en G' cuyos nodos son u_1, \dots, u_k , ($k \geq 2$) donde $u_i = pred(u_{i+1})$ para $1 \leq i \leq k-1$ y $u_k = pred(u_1)$. Llamamos, $\pi'(u_{i-1})$, el valor de π de u_{i-1} cuando se llegó al valor actual de π de u_i para $2 \leq i \leq k$, es decir que $\pi(u_i) = \pi'(u_{i-1}) + l(u_{i-1}, u_i)$ y $\pi'(u_k)$, el valor de π de u_k cuando se llegó al valor actual de π de u_1 , $\pi(u_1) = \pi'(u_k) + l(u_k, u_1)$.

Algoritmo de Ford (1956)

Proposición 2: G tiene un c.l.n. alcanzable desde v sii el subgrafo G' de G con solamente aristas de $(pred(u), u)$ (siempre que $pred(u)$ sea nodo de G) tiene un ciclo con al menos 2 aristas.

Prueba: Vamos a probar primero la vuelta. Sea C un ciclo en G' cuyos nodos son u_1, \dots, u_k , ($k \geq 2$) donde $u_i = pred(u_{i+1})$ para $1 \leq i \leq k-1$ y $u_k = pred(u_1)$. Llamamos, $\pi'(u_{i-1})$, el valor de π de u_{i-1} cuando se llegó al valor actual de π de u_i para $2 \leq i \leq k$, es decir que $\pi(u_i) = \pi'(u_{i-1}) + l(u_{i-1}, u_i)$ y $\pi'(u_k)$, el valor de π de u_k cuando se llegó al valor actual de π de u_1 , $\pi(u_1) = \pi'(u_k) + l(u_k, u_1)$. Sin pérdida de generalidad, podemos suponer que u_1 sea el último nodo de C en alcanzar su valor actual de π .

Algoritmo de Ford (1956)

Proposición 2: G tiene un c.l.n. alcanzable desde v sii el subgrafo G' de G con solamente aristas de $(pred(u), u)$ (siempre que $pred(u)$ sea nodo de G) tiene un ciclo con al menos 2 aristas.

Prueba: Vamos a probar primero la vuelta. Sea C un ciclo en G' cuyos nodos son u_1, \dots, u_k , ($k \geq 2$) donde $u_i = pred(u_{i+1})$ para $1 \leq i \leq k-1$ y $u_k = pred(u_1)$. Llamamos, $\pi'(u_{i-1})$, el valor de π de u_{i-1} cuando se llegó al valor actual de π de u_i para $2 \leq i \leq k$, es decir que $\pi(u_i) = \pi'(u_{i-1}) + l(u_{i-1}, u_i)$ y $\pi'(u_k)$, el valor de π de u_k cuando se llegó al valor actual de π de u_1 , $\pi(u_1) = \pi'(u_k) + l(u_k, u_1)$. Sin pérdida de generalidad, podemos suponer que u_1 sea el último nodo de C en alcanzar su valor actual de π . Llamamos $\pi''(u_1)$ el valor previo.

Algoritmo de Ford (1956)

Proposición 2: G tiene un c.l.n. alcanzable desde v sii el subgrafo G' de G con solamente aristas de $(pred(u), u)$ (siempre que $pred(u)$ sea nodo de G) tiene un ciclo con al menos 2 aristas.

Prueba: Vamos a probar primero la vuelta. Sea C un ciclo en G' cuyos nodos son u_1, \dots, u_k , ($k \geq 2$) donde $u_i = pred(u_{i+1})$ para $1 \leq i \leq k-1$ y $u_k = pred(u_1)$. Llamamos, $\pi'(u_{i-1})$, el valor de π de u_{i-1} cuando se llegó al valor actual de π de u_i para $2 \leq i \leq k$, es decir que $\pi(u_i) = \pi'(u_{i-1}) + l(u_{i-1}, u_i)$ y $\pi'(u_k)$, el valor de π de u_k cuando se llegó al valor actual de π de u_1 , $\pi(u_1) = \pi'(u_k) + l(u_k, u_1)$. Sin pérdida de generalidad, podemos suponer que u_1 sea el último nodo de C en alcanzar su valor actual de π . Llamamos $\pi''(u_1)$ el valor previo. Claramente, $\pi(u_1) < \pi''(u_1) \leq \pi'(u_1)$ y $\pi(u_i) \leq \pi'(u_i)$ en general.

Algoritmo de Ford (1956)

Proposición 2: G tiene un c.l.n. alcanzable desde v sii el subgrafo G' de G con solamente aristas de $(pred(u), u)$ (siempre que $pred(u)$ sea nodo de G) tiene un ciclo con al menos 2 aristas.

Prueba: Vamos a probar primero la vuelta. Sea C un ciclo en G' cuyos nodos son u_1, \dots, u_k , ($k \geq 2$) donde $u_i = pred(u_{i+1})$ para $1 \leq i \leq k-1$ y $u_k = pred(u_1)$. Llamamos, $\pi'(u_{i-1})$, el valor de π de u_{i-1} cuando se llegó al valor actual de π de u_i para $2 \leq i \leq k$, es decir que $\pi(u_i) = \pi'(u_{i-1}) + l(u_{i-1}, u_i)$ y $\pi'(u_k)$, el valor de π de u_k cuando se llegó al valor actual de π de u_1 , $\pi(u_1) = \pi'(u_k) + l(u_k, u_1)$. Sin pérdida de generalidad, podemos suponer que u_1 sea el último nodo de C en alcanzar su valor actual de π . Llamamos $\pi''(u_1)$ el valor previo. Claramente,

$$\pi(u_1) < \pi''(u_1) \leq \pi'(u_1) \text{ y } \pi(u_i) \leq \pi'(u_i) \text{ en general. Entonces}$$
$$\sum_{i=1}^k \pi(u_i) = l(C) + \sum_{i=1}^k \pi'(u_i) \text{ y}$$
$$l(C) = \sum_{i=1}^k \pi(u_i) - \sum_{i=1}^k \pi'(u_i) < 0.$$

Prueba de Proposición 2: Continuación

Ahora probamos la ida.

Prueba de Proposición 2: Continuación

Ahora probamos la ida. Sea G^* el subgrafo de G' con los nodos alcanzables desde v .

Prueba de Proposición 2: Continuación

Ahora probamos la ida. Sea G^* el subgrafo de G' con los nodos alcanzables desde v . Por los resultados anteriores, w está en G^* si $\pi(w) < \infty$ que es equivalente decir que hay una arista de G' que llega a w o $w = v$.

Prueba de Proposición 2: Continuación

Ahora probamos la ida. Sea G^* el subgrafo de G' con los nodos alcanzables desde v . Por los resultados anteriores, w está en G^* si $\pi(w) < \infty$ que es equivalente decir que hay una arista de G' que llega a w o $w = v$. Sean n^* y m^* , la cantidad de nodos y la cantidad de aristas de G^* , respectivamente.

Prueba de Proposición 2: Continuación

Ahora probamos la ida. Sea G^* el subgrafo de G' con los nodos alcanzables desde v . Por los resultados anteriores, w está en G^* si $\pi(w) < \infty$ que es equivalente decir que hay una arista de G' que llega a w o $w = v$. Sean n^* y m^* , la cantidad de nodos y la cantidad de aristas de G^* , respectivamente. Claramente, el grado de entrada de todo nodo de G^* es exactamente 1 salvo v que puede ser 1 u 0.

Prueba de Proposición 2: Continuación

Ahora probamos la ida. Sea G^* el subgrafo de G' con los nodos alcanzables desde v . Por los resultados anteriores, w está en G^* si $\pi(w) < \infty$ que es equivalente decir que hay una arista de G' que llega a w o $w = v$. Sean n^* y m^* , la cantidad de nodos y la cantidad de aristas de G^* , respectivamente. Claramente, el grado de entrada de todo nodo de G^* es exactamente 1 salvo v que puede ser 1 u 0. Por lo tanto, $m^* = n^*$ o $m^* = n^* - 1$.

Prueba de Proposición 2: Continuación

Ahora probamos la ida. Sea G^* el subgrafo de G' con los nodos alcanzables desde v . Por los resultados anteriores, w está en G^* si $\pi(w) < \infty$ que es equivalente decir que hay una arista de G' que llega a w o $w = v$. Sean n^* y m^* , la cantidad de nodos y la cantidad de aristas de G^* , respectivamente. Claramente, el grado de entrada de todo nodo de G^* es exactamente 1 salvo v que puede ser 1 u 0. Por lo tanto, $m^* = n^*$ o $m^* = n^* - 1$.

► Si $m^* = n^*$, entonces el grado de entrada de v es 1.

Prueba de Proposición 2: Continuación

Ahora probamos la ida. Sea G^* el subgrafo de G' con los nodos alcanzables desde v . Por los resultados anteriores, w está en G^* si $\pi(w) < \infty$ que es equivalente decir que hay una arista de G' que llega a w o $w = v$. Sean n^* y m^* , la cantidad de nodos y la cantidad de aristas de G^* , respectivamente. Claramente, el grado de entrada de todo nodo de G^* es exactamente 1 salvo v que puede ser 1 u 0. Por lo tanto, $m^* = n^*$ o $m^* = n^* - 1$.

- Si $m^* = n^*$, entonces el grado de entrada de v es 1. Tomemos cualquier nodo w_0 de G^* (en particular puede ser v).

Prueba de Proposición 2: Continuación

Ahora probamos la ida. Sea G^* el subgrafo de G' con los nodos alcanzables desde v . Por los resultados anteriores, w está en G^* si $\pi(w) < \infty$ que es equivalente decir que hay una arista de G' que llega a w o $w = v$. Sean n^* y m^* , la cantidad de nodos y la cantidad de aristas de G^* , respectivamente. Claramente, el grado de entrada de todo nodo de G^* es exactamente 1 salvo v que puede ser 1 u 0. Por lo tanto, $m^* = n^*$ o $m^* = n^* - 1$.

- Si $m^* = n^*$, entonces el grado de entrada de v es 1. Tomemos cualquier nodo w_0 de G^* (en particular puede ser v). Aplicar iterativamente en cada paso i , $w_i = \text{pred}(w_{i-1})$, hasta que encuentre uno visitado anteriormente y de esta forma hallamos un ciclo en G' .

Prueba de Proposición 2: Continuación 2

- Si $m^* = n^* - 1$, entonces el grado de entrada de v es 0.

Prueba de Proposición 2: Continuación 2

- Si $m^* = n^* - 1$, entonces el grado de entrada de v es 0. Consideramos el grafo subyacente de G^* , veamos que no es conexo.

Prueba de Proposición 2: Continuación 2

- Si $m^* = n^* - 1$, entonces el grado de entrada de v es 0. Consideramos el grafo subyacente de G^* , veamos que no es conexo. Supongamos que es conexo. Entonces G^* es un árbol T con raíz en v con altura a lo sumo $n^* - 1 \leq n - 1$.

Prueba de Proposición 2: Continuación 2

- Si $m^* = n^* - 1$, entonces el grado de entrada de v es 0. Consideramos el grafo subyacente de G^* , veamos que no es conexo. Supongamos que es conexo. Entonces G^* es un árbol T con raíz en v con altura a lo sumo $n^* - 1 \leq n - 1$. Por lo tanto en a lo sumo $n^* - 1$ iteraciones del algoritmo, todos los nodos de G (los que están en G^* y los que no) alcanzan sus valores de π actuales.

Prueba de Proposición 2: Continuación 2

- Si $m^* = n^* - 1$, entonces el grado de entrada de v es 0. Consideramos el grafo subyacente de G^* , veamos que no es conexo. Supongamos que es conexo. Entonces G^* es un árbol T con raíz en v con altura a lo sumo $n^* - 1 \leq n - 1$. Por lo tanto en a lo sumo $n^* - 1$ iteraciones del algoritmo, todos los nodos de G (los que están en G^* y los que no) alcanzan sus valores de π actuales. Es claro que cualquier nodo u de T , $\pi(u) = l(C_{v,u}^T)$ donde $C_{v,u}^T$ es el camino en T entre v y u (se prueba por inducción en el nivel de u en T).

Prueba de Proposición 2: Continuación 2

- Si $m^* = n^* - 1$, entonces el grado de entrada de v es 0. Consideramos el grafo subyacente de G^* , veamos que no es conexo. Supongamos que es conexo. Entonces G^* es un árbol T con raíz en v con altura a lo sumo $n^* - 1 \leq n - 1$. Por lo tanto en a lo sumo $n^* - 1$ iteraciones del algoritmo, todos los nodos de G (los que están en G^* y los que no) alcanzan sus valores de π actuales. Es claro que cualquier nodo u de T , $\pi(u) = l(C_{v,u}^T)$ donde $C_{v,u}^T$ es el camino en T entre v y u (se prueba por inducción en el nivel de u en T). Entonces no pudo haber mejora entre las iteraciones n^* y n . Contradicción.

Prueba de Proposición 2: Continuación 2

- Si $m^* = n^* - 1$, entonces el grado de entrada de v es 0. Consideramos el grafo subyacente de G^* , veamos que no es conexo. Supongamos que es conexo. Entonces G^* es un árbol T con raíz en v con altura a lo sumo $n^* - 1 \leq n - 1$. Por lo tanto en a lo sumo $n^* - 1$ iteraciones del algoritmo, todos los nodos de G (los que están en G^* y los que no) alcanzan sus valores de π actuales. Es claro que cualquier nodo u de T , $\pi(u) = l(C_{v,u}^T)$ donde $C_{v,u}^T$ es el camino en T entre v y u (se prueba por inducción en el nivel de u en T). Entonces no pudo haber mejora entre las iteraciones n^* y n . Contradicción. El grafo subyacente de G^* tiene varias componentes conexas.

Prueba de Proposición 2: Continuación 2

- Si $m^* = n^* - 1$, entonces el grado de entrada de v es 0. Consideramos el grafo subyacente de G^* , veamos que no es conexo. Supongamos que es conexo. Entonces G^* es un árbol T con raíz en v con altura a lo sumo $n^* - 1 \leq n - 1$. Por lo tanto en a lo sumo $n^* - 1$ iteraciones del algoritmo, todos los nodos de G (los que están en G^* y los que no) alcanzan sus valores de π actuales. Es claro que cualquier nodo u de T , $\pi(u) = l(C_{v,u}^T)$ donde $C_{v,u}^T$ es el camino en T entre v y u (se prueba por inducción en el nivel de u en T). Entonces no pudo haber mejora entre las iteraciones n^* y n . Contradicción. El grafo subyacente de G^* tiene varias componentes conexas. Tomemos una que no contenga a v .

Prueba de Proposición 2: Continuación 2

- Si $m^* = n^* - 1$, entonces el grado de entrada de v es 0. Consideramos el grafo subyacente de G^* , veamos que no es conexo. Supongamos que es conexo. Entonces G^* es un árbol T con raíz en v con altura a lo sumo $n^* - 1 \leq n - 1$. Por lo tanto en a lo sumo $n^* - 1$ iteraciones del algoritmo, todos los nodos de G (los que están en G^* y los que no) alcanzan sus valores de π actuales. Es claro que cualquier nodo u de T , $\pi(u) = l(C_{v,u}^T)$ donde $C_{v,u}^T$ es el camino en T entre v y u (se prueba por inducción en el nivel de u en T). Entonces no pudo haber mejora entre las iteraciones n^* y n . Contradicción. El grafo subyacente de G^* tiene varias componentes conexas. Tomemos una que no contenga a v . Todos los nodos de esa componente conexa tienen grado de entrada exactamente 1 que es similar al caso anterior donde todos los nodos de G^* tienen grado de entrada 1.

Prueba de Proposición 2: Continuación 2

- Si $m^* = n^* - 1$, entonces el grado de entrada de v es 0. Consideramos el grafo subyacente de G^* , veamos que no es conexo. Supongamos que es conexo. Entonces G^* es un árbol T con raíz en v con altura a lo sumo $n^* - 1 \leq n - 1$. Por lo tanto en a lo sumo $n^* - 1$ iteraciones del algoritmo, todos los nodos de G (los que están en G^* y los que no) alcanzan sus valores de π actuales. Es claro que cualquier nodo u de T , $\pi(u) = l(C_{v,u}^T)$ donde $C_{v,u}^T$ es el camino en T entre v y u (se prueba por inducción en el nivel de u en T). Entonces no pudo haber mejora entre las iteraciones n^* y n . Contradicción. El grafo subyacente de G^* tiene varias componentes conexas. Tomemos una que no contenga a v . Todos los nodos de esa componente conexa tienen grado de entrada exactamente 1 que es similar al caso anterior donde todos los nodos de G^* tienen grado de entrada 1. Para encontrar el ciclo, basta aplicar iterativamente *pred* desde cualquier nodo de esa componente conexa.

Algoritmo de Ford (1956)

Corolario 3: Después de aplicar el algoritmo de Ford (última versión) a un grafo pesado G y un nodo origen v .

1. $dist(v, w) = \infty$ sii $\pi(w) = \infty$ (w es inalcanzable desde v , equivalentemente, w no está en G^*).

Algoritmo de Ford (1956)

Corolario 3: Después de aplicar el algoritmo de Ford (última versión) a un grafo pesado G y un nodo origen v .

1. $dist(v, w) = \infty$ sii $\pi(w) = \infty$ (w es inalcanzable desde v , equivalentemente, w no está en G^*).
2. $dist(v, w) = -\infty$ (no existe un recorrido/camino mínimo entre v y w) sii $\pi(w) < \infty$ (equivalentemente, w está en G^*) y cumpla una de las siguientes condiciones,
 - 2.1 $pred(v)$ es un nodo de G

Algoritmo de Ford (1956)

Corolario 3: Después de aplicar el algoritmo de Ford (última versión) a un grafo pesado G y un nodo origen v .

1. $dist(v, w) = \infty$ sii $\pi(w) = \infty$ (w es inalcanzable desde v , equivalentemente, w no está en G^*).
2. $dist(v, w) = -\infty$ (no existe un recorrido/camino mínimo entre v y w) sii $\pi(w) < \infty$ (equivalentemente, w está en G^*) y cumpla una de las siguientes condiciones,
 - 2.1 $pred(v)$ es un nodo de G
 - 2.2 w es inalcanzable desde v en G^* (v y w están en diferentes componentes conexas del grafo subyacente de G^*)

Algoritmo de Ford (1956)

Corolario 3: Después de aplicar el algoritmo de Ford (última versión) a un grafo pesado G y un nodo origen v .

1. $dist(v, w) = \infty$ sii $\pi(w) = \infty$ (w es inalcanzable desde v , equivalentemente, w no está en G^*).
2. $dist(v, w) = -\infty$ (no existe un recorrido/camino mínimo entre v y w) sii $\pi(w) < \infty$ (equivalentemente, w está en G^*) y cumpla una de las siguientes condiciones,
 - 2.1 $pred(v)$ es un nodo de G
 - 2.2 w es inalcanzable desde v en G^* (v y w están en diferentes componentes conexas del grafo subyacente de G^*)
 - 2.3 w es alcanzable en G desde un nodo $u \neq w$ tal que $dist(v, u) = -\infty$

Implementación Práctica:

1. Trivial. $O(n)$

Implementación Práctica:

1. Trivial. $O(n)$
2. Buscar los nodos w de G^* tal que $dist(v, w) = -\infty$.

Implementación Práctica:

1. Trivial. $O(n)$
2. Buscar los nodos w de G^* tal que $dist(v, w) = -\infty$.
 - 2.1 Si $pred(v) \neq \infty$ entonces cualquier nodo w de G^*
 $dist(v, w) = -\infty$

Implementación Práctica:

1. Trivial. $O(n)$
2. Buscar los nodos w de G^* tal que $dist(v, w) = -\infty$.
 - 2.1 Si $pred(v) \neq \infty$ entonces cualquier nodo w de G^* $dist(v, w) = -\infty$ ya que w está en un c.l.n. alcanzable desde v o es alcanzable desde un nodo que sí está en un ciclo de estas características (aplicando iterativamente la función $pred$ para hallarlo).

Implementación Práctica:

1. Trivial. $O(n)$
2. Buscar los nodos w de G^* tal que $dist(v, w) = -\infty$.
 - 2.1 Si $pred(v) \neq \infty$ entonces cualquier nodo w de G^* $dist(v, w) = -\infty$ ya que w está en un c.l.n. alcanzable desde v o es alcanzable desde un nodo que sí está en un ciclo de estas características (aplicando iterativamente la función $pred$ para hallarlo).
 - 2.2 Si $pred(v) = \infty$, consideremos el grafo subyacente de G^* .

Implementación Práctica:

1. Trivial. $O(n)$
2. Buscar los nodos w de G^* tal que $dist(v, w) = -\infty$.
 - 2.1 Si $pred(v) \neq \infty$ entonces cualquier nodo w de G^* $dist(v, w) = -\infty$ ya que w está en un c.l.n. alcanzable desde v o es alcanzable desde un nodo que sí está en un ciclo de estas características (aplicando iterativamente la función $pred$ para hallarlo).
 - 2.2 Si $pred(v) = \infty$, consideremos el grafo subyacente de G^* . Los nodos de la componente conexa donde está v , induce en G^* un árbol T con v como raíz.

Implementación Práctica:

1. Trivial. $O(n)$
2. Buscar los nodos w de G^* tal que $dist(v, w) = -\infty$.
 - 2.1 Si $pred(v) \neq \infty$ entonces cualquier nodo w de G^* $dist(v, w) = -\infty$ ya que w está en un c.l.n. alcanzable desde v o es alcanzable desde un nodo que sí está en un ciclo de estas características (aplicando iterativamente la función $pred$ para hallarlo).
 - 2.2 Si $pred(v) = \infty$, consideremos el grafo subyacente de G^* . Los nodos de la componente conexa donde está v , induce en G^* un árbol T con v como raíz. Cualquier nodo w fuera de T , está en un c.l.n. alcanzable desde v o es alcanzable desde otro nodo que sí lo está (aplicando $pred$ iterativamente).

Implementación Práctica:

1. Trivial. $O(n)$
2. Buscar los nodos w de G^* tal que $dist(v, w) = -\infty$.
 - 2.1 Si $pred(v) \neq \infty$ entonces cualquier nodo w de G^* $dist(v, w) = -\infty$ ya que w está en un c.l.n. alcanzable desde v o es alcanzable desde un nodo que sí está en un ciclo de estas características (aplicando iterativamente la función $pred$ para hallarlo).
 - 2.2 Si $pred(v) = \infty$, consideremos el grafo subyacente de G^* . Los nodos de la componente conexa donde está v , induce en G^* un árbol T con v como raíz. Cualquier nodo w fuera de T , está en un c.l.n. alcanzable desde v o es alcanzable desde otro nodo que sí lo está (aplicando $pred$ iterativamente). Entonces, $dist(v, w) = -\infty$.

Implementación Práctica:

1. Trivial. $O(n)$
2. Buscar los nodos w de G^* tal que $dist(v, w) = -\infty$.
 - 2.1 Si $pred(v) \neq \infty$ entonces cualquier nodo w de G^* $dist(v, w) = -\infty$ ya que w está en un c.l.n. alcanzable desde v o es alcanzable desde un nodo que sí está en un ciclo de estas características (aplicando iterativamente la función $pred$ para hallarlo).
 - 2.2 Si $pred(v) = \infty$, consideremos el grafo subyacente de G^* . Los nodos de la componente conexa donde está v , induce en G^* un árbol T con v como raíz. Cualquier nodo w fuera de T , está en un c.l.n. alcanzable desde v o es alcanzable desde otro nodo que sí lo está (aplicando $pred$ iterativamente). Entonces, $dist(v, w) = -\infty$. Usar DFS desde v para calcular T con aristas de G^* . $O(n)$

Implementación Práctica:

1. Trivial. $O(n)$
2. Buscar los nodos w de G^* tal que $dist(v, w) = -\infty$.
 - 2.1 Si $pred(v) \neq \infty$ entonces cualquier nodo w de G^* $dist(v, w) = -\infty$ ya que w está en un c.l.n. alcanzable desde v o es alcanzable desde un nodo que sí está en un ciclo de estas características (aplicando iterativamente la función $pred$ para hallarlo).
 - 2.2 Si $pred(v) = \infty$, consideremos el grafo subyacente de G^* . Los nodos de la componente conexa donde está v , induce en G^* un árbol T con v como raíz. Cualquier nodo w fuera de T , está en un c.l.n. alcanzable desde v o es alcanzable desde otro nodo que sí lo está (aplicando $pred$ iterativamente). Entonces, $dist(v, w) = -\infty$. Usar DFS desde v para calcular T con aristas de G^* . $O(n)$
 - 2.3 Para hallar los nodos w de T tal que $dist(v, w) = -\infty$,

Implementación Práctica:

1. Trivial. $O(n)$
2. Buscar los nodos w de G^* tal que $dist(v, w) = -\infty$.
 - 2.1 Si $pred(v) \neq \infty$ entonces cualquier nodo w de G^* $dist(v, w) = -\infty$ ya que w está en un c.l.n. alcanzable desde v o es alcanzable desde un nodo que sí está en un ciclo de estas características (aplicando iterativamente la función $pred$ para hallarlo).
 - 2.2 Si $pred(v) = \infty$, consideremos el grafo subyacente de G^* . Los nodos de la componente conexa donde está v , induce en G^* un árbol T con v como raíz. Cualquier nodo w fuera de T , está en un c.l.n. alcanzable desde v o es alcanzable desde otro nodo que sí lo está (aplicando $pred$ iterativamente). Entonces, $dist(v, w) = -\infty$. Usar DFS desde v para calcular T con aristas de G^* . $O(n)$
 - 2.3 Para hallar los nodos w de T tal que $dist(v, w) = -\infty$, hay que ver cuáles nodos de T son alcanzables en G desde nodos de G^* fuera de T .

Implementación Práctica:

1. Trivial. $O(n)$
2. Buscar los nodos w de G^* tal que $\text{dist}(v, w) = -\infty$.
 - 2.1 Si $\text{pred}(v) \neq \infty$ entonces cualquier nodo w de G^* $\text{dist}(v, w) = -\infty$ ya que w está en un c.l.n. alcanzable desde v o es alcanzable desde un nodo que sí está en un ciclo de estas características (aplicando iterativamente la función pred para hallarlo).
 - 2.2 Si $\text{pred}(v) = \infty$, consideremos el grafo subyacente de G^* . Los nodos de la componente conexa donde está v , induce en G^* un árbol T con v como raíz. Cualquier nodo w fuera de T , está en un c.l.n. alcanzable desde v o es alcanzable desde otro nodo que sí lo está (aplicando pred iterativamente). Entonces, $\text{dist}(v, w) = -\infty$. Usar DFS desde v para calcular T con aristas de G^* . $O(n)$
 - 2.3 Para hallar los nodos w de T tal que $\text{dist}(v, w) = -\infty$, hay que ver cuáles nodos de T son alcanzables en G desde nodos de G^* fuera de T . Esta condición también es necesaria ya que los valores de π de nodos de T no sufrieron cambios en la iteración n .

Implementación Práctica:

1. Trivial. $O(n)$
2. Buscar los nodos w de G^* tal que $\text{dist}(v, w) = -\infty$.
 - 2.1 Si $\text{pred}(v) \neq \infty$ entonces cualquier nodo w de G^* $\text{dist}(v, w) = -\infty$ ya que w está en un c.l.n. alcanzable desde v o es alcanzable desde un nodo que sí está en un ciclo de estas características (aplicando iterativamente la función pred para hallarlo).
 - 2.2 Si $\text{pred}(v) = \infty$, consideremos el grafo subyacente de G^* . Los nodos de la componente conexa donde está v , induce en G^* un árbol T con v como raíz. Cualquier nodo w fuera de T , está en un c.l.n. alcanzable desde v o es alcanzable desde otro nodo que sí lo está (aplicando pred iterativamente). Entonces, $\text{dist}(v, w) = -\infty$. Usar DFS desde v para calcular T con aristas de G^* . $O(n)$
 - 2.3 Para hallar los nodos w de T tal que $\text{dist}(v, w) = -\infty$, hay que ver cuáles nodos de T son alcanzables en G desde nodos de G^* fuera de T . Esta condición también es necesaria ya que los valores de π de nodos de T no sufrieron cambios en la iteración n . Para que vuelva a cambiar $\pi(w)$ para algún nodo

Implementación Práctica: Continuación

w de T , ese cambio debe ser causado (directamente o no) por el cambio de $\pi(u)$ de un nodo u de G^* fuera de T y en tal caso u alcanza a w en G .

Implementación Práctica: Continuación

w de T , ese cambio debe ser causado (directamente o no) por el cambio de $\pi(u)$ de un nodo u de G^* fuera de T y en tal caso u alcanza a w en G . Para simplificar este cálculo, agregamos un nodo nuevo v^* y agregar una arista (v^*, u) para cada nodo u de G^* fuera de T .

Implementación Práctica: Continuación

w de T , ese cambio debe ser causado (directamente o no) por el cambio de $\pi(u)$ de un nodo u de G^* fuera de T y en tal caso u alcanza a w en G . Para simplificar este cálculo, agregamos un nodo nuevo v^* y agregar una arista (v^*, u) para cada nodo u de G^* fuera de T . Este grafo resultante tiene un nodo extra y a lo sumo $n - 1$ aristas más que G en comparación.

Implementación Práctica: Continuación

w de T , ese cambio debe ser causado (directamente o no) por el cambio de $\pi(u)$ de un nodo u de G^* fuera de T y en tal caso u alcanza a w en G . Para simplificar este cálculo, agregamos un nodo nuevo v^* y agregar una arista (v^*, u) para cada nodo u de G^* fuera de T . Este grafo resultante tiene un nodo extra y a lo sumo $n - 1$ aristas más que G en comparación. Luego, aplicar DFS desde v^* para ver qué nodos de T se pueden alcanzar y esos nodos son los que buscamos. $O(m + n)$

Algoritmos matriciales

Sea $G = (\{1, \dots, n\}, X)$ un digrafo y $l : X \rightarrow R$ una función de longitud/peso para las aristas de G . Definimos las siguientes matrices:

Algoritmos matriciales

Sea $G = (\{1, \dots, n\}, X)$ un digrafo y $l : X \rightarrow R$ una función de longitud/peso para las aristas de G . Definimos las siguientes matrices:

► $L \in R^{n \times n}$, donde los elementos l_{ij} de L se definen como:

$$l_{ij} = \begin{cases} 0 & \text{si } i = j \\ l(i \rightarrow j) & \text{si } i \rightarrow j \in X \\ \infty & \text{si } i \rightarrow j \notin X \end{cases}$$

Algoritmos matriciales

Sea $G = (\{1, \dots, n\}, X)$ un digrafo y $l : X \rightarrow R$ una función de longitud/peso para las aristas de G . Definimos las siguientes matrices:

- $L \in R^{n \times n}$, donde los elementos l_{ij} de L se definen como:

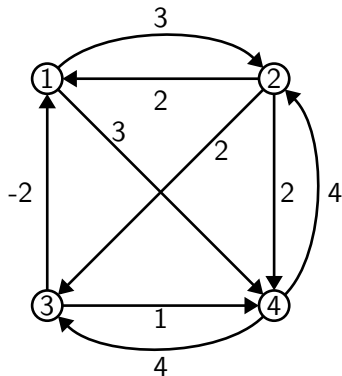
$$l_{ij} = \begin{cases} 0 & \text{si } i = j \\ l(i \rightarrow j) & \text{si } i \rightarrow j \in X \\ \infty & \text{si } i \rightarrow j \notin X \end{cases}$$

- $D \in R^{n \times n}$, donde los elementos d_{ij} de D se definen como:

$$d_{ij} = \begin{cases} \text{longitud del camino mínimo orientado de } i \text{ a } j & \text{si existe alguno} \\ \infty & \text{si no} \end{cases}$$

D es llamada matriz de distancias de G .

Algoritmos matriciales



$L =$

| | 1 | 2 | 3 | 4 |
|---|----------|----------|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | ∞ | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

Algoritmo de Floyd (1962)



Robert Floyd (1936–2001)

Algoritmo de Floyd (1962)

Llamamos v_1, \dots, v_n a los nodos de G . El algoritmo de Floyd se basa en lo siguiente:

Algoritmo de Floyd (1962)

Llamamos v_1, \dots, v_n a los nodos de G . El algoritmo de Floyd se basa en lo siguiente:

1. Si $L^0 = L$ y calculamos L^1 como

$$l_{ij}^1 = \min(l_{ij}^0, l_{i1}^0 + l_{1j}^0)$$

l_{ij}^1 es la longitud de un camino mínimo de i a j con nodo intermedio v_1 o directo.

Algoritmo de Floyd (1962)

Llamamos v_1, \dots, v_n a los nodos de G . El algoritmo de Floyd se basa en lo siguiente:

1. Si $L^0 = L$ y calculamos L^1 como

$$l_{ij}^1 = \min(l_{ij}^0, l_{i1}^0 + l_{1j}^0)$$

l_{ij}^1 es la longitud de un camino mínimo de i a j con nodo intermedio v_1 o directo.

2. Si calculamos L^k a partir de L^{k-1} como

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

l_{ij}^k es la longitud de un camino mínimo de i a j cuyos nodos intermedios están en $\{v_1, \dots, v_k\}$.

Algoritmo de Floyd (1962)

Llamamos v_1, \dots, v_n a los nodos de G . El algoritmo de Floyd se basa en lo siguiente:

1. Si $L^0 = L$ y calculamos L^1 como

$$l_{ij}^1 = \min(l_{ij}^0, l_{i1}^0 + l_{1j}^0)$$

l_{ij}^1 es la longitud de un camino mínimo de i a j con nodo intermedio v_1 o directo.

2. Si calculamos L^k a partir de L^{k-1} como

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

l_{ij}^k es la longitud de un camino mínimo de i a j cuyos nodos intermedios están en $\{v_1, \dots, v_k\}$.

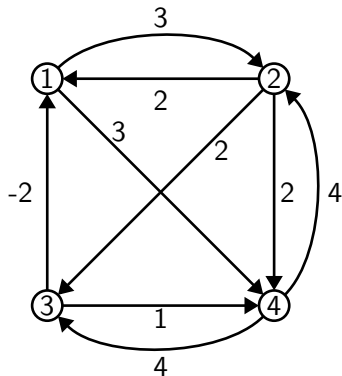
3. $D = L^n$

Algoritmo de Floyd (1962)

Asumimos que el grafo es orientado y que no hay ciclos de longitud negativa.

```
 $L^0 := L$   
para  $k$  desde 1 a  $n$  hacer  
  para  $i$  desde 1 a  $n$  hacer  
    para  $j$  desde 1 a  $n$  hacer  
       $l_{ij}^k := \min(l_{ij}^{k-1}, l_{ik}^{k-1} + l_{kj}^{k-1})$   
    fin para  
  fin para  
fin para  
retornar  $L^n$ 
```

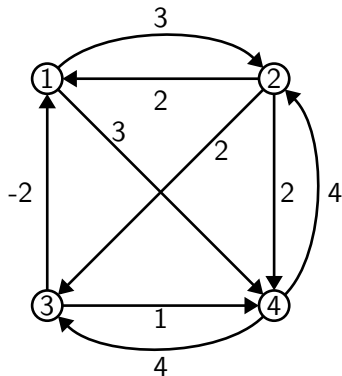
Algoritmo de Floyd (1962) - Ejemplo



$L^0 =$

| | 1 | 2 | 3 | 4 |
|---|----------|----------|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | ∞ | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

Algoritmo de Floyd (1962) - Ejemplo



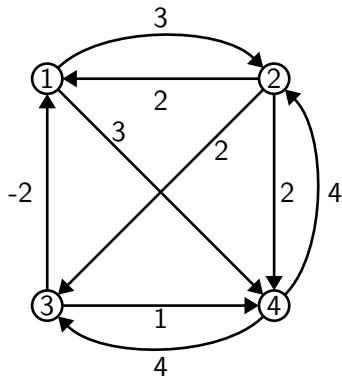
$L^0 =$

| | 1 | 2 | 3 | 4 |
|---|----------|----------|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | ∞ | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



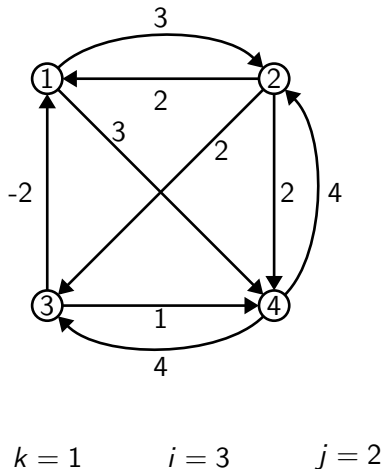
$L^0 =$

| | 1 | 2 | 3 | 4 |
|---|----------|----------|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | ∞ | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|----|---|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | | | |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



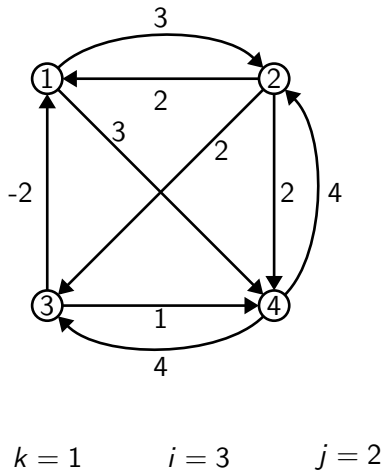
$L^0 =$

| | 1 | 2 | 3 | 4 |
|---|----------|----------|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | ∞ | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|----|---|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | | | |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



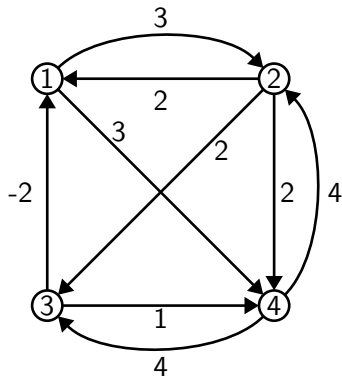
$L^0 =$

| | 1 | 2 | 3 | 4 |
|---|----------|----------|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | ∞ | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|----|---|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | 1 | | |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



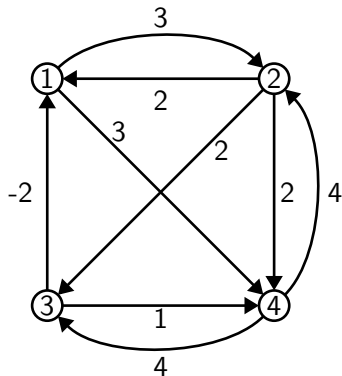
$L^0 =$

| | 1 | 2 | 3 | 4 |
|---|----------|----------|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | ∞ | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|----------|---|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

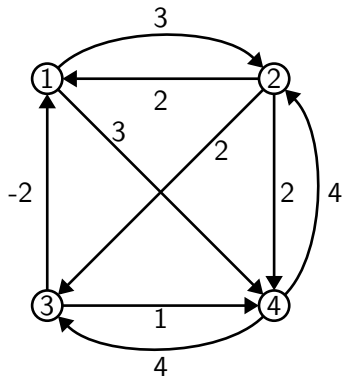
Algoritmo de Floyd (1962) - Ejemplo



$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|----------|---|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

Algoritmo de Floyd (1962) - Ejemplo



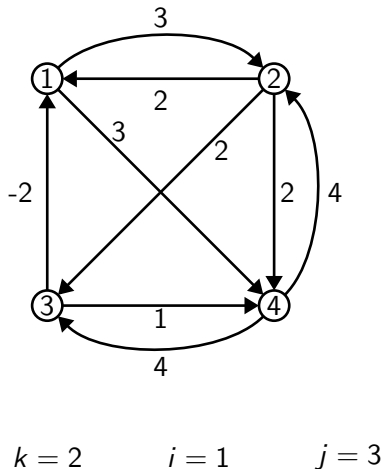
$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|----------|---|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

$L^2 =$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 3 | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



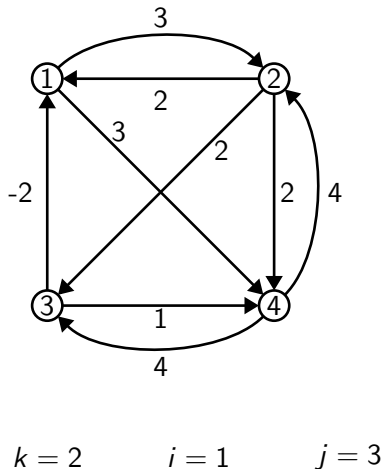
$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|----------|---|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

$L^2 =$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 3 | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



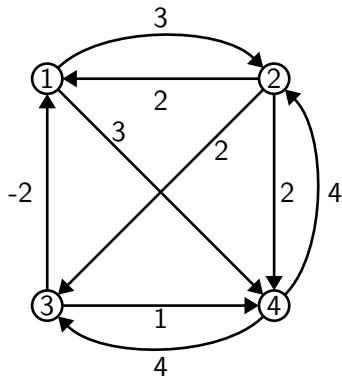
$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|----------|---|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

$L^2 =$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 3 | 5 | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



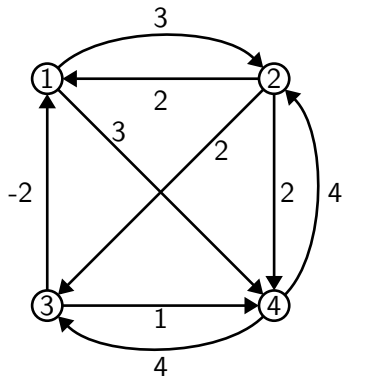
$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|----------|---|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

$L^2 =$

| | 1 | 2 | 3 | 4 |
|---|----|---|---|---|
| 1 | 0 | 3 | 5 | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



$k = 2$ $i = 4$ $j = 1$

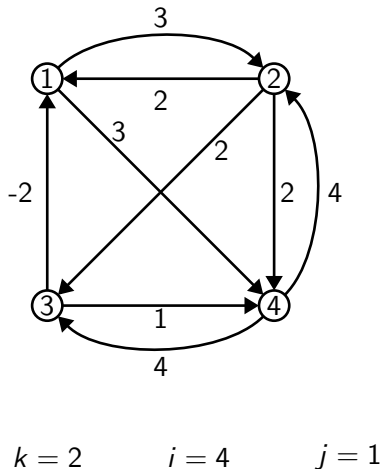
$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|----------|---|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

$L^2 =$

| | 1 | 2 | 3 | 4 |
|---|----|---|---|---|
| 1 | 0 | 3 | 5 | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



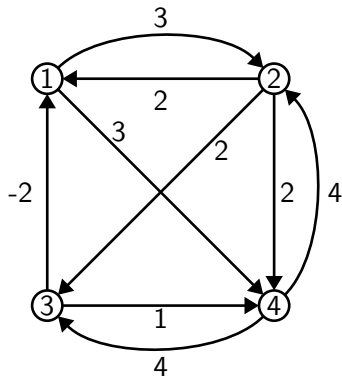
$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|----------|---|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

$L^2 =$

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

Algoritmo de Floyd (1962) - Ejemplo



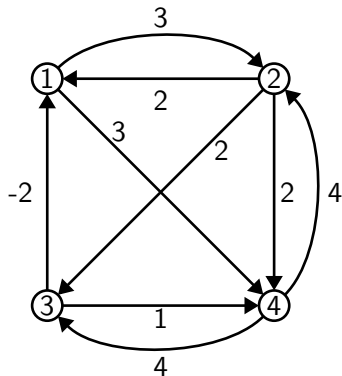
$L^1 =$

| | 1 | 2 | 3 | 4 |
|---|----------|---|----------|---|
| 1 | 0 | 3 | ∞ | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | ∞ | 4 | 4 | 0 |

$L^2 =$

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

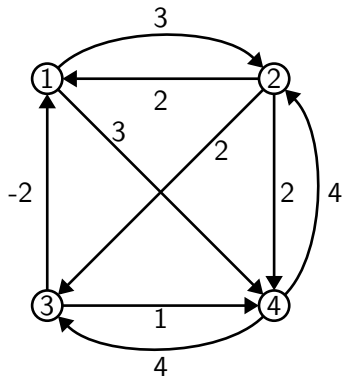
Algoritmo de Floyd (1962) - Ejemplo



$L^2 =$

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

Algoritmo de Floyd (1962) - Ejemplo



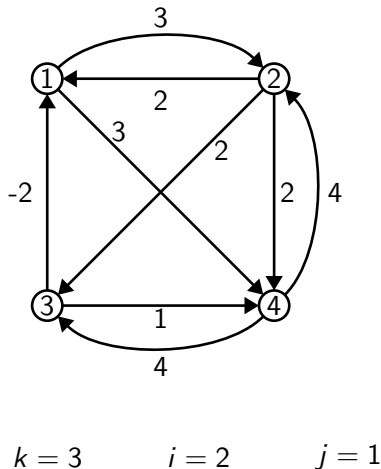
$L^2 =$

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

$L^3 =$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 3 | 5 | 3 |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



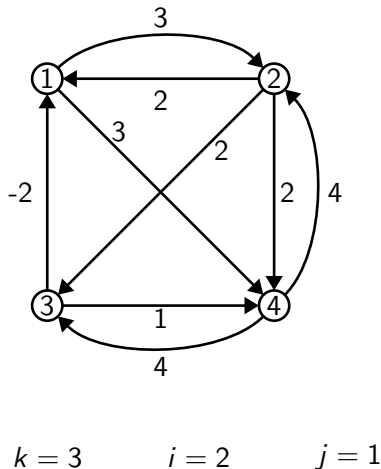
$L^2 =$

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

$L^3 =$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 3 | 5 | 3 |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



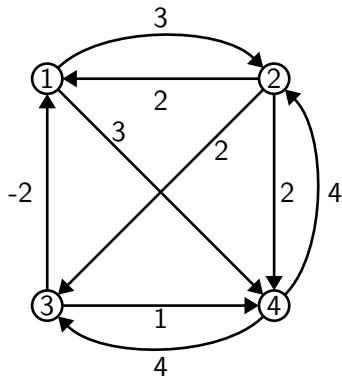
$L^2 =$

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

$L^3 =$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 3 | 5 | 3 |
| 2 | 0 | | | |
| 3 | | | | |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



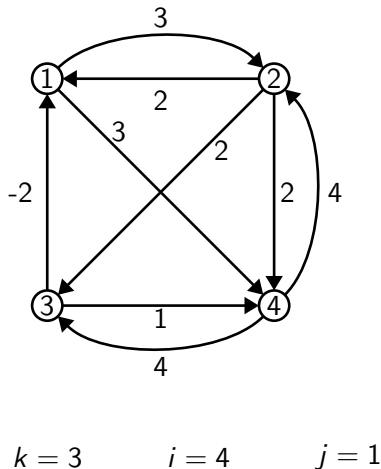
$L^2 =$

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

$L^3 =$

| | 1 | 2 | 3 | 4 |
|---|----|---|---|---|
| 1 | 0 | 3 | 5 | 3 |
| 2 | 0 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



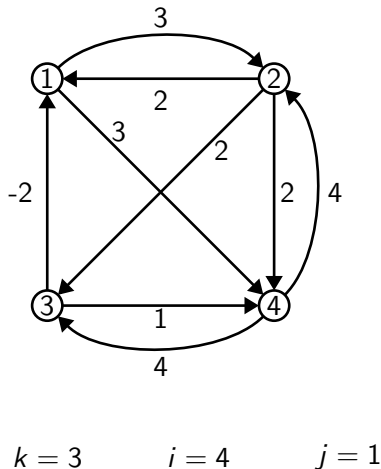
$L^2 =$

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

$L^3 =$

| | 1 | 2 | 3 | 4 |
|---|----|---|---|---|
| 1 | 0 | 3 | 5 | 3 |
| 2 | 0 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | | | | |

Algoritmo de Floyd (1962) - Ejemplo



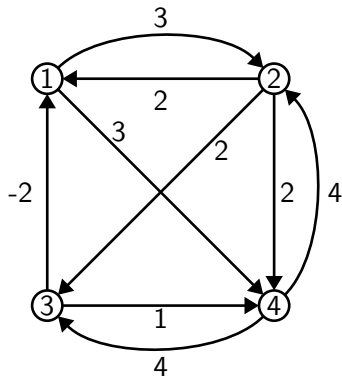
$L^2 =$

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

$L^3 =$

| | 1 | 2 | 3 | 4 |
|---|----|---|---|---|
| 1 | 0 | 3 | 5 | 3 |
| 2 | 0 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | 2 | | | |

Algoritmo de Floyd (1962) - Ejemplo



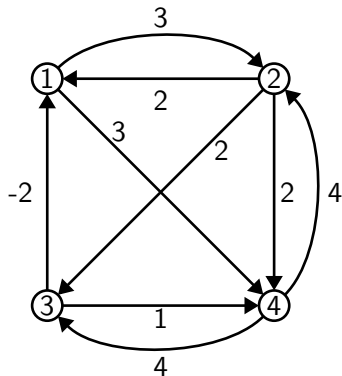
$L^2 =$

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

$L^3 =$

| | 1 | 2 | 3 | 4 |
|---|----|---|---|---|
| 1 | 0 | 3 | 5 | 3 |
| 2 | 0 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | 2 | 4 | 4 | 0 |

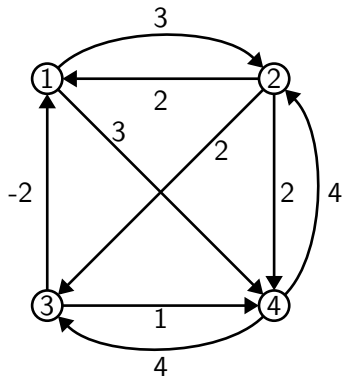
Algoritmo de Floyd (1962) - Ejemplo



$L^3 =$

| | 1 | 2 | 3 | 4 |
|---|----|---|---|---|
| 1 | 0 | 3 | 5 | 3 |
| 2 | 0 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | 2 | 4 | 4 | 0 |

Algoritmo de Floyd (1962) - Ejemplo



$$L^3 =$$

| | 1 | 2 | 3 | 4 |
|---|----|---|---|---|
| 1 | 0 | 3 | 5 | 3 |
| 2 | 0 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | 2 | 4 | 4 | 0 |

$$D = L^4 =$$

| | 1 | 2 | 3 | 4 |
|---|----|---|---|---|
| 1 | 0 | 3 | 5 | 3 |
| 2 | 0 | 0 | 2 | 2 |
| 3 | -2 | 1 | 0 | 1 |
| 4 | 2 | 4 | 4 | 0 |

Algoritmo de Floyd (1962)

Lema: Al finalizar la iteración k del algoritmo de Floyd, l_{ij} es la longitud de los caminos mínimos desde v_i a v_j cuyos nodos intermedios son elementos de $V_k = \{v_1, \dots, v_k\}$, si no existe ciclo de longitud negativa con todos sus vértices en V_k

Algoritmo de Floyd (1962)

Lema: Al finalizar la iteración k del algoritmo de Floyd, l_{ij} es la longitud de los caminos mínimos desde v_i a v_j cuyos nodos intermedios son elementos de $V_k = \{v_1, \dots, v_k\}$, si no existe ciclo de longitud negativa con todos sus vértices en V_k

Teorema: El algoritmo de Floyd determina los caminos mínimos entre todos los pares de nodos de un grafo orientado sin ciclos negativos.

Algoritmo de Floyd (1962)

Lema: Al finalizar la iteración k del algoritmo de Floyd, l_{ij} es la longitud de los caminos mínimos desde v_i a v_j cuyos nodos intermedios son elementos de $V_k = \{v_1, \dots, v_k\}$, si no existe ciclo de longitud negativa con todos sus vértices en V_k

Teorema: El algoritmo de Floyd determina los caminos mínimos entre todos los pares de nodos de un grafo orientado sin ciclos negativos.

- ¿Cuál es la complejidad de algoritmo de Floyd?

Algoritmo de Floyd (1962)

Lema: Al finalizar la iteración k del algoritmo de Floyd, l_{ij} es la longitud de los caminos mínimos desde v_i a v_j cuyos nodos intermedios son elementos de $V_k = \{v_1, \dots, v_k\}$, si no existe ciclo de longitud negativa con todos sus vértices en V_k

Teorema: El algoritmo de Floyd determina los caminos mínimos entre todos los pares de nodos de un grafo orientado sin ciclos negativos.

- ▶ ¿Cuál es la complejidad de algoritmo de Floyd?
- ▶ ¿Cuánta memoria requiere?

Algoritmo de Floyd (1962)

Lema: Al finalizar la iteración k del algoritmo de Floyd, l_{ij} es la longitud de los caminos mínimos desde v_i a v_j cuyos nodos intermedios son elementos de $V_k = \{v_1, \dots, v_k\}$, si no existe ciclo de longitud negativa con todos sus vértices en V_k

Teorema: El algoritmo de Floyd determina los caminos mínimos entre todos los pares de nodos de un grafo orientado sin ciclos negativos.

- ▶ ¿Cuál es la complejidad de algoritmo de Floyd?
- ▶ ¿Cuánta memoria requiere?
- ▶ ¿Cómo podemos hacer si además de las longitudes queremos determinar los caminos mínimos?

Algoritmo de Floyd (1962)

Lema: Al finalizar la iteración k del algoritmo de Floyd, l_{ij} es la longitud de los caminos mínimos desde v_i a v_j cuyos nodos intermedios son elementos de $V_k = \{v_1, \dots, v_k\}$, si no existe ciclo de longitud negativa con todos sus vértices en V_k

Teorema: El algoritmo de Floyd determina los caminos mínimos entre todos los pares de nodos de un grafo orientado sin ciclos negativos.

- ▶ ¿Cuál es la complejidad de algoritmo de Floyd?
- ▶ ¿Cuánta memoria requiere?
- ▶ ¿Cómo podemos hacer si además de las longitudes queremos determinar los caminos mínimos?
- ▶ ¿Cómo se puede adaptar para detectar si el grafo tiene ciclos de longitud negativa?

Algoritmo de Floyd (1962)

```
 $L^0 := L$   
para  $k$  desde 1 a  $n$  hacer  
  para  $i$  desde 1 a  $n$  hacer  
    si  $l_{ik}^{k-1} \neq \infty$  entonces  
      si  $l_{ik}^{k-1} + l_{ki}^{k-1} < 0$  entonces  
        retornar "Hay ciclos negativos."  
      fin si  
    para  $j$  desde 1 a  $n$  hacer  
       $l_{ij}^k := \min(l_{ij}^{k-1}, l_{ik}^{k-1} + l_{kj}^{k-1})$   
    fin para  
  fin si  
fin para  
retornar  $L$ 
```