

Algoritmos para determinar Caminos Mínimos en Grafos

Algoritmos y Estructuras de Datos III

Camino mínimo en grafos

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 camino C entre dos nodos v y u es la suma de las longitudes de las aristas del camino:

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

- ▶ Un **camino mínimo** C^0 entre u y v es un camino entre u y v tal que $l(C^0) = \min\{l(C) \mid C \text{ es un camino entre } u \text{ y } v\}$. Puede haber varios caminos mínimos.

Camino mínimo en grafos

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

- ▶ **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 camino de peso mínimo deja de estar bien definido.
- ▶ **Circuitos:** Un camino mínimo no puede contener circuitos.
- ▶ **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 - 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.

Algoritmo de Dijkstra (1959)

Asumimos que las longitudes de las aristas son positivas.

El grafo puede ser orientado o no orientado.

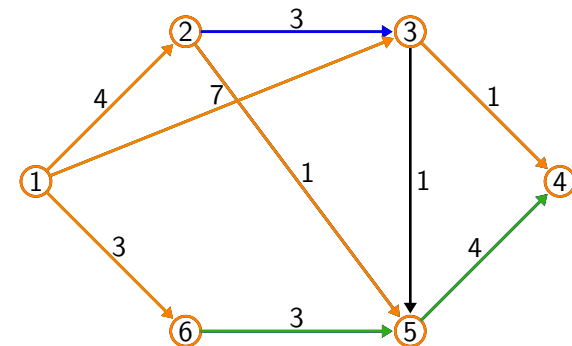
```
S := {v},  $\pi(v) := 0$ 
para todo  $u \in V$  hacer
  si  $(v, u) \in X$  entonces
     $\pi(u) := l(v, u)$ 
  si no
     $\pi(u) := \infty$ 
  fin si
fin para
mientras  $S \neq V$  hacer
  elegir  $w \in V \setminus S$  tal que  $\pi(w) = \min_{u \in V \setminus S} \pi(u)$ 
   $S := S \cup w$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

Algoritmo de Dijkstra (1959) - Determina camino mínimo

```
S := {v},  $\pi(v) := 0$ ,  $pred(v) := 0$ 
para todo  $u \in V$  hacer
  si  $(v, u) \in X$  entonces
     $\pi(u) := l(v, u)$ ,  $pred(u) := v$ 
  si no
     $\pi(u) := \infty$ ,  $pred(u) := \infty$ 
  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
       $\pi(u) := \pi(w) + l(w, u)$ 
       $pred(u) := w$ 
    fin si
  fin para
retornar  $\pi$ ,  $pred$ 
```

Algoritmo de Dijkstra - Ejemplo

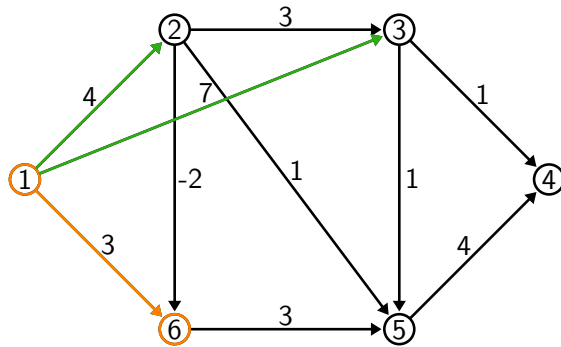
$S = \{1, 6, 2, 5, 3, 4\}$ $\pi = (0, 4, 7, 8, 5, 3)$



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 positivo 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 positivo en las aristas, el algoritmo de Dijkstra determina el camino mínimo entre el nodo v y el resto de los nodos de G .

Algoritmo de Bellman-Ford (1956)

Asumimos que el grafo es orientado y no tiene circuitos de longitud negativa.

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

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

$$\pi(u) := \infty$$

fin para

mientras hay cambios en π **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

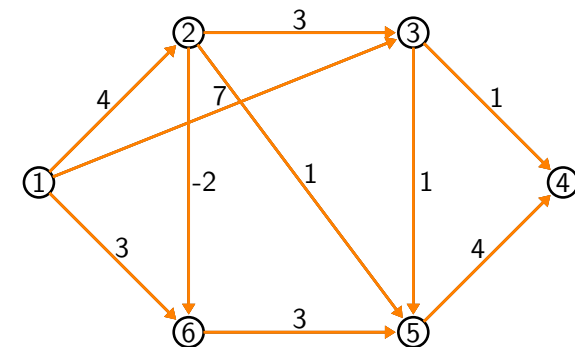
retornar π

Algoritmo de Bellman-Ford - Ejemplo

Iteración 2

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

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



Algoritmo de Bellman-Ford (1956)

Lema: Dado un grafo orientado G sin circuitos de longitud negativa, al finalizar la iteración k el algoritmo de Bellman-Ford determina el camino mínimo de a lo sumo k aristas entre el nodo v y los nodos de V .

Teorema: Dado un grafo orientado G sin circuitos de longitud negativa, el algoritmo de Bellman-Ford determina el camino mínimo entre el nodo v y el resto de los nodos de G .

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

Algoritmo de Bellman-Ford (1956)

Asumimos que el grafo es orientado. Detecta si hay circuitos de longitud negativa.

```

 $\pi(v) := 0, i := 0$ 
para todo  $u \in V \setminus \{v\}$  hacer
     $\pi(u) := \infty$ 
fin para
mientras hay cambios en  $\pi$  y  $i < n$  hacer
     $i := i + 1$ 
    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
si  $i = n$  entonces
    retornar "Hay circuitos de longitud negativa."
si no
    retornar  $\pi$ 
fin si
    
```

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:

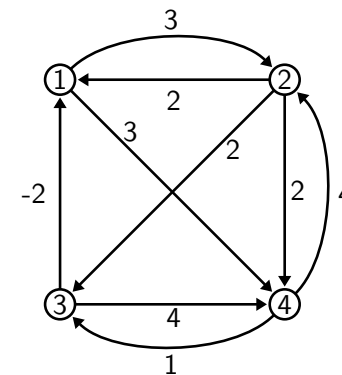
$$l_{ij} = \begin{cases} 0 & \text{si } i = j \\ l(i \rightarrow j) & \text{si } i \rightarrow j \in X \\ \infty & \text{si } i \rightarrow j \notin X \end{cases}$$

- $D \in R^{n \times n}$, 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



$$L = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 3 \\ 2 & 2 & 0 & 2 & 2 \\ 3 & -2 & \infty & 0 & 1 \\ 4 & \infty & 4 & 4 & 0 \end{array}$$

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 circuitos de longitud negativa.

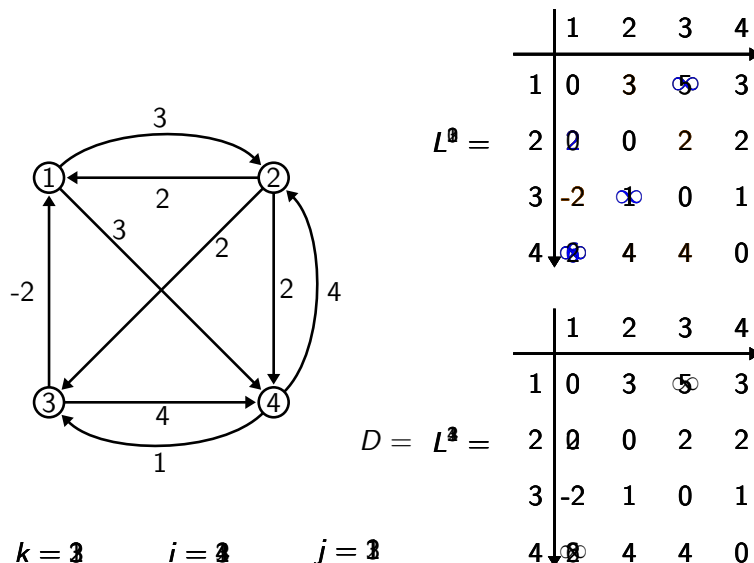
$$L^0 := L$$

```

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



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 $\{v_1, \dots, v_k\}$.

Teorema: El algoritmo de Floyd determina los caminos mínimos entre todos los pares de nodos de un grafo orientado sin circuitos 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 circuitos 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  $i \neq k$  y  $l_{ik}^{k-1} \neq \infty$  entonces
      si  $l_{ik}^{k-1} + l_{ki}^{k-1} < 0$  entonces
        retornar "Hay circuitos negativos."
      fin si
    para  $j$  desde 1 a  $n$  (si  $j \neq k$  y  $j \neq i$ ) hacer
       $l_{ij}^k := \min(l_{ij}^{k-1}, l_{ik}^{k-1} + l_{kj}^{k-1})$ 
    fin para
  fin si
fin para
retornar  $L$ 

```

Algoritmo de Dantzig (1966)

Al finalizar la iteración $k - 1$, el algoritmo de Dantzig genera una matriz de $k \times k$ de caminos mínimos en el subgrafo inducido por los vértices $\{v_1, \dots, v_k\}$.

Calcula la matriz L^{k+1} a partir de la matriz L^k para $1 \leq i, j \leq k$ como:

- ▶ $L_{i,k+1}^{k+1} = \min_{1 \leq j \leq k} (L_{ij}^k + L_{j,k+1}^k)$
- ▶ $L_{k+1,i}^{k+1} = \min_{1 \leq j \leq k} (L_{k+1,j}^k + L_{j,i}^k)$
- ▶ $L_{i,j}^{k+1} = \min(L_{i,j}^k, L_{i,k+1}^{k+1} + L_{k+1,j}^{k+1})$

Asumimos que el grafo es orientado. Detecta si hay circuitos de longitud negativa.

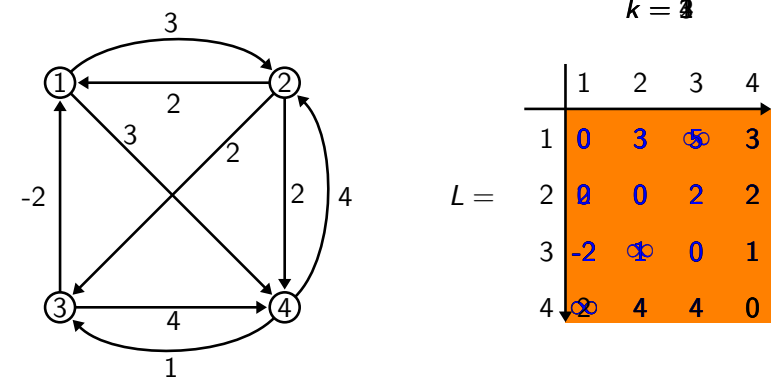
Algoritmo de Dantzig (1966)

```

para  $k$  desde 1 a  $n - 1$  hacer
  para  $i$  desde 1 a  $k$  hacer
     $L_{i,k+1} := \min_{1 \leq j \leq k} (L_{ij} + L_{j,k+1})$ 
     $L_{k+1,i} := \min_{1 \leq j \leq k} (L_{k+1,j} + L_{j,i})$ 
  fin para
   $t := \min_{1 \leq i \leq k} (L_{k+1,i} + L_{i,k+1})$ 
  si  $t < 0$  entonces
    retornar "Hay circuitos de longitud negativa"
  fin si
  para  $i$  desde 1 a  $k$  hacer
    para  $j$  desde 1 a  $k$  hacer
       $L_{i,j} := \min(L_{i,j}, L_{i,k+1} + L_{k+1,j})$ 
    fin para
  fin para
fin para
retornar  $L$ 

```

Algoritmo de Dantzig (1966) - Ejemplo



Algoritmo de Dantzig (1966)

Lema Al finalizar la iteración $k - 1$ del algoritmo de Dantzig, la matriz de $k \times k$ generada contiene la longitud de los caminos mínimos en el subgrafo inducido por los vértices $\{v_1, \dots, v_k\}$.

Teorema: El algoritmo de Dantzig determina los caminos mínimos entre todos los pares de nodos de un grafo orientado sin circuitos.

- ▶ ¿Cuál es la complejidad del algoritmo de Dantzig?
- ▶ ¿Qué diferencia tiene con el algoritmo de Floyd?
- ▶ ¿Qué ventajas o desventajas tiene?