**MA3705. Algoritmos Combinatoriales 2020.**
**Profesor:** José Soto
**Escriba(s):** José Delzo Escobar y Benjamín Tardy Donoso.
**Fecha:** 26 de octubre de 2020.

# Cátedra 12

## 1. Repaso

- **Bellman Ford (BF):** Permite trabajar en grafos con largos conservativos. Podemos tener arcos negativos, pero no ciclos negativos.

- **Dijkstra:** Solo para largos no negativos.

## 2. Dijkstra revisitado

### 2.1. Algoritmo de Dijkstra

En primer lugar recordemos el algoritmo de Dijkstra.

**Algorithm 1:** Dijkstra

**Entrada:** $G = (V, E)$ dirigido, $s \in V$ , $l : E \to \mathbb{R}^+$ [Introducimos el grafo y una función de costo asociada]
**for** $v \in V$ **do** $d(s, v) = +\infty$. $T(v) \leftarrow \infty$. Padre(v) $\leftarrow$ Null [Definimos la distancia al resto de vértices como $\infty$ y Padre(v) como vacío]
No-visitados $\leftarrow V$. [Al principio no hemos visitado ningún nodo]
$T(s) \leftarrow 0$. [La distancia al nodo en el cual estamos parados es nula]
**while** $T_{\text{mín}} \leftarrow \text{mín}_{v \in \textit{No-visitados}}\, T(v) < +\infty$ **do**
  Elegir $v \in$ No-visitados con $T(v) = T_{\text{mín}}$. [Elegimos el nodo que tenga menor T(v)]
  $d(s, v) = T(v)$ [Interpretamos T(v) como la distancia a dicho nodo]
  No-visitados $\leftarrow$ No-visitados-$v$ [Quitamos al nodo elegido de los no visitados, nos hemos "parado" en ese nodo].
  **for** $w \in N^+(v) \cap$ *No-visitados - v* **do**
    **if** $T(v) + l(vw) < T(w)$ **then**
      $T(w) \leftarrow T(v) + l(vw), Padre(w) \leftarrow v$ [Le damos la menor distancia posible a los nodos adyacentes].
    **fin**
  **fin**
**fin**
**Delvover:** d,Padre

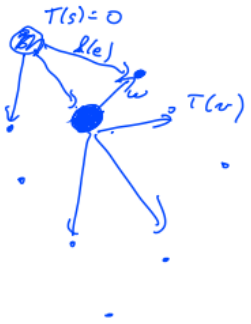



Figura 1: Algoritmo de Dijkstra

## 2.2. Correctitud de Dijkstra

Dividimos al proceso en 2 etapas:

1.Inicialización:

1 $(\forall v \in V)\ d(s,v) \leftarrow +\infty$, $T_1(v) \leftarrow +\infty$

2 NoVis $\leftarrow V$. $T_1(s) \leftarrow 0$.

En palabras, asignamos los valores $T(v)$ de cada nodo, agregamos todos los nodos a No Visitados y le damos al nodo inicial (s) el valor T(s)=0.

2.Iteración:
$T_{i+1} \equiv T_i$

1. $T_{\text{mín}} \leftarrow \text{mín}_{v \in \text{NoVis} T_i(v)}$ ( Elegimos menor nodo perteneciente a No Visitados).
2. **si** $T_{\text{mín}} = +\infty$ **entonces** terminar.
3. Elegir $v^* \in$ NoVis con $T_i(v^*) = T_{\text{mín}}$.
4. NoVis $\leftarrow$ NoVis - $v^*$ (Sacamos al nodo elegido de los no visitados).
5. $d(s,v^*) \leftarrow T_i(v^*)$ (El tiempo asignado a este nodo será la distancia entre este y el nodo inicial).
6. $(\forall w \in N^+(v^*) \cap$ NoVis) $T_{i+1}(w) \leftarrow \text{mín}(T_i(w), T_i(v^*) + l(vw))$. (Actualizamos el tiempo de los nodos adyacentes).

Ahora bien, ¿Cómo demostramos la correctitud? Para responder esta pregrunta, haremos uso del siguiente Teorema:

**Teorema 1.** *Si $H_i$ es el grafo de los arcos con colas visitadas al principio de la iteración i, entonces $\forall v$ visitado:*

$$d(s,v) = dist_G(s,v) = dist_{H_i}(s,v).$$

*$\forall v$ no visitado:*

$$T(v) = dist_{Hi}(s,v).$$

**Observación.** Antes de comenzar la demostración, podemos notar que en la iteración 1, tenemos $H_1 = (v, \emptyset)$ y $T(v) = 0$ si $v = s$ y $+\infty$ si $v \neq s$.

*Demostración.* La demostración se realizará a través de 4 casos.

1. Si $v$ visitado en $\leq i$, entonces:

$$d(s,v) = T_i(v) = dist_{H_i}(s,v) \geq dist_{H_{i+1}}(s,v) \geq dist_G(s,v) = dist_{Hi}(s,v).$$

2. Si $v = v^*$, entonces:

$$d(s,v^*) = T_i(v^*) = dist_{H_i}(s,v^*) = dist_{H_{i+1}}(s,v^*) \geq dist_G(s,v^*) = l(P),$$

donde $P$ es el camino mínimo de $s$ a $v^*$. De aquí se pueden observar dos subcasos.

1.1. Si $P$ **no** usa arcos fuera de $H_i$, entonces $l(P) = dist_{H_i}(s,v^*)$.

1.2. Si $P$ usa arcos fuera de $H_i$, entonces, sea $z$ un nodo no visitado dentro de $P$. Como $sPz \subseteq H_i$ ($sPz$ es el pedazo de $P$ que va desde $s$ a $z$), se tiene que

$$l(sPz) + l(zPv) \geq dist_{H_i}(s,z) = T_i(z) \geq T_i(v^*).$$

3. Si $v$ no visitado, $v \notin N^+(v^*)$, entonces:

$$T_{i+1}(v) = T_i(v) = dist_{H_i}(s, v) = dist_{H_{i+1}}(s, v).$$

4. Si $v$ no visitado, $v \in N^+(v^*)$ entonces:

$$T_{i+1}(v) = \text{mín}(T_i(v), T_i(v^*) + l(v^*v)) = \text{mín}(dist_{H_i}(s, v), d(s, v^*) + l(v*v)) \geq dist_{H_{i+1}}(s, v).$$

□

Explicaciones adicionales de los casos:

1. Si $v$ visitado en $\leq i$. Al haber más arcos, es posible que se agregue un camino menos largo entre un par de nodos. Un subgrafo tendrá un camino igual o más largo que un grafo que lo contenga. Siempre que tengamos una cadena de desigualdades que empieza y termina por el mismo término podemos decir que en realidad son todas igualdades
2. La primera igualdad se tiene por $*$, la segunda por ser no visitado, la tercera porque los caminos en $H_{i+1}$ ya estaban en $H_i$ (por la forma en que agregamos los arcos). Además, si $P$ usa arcos fuera de $i$, entonces $l(P) = T_i(v)$, esto porque el $T_i(v)$ fue elegido como el menor tiempo de todos.
3. $T_{i+1}(v) = T_i(v)$. Esto se debe a que el algoritmo solo modifica vértices en$H_i$ y sus vecinos]
4. $P$: camino mínimo en $H_{i+1}$ de $s$ a $v$. $T_i$ (v) tiene guardado mín$(dist_{H_i}(s, v)$. $T_i(v*) + l(v^*v)$ tiene guardado $d(s, v^*) + l(v, v^*)$.

### 2.3. Resumen de caminos mínimos desde un nodo $s$ en grafos dirigidos

| Función de largo | Algoritmo | Complejidad |
|---|---|---|
| Cualquiera en DAG | Orden topológico + prog. dinámica | $O(n+m)$ |
| Unitaria | BFS | $O(n+m)$ |
| Enteros entre 1 y M | BFS | $O(M(m+n))$ |
| No negativos | Dijkstra | $O(m + n \log n)$ |
| Conservativos | Bellman Ford | $O(n(n+m))$ |

- Tenemos un algoritmo muy rápido (Dijkstra) que se parece a Primm (va haciendo crecer un árbol de manera rápida).
- Calcularemos distancia en orden $(n+m)$ si es que no tenemos ciclos y los arcos son de largo 1.
- En caso de que los arcos tengan valores enteros "pequeñitos" [BFS O(M(m+n))]
- Dijikstra es para largos no negativos.
- Para largos conservativos (no hay ciclos negativos) tenemos a Bellman Ford con un orden de $O(n(n+m))$ (en grafos grandes tarda tiempo cuadrático).

**Observación** Para grafos no dirigidos solo agregamos 2 arcos de igual valor (uno de ida y otro de vuelta). Sin embargo, es posible que se creen ciclos negativos en algunos casos por lo que no podemos asegurar que Bellman Ford funcione correctamente.

## 3. Distancias en todos los pares

### 3.1. Como calcular las distancias $d(u, v)$ para todos los pares $u$, $v$

Una idea es correr distancia de fuente única desde cada nodo.

En particular para largos conservativos en grafos densos (es decir, $m = O(n^2)$), se tiene que la complejidad es $O(n^4)$, que es demasiado lento.

| Función de largo | Algoritmo | Complejidad |
|---|---|---|
| Cualquiera en DAG | Orden topológico + prog. dinámica | $O(n(n+m))$ |
| Unitaria | BFS | $O(n(n+m))$ |
| Enteros entre 1 y M | BFS | $O(nM(m+n))$ |
| No negativos | Dijkstra | $O(n(m+n\log n))$ |
| Conservativos | Bellman Ford | $O(n^2(n+m))$ |

## 3.2. Programación dinámica de Bellman-Ford

Una forma de mejorar Bellman Ford es usando: Programación Dinámica de Bellman Ford. Se permite usar Bellman Ford y luego verificare si es que existen ciclos negativos.

Bellman Ford usa (llamando $D^(k)(a,b) = d_{\leq k}(a,b)$):

$$D^{(i+1)}(s,t) = \text{mín}\left(D^{(i)}(s,t), \min_{u\in N^-(t)} D^{(i)}(s,u) + l(u,t)\right),$$

que es equivalente a:

$$\begin{aligned} D^{(i+1)}(s,t) &= \text{mín}\left(D^{(i)}(s,t), \min_{u\in N^-(t)} D^{(i)}(s,u) + D^{(1)}(u,t)\right) \\ &= \min_{u\in N^-(t)+t} D^{(i)}(s,u) + D^{(1)}(u,t) \\ &= \min_{u\in V} D^{(i)}(s,u) + D^{(1)}(u,t) \end{aligned}$$

## 3.3. Producto de matrices en semianillo ($\mathbb{R}_+^\infty$, mín, +)

Si $A$ y $B$ son matrices en $\mathbb{F}^{V\times V}$, con $\mathbb{F}$ un cuerpo cualquiera, entonces:

$$(A\cdot B)(s,t) = \sum_{u\in V} A(s,u)\cdot B(u,t).$$

Consideremos ahora matrices en $\mathbb{R}_+^{\infty\ V\times V}$. Entonces si definimos el nuevo producto, notado $\odot$, de tal forma que

$$(A\odot B)(s,t) = \min_{u\in V}(A(s,u) + B(u,t)),$$

se tiene que este producto es ascoativo, es decir:

$$(A\odot B)\odot C = A\odot(B\odot C).$$

Con lo anterior podemos notar que calcular distancias no es más que multplicar matrices. Para verificar esto veamos el siguiente algoritmo.

**Algorithm 2:** Calcular todas las distancias

**Entrada:** $G = (V, E)$ dirigido, $l : E \to \mathbb{R}$ conservativo

$$D^{(1)}(a,b) = \begin{cases} 0 & \text{si } a = b \\ l(ab) & \text{si } ab \in E \\ +\infty & \text{en otro caso} \end{cases}$$

**for** $k = 2, ..., n-1$ **do**

$D^{(k)} = D^{(k-1)} \odot D^{(1)}$

**fin**

**Devolver:** $D^{(n-1)}$.

Complejidad: Antes del for, debemos rellenar una matriz de nxn. Por tanto esta primera parte tiene orden de complejidad $n^2$. Luego debemos multiplicar 2 matrices (hayar el mínimo) lo que nos aumenta el orden a $O(n^3)$. Esto debe de hacerse un número n de veces, pues ya estamos dentro del for, por lo que el algoritmo termina siendo de orden $O(n^4)$.

Ahora bien, como

$$D^{(n-1)} = D^{(n)} = D^{(n+1)} = ...,$$

es decir las distancias no cambian después del $n-1$, ya que en largos conservativos con a lo más $k$ arcos se mantienen constantes después de $n-1$. Con esto basta calcular $D^{2^t}, t \in \mathbb{N}$. Así podemos presentar una mejora del algoritmo anterior.

**Algorithm 3:** Calcular todas las distancias

**Entrada:** $G = (V, E)$ dirigido, $l : E \to \mathbb{R}$ conservativo

Sea $M$ el menor entero tal que $2^M \geq n-1$.

$$D^{(1)}(a,b) = \begin{cases} 0 & \text{si } a = b \\ l(ab) & \text{si } ab \in E \\ +\infty & \text{en otro caso} \end{cases}$$

**for** $k = 1, ..., M$ **do**

$D^{(2^k)} = D^{(2^{k-1})} \odot D^{(2^{k-1})}$

**fin**

**Devolver:** $D^{(2^M)}$.

Lo que hicimos fue descomponer la matriz que queremos calcular utilizando descomposición binaria. Utilizamos potencias de 2 para disminuir el número de producto de matrices a realizar, de esta forma es posible reducir la complejidad al orden $O(n^3 * nlog(n))$

**Propuesto:** Modificar el algoritmo anterior para que guarde matrices de (argmins) $R^{(2^k)}_{k=1...M}$. Suponga acceso directo a todas las matrices calculadas y muestre como calcular un $s-t$ camino mínimo en tiempo $O(n)$.

## 3.4. Mejoras uando vértices internos

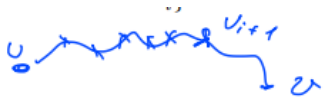
Roy (1959), Floyd(1952) y Warshall(1962) se dieron cuenta que la idea de iterativamente aumentar el conjunto de arcos usables es más eficiente que la multiplicación de matrices. Para ver esto consideremos un grafo $G = (V, E)$ dirigido con $l$ conservativo. Además enumeremos los vértices como $v_1, ..., v_n$.


$$D(u, v, i) = \text{mín}\{l(P) : P \text{ es } u - v \text{ camino con vértices internos en } V_i\}.$$

Con esto se tiene:

$$D(u, v, 1) = D^{(1)}(u, v)$$

$$D(u,v,i+1) = \text{mín}\left(D(u,v,i), D(u,v_{i+1},i) + D(v_{i+1},v,i)\right).$$




## 3.5. Algoritmo de (Roy)-Floyd-Wrshall

**Algorithm 4:** (Roy)-Floyd-Warshall

**Entrada:** $G = (V, E)$ dirigido, $l : E \to \mathbb{R}$ conservativo

$$D(a,b,1) = \begin{cases} 0 & \text{si } a = b \\ l(ab) & \text{si } ab \in E \\ +\infty & \text{en otro caso} \end{cases}$$

**for** $k = 2, ..., n,\ a, b \in V$ **do**
  $D(a,b,k) = \text{mín}(D(a,b,k-1), D(a,v_k,k-1) + D(v_k,b,k-1))$
  **fin**

**Devolver:** $D(a,b,n)$.

No se demostrará en este momento, pero el algoritmo anterior tiene orden $O(n^3)$

**Propuesto:** Suponga acceso directo al arreglo $D$ y muestre como calcular un $s-t$ camino mínimo en tiempo $O(n)$. Más aún, muestre como calcular un árbol de distancias desde $s$ en tiempo $O(n+m)$.

Una pregunta que nos podemos hacer es, ¿Cómo determinar si un largo es conservativo?

1. Bellman-Ford: Existe un ciclo negativo alcanzable desde $s$ si y solo si no hay desigualdad triangular en $d_{\leq n-1}$ . Es decir, existen $vw \in E$ tal que
$$d_{\leq n-1}(s,v) \geq d_{\leq n-1}(s,w) + l(w,v).$$
2. Floyd-Warshall: Existe ciclo negativo si y solo si $D(a,a,n) < 0$ para algún $a \in V$ .

   Propuestos:

1. Modificar BF para que devuelva el ciclo negativo encontrado.
2. Modificar FW para que devuelva el ciclo negativo encontrado.

   *Si el largo es conservativo $\Rightarrow$ paseo es lo mismo que camino*