DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Algoritmos y Estructuras de Datos III

Primer cuatrimestre 2021 *(dictado a distancia)*

Algoritmos para determinar Caminos Mínimos en Digrafos

Camino mínimo en digrafos

Queremos ir desde un punto a otro de una ciudad. Tenemos un mapa de las calles de la ciudad con las distancias entre cada par de intersecciones adyacentes.

- ¿Cómo determinamos el camino más corto entre esos dos puntos?
- Debemos considerar dos escenarios:
  - uno en el que todas las aristas o arcos tengan igual peso,
  - y otro donde no sucede ésto, es más, hasta podría haber aristas con peso negativo.
- Los grafos (pesados o no) son la estrutura natural para modelar redes en las cuales uno quiere ir de un punto a otro de la red atravesando una secuencia de enlaces.
- Sobre estas estructuras se han desarrollado algoritmos eficientes para resolver muchos de estos problemas.

Camino mínimo en digrafos

- Podemos modelar el mapa de la ciudad mediante un grafo:
  - los vértices representan las intersecciones de las calles
  - las aristas (o arcos si es orientado) los segmentos de calle entre dos intersecciones adyacentes
  - y la función de peso corresponde a la longitud de este segmento.
- Nuestro objetivo es encontrar un camino mínimo desde el vértice que respresenta la esquina de salida al vértice que representa la de llegada.
- El peso de cada arista puede ser interpretado como otras métricas diferentes a la distancia, como el tiempo que lleva recorrerlo, el costo que implica hacerlo o cualquier otra cantidad que se acumule linealmente a lo largo de un camino y que querramos minimizar.

Camino mínimo en digrafos

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

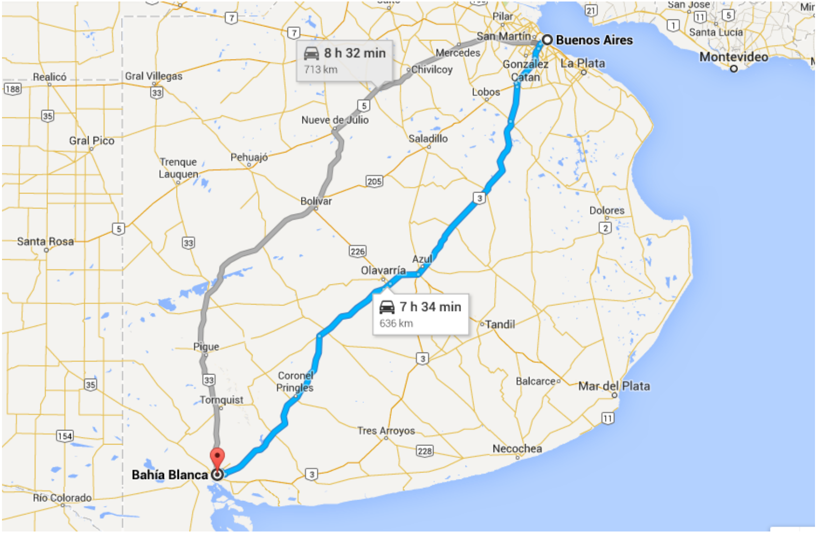
- La *longitud* de un camino C entre dos vértices v y u es la suma de las longitudes de las aristas del camino:

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

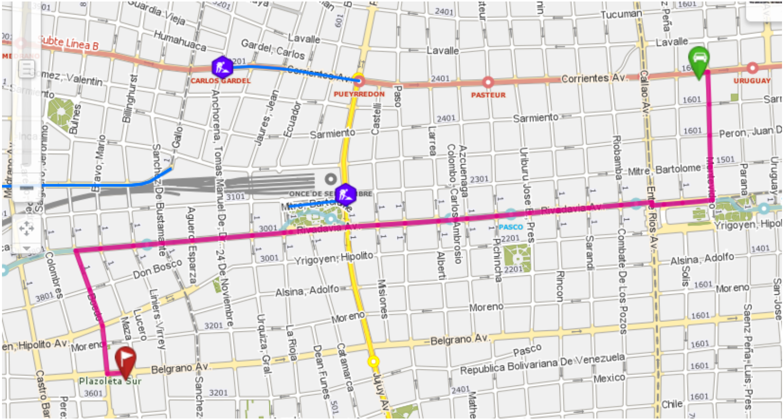
- Un *camino mínimo* C^0 entre v y u es un camino entre v y u tal que $l(C^0) = \min\{l(C) | C \text{ es un camino entre } v \text{ y } u\}$.

- Puede haber varios caminos mínimos.

Camino mínimo en digrafos



Camino mínimo en digrafos



Camino mínimo en digrafos

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 digrafos

- Todos estos conceptos se pueden adaptar cuando se estudian grafos orientados. En el resto de la clase, vamos a trabajar con este tipo de grafos.

- *Propiedad de subestructura óptima de un camino mínimo:*

  Dado un digrafo $G = (V, X)$ con una función de peso $l : X \to \mathbb{R}$, sea $P : v_1 \ldots v_k$ 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 digrafos

- *Aristas con peso negativo:*
  - Si el digrafo G no contiene ciclos de peso negativo (o contiene alguno pero no es alcanzable desde v), 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:* Siempre existe un camino mínimo que no contiene circuitos (si el problema está bien definido).

Camino mínimo - Único origen-múltiples destinos

Problema: Dados $G = (V, X)$ un digrafo, $l : X \to \mathbb{R}$ una función que asigna a cada arco una longitud y $v \in V$ un vértice del grafo. El objetivo es calcular los caminos mínimos desde v al resto de los vértices.

Distintas situaciones:

- El grafo puede ser orientado o no.
- Todos los arcos tienen igual longitud o no.
- Todos los arcos tienen longitud no negativa o no.
- Asumiremos que todo vértice es alcanzable desde v.

Camino mínimo - Único origen - Grafo no pesado

- En el caso de tener todas los arcos igual longitud, este problema se traduce en encontrar los caminos que definen las distancias (caminos con mínima cantidad de arcos).
- Para ésto podemos adaptar fácilmente el algoritmo BFS que vimos la clase pasada para calcular tanto la distancia como el camino mínimo desde v al resto de los vértices.

Camino mínimo - Único origen - Grafo no pesado

$BFS(G, v)$

entrada: $G = (V, X)$ de n vertices, un vertice v
salida: $pred[u]$ = antecesor de u en un camino minimo desde v
$dist[u]$ = distancia desde v a u

$pred[v] \leftarrow 0$, $dist[v] \leftarrow 0$, $COLA \leftarrow \{v\}$
para todo $u \in V \setminus \{v\}$ **hacer**
  $dist[u] \leftarrow \infty$
fin para
mientras $COLA \neq \emptyset$ **hacer**
  $w \leftarrow$ sacarPrimerElem($COLA$)
  para todo u tal que $(w \rightarrow u) \in X$ y $dist[u] = \infty$ **hacer**
    $pred[u] \leftarrow w$
    $dist[u] \leftarrow dist[w] + 1$
    insertarAlFinal($COLA, u$)
  para
fin mientras
retornar $pred$ y $dist$

Camino mínimo - Único origen - Grafo no pesado

Dado $G = (V, X)$ un digrafo y $v \in V$:

Lema: Sea $COLA = [v_1, \ldots, v_r]$. Se cumple que:

- $dist[v_1] + 1 \geq dist[v_r]$ y
- $dist[v_i] \leq dist[v_{i+1}]$, $\forall i = 1, \ldots r - 1$.

Corolario: Si el vértice u ingresa a $COLA$ antes que el vértice w, se cumple que $dist[u] \leq dist[w]$ en todo momento de la ejecución del algoritmo.

Lema: Se cumple que $dist[u] \geq d(v, u)$ para todo $u \in V$ en todo momento del algoritmo.

Camino mínimo - Único origen - Grafo no pesado

Teorema: Dado $G = (V, X)$ un digrafo y $v \in V$. El algoritmo BFS enunciado calcula $d(v, u)$ para todo $u \in V$.

Complejidad: La complejidad de este algoritmo es $\mathcal{O}(m)$, donde m es la cantidad de arcos del digrafo, ya que se examina cada arco exactamente una vez. Si es grafo no fuera dirigido, cada arista se examinaría dos veces, siendo también $\mathcal{O}(m)$.

Camino mínimo - Único origen - Grafo pesado
Algoritmo de Dijkstra (1959)

- Asume que las longitudes de los arcos son positivas. El grafo puede ser orientado o no orientado.
- Algoritmo goloso.
- Construye un árbol de caminos mínimos: un árbol enraizado en v conteniendo un camino mínimo desde v hacia todo otro vértice de V.
- Comienza con el vértice v y agrega un vértice a este árbol en cada iteración.
- En cada iteración agrega el vértice más cercano a v de entre todos los que todavía no fueron agregados al árbol.
- Es decir, en la iteración k agrega al árbol de caminos mínimos el k-ésimo vértice más cercano a v.

Camino mínimo - Único origen - Grafo pesado
Algoritmo de Dijkstra (1959)

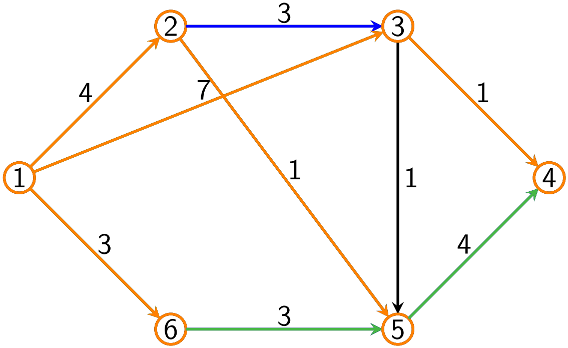
- Mantiene un conjunto S de vértices que ya han sido incorporados al árbol de caminos mínimos y cuya distancia está almacenada en el vector π.
- Inicialmente $S = \{v\}$ y $\pi[v] = 0$.
- Para cada $u \in V \setminus S$, determina el camino mínimo que puede ser construido siguiendo un camino desde v dentro de S hasta a algún $z \in S$ y luego el arco $(z \to u)$:

$$\forall u \in V \setminus S \text{ considera el valor } \pi'[u] = \min_{(z \to u) \in X, z \in S} \pi[z] + l((z \to u)).$$

- Elige el vértice w para el cual este valor es mínimo:
  - Agrega w a S
  - Fija $\pi[w] = \pi'[w]$
  - Actualiza π' para los adyacentes de w que no están en S.
- En la implementación no es necesario utilizar dos vectores distintos, π y π', ya que cada vértice tendrá sólo uno de estos valores activo.

Algoritmo de Dijkstra - Ejemplo

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



Algoritmo de Dijkstra (1959)

Longitudes de aristas no negativas, grafo orientado o no orientado.

$S \leftarrow \{v\}$, $\pi[v] \leftarrow 0$
para todo $u \in V$ **hacer**
    si $(v \rightarrow u) \in X$ **entonces**
        $\pi[u] \leftarrow l((v \rightarrow u))$
    si no
        $\pi[u] \leftarrow \infty$
    fin si
fin para
mientras $S \neq V$ **hacer**
    $w \leftarrow \arg \text{mín}\{\pi[u], u \in V \setminus S\}$
    $S \leftarrow S \cup \{w\}$
    para todo $u \in V \setminus S$ **y** $(w \rightarrow u) \in X$ **hacer**
        si $\pi[u] > \pi[w] + l((w \rightarrow u))$ **entonces**
            $\pi[u] \leftarrow \pi[w] + l((w \rightarrow u))$
        fin si
    fin para
fin mientras
retornar π

Algoritmo de Dijkstra (1959) - Determina camino mínimo

$S \leftarrow \{v\}\,,\;\; \pi[v] \leftarrow 0\,,\;\; pred[v] \leftarrow 0$

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

  si $(v \rightarrow u) \in X$ **entonces**

    $\pi[u] \leftarrow l((v \rightarrow u))\,,\;\; pred[u] \leftarrow v$

  si no

    $\pi[u] \leftarrow \infty\,,\;\; pred[u] \leftarrow \infty$

  fin si

fin para

mientras $S \neq V$ **hacer**

  $w \leftarrow \arg\min\{\pi[u], u \in V \setminus S\}$

  $S \leftarrow S \cup \{w\}$

  para todo $u \in V \setminus S$ **y** $(w \rightarrow u) \in X$ **hacer**

    si $\pi[u] > \pi[w] + l((w \rightarrow u))$ **entonces**

      $\pi[u] \leftarrow \pi[w] + l((w \rightarrow u))$

      $pred[u] \leftarrow w$

    fin si

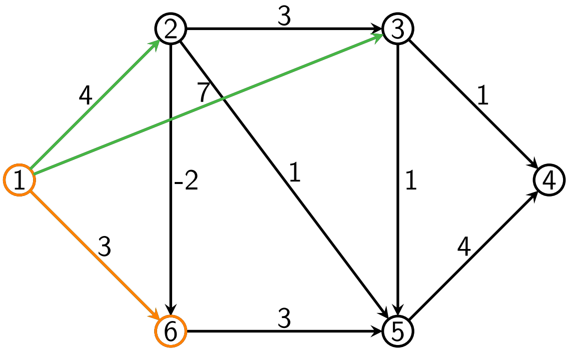
  fin para

fin mientras

retornar $\pi\,,\;\; pred$

Algoritmo de Dijkstra - Ejemplo (con peso negativo)

$S = \{1, 6\}$ $\qquad\qquad$ $\pi = (0, 4, 7, \infty, \infty, 3)$



¡Ya no actualizará $\pi(6)$!

Algoritmo de Dijkstra

Lema: Dado un grafo orientado G con pesos no negativos en las arcos, al finalizar la iteración k el algoritmo de Dijkstra determina el camino mínimo desde el vértice v a los vértices de S_k (siendo S_k el valor del 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 desde el vértice v al resto de los vértices de G.

Algoritmo de Dijkstra - Complejidad

Los pasos computacionales críticos de este algoritmo son:

1. encontrar el próximo vértice a agregar a S,
2. actualizar π.

Cada uno de estos pasos se realiza n veces.

- La forma más fácil de implementar (1) es buscar secuencialmente el vértice que minimiza, ésto se hace en $\mathcal{O}(n)$.

- En el paso (2), el vértice elegido tiene, a lo sumo n, adyacentes y para cada uno podemos actualizar π en $\mathcal{O}(1)$.

Considerando ésto, cada iteración es $\mathcal{O}(n)$, resultando $\mathcal{O}(n^2)$ el algoritmo completo.

Algoritmo de Dijkstra - Complejidad

- Podemos ser más cuidadosos en el cálculo de (2), teniendo en cuenta que en total (no por cada iteración) se realiza m veces.
- Con ésto el algoritmo completo es $\mathcal{O}(m + n^2)$ que sigue siendo $\mathcal{O}(n^2)$.
- Ésto sugiere que si queremos mejorar su complejidad, debemos revisar la implementación del paso (1).

Algoritmo de Dijkstra - Complejidad

- Para mejorar (1), se debe utilizar una estructura de datos que permita encontrar en forma más eficiente el mínimo.
- Por ejemplo, utilizando una cola de prioridades sobre heap con n elementos, crearla es $\mathcal{O}(n)$ y es posible borrar el elemento mínimo e insertar uno nuevo en $\mathcal{O}(\log_n)$.
- Si se mantiene un arreglo auxiliar apuntando a la posición actual de cada vértice en el heap, también es posible modificar el valor de π de un vértice en $\mathcal{O}(\log_n)$.

Algoritmo de Dijkstra - Complejidad

- Considerando todas las iteraciones, la operación (1) es $\mathcal{O}(nlog_n)$.
- Cada aplicación del paso (2) requiere, a lo sumo, $d(u)$ inserciones o modificaciones por cada vértice elegido u.
- Cada una de estas operaciones es $\mathcal{O}(log_n)$, dando en total $\mathcal{O}(d(u)log_n)$.
- Sumando sobre todas las iteraciones, la operación (2) es $\mathcal{O}(mlog_n)$.
- El algoritmo completo resulta $\mathcal{O}(n\log_n + m\log_n)$, que es $\mathcal{O}(m\log_n)$.
- Ésto es mejor que $\mathcal{O}(n^2)$ cuando m es $\mathcal{O}(n)$, pero peor si m es $\mathcal{O}(n^2)$.

Camino mínimo - Múltiples orígenes - múltiples destinos Algoritmos matriciales

Sea $G = (V, X)$ un digrafo de n vértices y $l : X \to \mathbb{R}$ una función de peso para las aristas de G. Definimos las siguientes matrices:

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

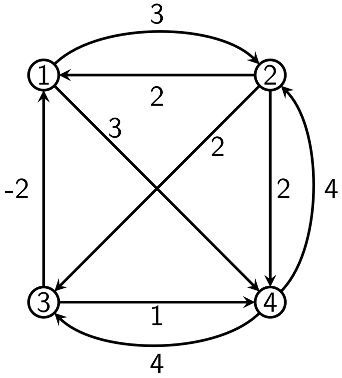
$$l_{ij} = \begin{cases} 0 & \text{si } i = j \\ l((v_i \to v_j)) & \text{si } (v_i \to v_j) \in X \\ \infty & \text{si } (v_i \to v_j) \notin X \end{cases}$$

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

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

D es llamada matriz de distancias de G.

Camino mínimo - Múltiples orígenes - múltiples destinos
Algoritmos matriciales



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

Camino mínimo - Múltiples orígenes - Alg. de Floyd (1962)

Calcula el camino mínimo entre todo par de vértices de un digrafo pesado.

Utiliza la técnica de programación dinámica y se basa en lo siguiente:

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

$$d_{ij}^1 = \text{mín}(d_{ij}^0, d_{i1}^0 + d_{1j}^0)$$

   d_{ij}^1 es la longitud de un camino mínimo de v_i a v_j con nodo intermedio v_1 o directo.

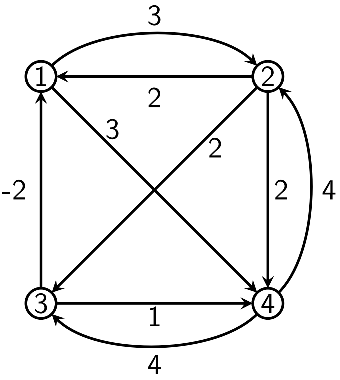
2. Si calculamos D^k a partir de D^{k-1} como

$$d_{ij}^k = \text{mín}(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$$

   d_{ij}^k es la longitud de un camino mínimo de v_i a v_j cuyos nodos intermedios están en $\{v_1, \ldots, v_k\}$.

3. $D = D^n$

Algoritmo de Floyd (1962) - Ejemplo



$D^{4} =$

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

$D = D^{4} =$

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

$k = 3$ $\quad i = 4$ $\quad j = 3$

Camino mínimo - Múltiples orígenes - Alg. de Floyd (1962)

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

$\mathtt{Floyd}(G)$

entrada: $G = (V, X)$ de n vertices

salida: D matriz de distancias de G

$D \leftarrow L$

para k **desde** 1 **a** n **hacer**

  para i **desde** 1 **a** n **hacer**

    para j **desde** 1 **a** n **hacer**

      $d[i][j] \leftarrow \min(d[i][j], d[i][k] + d[k][j])$

    fin para

  fin para

fin para

retornar D

Algoritmo de Floyd (1962)

Lema: Al finalizar la iteración k del algoritmo de Floyd, $d[i][j]$ es la longitud de los caminos mínimos desde v_i a v_j cuyos nodos intermedios son elementos de $V_k = \{v_1, \ldots, v_k\}$, si no existe circuito de peso negativo 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 circuitos negativos.

Algoritmo de Floyd (1962)

- ¿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)

```
Floyd(G)
    entrada: G = (V, X) de n vertices
    salida: D matriz de distancias de G
    D ← L
    para k desde 1 a n hacer
        para i desde 1 a n hacer
            si d[i][k] ≠ ∞ entonces
                si d[i][k] + d[k][i] < 0 entonces
                    retornar "Hay circuitos negativos"
                fin si
                para j desde 1 a n hacer
                    d[i][j] ← mín(d[i][j], d[i][k] + d[k][j])
                fin para
            fin si
        fin para
    fin para
    retornar D
```