MA3705. Algoritmos Combinatoriales. 2014.
Profesor: José Soto
Escriba(s): David Cares, Obed Ulloa e Ian Vidal.
Fecha: 5 de Septiembre 2014 .

Cátedra 11

1. Recuerdo Cátedra Anterior

De los métodos vistos en clases anteriores, tenemos para un digrafo $G = (V, E)$, con función de largo $\ell : E \to R$, los siguientes algoritmos:

- Paseo de número fijo de arcos (k arcos), que es de complejidad $O(k(n + m))$.
- Caminos/Paseos:
  - con ℓ constante mayor o igual a cero, cuya complejidad es $O(n + m)$.
  - con ℓ conservsativo, de complejidad $O(n(n+m))$ y aplicando Bellmand-Ford se lograba una complejidad $O(nm)$.
  - con ℓ programación dinámica sobre orden topológico y G acíclico, y en este caso la complejidad es $O(n + m)$ (visto en clase auxiliar).
- Para todo par de vertices y l conservativo, tenemos, con sus respectivas complejidades:
  - Bellman-Ford iterado, $O(n^2m)$.
  - Multiplicación de Matrices(álgebra tropical), $O(n^3log(n))$ (visto en clase auxiliar).
  - Floyd-Warshall, $O(n^3)$ (que veremos a continuación).

2. Método de Floyd-Warshall

En esta sección introduciremos un método para encontrar los caminos mínimos entre dos nodos. Este método se conoce como método de Floyd-Warshall. Dado un digrafo $G = (V, E)$ y $\ell : E \to R$ conservativo, queremos calcular la matriz $M(v, w) = d(v, w)$, matriz de las distancias entre dos nodos del digrafo.

Observación 1. Si conocemos M es fácil calcular los caminos.

Propuesto: Dado M, calcule un s-t-camino mínimo.

La idea tras el método de Floyd-Warshall es numerar los nodos $v_1, v_2, \ldots, v_n$ y definir:

$$\rho_i(u, v) = \text{mín}\{\ell(P) : P \text{ } u\text{-}v\text{-paseo} \mid \underbrace{V(P) \setminus \{u, v\}}_{\text{nodos internos}} \subseteq \{v_1, \ldots, v_n\}\}.^1$$

Si supiéramos $\rho_i \forall i$, entonces sólo basta considerar $d(u, v) = \rho_n(u, v)$.Esto se debe a que si ℓ es conservativo, los paseos mínimos y los caminos mínimos tienen igual largo,por esto, nos preguntamos cómo podemos calcular ρ_i en función de ρ_{i-1}. Pues bien, notemos que:

$$\rho_0(u, v) = \begin{cases} 0 & \text{si } u = v. \\ +\infty & \text{si } u \neq v. \end{cases}$$

$$\rho_i(u, v) = \begin{cases} \ell(P) : u\text{-}v\text{-paseo con nodos internos en } \{v_1, \ldots, v_i\} \text{ que tenga a } v_i \text{ como nodo interno.} \\ \ell(P) : u\text{-}v\text{-paseo con nodos internos en } \{v_1, \ldots, v_i\} \text{ que no tenga a } v_i \text{ como nodo interno.} \end{cases}$$

Luego,

$$\rho_i(u, v) = \text{mín}\{\rho_{i-1}(u, v_i) + \rho_{i-1}(v_i, v), \rho_{i-1}(u, v)\}.$$

[1] si bien es cierto en clase hablamos de caminos es más fácil argumentar en términos de paseos.

Consideremos un u-v-paseo óptimo P cuyos nodos internos están en $\{v_1, v_2, \ldots, v_i\}$ y que pase por v_i. Sean P_1 y P_2 subpaseos de P tal que P_1 es un u-v_i-paseo y P_2 un v_i-v-paseo. Luego es claro que $\rho_{i-1}(u, v_i) \leq \ell(P_1) \ \wedge \ \rho_{i-1}(v_i, v) \leq \ell(P_2)$ pues los ρ_{i-1} son óptimos. Podemos entonces enunciar y demostrar el siguiente lema.

Lema 1. $\ell(P) = \rho_{i-1}(u, v_i) + \rho_{i-1}(v_i, v)$

Dem.

- $\leq$: pues $\rho_{i-1}(u, v_i) + \rho_{i-1}(v_i, v)$ es el largo de un paseo de u a v que pasa por v_i cuyos nodos internos están en $\{v_1, \ldots, v_i\}$.
- $\geq$: pues $\ell(P) \geq \ell(P_1) + \ell(P_2) \geq \rho_{i-1}(u, v_i) + \rho_{i-1}(v_i, v)$. □

3. Algoritmo Floyd-Warshall

A partir de lo que definimos en la sección anterior, se presenta a continuación el algoritmo Floyd-Warshall. Luego mostraremos la correctitud y la complejidad del mismo.

Algoritmo 1: Algoritmo Floyd-Warshall

INPUT: $G = (V, E)$ digrafo, $\ell : E \to \mathbb{R}$ conservativo.
for $u \in V, v \in V$ **do**
  $\rho_0(u, v) = \begin{cases} 0 & \text{si } u = v. \\ +\infty & \text{si } u \neq v. \end{cases}$
end
for $i = 1, \ldots, n$ **do**
  for $u \in V, v \in V$ **do**
    $\rho_i(u, v) = \min\{\rho_{i-1}(u, v_i) + \rho_{i-1}(v_i, v), \rho_{i-1}(u, v)\}$.
  end
end
return $d = \rho_n$

Correctitud: Es fácil ver que el algoritmo funciona gracias a la demostración del lema 1. Es importante observar que funciona si ℓ es conservativo.

Complejidad: El segundo ciclo **for** del algoritmo realiza n iteraciones, mientras que el **for** anidado debe recorrer dos veces los nodos de V, con lo cual se tiene una complejidad de $O(n^3)$.

¿Qué sucede ahora si $\ell \geq 0$? La siguiente sección responde esta pregunta.

4. Algoritmo Dijkstra

Dijkstra se aprovecha de los largos mayores o iguales que cero para diseñar un algoritmo casi-glotón (no podemos hablar de un algoritmo glotón propiamente tal ya que Dijkstra involucra programación dinámica en cada paso). Dikjstra no funciona con largos negativos, como otros algoritmos de caminos/paseos de costo mínimo en digrafos, por las razones que ya se han mencionado en los algoritmos anteriores.

La idea de Dijkstra, es que el algoritmos comienza desde un nodo raíz s y va eligiendo (y agregando) los nodos que se encuentren a distancia minima de s en cada paso, manteniendo siempre una arborecencia de caminos mínimos desde la raíz s a los nodos ya agregados. A continuación se encuentra el pseudocódigo del algoritmo, el cual emplea de las funciones $D[v]$ y $\Pi[v]$ para vertices v, las cuales son como se detalla dentro del mismo algoritmo:

Algoritmo 2: Algoritmo Dijkstra

INPUT: $G = (V, E)$ digrafo, $\ell : E \to \mathbb{R}$ conservativo.
for $v \in V$ **do**
$$D[v] = \begin{cases} 0 & \text{si } v = s. \\ l(s,v) & \text{si } (s,v) \in E. \\ +\infty & \text{si no.} \end{cases}$$
$$\Pi[v] = \begin{cases} \perp & \text{si } v = s. \\ s & \text{si } (s,v) \in E. \\ \perp & \text{si no.} \end{cases}$$
end
$U = s$.
$F = \varnothing$.
while $U \neq V$ **do**
  if $D[w] = +\infty \forall w \in V \setminus U$ **then**
    $U \leftarrow V$.
  end
  Elegir $U^* \in U \setminus V$ con $D[u^*]$ mínimo.
  $U \leftarrow U + u^*$.
  $F \leftarrow F + (\Pi(u^*), u^*)$.
  for $w \in V \setminus U, (u^*, w) \in E$ **do**
    if $D[w] > D[u^*] + l(u^*, w)$ **then**
      $D[w] \leftarrow D[u^*] + l(u^*, w)$.
      $\Pi[w] \leftarrow U^*$.
    end
  end
end
return $H = (U, F)$

Idea: Dijkstra mantiene lo siguiente en cada iteración:

(1) $H = (U, F)$ es una arborecencia con raíz en s, tal que para todo nodo u en U, el único camino de s a u en H, denotado por (sHu) es de largo mínimo en G.

(2) $\forall u \in U, D[u] = l(sHu) = d(s, u)$.

(3) $\forall u \in U, D[u] = \text{mín}\{l(P) : P \text{ es } s - u \text{ dicamino en } G \text{ tal que } V(P) - u \subseteq U\}$
de hecho, si $w = \Pi[u] \in U$ (y $D[u] < \infty$), entonces: $D[u] = l(sHwu)$

Observación 2. Dijkstra es "glotón" pues agrega de todos los vértices de $V \setminus U$ a aquel que está mas cerca. Puesto que lo que trata de hacer Dijkstra es agregar un vértice fuera de los que ya tiene (en la arborescencia de caminos mínimos) tal que minimiza la distancia.

Demostración. Inducción en $i = |U|$

- $i = 1$: en este caso el algoritmo avanza sólo hasta el tercer paso. Luego se mantienen en cada paso:

  (1) Como sólo se ha llegado hasta el paso 3, es trivial que es arborecencia, puesto que solo se habrán agregado arcos que salen de s, por lo que o son nodos vecinos de s, por lo que el largo del camino entre estos nodos y s será el largo de la arista que los une a s, que será trivialmente mínimo; o es el mismo s, y en ese caso es cero; o si no es ninguno de los dos casos anteriores, entonces la distancia es infinita.

  (2) En U, sólo habrá nodos u vecinos a s, y por construcción $D[u] = l(sHu) = d(s, u)$.

  (3) Se tiene por (1) y (2).

- $i \geq 2$: sean D', Π' los vectores D y Π y sea H justo antes de que $|U| = i$:

(1) $H = H'$ agregando el nodo u^* y el arco $(\Pi(u^*), u^*)$(donde $\Pi(u^*) \in U$), luego H es arborecencia.

(2) Cuando $|\mathrm{U}| = i - 1$:
$D'[u] = l(sH'u) = d(s,u)\ \forall u \in U'$
$D'[u] = D[u]$ y $l(sH'u) = l(sHu) \Rightarrow D[u] = l(sH'u)$
Por lo que sólo debemos revisar u^*:
$D[u^*] = D'[u^*] = l(sH'wu)$, donde $w = \Pi(u)$ y la igualdad con $l(sH'wu)$ se tiene por inducción.
sabemos que:
$D[u^*] = l(sHu^*)$ $(sHu^* = sH'wu^*)$
Si $D[u^*] \neq dist(s,u^*)$: $\exists$ camino P de s a u^* tal que $l(P) < D[u^*]$
si este camino es tal que $V(P) \subseteq U \Rightarrow p(P) = D[u^*]$
luego debe existir $z \notin U$ en $V(P)$.

En la siguente imagen se puede ver un ejemplo de cómo es lo que se tiene en cada paso de Dijkstra:

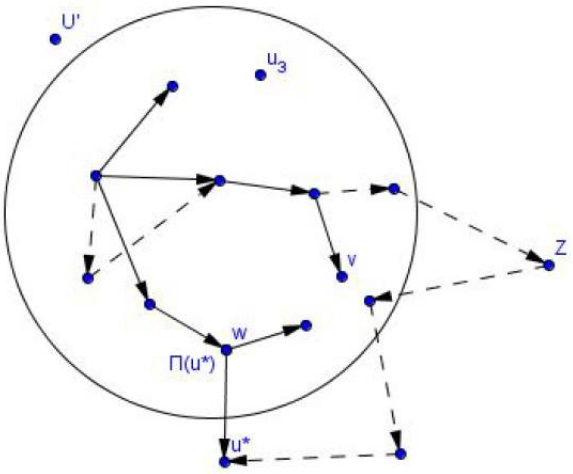

Figura 1.

Sea $z \in V \setminus U$ el primer nodo P en $V \setminus U$:
$D'[z]$ =largo del camino más corto de s a z que pasa solo por nodos de u.
$D'[z] \leq l(sPz) \leq l(P) < D[u^*] = D'[u^*]$.
Donde $l(sPz)$ es el largo del subcamino P entre s y z que es menor que $l(P)$ puesto que se consideran largos positivos, pero esto contradice la elección de u^*.
$\therefore D[u^*] = d(s,u)$

(3) Sea $u \in V \setminus U$,con $D[u]$ el largo mínimo de s a u que usa sólo nodos de U. Tenemos de esta froma dos casos:
- Si el camino mínimo que usa nodos de U no toca u^*, entonces se tiene que $D[u] = D'[u]$.
- si no: $D[u] = D[\Pi[u]] + l(\Pi[v], u)$.

Para visualizar cómo es U, U' y u^*, se puede ilustrar de la siguiente manera:

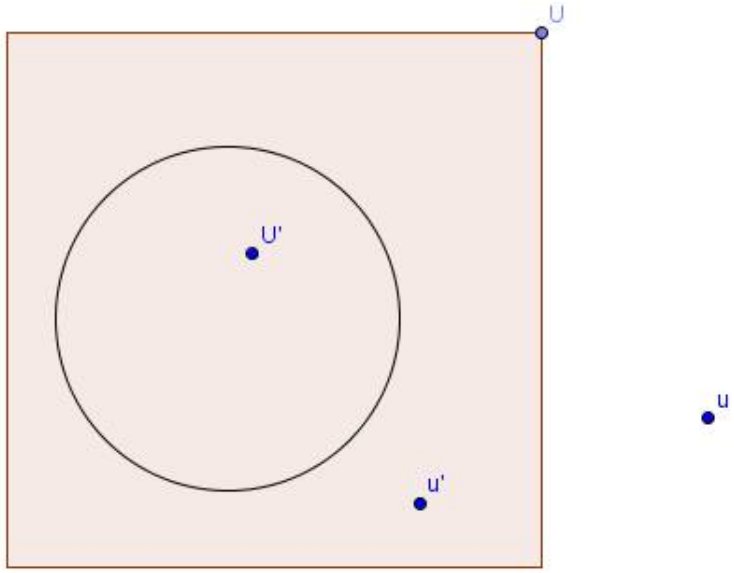

Figura 2.

□

La demostración de los hechos del punto 3 se probarán en la siguiente clase.