MA4701. Optimización Combinatorial. 2013.
Profesor: José Soto
Escriba(s): Gabriel Parra y Sélim Cornet.
Fecha: 27 de Septiembre de 2013.

Cátedra 11

1. Algoritmo de Dijkstra

En las clases anteriores, vimos un algoritmo (Bellman - Ford) que permite resolver el problema del camino de largo mínimo en un digrafo, en el caso general en que solo asumimos que la función de largo es conservativa. Además, vimos que en el caso particular en lo que el digrafo no tiene ciclos, existe un algoritmo más rápido que permite resolver este problema. Veamos ahora que cuando la función de largo es no-negativa, también existe un algoritmo mas eficiente: el algoritmo de Dijkstra (1959).

1.1. Descripción del algoritmo

Este algoritmo fue diseñado para encontrar los caminos más cortos desde un nodo s fijo a todos los otros nodos del digrafo. La idea tiene ambos del algoritmo glotón y de la programación dinámica: se trata de guardar, durante toda la ejecución del algoritmo, la informacion siguiente.

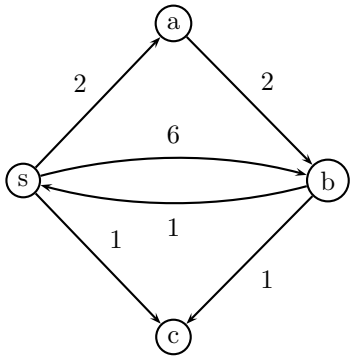
- Para cada $v \in V, D[v]$ tal que al final, $D[v]$ sea la distancia de s a v.
- Para cada $v \in V, \pi(v)$ tal que al final, $\pi(v)$ sea un predecesor de v en un (s,v)-camino de largo mínimo.
- $F \subseteq V$, un conjunto de nodos *fijos* tal que cuando $u \in F, D[u] = d(s,u)$.

El algoritmo es lo siguiente :

Algoritmo 1 Algoritmo de Dijkstra

Dado $G = (V, E)$ digrafo, $\ell : E \to \mathbb{R}_+$ función de largo.
Para cada $v \in V, (D[v], \pi[v]) = \begin{cases} (0, \perp) & \text{si } v = s. \\ (\ell(s,v), s) & \text{si } (s,v) \in E. \\ (+\infty, \perp) & \text{en los otros casos.} \end{cases}$
Sea $F = \{s\}$. {nodos fijos}
while $F \neq V$ **do**
  Sea $u^* \in V \backslash F$ que minimiza $D[u^*]$.
  if $D[u^*] = +\infty$ **then**
    $F \leftarrow V$ y terminar el algoritmo.
  end if
  for w tal que $(u^*, w) \in E$ **do**
    if $D[w] > D[u^*] + \ell(u^*, w)$ **then**
      $D[w] \leftarrow D[u^*] + \ell(u^*, w)$.
      $\pi(v) \leftarrow u^*$.
    end if
  end for
end while
Devolver (D, π).

Ejemplo. Vamos a ejecutar el algoritmo sobre el grafo siguiente.



	s	a	b	c	
$D_1[v]$	0	2	6	1	$F_1 = \{s\}$
$D_2[v]$	0	2	6	1	$F_2 = \{s, c\}$
$D_3[v]$	0	2	4	1	$F_3 = \{s, c, a\}$
$D_4[v]$	0	2	4	1	$F_3 = \{s, c, a, b\}$

1.2. Correctitud

Probemos ahora que el algoritmo de Dijkstra es correcto, es decir que devuelve las distancias mínimas y los caminos más cortos de s a todos los nodos.

Lema 1. *Al principio de cada iteración (linea 4), tenemos :*

(i) Para cada $v \in F$, si P es un (s, v)-camino que solo usa nodos en F, entonces $D[v] = d(s, v) = \ell(P)$ y $\pi(v)$ es el predecesor de v en P.

(ii) Para cada $v \in V \backslash F$, $D[v]$ es el largo de un camino P más corto de s a v que solo usa nodos internos en F, y $\pi(v)$ es el predecesor de v en P.

Demostración. Vamos a demostrar este lema por inducción sobre el número de iteraciones.
A la primera iteración, $F = \{s\}$ y el resultado es cierto.
Sean F conjunto de nodos fijos y D, π vectores al principio de una iteración, y F', D', π' sus valores al final de la misma iteración. Partamos probando (i).

Notamos que $F' = F + u^*$ y $(D', \pi')_{|F'} = (D, \pi)_{|F'}$ así que basta probar el resultado para $v = u^*$. Sea P un (s, u^*)-camino óptimo en G. Sea j el primer nodo de P en $V \backslash F$ y sean P_1 el subcamino de P de s a j, P_2 el subcamino en P de j a u^*. Tenemos $d(s, u^*) = \ell(P) = \ell(P_1) + \ell(P_2) \geq \ell(P_1)$ pero P_1 es subcamino de un camino óptimo, por lo que es óptimo y además solo usa nodos en F, por lo tanto $\ell(P_1) = D[j] \geq D[u^*] = D'[u^*]$. Concluimos que todo (s, u^*)-camino tiene un largo mayor o igual a $D[u^*]$.

Por hipótesis de inducción (ii), existe un (s, u^*)-camino Q con nodos internos en F tal que $D[u^*] = \ell(Q)$ y $\pi(u^*)$ es el penúltimo nodo de Q. Luego, $\ell(P) \leq \ell(Q) = D'[u^*] \leq \ell(P)$ ya que P es óptimo. Entonces $\ell(Q) = \ell(P)$ y Q es un (s, u^*)-camino que soló usa nodos en F : eso prueba (i).

Mostremos (ii), sea $v \in V \backslash F' = V \backslash F - u^*$. Por el algoritmo tenemos que $D'(v) \leq D(v)$ y elijamos de todos los (s, v)-caminos P que usan nodos internos en F' a aquel que usa u^* lo más tarde posible (o no lo usa).

Afirmación. *Si P usa a u^*, entonces u^* es el penúltimo nodo de P.*

Demostración. Si existe otro nodo después de u^* (lo denotaremos por j), sabemos que el subcamino (s-j) de P es óptimo por hipótesis de inducción, además $j \in F$ pues P solo usa nodos internos en F'. Por (i) existe Q un (s, j)-camino más corto que el subcamino $(s - j)$ de P y que no usa a u^*. Concatenando Q con el subcamino (j-v) de P, tenemos un camino más corto que P con nodos internos en F y que no usa a u^*, lo cual es una contradiccion por como elegimos P. □

Entonces si P usa a u^*, por la afirmacion tenemos que u^* es el último nodo, por lo tanto u^* es vecino de v, en este caso tenemos que $D'[v] = \min\{D[v], D[u^*] + \ell(u^*, v)\} \geq \ell(P)$, pero $D[u^*] + \ell(u^*, v) = \ell(P)$ con lo que se concluye la igualdad. Si P no usa a u^* el razonamiento es análogo y en ambos casos se prueba que $\ell(P) = D'[v]$. □

Corolario 1. *Dijkstra funciona y encuentra los caminos más cortos de s a todos los demás nodos.*

1.3. Complejidad

Analicemos la complejidad del algoritmo de Dijkstra.

- Inicializacioń : $O(n)$ pues se calcula el par $(D[v], \pi[v])$ para cada nodo.
- Desde linea 4 a linea 8: Calcular el ḿinimo toma $O(n)$.
- Desde linea 9 a linea 14: Actualizar los valores toma $O(|N^+(u^*)|)$.

Por lo tanto, en total la complejidad es $\sum_{v \in V} O(n + |N^+(u^*)|) = O(n^2 + m)$.

Es interesante notar que existen otras implementaciones de este mismo algoritmo que permiten mejorar la complejidad.

Definición 1. Llamaremos *cola de prioridad* a una estructura de datos en la que se pueden realizar las siguientes operaciones:

- Agregar un par (etiqueta, valor).
- Extraer mínimo.
- Cambiar el valor de alguna etiqueta.

Usando estas estructuras se puede reducir considerablemente la complejidad de este algoritmo, algunas implementaciones que se destacan son mediante *Heaps Binarios*, con la cual el algoritmo Dijkstra tiene complejidad $O(m \log n)$ y mediante *Heaps de Fibonacci* que logra una complejidad $O(m + n \log n)$.

Observación. En la practica usar *Heaps binarios* es mucho mas conveniente que usar *Heaps de Fibonacci*, debido a su fácil implementación y además garantiza $O(\log n)$ para todas las operaciones de cola de prioridad.

Observación. Usando colas de prioridad se tiene que:

$$\text{Dijkstra es } O(m + n \log n). \quad \text{Bellman-Fort es } O(mn).$$

Observación. Si quisiéramos calcular todas las distancias, se tiene que:

$$\text{Dijkstra iterado es } O(nm + n^2 \log n). \quad \text{Floyd-Warshall es } O(n^3).$$

Observación. Dijkstra funciona también en grafos no dirigidos con largos positivos. Para aplicar Dijkstra en estos grafos, basta reemplazar cada arista $e = \{u, v\} \in E$ por dos arcos antiparalelos (u, v) y (v, u) y asignarles $\ell(e)$ a ambos arcos. Para encontrar caminos mínimos en el grafo original, basta encontrar caminos dirigidos mínimos en el grafo dirigido auxiliar.

Finalmente, es bueno notar que si todos los arcos (o aristas) tienen el mismo largo, digamos igual à 1, entonces podemos encontrar un árbol de caminos de costo mínimo a partir de s calculando simplemente un árbol de búsqueda horizontal (BFS). Como consecuencia, calcular caminos mínimos con respecto al número de arcos (o aristas), se puede hacer en tiempo $O(n + m)$, lo cual es incluso más rápido que Dijkstra.