

MA37A Optimización

Caminos de peso mínimo

Profesor: Héctor Ramírez
Auxiliar: Oscar Peredo

23 de octubre de 2006

El problema del camino de peso mínimo recibe como entradas un grafo dirigido $G = (V, E)$ (con V el conjunto de vértices y $E \subseteq V \times V$ el conjunto de arcos), un nodo raíz $r \in V$ y una función de pesos $c : E \rightarrow \mathbb{R}^+ \cup \{0\}$.

El objetivo es encontrar, para cada $v \in V$, un camino dirigido desde r hasta v de peso mínimo.

Para ello ocuparemos el algoritmo de **Dijkstra**, que guarda un arreglo p con el nodo predecesor en el camino óptimo, es decir, $p[v] = u$ significa que el arco $(u, v) \in E$ pertenece al camino de peso mínimo. Además guarda el peso total de llegar desde la raíz a un nodo v utilizando el camino óptimo en un arreglo y .

Algorithm 1 Dijkstra($G = (V, E), c, r$)

```
1: ▷ INICIALIZACIÓN  
2: for  $v \in V$  do  
3:    $y[v] = \infty$   
4:    $p[v] = \text{NULO}$   
5: end for  
6:  $y[r] = 0$   
7: ▷ ITERACIÓN  
8:  $S \leftarrow \{r\}$   
9:  $Q \leftarrow V$   
10: while  $Q \neq \{r\}$  do  
11:   for  $v$  tal que  $(s, v) \in (S \times Q - \{r\}) \cap E$  do  
12:      $y[v] \leftarrow \min\{y[v], y[s] + c(s, v)\}$  ▷ Actualizar valores de  $y$  en los nodos de llegada partiendo de  $S$   
13:   end for  
14:   Encontrar  $u \in Q - \{r\}$  y  $s \in S$  tal que  $c(s, u)$  sea mínimo  
15:    $p[u] = s$   
16:    $S \leftarrow S \cup \{u\}$   
17:    $Q \leftarrow Q \setminus \{u\}$   
18: end while
```

Las estructuras para guardar la información, en este caso Q, S, p e y son arreglos. Este algoritmo sólo sirve en el caso $c(E) \subset [0, \infty)$. Para el caso $c(E) \subset (-\infty, \infty)$ se puede revisar el algoritmo de Bellman-Ford.

En internet existen diversas implementaciones del algoritmo, por ejemplo, la que se encuentra en

<http://renaud.waldura.com/doc/java/dijkstra/>

En esa implementación, se utilizan clases especiales para guardar los arreglos, esas clases son `HashMap` y `Set`. Para guardar los nodos, se utiliza la clase `City`. Para guardar una instancia del problema se utiliza la clase `DijkstraEngine`.

Esa implementación del algoritmo utiliza la siguiente función principal:

```

public void execute(City start, City destination)
{
    init(start);
    City u;
    while ((u = extractMin()) != null)
    {
        assert !isSettled(u);
        // destination reached, stop
        if (u == destination) break;
        markSettled(u);
        relaxNeighbors(u);
    }
}

```

Puede chequear diferentes soluciones contruyendo un driver `main.java`. Para el problema 1, el driver es de la forma:

```

class main{
    public static void main(String args[]){
        City a= new City('A');
        City b= new City('B');
        City c= new City('C');
        City d= new City('D');
        City e= new City('E');
        DenseRoutesMap rt = new DenseRoutesMap(5);
        rt.addDirectRoute(e,a,5);
        rt.addDirectRoute(e,b,10);
        rt.addDirectRoute(a,b,3);
        rt.addDirectRoute(b,a,2);
        rt.addDirectRoute(b,d,1);
        rt.addDirectRoute(a,c,2);
        rt.addDirectRoute(a,d,9);
        rt.addDirectRoute(c,d,6);
        rt.addDirectRoute(d,c,4);
        rt.addDirectRoute(c,e,7);
        DijkstraEngine de = new DijkstraEngine(rt);
        de.execute(e,null);
    }
}

```

La salida en pantalla es:

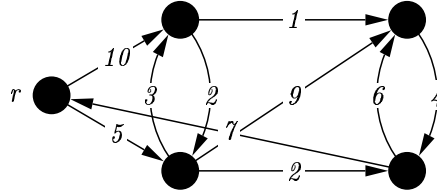
```

E->0
A->5
predecesor(A)=E
C->7
predecesor(C)=A
B->8
predecesor(B)=A
D->9
predecesor(D)=B
E->14
predecesor(E)=C

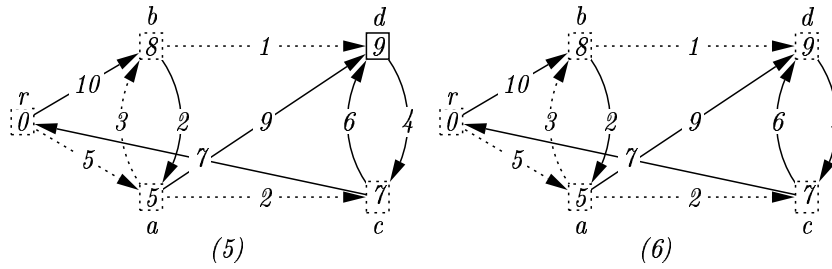
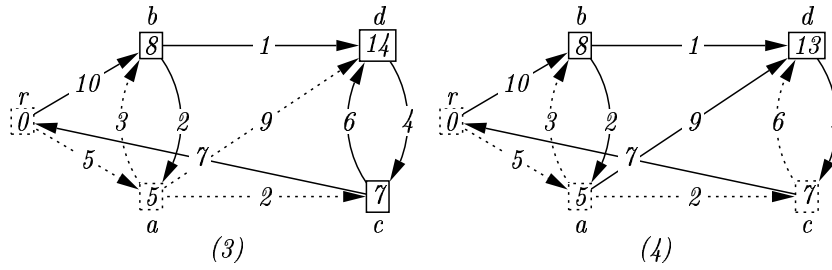
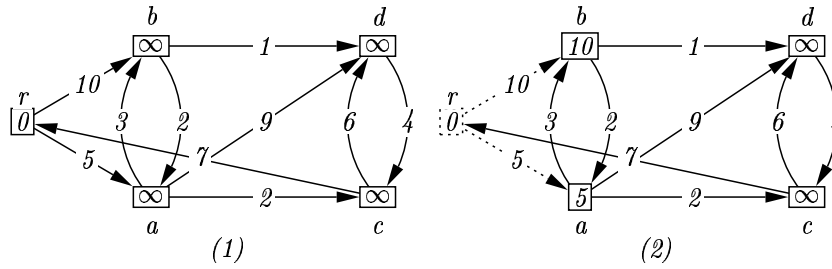
```

Lo cual coincide con la solución como lo veremos a continuación.

Problema 1. Encontrar los caminos de peso mínimo partiendo de la raíz en el siguiente grafo:



Solución 1. Dentro de los nodos se representa el valor de $y[v]$, que representa la suma de los costos del camino mínimo hasta el nodo v :



(1) Se inicializa el arreglo y con $y[r] = 0$ e $y[v] = \infty$ para $v \neq r$. Se setean los conjuntos $S = \{r\}$, $Q =$

$\{r, a, b, c, d\}$

(2) *Primera iteración.*

Se actualizan los valores de $y[a] = 5$ e $y[b] = 10$, pues son los nodos que se pueden alcanzar partiendo de algun nodo de $S = \{r\}$ (lineas 11,12).

Se escoge $u = a$ y $s = r$, pues $c(r, a) < c(r, b)$ (línea 14).

Se actualiza $p[a] = r$, $S \leftarrow \{r\} \cup \{a\}$ y $Q \leftarrow \{r, a, b, c, d\} \setminus \{a\}$ (lineas 15,16,17).

(3) *Segunda iteración.*

Se actualizan los valores de $y[b] = 8$, $y[c] = 7$ e $y[d] = 14$, pues son los nodos que se pueden alcanzar partiendo de algun nodo de $S = \{r, a\}$ (siempre se actualiza con el minimo valor posible, en este caso pudo haberse actualizado el nodo b como $y[b] = 10$, pero existe una actualización menor, que utiliza el arco (a, b) y resulta $y[b] = 5 + c(a, b) = 8$) (lineas 11,12).

Se escoge $u = c$ y $s = a$, pues $c(a, c) = \min\{3, 9, 2\} = 2$ (línea 14).

Se actualiza $p[c] = a$, $S \leftarrow \{r, a\} \cup \{c\}$ y $Q \leftarrow \{r, b, c, d\} \setminus \{c\}$ (lineas 15,16,17).

(4) *Tercera iteración.*

Se actualizan los valores de $y[b] = 8$ e $y[d] = 13$, pues son los nodos que se pueden alcanzar partiendo de algun nodo de $S = \{r, a, c\}$ (lineas 11,12).

Se escoge $u = b$ y $s = a$, pues $c(b, a) = \min\{3, 6\}$ (línea 14).

Se actualiza $p[b] = a$, $S \leftarrow \{r, a, c\} \cup \{b\}$ y $Q \leftarrow \{r, b, d\} \setminus \{b\}$ (lineas 15,16,17).

(5) *Cuarta iteración.*

Se actualizan el valor de $y[d] = 9$, pues es el único nodo que se puede alcanzar partiendo de algun nodo de $S = \{r, a, b, c\}$ (lineas 11,12).

Se escoge $u = d$ y $s = b$, pues $c(d, b) = \min\{1\}$ (línea 14).

Se actualiza $p[d] = b$, $S \leftarrow \{r, a, b, c\} \cup \{d\}$ y $Q \leftarrow \{r, d\} \setminus \{d\}$ (lineas 15,16,17).

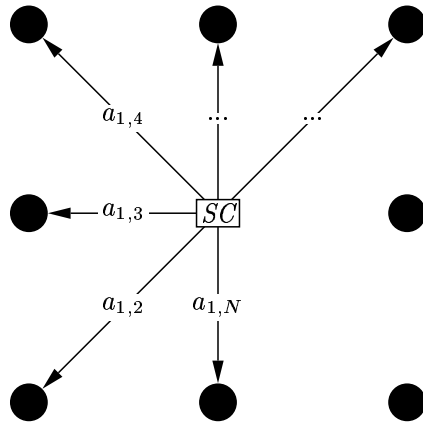
(6) Se sale de la iteración pues $Q = \{r\}$.

Problema 2. En una empresa se instala una red de N computadores, todos conectados entre si y además conectados a un supercomputador central (SC).

Para testear la velocidad de la comunicación en la red, los técnicos deciden probar la eficiencia del sistema enviando $N-1$ mensajes en forma paralela (el SC tiene N procesadores) desde el SC hacia los demás equipos. Desafortunadamente, el SC esta conectado solamente a uno de los equipos.

Describe de que manera se podría realizar el envío de mensajes de tal forma que se minimice el tiempo entre el envío por parte del SC y la recepción por parte de un equipo (considere que el tiempo de demora del mensaje es $a_{i,j}$ entre el equipo i y j . El SC es el equipo 1).

Solución 2. Si el SC estuviera conectado a todos los equipos, la situación seria de la forma:



y el tiempo de espera sería $\max_{i=2,\dots,n} \{a_{1,i}\}$, pues como los mensajes se envían en paralelo, la demora total será la demora máxima.

Ahora, como el SC está conectado a un solo equipo, supongamos al equipo 2, una opción es que SC envíe un mensaje a cada equipo a través del 2, lo cual demoraría $\sum_{i=2}^n a_{2,i}$, pues solo tiene un procesador y no puede realizar tareas en paralelo, solo de manera secuencial.

Sin embargo, si 2 le envía un mensaje a 3, 3 también podrá enviar el mensaje a otro equipo y sucesivamente los que reciban en mensaje. Para encontrar el camino que toma menos tiempo en enviar el mensaje, podemos utilizar Dijkstra, pues se forma un grafo dirigido, y nos interesa encontrar el camino de costo mínimo entre un equipo i y el equipo 2, que actúa como despachador.

Problema 3. Se desea contratar una empresa de camiones para transportar materiales desde un punto de una ciudad a otro. La ciudad se puede modelar como un grafo dirigido $G = (V, E)$, donde los nodos representan intersecciones de calles y los arcos las calles que las unen. La empresa está planeando las rutas desde un nodo r (origen) hasta un nodo t (destino).

- (a) Es bastante costoso cuando se producen demoras en las rutas de los camiones. Para solucionar esto, la empresa ha calculado la probabilidad $p(e) \in [0, 1]$ de que un calle $e \in E$ tenga demasiada congestión vehicular. Entregue un algoritmo que encuentre una ruta que tenga una mínima probabilidad de tener un camino congestionado (en un tiempo fijo). Puede asumir que todas las calles son independientes, es decir, la probabilidad de que n calles $(e_i)_{i=1}^n \subset E$ estén congestionadas se representa $p(e_1, \dots, e_n)$ y se cumple la igualdad $p(e_1, \dots, e_n) = \prod_{i=1}^n p(e_i)$. Lo mismo sucede con la probabilidad de que la calle no tenga congestión $q(e) = 1 - p(e)$.

Indicación: Adapte el problema para poder utilizar Dijkstra. Recuerde que Dijkstra minimiza la suma de los pesos (positivos) desde el nodo raíz hasta cualquier otro nodo. Si considera que los pesos del grafo G son las probabilidades $p(e)$, observe que estará minimizando $\sum_{i=1}^n p(e_i)$ y no $\prod_{i=1}^n p(e_i)$.

- (b) Suponga que cada calle $e \in E$ soporta un tonelaje máximo por vehículo $m(e) \in \mathbb{R}_+$. Encuentre el máximo tonelaje de un camión que puede transitar entre r y t .

Solución 3. (a) Como se quiere encontrar la mínima probabilidad de un camino que no tenga congestión, es mejor considerar la probabilidad del complemento, $q(e) = 1 - p(e)$, que indica la probabilidad de que el camino no tenga congestión.

Sea (e_1, \dots, e_n) un camino entre r y t , con $e_1 = r$ y $e_n = t$. La probabilidad de que este camino no tenga congestión vehicular es $q(e_1, \dots, e_n) = \prod_{i=1}^n q(e_i)$.

Para poder utilizar Dijkstra, los pesos a considerar en cada arco deben ser positivos y la solución que nos entrega Dijkstra son los caminos de peso mínimo entre la raíz r y cualquier otro nodo. Este peso mínimo se calcula como la suma de los pesos en cada arco desde la raíz hasta otro nodo.

Si aplicamos log a la pitatoria, se obtiene:

$$\log \prod_{i=1}^n q(e_i) = \sum_{i=1}^n \log(q(e_i))$$

Si la función de peso fuera $\log q(e_i)$ habría problemas cuando $q(e_i) = 0$, o equivalentemente $1 - p(e_i) = 0$, es decir, cuando la probabilidad de que haya congestión vehicular es 1 (hay congestión con toda seguridad). Una forma de evitar este problema es eliminar del grafo los arcos que satisfacen $p(e) = 1$ pues los camiones nunca pasarán por ahí.

Otro problema se presenta al ver que $p(e) \in [0, 1]$, luego $q(e) \in (0, 1]$ y por ende $\log q(e) \leq 0$, lo cual no nos sirve pues los pesos deben ser positivos. La solución a esto es simplemente considerar $-\log$.

Luego:

$$-\log \prod_{i=1}^n q(e_i) = \sum_{i=1}^n -\log(q(e_i))$$

donde $-\log(q(e_i))$ es el peso de la calle e_i .

Con esto tenemos un grafo dirigido G con una función de peso $w(e) = -\log(q(e))$ y un nodo raíz r . Aplicando Dijkstra se encuentra un camino de peso mínimo entre r y t que a su vez tiene la menor probabilidad de tener congestión vehicular.

(b) Si cada calle $e \in E$ soporta un peso máximo $m(e) \geq 0$ se puede utilizar Dijkstra cambiando solamente algunos pasos en el algoritmo de la siguiente forma (ver el algortimo en la clase auxiliar de Dijkstra):

- La línea 3 se reemplaza por: $y[v] = 0$.
- La línea 6 se reemplaza por: $y[r] = \infty$.
- La línea 12 se reemplaza por: $y[v] \leftarrow \max\{y[v], y[s] + c(s, v)\}$.
- La línea 14 se reemplaza por: Encontrar $u \in Q - \{r\}$ y $s \in S$ tal que $c(s, u)$ sea máximo.

Con esto, se obtienen los caminos de peso máximo desde la raíz hasta cualquier otro nodo, en particular t . Llamemos $C_{r,t}$ al camino entregado por Dijkstra modificado, entre r y t . El peso máximo que puede tener un camión para ir desde r hasta t , por el camino $C_{r,t}$, será el máximo de los $m(e)$ con $e \in C_{r,t}$.