Caminos de costo mínimo. DAG y Dijkstra

Colectivo Estructuras de Datos y Algoritmos

Octubre 2020

1. Resuelva el problema de los caminos de costo mínimo partiendo de un solo origen en un grafo dirigido y acíclico, utilizando la técnica de **Programación Dinámica en DAG** vista en la Clase Práctica 5.

Respuesta

Resolvamos este problema definiendo una función que calcule la longitud del camino de costo mínimo desde un vértice origen s hacia un vértice v cualquiera de G:

$$F(v) = \begin{cases} 0 & \text{si } v = s \\ +\infty & \text{si } indegree(v) = 0 \\ \text{mín}_{u_i \mid \langle u_i, v\rangle \in E} \left(F(u_i) + w(\langle u_i, v\rangle)\right) & \text{en otro caso} \end{cases}$$

Noten que la función descrita computa correctamente la longitud del camino de costo mínimo desde s hacia un vértice v, puesto que dicho camino tiene que pasar por uno de los vértices que tienen a v en su lista de adyacencia. Y, de acuerdo al lema 24.1 visto en conferencia: cualquier subcamino de un camino de costo mínimo, tiene que ser un camino de costo mínimo entre los vértices que une.

2. Implemente un algoritmo que dado un grafo dirigido y ponderado $G = \langle V, E\rangle$ con función de costo no negativa, un vértice $s \in V$ y la salida del algoritmo de Dijkstra (array d) aplicado sobre el grafo G a partir del vértice origen s; determine en $O(|V| + |E|)$ un camino de costo mínimo que comience en s y termine en un vértice $a \in V$ dado.

Respuesta

Dado un vértices s, la salida d del algoritmo Dijkstra solo contiene, para cada vértice $v \in V$ el valor de la longitud del camino de costo mínimo de s a v (asumiendo que dicho camino exista). Lo que es importante notar en este ejercicio, es que si existe el arco $\langle u, v\rangle$ en E, entonces existe un camino de costo mínimo $C = [s, ..., u, v]$ (con $u = s$ potencialmente), si y solo si $d[u]+\omega(u, v) = d[v]$. Demostremos esto

($\Rightarrow$)

Sea $C = [s, ..., u, v]$ un camino de costo mínimo cualquiera de s a v, tal que su penúltimo vértice es u. Sea C' el subcamino de C que va de s a u. Entonces, por el lema 24.1 visto en Conferencia, se cumple que C' es un camino de costo mínimo de s a u. Entonces se cumple que:

$$\delta(s, v) = \delta(s, u) + \omega(u, v)$$

Pero, debido a la correctitud del algoritmo de Dijkstra, se cumple que $d[u] = \delta(s, u)$ y que $d[v] = \delta(s, v)$. Sustituyendo nos queda que:

$$d[v] = d[u] + \omega(u, v)$$

■

($\Leftarrow$)

Si se cumple que $d[u] + \omega(u, v) = d[v]$, entonces existe un camino que pasa por u que tiene costo $d[v]$, que además sabemos, por la correctitud de Dijkstra que $d[v] = \delta(s, v)$. ■

Basado en esta propiedad, el siguiente pseudocódigo nos muestra una forma de construir un camino de costo mínimo de s a v.

```
1: d ← Dijkstra(s)
2: C ← Stack()
3: C.push(v)
4: G^T ← transpuesto de G
5: current ← v
6: while current ≠ s do
7:    for u ∈ G^T[current] do
8:       if d[u] + ω(u, current) = d[current] then
9:          C.push(u)
10:         current ← u
11:      end if
12:   end for
13: end while
14: C.push(s)
15: return C
```

En cuanto a la complejidad temporal, se asume que el *array* d es dado, la línea 4 se ejecuta en $O(|V|+|E|)$, mientras que los ciclos anidados en conjunto solo visitan cada vértice de un camino de costo mínimo de s a v, que son menos de $|V|$, porque los caminos de costo mínimo son simples. Además, visitan todas los arcos incidentes en dichos vértices a lo sumo. En total la complejidad de los ciclos anidados es $O(|V| + |E|)$. Luego, por la regla de la suma, la complejidad temporal del algoritmo es $O(|V| + |E|)$.
Queda indicado diseñar un algoritmo (con la misma complejidad temporal) que en vez de devolver un camino, devuelva una estructura que permita e $O(|C|)$ imprimir cualquier camino C desde s hacia cualquier vértice u.

3. Sea $G = \langle V, E\rangle$ un grafo dirigido y ponderado con función de costo $\omega : E \to \mathbb{R}$ tal que $\omega(e) >= 1$ para todo $e \in E$. Se define el costo de un camino $p = [v_0, ..., v_k]$ de G como $\omega(v_0, v_1) * \omega(v_1, v_2) * ... * \omega(v_{k-1}, v_k)$. Diseñe un algoritmo que permita determinar el camino de costo mínimo en G de acuerdo a la función de costo ω. La complejidad temporal de su algoritmo debe ser $O(|E| \log |V|)$.

Respuesta

La primera idea que se les debe ocurrir en este ejercicio es tratar de modificar el algoritmo de *Dijkstra* utilizando el producto en lugar de la suma para hacer *RELAX* a cada arco. Concretamente el método *RELAX* del arco $\langle u, v\rangle$ sería:

```
1: if d[u] * ω(u, v) ≤ d[v] then
2:    d[v] = d[u] * ω(u, v)
3: end if
```

Esta solución es correcta. Sin embargo, para demostrar su correctitud, como se está modificando el algoritmo, habría que seguir todos los pasos realizados en Conferencia para demostrar la correctitud del algoritmo de *Dijkstra*.
Por lo tanto, vamos a tratar de encontrar una solución equivalente, que sí utilice directamente el algoritmo de *Dijkstra* tal y como fue visto en Conferencia. Y la pregunta que tenemos que hacernos es: ¿Cómo pasamos de minimizar el producto, a minimizar la suma? Recordemos la propiedad de los logaritmos que dice que para un conjunto de n números reales no negativos $a_1, a_2, ..., a_n$ se cumple que:

$$\log(a_1) + \log(a_2) + ... + \log(a_n) = \log(a1 * a2 * ... * a_n)$$

Además, el logaritmo es una función creciente en todo su dominio, por lo que si a y b son reales no negativos entonces:

$$a \leq b \Leftrightarrow \log(a) \leq \log(b)$$

Luego, minimizar el producto de una secuencia de reales no negativos equivale a minimizar la suma de sus logaritmos. Por tanto, si definimos la función de costo ω' como:

$$\omega'(u, v) = \log(\omega(u, v))$$

el problema se reduce al problema de caminos de costo mínimo con función de costo ω'. Noten que como $\omega(u, v) \geq 1$ para todo u, v; entonces $\omega'(u, v) \geq 0$. Por tanto, se puede resolver el problema utilizando el algoritmo de *Dijkstra*.

4. Sea $G = \langle V, E \rangle$ un grafo dirigido y ponderado con función de costo no negativa ω, y un par de vértices $s, t \in V$. Diseñe un algoritmo que permita determinar el conjunto de todos los vértices de G que pertenecen a algún camino de costo mínimo de s a t. La complejidad temporal de su algoritmo debe ser $O(|E| \log |V|)$.

Respuesta

Siguiendo el mismo razonamiento del ejercicio 2, llegamos a una propiedad un poco más fuerte, pero igualmente sencilla de demostrar. Esto es: un vértice u pertenece a algún camino de costo mínimo entre s y t si y solo si $\delta(s, u) + \delta(u, t) = \delta(s, t)$. Demostremos esto.

($\Rightarrow$)

Esta parte sale directamente del lema 24.1 visto en Conferencia. Si u pertenece a un camino de costo mínimo entre s y t, entonces cualquier subcamino tiene que ser también un camino de costo mínimo entre los vértices que une. Particularmente el subcamino que va de s a u y el que va de u a t, lo son. Luego $\delta(s, u) + \delta(u, t) = \delta(s, t)$ ■

($\Leftarrow$)

En este sentido es trivial. Si $\delta(s, u) + \delta(u, t) = \delta(s, t)$ entonces el camino de s a u con costo $\delta(s, u)$ concatenado con el camino de u a t con costo $\delta(u, t)$ es un camino de costo mínimo de s a t que pasa por u. ■

Entonces, para poder determinar la respuesta del problema basta con conocer, para todo vértice u, los valores de $\delta(s, u)$ y $\delta(u, t)$. Pero noten que el valor de $\delta(u, t)$ en G es igual al valor de $\delta(t, u)$ en G^T, ya que todo camino que existe en G de x a y existe en G^T de y a x para todo par de vértices x, y.

Basados en estas ideas, podemos correr el algoritmo de *Dijkstra* sobre G y G^T, a partir de s y t respectivamente, para saber, para todo vértice u los valores $\delta(s, u)$ y $\delta(u, t)$. Luego para cada vértice u comprobamos la condición necesaria y suficiente para que pertenezca a un camino de costo mínimo de s a t. El siguiente pseudocódigo ilustra esto.

```
1: d1 ← Dijkstra(G, s)
2: d2 ← Dijkstra(G^T, t)
3: answer ← lista vacía
4: for u ∈ V(G) do
5:     if d1[u] + d2[u] = d1[t] then
6:         answer.append(u)
7:     end if
8: end for
9: return answer
```

Obtener el grafo transpuesto de G es $O(|V| + |E|)$. Aplicar el algoritmo de *Dijkstra* dos veces es $O(|E| \log |V|)$. El ciclo de la línea 4 es $O(|V|)$. Por la regla de la suma, la complejidad temporal del algoritmo es $O(|E| \log |V|)$.

5. Sea $G = \langle V, E \rangle$ un grafo dirigido y ponderado con función de costo no negativa ω, y un par de vértices $s, t \in V$. Se dice que un par ordenado de vértices $\langle u, v \rangle$ es X-*cool*, si al añadir el arco $\langle u, v \rangle$ a G con $\omega(u, v) = X$ se reduce el costo del camino de costo mínimo de s a t. Diseñe un algoritmo que dados G, s, t y X; permita determinar en $O(|V|^2 + |E| \log |V|)$ el conjunto de pares X-*cool* en G.

Respuesta

Siguiendo la idea del ejercicio anterior, notemos que un arco $\langle u, v \rangle$ con costo X pertenece a algún camino de costo mínimo entre s y t cuando $\delta(s, u) + X + \delta(v, t) = \delta(s, t)$. Luego, saber si al añadir el arco $\langle u, v \rangle$ el camino de costo mínimo de s a t mejora, basta comprobar si el valor $\delta(s, u) + X + \delta(v, t)$ es menor que el costo del camino de costo mínimo antes de poner dicho arco.

El siguiente pseudocódigo ilustra la idea anterior.

```
1: d1 ← Dijkstra(G, s)
2: d2 ← Dijkstra(G^T, t)
3: answer ← lista vacía
4: for u, v ∈ E(G) do
5:    if d1[u] + d2[v] + X < d1[t] then
6:       answer.append(u, v)
7:    end if
8: end for
9: return answer
```

La demostración de la correctitud y el análisis de la complejidad temporal son similares al ejercicio anterior.

6. Sea $G = \langle V, E \rangle$ un grafo dirigido y ponderado con función de costo no negativa ω, un par de vértices $s, t \in V$ y dos enteros $1 \leq k \leq |V|$ y $p \geq 0$. Diseñe un algoritmo que permita determinar el costo del camino de costo mínimo de s a t que se puede lograr poniéndole costo p a lo sumo a k arcos de G. La complejidad temporal de su algoritmo debe ser $O(k|E| \log |V|)$.

Respuesta

Para llegar a una solución vamos a tratar de pensar en resolver un problema más pequeño, a ver si podemos generalizar dicha idea a la solución del problema dado. ¿Cómo pudiéramos resolver el problema si $k = 1$? Bueno, en este caso, se convierte exactamente en el problema anterior. Pero dicha solución no es extensible eficientemente a un valor de k más grande, porque implicaría probar con todas las posibles combinaciones de k arcos.

¿Qué tal si en lugar de tratar de sustituir explícitamente el costo de ciertos arcos, simulamos, con arcos ficticios estos nuevos arcos con los costos cambiados? Pero, ¿cómo hacer eso sin remplazar los arcos originales? Podemos tener un multigrafo G^m, en el que por cada arco $\langle u, v \rangle$ con costo c en el grafo G, van a existir, además del arco $\langle u, v \rangle$ con costo c, otro arcos entre esos mismos vértices pero con costo p. Esta idea tiene un problema, y es que no estamos controlando que un camino de costo mínimo de s a t tenga exactamente un arco de los "nuevos"que se crearon con costo p. Para ello tenemos que garantizar que una vez un camino transita por uno de dichos arcos, no lo vuelva a hacer.

Para lograr esto, en lugar de tener un multigrafo, podemos tener dos grafos, G_0 y G_1, idénticos al grafo G. Y ubicar adicionalmente, por cada arco $\langle u, v \rangle$ en G, un arco que vaya desde el vértice u en G_0 hacia el vértice v en G_1. De esta forma, un camino que vaya desde el vértice s en el grafo G_0 hacia el vértice t en el grafo G_1, pasa necesariamente por uno de los arcos ficticios con costo p y, por tanto, dicho camino equivale a un camino en el grafo G donde se cambio por p el costo de exactamente un arco.

Esta idea la podemos extender a tener $k + 1$ grafos: $G_0, G_1, ..., G_k$, con arcos ficticios que unan al grafo G_i con el G_{i+1} para todo $0 \leq i < k$. A este grafo lo denominaremos G'. A un vértice v en el grafo G_i

lo denotaremos como v_i.

Noten que un camino que vaya desde el vértice s_0 (el vértice s en el grafo G_0), hacia el vértice t_i, pasa exactamente por i arcos ficticios, debido a que no existen arcos que vayan desde el grafo G_{i+1} hacia el G_i. Y un camino de estos, equivale igualmente a un camino en G con exactamente i arcos cuyo costo fue cambiado.

Entonces, como el problema pide que **a lo sumo** se cambien k arcos, basta con saber el menor de los caminos desde s_0 hacia el vértice t_i para cualquier $0 \leq i \leq k$. Para ellos utilizamos el algoritmo de *Dijkstra* sobre el grafo G', partiendo desde s_0.

Este es un pseudocódigo del algoritmo.

```
1:  n ← |V|
2:  G' ← grafo vacío de tamaño n * (k + 1)
    {Construcción del grafo}
3:  for i = 0 to k do
4:     for u, v ∈ E(G) do
5:        G'[i * n + u].append(i * n + v)
6:     end for
7:  end for
8:  for i = 1 to k do
9:     for u, v ∈ E(G) do
10:       G'[(i − 1) * n + u].append(i * n + v)
11:    end for
12: end for
    {Dijkstra}
13: d ← Dijkstra(G', s)
    {Encontrar la respuesta}
14: return mín_{0≤i≤k} d[i * n + t]
```

Noten como para construir dicho grafo se mapea el vértice v_i al entero $i * n + v$, que indica el vértice v en el i-ésimo grafo. La línea de retorno simplemente busca por todos los vértices t_i para cualquier i, el menor valor de $d[t_i]$.

Analicemos la complejidad temporal de dicho algoritmo. Sea $G' = \langle V', E' \rangle$, construir G' es $O(|V'|+|E'|)$. Y aplicar *Dijkstra* sería $O(|E'| \log |V'|)$. Mientras que la línea de retorno es $O(k)$. Analicemos cuáles son las dimensiones del grafo. Según la definición de G' se cumple que:

$$|V'| = (k+1)|V|$$
$$|E'| = (k+1)|E| + k|E| = (2k+1)|E|$$

En el caso de la cardinalidad de E', el primer sumando representa los arcos duplicados de G en cada subgrafo, mientras que el segundo sumando representa cada arco ficticio que une a dos de los subgrafos. Sustituyendo:

$$|V'| + |E'| = (k+1)|V| + (2k+1)|E| = O(k(|V| + |E|)$$

$$|E'| \log |V'| = (2k+1)|E| \log((k+1)|V|) = O(k|E| \log |V|)$$

Finalmente, por regla de la suma, la complejidad temporal del algoritmo de $O(k|E| \log |V|)$.