Algoritmos y Estructuras de Datos III

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Mayo 2017

Trabajo Práctico 1

Alumno	LU	Correo electrónico
Seijo, Jonathan Adrián	592/15	jon.seijo@gmail.com
Reyes Mesarra, Darío René	838/15	eran6reyes@gmail.com
Lucas Cordoba	94/15	lmcordobaa@gmail.com
Alexis Balbachan	994/12	alexisbalbachan@gmail.com

Índice

1. Problema 1

1.1. Introduccion

1.2. Pseudocodigo

1.3. Complejidad

1.4. Experimentos

2. Problema 2

2.1. Explicación

El contexto del problema es el siguiente: tenemos un conjunto de ciudades conectadas por rutas con peajes. El costo de un peaje es cuanto se debe pagar por transitar la ruta, y el mismo puede ser negativo (uno en vez de pagar por pasar, cobra). Decimos que hay un 'abuso' si podemos partir de una ciudad c arbitraria y volver, teniendo un saldo positivo de dinero. El problema es encontrar el máximo c que le puedo restar a todos los peajes sin que haya ningún abuso.

Notemos que podemos considerar a las ciudades como nodos, a las rutas como aristas, y a los peajes como el peso de las mismas; modelando el contexto como un grafo . Visto esto, el problema se reduce a encontrar el máximo k tal que en el grafo en el que todos los pesos decrementan en k, no hay circuito simples negativos. rotulado?

2.2. Correctitud

Supongamos que queremos calcular el camino mínimo de un nodo a a b, en un grafo G arbitrario. De haber un ciclo negativo entre ambos nodos, el algoritmo de Bellman Ford es capaz de detectarlo. Parece una buena herramienta para darnos cuenta si estamos frente a un abuso o no, pero no es tan sencillo. La sutileza consiste en notar que la ausencia de ciclos negativos entre a y b no implica que no exista ninguno en todo G. Esto puede ocurrir porque no se nos garantiza que la entrada forme un grafo orientado fuertemente conexo. Como consecuencia, podríamos llegar a tener un ciclo negativo cuyos nodos no sean alcanzables por a, con lo cual Bellman Ford no nos alcanzaría para detectar infaliblemente cuando no hay abuso.

Puede solventarse esta situación con la siguiente idea. Supongamos que tuviesemos un nodo a con un camino hacia todos los nodos de nuestro grafo. De existir un ciclo negativo, todos sus nodos serían alcanzables por a, por lo que el mismo sería detectable aplicando Bellman Ford. Asimismo, de no existir ningún ciclo negativo, el algoritmo devolvería el mínimo recorrido hacia todos los nodos partiendo de a, lo cual nos informaría que no hay ciclos negativos.

Para aprovechar esto, dado un grafo G válido en nuestro problema, podemos extender G agregándole un nuevo nodo u que tenga un arco de ida hacia todos los demás ejes. Llamemoslé a este nuevo grafo G'. Como acabamos de ver, aplicar Bellman Ford sobre u en G' nos permite determinar univocamente si hay algún circuito negativo o no. Lo que faltaría ver es que no agregamos circuitos negativos al realizar la extensión. Esto no ocurre, porque si tengo circuitos nuevos, deberán pasar por los ejes que acabamos de agregar, lo cual es absurdo, pues por construcción no hay ningún camino orientado a u. Entonces, los circuitos G y G' siguen siendo los mismos. Entonces, si aplicamos Bellman Ford desde u en G', podemos saber si hay ciclos negativos en G, o en otras palabras, si hay abuso. rotulado?

En `EXTENDERGRAFO` lo que hacemos es conseguir la extensión. Como al crear el grafo dejamos el primer nodo 'libre', lo que hacemos es usarlo como nuestro u. Agregamos a la lista de adyacencia de u a todos los nodos que representan ciudades, y les ponemos un costo arbitrario a los nuevos ejes (en particular 0). Al aplicarse esta función, el algoritmo de Bellman Ford va a aplicarse sobre este nuevo grafo, que como ya vimos, nos da la información que queremos sobre la entrada original.

La función que se encarga de implementar el algoritmo es `HAYABUSO`. Lo que hacemos en le mismo es aplicar el algoritmo de Ford $n+1$ veces, donde la última iteración se encuentra dentro de `HAYCICLONEGATIVO`, que se encarga de preguntar si las distancias mínimas de los nodos siguen cambiando. Notar que el peso que consideramos para cada eje es el peso original menos *resta*. De esta forma, estamos trabajando sobre el grafo con los pesos decrementados en *resta*, a pesar de que no lo tengamos almacenado explicitamente. Entonces, `HAYABUSO` nos devuelve si hay un ciclo negativo en el grafo restado, desde el nodo 0 hacia todos los demás nodos. Por todo lo que vimos anteriormente, esto es lo mismo que devolver si en nuestro grafo con los pesos restados hay abuso.

Visto esto, analicemos el pseudocódigo de `RESOLVER` para garantizar que devuelve la respuesta correcta. En principio, observemos que consta de una búsqueda binaria sobre el rango $[0..c+2)$, donde c es el máximo costo de peaje. Supongamos por un segundo que sabemos que el k que buscamos está en ese rango. En cada partición de nuestro espacio de búsqueda, tomamos la mitad superior incluyendo al pivote m si no hay abuso, y la mitad

inferior si lo hay. Si restando m tenemos un abuso, entonces sé que nuestro k no es m (pues estamos buscando un k en el que no se produzca abuso); y que si ya con m tenemos un abuso, con valores mayores lo seguiremos teniendo, por lo que puedo descartar a m y la mitad superior de nuestro espacio de búsqueda. Si por el contrario, restando m no nos da un abuso, entonces m es un posible candidato a k. Los valores menores a m tampoco nos darán abuso, pero son todos valores menores a k, y lo que estoy buscando es el máximo. Por ende, puedo descartar la mitad inferior de nuestro espacio de búsqueda. Entonces, lo que estamos buscando efectivamente es el máximo k tal que no hay abuso en el grafo producto de haber decrementado todos los pesos de los ejes en k.

Ahora, ¿cómo sabemos que se encuentra el k en el rango $[0..c+2)$? Bueno, no puede ser negativo, porque sabemos que sin ninguna modificación no tenemos ningún abuso. Restarle a los costos un número negativo implicaría hacer los peajes más caros, lo que haría menos posible un abuso. Tampoco puede ser mayor a $c+1$, porque como c es máximo, y quedarían todos los costos de los peajes negativos. Por el enunciado, sabemos que todas las ciudades tienen una ruta de salida, con lo cual siempre podemos asegurar que tenemos un ciclo. Si todos los peajes son negativos, esto implica que tendríamos tener algún ciclo negativo. Entonces, el k que buscamos está entre 0 y $c+1$ inclusive.

En muchos lados se menciona la obviedad de que con k más grande hay 'más' abuso, pero nunca se da una pseudo-demostración de por qué. Quizás no hace falta, tho.

2.3. Pseudocódigo

Falta agregar comentarios sobre las complejidades

Vamos a utilizar como entrada en nuestro algoritmo a las siguientes variables:

- n: La cantidad de ciudades
- $grafo$: El grafo de entrada representado con listas de adyacencia. Las posiciones del 1 a n representan nuestras ciudades, mientras que la posición 0 se deja libre para la extensión
- $costo$: La matriz con los costos de peaje, en donde en la posición (i, j) está el costo de la ruta que va desde i hasta j
- c: El máximo costo de peaje

```
function RESOLVER
    EXTENDERGRAFO(())
    d ← 0
    h ← c + 2
    while h − d > 1 do
        m ← (h + d)/2
        if ¬HAYABUSO(m) then
            d ← m
        else
            h ← m
    return d
```

```
function EXTENDERGRAFO
    for i ∈ [1..n) do
        AGREGARADELANTE(grafo[0], i)
        costo[0][i] ← 0
```

```
function HAYABUSO(resta : Int)
    dist ← INITMATRIZINFINITO(n + 1)
    dist[0] ← 0
    for i ∈ [1..n) do
        for nodo ∈ [0..n] do
            for vecino ∈ grafo[nodo] do
                peso ← grafo[nodo][vecino] − resta
                dist[vecino] ← min(dist[vecino], dist[nodo] + peso)
    return HAYCICLONEGATIVO(dist, resta)
```

```
function hayCicloNegativo(dist : Int[], resta : Int)
    for nodo ∈ [0..n] do
        for vecino ∈ grafo[nodo] do
            peso ← grafo[nodo][vecino] − resta
            if dist[vecino] > dist[nodo] + peso then return true
    return false
```

2.4. Complejidad

2.5. Experimentos

3. Problema 3

3.1. Explicación

En una provincia, hay ciudades conectadas por rutas bidireccionales. No todas estan conectadas. Se quiere que exista una única forma de llegar de una ciudad a cualquier otra. Para logar esto, se pueden **destruir** rutas existentes o **construir** nuevas rutas. La construcción y destrucción de cada ruta tiene un costo asociado, por lo que se quiere además minimizar el costo.

Para resolver el problema podemos pensarlo como un problema de grafos. La provincia es un *grafo*, donde cada ciudad es un *nodo* y las rutas son *aristas*.

Si consideramos al grafo como el completo de n nodos (donde n es la cantidad de ciudades). Que exista una y solo una ruta para llegar de una ciudad a cualquier otra, significa que tenemos que lograr que las rutas existentes formen un árbol (que incluya a todos los nodos).

Como podemos construir rutas nuevas y destruit las existentes, podríamos en principio quedarnos con cualquier árbol generador del grafo completo de n nodos. Esto hace que las rutas elegidas cumplan la condición de conexiones. Restaría considerar que las rutas elegidas tienen además que tener el mínimo costo posible.

Las observaciones claves son las siguientes:
- Si se tiene que elegir entre construir dos rutas que no existen, lo mejor es construir la mas barata.
- Si se tiene que elegir entre destruir dos ya existentes, es mejor quedarse con la mas cara de destruir.
- Si se tiene que elegir entre mantener una ruta existente o destruirla y construir otra, es mas barato manterla. Mantener una ruta cuesta 0, mientras que destruirla y construir otra tiene costo.

Teniendo esto en cuenta, podemos armar nuestro grafo de la siguiente manera:

Tomamos un grafo completo de n nodos. Las rutas que **no** existian las colocamos con su costo de construcción. Las rutas que **sí** existían las colocammos con el peso **negativo** de su destrucción. ¿Por qué el negativo? Al tratar de elegir los menores costos, se prioriza elegir una ya construida a una que no lo está, y entre dos construidas prioriza aquella que cuesta mas destruir.

La solución final es el Arbol Generador Mínimo de este grafo.

3.2. Correctitud

3.3. Pseudocódigo

3.4. Complejidad

$$O(n^2)$$

usando Prim naive

3.5. Experimentos

ideas:
0 rutas existentes
m rutas existentes
random rutas existentes

En general todo deberia dar los mismos tiempos, porque siempre construyo el arbol completo y aplico prim al completo