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 termina reduciendo a encontrar el máximo c tal que en el grafo en el que todos los pesos decrementan en c, no hay circuito simples negativos.

rotulado?

2.2. Correctitud

Supongamos que queremos saber el camino mínimo de un nodo a a b, en un grafo G arbitrario. El algoritmo de caminos mínimos de Bellman Ford es capaz de detectar si hay un circuito simple negativo sobre el que puedo ciclar para conseguir infinitamente pesos más pequeños. La sutileza consiste en notar que el hecho de que el algoritmo no detecte un ciclo negativo de a a b, no implica que no exista alguno en G. Esto puede ocurrir en grafos orientados no fuertemente conexos, en los que podría tener un ciclo negativo cuyos nodos no sean alcanzables por a. En el problema, no se nos garantiza que el grafo de entrada sea conexo por lo que no podemos aplicar Bellman Ford de forma naive para resolverlo.

Supongamos que tuviesemos un nodo a con un camino hacia todos los nodos de nuestro grafo. Entonces, de existir un ciclo negativo, todos sus nodos serían alcanzables por a. Por ende, aplicar Bellman Ford sobre a detectaría el circuito negativo, pues desde a sería posible viajar a algún nodo del mismo, y ciclar indefinidamente. Asimismo, de no existir ningún ciclo negativo, Bellman Ford devolvería el mínimo recorrido hacia todos los nodos, negando que hubiese ciclo negativo alguno.

Con esta idea, dado un grafo G válido de nuestro problema, podemos extender G a G' con un nuevo nodo u que tenga un arco de ida a todos los demás ejes. Como acabamos de ver, aplicar Bellman Ford sobre u en G' me 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. Esto 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.

rotulado?

Falta la parte de por qué hacemos búsqueda binaria

Falta conectar con el algoritmo, pero para la idea de que la extensión y usar Ford tiene sentido está

2.3. Pseudocódigo

Falta agregar comentarios sobre las complejidades

```
function RESOLVER
    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 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

Que haya una y solo una ruta para llegar de una ciudad a cualquier otra, significa que tenemos que lograr que las rutas formen un arbol. (minimizando costo total). Construir alguna ruta o destruir alguna ruta tiene un costo asociado. Quedarme con una ruta que ya existe me cuesta 0, porque no la construyo ni destruyo. Por lo tanto, lo mas eficiente es quedarme con rutas que ya existen.

¿Da igual quedarme cualquier ruta ya existente? No, porque como quiero usar la minima cantidad de rutas, es posible que necesite destruir turas que estan de mas, en ese caso voy a elegir destruir las que son mas baratas de destruir. En otras palabras, priorizo quedarme con las que son mas caras de destruir.

Entonces, a las rutas que ya existen les asigno el **negativo** costo de destruirlas, de esta forma la ruta con el nuevo costo minimo sera en realidad la ruta con mayor costo de destrucción. En caso de necesitar rutas extra, no queda otra alternativa que construir nuevas rutas con el costo de construccion dado.

La solución del problema es la siguiente: Considero el grafo **completo**. Las rutas que ya existían las coloco con su peso de destrucción negativo, y las que no existían las coloco con su peso de construcción normal.

Consiguiendo el arbol generador minimo, el costo total es el costo de destruccion de **todas las aristas que existían**, sumado a los costos (incluyendo negativos) de las aristas del arbol. Si la ruta del arbol era negativa, entonces se resta al costo total (esta bien pues ya existia), y si era positiva se suma al costo total (esta bien porque no existia)

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