75.29 - Teoría de Algoritmos I: Trabajo Práctico n. 3

Equipo Q:
Lavandeira, Lucas (#98042)
lucaslavandeira@gmail.com

Rozanec, Matias (#97404)
rozanecm@gmail.com

Sbruzzi, José (#97452)
jose.sbru@gmail.com

16.junio.2018

Facultad de Ingeniería, Universidad de Buenos Aires

Índice

Parte I

Resolución

1. Parte 1: Un juego de batalla naval

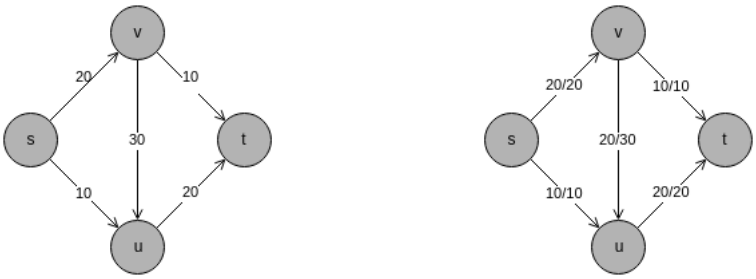

Figura 1: Red de flujo con capacidades totales y con flujo/capacidad total

2. Parte 2: Sabotaje!

2.1. Análisis preliminar y consideraciones previas

El problema planteado implica un conocimiento de redes de flujo. Una definición con la que trabajaremos es la siguiente: se entiende como red de flujo a un grafo dirigido con aristas pesadas. Llamamos al peso de cada arista la capacidad entre los dos vértices que conecta. Las redes de flujo cumplen las condiciones:

- Existe un único vértice con únicamente aristas salientes a él, sin aristas entrantes, llamado fuente.
- Existe un único vértice con únicamente aristas entrantes a él, sin aristas salientes, llamado sumidero.
- Todos los demás vértices cumplen la condición de que la capcidad entrante, es decir, la sumatoria de todas sus aristas que terminan en este vértice, es igual a la capcidad saliente, la sumatoria de todas las aristas con fuente en este vértice.

La última condición implica una conservación de la capacidad: en la red, todos los nodos intermedios entre la fuente y el sumidero tienen una capacidad neta nula, solo transportan capacidad entre las puntas, sin crear o destruir capacidad adicional.

Las redes de flujo modelan problemas en donde, valga la redundancia, fluye una cantidad apreciable de un bien sobre ciertas vías (los ejes o aristas), en donde cada vértice marca una intersección donde se puede redirigir la dirección del flujo. Las aplicaciones más comunes es en líquidos en cañerías en un problema de hidráulica, corriente eléctrica en un circuito, o el tránsito en un sistema de autovías, entre otros.

Definimos flujo de un eje como la cantidad de ese valor modelado (corriente, líquido, tránsito), que está circulando actualmente entre los nodos determinados. Este valor, por supuesto, debe ser menor a la capacidad total de ese eje, y también debe cumplir la condición de conservación: el flujo entrante a un nodo es igual al saliente.

El cuello de botella, *bottleneck*, de un camino entre fuente y sumidero es la capacidad mínima de todos los ejes del camino. Este valor es el flujo máximo que puede llegar a circular por ese camino, debido a la condición anterior de que el flujo no puede ser mayor a las capacidades de los ejes por el que pasa.

Leyendo el enunciado se toman varias suposiciones. La más importante es que la red que tenemos que proteger es una red de flujo de manera tal que cumpla con la definición dada anteriormente. Lo segundo es la interpretación de "los saboteadores quieren hacer el máximo daño posible"del enunciado. Se entiende que los saboteadores actúan de manera completamente óptima: Primero sabotearán el eje más importante de la red (a definir más adelante), y luego, el segundo más importante, que será el eje más importante de la red de flujo resultante al tener inoperante el primer eje. Dicho de otra manera, los saboteadores atacan como segundo eje al que sería más importante en una red de flujo que no contenga al primer eje saboteado.

No está de más aclarar también que la solución implementada es óptima y termina con el orden determinado únicamente cuando las capacidades de los ejes son números enteros.

2.2. Algoritmo propuesto

El problema se resume en determinar las dos aristas más importantes de la red. Es sencillo de ver que si encontramos el flujo máximo que puede pasar por nuestra red, las aristas más importantes serán las que mayor flujo pasen por ellas. El algoritmo a proponer se resume a este concepto, hallamos el flujo máximo de la red, y luego iteramos sobre todas las aristas de la red y detectamos las de mayor flujo. El flujo máximo se calcula por el algoritmo de Ford Fulkerson, con la búsqueda de caminos fuente-sumidero usando DFS.

El algoritmo implementado es óptimo gracias a la garantía de que el flujo máximo encontrado es óptimo, Ford Fulkerson determina la mayor cantidad de flujo pasable por la red. Sabiendo que los flujos son óptimos, el resultado de iterar sobre ellos devolverá las aristas más importantes también será óptimo.

Algorithm 1: Pseudocódigo del algoritmo propuesto

```
Data: G: red de flujo a determinar vulnerabilidades, n: cantidad de ejes a proteger
Result: vul: lista de ejes a proteger
R ← FordFulkerson(G); // Grafo residual
vul ← [];
while vul.length < n do
    foreach e ∈ E(R) do
        if vul.length < n then
            vul.append(e);
        else if e.flow > min(vul) then
            vul.remove(min(vul));
            vul.append(e);
return vul;
```

La complejidad del algoritmo está acotada por Ford Fulkerson, de orden $O(mC)$, siendo m la cantidad de vértices de la red, y C la capacidad saliente de la fuente. El hallado de los ejes más importantes de la red es una búsqueda lineal sobre las mismas, $O(n) = O(2m) = O(m)$. Se podría ver como una reducción sobre Ford Fulkerson: se está usando el algoritmo de flujo máximo como una caja negra, y simplemente iterando sobre sus resultados (transformación de valores de salida) para determinar la solución a nuestro problema.

La complejidad es calculada de esta manera gracias a la suposición de que las capacidades son valores enteros. El algoritmo de Ford Fulkerson utiliza esta suposición para dar su propia cota de complejidad. De ser las capacidades un valor fraccionario, ni siquiera se podría garantizar la terminación del algoritmo, debido a que en el peor caso Ford Fulkerson podría llegar a aumentar su flujo de a cantidades arbitrariamente pequeñas hasta llegar al máximo.

Analizando los flujos máximos antes y después del sabotaje podemos apreciar que el flujo no necesariamente difiere en la cantidad exacta de los flujos de las aristas saboteadas. Si borramos esas aristas, el flujo puede ser redirigido y tomar una dirección con capacidad disponible que antes no hacía porque simplemente no era conveniente.

2.3. Extensión con varias fuentes y sumideros

Se plantea la situación del mismo problema, pero ahora con una red que tenga varios nodos fuente y sumidero. Se puede ver de manera sencilla que una red con varias fuentes y sumideros es una composición de varias redes individuales de una única fuente y único sumidero.

Comencemos con una red de 1 fuente, 1 sumidero. Esta red ya fue definida anteriormente, en particular restaltamos la propiedad de conservación del flujo en cada nodo interior. Agregar una fuente y un sumidero nuevo implicaría agregar esos dos nodos, más los ejes que transporten flujos entre ellos, agregando también nodos adicionales si se quiere, pero también podrían utilizarse los mismos nodos de la red original. Agregar un eje acá puede implicar simplemente aumentar la capacidad máxima del eje entre dos nodos, no necesariamente haya que agregar una nueva arista al grafo que representa la red.

De esta manera se puede ver que para cualquier fuente de flujo existe al menos un camino desde una fuente a algún sumidero. La idea principal para resolver este caso, entonces, es descomponer a la red de múltiples fuentes - sumideros en caminos o subgrafos en donde haya una correspondencia 1:1, y luego aplicar el algoritmo de flujo máximo sobre cada uno, y sumar los resultados. Esta suma de flujos es el flujo máximo de la red original propuesta. Desde allí basta con detectar los dos ejes de mayor flujo.

La descomposición puede resultar difícil de plantear como un algoritmo pero basta modificar la Implementación actual que busca caminos entre la única fuente y el único sumidero a buscar caminos desde *cualquier* fuente hasta *cualquier* sumidero, que es equivalente a la descomposición, por lo menos para el propósito que se estaría usando en este caso. El orden del algoritmo no es modificado, sigue estando acotado por la cantidad de nodos, y la capacidad mínima de la red, debido al uso de Ford-Fulkerson internamente.

2.4. Implementación

Se implementó el algoritmo propuesto (junto con Ford Fulkerson) en Nodejs. Para correr el código, asegurarse de contar con una versión de Node >= 8.0, correr *npm i*, y luego *npm start*.