Algoritmo de Ford-Fulkerson

Pablo R. Ramis

Universidad Nacional de Rosario, Instituto Politécnico, Dto. de Informática,
prramis@ips.edu.ar,
WWW home page: http://informatica.ips.edu.ar

Resumen Veremos un problema del mundo real y la solución (como tantas otras) la ilustraremos con un grafo con pesos. Existen varios planteos de estos problemas, aquí se presentará una de las soluciones.
Supongamos un sistema de tuberías de agua (se elige esta alegoría por simpleza, podría ser una red de computadoras también) como vemos en la Figura 1. Cada arista es una tubería y el valor sobre ella es su capacidad de litros por minuto. Los vértices o nodos son puntos de uniones entre tuberías y el agua se transmite de una tubería a otra. Dos de esos nodos S y T están designados como ***fuente*** de agua y ***usuario*** o ***sumidero*** respectivamente. Esto significa que el agua se origina en S y debe ser llevada hasta T mediante las tuberías. Las tuberías tienen una sola dirección, y no hay tuberías que ingresen a S ni tuberías que salgan de T.

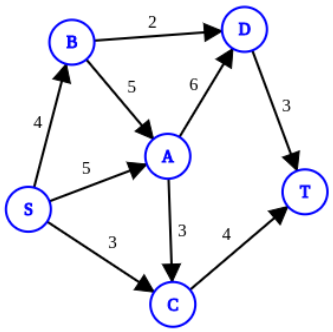


Figura 1. Red de tuberias

El objetivo es maximizar la cantidad de agua que fluye de la fuente al sumidero. Aunque la fuente puede ser capaz de producir agua en una gran proporción y el sumidero capaz de de consumirla de manera comparable, el sistema de tuberías puede no tener la capacidad para conducirla.

De esta forma vemos que el factor limitante del sistema completo es la capacidad de la tubería.
Como dijimos antes, este problema tiene múltiples aplicaciones, una red electrica, un sistema de cargas de ferrocarril, un sistema de comunicaciones o de envios.

1. La función de flujo

Se define una ***función de capacidad***, *c(a, b)*, donde a y b son nodos como sigue: si *adjacent(a, b)* es verdadero (es decir, si hay una tubería de a a b), *c(a, b)* es la capcidad de la tubería de a a b. Si no hay tubería de a a b, *c(a, b)* = 0. En cualquier punto de la actividad del sistema fluye una cantidad de agua dada (podria ser 0) a través de cada tubería.

Definimos una ***función de flujo***, *f(a, b)*, donde a y b son nodos, como 0 si b no es adyacente a a y como la cantidad de agua que esta fluyendo de a a b en caso contrario.

Con esto, resulta claro que: *f(a, b)* >= 0, además, *f(a, b)* <= *c(a, b)* ya que no se puede conducir más agua que lo que su capacidad le permite.

Si consideramos que v es la cantidad de agua que fluye a través del sistema desde S a T, entonces la cantidad de agua que emite S a través de todas las tuverías es igual a la cantidad de agua que entra a T a través de todas las tuberías y ambas cantidades son iguales a v. De esto tambien podemos desprender que si solo S puede producir agua y solo T puede absorberla, la cantidad de agua que abandona cada nodo que no sea S, es igual a la cantidad de agua que entra a dicho nodo.

Definimos el ***flujo de entrada*** de un nodo x como el total de flujo que entra a x y el ***flujo de salida*** como el flujo total que abandona x. Esto se puede resumir asi:

flujo de salida(S) = *flujo de entrada(T)* = v
flujo de entrada(x) = *flujo de salida(x)* para toda x distinto a S y T.

1.1. La función flujo

Tenemos dos maneras de ir optimizando el flujo.

Una de ellas consiste en encontrar un camino $S = x_1, x_2, ..., x_n$ de S a T de manera que el flujo a lo largo de cada arco en el camino sea estrictamente menor que la capacidad, es decir, $f(x_{k-i}, ..., x_k) < c(x_{k-i}, ..., x_k)$ para todo k entre 1 y *n-1*. Se puede incrementar el flujo en cada arco de un camino tal en el valor mínimo de $c(x_{k-i}, ..., x_k) - f(x_{k-i}, ..., x_k)$ para todo k entre 1 y *n-1* de manera que cuando el flujo ha sido incrementado a lo largo de todo el camino hay al menos un arco $< x_{k-i}, x_k >$ en el camino para el cual $f(x_{k-i}, x_k) = c(x_{k-i}, x_k)$ y a través del cual no puede incrementarse el flujo.

Esto se ve en la Figura 2 que da la capacidad y el flujo actual de manera respectiva para cada arco. Hay dos caminos de S a T con flujo positivo $[S, A, C, T]$

y $[S, B, D, T]$. Sin embargo, cada uno de esos caminos contiene un arco $< A, C >$ y $< B, D >$ en el cual el flujo es igual a la capacidad. Así, el flujo a través de esos caminos no puede perfeccionarse. Igualmente, el camino $[S, A, D, T]$ es tal que la capacidad de arco en el camino es mayor que su flujo actual. La cantidad máxima en la cual puede incrementarse el flujo a lo largo de este camino es 1, dado que el flujo a través del arco $< D, T >$ no puede exceder 3.

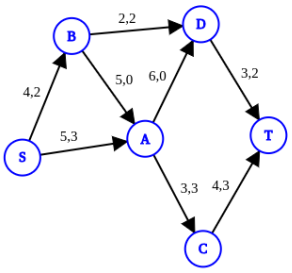


Figura 2. Incremento del flujo en la red

La función de flujo resultante se muestra en la Figura 3. El flujo total de S a T se incrementó de 5 a 6. Para ver que el resultado es aún una función de flujo válida nótese que en cada nodo (excepto T) donde se incrementa el flujo de entrada, se incrementa el flujo de salida en la misma cantidad.

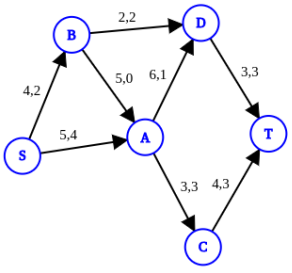


Figura 3. Función de flujo resultante

¿Hay otros caminos que pueden perfeccionarse? En este grafo, en esta red, no. Sin embargo, si volvemos al grafo original de la Figura 2 podriamos haber elegido

perfeccionar el camino $[S, B, D, T]$. La función de flujo resultante se ilustra en la Figura 4, también proporcionando un flujo neto de 6 y en consecuencia no es ni mejor ni peor que la función de flujo de la Figura 3.

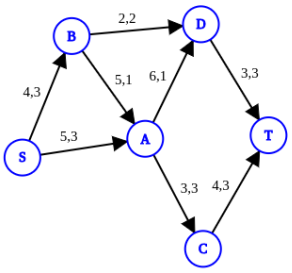
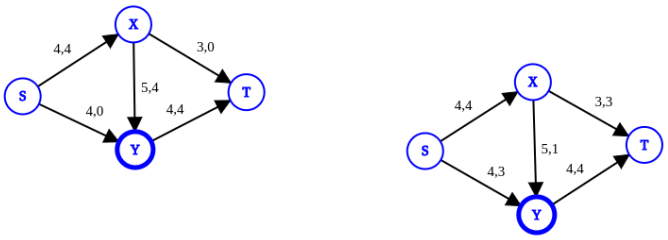


Figura 4. Función de flujo resultante (Variante)

Al inicio dijimos que había dos métodos de perfeccionamiento. Aun cuando parece no haber flujo para perfeccionarse, como se muestra en la Figura 5, podemos utilizar el segundo método. Si reducimos el flujo de X a Y, que vemos en la Figura 5(a) se puede incrementar el flujo de X a T. Para compensar el decrecimiento en el flujo de entrada de Y podria incrementarse el flujo de S a Y, incrementando por lo tanto el flujo neto de S a T.



(a) Red de flujo. (b) Incremento de flujo

Figura 5. Segundo método de perfeccionamiento

En la Figura 5(b) vemos que flujo de X a Y puede ser redirigido a T y el flujo neto de S a T puede ser por ello incrementado de 4 a 7.

Este segundo método lo podemos generalizar de la siguiente manera: Suponer que hay un camino de S a algún nodo Y, un camino de algún nodo X a T y un camino de X a Y con flujo positivo. Entonces, el flujo a lo largo del camino de X a Y puede ser reducido y esta cantidad es el mínimo del flujo de X a Y y las diferencias entre capacidades y flujo en los caminos de S a Y y X a T.

Estos dos métodos pueden ser combinados procediendo a través del grafo de S a T como sigue: la cantidad de agua que emana de S hacia T puede ser incrementada en cualquier cantidad (dado que no se asumió limite sobre la cantidad que puede producir la fuente) sólo si las tuberías de S a T aguantan el incremento. Supongamos que la tubería tiene una capacidad entre S y X que permite que la cantidad de agua que entra a X sea incrementada en una cantidad a. Si la tubería tiene capacidad para aguantar el incremento de X a T entonces se puede incrementar. Entonces si un nodo Y es adyacente a X (es decir existe una arista $< x, y >$), la cantidad de agua que emana de Y hacia T puede ser incrementada en el mínimo de a y la capacidad no usada del arco $< x, y >$. Esto es una aplicación del primer método. De manera similar, si el nodo X es adyacente a algún nodo Y, la cantidad de agua que emana de Y hacia T puede ser incrementada en el mínimo de a y el flujo existente entre Y y X. Esto puede hacerse reduciendo el flujo de Y a X como en el segundo método. Procediendo de esta manera de S a T, se puede determinr la cantidad en la que puede ser incrementado el flujo hacia T.

Definamos a un ***semicamino*** de S a T como una secuencia de nodos $S = x_1, x_2, ..., x_n = T$ tal que, para todo $0 < i <= n-1$ o $< x_{i-1}, x_i >$ o $< x_i, x_{i-1} >$ es un arco. Usando la técnica anterior, podemos describir un algoritmo para descubrir un semicamino de S a T de manera que pueda ser incrementado el flujo hacia cada nodo en el semicamino. Esto se hace construyendo sobre semicaminos parciales desde S ya descubiertos. Si el último nodo en un semicamino parcial descubierto que parte de S es A, el algoritmo considera la extensión de éste a algún nodo B tal que $< A, B >$ o $< B, A >$ sea una arista. El camino parcial se extiende A B sólo si la extensión puede hacerse de tal manera que se pueda incrementar el flujo de entrada A B. Una vez que un semicamino parcial ha sido extendido a un nodo B, se deja de considerar ese nodo como una extensión de algún otro semicamino parcial (esto ocurre porque en ese punto estamos tratando de descubrir un solo semicamino de S a T) Por supuesto, el algoritmo toma en cuenta la cantidad en la que puede incrementarse el flujo de entrada a B y si su incremento se hace en consideración al arco $< a, b >$ o al $< b, a >$.

Este proceso continúa hasta completar algún semicamino parcial que parte de S extendiendose hacia T. El algoritmo procede entonces en sentido inverso ajustando todos los flujos hasta alcanzar S. El proceso completo es entonces repetido como un intento para descubrir otro semicamino de S a T. Cuando no hay ningún semicamino parcial que pueda ser extendido con éxito, el flujo no puede ser incrementado y el flujo existente es el óptimo.

1.2. Un ejemplo

En la figura 6 se ilustra la situación inicial. Los numeros sobre las aristas indican la capacidad de cada arco y el flujo actual se encuentra en 0.

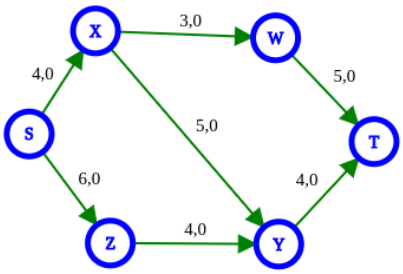


Figura 6. Red - Situación inicial

Podemos extender un semicamino de S a (S, X) y (S, Z) respectivamente. Los flujos pueden incrementarse en 4 y en 6 respectivamente. El semicamino (S, X) puede extenderse a (S, X, W) y (S, X, Y) con incrementos de flujos hacia W y Y en 3 y 4 respectivamente. El semicamino (S, X, Y) puede ser extendido hasta T con un flujo de 4. Como podemos ver, al analizar al grafo, podriamos haber elegido extender (S, X, W) a (S, X, W, T). De manera similar podriamos haber extendido (S, Z) a (S, Z, Y) en lugar de (S, X). Estas decisiones son arbitrarias.

Cómo hemos alcanzado T por medio del semicamino (S, X, Y, T) con un incremento neto de 4, incrementamos el flujo a lo largo de cada arco en este semincamino en esa cantidad como vemos en la figura 7

Ahora repetimos el proceso anterior, a partir de la situación en que se ha quedado. (S) se puede extender a (S, Z), dado que es la arista que no saturó flujo. El incremento neto a Z a través de este semicamino es 6. (S, Z) puede extenderse a (S, Z, Y) produciendo un incremento neto de 4 a Y. (S, Z, Y) no puede ser extendido a (S, Z, Y, T) dado que la arista $< Y, T >$ está a capacidad máxima. Sin embargo, puede ser extendido a (S, Z, Y, X) con un incremento neto de 4 al nodo X. Hay que advertir que como este seminacmino contiene una arista en dirección inversa $< Y, X >$, esto implica una reducción del flujo de X a Y de 4. Este semincamino resultante, (S, Z, Y, X) puede ser extendido a W con un incremento neto de 3 ya que es la capacidad que posee $< X, W >$ quedando el semicamino (S, Z, Y, X, W). Y este camino a su vez se extiende con el mismo incremento de 3 a T formando el semincamino (S, Z, Y, X, W, T). Como hemos

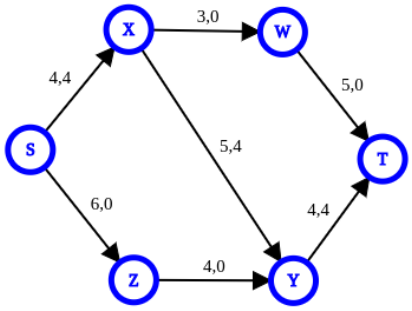


Figura 7. Red - Primera iteración

alcanzado T con un incremento de 3, procedemos en dirección inversa a través de este semincamino. Puesto que $< W, T >$ y $< X, W >$ son arcos en la dirección hacia el lavabo, su flujo puede ser incrementado en 3. Como $< X, Y >$ es un arco en dirección contraria a la que quedo el semicamino, el flujo a lo largo de $< X, Y >$ se reduce en 3, como $< Z, Y >$ y $< S, Z >$ son arcos en la dirección hacia el lavabo, su flujo puede ser incrementado en 3. Esto lo podemos ver en la Figura 8

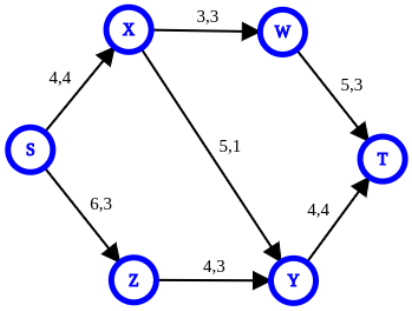


Figura 8. Red - Segunda iteración

Después intentamos repetir el proceso. (S) podria extenderse a (S, Z) ya que su capacidad no esta saturada, y de allí a (S, Z, Y) con un incremento de 1 para alcanzar la capacidad total de la arista $< Z, Y >$. De allí, no es posible extender a T ya que $< Y, T >$ esta en su capacidad total. Podriamos extender como antes a X, pero desde allí todas las aristas posibles poseen su capacidad saturada, por lo tanto se ha encontrado el flujo máximo.

Igualmente, como se dijo al principio, hubo decisiones arbitrarias en el armado de los semicaminos, otro flujo óptimo podría ser el que se muestra en la figura 9 el cual también sería válido.

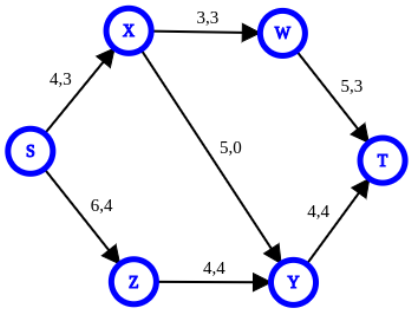


Figura 9. Red - Flujo máximo alternativo

2. El Algoritmo y el programa

Dado un grafo con pesos (podría representarse con una matriz de adyacencia y una de capacidades) con una fuente S y un sumidero T, el algoritmo para producir una función de flujo óptima para ese grafo puede delinearse de la siguiente manera:

```
1
2  Ford-Fulkerson(G,s,t) {
3
4      for (cada arco (u,v) de E) {
5          f[u,v]= 0;
6          f[v,u]= 0;
7      }
```

```
8
9       while (exista un camino p desde s a t en la red residual
            Gf) {
10          cf(p) = min{cf(u,v): (u,v) áest sobre p};
11
12          for (cada arco (u,v) en p) {
13              f[u,v]= f[u,v] + cf(p);
14              f[v,u]= - f[u,v];
15          }
16      }
17  }
```

Si bien el pseudocódigo esta muy simplificado, muestra las partes fundamentales que se deben atender. El programa se puede resolver con un par de matrices y unos arrays para guardar los datos necesarios.

```
1
2   // Matriz de adyacencia con los pesos o capacidades
3   int Grafo[V][V]
4
5   //inicia siendo igual que Grafo, pero se iran modificando los
        pesos
6   //segun el flujo
7   int Grafo_aux[V][V]
8
9   // Se armaran los semicaminos de s a T usando a los nodos
        como indices
10  // en el pseudocodigo lo llamamos p
11  int semicamino[V]
12
13  //Marcaremos con 0/1 si se visito al vertice para no volver a
        hacerlo
14  int visitado[V];
```

El armado de los semicaminos puede hacerse en una función separada que recorra al grafo y vaya marcando los vertices como visitados, para luego tomar esos arreglos y realizar los calculos de flujo.

el programa podría estructurarse de este modo:

```
1   // Ford Fulkerson
2
3   #include <stdio.h>
4   #include <limits.h>
5   #include <string.h>
6   #include "tail.h"
7
```

```
8  // úNmero de évrtices en el ágrfico dado
9
10 #define V 6
11 #define true 1
12 #define false 0
13
14 //macro que compara dos numeros y retorna el menor
15 #define min(X, Y) (((X) < (Y)) ? (X) : (Y))
16
17 //Definimos la red
18 int red[V][V] = { {0, 4, 6, 0, 0, 0},
19                   {0, 0, 0, 3, 5, 0},
20                   {0, 0, 0, 0, 6, 0},
21                   {0, 0, 9, 0, 0, 5},
22                   {0, 0, 0, 0, 0, 4},
23                   {0, 0, 0, 0, 0, 0}
24                  };
25
26 /* Devuelve true si hay un camino desde la fuente de 's'
       bajando a 't' en
27 red auxiliar. éTambin se llena de los semicamino [] para
       almacenar el camino*/
28
29 int busqueda_camino(int red_aux[V][V], int s, int t, int
       semicamino[]){
30     // Crear vector visitado y marcar todos los évrtices como
           no visitados
31
32     int visitado[V];
33     memset(visitado, 0, sizeof(visitado));
34
35     // óCreacin de una cola, évrtice fuente encola y marcar
           évrtices fuente
36     // como visitado
37
38     // Se recorre red_aux para encontrar las adyacencias y
           los semicaminos
39     // Consejo: Buscar, investigar sobre algoritmo de
           recorrido de grafos BSF
40
41
42     return (visitado[t] == true);
43 }
44 // Retorna el maximo flujo de  s a t en el grafico dado
45
46 int fordFulkerson(int red[V][V], int s, int t){
47
48 }
49
50 int main()
```

```
51  {
52      printf("El flujo ámximo posible es: %d\n", fordFulkerson(
            red, 0, 5));
53
54      return 0;
55  }
```