Flujo Máximo

Universidad Complutense de Madrid

Métodos Algorítmicos de Resolución de Problemas

Doble Grado en Matemáticas e Ingeniería Informática

Javier Pellejero
Curso 2016-2017

Índice

1. Introducción

Cuando hablamos del flujo máximo de una red de flujo, estamos tratando el campo que busca la optimización de la afluencia entre vértices de un grafo, de manera que uno o varios nodos denominados sumideros reciba la mayor cantidad de dicha afluencia siempre de acuerdo a unas restricciones de capacidad entre los nodos del grafo. Es decir, limitando la capacidad de transporte de las aristas.

En definitiva, nos son dados unos vértices conectados o no por aristas a las que queremos asignar valores de flujo maximizando la recepción de "mercancía" de los vértices o nodos sumideros desde unos vértices iniciales a los que denominaremos fuentes.

Definición 1.1. A continuación, definiremos una red de flujo, pues no es equivalente a un grafo cualquiera, sino que tiene unos requisitos muy concretos.

1) Una red de flujo es un **grafo dirigido** $G = (V, E)$ que cumple que para cada arista $(u, v) \in E$, su capacidad es mayor o igual que cero ($c(u, v) \geq 0$). Denotamos dicha propiedad como de **no negatividad**. Además, si $(u, v) \in E$ tenemos que $(v, u) \notin E$. Cabe destacar que normalizamos $c(u, v) = 0,\ \forall (u, v) \notin E$.

2) Definimos dos nodos fundamentales, el nodo **fuente** (s) y el nodo **sumidero** (t). Entre estos dos vértices existe al menos un camino formado por el resto de nodos de V; es más, para cada par de vértices de V existe al menos un camino que los conecta: tenemos que el grafo G es **conexo**. Por ser G conexo obtenemos el resultado $\text{Card}(E) \geq \text{Card}(V) - 1$, puesto que necesitamos al menos $n - 1$ aristas para unir todos los vértices de un grafo de n componentes.
El nodo fuente s solo tiene aristas salientes, por lo que si $(u, v) \in E \implies v \neq s$, mientras que el nodo sumidero t solo tiene aristas entrantes, así que si $(u, v) \in E \implies u \neq t$.

3) Definimos la **función de flujo** $f \colon V \times V \longrightarrow \mathbb{R}$ sobre el grafo G que cumple:

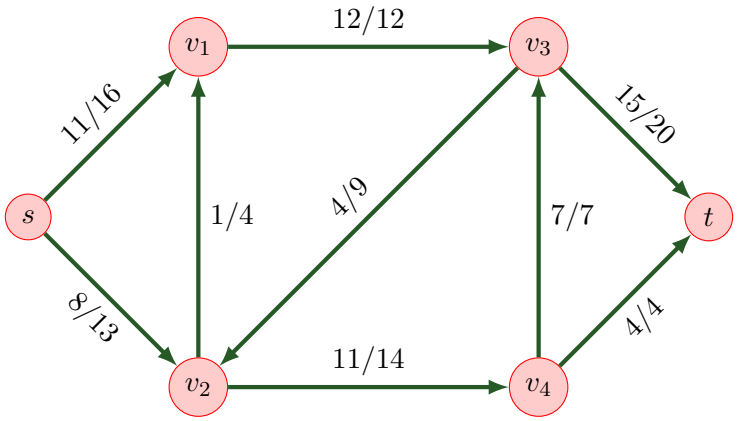
   - Para todo $u, v \in V$, se cumple que $0 \leq f(u, v) \leq c(u, v)$. A esta propiedad la denominamos **restricción de capacidad**.
   - Si una arista $(u, v) \notin E$, entonces su función de flujo es $f(u, v) = 0$.
   - Para todo $u \in V \setminus \{s, t\}$, tenemos que $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$; es decir, el flujo entrante en un nodo distinto de la fuente y el sumidero es el mismo que el flujo saliente. A esta propiedad la denominamos **conservación de flujo**.
   - Además, denotamos el valor $|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$. $|f|$ es la diferencia entre el flujo saliente y el entrante de la fuente ($|\cdot|$ no denota ni cardinalidad ni valor absoluto). Aunque, en general, no tenemos aristas dirigidas hacia la fuente $\left(\sum_{v \in V} f(v, s) = 0\right)$, lo incluimos porque en ocasiones podemos tener redes residuales en las que el flujo dirigido a la fuente puede ser significante. Es importante destacar

que $|f| = \sum_{v \in V} f(v,t) - \sum_{v \in V} f(t,v)$. Este resultado es evidente, puesto que tenemos que los vértices intermedios (distintos de s y t) tienen mismo flujo saliente que entrante, por lo tanto el flujo saliente de la fuente debe ser el mismo que el entrante en el sumidero.

Ejemplo 1.1. Veamos a continuación un ejemplo de red de flujo.
El grafo representado en la **Figura 1.1** se identifica con un problema de flujo máximo, donde cada arista dirigida contempla dos números. El primero de ellos se identifica con la función de flujo en (u,v), es decir, $f(u,v)$; mientras que el segundo con la capacidad, $c(u,v)$.

Podemos ver que se cumplen las propiedades de *restricción de capacidad* y de *conservación de flujo*.

Figura 1.1.



En particular, $0 \leq f(u,v) \leq c(u,v)\ \forall (u,v) \in E$, y además, analizando $\{v_1, v_2, v_3, v_4\} = V \setminus \{s,t\}$ tenemos:

- En v_1 tenemos la arista saliente (v_1, v_3) y las entrantes (s, v_1) y (v_2, v_1) que cumplen $f(v_1, v_3) = 12 = f(s, v_1) + f(v_2, v_1)$.
- En v_2 tenemos las aristas salientes (v_2, v_1) y (v_2, v_4) y las entrantes (s, v_2) y (v_3, v_2) que cumplen $f(v_2, v_1) + f(v_2, v_4) = 12 = f(s, v_2) + f(v_3, v_2)$.
- En v_3 tenemos las aristas salientes (v_3, t) y (v_3, v_2) y las entrantes (v_1, v_3) y (v_4, v_3) que cumplen $f(v_3, t) + f(v_3, v_2) = 19 = f(v_1, v_3) + f(v_4, v_3)$.
- En v_4 tenemos las aristas salientes (v_4, v_3) y (v_4, t) y la entrante (v_2, v_4) que cumplen $f(v_4, v_3) + f(v_4, t) = 11 = f(v_2, v_4)$.

Por último señalamos que $|f| = 19$.

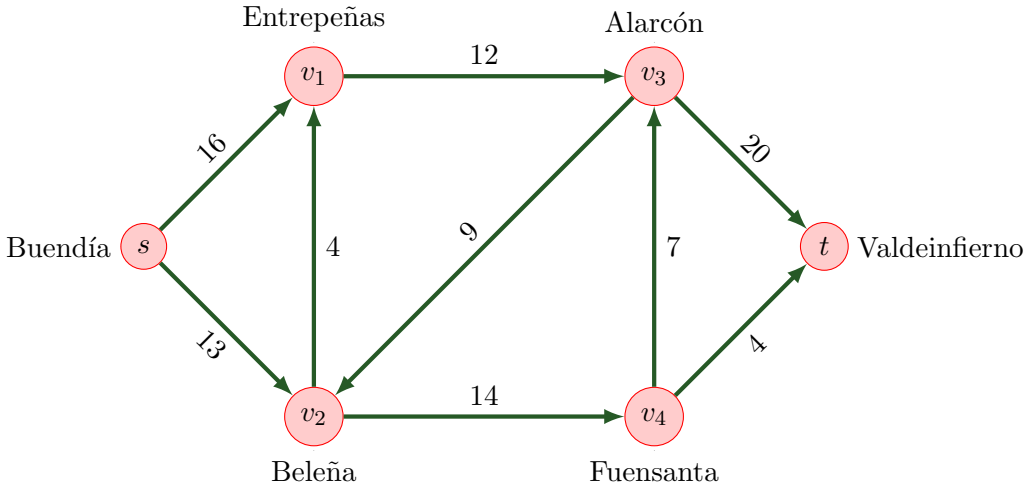
Para finalizar esta introducción plantearemos un problema basado en la **Figura 1.1**.

Ejemplo 1.2. Los regantes murcianos, aquejados de su continua sequía, no tienen con qué regar sus preciados latifundios y campos de golf en medio del árido desierto. Para resolver este problema deciden robar ingentes cantidades de agua a la humilde población castellano-manchega. El embalse de Buendía, situado entre Cuenca y Guadalajara y de gran capacidad aunque muy mermado por los constantes saqueos murcianos, parece un lugar ideal del que abastecerse. Para hacer llegar el agua a las calurosas tierras murcianas se dispone de diversos embalses y pantanos conectados por trasvases cuyos caudales diarios en cierta unidad son conocidos. Además tenemos las siguientes conexiones:

- Buendía está conectado a los embalses de Entrepeñas y Beleña.
- A su vez, Entrepeñas puede mandar agua exclusivamente al pantano de Alarcón.
- Beleña puede enviar agua a Entrepeñas y al embalse de la Fuensanta.
- Alarcón tiene trasvases hacia Beleña y hasta el destino final: Valdeinfierno (Murcia).
- Por último, el embalse de la Fuensanta envía agua a Alarcón y a Valdeinfierno.

En la **Figura 1.2** están representados los embalses y pantanos, los trasvases y los caudales de los mismos. En las siguientes secciones contemplaremos algoritmos que maximizan la solución a estos problemas, es decir, nos permitirán calcular cuál es el caudal máximo diario que puede llegar a Murcia. Adelantamos que la solución máxima a este problema (que no es complicado calcular "a ojo") es:
$f(s, v_1) = 12,\ f(s, v_2) = 11,\ f(v_1, v_3) = 12,\ f(v_2, v_1) = 0,\ f(v_2, v_4) = 11,\ f(v_3, v_2) = 0,$
$f(v_3, t) = 19, f(v_4, v_3) = 7$ y $f(v_4, t) = 4$. Por lo que podemos llevar a Murcia un máximo de 23 unidades de agua diarias.

Figura 1.2.



2. Modelación de problemas de flujo

En esta sección vamos a tratar de solucionar y normalizar algunos casos de problemas de flujo que no cumplan todas y cada una de las especificaciones dadas en la anterior sección.

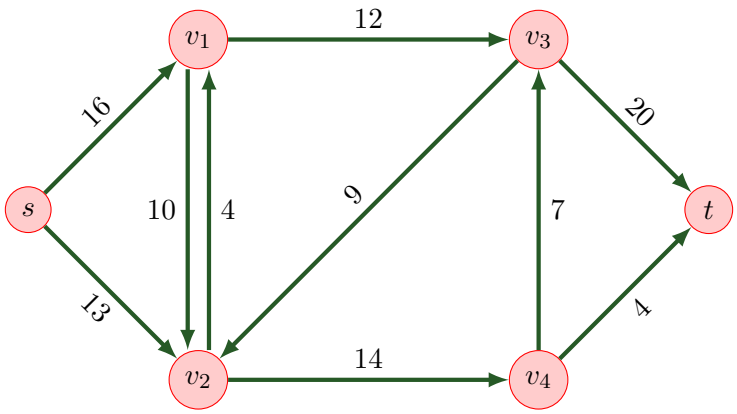
2.1. Modelación de redes con aristas antiparalelas

Decimos que dos aristas (u, v) y (v, u) son antiparalelas si ambas pertenecen a E, es decir, son paralelas pero con sentidos opuestos. En la *sección 1* indicamos que si $(u, v) \in E$ entonces $(v, u) \notin E$. Para solucionar este inconveniente tomaremos un vértice auxiliar v' y convertiremos el grafo $G = (V, E)$ en el grafo $G' = (V', E')$.

Tenemos que $V' = V \cup \{v'\}$, veamos como redefinimos E':
Puesto que (u, v) y $(v, u) \in E$, una de estas dos debe "desaparecer". Sin pérdida de generalidad quitaremos (u, v) para añadir (u, v') y (v', v) de tal manera que
$c(u, v') = c(v', v) = c(u, v)$ (a la anterior $c(u, v)$ puesto que al quitarla esta pasa a ser cero). De esta manera hemos tendido un puente auxiliar entre u y v y ahora G' ya está listo para poder trabajar sobre él como un problema de flujo máximo. Así, E' ha quedado como $E' = (E \setminus \{(u, v)\}) \cup \{(u, v'), (v', v)\}$.

Ejemplo 2.1. Veamos un ejemplo gráfico de unos posibles G con aristas antiparalelas y su normalización y G'.

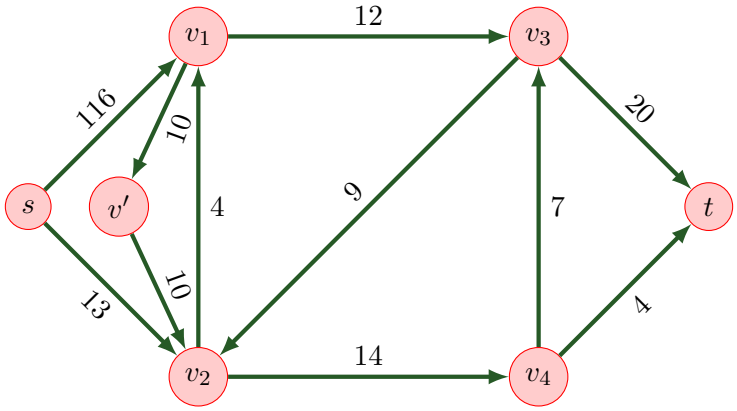
Figura 2.1.



En la **Figura 2.1** podemos observar un grafo al que denotaremos por $G = (V, E)$, en el que tenemos dos aristas antiparalelas: (v_1, v_2) y (v_2, v_1), con capacidades
$c(v_1, v_2) = 10$ y $(v_2, v_1) = 4$. Para solucionar este hecho incluimos en V el vértice v' dando lugar a V'. Tomamos la arista (v_1, v_2) y la sustituimos por las aristas (v_1, v') y (v', v_2) ambas

de capacidad 10 dando lugar a E'. Así ya tenemos el nuevo grafo G' compuesto por (V', E') que podemos observar en la **Figura 2.2**.

Figura 2.2.

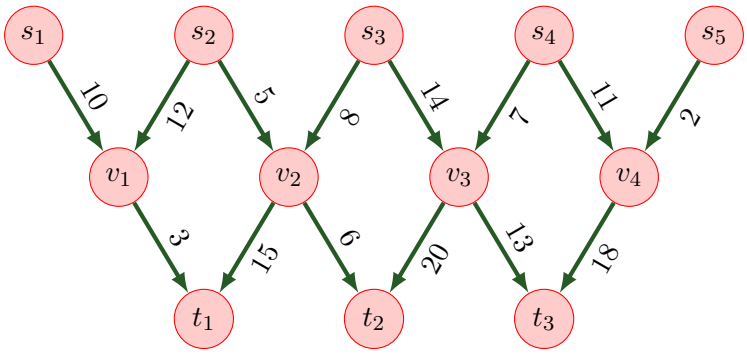


2.2. Modelación de redes con múltiples fuentes y sumideros

Una red de un problema de flujo máximo puede tener varias fuentes y sumideros. En el ejemplo de los trasvases, es altamente probable que el agua provenga originalmente de más de un embalse fuente y llegue hasta distintos embalses, pantanos y campos de regadío sumidero murcianos. Tenemos así un problema con nuestra definición original puesto que definimos una única fuente y un único sumidero.

En la **Figura 2.3** tenemos un ejemplo de un grafo con 5 fuentes, representadas por s_1, s_2, s_3, s_4, s_5 y 3 sumideros, t_1, t_2, t_3

Figura 2.3.



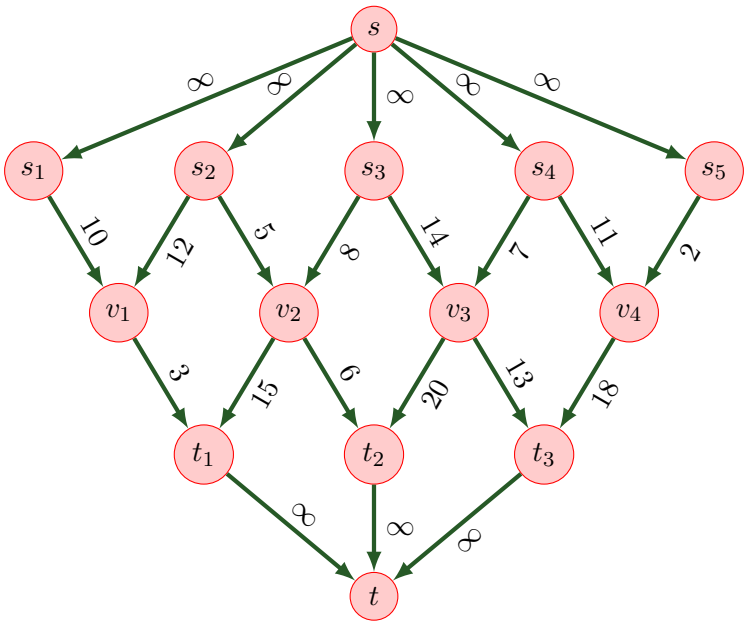
Para resolver este inconveniente basta con añadir dos vértices auxiliares a los que denotaremos s y t que denotaremos **superfuente** y **supersumidero** respectivamente. Tenemos así un nuevo conjunto de vértices $V' = V \cup \{s, t\}$ y nos falta añadir ciertas aristas a E para normalizar nuestro problema.

Nuestro nuevo E' estará compuesto por $E' = E \cup \{(s, s_i) \mid s_i \in E\} \cup \{(t_i, t) \mid t_i \in E\}$ de manera que las capacidades de estas aristas sean $c(s, s_i) = c(t_i, t) = \infty$. Hemos construido así puentes entre nuestras fuentes y la superfuente auxiliar y entre nuestros sumideros y el supersumidero auxiliar con aristas de capacidad infinita.

Intuitivamente, cualquier movimiento de flujo en el nuevo grafo $G' = (V', E')$ se comporta igual que en $G = (V, E)$. s sólo provee la suma de flujo que ya proveían los distintos s_i y t sólo "engulle" la suma de lo que ya "engullían" los t_i.

En la **Figura 2.4** podemos observar cómo queda el grafo de la **Figura 2.1** tras la adición de la superfuente y el supersumidero y sus correspondientes aristas.

Figura 2.4.



3. Representación de una red de flujo como problema de programación lineal

Aprovechando el tema de *programación lineal* que traté el cuatrimestre anterior para la composición de mi trabajo, me gustaría, antes de comenzar con los algoritmos de resolución de problemas de flujo, plantear un ejemplo en el que convertimos una de estas redes en un problema de programación lineal.

En el anterior trabajo comenté un problema de mínimo coste en una red de flujo, similar al problema que estamos tratando. Con un grafo con características similares al introducido

en la primera sección, podemos plantear un problema de mínimo coste en el que tenemos prefijadas una oferta del nodo fuente y una demanda del nodo sumidero cuyos valores son idénticos.

En este caso no tratamos de calcular el máximo flujo que podemos llevar hasta el nodo sumidero, si no que el objetivo es minimizar los costes de transportar la prefijada mercancía a través de las aristas disponibles que tienen una capacidad y unos costes por unidad de mercancía transportada conocidos.

No indagaremos más sobre este tipo de problemas puesto que ya fueron expuestos en el trabajo anterior, pero las ecuaciones lineales resultantes son similares en ambos problemas. Sin más dilación plantearemos una red de flujo máximo como un programa lineal.

Sea un grafo $G = (V, E)$ con capacidades $c(u, v)$ conocidas $\forall u, v \in V, u \neq v$ y denotamos $f(u, v)$ el suministro de mercancía que mandamos del vértice u al v.
Recordemos que un programa lineal busca maximizar o minimizar una función objetivo (en nuestro caso maximizar) de acuerdo a unas restricciones. Obtengamos primero nuestra función objetivo:

Nuestro cumplido es hacer llegar al vértice sumidero el mayor número de unidades de mercancía posible, es decir, debemos maximizar el flujo entrante en el sumidero, o debido a su equivalencia (vista en la *Sección 1*), maximizar el flujo saliente de la fuente. Esto es:

$$\text{máx}\, z = |f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

Veamos ahora las restricciones. En primer lugar tenemos que $\forall u, v \in V$:

$$0 \leq f(u, v) \leq c(u, v)$$

Y en segundo lugar, para cada vértice $u \in V$, tenemos:

$$\sum_{v \in V} f(u, v) - \sum_{v \in V} f(v, u) = 0$$

Ejemplo 3.1. Veamos para acabar un ejemplo concreto, representemos el problema de los embalses de la *Sección 1* como un problema de programación lineal.

Nuestra función objetivo busca sacar el máximo posible de agua de Buendía, así, tenemos:

$$\text{máx}\, z = f(s, v_1) + f(s, v_2) \equiv \text{máx}\, z' = f(v_3, t) + f(v_4, t)$$

En cuanto a restricciones tenemos:

$$\begin{array}{lll} 0 \leq f(s, v_1) \leq c(s, v_1) & 0 \leq f(v_2, v_1) \leq c(v_2, v_1) & 0 \leq f(v_3, t) \leq c(v_3, t) \\ 0 \leq f(s, v_2) \leq c(s, v_2) & 0 \leq f(v_2, v_4) \leq c(v_2, v_4) & 0 \leq f(v_4, v_3) \leq c(v_4, v_3) \\ 0 \leq f(v_1, v_3) \leq c(v_1, v_3) & 0 \leq f(v_3, v_2) \leq c(v_3, v_2) & 0 \leq f(v_4, t) \leq c(v_4, t) \end{array}$$

$$f(s,v_1)+f(s,v_2)=f(v_3,t)+f(v_4,t) \qquad f(v_1,v_3)=f(s,v_1)+f(v_2,v_1)$$
$$f(v_2,v_1)+f(v_2,v_4)=f(s,v_2)+f(v_3,v_2) \qquad f(v_3,v_2)+f(v_3,t)=f(v_1,v_3)+f(v_4,v_3)$$
$$f(v_4,v_3)+f(v_4,t)=f(v_2,v_4)$$

Una vez formulado el problema ya estaríamos listos para normalizar el problema a forma estándar y aplicar el algoritmo *Símplex* o cualquier otro método de resolución de problemas de programación lineal. Como hemos indicado anteriormente, todo esto ya fue tratado en las notas presentadas para el cuatrimestre anterior y no entraremos en más detalle.

4. El método Ford-Fulkerson

Vamos a introducir en esta sección el método de ***Ford-Fulkerson*** para la resolución de problemas de flujo máximo sobre un grafo $G=(V,E)$. Se trata de un método iterativo que comienza con una función de flujo nula ($f(u,v)=0\ \forall u,v\in V$) y en cada iteración incrementa el valor del flujo en G mediante la búsqueda de caminos o cadenas de aumento en un grafo asociado G_f que denotamos como **red residual**, que definiremos a continuación.

La idea del método es facilitar la búsqueda de aristas de G a las que podamos modificar su valor para aumentar el flujo de G, en definitiva, aumentar $|f|$. El cambio de flujo en dichas aristas en cada iteración no es necesariamente un incremento, sino que en multitud de ocasiones decreceremos el flujo de una o más determinadas aristas para mejorar $|f|$. El método finaliza una vez no encontremos más cadenas de aumento.

4.1. Red residual y función de aumento de flujo

Definición 4.1. Definamos a continuación el concepto de **red residual**.

Dado un grafo $G=(V,E)$ y un flujo f, definimos la red residual $G_f=(V,E_f)$ como el grafo de aristas de G con capacidades que representan el cambio de flujo que podemos realizar sobre las aristas de G. Una arista del grafo G puede admitir un aumento adicional de flujo igual a la capacidad de la arista menos el flujo que ya transporta. Así, las únicas aristas que pertenecen a E_f son las que admiten más flujo. Definimos entonces
$c_f(u,v)=c(u,v)-f(u,v)\ \forall u,v\in V$.

Sin embargo, como comentamos anteriormente, existe la posibilidad de que queramos decrecer el flujo de una arista para obtener el bien mayor de maximizar el flujo de nuestra red. Surge así la necesidad de añadir ciertas aristas a G_f que no pertenecen a G. Para representar el posible decrecimiento de una arista $(u,v)\in E$ con flujo positivo tomaremos $(v,u)\in E_f$ de manera que $c_f(v,u)=f(u,v)$. Es decir, podemos llevar flujo de v a u, en sentido contrario al original, que sería equivalente a hacer disminuir el flujo de (u,v).

Definamos formalmente la capacidad de cada arista de G_f:

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{si } (u,v) \in E \\ f(v,u) & \text{si } (v,u) \in E \\ 0 & \text{en otro caso} \end{cases}$$

Recordemos que si $(u,v) \in E \implies (v,u) \notin E$, luego la anterior definición está bien construida.

Ejemplo 4.1. Sea una arista (u,v) de cierto grafo G tal que su capacidad $c(u,v) = 16$ y su flujo $f(u,v) = 11$, entonces $c_f(u,v) = 5$ y $c_f(v,u) = 11$.

Por último, recalcar que una vez calculado $c_f(u,v)$ $\forall u, v \in V$ podemos definir $E_f = \{(u,v) \in V \times V \mid c_f(u,v) > 0\}$ y que por la definición, es fácil ver que $\text{Card}(E_f) \leq 2 \cdot \text{Card}(E)$.

Definida la red residual por el grafo G_f, tenemos que G_f no cumple la definición de red de flujo dada en la *Sección 1*, puesto que en múltiples ocasiones nos encontraremos con $(u,v), (v,u) \in E_f$ para ciertos $u, v \in V$. Sin embargo, podemos definir una función de flujo f' sobre ella de manera que cumpla la propiedad de **restricción de capacidad** que es la que nos interesa de cara al desarrollo de estas ideas.

Definición 4.2. Definimos el **aumento de flujo de f por f'**, que denotamos $(f \uparrow f')$ como:

$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{si } (u,v) \in E \\ 0 & \text{en otro caso} \end{cases}$$

Proposición 4.1. Sea $G = (V,E)$ un grafo de una red de flujo y f su función de flujo. Sea G_f la red residual de G inducida por f y f' la función de flujo de G_f. Entonces la función de aumento $f \uparrow f'$ es una función de flujo sobre G bien definida con valor $|f \uparrow f'| = |f| + |f'|$.

La demostración de esta proposición, pese a no ser tediosa, no es lo suficientemente corta como para incluirse en estas notas. Puede consultarse en *Cormen, T., Leiserson, C., Rivest, R. and Stein, C. (2009). Introduction to algorithms*, capítulo 26, páginas 717, 718 y 719.

4.2. Cadenas de aumento y cortes

Definición 4.3. Procedamos a definir el concepto de **cadenas de aumento**.
Una cadena de aumento p es un camino entre los vértices s y t de la red residual G_f. Por cómo hemos definido G_f y f', una cadena de aumento p de G_f es un incremento de flujo para la red de G siempre que no violemos ninguna capacidad $c_f(u,v)$.

Para asegurarnos de que dicha violación de capacidad no ocurra, definimos la **capacidad residual de** p como $c_f(p) = \text{mín}\{c_f(u,v) \mid (u,v) \text{ está en } p\}$.

Veamos en la siguiente proposición un argumento más preciso de qué hemos logrado al encontrar una cadena de aumento.

Proposición 4.2. Sea $G = (V, E)$ una red de flujo y f un flujo en G. Sea G_f la red residual de G y p una cadena de aumento en G_f. Definamos $f_p \colon V \times V \longrightarrow \mathbb{R}$ tal que

$$f_p(u, v) = \begin{cases} c_f(p) & \text{si } (u, v) \text{ está en } p \\ 0 & \text{en otro caso} \end{cases}$$

Entonces f_p es una función de flujo en G_f con valor $|f_p| = c_f(p) > 0$ y $f \uparrow f_p$ es un aumento de flujo de f. Además tenemos que $|f \uparrow f_p| = |f| + |f_p| > |f|$, por lo que hemos mejorado el flujo en G que teníamos anteriormente.

Definición 4.4. Concluyamos las definiciones de esta sección explicando en qué consiste un corte.
Un **corte** (S, T) **de una red de flujo** $G = (V, E)$ es una partición de V en dos conjuntos S y $T = V \setminus S$ (es decir, se cumple que $S \cup T = V$ y $S \cap T = \emptyset$) tal que $s \in S$ y $t \in T$.

Si f es un flujo en G, definimos el **flujo neto** $f(S, T)$ a través del corte (S, T) como

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u)$$

Además definimos la **capacidad** del corte (S, T) como $c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v)$.

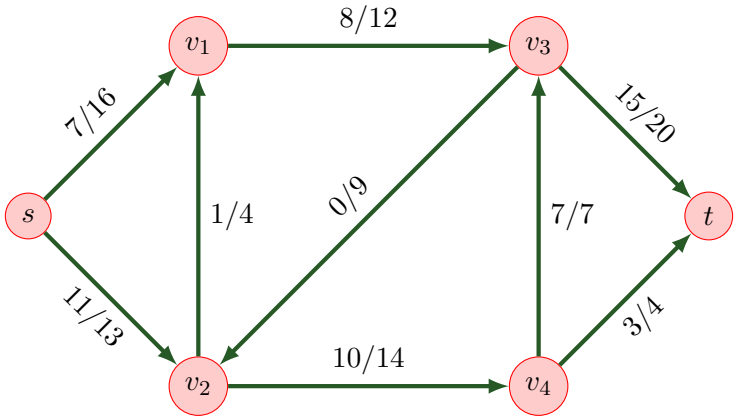
Proposición 4.3. Sea f un flujo en una red G y (S, T) un corte cualquiera de G, entonces el flujo neto $f(S, T) = |f|$.

De nuevo la demostración puede verse en *Cormen, T., Leiserson, C., Rivest, R. and Stein, C. (2009). Introduction to algorithms*, capítulo 26, página 722.

Es decir, nos es indiferente el corte que realicemos en G, siempre obtendremos el mismo flujo neto que es equivalente al flujo $|f|$ de G. Además no es difícil comprobar que $|f| = f(S, T) \leq c(S, T)$.

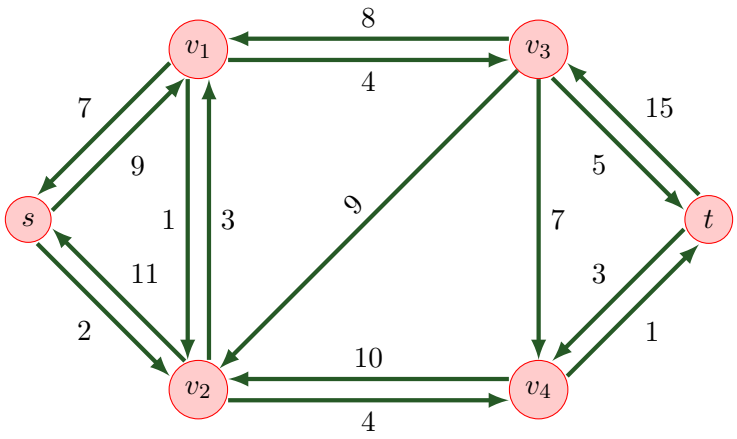
A continuación enunciaremos un importante teorema que nos ayudará a saber si ya tenemos o no un flujo máximo para nuestra red G, pero antes veremos algunos ejemplos gráficos sobre el material introducido en esta sección.

Figura 4.1.



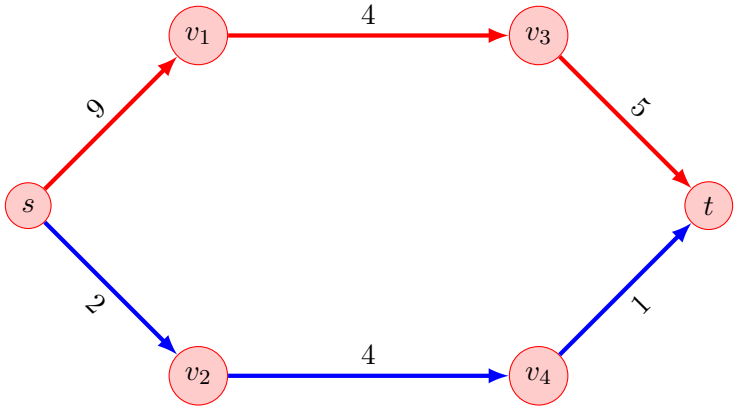
En la **Figura 4.1** hemos representado una red de flujo $G = (V, E)$ con una función de flujo y sus capacidades. Veamos como queda su red residual G_f.

Figura 4.2.



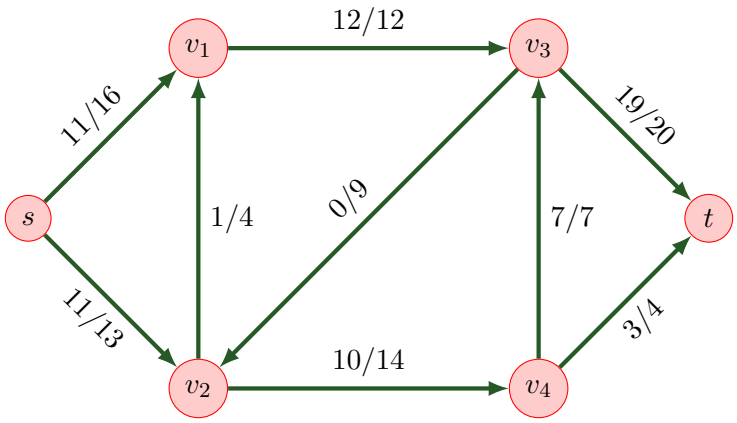
En la **Figura 4.2** tenemos representado el grafo residual G_f derivado del grafo G y su función de flujo f.

Figura 4.3.



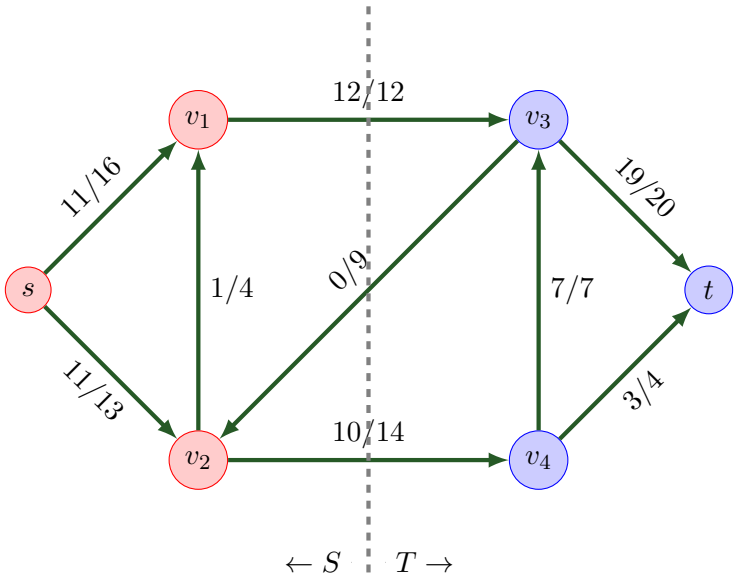
En la **Figura 4.3** hemos dibujado dos cadenas de aumento (en rojo y azul respectivamente). Si denotamos el camino rojo como p_1, tenemos que $c_f(p_1) = 4$ y que $f_{p_1}(s, v_1) = f_{p_1}(v_1, v_3) = f_{p_1}(v_3, t) = 4$. Mientras que si denotamos el azul por p_2, obtenemos $c_f(p_2) = 1$ y que $f_{p_2}(s, v_2) = f_{p_1}(v_2, v_4) = f_{p_1}(v_4, t) = 1$. Nos quedaremos por ejemplo con el rojo y así surge el flujo de aumento $f \uparrow f_{p_2}$ presente en la **Figura 4.4**.

Figura 4.4.



En este caso hemos aumentado de un flujo $|f| = 18$ a 22. Podríamos redibujar la nueva red residual del grafo G resultante y continuar buscando otra cadena de aumento, así hasta que no encontráramos ninguna otra.

Figura 4.5.



Para finalizar con estos ejemplos, en la **Figura 4.5** hemos practicado un corte (S,T) sobre el grafo de la **Figura 4.4**. Su flujo neto $f(S,T) = \sum_{u\in S}\sum_{v\in T} f(u,v) - \sum_{u\in S}\sum_{v\in T} f(v,u) =$ $= f(v_1,v_3) + f(v_2,v_4) - f(v_3,v_2) = 12 + 10 - 0 = 22$. Mientras que su capacidad de corte es $c(S,T) = \sum_{u\in S}\sum_{v\in T} c(u,v) = c(v_1,v_3) + c(v_2,v_4) = 12 + 14 = 26$.

4.3. Teorema del flujo máximo y corte mínimo

Teorema 1. *Sea f un flujo en una red $G = (V,E)$ con una fuente s y un sumidero t, entonces son equivalentes:*

1) f es máximo flujo en G.

2) La red residual G_f no contiene cadenas de aumento.

3) $|f| = c(S,T)$ para cierto corte (S,T) de G.

Demostración.

- (1 $\implies$ 2) Supongamos que no. Supongamos que f es flujo máximo en G y existe una cadena de aumento p en G_f. Entonces existe una función de flujo f_p en G_f y por la **Proposición 4.2**, la función de flujo $f \uparrow f_p$ está bien definida sobre G y $|f \uparrow f_p| = |f| + |f_p| > |f|$ lo que contradice que f sea flujo máximo.

- ($2 \implies 3$) Supongamos que no existen cadenas de aumento, esto es que no hay ningún camino de s a t en G_f. Definimos $S = \{v \in V \mid \text{existe un camino de } s \text{ a } v\}$, evidentemente $t \notin S$ puesto que no existe ningún camino de s a t y definimos $T = V \setminus S$. (S,T) es un corte y para cada par (u,v) con $u \in S, v \in T$, $(u,v) \notin E_f$, puesto que no existe un camino entre ellos. Tenemos que si $(u,v) \in E$, por lo anterior, $f(u,v) = c(u,v)$. Si $(v,u) \in E$, entonces $f(v,u) = 0$ ya que en otro caso (u,v) pertenecería a E_f y esto no es posible. Así tenemos que
$f(S,T) = \sum_{u\in S}\sum_{v\in T} f(u,v) - \sum_{u\in S}\sum_{v\in T} f(v,u) = \sum_{u\in S}\sum_{v\in T} f(u,v) = \sum_{u\in S}\sum_{v\in T} c(u,v) = c(S,T)$.
- ($3 \implies 1$) Por la **Proposición 4.3** tenemos que $|f| \leq c(S,T)$ para cualquier corte (S,T) de G, como $|f| = c(S,T) \implies f$ es máximo.

□

4.4. Implementación en pseudocódigo del método Ford-Fulkerson y complejidad

Para finalizar esta sección veamos un algoritmo del método ***Ford-Fulkerson*** implementado en pseudocódigo. Como observaremos, el código es bastante simple.

```
/*********************
* FordFulkerson (G, s, t, f)
*********************/
for cada arista (u, v) ∈ G.E do
    f(u, v) = 0
end for
while ∃p cadena de aumento en G_f do
    c_f(p) = mín{c_f(u, v) | (u, v) está en p}
    for cada arista (u, v) en p do
        if (u, v) ∈ G.E then
            f(u, v) = f(u, v) + c_f(p)
        else
            f(u, v) = f(u, v) − c_f(p)
        end if
    end for
end while
```

Para la búsqueda del camino podemos utilizar uno de los algoritmos de búsqueda en profundidad sobre el grafo G_f vistos en clase. Recordemos que el coste de cualquiera de estas búsquedas tiene $\mathcal{O}(V + E')$ que podemos reducir a $\mathcal{O}(E')$, puesto que en general tendremos muchas más aristas que vértices. Además recordemos que $\text{Card}(E') \leq 2 \cdot \text{Card}(E)$, luego $\mathcal{O}(E') = \mathcal{O}(2E) = \mathcal{O}(E)$. Por último, el número máximo de veces que buscaremos si hay o no este camino será cuando lleguemos a alcanzar el flujo máximo, es decir, $|f|$. Luego el orden de ***Ford-Fulkerson*** puede ser definido como $\mathcal{O}(|f|E)$.

5. Algoritmo de Edmonds-Karp

En esta sección hablaremos del **algoritmo de *Edmonds-Karp*** que es una implementación del método *Ford-Fulkerson*. Como acabamos de ver, la complejidad de la implementación anterior depende de $|f|$ y, aunque es esperado que las cadenas de aumento aumenten en varias unidades en cada iteración, es un coste muy caro para flujos máximos muy altos.

Por eso surge la necesidad de encontrar una implementación del método que no dependa del tamaño de nuestro máximo, y es aquí donde entra en juego el algoritmo de *Edmonds-Karp*. La implementación de este algoritmo es básicamente como la anterior, sólo que la búsqueda de caminos la realizaremos concretamente con una *búsqueda en anchura* en lugar de una *en profundidad* o cualquier otra.

Esta búsqueda nos arroja el camino **más corto** de s a t de la red residual G_f. Denotaremos por $\delta_f(u, v)$ como la distancia del camino más corto de u a v, por lo que la búsqueda en anchura nos proporciona un camino entre s y t con $\delta_f(s, t)$ mínimo.

Proposición 5.1. Sea una red de flujo $G = (V, E)$ y aplicamos *Edmonds-Karp* sobre ella, entonces, para todo vértice $v \in V \setminus \{s, t\}$, el camino más corto de distancia $\delta_f(s, v)$ en la red residual G_f es monótonamente creciente. Es decir, si $\delta_{f'}(s, v)$ es la distancia mínima una vez aplicado el aumento de flujo, entonces $\delta_f(s, v) \leq \delta_{f'}(s, v)$.

La proposición parece intuitiva, puesto que las únicas aristas modificadas de la red residual son las del camino más corto, cuya arista con capacidad $c_f(u, v)$ mínima desaparecerá y aparecerán tal vez otras pero en sentido contrario al camino (las que nos sirven para decrementar el flujo de una arista). No desarrollaremos una demostración formal, esta puede consultarse en *Cormen, T., Leiserson, C., Rivest, R. and Stein, C. (2009). Introduction to algorithms*, capítulo 26, página 728.

Comentemos ahora una proposición más importante, con ella obtendremos que el coste del algoritmo *Edmonds-Karp* está en $\mathcal{O}(VE^2)$.

Proposición 5.2. Sea una red de flujo $G = (V, E)$ y aplicamos *Edmonds-Karp* sobre ella, entonces, el número total de aumentos de flujo está en $\mathcal{O}(VE)$. Por tanto, como el coste de la búsqueda en anchura está en $\mathcal{O}(E)$, el algoritmo *Edmonds-Karp* tiene orden $\mathcal{O}(VE^2)$.

Demostración.
Decimos que (u, v) es una **arista crítica** de G_f en un camino de aumento p si $c_f(p) = c_f(u, v)$. Por cada aumento tenemos al menos una arista crítica que desaparece tras el aumento. Si dicha arista no pudiera reaparecer, entonces el número de aumentos estaría en $\mathcal{O}(E)$, pero, como sabemos, sí que pueden reaparecer siempre que otro camino incluya la arista (v, u). Planteemos esta situación.

Sea (u, v), $u \neq t$, una arista crítica por primera vez, tenemos que antes del aumento se cumple $\delta_f(s, v) = \delta_f(s, u) + 1$. Tras el aumento de flujo, (u, v) reaparecerá si el flujo en

dicha arista decrece, es decir, si (v, u) aparece en un camino de aumento. Sea f' el flujo cuando esto ocurre.

Tenemos ahora que $\delta_{f'}(s, u) = \delta_{f'}(s, v)+1$ y por la **Proposición 5.1** $\delta_f(s, v) \leq \delta_{f'}(s, v)$. Esto implica que $\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1 \geq \delta_f(s, v) + 1 = \delta_f(s, u) + 2$.

Esto es que entre desaparición y desaparición (entre que (u, v) es crítica, reaparece y vuelve a ser crítica), la distancia entre s y u ha aumentado al menos 2. Hemos tomado $u \neq t$, luego al principio, la distancia entre s y u era al menos cero, mientras que la distancia máxima entre s y un vértice distinto de t es a lo sumo $\mathrm{Card}(V) - 2$. Por lo tanto, tras ser crítica una primera vez, puede llegar a serlo $(\mathrm{Card}(V) - 2)/2 = \mathrm{Card}(V)/2 - 1$ veces más, es decir, un total de $\mathrm{Card}(V)/2$ veces en total.

Puesto que el cardinal de las aristas en una red residual es a lo sumo 2 veces el cardinal de las aristas en la red de flujo ($\mathrm{Card}(E_f) \leq 2 \cdot \mathrm{Card}(E)$), hemos probado que el número de aumentos está en $\mathcal{O}(VE)$ y por tanto *Edmonds-Karp* tiene orden $\mathcal{O}(VE^2)$. □

6. Algoritmos push–relabel y relabel-to-front

En esta última sección vamos a ver, sin profundizar, unos algoritmos con mejores costes que los vistos anteriormente. El algoritmo *push–relabel* trabaja en $\mathcal{O}(V^2E)$ mejorando el $\mathcal{O}(VE^2)$ del algoritmo *Edmonds-Karp* (recordemos que en general el cardinal de V es mucho menor que el cardinal de E). Veremos también el algoritmo *relabel-to-front*, variación de *push–relabel* con mejor coste: $\mathcal{O}(V^3)$.

6.1. Algoritmo push–relabel

En este algoritmo no son válidas algunas de las definiciones dadas en la *Sección 1*. Para empezar, sustituimos la función de flujo por la siguiente.

Definición 6.1. Definimos la función **preflujo** como $f\colon V \times V \longrightarrow \mathbb{R}$ que cumple:

- $\forall u \in V \setminus \{s\}$ tenemos que $\sum_{v \in V} f(v, u) - \sum_{v \in V} f(u, v) \geq 0$.

Definimos como **exceso de flujo** al valor $e(u) = \sum_{v \in V} f(v, u) - \sum_{v \in V} f(u, v)$. Decimos que un vértice $u \in V \setminus \{s, t\}$ está **desbordado** si $e(u) > 0$.

Definición 6.2. Sea un preflujo f sobre una red de flujo $G(V, E)$, definimos **función de peso** o **función de distancia** como $h\colon V \longrightarrow \mathbb{N}$ que cumple:

- $h(s) = \mathrm{Card}(V)$.
- $h(t) = 0$.

- $h(u) \leq h(v) + 1$ para toda arista residual $(u, v) \in E_f$. Luego si $h(u) > h(v) + 1 \implies$ $\implies (u, v) \notin E_f$.

No entraremos más en detalle acerca de este algoritmo, procederemos a explicar la intuición del mismo.

El algoritmo trata el problema vértice a vértice trabajando con sus vecinos en lugar de con todo el grafo. El objetivo es llevar la mayor cantidad posible de flujo de un vértice a sus vecinos, de ahí que pueda producirse el desbordamiento de flujo en cada iteración. Siempre llevamos flujo de vértices con "pesos" más altos a "pesos" más bajos ("cuesta abajo"), es decir, según el valor de h en dos determinados vértices llevamos flujo de uno a otro o no. A esta operación se le denomina **empuje** (*push*). Cuando no tenemos vértices con pesos más bajos a los que llevar flujo entra en juego la operación de **reetiquetado** (*relabel*), consistente en aumentar el peso del vértice en el que nos encontramos, y comenzamos de nuevo nuestra operación de empuje.

6.2. Algoritmo relabel-to-front

Este algoritmo es una variante de *push–relabel*. En este caso, en lugar de un recorrido vértice a vértice, poseemos una lista enlazada con los vértices de la red ordenados por su admisión de flujo. El método recorre la lista y realiza la operación de empuje sobre los nodos con desbordamiento. Si un nodo es reetiquetado, se mueve al principio de la lista y comenzamos a recorrer de nuevo la lista desde el principio.

7. Bibliografía

1. *Cormen, T., Leiserson, C., Rivest, R. and Stein, C. (2009). Introduction to algorithms, chapter 26.*

2. *Sedgewick, R., Wayne, K. (2011). Algorithms, chapter 6.4.*