Resumen Flujo en Redes

Tomás F Melli

November 2024

Índice

1 Introducción **2**

2 Problema de Flujo Máximo **3**
2.1 Corte Mínimo . . . 3
2.1.1 Capacidad de un Corte . . . 4
2.1.2 Problema de Corte Mínimo . . . 5
2.2 Problema DUAL (relación entre Flujo Máximo y Corte Mínimo) . . . 6
2.3 Red Residual . . . 7

3 Camino de Aumento **8**
3.1 Delta del Camino de Aumento . . . 9

4 Algortimo de Ford Fulkerson **11**
4.1 Proposiciones . . . 20
4.2 Análisis de Complejidad . . . 20
4.3 Código C++ . . . 21

5 Algoritmo de Edmonds y Karp **22**
5.1 Análisis de Complejidad . . . 22
5.2 Código C++ . . . 22

6 Matching Máximo en Grafos Bipartitos **23**

1 Introducción

Muchos sistemas tienen limitaciones de capacidad y estructuras de conexión. De ahí nace la necesidad de modelar una solución en la que cierto recurso se mueva eficientemente de un punto a otro. A partir de esto surge el concepto de **red**. Una red es **un grafo orientado conexo que tiene dos vértices distinguidos, una fuente s con grado de salida positivo y un sumidero t con grado de entrada positivo**:

$$\text{Sea } N = (V, E) \text{ y } s \in V \text{ tal que } \deg^+(s) > 0 \text{ y } t \in V \text{ tal que } \deg^-(t) > 0.$$

Dicho esto, la idea es modelar que desde cierto vértice (fuente) se mueve el recurso y atraviesa toda una estructura de conexiones hasta llegar a destino (el sumidero). Existe una limitación que es la **capacidad** que tiene cierta arista para transportar el recurso, y se define la **función de capacidad** :

$$\text{Sea } c : E \to Z^+ \text{ una función de capacidad que asigna a cada arista una capacidad positiva.}$$

El valor de **flujo** es esa cantidad de recurso que circula por cierta red y se define la función de **flujo factible** :

$$\text{Sea } x : E \to Z^+ \text{ un flujo factible en la red, donde } x(e) \text{ representa el flujo en la arista } e \in E.$$

$$0 \leq x(e) \leq c(e) \quad \text{para toda arista } e \in X$$

Es decir, que para poder considerar ese valor de flujo de la red factible, se tienen que cumplir dos condiciones, que el flujo que circula por cierta arista e no sea mayor a la capacidad para transportar ese recurso y que cumpla con que:

En el medio **no se pierde recurso**, es lo que llamamos **ley de conservación de flujo** :

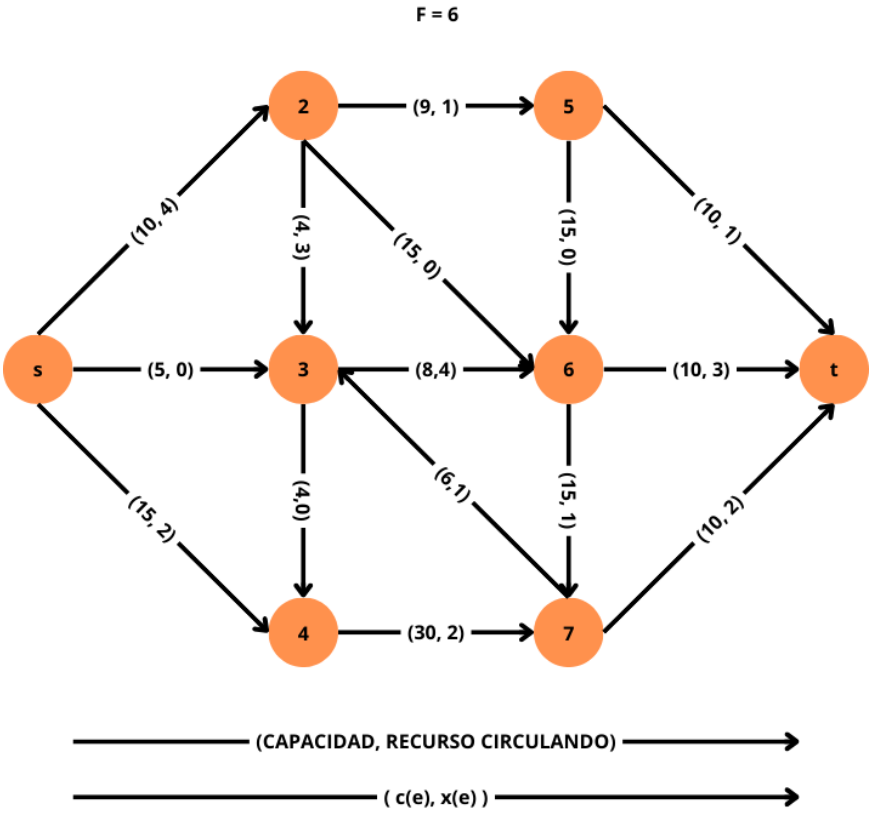
$$\forall v_i \in V - \{s, t\}, \quad \sum_{u \in N^-(v)} x(v \to u) = \sum_{u \in N^+(v)} x(u \to v)$$

Es decir, que para todo vértice que no es ni fuente ni sumidero, se va a cumplir que si sumamos todo el recurso que ingresa (que se representa como la sumatoria de los valores de flujo de las aristas que llegan a cierto vértice) esa suma, nos va a dar lo mismo que todo el recurso que egresa de cierto vértice (que la representamos con la suma de todos los valores de flujo de las aristas que salen de cierto vértice). **Aclaracioón : vale que del sumidero salgan arcos, en el ejemplo no se observará, pero existe la posibilidad que suceda. Lo que debe valer siempre es que el grado de entrada sea positivo para el sumidero.

A nosotros nos va a interesar **cuánto de este recurso está siendo transportado por la red desde la fuente hacia el sumidero**. Esta magnitud la representamos con la **función de flujo** que representa el **flujo neto desde la fuente hacia el sumidero** :

$$F = \sum_{e \in N^+(t)} x(e) - \sum_{e \in N^-(t)} x(e)$$

**Es lo mismo hacerlo sobre t que sobre s.



2 Problema de Flujo Máximo

El problema de flujo máximo consiste en, dada una red N = (V, E) con capacidades en los arcos y dos vértices específicos, una fuente s, y un sumidero t, **determinar el flujo factible de valor máximo F que se puede enviar de s a t.** En el ejemplo vimos que F = 6, pero es realmente 6 la máxima cantidad de recurso que puede ser enviada desde s a t respetando las capacidades de los arcos (es decir, cumpliendo con la propiedad de flujo factible)?

2.1 Corte Mínimo

Un **corte** en una red N es un subconjunto $S \subseteq V \setminus \{t\}$, tal que $s \in S$. Este subconjunto tiene a la fuente pero no al sumidero. Es un subconjunto de vértices que desconectan a s y a t. Se da la particularidad que los vértices que pertenecen al corte tienen como vecinos a vértices que pertenecen a T, el subconjunto que contiene al sumidero.

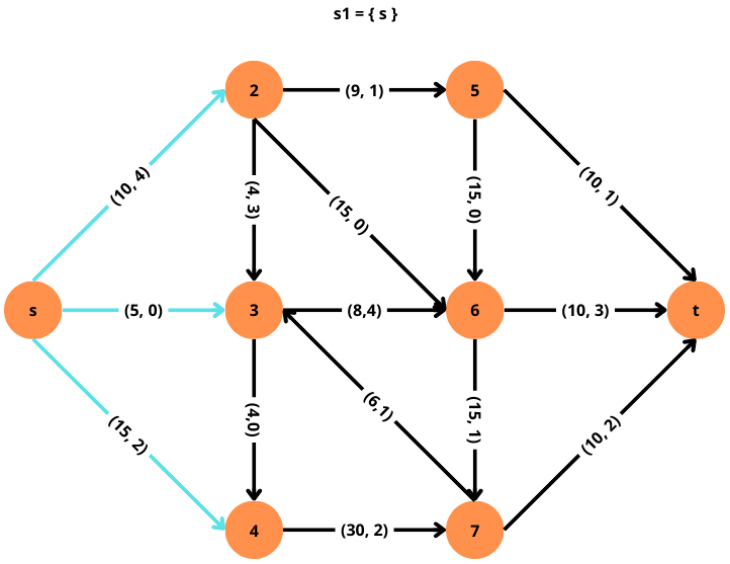
$$\text{Dados } S, T \subseteq V \text{ llamaremos } ST = \{(u \to v) \in E : u \in S \text{ y } v \in T\}$$

Es decir, la notación ST quiere decir que : dados dos conjuntos de vértices, **ST es el conjunto de arcos que salen de algún vértice de S y entran en algún vértice de T.**
Existe una relación entre Flujo Máximo y Corte Mínimo, esta es que, dado un flujo factible y S un corte, entonces el valor de flujo factible es igual al flujo que esta pasando sobre los arcos que salen de S y van a $\overline{S}$ (cruzan la frontera de S) menos el flujo de los arcos que van de $\overline{S}$ a S, dado cualquier flujo factuble y cualquier corte.

Proposición : Sea x un flujo factible definido en una red N con valor F y sea S un corte, entonces

$$F = \sum_{e \in S\overline{S}} x(e) - \sum_{e \in \overline{S}S} x(e) \quad \text{donde} \quad \overline{S} = V \setminus S.$$

Ejemplos :

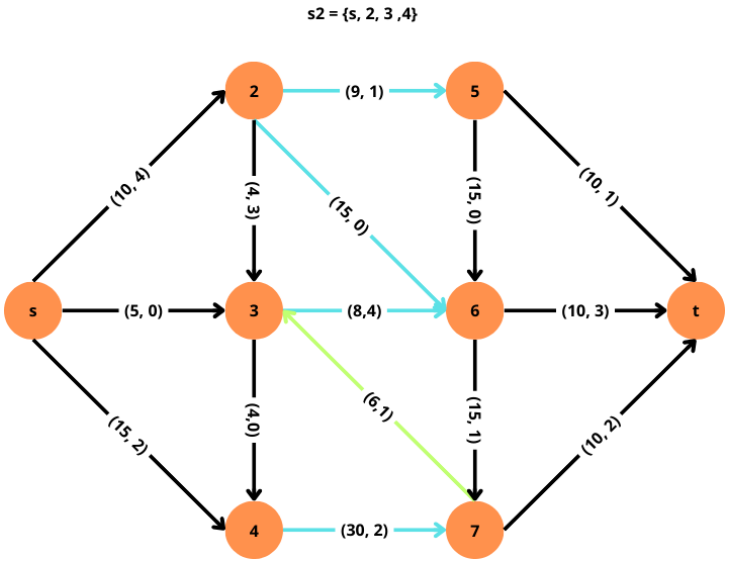


En este caso, la suma de recurso que sale del corte - la suma de recurso que entra al corte es :

$$F = \sum_{e \in S\overline{S}} x(e) - \sum_{e \in \overline{S}S} x(e)$$

$$F = 6 - 0$$

$$F = 6$$

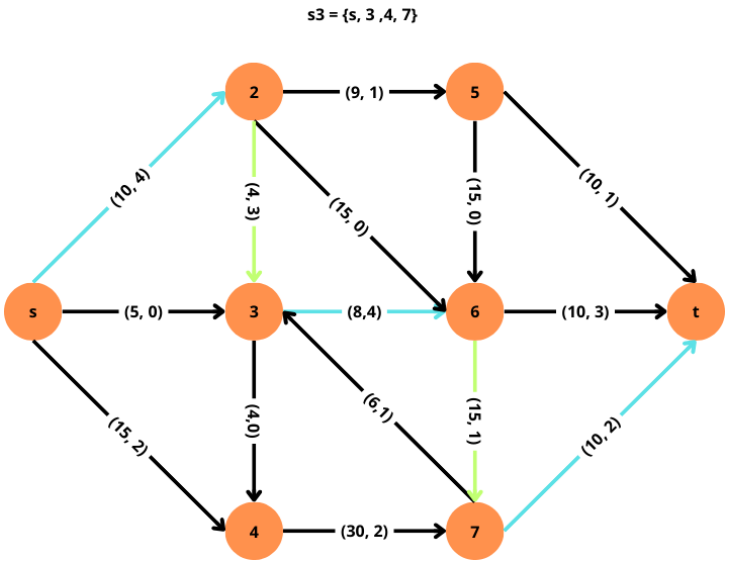


En este caso, la suma de recurso que sale del corte (celestes) - la suma de recurso que entra al corte (verde) es :

$$F = \sum_{e \in S\overline{S}} x(e) - \sum_{e \in \overline{S}S} x(e)$$

$$F = 7 - 1$$

$$F = 6$$



En este caso, la suma de recurso que sale del corte (celestes) - la suma de recurso que entra al corte (verde) es :

$$F = \sum_{e \in S\overline{S}} x(e) - \sum_{e \in \overline{S}S} x(e)$$

$$F = 10 - 4$$

$$F = 6$$

2.1.1 Capacidad de un Corte

Dada una red $N = (V, X)$ con función de capacidad c, la capacidad de un corte S se define como:

$$c(S) = \sum_{e \in S \setminus \overline{S}} c(e).$$

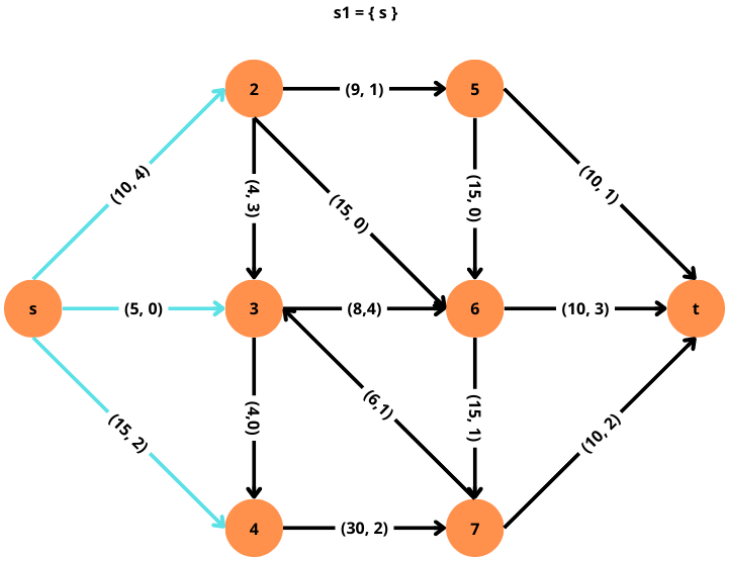
Es decir, la capacidad de un Corte es la suma de las capacidades de los arcos **que salen del corte.**

2.1.2 Problema de Corte Mínimo

El problema de corte mínimo consiste en, dada una red $N = (V, X)$ con función de capacidades en los arcos c, determinar un corte de capacidad mínima. Es decir, encontrar S, corte de N, tal que:

$$c(S) = \min\{c(\overline{S}) \mid \overline{S} \text{ es un corte de } N\}.$$

Miremos los ejemplos de las redes anteriores :

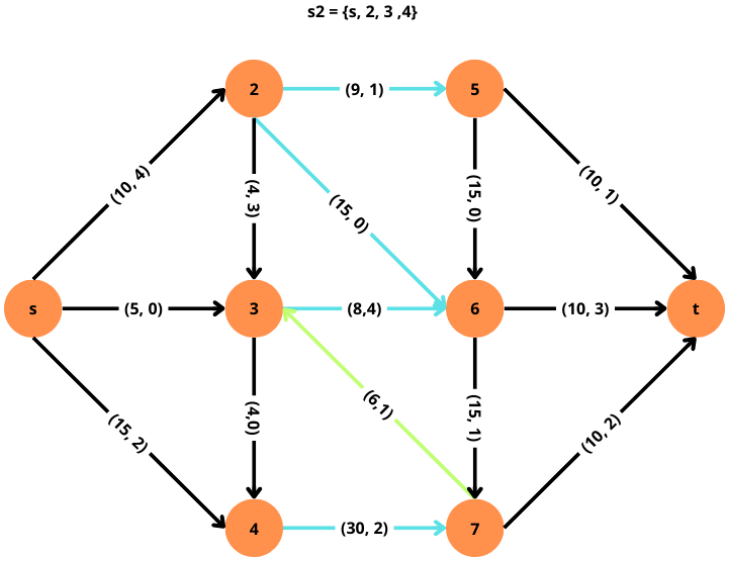


En este caso, la capacidad del corte es :

$$c(s1) = \sum_{e \in S \setminus \overline{S}} c(e).$$

$$c(s1) = 10 + 5 + 15$$

$$c(s1) = 30$$

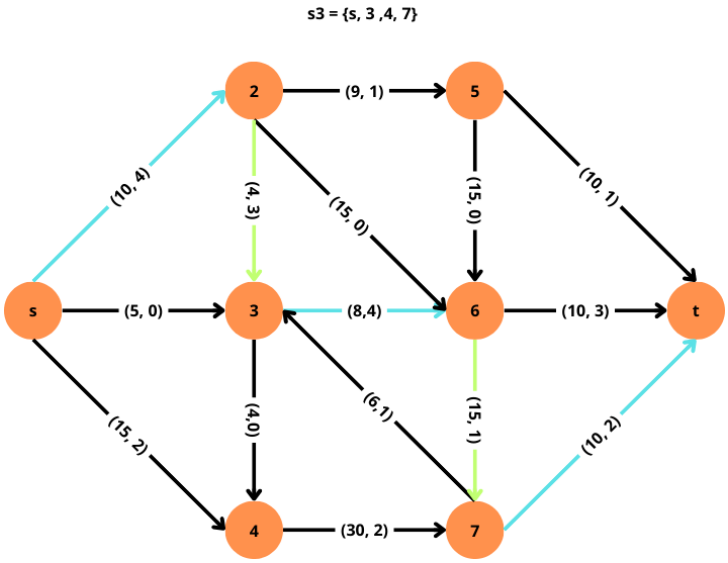


En este caso, la capacidad del corte es :

$$c(s2) = \sum_{e \in S \setminus \overline{S}} c(e).$$

$$c(s2) = 9 + 15 + 8 + 30$$

$$c(s2) = 62$$



En este caso, la capacidad del corte es :

$$c(s3) = \sum_{e \in S \setminus \overline{S}} c(e).$$

$$c(s3) = 10 + 8 + 10$$

$$c(s3) = 28$$

Habiendo visto esto, el **el problema de corte mínimo** es un problema de optimización combinatoria.

2.2 Problema DUAL (relación entre Flujo Máximo y Corte Mínimo)

Dada una red $N = (V, X)$ con función de capacidad c, una función de flujo con valor F y un corte S, se cumple que:

$$F \leq c(S).$$

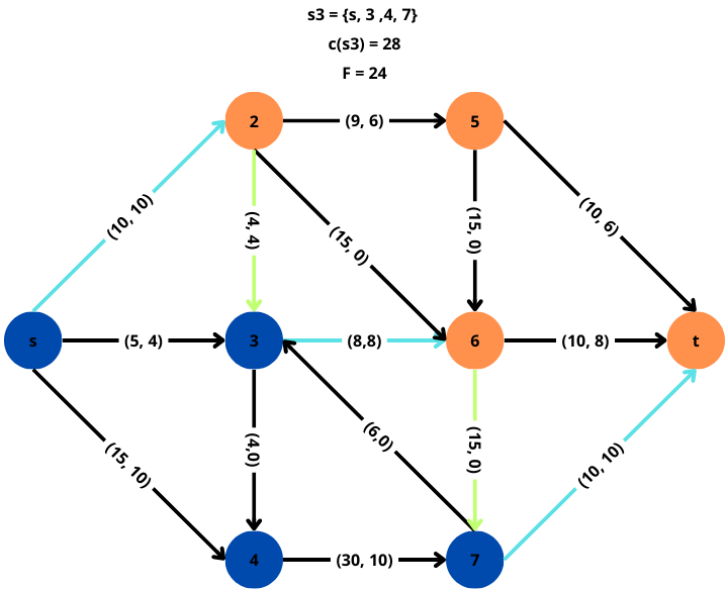
Si F es un flujo factible cualquiera y S es un corte cualquiera, siempre vale que el valor del flujo es menor o igual que la capacidad del corte.

A partir de esto, si se cumple la igualdad, es decir $F = c(S)$ donde F es el valor de un flujo f y S un corte en una red N :

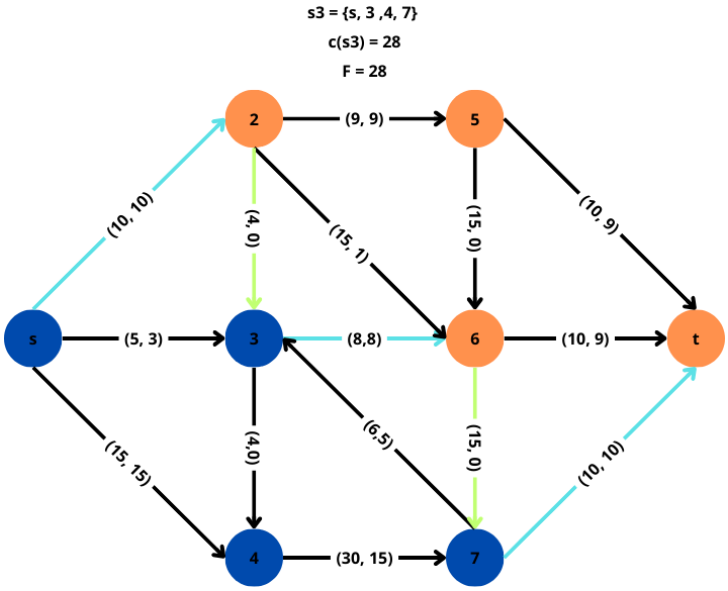
- f **define un flujo máximo.**
- S **es un corte de capacidad mínima.**

Esto se conoce como **Certificado de Optimalidad**, es un corolario del Lema anterior. Siempre voy a poder encontrar un flujo y un corte en donde valga la igualdad. Dijimos que se trata de un problema DUAL porque son dos problemas : uno de maximización (el del flujo) y uno de minimización (el de la capacidad del corte) y el óptimo de ambos es el mismo valor.

Veamos un ejemplo :



Para este ejemplo, vemos que el Corte tiene capacidad 28 y el valor de flujo (6 + 8 + 10) es 24. Es un valor de flujo factible, ya que cumple las propiedades vistas en Introducción 1



En este caso, el valor de flujo es factible y cumple que $F = c(S)$ lo cual, por el **certificado de optimalidad** podemos afirmar que el corte S es un corte de capacidad mínima y que el valor de flujo es de valor máximo.

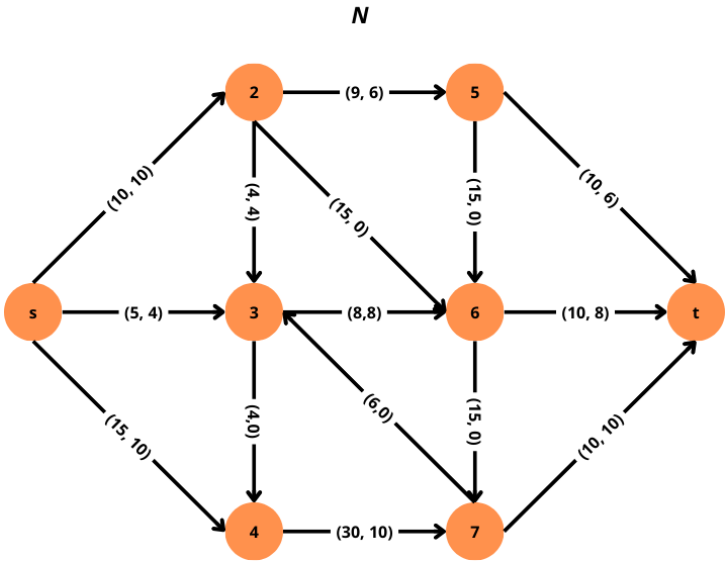
2.3 Red Residual

Dada una red $N = (V, X)$ con función de capacidad c y un flujo factible x:

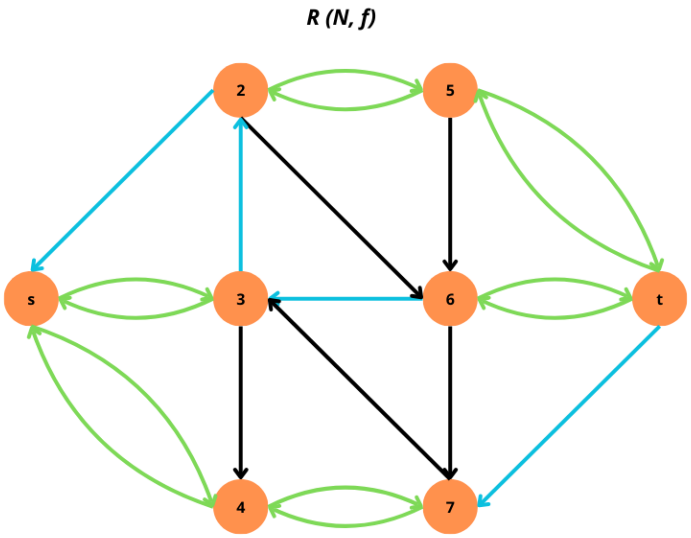
- Definimos la red residual, $R(N, x) = (V, X_R)$, donde para todo $(v \to w) \in X$:
  - $(v \to w) \in X_R$ si $x((v \to w)) < c((v \to w))$,
  - $(w \to v) \in X_R$ si $x((v \to w)) > 0$.

Definimos una red que tiene **los mismos vértices que la original, pero tiene diferentes arcos**. Estos último son de **dos tipos**.
Si el arco **no está saturado** (la cantidad de flujo que viaja por ese arco es menor estricta que su capacidad), estará en la red residual.
Si hay flujo viajando por un arco, es decir **la cantidad de flujo que atraviesa cierto arco es positiva**, entonces, en la red residual ponemos el arco **en la dirección opuesta**.
Definimos entonces la **red residual de una red con respecto a un flujo**.

La idea de poner los arcos en el sentido inverso es que disminuye el flujo que pasa por ahí.
Veamos : Sea la Red N :



Su red residual será :

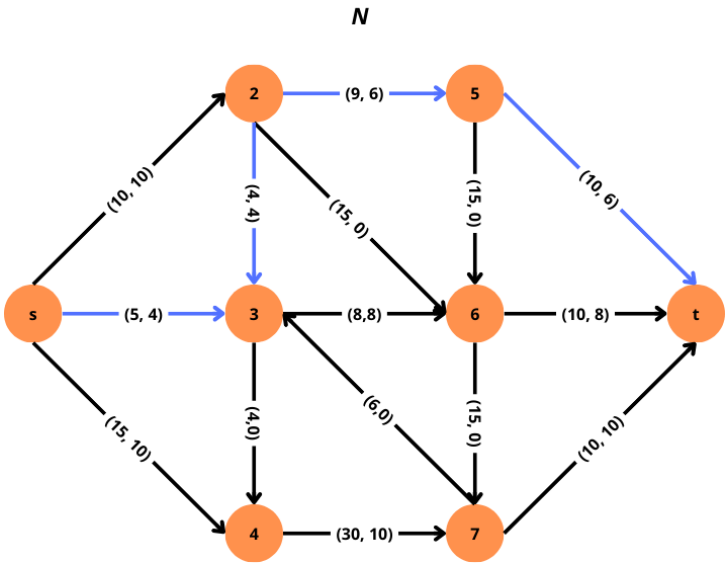


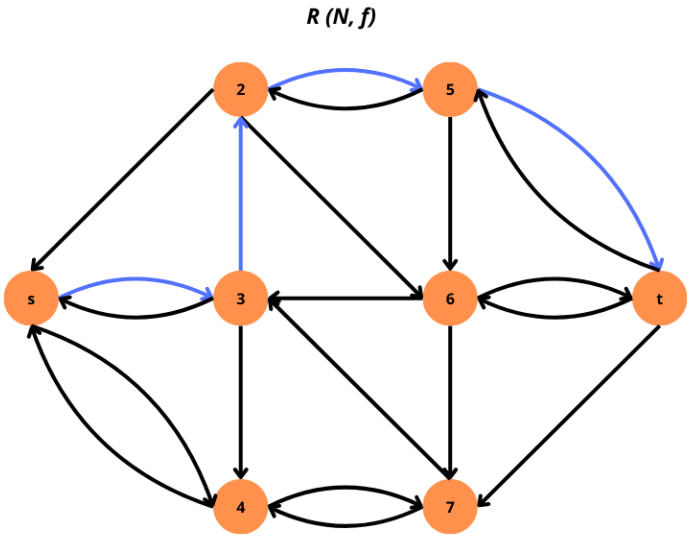
Veamos los distintos colores :

- **Negro** : Si el arco en la red residual es de color negro, es porque cumple la primer condición de 2.3 . Por ello, se mantiene tal cuál está en N. Observar que son los arcos que no transportan recurso alguno.
- **Verde** : Si el arco **no está saturado y transporta recurso en la red N** entonces en la red residual agregamos el arco en el sentido contrario y lo notamos con el color verde. Observar que cumple los dos enunciados
- **Celeste** : Si el arco **está saturado en la red N** entonces no cumple con la primer propiedad, pero sí cumple que transporta recurso. Por ello, le invertimos el sentido y lo marcamos en celeste.

3 Camino de Aumento

Cualquier camino orientado en la red residual de s a t en $R(N, x)$. Un camino sobre el cual **puedo aumentar el flujo que está circulando**. Tenemos la red, un flujo factible, construimos la red residual y buscamos un Camino de aumento en ella :





Algorithm 1 CaminoAumento(N, f, R)

Entrada: $N = (V, X)$, flujo f, red residual $R(N, x) = (V, X_R)$
Salida: P camino de aumento o S corte (que será mínimo)

$S \leftarrow \{s\}$
while $t \notin S$ y $\exists (v \to w) \in X_R$ tal que $v \in S$ y $w \notin S$ **do**
    ant$[w] \leftarrow v$
    $S \leftarrow S \cup \{w\}$
end while
if $t \notin S$ **then**
    return S corte de N
else
    Reconstruir P entre s y t usando ant a partir de t
    return P camino de aumento
end if

Este algoritmo va a iterar sobre los arcos que tienen como cola a vértices que pertenecen a S y como cabeza a vértices de $\overline{S}$. La idea es ir guardando **padres** y **construyendo S**. Puede suceder que en la búsqueda se encuentre a t (sumidero), en dicho caso, queremos **reconstruir un camino de s (fuente) a t** y lo logramos con padres. Caso contrario, no logramos dar con el sumidero, pero nos queda definido S **un corte de N**.

3.1 Delta del Camino de Aumento

Dada una red $N = (V, X)$ con función de capacidad c, un flujo factible x y un camino de aumento P en $R(N, x)$:

- Para cada arco $(v \to w)$ de P, definimos:

$$\Delta((v \to w)) = \begin{cases} c((v \to w)) - x((v \to w)) & \text{si } (v \to w) \in X, \\ x((w \to v)) & \text{si } (w \to v) \in X. \end{cases}$$

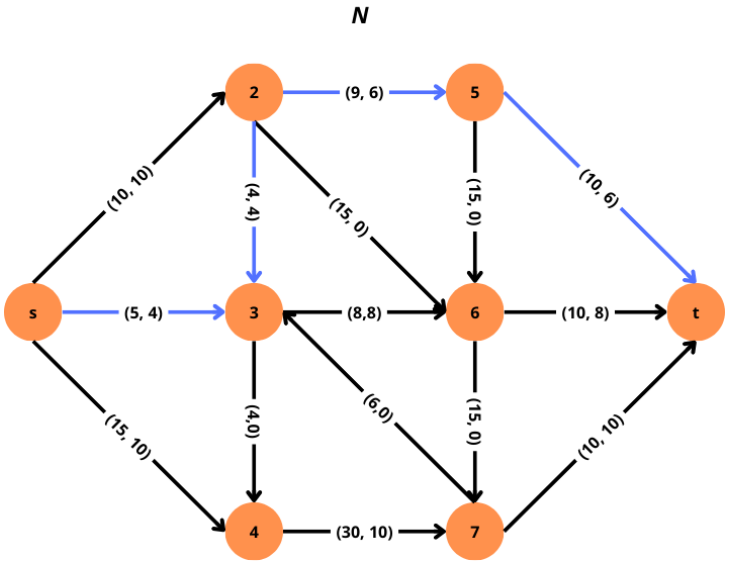
-

$$\Delta(P) = \min_{e \in P} \{\Delta(e)\}.$$

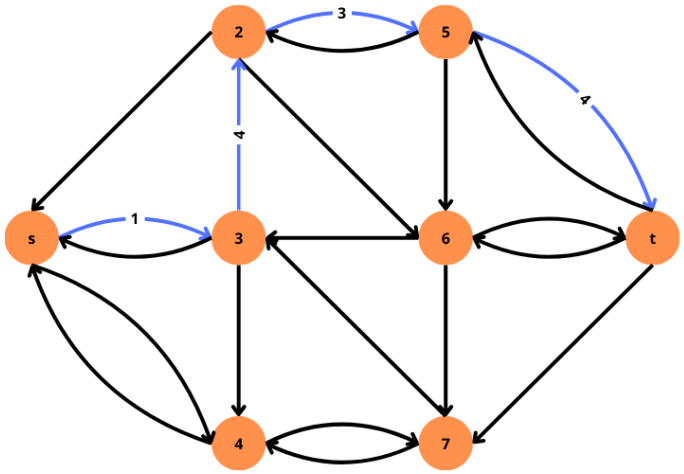
La idea es definir un **delta** que nos diga **en cuánto puedo aumentar el flujo sobre ese camino de aumento**. Para lograrlo, **para cada arco del camino de aumento** :

- Si el arco es $(v \to w)$ y en la red original es $(v \to w)$, en cuánto lo voy a poder aumentar al flujo que circula por ese arco ? A lo sumo en la diferencia entre la capacidad y el flujo. (Más no pues dejaría de ser factible)
- Si el arco fue invertido, es decir, en la red original estaba $(w \to v)$, en cuánto voy a poder disminuirlo (esto es porque lo estamos usando al revés en el camino de aumento) y entonces, lo máximo que lo podemos disminuir es el flujo que está circundo a través de este arco.

Por qué necesito un Delta del camino ? Porque tenemos que asegurar que se cumpla la ley de conservación de flujo.
Ejemplo con lo visto anteriormente:



R (N, f) *delta(P) = 1*



Hacemos los calculos de los deltas sobre cada arco:

- $(s \to 3)$ es 1, porque es la diferencia entre la capacidad y el flujo que circula en este momento.
- $(3 \to 2)$ originalmente estaba al revés (segunda guarda), por ello, va a ser igual al flujo que está pasando, 4.
- $(2 \to 5)$ 3, capacidad menos flujo
- $(5 \to t)$ 4, capacidad menos flujo

Habiendo calculado los deltas de las arista concluimos que :

$$\Delta(P) = \min_{e \in P}\{\Delta(e)\}$$

$$\Delta(P) = 1$$

De qué nos sirve esto ? En la red N vamos a :

- $(s \to 3)$ sumar 1 unidad de
- $(2 \to 3)$ restar 1 unidad
- $(2 \to 5)$ sumar 1 unidad
- $(5 \to t)$ sumar 1 unidad

Proposición: Sea x un flujo definido sobre una red N con valor F y sea P un camino de aumento en $R(N,x)$. Entonces el flujo $\bar{x}$, definido por:

$$\bar{x}((v \to w)) = \begin{cases} x((v \to w)) & \text{si } (v \to w) \notin P, \\ x((v \to w)) + \Delta(P) & \text{si } (v \to w) \in P, \\ x((v \to w)) - \Delta(P) & \text{si } (w \to v) \in P, \end{cases}$$

es un flujo factible sobre N con valor $\bar{F} = F + \Delta(P)$.

Es decir, si yo tenía un flujo factible definido sobre una red y un camino de aumento en la red residual de esa red respecto de cierto flujo :

- si para todo arco que no está en el camino de aumento lo dejo igual.
- Si es un arco que en el camino de aumento estaba utilizado en el mismo sentido que el original, le **sumamos** $\Delta(P)$
- Si es un arco que en el camino de aumento estaba utilizado al revés, le **restamos** $\Delta(P)$

Como conclusión, esta proposición nos asegura que el flujo$\bar{x}$ que encuentro con esta cuenta, si f era factible, $\bar{x}$ tambien es factible y ademas tiene **mayor valor**. **Cuánto mayor?** El valor que tenía F más el $\Delta(P)$.

Teorema: Sea f un flujo definido sobre una red N. Entonces f es un flujo máximo si y solo si no existe un camino de aumento en $R(N,x)$.

$$f \text{ es un flujo máximo} \iff \text{no existe camino de aumento en } R(N,x).$$

Teorema: Dada una red N, el valor del flujo máximo es igual a la capacidad del corte mínimo.

$$\text{Valor del flujo máximo} = \text{Capacidad del corte mínimo.}$$

4 Algortimo de Ford Fulkerson

Algorithm 2 FordFulkerson(N)

Entrada: $N = (V, X)$ con función de capacidad c.
Salida: x, flujo máximo.

```
1: Definir un flujo inicial en N (por ejemplo, x(e) ← 0 para todo e ∈ X).
2: while exista P, un camino de aumento en R(N,x) do
3:     for cada arco (v → w) de P do
4:         if (v → w) ∈ X then
5:             x((v → w)) ← x((v → w)) + Δ(P)
6:         else if (w → v) ∈ X then
7:             x((w → v)) ← x((w → v)) − Δ(P)
8:         end if
9:     end for
10: end while
11: return f flujo máximo.
```

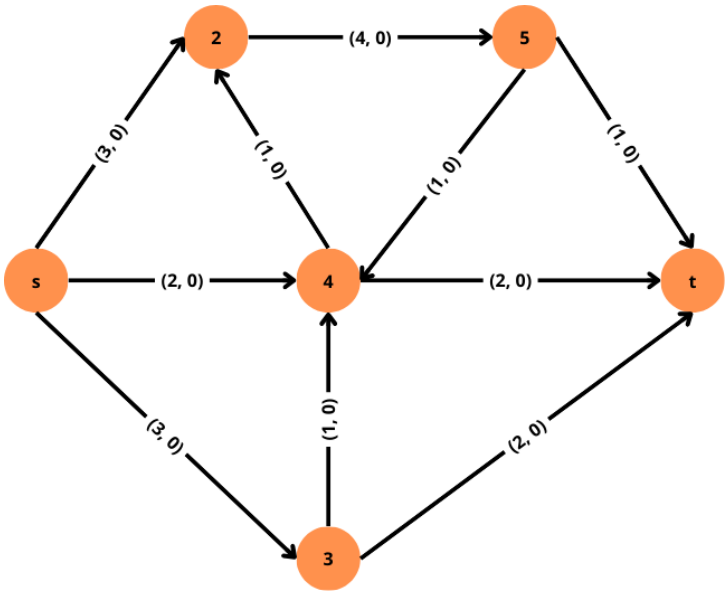
Comenzamos con un flujo factible, si no tengo capacidades minimas podría ser todo cero (a todos los arcos les pongo flujo 0, es factible pues cumple que sea mayor igual que 0 y menor o igual que la capacidad y también la ley de conservación). Mientras exista camino de aumento en la red residual, para cada arco del camino de aumento:

- **Si el arco en N está en el mismo sentido que en el camino de aumento, le sumo** $\Delta(P)$.
- **Si el arco en N está en sentido contrario, le resto** $\Delta(P)$

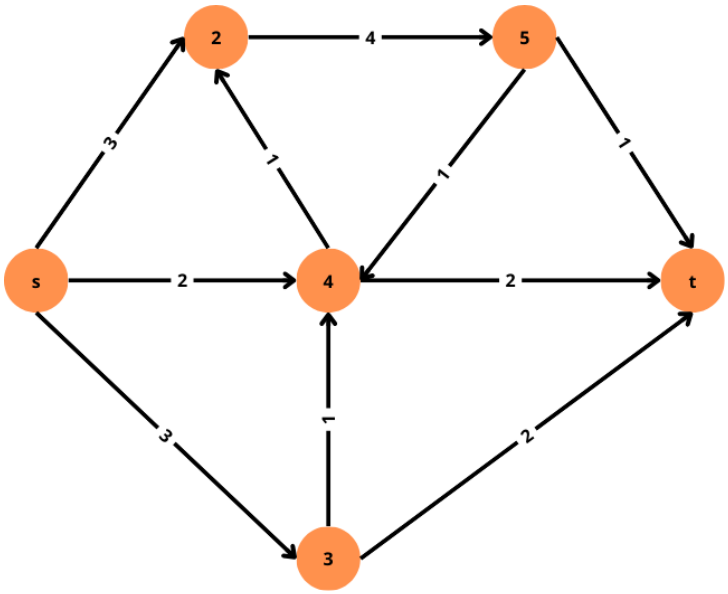
Cuando sale del ciclo, es porque **no hay camino de aumento**, entonces F define un flujo máximo (por lo visto en el teorema anterior). En resumen, busco camino de aumento, modifico el flujo para aumentar el valor y así sucesivamente hasta que no exista camino de aumento.

Veamos esquemáticamente cómo funciona :

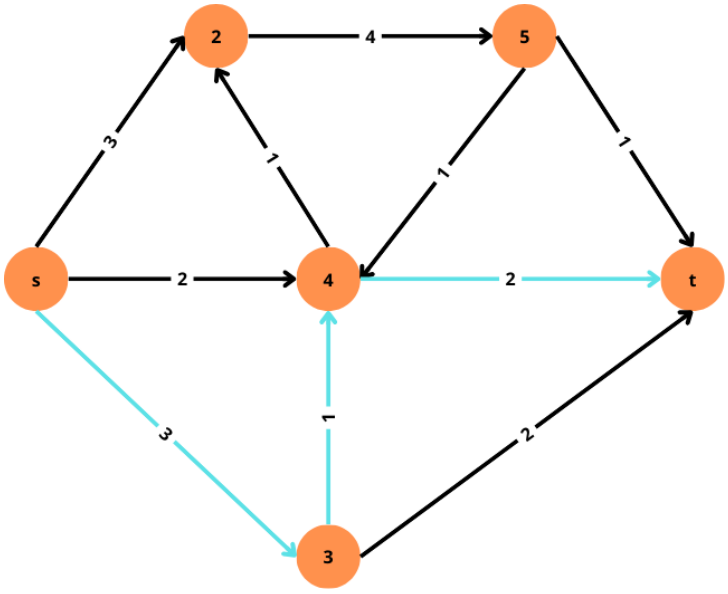
a) Arrancamos con una red, asignamos el flujo de cada arista con valor cero (flujo factible)



b) Armamos la red residual, que como todos los arcos tienen flujo 0, no va a haber arcos de sentido opuesto



c) Buscamos algún camino de aumento en la red residual



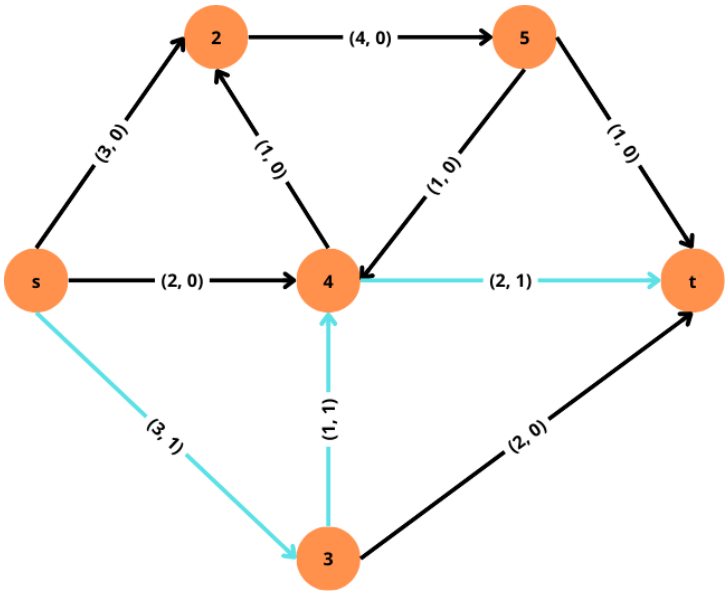
d) Para ese camino de aumento le calculamos el $\Delta(P)$

$$\Delta(P) = \min_{e \in P}\{\Delta(e)\}$$

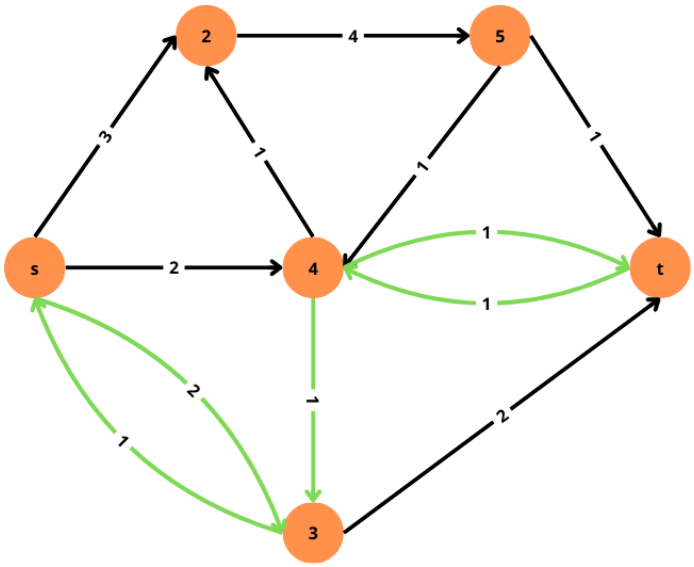
$$\Delta(P) = \min\{3, 1, 2\}$$

$$\Delta(P) = 1$$

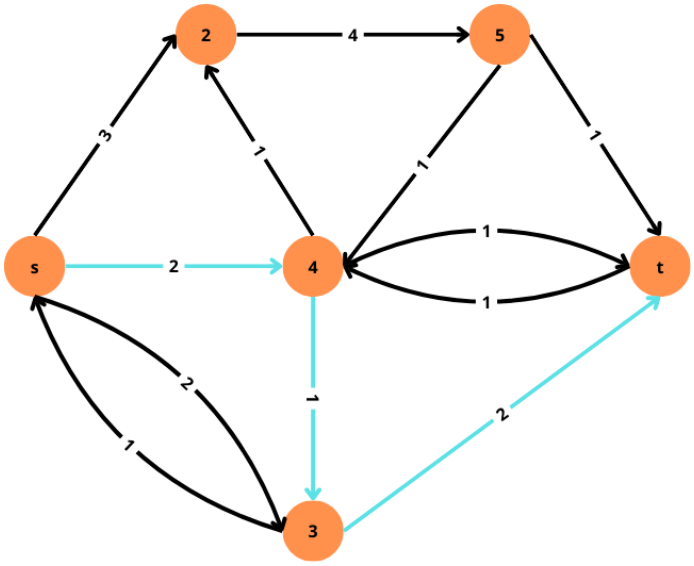
e) Con este $\Delta(P)$ calculado, procedemos a aumentar/restar según corresponda en la red original (en este caso sumamos a todos 1) a los arcos que forman el camino de aumento.



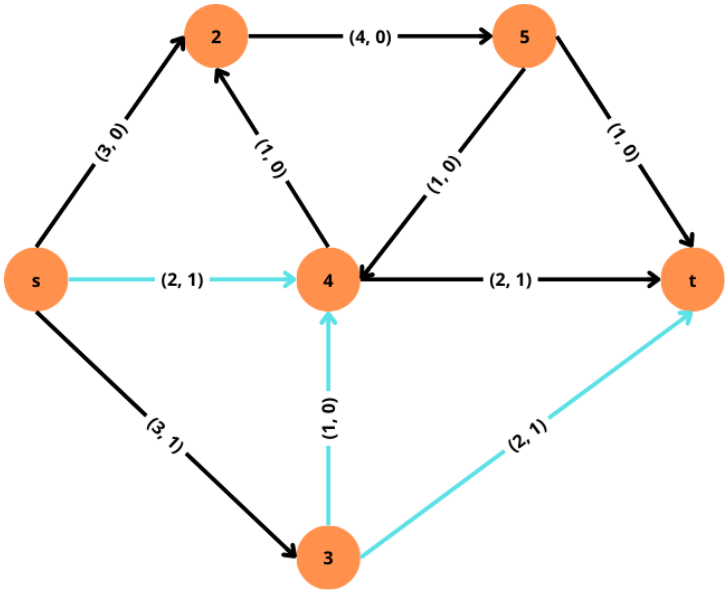
f) Ahora tenemos una **nueva red residual** considerando el nuevo flujo factible (recordar que la red residual es respecto a una red y su flujo factible). Miremos que $(s \to 3)$ como tiene un valor positivo y no está saturado, en la red residual agregamos $(3 \to s)$. Para el caso de $(3 \to 4)$ como está saturado el arco, lo invertimos. Y $(4 \to t)$ lo mismo que el primero, le agregamos el arco que vuelve $(t \to 4)$



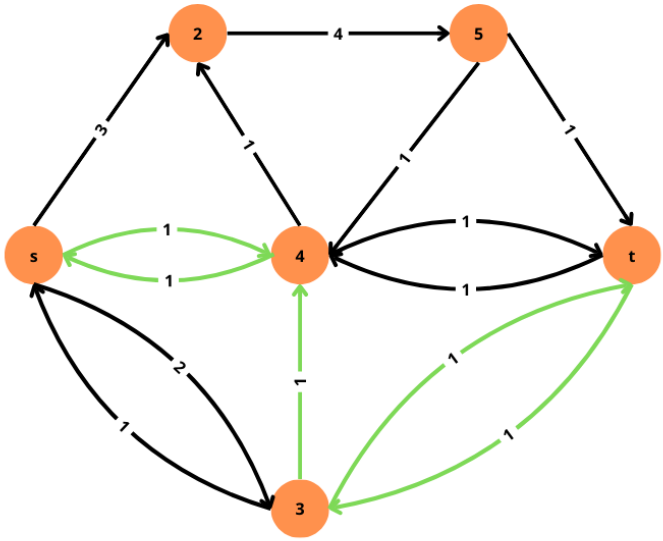
g) Buscamos otro camino de aumento en esta nueva red residual y le tomamos $\Delta(P)$ que es igual a 1.



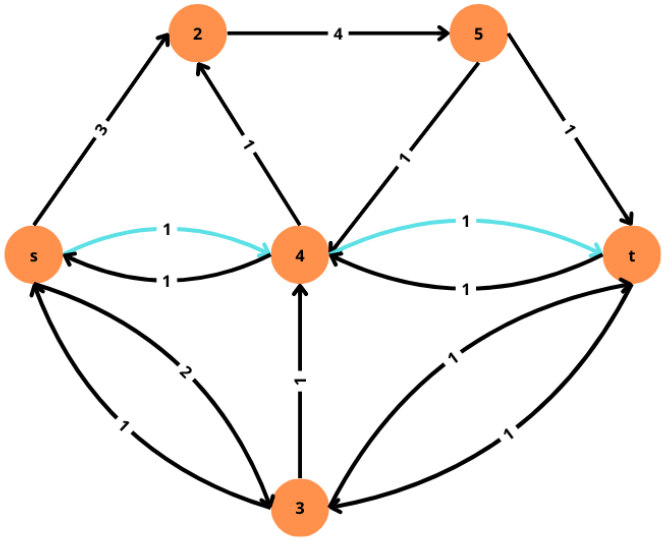
h) Modificamos la red para que en $(s \to 4)$ aumente en 1 unidad el flujo, le restamos $(3 \to 4)$ una unidad y finalmente, le sumamos una unidad a $(3 \to t)$



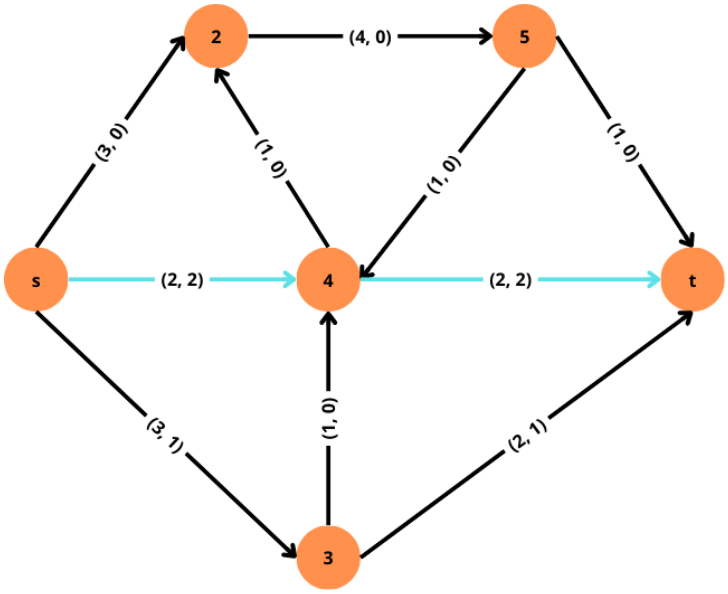
i) La red residual resultante es



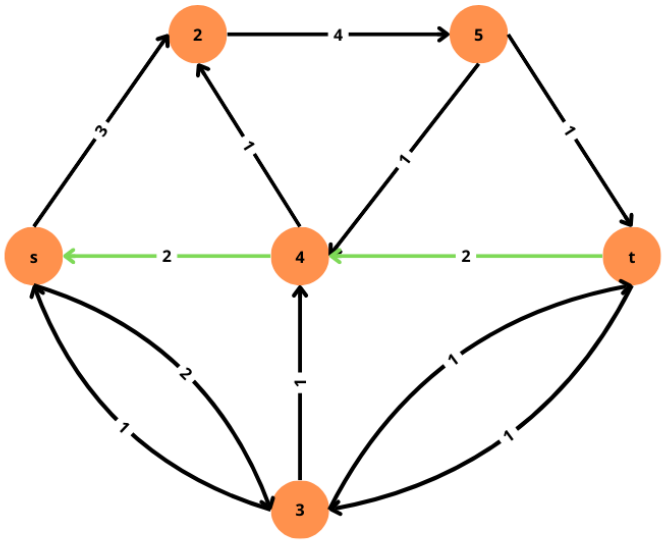
j) Elegimos un nuevo camino de aumento en la red residual



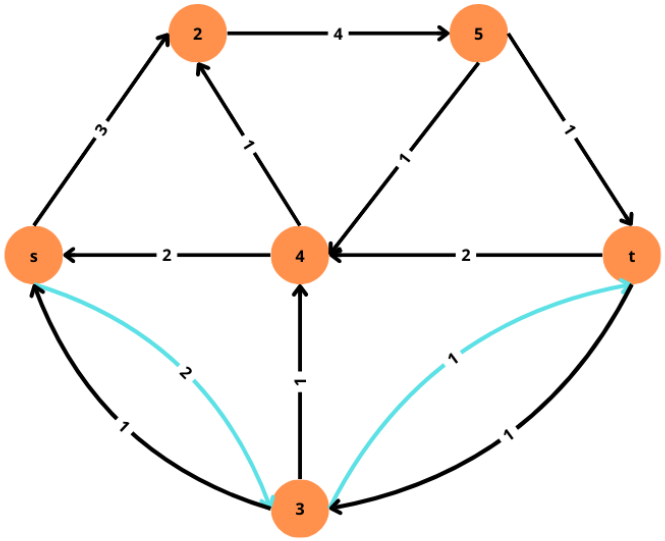
k) Hacemos las modificaciones en la red original con el $\Delta(P)$ igual a 1



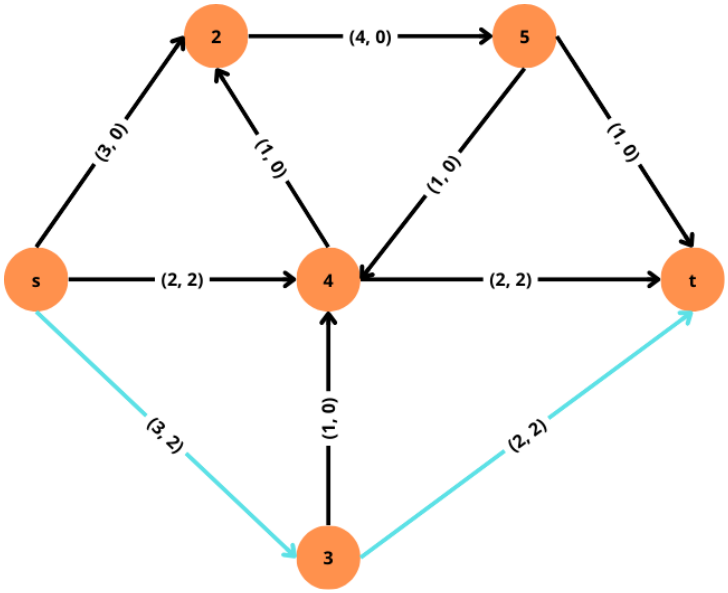
l) Como se saturó $(s \to 4)$ y $(4 \to t)$ en la red, nuestra red residual será



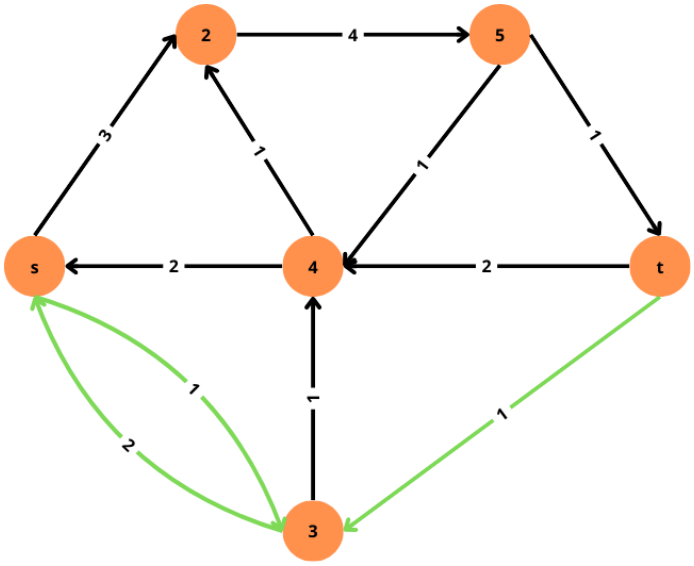
m) Elegimos un nuevo camino de aumento en la red residual



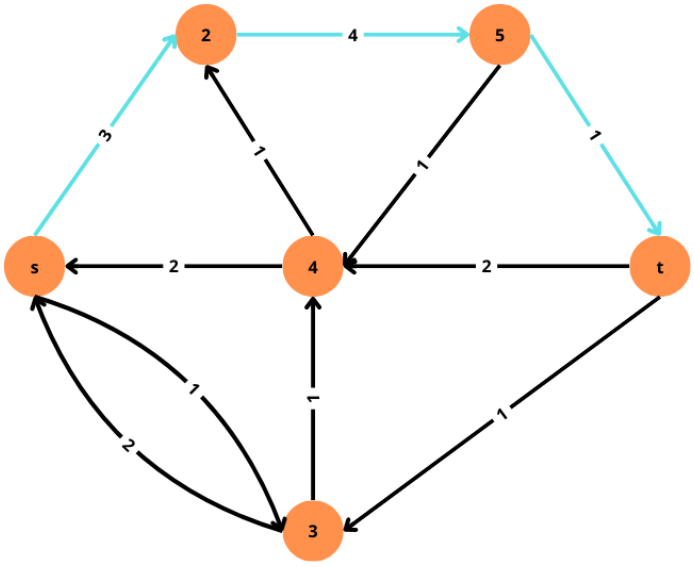
n) Hacemos las modificaciones en la red original con el $\Delta(P)$ igual a 1



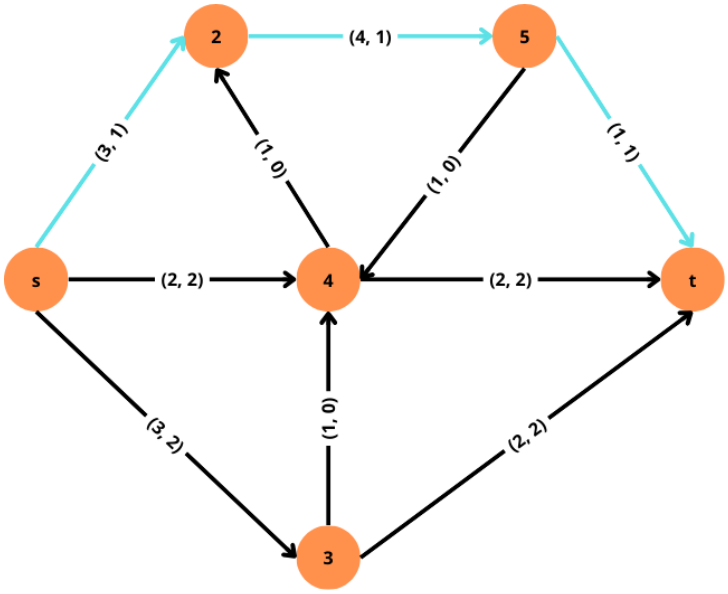
o) La red residual resultante es



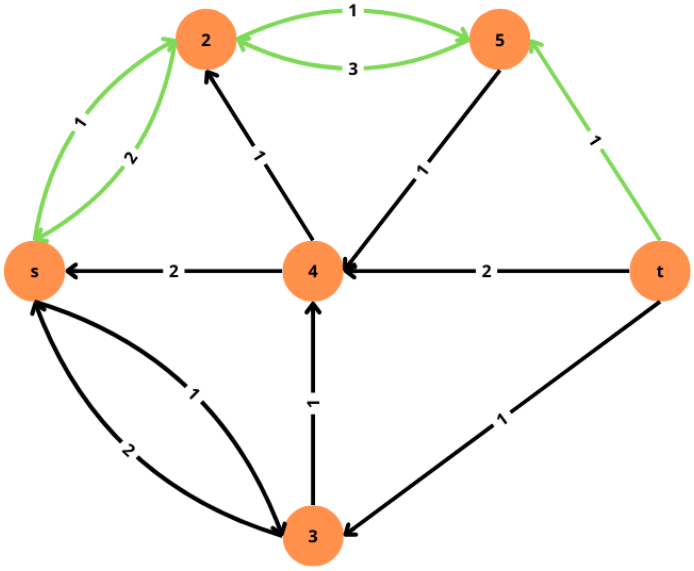
p) Elegimos un nuevo camino de aumento en la red residual



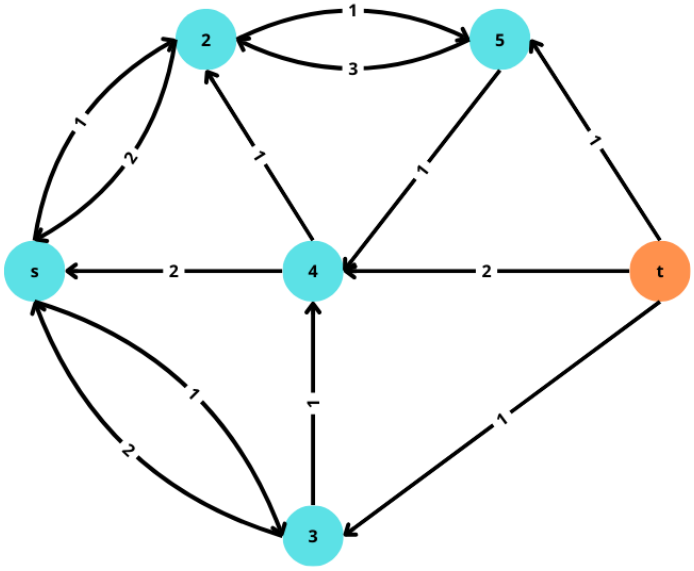
q) Hacemos las modificaciones en la red original con el $\Delta(P)$ igual a 1



r) La red residual resultante es



s) Y ya vemos que **NO HAY MANERA DE LLEGAR A T**(no tiene arcos de entrada). Ya no tenemos caminos de aumento en la red residual y por tanto, el flujo es **máximo**. El algortimo devuelve el conjunto S de camino de aumento, que es un CORTE y es de capacidad MÍNIMA. Si miramos los arcos que salen del corte y sumamos sus capacidades = 5 que es el valor del flujo que está pasando y es máximo.



4.1 Proposiciones

Proposición: Si las capacidades de los arcos de la red son enteras, el problema de flujo máximo tiene un flujo máximo entero.

Proposición: Si los valores del flujo inicial y las capacidades de los arcos de la red son enteros, F&F realiza a lo sumo F iteraciones, siendo F el valor del flujo máximo.

En esta red el flujo máximo es $F = 2M$. Si los caminos de aumento P de todas las iteraciones incluyen el arco $(v_2 \to v_3)$, $\Delta(P) = 1$ en toda iteración, debiendo hacer $F = 2M$ iteraciones.

Proposición: Si los valores del flujo inicial y las capacidades de los arcos de la red son enteros, F&F es $O(nmU)$, donde U es una cota superior para el valor de las capacidades.

Si no se especifica el orden en el que se eligen los arcos y vértices a marcar en el algoritmo de camino de aumento, el número de iteraciones de F&F, aún con capacidades enteras, puede ser no polinomial respecto del tamaño del problema.

Si las capacidades o el flujo inicial son números irracionales, F&F puede no parar (realizar un número infinito de pasos).

4.2 Análisis de Complejidad

Para analizar la complejidad del esquema de Ford-Fulkerson, es necesario acotar la cantidad de iteraciones que realiza el ciclo exterior. Para ello, nos valemos del siguiente teorema:

Teorema: Si las capacidades de una red $N = (V, E, u)$ cumplen $u(e) \in Z$, el flujo máximo de N tiene también valores enteros, y el esquema de Ford-Fulkerson puede encontrar siempre un camino de aumento P con $\Delta(P) \in Z$.

A partir de este teorema, se puede ver que, cuando las capacidades son enteras, el esquema de Ford-Fulkerson realiza como máximo F iteraciones: en cada paso, se encuentra un camino de aumento P que cumple $\Delta(P) \geq 0$ y $\Delta(P) \in Z$, lo que implica que $\Delta(P) \geq 1$. Así, en a lo sumo F pasos, se llega a un flujo de valor F.

Esto implica que la complejidad total del algoritmo es $O(|V| \cdot F)$, porque en cada iteración se realiza una operación constante para cada vértice del camino P (que puede contener a todos los vértices).

Para expresar la complejidad sin conocer el flujo máximo, se pueden encontrar cotas superiores para el mismo, como la suma de las capacidades $\sum_{e \in E} u(e)$ o el máximo $\max\{u(e) \mid e \in E\}$.

4.3 Código C++

```cpp
#include <iostream>
#include <limits.h>
#include <queue>
#include <string.h>
using namespace std;

#define V 6

bool bfs(int rGraph[V][V], int s, int t, int parent[]){
    bool visited[V];
    memset(visited, 0, sizeof(visited));

    queue<int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (int v = 0; v < V; v++) {
            if (visited[v] == false && rGraph[u][v] > 0) {
                if (v == t) {
                    parent[v] = u;
                    return true;
                }
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }

    return false;
}

int fordFulkerson(int graph[V][V], int s, int t){
    int u, v;

    int rGraph[V][V];
    for (u = 0; u < V; u++)
        for (v = 0; v < V; v++)
            rGraph[u][v] = graph[u][v];

    int parent[V];

    int max_flow = 0;

    while (bfs(rGraph, s, t, parent)) {
        int path_flow = INT_MAX;
        for (v = t; v != s; v = parent[v]) {
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }

        for (v = t; v != s; v = parent[v]) {
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }

        max_flow += path_flow;
    }

    return max_flow;
}

int main(){
    int graph[V][V] = { { 0, 16, 13, 0, 0, 0 },
    { 0, 0, 10, 12, 0, 0 },{ 0, 4, 0, 0, 14, 0 },
    { 0, 0, 9, 0, 0, 20 },{ 0, 0, 0, 7, 0, 4 },
    { 0, 0, 0, 0, 0, 0 } };
```

```
74
75      cout << "The maximum possible flow is "
76           << fordFulkerson(graph, 0, 5);
77
78      return 0;
79  }
```

5 Algoritmo de Edmonds y Karp

La diferencia principal entre Edmonds-Karp y Ford-Fulkerson es que Edmonds-Karp utiliza la búsqueda en anchura (*BFS*) para encontrar caminos de aumento, mientras que Ford-Fulkerson emplea la búsqueda en profundidad (*DFS*).

Esto significa que el tiempo de ejecución de Edmonds-Karp es más predecible que el de Ford-Fulkerson, ya que Edmonds-Karp no está afectado por el valor del flujo máximo.

5.1 Análisis de Complejidad

Dado un grafo con V vértices y E aristas, la complejidad temporal de Edmonds-Karp es:

$$O(V \cdot E^2).$$

Esto implica que Edmonds-Karp depende del número de vértices y aristas del grafo, pero no del valor del flujo máximo, como ocurre con Ford-Fulkerson.

a) Edmonds-Karp ejecuta *BFS*, cuya complejidad es:

$$O(E + V).$$

b) En el peor caso, asumimos un grafo denso, donde $|E| \in \Omega(|V^2|)$, y la complejidad de *BFS* se aproxima a:

$$O(E^2).$$

c) Durante la ejecución del algoritmo, se pueden encontrar hasta $V \cdot E$ caminos de aumento en el peor caso. El motivo es que cada arista en la red residual puede ser saturada, a lo largo del algoritmo, a lo sumo $\frac{|V|}{2}$ veces. Asimismo, cada vez que aumentamos el flujo, saturamos al menos una arista. Luego, el número de veces que podemos aumentar el flujo es a lo sumo $\frac{2|E||V|}{2}$. El $2|E|$ es porque la red residual tiene a lo sumo $2|E|$ aristas. Cada vez usamos $O(|E| + |V|)$ operaciones para buscar un camino de aumento, y como cada vértice tiene al menos una arista que entra y otra que sale, tenemos $|E| \geq 2|V|$, y esto es $O(|E|)$.

d) Dado que *BFS* se ejecuta una vez por cada camino de aumento, la complejidad total de Edmonds-Karp es:

$$O(V \cdot E \cdot E) = O(V \cdot E^2).$$

5.2 Código C++

```
1  #include <iostream>
2  #include <climits>
3  #include <cstring>
4  #include <queue>
5  #include <vector>
6  using namespace std;
7
8  const int INF = INT_MAX;
9  const int MAX_V = 100;
10
11 int capacity[MAX_V][MAX_V];
12 int parent[MAX_V];
13
14 int bfs(int source, int sink, vector<vector<int>> &graph) {
15     fill(parent, parent + MAX_V, -1);
16     parent[source] = source;
17     queue<pair<int, int>> q;
18     q.push({source, INF});
19     while (!q.empty()) {
20         int current = q.front().first;
21         int flow = q.front().second;
22         q.pop();
```

```
        for (int next : graph[current]) {
            if (parent[next] == -1 && capacity[current][next] > 0) {
                parent[next] = current;
                int new_flow = min(flow, capacity[current][next]);
                if (next == sink)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }
    return 0;
}

int edmondsKarp(int source, int sink, vector<vector<int>> &graph) {
    int max_flow = 0;
    int new_flow;
    while ((new_flow = bfs(source, sink, graph)) > 0) {
        max_flow += new_flow;
        int current = sink;
        while (current != source) {
            int prev = parent[current];
            capacity[prev][current] -= new_flow;
            capacity[current][prev] += new_flow;
            current = prev;
        }
    }
    return max_flow;
}

int main() {
    int source = 0;
    int sink = 5;
    vector<vector<int>> graph(MAX_V);
    capacity[0][1] = 10;
    capacity[0][2] = 5;
    capacity[1][2] = 15;
    capacity[1][3] = 5;
    capacity[2][3] = 10;
    capacity[2][4] = 10;
    capacity[3][4] = 5;
    capacity[3][5] = 10;
    capacity[4][5] = 10;
    for (int i = 0; i < MAX_V; ++i) {
        for (int j = 0; j < MAX_V; ++j) {
            if (capacity[i][j] > 0) {
                graph[i].push_back(j);
            }
        }
    }
    int max_flow = edmondsKarp(source, sink, graph);
    cout << "Maximum Flow: " << max_flow << endl;
    return 0;
}
```

6 Matching Máximo en Grafos Bipartitos

Un *matching* o correspondencia entre los vértices de G, es un conjunto $M \subseteq E$ de aristas de G tal que para todo $v \in V$, v es incidente a lo sumo a una arista de M.

- El problema de *matching máximo* consiste en encontrar un matching de cardinal máximo entre todos los matchings de G.
- El problema de *matching máximo* es resoluble en tiempo polinomial para grafos en general (Edmonds, 1961–1965).
- Pero en el caso de grafos bipartitos, podemos enunciar un algoritmo más simple transformándolo en un problema de flujo máximo en una red.

Dado el grafo bipartito $G = (V_1 \cup V_2, E)$, definimos la siguiente red $N = (V', E')$:

- $V' = V_1 \cup V_2 \cup \{s, t\}$, con s y t dos vértices ficticios representando la fuente y el sumidero de la red.
- $E' = \{(i, j) : i \in V_1, j \in V_2, ij \in E\} \cup \{(s, i) : i \in V_1\} \cup \{(j, t) : j \in V_2\}$.

- $u_{ij} = 1$ para todo $ij \in E$.

El cardinal del *matching* máximo de G será igual al valor del flujo máximo en la red N.