Definición, **Lado crítico:** Diremos que un lado $x \to y$ **se vuelve crítico** durante la construcción de uno de los flujos intermedios (digamos, f_{k+1}) si para la construcción de f_{k+1} pasa una de las dos cosas siguientes:

1. Se usa el lado en forma forward, saturándolo (es decir $f_k(x \to y) < c(x \to y)$, pero luego $f_{k+1}(x \to y) = c(x \to y)$.
2. O se usa el lado en forma backward, vaciándolo (es decir, $f_k(x \to y) > 0$ pero $f_{k+1}(x \to y) = 0$).

Complejidad de Edmonds-Karp $O(n\, m^3)$ [1]

Suponemos que si el network está el lado $x \to y$, entonces no está el lado $y \to x$.

Si $f_0, f_1, f_2, \dots$, *etc* son los flujos parciales producidos al correr Edmonds-Karp, entonces queremos ver que hay una cantidad finita de ellos, y dar una cota para ese número.

Como la búsqueda y construcción de cada camino aumentante se hace con BFS, cada incremento del flujo tiene complejidad $O(m)$.

Así que si probamos que solo puede haber $O(n\, m)$ flujos aumentantes, tendremos una prueba de la complejidad de Edmonds-Karp: $O(mn).O(m) = O(nm^2)$.

En cada construcción de un camino aumentante al menos un lado se vuelve crítico. Pues la única diferencia entre Edmonds-Karp y Ford-Fulkerson es solo cómo se elige el camino.

El problema es que un lado puede volverse crítico **muchas veces**.
Se satura, se vacía un poco, vuelve a saturarse, se vacía completamente, etc.. Veamos cuántas veces puede pasar esto. (Acotar cuántas veces puede un lado volverse crítico)
Supongamos que un $x \to y$ se vuelve crítico en el paso k y luego en el paso r con $r > k$.
Tenemos que analizar dos casos: se vuelve crítico en el paso k porque se saturó, o porque se vació.

Caso donde el lado se saturó en el paso *k*:
Como se saturó en el paso k, entonces:

- Para construir f_{k+1} se debe usar un f_k camino aumentante de la forma $s \dots xy \dots y$.
- **Como estamos usando Edmonds-Karp,** ese camino es de longitud mínima.
- Por lo tanto $d_k(y) = d_k(x) + 1$. (1)

Para que se vuelva a volver crítivo en el paso r, debe pasar una de dos cosas:

- Se vuelve crítico en el paso r porque se vacía
- Se vuelve crítico en el paso r porque vuelve a saturarse.

Para que ocurra el segundo caso, debe haberse vaciado ANTES aunque sea un poco, si no es imposible que vuelva a saturarse.
Entonces en cualquiera de estos casos, deducimos que existe l con $r \geq l \geq k$ tal que el flujo en $x \to y$ **disminuye** al pasar de f_l a f_{l+1} ya sea vaciándose completamente o un poco.
Esto implica que para construir f_{l+1} se usa un f_l camino aumentante de la forma
$s \dots y \leftarrow x \dots t$.
Como estamos usando Edmonds-Karp, ese camino es de longitud mínima.
Por lo tanto $d_l(x) = d_l(y) + 1$ (2)

Entonces

$$\begin{aligned} d_l(t) &= d_l(x) + b_l(x) \\ &= d_l(y) + 1 + b_l(x) \text{ por (2)} \\ &\geq d_k(y) + 1 + b_k(x) \text{ porque las distancias no disminuyen} \\ &= d_k(x) + 1 + 1 + b_k(x) \text{ por (1)} \\ &= d_k(t) + 2 \end{aligned}$$

Caso donde el lado se vació en el paso *k*:
Este análisis era si el lado $x \to y$ se volvía crítico en el paso k porque se saturaba.
Supongamos ahora que se vuelve crítico en ese paso porque se vacía.
El análisis es similar:

- Como se vacía, existe un camino (de longitud mínima) de la forma $s \dots y \leftarrow x \dots t$ que se usa para pasar de f_k a f_{k+1}
- Por lo tanto $d_k(x) = d_k(y) + 1$ (3)

Para poder volver a ser crítico en el paso r, debe o bien volverse a vaciar, o bien saturarse.
En el segundo de esos casos estamos obviamente mandando más flujo en el paso r, pero en el primero, si estamos diciendo que debe **vaciarse** de vuelta, entonces **antes** debe haberse llenado aunque sea un poco.

Así que en cualquier caso dee haber un l tal que $r \geq l \geq k$ para el cual se manda flujo a través de él.
Entonces, existe un camino(de longitud mínima) de la forma $s \dots x\ y \dots t$ que se usa para pasar de f_l a f_{l+1}.
Por lo tanto $d_l(x) + 1$. (4)

(3) y (4) son iguales a (1) y (2), solo que con x e y intercambiados.

- Por lo tanto podemos deducir $d_l(t) \geq d_k(t) + 2$ de la misma forma que lo que hicimos con el caso en que seaturaba en el paso k, simplemente intercambiando x e y en la prueba.
- Así que también en este caso concluimos que $d_r(t) \geq d_l(t) \geq d_k(t) + 2$.

Concluimos que en cualquiera de los casos, luego de que un lado se vuelve crítico, para que pueda volverse crítico otra vez, la distancia entre s y t debe aumentar en al menos 2.

Como la distancia entre s y t puede ir desde un mínimo de 1 a un máximo de $n - 1$ concluímos que **un lado puede volverse crítico un máximo de O(n/2) = O(n) veces**.

Como cada camino aumentante que se usa en Edmonds-Karp tiene al menos un lado que se vuelve crítico, entonces el total de flujos intermedios está acotado por $O(m\ n)$, pues hay m lados y cada uno se puede volver crítico a lo sumo $O(n)$ veces.
Como cada construcción de un flujo tiene complejidad $O(m)$ concluímos que la complejidad de Edmonds-Karp es $O(m) * O(m\ n) = O(n\ m^2)$.

Definición, vecino *f FF*: Dado un flujo *f* y un vértice *x*, diremos que un vértice *z* es un vecino *f FF* de *x* si pasa alguna de las siguientes condiciones:

- $x \to z \in E$ y $f(x \to z) < c(x \to z)$
- $z \to x \in E$ y $f(z \to x) > 0$

Obsrvación*:* Si *z* es un f_k *FF* vecino de *x*, entonces $d_k(z) \leq d_k(x) + 1$.
Demostración:
Si no existe f_k-camino aumentane entre *s* y *x*, entonces $d_k(x) = \infty$ y el lema es obvio.
Supongamos entonces que existen f_k -caminos aumentantes entre *s* y *x*, y tomemos de entre ellos, alguno con longitud mínima.
Llamémosle *P* a ese camino. Por lo tanto, la longitud de *P* es $d_k(x)$.
Como *z* es f_k*FF* vecino de x, si tomamos *Q* el camino que consiste en agregar *z* al final de P, tenemos que *Q* es un f_k-camino aumentante entre *s* y *z*.
La longitud de *Q* es igual a la longitud de *P* más uno, es decir, $d_k(x) + 1$.
Q es UN f_k-camino aumentante entre *s* y *z*.
Por lo tanto la MENOR longitud posible entre TODOS los f_k-caminos aumentantes entre s y x es $d_k(z)$ y la longitud de *Q* es $d_k(x) + 1$, hemos probado que $d_k(z) \leq d_k(x) + 1$.

Distancia entre *x* y *z*, $d_f(x, z)$,:
Dados vértices *x*, *z* y flujo *f* definimos la distancia entre *x* y *z*, relativa a *f* como la longitud del menor *f*-camino aumentante entre *x* y *z*, si es que existe tal camino o infinito si no existe o 0 si $x = z$, denotándola por $d_f(x, z)$,

***$d_{fk}(s, x)$*:** $d_k(x) = d_{fk}(s, x)$ donde f_k es el *k*-ésimo flujo en una corrida de Edmonds-Karp.

Lema de las distancias [4]

Dados vértices *x*, *z* y flujo *f* , entonces $d_k(x) \leq d_{k+1}(x)$.

Demostración:
Idea: Por absurdo.

> Sea $A = \{y: d_{k+1}(y) < d_k(y)\}$
> Si el lema es cierto, *A* es vacío, así que asumamos que $A \neq \varnothing$ y lleguemos a una contradicción.

Como $A \neq \varnothing$, tiene algún elemento.
Tomaremos $x \in A$ tal que:
$d_{k+1}(x) = Min\{d_{k+1}(y) : y \in A\}$
Como $x \in A$ entonces $d_{k+1}(x) < d_k(x)$ (1)
En particular, $d_{k+1}(x) < \infty$, así que existe un f_{k+1} camino aumentante entre *s* y *x*, de entre todos ellos tomamos uno de la menor longitud.

Observemos que $x \neq s$ pues $x \in A$ y $s \notin A$ (pues $d_k(s) = d_{k+1}(s) = 0$)
Así que ese camino no es el camino formado sólo por el vértice s.
Concluímos que debe existir un vértice $z \neq x$ inmediatamente anterior a *x* en ese camino.

Veamos que propiedades tienen *z*, *x* y lleguemos a la contradicción.
Primero observemos que como *z* es el vecino inmediatamente anterior a *x* en un f_{k+1} camino aumentante, entonces *x* es un f_{k+1} *FF* vecino de *z*. (*Recordar**)

Como $d_{k+1}(x)$ es la menor de las distancias d_{k+1} de elementos de *A*, deducimos que en cualquier otro elemento de que tenga d_{k+1} menor que $d_{k+1}(x)$, no puede estar en *A*. *

z forma parte de un f_{k+1} camino aumentante de menor logitud entre *s* y *x*, el fragmento de ese camino que va de *s* a *z* es un f_{k+1} camino aumentante de longitud **mínima** entre *s* y *z*. *
Como z está justo antes de *x*, concluímos que $d_{k+1}(z) = d_{k+1}(x) - 1$. *(2)
Entonces $d_{k+1}(z) = d_{k+1}(x) - 1 < d_{k+1}(x)$, así que por todo lo anterior *, $z \notin A$.
Concluímos que $d_k(z) \leq d_{k+1}(z)$ (3)

Poniendo todo junto:
$d_k(x) > d_{k+1}(x)$ (1)
$= d_{k+1}(z) + 1$ (2)
$\geq d_k(z) + 1$ (3)

Es decir, $d_k(x) > d_k(z) + 1$ □
(Por la observación trivial), esto implica que x no es f_k *FF* vecino de *z*.

Entonces, la situación es:

x no es f_k *FF* vecino de *z*, pero
x es f_{k+1} *FF* vecino de *z* (por (*Recordar**))

La única forma en la que esto puede pasar es que el f_k camino aumentante que usamos para construir f_{k+1} sea un camino aumentante que pase primero por *x* y luego por *z*,

Expliquemos un poco más esto, considerando los dos casos posibles, según sea que $x \vec{} z \in E$ o que $z \vec{} x \in E$.

***Caso* $x \vec{} z$ es lado.**
Si $x \vec{} z \in E$, entonces:

1. *x* no es f_k *FF* vecinoo de z implica que $f_k(x \vec{} z) = 0$.
2. *x es* f_{k+1} *FF* vecino de z implica que $f_k(x \vec{} z) > 0$.

1 y 2 implican que $f_{k+1}(x \vec{} z) > f_{k+1}(x \vec{} z)$
Esto solo puede pasar si al construir f_{k+1} ENVIAMOS flujo por el lado $x \vec{} z$.
Por lo tanto el camino pasó primero por *x* y luego por *z*.

Caso $z \vec{} x$ es lado.
Si $z \vec{} x \in E$, entonces:

1. *x* no es f_k *FF* vecino de z implica que $f_k(x \vec{} z) = c(z \vec{} x)$.
2. *x* es f_k *FF* vecino de z implica que $f_{k+1}(x \vec{} z) < c(z \vec{} x)$.

1 y 2 implican que $f_{k+1}(z \vec{} x) < f_k(z \vec{} x)$.
Esto solo puede pasar si al construir f_{k+1} hubo una DISMINUCIÓN de flujo en el lado $z \vec{} x$.
Esto último solo puede pasar si se usó BACKWARDs.
Para usarse backward, el camino debe haber pasado primero por x y luego por z.

Para cualquiera de los casos, significa que para pasar de f_k a f_{k+1} usamos un camino aumentante *P* de la forma *s...xz...t* (o *s...*$x \vec{} z$*...t* en el caso backward).

Como estamos usando Edmonds-Karp, ese camino es de longitud mínima.
Por lo tanto $d_k(z) = d_k(x) + 1$
Pero, por □, teníamos que $d_k(x) + 1$.
Entonces $d_k(x) > d_k(z) + 1 = d_k(x) + 1 + 1$ implica $0 > 2$
Absurdo.

La distancia en networks auxiliares sucesivos aumenta [5]

La distancia entre *s* y *t* **aumenta** entre networks auxiliares consecutivos.

Demostración:
Sea NA un network auxiliar y NA' el network auxiliar siguiente.
Sea *f* el flujo (del network original) inmediatamente anterior a NA y *f'* el inmediatamente anterior a NA'.
Es decir, NA se construye usando *f* y NA' se construye usando f'.
Sea $d(x) = d_f(s, x)$ y $d'(x) = d_{f'}(s, x)$
Sabemos que $d(t) \leq d'(t)$ por la prueba de Edmonds – Karp.
Queremos probar que vale el < ahí.

Si NA' no tiene a *t* entonces $d'(t) = \infty > d(t)$ y ya está.

Asumamos entonces que $t \in NA'$.

Entonces existe un camino dirigido $x_0 x_1 \dots x_{r-1} x_r$ entre *s* y *t* **en NA'.**

Pero no puede ser un camino **en NA:**

- Si lo fuera, al construir *f'* habríamos saturado ese camino, es decir, saturar al menos un lado.
- Pues para terminar con NA y pasar de *f* a *f'* debemos saturar **todos** los caminos de NA.
- Pero si quedó saturado, no podría ser un camino en NA'.

Como $x_0 x_1 \dots x_{r-1} x_r$ **no es** un camino en NA, entonces pasa una de dos cosas:

1. Algún vértice x_i está en NA.
2. Están todos los x_i en NA pero falta algún lado $x_i \rightarrow x_{i+1}$

Veamos estos dos casos,

Caso 1: algún vértice x_i no está en NA.

Como *t* está en NA, entonces $x_i \neq t$

Recordemos que todos los vértices $\neq t$ que estén a distancia mayor o igual que *t* no se incluyen.

Pero todos los que tengan distancia menor a *t* están pues construímos NA con BFS.

Entonces la única forma en que x_i no esté en NA es que $d(t) \leq d(x_i)$ (1)

Como probamos en Edmonds-Karp que $d \leq d'$, tenemos:

$d(x_i) \leq d'(x_i)$ (2)

Ahora bien, como $x_0 x_1 \dots x_{r-1} x_r$ es un camino en NA' y NA' es un network por niveles, concluímos que:

$d'(x_i) = i$ *para todo* i (3)

Además vimos que $x_i \neq t$, así que $i < r$ (4)

Entonces, $d(t) \leq d(x_i) \leq d'(x_i) = i < r = d'(t)$ por (1), (2), (3) y (4)

Y hemos probado lo que queríamos en este caso.

Caso 2: están todos los x_i en NA pero falta algún lado $x_i \rightarrow x_{i+1}$

Tomamos el primer *I* para el cual se cumple la condición.

Por Edmonds-Karp, sabemos que $d(x_{i+1}) \leq d'(x_{i+1})$.

Así que tenemos dos subcasos: que ese ≤ sea < o que sea =.

Subcaso A: $d(x_{i+1}) < d'(x_{i+1})$.

Sea $b(x) = b_f(x, t)$, $b'(x) = b_{f'}(x, t)$,

Por lo que vimos en Edmonds – Karp, tenemos que $b \leq b'$ y $d(t) = d(x) + b(x)$ para todo *x*, y similar para d', b'.

$$
\begin{aligned}
d(t) &= d(x_{i+1}) + b(x_{i+1}) \\
&\leq d(x_{i+1}) + b'(x_{i+1}) && \text{(Edmonds – Karp)} \\
&< d'(x_{i+1}) + b'(x_{i+1}) && \text{(5)} \\
&= d'(t)
\end{aligned}
$$

Hemos probado $d(t) < d'(t)$ en este subcaso.

Subcaso B: $d(x_{i+1}) = d'(x_{i+1}) = i + 1$

Como *i* es el primer *i* para el cual $x_i \rightarrow x_{i+1}$ no está en NA:
Entonces la porción del camino $x_0\ x_1 \ldots x_i$ **sí está** en NA.
Esto implica (al ser NA un network por niveles) que $d(x_i) = i$.
Entonces, en NA, x_i está en el nivel *i*, y x_{i+1} en el nivel $i + 1$.
En particular, concluímos que no solo no está en NA el lado $x_i \rightarrow x_{i+1}$ sino que tampoco está el lado $x_{i+1} \rightarrow x_i$ (pues los niveles no son legales para que ese lado esté). □
Como $d(x_i) = i$, $d(x_{i+1}) = i + 1$, entonces x_i , x_{i+1} están a distancia "legal" para que pueda existir el lado $x_i \rightarrow x_{i+1}$.
Pero ese lado no está en NA. Por lo que concluímos que:

1. $x_i \rightarrow x_{i+1}$ es lado del network original pero esta saturado, o:
2. $x_{i+1} \rightarrow x_i$ es lado del network original pero esta vacio.

Pero $x_i \rightarrow x_{i+1}$ si es un lado en NA' .

Así que la situación es:

1. $x_i \rightarrow x_{i+1}$ es lado del network original, estaba saturado al construir NA pero des-saturado al construir NA' , o
2. $x_{i+1} \rightarrow x_i$ es lado del network original, estaba vacío al construir NA pero no vacio al construir NA'.

La única forma en que pase [1] es que al pasar de *f* a *f'* dessaturamos el lado x i x i+1 , es decir, devolvimos flujo, lo que dice que usamos en algún momento el lado como backward.
Como pasamos de *f* a *f'* usando NA, esto significa que en NA debemos haber usado el $x_i \rightarrow x_{i+1}$.
Esto es un absurdo pues habíamos visto que ese lado no puede estar en NA, pues estaríamos yendo del nivel $i + 1$ al *i*.
La única forma en que pase [2] es que al pasar de *f* a *f'* mandamos algo de flujo por el lado $x_{i+1} \rightarrow x_i$. Asi que ese lado también debe ser un lado en NA, lo cual como dijimos es un **absurdo**.

Corolario de la distancia en netwoks auxiliares sucesivos aumenta

En cualquier corrida de una algoritmo “tipo” Dinic, hay a lo sumo *n* networks auxiliares.

Demostración

La distancia entre *s* y *t* aumenta en al menos 1 por cada network auxiliar.

Salvo por el último network auxiliar, donde la distancia es infinita, en los demas la distancia debe ser un número natural entre *1* y *n – 1*.

Por lo tanto hay a lo sumo *n* networks auxiliares.

Complejidad del algoritmo de Dinic [2]

Si probamos que la complejidad de hallar un flujo bloqueante en un network auxiliar (CFB) es *O(m n)* tanto en Dinitz como en la versión de Even, entonces por el teorema general tendremos que la complejidad es:

$O(n*(m+CFB)) = O(n*(m+O(m\ n)) = O(n*\ O(m\ n)\ ) = O(m\ n^2)$

Complejidad = (número de networks auxiliares ≤ n)*(complejidad de crear un network auxiliar *O(m)* por BFS + complejidad de hallar un flujo bloqueante en ese network auxiliar *CFB*)

Dinitz Original

Recordemos que en la versión original de Dinitz, cada camino entre *s* y *t* se encuentra usando *DFS*, pero el network auxiliar tiene la propiedad extra que se garantiza que toda búsqueda *DFS* nunca va a tener que hacer backtracking, pues cada vértice con lado entrante tiene un lado saliente.

Pero esta propiedad tiene el costo de tener que manternerla entre camino y camino, revisando el network auxiliar.

A esta operación Dinitz la llama “PODAR”. Entonces no solo tenemos que calcular la complejidad de encotrar todos los caminos, sino tambien la complejidad de todos los “PODAR”

Complejidad de encontrar todos los caminos

Construir un camino dirigido desde *s* a *t* es muy fácil:

```
Tomar p = s
while ( p ≠ s )
       tomar cualquier vecino de p // siempre hay un q
       agregar p→q al camino
       tomar p = q
endwhile
```

Entonces la construcción de cada camino es *O(r)*, donde *r* es el número de niveles.

Como $r < n$ podemos decir que esta parte es *O(n)*.

Luego de cada camino, se borran del network auxiliar los lados saturados, pero esto es simplemente recorrer una vez más el camino, así que es otro *O(n)*.

Como cada camino satura y por lo tanto borra al menos un lado, hay a lo sumo *m* caminos.

Así que el total de la complejidad de entontrar todos los caminos es *O(m n)*

Complejidad de todos los PODAR

PODAR va recorriendo todos los vértices, desde niveles más altos a más bajos, chequeando si tienen lados de salida, y borrándolos si no tienen lados de salida.

Puede ser que no se tenga que borrar ningún vértice, que se borren algunos o muchos. Pero cada vértice solo se lo puede borrar una vez. La clave está en no contar la complejidad de cada PODAR y cuántos hay, sino la complejidad de **todos** los PODAR en **conjunto**.

Para analizar la complejidad, llamemos *PV* a la parte de PODAR de simplemente recorrer los vértives mirando si tienen lados de salida o no.

Y llamemos *B(x)* a borrar todos los lados de entrada de un vértice *x*.

Entonces PODAR es hacer *PV,* deteniéndonos en los vértices *x* para los cuales no tienen lados de salida y activando *B(x)* para esos.

Complejidad de *PVs*

La complejidad de cada *PV* es *O(n)*, pues chequea todos los vértices.

Hay un *PV* en cada PODAR, así que hay a lo sumo *m + 1 PV* en total.

La complejidad total de **todos** ellos es entonces *O(n m).*

Complejidad de *B(x)*

La complejidad de un *B(x)* es *O(d(x)).* En realidad, del grado de entrada de *x).* Esto depende de cómo sea la estructura que se use para guardar los lados.

Se podría programar *B(x)* para que fuese *O(1)*, o también, podría demorar *O(n)* o incluso *O(m)*.

También observemos que si se "activa" la llamada a *B(x)*, no se vuelve a hacer *B(x)* para ese vértice *x* nunca más, porque borramos todos los lados y además borramos el vértice.

Entonces, independientemente de cuantos *B(x)s* hau en **un** *PV,* lo que sabemos es que al final de todo, habrá a lo sumo un *B(x)* por cada vértice *x*.

Como la complejidad de *B(x)* es *O(d(x))*, entonces la complejidad del conjunto de *B(x)s* es

$O(\Sigma_x\, d(x)\,) = O(m)$ lema del apretón de manos: $\Sigma_x\, d(x) = 2m$.

Resumiendo

- La complejidad total de buscar todos los caminos y borrar los lados saturados es *O(n m).*
- La de los *Pvs* es *O(n m)*
- La de todos los *B(x)s* es *O(m)*

Entonces la complejidad total del paso bloqueante es:

O(n m) + O(n m) + O(m) = O(n m)

Y la del algoritmo completo, $O(m\, n^2)$ como vimos.

Dinic – Even

La versión de Ever no tiene los PODAR, pero ahora DFS puede hacer backtrack. Ya no sabemos que cada DFS sea *O(n)* y el análisis se complica por ahí.

Para poder analizar la complejidad de la versión de Ever nos va a convenir dar un pseudocódigo (para la parte de encontrar un flujo bloqueante en el network auxiliar).

En el pseudocódigo $\Gamma^+(x)$ es del **network auxiliar** no del original. Al igual que "borrar lados" se refiere al network auxiliar.

Recordemos que lo que hace Ever es un DFS normal, excepto que cuando se ve forzado a hacer un backtrack "guarda" la información de que debió hacer un backtrack ahí. Una forma de hacerlo es borrar el lado por el que se hace backtrack.

Para poder esctibir el pseudocódigo, llamaremos a partes del código de la siguiente forma:

A **AVANZAR(x)**: Elegimos algún vecino de $\Gamma^+(x)$, agregamos $x \to y$ al camino y cambiamos $x = y$. Solo si $\Gamma^+(x)$ no es vacío

R RETROCEDER(x): Tomamos *z* el vértice anterior a *x* en la pila, borramos $z \to x$ del camino y del network auxiliar, y hacemos $x = z$. Solo retrocedemos si $x \neq s$.

I **INCREMENTAR:** Una vez construido el camino, calculamos cuánto se puede mandar por él, mandamos eso y borramos del network cualquier lado que haya sido saturado.

```
G = 0                          // flujo bloqueante g
stopflag := 1                  // para saber cuando parar
while (stopflag)
       p = [s], x =  s              // inicialización inicial de x y del camino p
       while ((x ≠ t) and stopflag)
              if Γ⁺(x) ≠ ∅ then AVANZAR(x)
              else if (x ≠ s) then RETROCEDER(x)
              else stopglag = 0
       if (x == t) then INCREMENTAR
return g
```

Una vez que se tiene creado el camino para aumentar el flujo, el `INCREMENTAR` es *O(n),* pues la longitud del camino es a lo sumo *n*, y `INCREMENTAR` aumenta el flujo a lol largo de ese camino y borra los lados saturados.

Además, la complejidad de `AVANZAR` es *O(1),* pues consiste en buscar al primer vértice en la lista de $\Gamma^+(x)$, agregar un lado al camino, y cambiar x.

La complejidad de `RETROCEDER` también es *O(1)* pues simplemente borramos un lado y cambiamos quién es *x*.

Dinic-Even es una sucesión de As, Rs, Is. Podemos dividir esta sucesión de letras en "palabras" de la forma AAA...AAX donde X es R o I.

¿Cuántas palabras hay?

Cada palabra termina en un X= R ó I
R borra el lado por el cual retrocede.
I manda flujo por un camino en el cual se satura al menos un lado, y borra todos los lados saturados.
Concluímos que X, sea R o I, borra al menos un lado.
Por lo tanto la cantidad de palabras es a lo sumo *m*

¿Cuál es la complejidad de cada palabra?

Vimos que R y A son *O(1).* I es *O(n).*
Así que si una palabra A… AX tiene *k* As, la complejidad de la palabra será *O(k+1) = O(k)* si X es R y *O(k + n)* si X es I.
Pero A mueve el extremo del camino donde estamos parados un nivel para adelante. Como hay *r* niveles, entonces $k \leq r$.
Como puede haber a lo sumo *n* niveles, entonces $r \leq n$.
Por lo tanto la complejidad de A… AX es *O(n)* si X = R y *O(n + n) = O(n)* si X = I.

En resumen

- Hemos probado que :
  - Hay a lo sumo *m* palabras
  - La complejidad de cada palabra es *O(n)*
- La complejidad total del paso de encontrar un flujo bloqueante es (n° de palabras)*(compl de c/palabra) = *O(m n).*
- Por lo tanto, como vimos al principiom la complejidad total de Dinic-Even es *O(n) * O(m n) = O(m n²)*

Complejidad del algoritmo de Wave [3]

FB: ForwardBalance de Tarjan

BB: BackwardBalance de Tarjan

La complejidad de Wave es $O(n^3)$.

Demostración:

Como Wave es un algoritmo que usa la técnica de los networks auxiliares y vimos que puede haber a lo sumo n networks auxiliares antes de encontrar un flujo maximal, vemos que la complejidad de Wave es n veces la complejidad de encontrar un flujo bloqueante en un network auxiliar.

Así que basta con demostrar que la complejidad de encontrar un flujo bloqueante en un network auxiliar es $O(n^2)$.

Para encontrar un flujo bloqueante, Wave hace una serie de ciclos de olas hacia adelante y olas hacia atrás.

En cada ola hacia adelante hacemos una serie de $FB(x)$ y en cada ola hacia atrás una serie de $BB(x)$.

En cada uno de esos, revisamos una serie de y's, ya sea en $\Gamma^+(x)$ o en $M(x)$.

Para cada uno de esos y que miramos, tardamos $O(1)$ (pues hay que calcular cuánto mandar/devolver, restarlo de $D(x)$, sumarlo a $D(y)$, cambiar cuánto vale g en el lado correspondiente, y hacer algún `if`.

Por lo tanto la complejidad de encontrar un flujo bloqueante con Wave es simplemente la cantidad total de veces que hacemos estas cosas.

Para eso, podemos dividirlas en categorías. Cuando estamos mirando un $y \in \Gamma^+(x)$ para ver cuánto mandar desde x a y, pueden pasar dos cosas:

1. Luego de terminar, el lado $x \to y$ queda saturado.
2. No queda saturado.

Sea S la cantidad total sobre todas las olas hacia adelante de la categoría [1.] arriba, y P la cantidad total, sobre todas las olas hacia adelante, de la categoría [2.] arriba.

Similarmente, cuando estamos mirando $y \in M(x)$ para que x le devuelva flujo a y, pueden pasar dos cosas:

1. Luego de procesarlo, el lado $y \to x$ queda vacío.
2. No queda vacío.

Sea V la cantidad total sobre todas las olas hacia atrás, de procesamientos de la categoría [1.] arriba, y Q la cantidad total de procesamientos sobre todas las olas hacia atrás, de lados de la categoría [2.] arriba.

Entonces la complejidad del paso bloqueante de Wave es $S + P + V + Q$

S:

Supongamos que un lado $x \to$ y se satura. En el algoritmo de Wave se borra y de $\Gamma^+(x)$. Así que efectivamente borramos el lado $x \to y$ del network auxiliar.

Y como nunca más lo agregamos, nunca más podríamos saturar el lado $x \to y$, los lados se saturan una sola vez. Así que S está acotado por m.

Está bien borrar $x \to y$ porque **no puede volver a ser usado.** Para poder ser usado otra vex, al estar saturado primero debería des-saturarse, es decir primero debería DEVOLVERLE flujo

a *x*. Y la única forma en que *y* le puede devolver flujo a *x* es si *y* ya está bloqueado. Pero si está bloqueado, x no puede mandarle más flujo a y.

V:

Similarmente, supongamos que $y \rightarrow x$ se vacía.

$y \rightarrow x$ solo se puede vaciar si *x* está bloqueado pues de otra forma no podría devolverle flujo a *y*. Pero si x está bloqueado, el vértice *y* nunca más puede mandarle flujo.

Si $y \rightarrow x$ nunca más puede recibir flujo, entonces menos aún va a poder volver a vaciarse. Así que *V* también está acotado por el número total de lados, *m*.

P:

En cada *FB(x)* buscamos vecinos de *x* y les mandamos todo el flujo que podamos:

$Min\{D(x), c(x \rightarrow y) - g(x \rightarrow y)\}$

Si ese mínimo es igual a $c(x \rightarrow y) - g(x \rightarrow y)$ el lado $x \rightarrow y$ queda saturado, lo returamos de $\Gamma^{+}(x)$ y continuamos con otro.

Entonces, de entre todos los vértices de *x* hay **a lo sumo** un solo tal que ese mínimo es *D(x).*

Es decir, al hacer *FB(x),* todos los lados $x \rightarrow y$ que miramos, salvo a lo sumo uno, se saturan.

Concluímos entonces que en cada *FB* hay a lo sumo UN lado procesado como parte de *P.* Con lo cual *P* está acotado superiormente por la cantidad total de *FB*.

En cada ola hacia adelante, hay a lo sumo $n - 2$ *FB* así que solo necesitaríamos calcular cuántas olas hacia adelante hay.

En cada ola hacia adelante, salvo la última, **al menos un** vértice es bloqueado. Porque si una ola hacia adelante no bloquea ningún vértice, esto significa que los balanceó a todos, y si los balancea a todos, el algoritmo termina, así que debe ser la última ola.

Como los vértices **nunca se desbloquean** puede haber a lo sumo *n* olas hacia adelante.

Por lo tanto tenemos a lo sumo $n - 2$ *FB* por cada ola hacia adelante, y tenemos $O(n)$ olas hacia adelante, lo que significa que tenemos en total $O(n^2)$ *FB* y por lo tanto, la cantidad de procesamientos de *P* es $O(n^2)$

Q:

Similarmente, al hacer un *BB*, a lo sumo un lado no se vacía, así que *Q* es la cantidad total de *BB*, que son $n - 2$ por cada ola hacia atrás.

Y como el número de olas hacia atrás es igual al número de olas hacia adelante y vimos que estas están acotadas por *n*, tenemos que Q también es $O(n^2)$.

En resumen

La complejidad de hallar un flujo bloqueante en un network auxiliar con Wave es:

$S + P + V + Q = O(m) + O(n^2) + O(m) + O(n^2) = O(n^2)$

Como dijimos al comienzo, esto implica que la complejidad total de Wave es $O(n^3)$.

Observación trivial

Sea f definida en lados y $A, B, C \subseteq V$ tales que $A \cap C = \emptyset$, entonces es trivial ver que $f(A \cup C, B) = f(A, B) + f(C, B)$.

Y similarmente $f(B, A \cup C) = f(B, A) + f(B, C)$.

teorema de Ford-Fulkerson A

Si f es un flujo y S es un corte, entonces $v(f) = f(S, \overline{S}) - f(\overline{S}, S)$

Demostración:

Sea f un flujo cualquiera y S un corte cualquiera, por la propiedad de la conservación, tenemos que $out_f(x) - in_f(x) = 0 \ \forall \ x \neq s, t$.

Y para $x = s$, tenemos $out_f(x) - in_f(x) = v(f)$.

Entonces,

$$\Sigma_x \, (out_f(x) - in_f(x))\,[x \in S] = out_f(s) - in_f(s) + \Sigma_x \, (out_f(x) - in_f(x))\,[x \in S][x \neq s]$$

$$= v(f) + \Sigma_x 0[x \in S][x \neq s]$$

$$= v(f)$$

$$\text{v(f)} = \Sigma_x \, (out_f(x) - in_f(x))\,[x \in S]$$

$$= \Sigma_x \, (\, f(\{x\}, V) - f(V, \{x\})\,)\,[x \in S]$$

$$= \Sigma_x \, (\, f(\{x\}, V)\,[x \in S] \; - \Sigma_x \; f(V, \{x\})\,)\,[x \in S]$$

$$= f(S, V) - f(V, S)$$

$$= f(S, S \cup \overline{S}) - f(S \cup \overline{S}, S) \qquad \text{// por observación trivial}$$

$$= f(S, S) + f(S, \overline{S}) - f(S,S) - f(\overline{S}, S)$$

$$= f(S, \overline{S}) - f(\overline{S}, S)$$

Max Flow Min Cut [6]

Si f es un flujo, las siguientes afirmaciones son equivalentes:

1. Existe un corte S tal que $v(f) = cap(S)$.
2. f es maximal.
3. No existen f-caminos aumentantes.

Demostración

Para probar la equivalencia de 1, 2, 3 probaremos que $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 1$

1 ⇒ 2

Sean S, f como en [1] y sea g un flujo cualquiera.

Como **el valor de todo flujo es menor o igual que la capacidad de todo corte**,

$v(g) \leq cap(S)$.

Pero por hipótesis tenemos que $cap(S) = v(f)$.

Concluímos que $v(g) \leq v(f)$ y por lo tanto f es maximal.

Además, si T es un corte, $cap(T) \geq v(f) = cap(S)$, es decir, S es minimal.

2 ⇒ 3

Si existiese un f-camino aumentante, podríamos mandar un $\varepsilon > 0$ a través de él obtendríamos un flujo f^* tal que $v(f^*) = v(f) + \varepsilon$. Esto diría que $v(f^*) > v(f)$ lo cual contradice que f sea maximal.

3 ⇒ 1

Necesitamos construir un *S* que sea corte con *cap(S) = v(f).* Primero lo definimos y luego probamos sus propiedades:

$S = \{s\} \cup \{x \in V: \exists\ f$ - camino aumentante desde *s* a *x}*

Como estamos suponiendo que vale **(3)**, entonces $t \notin S$. Como $s \in S$ por definición, y $t \notin S$, entonces *S* es un corte.

Por lo tanto, por **teorema de Ford-Fulkerson A** tenemos que $v(f) = f(S, \overline{S}) - f(\overline{S}, S)$

f(S, $\overline{S}$)

$f(S, \overline{S}) = \Sigma_{x,y}\, f(x \to y)\ [x \in S][y \notin S][x \to y \in E]$

Consideramos un par *x, y* de los que aparecen en esa suma.

Como $x \in S$, entonces existe un f-camino aumentante entre *s* y *x*, digamos

$s=x_0,\ x_1, \ldots, x_r=x$.

Pero como $y \notin S$, entonces no existe ningún *f*-camino aumentante entre *s* e *y*.

En particular,

$s=x_0, x_1, \ldots, x_r=x, y$ **NO es un *f* – camino aumentante**.

Pero $x \to y \in E$, así que **podría** serlo. Pero no lo es porque no podemos usar el lado $x \to y$ por estar saturado, es decir: $f(x \to y) = c(x \to y)$. Esto es cierto para cualesquiera x, y que aparezcan en esa suma.

Entonces,

$$f(S, \overline{S}) = \Sigma_{x,y}\, f(x \to y)\ [x \in S][y \notin S][x \to y \in E]$$
$$= \Sigma_{x,y}\, c(x \to y)\ [x \in S][y \notin S][x \to y \in E]$$
$$= c(S, \overline{S}) = cap(S)$$

f($\overline{S}$, S)

$f(\overline{S}, S) = \Sigma_{x,y}\, f(x \to y)\ [x \notin S][y \in S][x \to y \in E]$

Consideremos un par *x, y* de los que aparecen en esa suma.

Como $y \in S$, entonces existe un *f* – camino aumentante entre *s* e *y*, digamos $s=x_0, x_1, \ldots, x_r=y, x$ **NO es un *f* – camino aumentante.**

Pero $x \to y \in E$, así que podría serlo, usando *y,x* como lado backward. Pero no lo es porque $f(x \to y) = 0$. Esto es cierto para cualesquiera *x, y* que aparezcan en esa suma.

Entonces,

$$f(\overline{S}, S) = \Sigma_{x,y}\, f(x \to y)\ [x \notin S][y \in S][x \to y \in E]$$
$$= \Sigma_{x,y}\, 0\ [x \notin S][y \in S][x \to y \in E]$$
$$= 0$$

Conclusión

Hemos probado que para este *S*

- $f(S, \overline{S}) = cap(S)$
- $f(\overline{S}, S) = 0$

Por lo tanto:

$$v(f) = f(S, \overline{S}) - f(\overline{S}, S) = cap(S)$$

Cota de Hamming [7]

Sea C un código de longitud *n*, $\delta = \delta(C)$ y $t = \lfloor \frac{\delta - 1}{2} \rfloor$. Entonces:

$$\# \quad C \leq \frac{2^n}{1 + n + \ldots + \binom{n}{t}}$$

Demostración:

Sea $A = \cup_{v \in C} D_t(v)$.

Como $t = \lfloor \frac{\delta - 1}{2} \rfloor$, ent *C* corrige *t* errores, lo cual implica que $D_t(v) \cap D_t(w) = \emptyset$

Por lo tanto la unión que forma *A* es una unión disjunta, lo cual nos permite calcular la cardinalidad de *A* como:

$\#A = \Sigma_{v \in C} \#D_t(v)$

Para calcular la cardinalidad de los D_t's, definamos los siguientes conjuntos

$S_r(v) = \{w \in \{0, 1\}^n : d_H(v, w) = r\}$

Recordemos que $D_t(v) = \{w \in \{0, 1\}^n : d_H(v, w) \leq t\}$, por lo tanto $D_t(v) = \cup^t_{r=0} S_r(v)$.

Y como no podemos tener $d_H(v, w) = r$ y al mismo tiempo $d_H(v, w) = r'$ para $r' \neq r$, entonces esa unión es disjunta.

Así que $\#D_t(v) = \Sigma^t_{r=0} \#S_r(v)$.

Necesitamos calcular $\#S_r(v)$.

Un $w \in S_r(v)$ es una palabra que difiere de *v* en exactamente *r* de los *n* lugares posibles.

Por lo tanto, podemos asociar a cada $w \in S_r(v)$ un subconjunto de cardinalidad *r* de $\{1, 2, \ldots, n\}$: el conjunto de índices *i* tales que $w_i \neq v_i$.

Como estamos trabajando con códigos binarios, podemos hacer la vuelta: dado un subconjunto de cardinalidad *r* de $\{1, 2, \ldots, n\}$, crear un $w \in S_r(v)$, pues dado un conjunto de *i*'s, simplemente cambiamos los bits de *v* en esos *i*'s.

Formalmente, sea $\Xi_r = \{Z \subseteq \{1, 2, \ldots, n\} : \#Z = r\}$

Definamos $\Psi : S_r(v) \rightarrow \Xi_r$ por medio de $\Psi(w) = \{i : w_i \neq v_i\}$.

Y definimos $\Phi : \Xi_r \rightarrow S_r(v)$ por medio de

$\Phi(Z) =$ *el único* $w : w_i = v_i$ *si* $i \notin Z$ *y* $w_i = 1 \oplus v_i$ *si* $i \in Z$.

Entonces Ψ , Φ son inversas una de la otra. Por lo tanto, $\#S_r(v) = \# \Xi_r = \binom{n}{r}$

Por lo tanto,

$$\#S_r(v) = \binom{n}{r} \quad , \quad \#D_t(v) = \sum_{r=0}^{t} \binom{n}{r} \quad , \text{ y } \quad \#A = \sum_{v \in C} \left(\sum_{r=0}^{t} \binom{n}{r} \right)$$

Observar que la suma no depende de *v*, así que esto queda;

$$\#A = \left(\sum_{r=0}^{t} \binom{n}{r} \right) \quad x \#C.$$ Por lo tanto,

$$\#C = \#A / \left(\sum_{r=0}^{t} \binom{n}{r} \right) \quad \leq \quad \frac{2^n}{\sum_{r=0}^{t} \binom{n}{r}}$$

Solo resta observar que como *A* es un subconjunto de $\{0, 1\}^n$, su cardinalidad es menor o igual que 2^n, lo cual finaliza la prueba.

δ a partir de matriz de chequeo [8]

Si H es matriz de chequeo de C, entonces
$\delta(C) = Min\{j : \exists$ un conjunto de j columnas LD de $H\}$
(LD es "linealmente dependiente")

Demostración:
Idea:

Sea $s = Min\{j : \exists$ un conjunto de j columnas LD de $H\}$

Probaremos que $\delta(C) \leq s$ y luego $s \leq \delta(C)$

Denotaremos la columna *j-ésima* de H como $H^{(j)}$.

δ(C) ≤ s

Por definición de s, existe un conjunto LD de columnas de $H : \{H^{(j1)}, H^{(j2)}, \dots, H^{(js)}\}$.
Esto significa que existen $c_1, \dots c_s$ **no todos nulos** tales que:
$c_1 H^{(j1)} + c_2 H^{(j2)} + \dots + c_s H^{(js)} = 0$
Sea $w = c_1 e_{j1} + c_2 e_{j2} + \dots + c_s e_{js}$
Donde e_j es el vector con todos 0 salvo un 1 en la coordenada j.
Como no todos los c_j son 0 entonces $w \neq 0$.

$$
\begin{aligned}
H w^t &= H(c_1 e_{j1} + c_2 e_{j2} + \dots + c_s e_{js})^t \\
&= c_1 He_{j1}^t + c_2 He_{j2}^t + \dots + c_s He_{js}^t \\
&= c_1 H^{(j1)} + c_2 H^{(j2)} + \dots + c_s H^{(js)} \qquad // \text{ pues } He_i^t = H^{(j)} \\
&= 0
\end{aligned}
$$

Por lo tanto, $w \in C$, pues $C = Nu(H)$.
Pero su peso es ≤ s, pues es suma de a lo sumo s e_j.
Y vimos que $w \neq 0$.

Sabemos que
$\boldsymbol{\delta(C) = Min\{ |v| : v \in C, v \neq 0\} \leq |w|}$ *//hace falta demostrar esto?*
Por lo tanto, como $w \in C$, y $w \neq 0$, entonces w está en ese conjunto.
Y su peso es ≤ s así que tenemos $\delta(C) \leq s$.

s ≤ δ(C)
Sea $v \in C$ tal que $\delta(C) = | v |$.
$\delta(C) = | v |$ implica que existen $i_1, \dots, i_{\delta(C)}$ con $v = e_{i1}, \dots, e_{i\,\delta(C)}$.

Como $v \in C$, entonces $Hv^t = 0$.

Con el mismo cálculo que antes, concluímos que

$H^{(i1)} + H^{(i2)} + \dots + H^{(i\,\delta(C))} = Hv^t = 0$
Pero $H^{(i1)} + H^{(i2)} + \dots + H^{(i\,\delta(C))} = 0$ dice que $\{H^{(i1)}, H^{(i2)}, \dots, H^{(i\,\delta(C))}\}$ es un conjunto LD de columnas de H.
Por lo tanto, $s = Min\{j : \exists$ un conjunto de j columnas LD de $H\} \leq \delta(C)$

propiedad clave para utilidad de los códigos cíclicos
Sea C un códifo cíclico, $w \in C$ y v una palabra cualquiera. Entonces $v \odot w \in C$
Demostración:
Por la propiedad anterior , $x \odot w = rot(w) \in C$ (pues C es cíclico)
Por lo tanto $x^i \odot w \in C$ para todo i.

Como C, al ser cíclico, es lineal, entonces cualquier combinación lineal de $x^i \odot w$ estará en C.
Es decir, $\Sigma\ a_i\ (x^i \odot w) \in$ C para cualesquiera a_i .
Pero $\Sigma\ a_i\ (x^i \odot w) = (\Sigma\ a_i\ x^i) \odot w$.
Concluimos que $v \odot w \in C$ para cualesquiera $v = \Sigma\ a_i x^i$.

Teorema fundamental de códigos cíclicos [9]

Sea *g(x)* el polinomio generador de un código cíclico C de longitud *n*. Entonces:

1. *C* está formado por los múltiplos de *g(x)* de grado menor que *n*:
   C = {p(x) : gr(p) < n & g(x) | p(x) }
2. *C = {v (x) ⊙ g(x) : v* es un polinomio cualquiera}
3. *gr(g(x)) = n – k*
4. *g(x)* divide a $1 + x^n$

Demostración
1 y 2
Sea $C_1 = \{p(x) : gr(p) < n \ \&\ g(x) \mid p(x)\}$ y
$C_2 = \{v(x) \odot g(x) : v$ es un polinomio cualquiera$\}$.

Por la **propiedad clave para utilidad de los códigos cíclicos** , $C_2 \subseteq C$, pues $g(x) \in C$.
Sea $p(x) \in C$.
Dividimos $p(x)$ por $g(x)$, obteniendo polinomios $q(x)$ y $r(x)$, con $gr(r) < gr(g)$ tal que $p(x) = q(x)g(x) + r(x)$.
Por lo tanto $r(x) = p(x) + q(x)g(x)$.
Como $gr(r) < gr(g) < n$, entonces $r(x) = r(x)\ mod\ (1 + x^n)$
Como $p(x) \in C$, entonces $gr(p) < n$, y $p(x) = p(x)\ mod\ (1 + x^n)$
Entonces:

$$\begin{aligned} r(x) &= r(x)\ mod\ (1 + x^n) \\ &= (p(x) + q(x)g(x))\ mod\ (1 + x^n) \\ &= p(x)\ mod\ (1 + x^n) + (q(x)g(x))\ mod\ (1 + x^n) \\ &= p(x) + q \odot g \end{aligned}$$

Como $p(x) \in C$ y $q \odot g \in C$ y C es lineal, concluímos que $r \in C$.
Pero $gr(r) < gr(g)$ que es el polinomio **no nulo** de **menor** grado de C. Concluímos que $r = 0$.
Por lo tanto, $p(x) = q(x)\ g(x) + r(x) = q(x)\ g(x) \in C_1$.

Entonces concluímos que $C \subseteq C_1$, y $C_2 \subseteq C$, solo nos resta ver que $C_1 \subseteq C_2$.
Pero esta inclusión es obvia, pues si $gr(p) < n$ y $p(x) = q(x)\ g(x)$, entonces:
$p(x) = p(x)\ mod\ (1 + x^n)$ (pues $gr(p) < n$)
Y por lo tanto $p(x) = q(x)\ g(x)\ mod\ (1 + x^n) = q \odot g \in C_2$.

3
Sea t el grado de $g(x)$.
Por **1** $p(x) \in C$ sii es de la forma $q(x)\ g(x)$ para algún polimonio $q(x)$.

Pero como el grado de los elementos de C es menor que n, entonces el grado de $q(x)g(x)$ debe ser menor que n.
Por lo tanto el grado de $q(x)$ debe ser menor que $n - t$. Así, para cada polinomio de grado menor que $n - t$ corresponde un polinomio de C, y vicecersa.
Por lo tanto la cardinalidad de C es igual a la cardinalidad del conjunto de polinomios de grado menor que $n - t$.

Los polinomios de grado menor que $n - t$ tienen $n - t$ coeficientes (los de los términos de grado *0, 1, ..., n − t − 1*).
Cada uno de esos coeficientes puede ser 1 o 0, así que cada uno tiene dos posibilidades.
Como son $n - t$, el total de polinomios posibles es 2^{n-t}.
Entonces hemos probado que la cardinalidad de C es 2^{n-t}. Pero como C es lineal, sabemos que su cardinalidad es 2^k.
Asi que $2^k = 2^{n-t}$, por lo tanto $k = n - t$, y el grado de $g(x)$ es $t = n - k$.

4
Dividimos $1 + x^n$ por $g(x)$, obteniendo $q(x)$, $r(x)$ con $gr(r) < gr(g)$ tal que $1 + x^n = q(x)\, g(x) + r(x)$.
Por lo tanto $r(x) = 1 + x^n + q(x)\, g(x)$.
Como $gr(r) < gr(g) < n$, $r(x) = r(x) \bmod (1 + x^n)$.
Así: $r(x) = (1 + x^n + q(x)\, g(x)) \bmod (1 + x^n) = q \odot g \in C$
(en la igualdad anterior usamos $(1 + x^n) \bmod (1 + x^n) = 0$)
Como $r \in C$ y $gr(r) < gr(g)$, entonces $r = 0$ y $g(x) | (1 + x^n)$

Teorema de Hall [10]

Si G es un grafo bipartito con partes X e Y y $|S| \leq |\Gamma(S)|$ para todo $S \subseteq X$, entonces existe un matching completo sobre X.

Demostración:
Supongamos que el teorema no es cierto, es decir que $|S| \leq |\Gamma(S)|$ para todo $S \subseteq X$ pero que no existe un matching completo sobre X.
Probaremos que el hecho de que no exista un matching completo sobre X implica que existe $S \subseteq X$ con $|S| > |\Gamma(S)|$ lo cual contradice la hipotesis.

La prueba de que existe tal S la haremos analizando el algoritmo que dimos antes.
Como estamos suponiendo que no existe matching completo sobre X, entonces corriendo el algoritmo para hallar matching maximal, usando Edmonds-Karp, obtenemos un matching maximal que NO cubre todos los vértices de X.

Sea S_0 aquellos vértices de X que, al terminar el algoritmo, han quedado sin cubrir. ($S_0 \neq \emptyset$ porque estamos suponiendo que no hemos cubierto a todos los vértices de X)

Entonces $x \in S_0$ si y solo si no existe $y \in Y$ tal que xy sea lado del matching.

Esto significa que $f(x \to y) = 0$ para todo y, donde f es el flujo maximal encontrado.
Por lo tanto $out_f(x) = 0$ para todo $x \in S_0$.
Como f es flujo, eso significa que $in_f(x) = 0$ para todo $x \in S_0$.

Viceversa, supongamos que x es un vértice tal que $in_f(x) = 0$.
Entonces $out_f(x) = 0$ y por lo tanto $f(x \to y) = 0$ para todo y lo cual significa que x no forma parte del matching.

Conclusión: $S_0 = \{x \in X : in_f(x) = out_f(x) = 0\}$.
En el último paso del algoritmo, Edmonds-Karp encuentra un corte minimal.
Digamos que el corte es C.

Ese corte C incluye a s, y los otros elementos de C son vértices de G, que pueden estar en X o en Y.
Así que existen $S \subseteq X$, $T \subseteq Y$ tal que $C = \{s\} \cup S \cup T$. Recordemos que C se obtiene al hacer la última cola BFS: C consiste de todos los vértices que en algún momento están en esa última cola.

Como $x \in S_0 \Leftrightarrow in_f(x) = 0 \Leftrightarrow f(s \to x) = 0$,
entonces los vértices de S_0 son precisamente todos los vértices que s agrega a la cola.
Por lo tanto, $S_0 \subseteq S$.

Ahora bien, todos los otros elementos de S (los que no están en S_0, si es que hay alguno) deben haber sido agregados a la cola por algún elemento distinto de s.
Pero un vértice solo puede agregar a algún vecino, ya sea vecino hacia adelante o hacia atrás.
Esto quiere decir que los vértices de S no pueden haber sido agregado por otros vértices de S, pues como $S \subseteq X$, no hay lados entre ellos.
Así que los vértices de S que no están en S_0 deben haber sido agregados por vértices de T.

Pero $T \subseteq Y$ y todo vértice y de Y tiene $\Gamma^+(y) = \{t\}$. Así que T solo puede agregar a un vértice de X a la cola si lo agrega por medio de un lado **backward .**
Para que y agregue a x como backward, debe pasar que $f(x \to y) > 0$.

En nuestro caso, esto implica $f(x \to y) = 1$ y que xy sea parte del matching.
Por lo tanto, cada y que pueda agregar a la cola a algún vértice en forma backward, puede agregar UN SOLO vértice.
Entonces cada vértice de T puede agregar un sólo vértice a la cola.

Pero ademas, **cada vértice de T efectivamente agrega un vértice a la cola**.
¿Por qué?
Supongamos que no, que exista $y \in T$ que no agrega a nadie a la cola.
Como C es un corte, $t \notin C$, así que y no debería poder agregar a t a la cola.

Con lo cual, debemos tener que $f(y \to t) = 1$.
Entonces $out_f(y) = 1$ y por lo tanto $in_f(y) = 1$ y debe existir $x \in X$ tal que $f(x \to y) = 1$.
Pero entonces x es "candidato" a ser agregado a la cola como backward por y.

Como estamos suponiendo que y no agrega a nadie a la cola, debe haber una razón para que y no agregue a x.
La única razón posible es que x ya esté en la cola.
Pero ¿quien lo agregó?
No lo puede haber agregado alguien de T pues si fuese así:

- y no lo agrega así que debería ser otro vértice z de T, agregandolo backward.
- Pero esto significaría que tanto xz como xy estan en el matching, absurdo.

Entonces la única posibilidad que queda es que haya sido agregado por s, es decir, que x esté en S_0.

Pero si $x \in S_0$, entonces $out_f(x) = 0$ lo que contradice que $f(x \to y) = 1$.
Concluimos entonces que todo y en T agrega a alguien a la cola, y por lo que vimos antes, agrega **exactamente un** tal alguien a la cola.
Esos elementos agregados son los de $S - S_0$, así que concluímos que $|S - S_0| = |T|$.

Analicemos ahora un poco a los vértices de T.
Todos ellos deben haber sido agregados por alguien de X de la cola, es decir, por alguien de S.
Esto sólo puede pasar si cada vértice de T es un vécino de algún vértice de S.
Por lo tanto, $T \subseteq \Gamma(S)$.

Habiendo visto que $T \subseteq \Gamma(S)$, ahora nos preguntamos si son iguales o no. Si no son iguales, entonces existe $y \in \Gamma(S)$ tal que $y \notin T$.

Como $y \in \Gamma(S)$ entonces existe $x \in S$ con $xy \in E$.
Como $x \in S$, x está en C.
Pero $y \notin T$, así que y no está en C.
Así que x nunca agregó a y.
Pero $xy \in E$, así que para que x no agregue a y, debe pasar que $f(x \to y) = 1$.

Pero entonces, ¿quien agregó a x?
y no fue porque nunca estuvo en la cola.
No puede ser ningún otro elemento de T pues $f(x \to y) = 1$ implica que no existe ningún otro z con $f(x \to z) = 1$.

Debe haber sido agregado por s, pero esto implica que $out_f(x) = 0$ **absurdo** pues $f(x \to y) = 1$
El absurdo vino de suponer que T y $\Gamma(S)$ no eran iguales, asi que concluimos que $T = \Gamma(S)$.

Entonces:
$|\Gamma(S)| = |T| = |S - S_0| = |S| - |S_0| < |S|$
(la ultima desigualdad pues $S_0 \neq \emptyset$).

Hemos probado que $|\Gamma(S)| < |S|$, lo que contradice la hipotesis de que $|S| \leq |\Gamma(S)|$ para todo $S \subseteq X$.

Teorema del matrimonio de König [11]

Todo grafo bipartito regular tiene un matching perfecto.

Demostración:

Dado un conjunto $W \subseteq V$, definamos $E_W = \{zw \in E \mid w \in W\}$
Sean X e Y las partes del grafo bipartito. Supongamos que $W \subseteq X$. Entonces:
$|E_W| = |\{zw \in E \mid w \in W\}| = \Sigma_{w \in W} |\{z : zw \in E\}|$

La última igualdad es así pues como $W \subseteq X$ entonces no estamos contando un lado zw dos veces, pues $zw \in E$ implica que $z \in Y$ y por lo tanto no puede estar en W.

Entonces $|E_W| = \Sigma_{w \in W} |\{z : zw \in E\}| = \Sigma_{w \in W} d(w)$
Como G es regular, $d(w) = \Delta$ para todo w, así que tenemos:
$|E_W| = \Sigma_{w \in W} d(w) = \Sigma_{w \in W} \Delta = \Delta . |W|$

Con un razonamiento similar, tenemos que si $W \subseteq Y$ entonces tambien vale que
$|E_W| = \Delta . |W|$
Apliquemos esta propiedad para diversos W.
Si tomamos $W = X$ tendremos que $|E_X| = \Delta . |X|$.
Pero G es bipartito, así que $E_X = \{xy \in E : x \in X\} = E$
Concluimos que $|E| = \Delta . |X|$.

Tomando $W = Y$ concluimos $|E_Y| = \Delta . |Y|$
Pero $E_Y = E$ así que tenemos que $|E| = \Delta . |Y|$
Entonces $|E|$ es igual a $\Delta . |X|$ y a $\Delta . |Y|$.

Por lo tanto $\Delta . |X| = \Delta . |Y|$ así que $|X| = |Y|$.

Entonces, como $|X| = |Y|$, para probar que existe un matching perfecto basta probar que existe un matching completo sobre X. Para probar esto, usaremos el teorema de Hall.

Sea entonces $S \subseteq X$. Sea $l \in E_S$.
Entonces existe $x \in S, y \in Y$ tal que l es de la forma $l = xy$.

Por lo tanto $y \in \Gamma(x) \subseteq \Gamma(S)$ (pues $x \in S$).
Entonces l es de la forma xy con $y \in \Gamma(S)$, lo cual implica que $l \in E_{\Gamma(S)}$.
Como l era cualquier elemento de E_S esto dice que $E_S \subseteq E_{\Gamma(S)}$.
Por lo tanto $|E_S| \leq |E_{\Gamma(S)}|$

Como $S \subseteq X$, entonces lo que probamos al principio de la prueba dice que
$|E_S| = \Delta . |S|$.
Como $\Gamma(S) \subseteq Y$, entonces $|E_{\Gamma(S)}| = \Delta . |\Gamma(S)|$.
Como probamos que $|E_S| \leq |E_{\Gamma(S)}|$, concluímos que $\Delta . |S| \leq \Delta . |\Gamma(S)|$.
Por lo tanto $|S| \leq |\Gamma(S)|$ y como esto vale para cualquier $S \subseteq X$, Hall nos dice que existe un matching completo sobre X lo cual completa la prueba.

Baby Brooks(caso no regular) [12]

Si G es conexo, entonces $\chi(G) \leq \Delta$, a menos que *G* sea un ciclo impar o un grafo completo.

Demostración

Si G es conexo, entonces existe un ordenamiento de los vértices tal que Greedy colorea todos los vértices, salvo uno, con Δ colores o menos:

Sea *x* tal que $d(x) = \delta$.

Corramos *BFS(x).* Cuando *BFS*(x) termina, obtiene la componente conexa donde esta *x*, pero como *G* es conexo, hay una sola componente conexa, así que se obtienen todos los vértices de *G*.

Al correr BFS(x), se irán incorporando ciertos vértices en cierto orden.

Ordenemos los vértices de *G* en el orden inverso al cual fueron incorporados por *BFS(x)*.

El orden será x_1 , x_2 , ..., x_n con $x_n = x$ porque *x* es el primero al correr *BFS(x)* así que es el último en el orden inverso.

Salvo *x*, todos los demás vértices son incorporados a la componente conexa que está construyendo *BFS(x)* por un vértice que es vecino y **que ya está**.

Así que en el orden en que se incorporan los vértices, todo vértice, salvo *x*, tiene un vecino **anterior**.

Entonces, en el orden inverso, tenemos que todo vértice (salvo *x*) tiene un vecino **posterior**.

Greedy le da el color 0 a x_1 . A los demas vértices, Greedy les revisa los colores de los vecinos anteriores.

Pero si tomamos un vértice x_i con $1 < i < n$, entonces x_i , por lo que dijimos antes, tiene al menos a un vértice posterior que es vecino de *x*.

Lo cual quiere decir que no todos sus vécinos estan antes que él en el orden.

Entonces x_i tiene a lo sumo $d(x_i) - 1$ vecinos anteriores, con lo cual Greedy elimina a lo sumo esa cantidad de colores.

Como $d(x_i) - 1 \leq \Delta - 1$, entonces si tenemos Δ colores disponibles, siempre habrá uno sobrante para colorear x_i. (siempre que $i < n$)

Nos queda colorear *x*

Si *G* no es regular, es trivial. Pues $d(x) = \delta$ y $\delta < \Delta$. Entonces para colorear *x* Greedy elimina a lo sumo $d(x) = \delta$ colores, y como $\delta < \Delta$, le queda al menos un color para colorear *x*.

Brooks en el caso regular **Demostración:** [13]

Vimos ya que podíamos colorear todos los vértices salvo posiblemente uno con Δ colores. Tenemos que dividir la prueba en varios casos:

- **Caso 1: Δ ≤ 1**
  - Entonces $G = K_2$ *ó* $G = K_1$, lo cual contradice la hipótesis.
- **Caso 2: Δ = 2**
  - Como *G* es regular y conexo, si tiene Δ = *2* debe ser un ciclo.
  - Como no puede ser un ciclo impar, debe ser un ciclo par.
  - Por lo tanto $\chi(G) = 2 = \Delta$
- **Caso 3: Δ ≥ 3**
  - Como observamos en **baby brooks** podemos colorear todos los vértices salvo, uno al cual llamaremos *x*, con Δ colores.
  - Sea $H = G - \{x\}$ es decir, *H* es el subgrado de G que se obtiene al remover *x* y todos los lados entre *x* y sus vecinos
  - Como estamos con el caso *G* regular, todos los vértices cumplen que su grado es igual a δ = Δ
  - Denotamos a los Δ vecinos de *x* como $\{x_0, x_1, \ldots, x_{\Delta-1}\}$
  - **Caso 3A: En $\{x_0, x_1, \ldots, x_{\Delta-1}\}$ hay menos de Δ colores**
    - En este caso, coloreamos *x* con el color que "falta" en $\{x_0, x_1, \ldots, x_{\Delta-1}\}$ y tenemos un coloreo con Δ colores de *G*.
  - **Caso 3B : En $\{x_0, x_1, \ldots, x_{\Delta-1}\}$ hay Δ colores**
    - Cada x_i es coloreado con un color distinto. Sin pérdida de generalidad, podemos suponer que el color de x_i es *i.*
    - Cada uno de esos vecinos tiene a su vez Δ vecinos pues estamos en el caso *G* regular.

      Uno de esos vecinos es *x*, que no está coloreado, pero los otros $\Delta - 1$ vecinos estan coloreados con algún color.

      ¿Cuántos colores hay entre los $\Delta - 1$ vecinos coloreados de x_i ?
    - **Caso 3B1: Existe *i* tal que x_i tiene dos vecinos del mismo color.**
      - En este caso, entre los Δ - *1* vecinos coloreados de x_i hay a lo sumo Δ - *2* colores, pues al menos un color está repetido.
      - Así que de entre los Δ colores que estamos usando hay uno que no es *i* ni está entre los colores de los vecinos de *i*.
      - Sea *r* un color ≠ *i* que no es color de ninguno de los vecinos de x_i. Entonces podríamos darle el color *r* a x_i sin problema.
      - Si hacemos eso, entonces ahora el color *i* no es color de ningún vecino **de *x*.** Y por lo tanto podemos colorear a *x* con el color *i.*
    - **Caso 3B2***:* **$\forall i$ x_i tiene a todos sus vecinos coloreados con distintos colores**
      - Ver **[Subgrafo generado por W]**
      - Teníamos los vértices x_i , cada uno de color *i*, y el grafo $H = G - \{x\}$ está coloreado con esos Δ colores. Y definimos $H_{i,j}$ = *el subgrafo de* $H = G - \{x\}$ *generado por los vertices de color i o j*.
      - Por más que *G* sea conexo, no sabemos ni siquiera si *H* lo es, y mucho menos sabemos si $H_{i,j}$ lo es. Así que miraremos a sus componentes conexas. (obviamente si H i,j es conexo habrá una sola, pero podria haber varias).
      - Concretamente, definimos, si i ≠ j:
        $CC_{i,j} =$ componente conexa de $H_{i,j}$ que tiene a x_i
      - **Caso 3B2A: Existen $i \neq j$ tales que $CC_{i,j} \neq CC_{j,i}$**

- Observemos que en una cadena de Kempe $CC_{i,j}$ por definición, los únicos colores de sus vértices son *i* o *j*. Cualquier otro vértice con el cual estén unidos (en *G*) debe tener color distinto de *i* o *j*.
- Por lo tanto si en $CC_{i,j}$ **intercambiamos** los colores *i, j* el coloreo sigue siendo propio. Al intercambiar los colores *i, j* los vértices antiguamente coloreados con *i, j* no tienen problemas entre ellos; y el resto de los vértices tampoco, porque tienen todos colores distintos.
- Como hipótesis del caso tenemos que $CC_{i,j} \neq CC_{j,i}$, entonces $x_j \notin CC_{i,j}$
- Entonces si intercambiamos los colores solamente de $CC_{i,j}$ el color de x_j **no se verá afectado**. Pero el color de x_i sí: como antes era *i*, ahora será *j*.
- Pero entonces el color *i* no es más un color de algún vecino de *x*, y podemos colorear a *x* con el color *i*.

- **Caso 3B2B: $CC_{i,j} = CC_{j,i}$ $\forall i, j$ con $i \neq j$**
  - **Caso 3B2B1: $CC_{i,j} = \{x_i, x_j\}$ $\forall i, j$ con $i \neq j$**
    - Pero $CC_{i,j}$ es una componente conexa, así que si $CC_{i,j} = \{x_i, x_j\}$ entonces tenemos que $x_i x_j$ es un lado, para todo $i \neq j$.
    - Si $x_i x_j \in E$ $\forall$ $i \neq j$ entonces $G\tilde{} = G[x, x_0, x_1, x_2, \ldots, x_{\Delta-1}]$ es un grafo completo. Y como además tiene $\Delta + 1$ vértices, todos sus vértices tienen grado Δ en $G\tilde{}$
    - Pero como tienen grado Δ en G, resulta que ninguno de los vértices de $G\tilde{}$ tiene un vecino (en G) **fuera** de $G\tilde{}$
      Por lo tanto $G\tilde{}$ es una componente conexa de G.
    - Pero G es conexo, así que concluimos que $G\tilde{} = G$.
    - Como $G\tilde{}$ es completo, esto implica que G es un grafo completo, absurdo por hipotesis.
  - **Caso 3B2B2: $\exists i,j$ $i \neq j$ con al menos un vértice en $Cc_{i,j}$ distinto de x_i, x_j, con al menos tres vecinos en $CC_{i,j}$**
    - Sea *w* el primer vértice cuando vamos desde x_i hasta x_j en $CC_{i,j}$ tal que *w* tenga al menos 3 vecinos.
    - Como solo hay dos colores en $CC_{i,j}$ todos los vecinos de *w* en $CC_{i,j}$ tienen el mismo color (*i* si el color de *w* es *j*, y *j* si el color de *w* es *i*) Entonces *w* tiene al menos 3 vecinos del mismo color.
    - Sea $p \in \{i, j\}$ el color de *w* y $q \in \{i, j\}$ distinto de *p* (es decir, *q* es el color de los vecinos de *w* en $CC_{i,j}$).
    - Como $w \neq x_i, x_j$ entonces *x* no es vecino de *w.* Por lo tanto *w* tiene Δ vecinos en $H = G - \{x\}$.
    - Como tiene 3 vecinos del mismo color, entonces $\Gamma(w)$ tiene a lo sumo $\Delta - 2$ colores. Uno de esos $\Delta - 2$ colores es *q*, y *p* NO ES uno de esos colores, pues *p* es el color de *w.*
    - Entonces si tomamos los colores de los vecinos de w y le agregamos el color *p* nos quedan $\Delta - 2 + 1 = \Delta - 1$ colores, entre los cuales estan *p* y *q*. (es decir, *i* y *j*).
    - Por lo tanto tenemos al menos un color, distinto de *i, j* que no es color ni de los vecinos de *w* ni de *w.* Entonces le podemos dar a *w* ese color y el coloreo sigue siendo propio.
    - Como *w* era el primer vértice en $CC_{i,j}$ con tres vecinos, entre x_i y *w* tenemos que $CC_{i,j}$ forma un camino. Al darle a *w* un color distinto de *i, j* esto hace que la **nueva** $CC_{i,j}$ consista sólo de esos vértices entre x_i y antes de *w.*
    - Como consecuencia la nueva $CC_{i,j}$ es distinta de la nueva $CC_{j,i}$.
    - Podemos proceder como en el **caso 3B2A.**
  - **Caso 3B2B3: 2B2B3: $\exists i \neq j$ tal que $CC_{i,j} \neq \{x_i, x_j\}$ y $\forall$ par i, j así, $CC_{i,j}$ es un CAMINO entre x_i y x_j**
    - Como $\Delta \geq 3$, entonces *x* tiene al menos 3 vecinos. Sea $x_k \neq x_i$, x_j un tercer vecino de *x*.

- Tenemos entonces las componentes conexas $CC_{i,k}$ y $Cc_{j,k}$. En particular, x_i es un elemento tanto de $CC_{i,j}$ como de $CC_{i,k}$.
- **Caso 3B2B3A: $CC_{i,j} \cap CC_{i,k} \neq \{x_i\}$**
  - Entonces $\exists u : u \in CC_{i,j} \cap CC_{i,k}$ con $u \neq x_i$
    1) $u \in CC_{i,j} \Rightarrow$ el color de u es i o j.
    2) $u \in CC_{i,k} \Rightarrow$ el color de u es i o k.
  - Entonces [1] y [2] implican que el color de u debe ser i, entonces $u \neq x_j$, x_k y por hipótesis sabemos que $u \neq x_i$. Así que u es un vértice en $CC_{i,j}$ distinto de x_i , x_j y un vértice en $CC_{i,k}$ distinto de x_i , x_k (y tiene el color i).
  - Como $u \in CC_{i,j}$, es de color i y no es x_i , entonces tiene 2 vecinos de color j. Como $u \in CC_{i,k}$, es de color i y no es x_i , entonces tiene 2 vecinos de color k .
  - Tenemos entonces que entre los vecinos de u hay dos vertices con el mismo color y otros dos vertices con el mismo color (distinto al primero). Por lo tanto entre los vecinos de u hay a lo sumo $\Delta - 2$ colores. (entre los cuales estan j y k).
  - Asi que podemos hacer con u lo mismo que hicimos con w en el **caso 2B2B2:** entre los $\Delta - 2$ colores de sus vecinos y el color i de u, suman solo $\Delta - 1$ colores, así que hay un color faltante en esa lista, y ese color es distinto de j, k , i.
  - Si le damos a u ese color, el coloreo que queda es propio, pero ahora x_i queda desconectado de x_j , intercambiamos los colores i, j en la nueva $CC_{i,j}$ y le damos el color i a x.
- **Caso 3B2B3B: $CC_{i,j} \cap CC_{i,k} = \{x_i\}$**
  - Sea v el (único) vecino de x_i en $CC_{i,j}$. Al ser vecino de x_i no puede tener su color, asi que el color de v es j. Además, como $CC_{i,j}$ es un camino pero no es igual a $\{x_i , x_j\}$, tenemos que $v \neq x_j$.
  - Intercambiemos los colores en $CC_{i,k}$. Entonces, el nuevo color de x_i es k. Al intercambiar estos colores, la componente $CC_{i,k}$ no cambia. Pero las otras componentes conexas pueden cambiar. Como el color de x_i es ahora k , entonces ahora podría fijarme en la componente conexa de G que tiene colores j y el nuevo k y que tiene a x_i .
  - Llamemos $CC^*_{k,j}$ a esa componente conexa. De la misma forma, llamaremos $CC^*_{j,k}$ a la componente conexa con los colores j y el nuevo k a la cual pertenece xj. Y llamaremos $CC^*_{j,i}$ a la componente conexa con colores j y el nuevo i que tiene a x_j ; y $CC^*_{i,j}$ a la que tiene a j y el nuevo i y tiene a x_k (porque x_k ahora tiene color i).
  - Si $CC^*_{k,j} \neq CC^*_{j,k}$ o $CC^*_{i,j} \neq CC^*_{j,i}$ o alguna no es un camino, podemos repetir los razonamientos anteriores para intercambiar colores y dejar un color libre para colorear a x
  - Asi que podemos suponer $CC^*_{k,j} = CC^*_{j,k}$, $CC^*_{i,j} = CC^*_{j,i}$ y que ambas son caminos.
  - Ahora bien: Sólo cambiamos colores en $CC_{i,k}$ y estamos suponiendo que $CC_{i,j} \cap CC_{i,k} = \{x_i\}$. Por lo tanto los colores de $CC_{i,j} - \{x_i\}$ **no cambian.** En particular, sigue habiendo un camino de colores i, j entre v y x_j . Es decir, $v \in CC^*_{j,i}$.
  - Como estamos suponiendo $CC^*_{j,i} = CC^*_{i,j}$, este fragmento de $CC^*_{j,i}$ debe extenderse hasta x_k , que es el vértice que ahora tiene el color i.
  - Por otro lado, v sigue siendo un vecino de color j de x_i .Solo que x_i tiene ahora el color k . Así pues, $v \in CC^*_{k,j}$. Pero dijimos que $v \in CC^*_{j,i}$. Entonces $v \in CC^*_{j,i} \cap CC^*_{k,j}$ y v no es x_j.
  - Podemos hacer lo mismo que en el **caso 3B2B3A**, usando v en vez de u.

[Subgrafo generado por W]

Dado $W \subseteq V$ defininimos el **subgrafo generado por W** como aquel subgrafo de G que tiene:

- como conjunto de vértices a *W*
- como lados aquellos lados de *G* cuyos extremos están en *W.*

Se denota por *G[W]*.

Es decir, $\boldsymbol{G[W] = (W, \{xy : x, y \in W, xy \in E(G)\})}$

Algunos de estos subgrafos se llaman **Cadenas de Kempe**

Las cadenas de Kempe tienen una propiedad estructural básica:

Dentro de una cadena de Kempe, dado que los únicos vértices tienen color *i* o *j*, y el coloreo es propio tenemos que:

- Todos los vecinos **en la cadena** de un vértice coloreado con color *i* tendrán que tener el color *j*.
- Todos los vecinos en la cadena de un vértice coloreado con un color *j* tendrán que tener el color *i*.

3-COLOR ES NP-COMPLETO [14]

Demostración

Veremos que *3-SAT* $\leq_P$ *3-COLOR*

Para ello, dada una instancia *B* de *3-SAT* crearemos **polinomialmente** una instancia *A(B) = G* de *3-COLOR* tal que *B* sea satisfacible si y solo si $\chi(G) \leq 3$.

Supongamos que las variables de *B* son $x_1, \ldots, x_n$ y que $B = D_1 \wedge \ldots \wedge D_m$ con las disjunciones $D_j = l_{1,j} \vee \ldots \vee l_{3,j}$. $l_{k,j}$ literales

Construiremos el grafo G dando sus vértices y sus lados.

Los **vértices** son:

- *2n* vértices v_l, uno por cada literal *l* (es decir, por cada variable x_i hay dos vértices v_{xi} y $v_{\bar{xi}}$
- *6m* vértices $\{e_{k,j}\}_{j=1,\ldots,m}^{k=1,2,3} \cup \{a_{k,j}\}_{j=1,\ldots,m}^{k=1,2,3}$ es decir, para cada j = 1, 2, …, m, seis vértices $e_{1,j}$, $e_{2,j}$, $e_{3,j}$, $a_{2,j}$, $a_{2,j}$, $a_{3,j}$
- Dos vértices especiales, *s* y *t*.

Observar que los vértices se construyen en tiempo polinomial, porque simplemente los listamos siguiendo los nombres de las variables, el número de las disjunciones y son *2 + 2n + 6m*

Los **lados** son:

- *3m* lados $(a_{1,j}, a_{2,j})$, $(a_{1,j}, a_{3,j})$, $(a_{3,j}, a_{2,j})$ (para $j = 1, 2, \ldots, m$)
- *3m* lados $(e_{k,j}, a_{k,j})$ con $k = 1, 2, 3$ y $j = 1, 2, \ldots, m$
- *2n* lados (t, v_l) uno por cada literal *l*.
- *n* lados $(v_x, v_{\bar{x}})$ uno por cada variable *x*
- El lado (s, t)
- *3m* lados $(s, e_{k,j})$ con $k = 1, 2, 3$ y $j = 1, 2, \ldots, m$
- *3m* lados $(e_{k,j}, v_{lk,j})$ con $k = 1, 2, 3$ y $j = 1, 2, \ldots, m$

Los hemos construido simplemente leyendo las variables y los literales de cada conjunción, y son *1 + 3n + 12m* así que la construcción de *G* es polinomial.

Como G tiene triángulos, entonces $\chi(G) \geq 3$. Así que $\chi(G) \leq 3$ si y solo si $\chi(G) = 3$. Hay que probar que: ***B* es satisfacible si y solo si $\chi(G) = 3$**

Vamos a necesitar distinguir entre las variables y los asignamientos de valores a las variables.

Un asignamiento de valores, dado que las variables son booleanas, es darle valor 1 o 0 a cada variable. Por lo tanto un asignamiento de valores es un vector de bits en $\{0, 1\}^n$. Así que lo denotaremos por $\boldsymbol{b}\rightarrow$. Y $\boldsymbol{x(b\rightarrow)}$ es el valor que asume la variable *x* en ese asignamiento. Similarmente, denotaremos por $\boldsymbol{l(b\rightarrow)}$ el valor que asume el literal *l*. Y $\boldsymbol{D_j(b\rightarrow)}$ el valor que asume toda la disyunción y $\boldsymbol{B(b\rightarrow)}$ el valor que asume toda la extpresión booleana.

***B* SATisfacible $\Rightarrow \chi(G) = 3$**

Idea:

> Debemos dar un coloreo propio con 3 colores de G, así que debemos colorear todos los vértices de G, uno por uno.
>
> Tenemos que usar que hay un asignamiento de valores a las variables de *B* que la vuelve verdadera. Es decir, existe un $b\rightarrow \in \{0, 1\}^n$ tal que $B(b\rightarrow) = 1$.
>
> Al ir coloreando los vértices, iremos chequeando que el coloreo sea propio. Para ello, chequearemos que en cada lado donde dos vértices ya hayan sido coloreados, los extremos tengan colores distintos. Al chequearlo, diremos que el lado "no crea problemas".

Coloreamos los vértices v_l por medio de $\boldsymbol{c(v_l) = l(b\rightarrow)}$.

- $c(v_{\bar{x}}) = \bar{x}\,(b\rightarrow) = 1 - x(b\rightarrow) \neq x(b\rightarrow) = c(v_x)$ así que $(v_x, v_{\bar{x}})$ no crea problemas.

Coloreamos ahora $\boldsymbol{c(s) = 1}$ y $\mathbf{c(t) = 2}$.

- Como $c(s) \neq c(t)$ entonces (s, t) no crea problemas.
- Como $c(t) = 2$ y $c(v_l) \in \{0,1\}$ entonces (t, v_l) no crea problemas

Coloreamos *a's:* Como $B(b\rightarrow) = 1$ y $B = D_1 \wedge \ldots \wedge D_m$, entonces $D_j\,(b\rightarrow) = 1\ \forall\, j$ y como $D_j = l_{1,j} \vee l_{2,j} \vee l_{3,j}$ entonces para todo j existe al menos un k_j tal que $l_{k\,j\,,j}\,(b\rightarrow) = 1$. (1)

Si hay mas de un tal k_j, elegimos uno sólo. Coloreamos, para cada j: $\boldsymbol{c(a_{kj\,,j}) = 2}$ y para los $\boldsymbol{a_{r\,,j}}$ **con** $\mathbf{r \neq k_j}$, que son los dos que quedan, coloreamos uno de ellos con **1**, y el otro con **0**.

- Esto significa que el triángulo formado por los $a_{k,\,j}$ tiene los tres vértices de distinto color, por lo tanto no crea problemas.

Coloreamos los *e*'s: para caja j $\boldsymbol{c(e_{r,\,j}) = 2}$ para los dos $r \neq k_j$. Y $\mathbf{c(e_{kj,\,j}) = 0.}$

- Como para $r \neq k_j$ tenemos $c(e_{r,\,j}) = 2$ y $c(a_{r,\,j}) \in \{0,1\}$ entonces $(e_{r,\,j}, a_{r,\,j})$ no crea problemas para esos *r.*
- Como $c(e_{kj,\,j}) = 0$ y $c(a_{kj,\,j}) = 2$ entonces $(e_{kj,j}\,, a_{kj,\,j})$ no crea problemas.

Veamos si no hay problemas con los lados que conectan las *e's y a's* al resto del grafo.

- $(s, e_{i,\,j})$ no crea problemas pues $c(s) = 1$ y $c(e_{i,\,j}) \in \{2, 0\}$
- Los lados $(e_{r,\,j}, v_{lr,\,j})$ no generan problemas
  - Para el caso $r \neq k_j$ tenemos $c(e_{r,\,j}) = 2$ y $c(v_{lr,\,j}) \in \{0,1\}$
  - Para el caso $r = k_j$ tenemos $c(v_{lkj,\,j}) = l_{kj,\,j}\,(b\rightarrow) = 1$ por (1), por lo tanto $c(e_{kj,\,j}) = 0$ y $c(v_{lkj,\,j}) = 1$

Hemos terminado de colorear todos los vértices con 3 colores y chequeado que el coloreo es propio. Por lo tanto, hemos completado la implicación $B\ SAT \Rightarrow \chi(G) = 3$.

χ(G) = 3 ⇒ B SATisfacible

Idea: Suponemos entonces que existe un coloreo propio *c* de *G* con tres colores. **A partir de *c*** debemos definir un $b\rightarrow \in \{0, 1\}^n$ tal que $B(b\rightarrow) = 1$.

Definimos $b\rightarrow$ usando el coloreo *c* así: $\boldsymbol{b_i = [c(v_{xi}) = c(s)]}$. Es decir $b_i = 1$ si $c(v_{xi}) = c(s)$ y $b_i = 0$ si $c(v_{xi}) \neq c(s)$

Sea *l* un literal cualquiera tal que $c(v_l) = c(s)$.

- Si *l* es una variable: $\exists\ i\ con\ l = x_i$.
  - Entonces decir $c(v_l) = c(s)$ es lo mismo que decir $c(v_{xi}) = c(s)$.
  - Por la definición de $b\rightarrow$ eso implica que $b_i = 1$.
  - Con lo que $l(b\rightarrow) = x_i\,(b\rightarrow) = b_i = 1$.
- Si *l* es negación de una variable $\exists\ i\ con\ l = x_i$.
  1. Entonces decir $c(v_l) = c(s)$ es lo mismo que $c(v_{\bar{xi}}) = c(s)$.
  2. Como $(v_{xi}\,, v_{\bar{xi}}) \in E$ entonces $c(v_{xi}) \neq c(v_{\bar{xi}})$.
  3. Juntando [1.] y [2.] concluimos que $c(v_{xi}) \neq c(s)$.
  4. El punto [3.] implica que $b_i = 0$.
  5. Entonces $l(b\rightarrow) = x_i\,(b\rightarrow) = 1 - x_i\,(b\rightarrow) = 1 - b_i = 1 - 0 = 1$.

En cualquiera de los casos hemos probado que $c(v_l) = c(s) \Rightarrow l(b\rightarrow) = 1$.

Sea ahora $j \in \{1, 2, ..., m\}$. Como los $a_{k\,,j}$ forman un triángulo, y *c* colorea *G* con

tres colores, entonces los tres colores deben estar representados en ese triángulo.

En particular, existe un *q* tal que $c(a_{q,j}) = c(t)$. Analicemos el color posible del $e_{q,j}$ asociado a $a_{q,j}$.

1. $(e_{q,j}\,, a_{q,j}) \in E \Rightarrow c(e_{q,j}) \neq c(a_{q,j}) = c(t)$.
2. $(e_{q,j}\,, s) \in E \Rightarrow c(e_{q,j}) \neq c(s)$.

Por lo tanto $e_{q,j}$ debe tener el "tercer" color, es decir, el color que no es el color de *s* ni el color de *t*. (Como st ∈ E, entonces necesariamente c(s) 6 = c(t) asi que efectivamente sólo queda un color libre para e q,j)

1. $(v_{lq,j}\,, e_{q,j}) \in E \Rightarrow c(v_{lq,j}) \neq c(e_{q,j})$.
2. $(v_{lq,j}\,, t) \in E \Rightarrow c(v_{lq,j}) \neq c(t)$.

Como $e_{q,j}$ tiene el tercer color, los dos puntos anteriores implican que $v_{lq,j}$ no puede tener ni el tercer color ni el color de *t*.

Pero si $v_{lq,j}$ no tiene ni el color de *t* ni el tercer color, sólo le queda tener el color de *s*: $c(v_{lq,j}) = c(s)$, pues sólo hay 3 colores disponibles. Probamos antes que eso implica que $l_{q,j}\,(b\rightarrow) = 1$. Como $D_j = l_{1,j} \vee \ldots \vee l_{3,j}$. entonces $l_{q,j}\,(b\rightarrow) = 1$ implica que $D_j\,(b\rightarrow) = 1$.

El *j* era cualquiera en $\{1, 2, ..., m\}$, es decir, hemos probado que $D_j\,(b\rightarrow) = 1$ para todo *j*. Y como $B = D_1 \wedge ... \wedge D_m$, lo anterior implica que $B(b\rightarrow) = 1$ así que *B* es satisfacible y hemos concluido la prueba.