# Flujos

Miguel Raggi

Graph Algorithms
Escuela Nacional de Estudios Superiores
UNAM

16 de abril de 2018

# Índice:

# Índice:

# Flujo

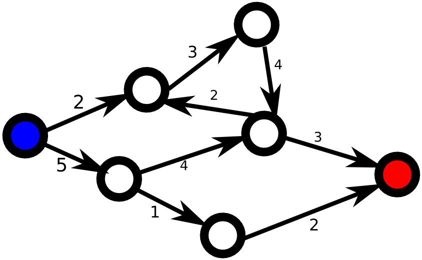
## Definición

*Una red de flujo es una digráfica dirigida $G$ con dos nodos $s, t$ marcados (fuente y pozo), y una función de capacidad $c : A(G) \to \mathbb{R}^+$.*

# Flujo

**Definición**

*Una* *red de flujo* *es una digráfica dirigida* $G$ *con dos nodos* $s, t$ *marcados (fuente y pozo), y una función de capacidad* $c : A(G) \to \mathbb{R}^+$.

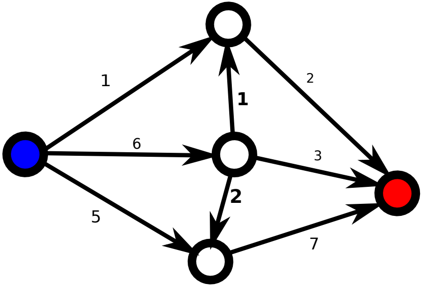


# Flujo

## Definición

*Dada una red de flujo $G$, un flujo es una función $f : A(G) \to \mathbb{R}^+$ tal que $f(a) \leq c(a) \forall a \in A(G)$ y además tenemos que "todo lo que entra, tiene que salir." Es decir, para todo $v \in V(G) \setminus \{s,t\}$ tenemos que:*

$$\sum_{u \in in(v)} f(u,v) = \sum_{u \in ex(v)} f(v,u)$$



# Conversión a problema lineal

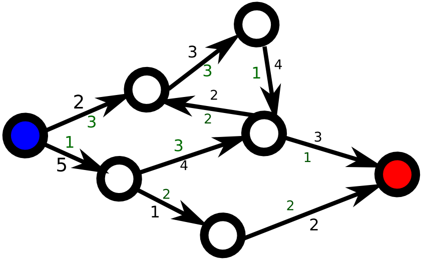
- Sea $G$ una red de flujo. Es decir, $G$ digráfica con dos vértices $s, t$ marcados (fuente y sumidero), y una función de capacidad $c : E(G) \to \mathbb{R}^+$.

# Conversión a problema lineal

- Sea $G$ una red de flujo. Es decir, $G$ digráfica con dos vértices $s, t$ marcados (fuente y sumidero), y una función de capacidad $c : E(G) \to \mathbb{R}^+$.
- Supongamos que cada "unidad" de flujo que enviamos por una arista $e$ tiene costo $a(e)$, donde $a : E(G) \to \mathbb{R}^+$.

# Conversión a problema lineal

- Sea $G$ una red de flujo. Es decir, $G$ digráfica con dos vértices $s, t$ marcados (fuente y sumidero), y una función de capacidad $c : E(G) \to \mathbb{R}^+$.
- Supongamos que cada "unidad" de flujo que enviamos por una arista $e$ tiene costo $a(e)$, donde $a : E(G) \to \mathbb{R}^+$.
- Queremos enviar un flujo $d$ de $s$ en $t$ de manera que se minimice el costo.



# Conversión a problema lineal

Resolver el problema es equivalente al siguiente problema lineal:

$$\text{minimizar } \sum_e a(e) f_e$$

dadas las condiciones

$$f(e) \le c(e) \qquad \forall e \in E$$

$$\sum_{e \in in(v)} f(e) = \sum_{e \in ex(v)} f(e) \qquad \forall v \in V \setminus \{s, t\}$$

$$\sum_{e \in ex(s)} f(e) = d, \qquad \sum_{e \in in(s)} f(e) = 0$$

$$\sum_{e \in ex(t)} f(e) = 0, \qquad \sum_{e \in in(t)} f(e) = d$$

# Teorema de Máximo Flujo-Mínimo Corte

## Definición

*Sea $G$ una red de flujo. Un corte es una partición de los nodos de $G$, $V = S \cup T$ en donde $s \in S$, $t \in T$. Decimos que el valor del corte es:*

$$\sum_{u \in S, v \in T} c(u, v)$$

# Teorema de Máximo Flujo-Mínimo Corte

## Definición

*Sea $G$ una red de flujo. Un corte es una partición de los nodos de $G$, $V = S \cup T$ en donde $s \in S$, $t \in T$. Decimos que el valor del corte es:*

$$\sum_{u \in S, v \in T} c(u, v)$$

## Teorema

*En una red de flujo $G$, el máximo flujo es igual al valor del mínimo corte.*

# Aplicaciones

- Máximo flujo y Mínimo corte tienen muchas aplicaciones prácticas, pero la principal es encontrar “cuellos de botella” para saber en dónde aumentar.

# Aplicaciones

- Máximo flujo y Mínimo corte tienen muchas aplicaciones prácticas, pero la principal es encontrar "cuellos de botella" para saber en dónde aumentar.
- Imaginamos que las aristas son tubos, o carreteras, o lo que sea por lo que pueda pasar un flujo.

# Aplicaciones

- Máximo flujo y Mínimo corte tienen muchas aplicaciones prácticas, pero la principal es encontrar "cuellos de botella" para saber en dónde aumentar.
- Imaginamos que las aristas son tubos, o carreteras, o lo que sea por lo que pueda pasar un flujo.
- El mínimo corte representa en donde debo "ensanchar" para obtener más flujo.

# Aplicaciones

- Máximo flujo y Mínimo corte tienen muchas aplicaciones prácticas, pero la principal es encontrar "cuellos de botella" para saber en dónde aumentar.
- Imaginamos que las aristas son tubos, o carreteras, o lo que sea por lo que pueda pasar un flujo.
- El mínimo corte representa en donde debo "ensanchar" para obtener más flujo.
- Pronto veremos el problema generalizado de flujos: con límites a la capacidad por arriba y por abajo y con costos.

# Aplicaciones

- Máximo flujo y Mínimo corte tienen muchas aplicaciones prácticas, pero la principal es encontrar “cuellos de botella” para saber en dónde aumentar.
- Imaginamos que las aristas son tubos, o carreteras, o lo que sea por lo que pueda pasar un flujo.
- El mínimo corte representa en donde debo “ensanchar” para obtener más flujo.
- Pronto veremos el problema generalizado de flujos: con límites a la capacidad por arriba y por abajo y con costos.
- Muchísimos problemas se pueden transformar al problema generalizado de flujos.

# Programa Lineal

- Podemos ver el problema de máximo flujo como un "programa lineal".

# Programa Lineal

- Podemos ver el problema de máximo flujo como un "programa lineal".
- Sean $c(u, v)$ y $f(u, v)$ la capacidad y el flujo de $u$ a $v$ respectivamente.

# Programa Lineal

- Podemos ver el problema de máximo flujo como un "programa lineal".
- Sean $c(u, v)$ y $f(u, v)$ la capacidad y el flujo de $u$ a $v$ respectivamente.

$$\max \quad F_s$$

dadas las condiciones

$$f(u, v) \leq c(u, v) \qquad \forall u, v \in V$$

$$f(u, v) = -f(v, u) \qquad \forall u, v \in V$$

$$\sum_{u \in V} f(v, u) - \sum_{u \in V} f(u, v) \leq 0 \qquad \forall v \in V \setminus \{s, t\}$$

$$F_s + \sum_{u \in V} f(u, s) - \sum_{u \in V} f(s, u) \leq 0$$

$$F_t + \sum_{u \in V} f(u, t) - \sum_{u \in V} f(t, u) \leq 0$$

$$f(u, v) \geq 0 \qquad \forall u, v \in V$$

# Dual

Encontrar el mínimo corte es el problema dual de encontrar el máximo flujo!

$$\min \quad \sum_{u \in V, v \in V} c(u,v) d(u,v)$$

dado:

$$d(u,v) - p_u + p_v \geq 0$$

$$p_s \geq 1$$

$$p_t \geq 0$$

$$p_v \geq 0 \qquad \forall v \in V \setminus \{s,t\}$$

$$d(u,v) \geq 0$$

# ¿Usar Optimización Lineal?

- Veamos un algoritmo más directo que se llama Ford-Fulkerson.

# ¿Usar Optimización Lineal?

- Veamos un algoritmo más directo que se llama Ford-Fulkerson.
- No es un algoritmo muy bueno, pero es la base para los algoritmos que siguen (y cada vez hay más y mejores).

# ¿Usar Optimización Lineal?

- Veamos un algoritmo más directo que se llama Ford-Fulkerson.
- No es un algoritmo muy bueno, pero es la base para los algoritmos que siguen (y cada vez hay más y mejores).
- Ford-Fulkerson tiene la desventaja que no necesariamente termina.

# ¿Usar Optimización Lineal?

- Veamos un algoritmo más directo que se llama Ford-Fulkerson.
- No es un algoritmo muy bueno, pero es la base para los algoritmos que siguen (y cada vez hay más y mejores).
- Ford-Fulkerson tiene la desventaja que no necesariamente termina.
- Una variante simple, llamada Edmond-Karp, sí termina.

# ¿Usar Optimización Lineal?

- Veamos un algoritmo más directo que se llama Ford-Fulkerson.
- No es un algoritmo muy bueno, pero es la base para los algoritmos que siguen (y cada vez hay más y mejores).
- Ford-Fulkerson tiene la desventaja que no necesariamente termina.
- Una variante simple, llamada Edmond-Karp, sí termina.
- Hay varias variantes más complicada, que no veremos que lo hacen más rápido.

# ¿Usar Optimización Lineal?

- Veamos un algoritmo más directo que se llama Ford-Fulkerson.
- No es un algoritmo muy bueno, pero es la base para los algoritmos que siguen (y cada vez hay más y mejores).
- Ford-Fulkerson tiene la desventaja que no necesariamente termina.
- Una variante simple, llamada Edmond-Karp, sí termina.
- Hay varias variantes más complicada, que no veremos que lo hacen más rápido.
- Finalmente, hay otro algoritmo más rápido que todos estos llamado Empujar-Levantar que veremos (probablemente) la siguiente clase.

# Pre-Algoritmo

El primer algoritmo que a uno se le ocurre es simple:

# Pre-Algoritmo

El primer algoritmo que a uno se le ocurre es simple:

- Empezamos con todos los flujos iguales a 0.

# Pre-Algoritmo

El primer algoritmo que a uno se le ocurre es simple:

- Empezamos con todos los flujos iguales a 0.
- Mientras no hayamos terminado, encontramos un camino de $s$ a $t$ que aún tenga capacidad.

# Pre-Algoritmo

El primer algoritmo que a uno se le ocurre es simple:

- Empezamos con todos los flujos iguales a 0.
- Mientras no hayamos terminado, encontramos un camino de $s$ a $t$ que aún tenga capacidad.
- Enviamos flujo por ese camino.

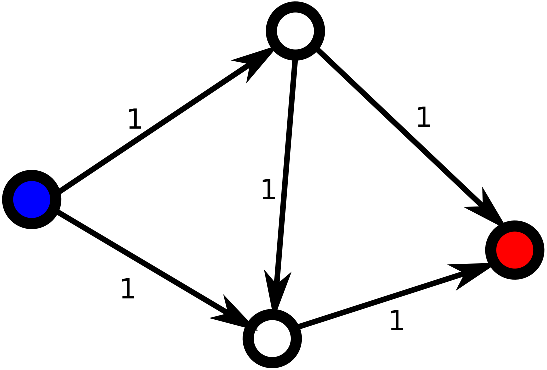
# Pre-Algoritmo

El primer algoritmo que a uno se le ocurre es simple:

- Empezamos con todos los flujos iguales a 0.
- Mientras no hayamos terminado, encontramos un camino de $s$ a $t$ que aún tenga capacidad.
- Enviamos flujo por ese camino.
- Si ya no queda flujo, terminamos.

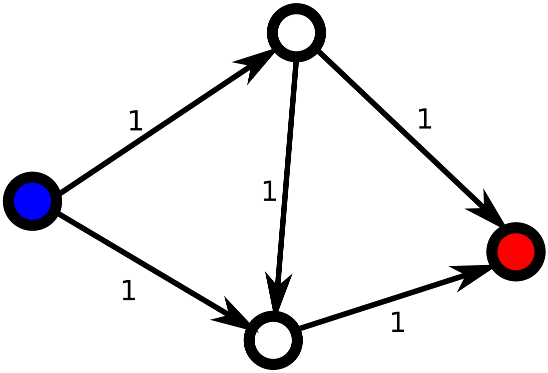
# Problema

¿Qué ocurre aquí?



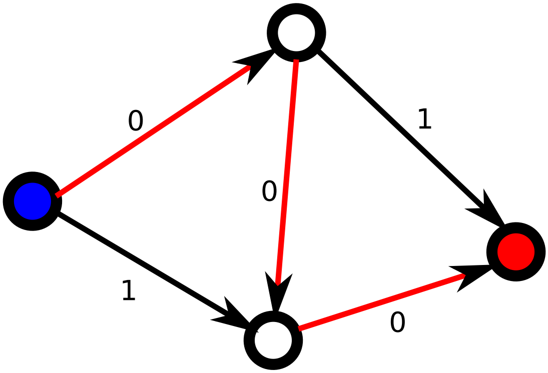
# Problema

¿Qué ocurre aquí?



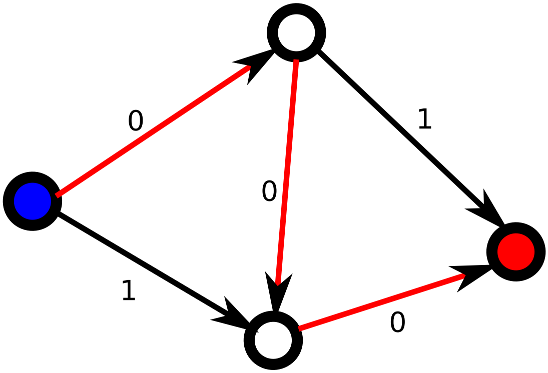
# Problema

Si escojo el camino rojo, ya no puedo mandar más flujo!



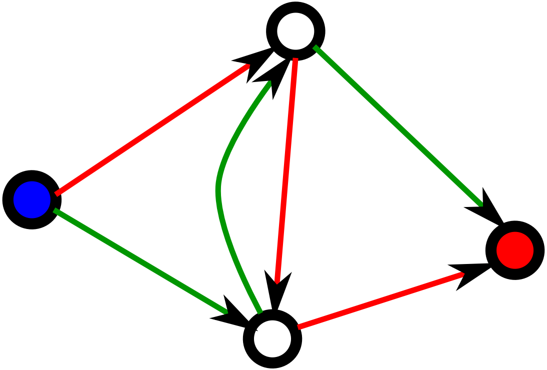
# Problema

Si escojo el camino rojo, ya no puedo mandar más flujo!



# Ford-Fulkerson

Podemos corregir eso: En realidad sí queda un camino, el verde:



# Ford-Fulkerson

- La idea es que cuando agregamos un camino, modificamos las "capacidades" de regreso.

# Ford-Fulkerson

- La idea es que cuando agregamos un camino, modificamos las "capacidades" de regreso.
- A un camino de la fuente al pozo que al enviar flujo por él aumente el flujo total le llamamos camino de aumento.

# Ford-Fulkerson

- La idea es que cuando agregamos un camino, modificamos las “capacidades” de regreso.
- A un camino de la fuente al pozo que al enviar flujo por él aumente el flujo total le llamamos camino de aumento.
- Si ya no existe ningún camino de aumento, debe existir un corte saturado por completo.

# Ford-Fulkerson

- La idea es que cuando agregamos un camino, modificamos las "capacidades" de regreso.
- A un camino de la fuente al pozo que al enviar flujo por él aumente el flujo total le llamamos camino de aumento.
- Si ya no existe ningún camino de aumento, debe existir un corte saturado por completo.
- Así, cuando el algoritmo termina, tenemos el máximo flujo posible.

# Ford-Fulkerson

- La idea es que cuando agregamos un camino, modificamos las "capacidades" de regreso.
- A un camino de la fuente al pozo que al enviar flujo por él aumente el flujo total le llamamos camino de aumento.
- Si ya no existe ningún camino de aumento, debe existir un corte saturado por completo.
- Así, cuando el algoritmo termina, tenemos el máximo flujo posible.
- Enséñales el ejemplo

# Problema con Ford-Fulkerson

- Sabemos que si Ford-Fulkerson termina, entonces encuentra el óptimo.

# Problema con Ford-Fulkerson

- Sabemos que si Ford-Fulkerson termina, entonces encuentra el óptimo.
- Sin embargo, es posible que Ford-Fulkerson no termine!

# Problema con Ford-Fulkerson

- Sabemos que si Ford-Fulkerson termina, entonces encuentra el óptimo.
- Sin embargo, es posible que Ford-Fulkerson no termine!
- Cada vez que encuentro un camino con capacidad, se aumenta el flujo total.

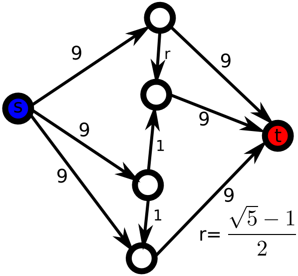
# Problema con Ford-Fulkerson

- Sabemos que si Ford-Fulkerson termina, entonces encuentra el óptimo.
- Sin embargo, es posible que Ford-Fulkerson no termine!
- Cada vez que encuentro un camino con capacidad, se aumenta el flujo total.
- Si por ejemplo todas las capacidades son enteros (o racionales), es claro que sí termina.

# Problema con Ford-Fulkerson

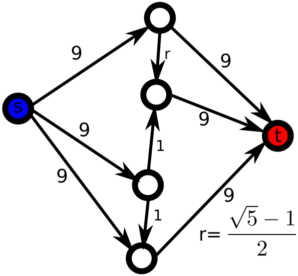
- Sabemos que si Ford-Fulkerson termina, entonces encuentra el óptimo.
- Sin embargo, es posible que Ford-Fulkerson no termine!
- Cada vez que encuentro un camino con capacidad, se aumenta el flujo total.
- Si por ejemplo todas las capacidades son enteros (o racionales), es claro que sí termina.
- Pero con capacidades irracionales puede ser que nunca termine.

# Ejemplo que Ford-Fulkerson no termina

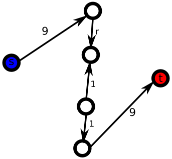


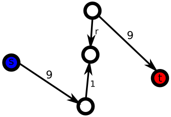

r= $\frac{\sqrt{5}-1}{2}$

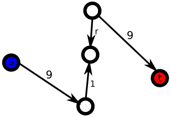
# Ejemplo que Ford-Fulkerson no termina




r= $\dfrac{\sqrt{5}-1}{2}$

$P_1 =$ 


$P_2 =$ 


$P_3 =$ 

# Ejemplo que Ford-Fulkerson no termina

- Claramente el camino máximo es 19.

# Ejemplo que Ford-Fulkerson no termina

- Claramente el camino máximo es 19.
- Primero tomamos el camino que va de $s$ a $t$ naturalmente.

# Ejemplo que Ford-Fulkerson no termina

- Claramente el camino máximo es 19.
- Primero tomamos el camino que va de $s$ a $t$ naturalmente.
- Después tomaremos en sucesión $P_1$, $P_2$, $P_1$, $P_3$,

# Ejemplo que Ford-Fulkerson no termina

- Claramente el camino máximo es 19.
- Primero tomamos el camino que va de $s$ a $t$ naturalmente.
- Después tomaremos en sucesión $P_1$, $P_2$, $P_1$, $P_3$,
- Recordamos que $r^2 = r - 1$ Pizarrón

# Edmond-Karp

- Si siempre usamos búsqueda a lo ancho, el camino de aumento de flujo será lo más corto posible (en el sentido que usará la menor cantidad de aristas posibles).

# Edmond-Karp

- Si siempre usamos búsqueda a lo ancho, el camino de aumento de flujo será lo más corto posible (en el sentido que usará la menor cantidad de aristas posibles).
- A esta variante se le conoce como Edmond-Karp y *siempre* termina:

# Edmond-Karp

- Si siempre usamos búsqueda a lo ancho, el camino de aumento de flujo será lo más corto posible (en el sentido que usará la menor cantidad de aristas posibles).
- A esta variante se le conoce como Edmond-Karp y *siempre* termina:
- Veremos que utiliza a lo más $O(VE^2)$ pasos.

# Edmond-Karp

- Si siempre usamos búsqueda a lo ancho, el camino de aumento de flujo será lo más corto posible (en el sentido que usará la menor cantidad de aristas posibles).
- A esta variante se le conoce como Edmond-Karp y *siempre* termina:
- Veremos que utiliza a lo más $O(VE^2)$ pasos.
- Búsqueda a lo ancho utiliza $O(E)$, y cada vez que encontramos un camino, una arista (de las $E$) se satura y la arista saturada está más lejos que la última saturada cada vez, y que el camino más largo tendrá tamaño a lo más $V$.

# Empujar-Levantar

- Los algoritmos Empujar-Levantar (en su variante “empujar al frente”, que es la que veremos) corre en tiempo $O(V^3)$ (¡bastante mejor que Edmond-Karp!).

# Empujar-Levantar

- Los algoritmos Empujar-Levantar (en su variante "empujar al frente", que es la que veremos) corre en tiempo $O(V^3)$ (¡bastante mejor que Edmond-Karp!).
- Se puede disminuir a $O(VE\log(V^2/E))$ usando estructuras de datos exóticas, pero no veremos como.

# Empujar-Levantar

- Los algoritmos Empujar-Levantar (en su variante “empujar al frente”, que es la que veremos) corre en tiempo $O(V^3)$ (¡bastante mejor que Edmond-Karp!).
- Se puede disminuir a $O(VE\log(V^2/E))$ usando estructuras de datos exóticas, pero no veremos como.
- Haremos dos tipos de operaciones en los nodos:
  - Empujar
  - Levantar

# Empujar-Levantar

- Los algoritmos Empujar-Levantar (en su variante "empujar al frente", que es la que veremos) corre en tiempo $O(V^3)$ (¡bastante mejor que Edmond-Karp!).
- Se puede disminuir a $O(VE\log(V^2/E))$ usando estructuras de datos exóticas, pero no veremos como.
- Haremos dos tipos de operaciones en los nodos:
  - Empujar
  - Levantar
- En la variante que veremos, también haremos la operación "descargar".

# Idea de Empujar-Levantar

- La idea es imaginarse que levantamos a $s$ y dejamos fluir agua.

# Idea de Empujar-Levantar

- La idea es imaginarse que levantamos a $s$ y dejamos fluir agua.
- Nos preguntamos: ¿Qué nodos van en cada "nivel"?

# Idea de Empujar-Levantar

- La idea es imaginarse que levantamos a $s$ y dejamos fluir agua.
- Nos preguntamos: ¿Qué nodos van en cada "nivel"?
- Nos imaginamos al principio a todos los nodos en el suelo (nivel 0) y a $s$ lo levantamos a una altura $V$ (no tiene caso levantarlo aún más).

# Idea de Empujar-Levantar

- La idea es imaginarse que levantamos a $s$ y dejamos fluir agua.
- Nos preguntamos: ¿Qué nodos van en cada "nivel"?
- Nos imaginamos al principio a todos los nodos en el suelo (nivel 0) y a $s$ lo levantamos a una altura $V$ (no tiene caso levantarlo aún más).
- Dejamos "caer" agua de $s$ a todos los que tiene tubería. No nos preocuparemos que haya "fugas" en los nodos.

# Idea de Empujar-Levantar

- La idea es imaginarse que levantamos a $s$ y dejamos fluir agua.
- Nos preguntamos: ¿Qué nodos van en cada "nivel"?
- Nos imaginamos al principio a todos los nodos en el suelo (nivel 0) y a $s$ lo levantamos a una altura $V$ (no tiene caso levantarlo aún más).
- Dejamos "caer" agua de $s$ a todos los que tiene tubería. No nos preocuparemos que haya "fugas" en los nodos.
- Poco a poco iremos levantando otros nodos y mandando el flujo que nos sobra a los nodos de altura menor.

# Qué necesitamos

- Vamos a mantener 3 cosas mientras corre el algoritmo:

# Qué necesitamos

- Vamos a mantener 3 cosas mientras corre el algoritmo:
  - $f(u,v)$: un pre-flujo (flujo donde permitimos fugas). La capacidad restante será $c(u,v) - f(u,v)$

# Qué necesitamos

- Vamos a mantener 3 cosas mientras corre el algoritmo:
  - $f(u, v)$: un pre-flujo (flujo donde permitimos fugas). La capacidad restante será $c(u, v) - f(u, v)$
  - altura$(u)$: será un entero para cada $u \in V$. Sólo "empujaremos" flujo de un nodo de altura mayor a uno de altura menor.

# Qué necesitamos

- Vamos a mantener 3 cosas mientras corre el algoritmo:
  - $f(u, v)$: un pre-flujo (flujo donde permitimos fugas). La capacidad restante será $c(u, v) - f(u, v)$
  - altura$(u)$: será un entero para cada $u \in V$. Sólo "empujaremos" flujo de un nodo de altura mayor a uno de altura menor.
  - exceso$(u)$: lo que le entra de flujo a $u$ pero que ya no le sale, y será siempre un número mayor o igual a 0.

# Qué necesitamos

- Vamos a mantener 3 cosas mientras corre el algoritmo:
  - $f(u,v)$: un pre-flujo (flujo donde permitimos fugas). La capacidad restante será $c(u,v) - f(u,v)$
  - altura$(u)$: será un entero para cada $u \in V$. Sólo "empujaremos" flujo de un nodo de altura mayor a uno de altura menor.
  - exceso$(u)$: lo que le entra de flujo a $u$ pero que ya no le sale, y será siempre un número mayor o igual a 0.
- En cada paso del algoritmo, $f$ será un pre-flujo y tendremos que satisface:

# Qué necesitamos

- Vamos a mantener 3 cosas mientras corre el algoritmo:
  - $f(u, v)$: un pre-flujo (flujo donde permitimos fugas). La capacidad restante será $c(u, v) - f(u, v)$
  - altura$(u)$: será un entero para cada $u \in V$. Sólo "empujaremos" flujo de un nodo de altura mayor a uno de altura menor.
  - exceso$(u)$: lo que le entra de flujo a $u$ pero que ya no le sale, y será siempre un número mayor o igual a 0.
- En cada paso del algoritmo, $f$ será un pre-flujo y tendremos que satisface:
  - $f(u, v) \leq c(u, v)$
  - $f(u, v) = -f(v, u)$
  - $\sum_v f(u, v) = \text{exceso}(u) \geq 0 \quad \forall u \in V \setminus \{s\}$

# Operaciones operaciones

**Definición**

*Una arista* $(u, v)$ *es* *admisible* *si* $h(u) = h(v) + 1$.

# Operaciones operaciones

**Definición**

*Una arista* $(u, v)$ *es admisible si* $h(u) = h(v) + 1$.

Vamos a considerar 2 operaciones:

1. Levantar: Aumentar la altura de un nodo para conseguir una arista admisible con capacidad.
2. Empujar: Aumentar el flujo en una arista admisible.

# Levantar

Levantar significa aumentar la altura de un nodo $u$. Para hacerlo queremos que:

# Levantar

Levantar significa aumentar la altura de un nodo $u$. Para hacerlo queremos que:

- exceso$(u) > 0$: Si no, ¿para qué levantarlo?

# Levantar

Levantar significa aumentar la altura de un nodo $u$. Para hacerlo queremos que:

- exceso$(u) > 0$: Si no, ¿para qué levantarlo?
- altura$(u) \leq$ altura$(v)$ para todos los $v$ tales que $c(u, v) - f(u, v) > 0$. Es decir, sólo levantaremos la altura de uno que ya no podamos empujar flujo a uno de altura menor.

# Levantar

Levantar significa aumentar la altura de un nodo $u$. Para hacerlo queremos que:

- exceso$(u) > 0$: Si no, ¿para qué levantarlo?
- altura$(u) \leq$ altura$(v)$ para todos los $v$ tales que $c(u,v) - f(u,v) > 0$. Es decir, sólo levantaremos la altura de uno que ya no podamos empujar flujo a uno de altura menor.

Cuando levantamos a $u$, ponemos altura$(u)$ como el mínimo valor $h$ tal que existe un $v$ con $h >$ altura$(v)$ y $c(u,v) - f(u,v) > 0$.

# Empujar

Empujar significa enviar parte del exceso de un nodo $u$ a uno de menor altura $v$ (con $h(u) = h(v) + 1$). Queremos hacerlo cuando:

# Empujar

Empujar significa enviar parte del exceso de un nodo $u$ a uno de menor altura $v$ (con $h(u) = h(v) + 1$). Queremos hacerlo cuando:

- altura$(u) >$ altura$(v)$: Sólo echamos flujo a los de altura menor.

# Empujar

Empujar significa enviar parte del exceso de un nodo $u$ a uno de menor altura $v$ (con $h(u) = h(v) + 1$). Queremos hacerlo cuando:

- altura$(u) >$ altura$(v)$: Sólo echamos flujo a los de altura menor.
- exceso$(u) > 0$: Tenemos flujo disponible en el nodo $u$.

# Empujar

Empujar significa enviar parte del exceso de un nodo $u$ a uno de menor altura $v$ (con $h(u) = h(v) + 1$). Queremos hacerlo cuando:

- altura$(u) >$ altura$(v)$: Sólo echamos flujo a los de altura menor.
- exceso$(u) > 0$: Tenemos flujo disponible en el nodo $u$.
- $c(u, v) - f(u, v) > 0$: Debemos poder mandar flujo de $u$ a $v$.

# Empujar

Empujar significa enviar parte del exceso de un nodo $u$ a uno de menor altura $v$ (con $h(u) = h(v) + 1$). Queremos hacerlo cuando:

- altura$(u) >$ altura$(v)$: Sólo echamos flujo a los de altura menor.
- exceso$(u) > 0$: Tenemos flujo disponible en el nodo $u$.
- $c(u, v) - f(u, v) > 0$: Debemos poder mandar flujo de $u$ a $v$.

En caso de que se cumplan esas 3 condiciones, podemos aumentar el flujo $f(u, v)$ por

$$\text{mín}(\text{exceso}(u), c(u, v) - f(u, v))$$

# Descargar

¿Cómo seleccionar qué operación hacer?

# Descargar

¿Cómo seleccionar qué operación hacer? Consideraremos otra operación:

# Descargar

¿Cómo seleccionar qué operación hacer? Consideraremos otra operación: Descargar a un nodo $u$ tal que exceso$(u) > 0$ significará que:

# Descargar

¿Cómo seleccionar qué operación hacer? Consideraremos otra operación: Descargar a un nodo $u$ tal que exceso$(u) > 0$ significará que:

- Intentaremos empujar el exceso de flujo de $u$ a algún (ex)vecino con menor altura.

# Descargar

¿Cómo seleccionar qué operación hacer? Consideraremos otra operación: Descargar a un nodo $u$ tal que exceso$(u) > 0$ significará que:

- Intentaremos empujar el exceso de flujo de $u$ a algún (ex)vecino con menor altura.
- Si no podemos, levantaremos a $u$.

# Descargar

¿Cómo seleccionar qué operación hacer? Consideraremos otra operación: Descargar a un nodo $u$ tal que exceso$(u) > 0$ significará que:

- Intentaremos empujar el exceso de flujo de $u$ a algún (ex)vecino con menor altura.
- Si no podemos, levantaremos a $u$.

Usualmente podemos "recordar" a cuales vecinos de $u$ ya intentamos enviar flujo antes y no pudimos porque su capacidad estaba llena, para no intentarlo de nuevo.

# Empujar-Levantar al frente

El algoritmo será el siguiente:

# Empujar-Levantar al frente

El algoritmo será el siguiente:

- altura$(s) = V$ y altura$(u) = 0$ para todos los demás.

# Empujar-Levantar al frente

El algoritmo será el siguiente:

- altura$(s) = V$ y altura$(u) = 0$ para todos los demás.
- Ponemos $f(s,u) = c(s,u)$ para todo $u$ (es decir, enviamos tanto flujo como pueda salir de $s$ a sus vecinos.)

# Empujar-Levantar al frente

El algoritmo será el siguiente:

- altura$(s) = V$ y altura$(u) = 0$ para todos los demás.
- Ponemos $f(s, u) = c(s, u)$ para todo $u$ (es decir, enviamos tanto flujo como pueda salir de $s$ a sus vecinos.)
- Creamos una lista TodosLosNodos con todos los nodos (excepto $s$ y $t$) en algún orden.

# Empujar-Levantar al frente

El algoritmo será el siguiente:

- altura$(s) = V$ y altura$(u) = 0$ para todos los demás.
- Ponemos $f(s, u) = c(s, u)$ para todo $u$ (es decir, enviamos tanto flujo como pueda salir de $s$ a sus vecinos.)
- Creamos una lista TodosLosNodos con todos los nodos (excepto $s$ y $t$) en algún orden.
- Recorremos TodosLosNodos en orden y descargamos a cada nodo $u$.

# Empujar-Levantar al frente

El algoritmo será el siguiente:

- altura$(s) = V$ y altura$(u) = 0$ para todos los demás.
- Ponemos $f(s, u) = c(s, u)$ para todo $u$ (es decir, enviamos tanto flujo como pueda salir de $s$ a sus vecinos.)
- Creamos una lista TodosLosNodos con todos los nodos (excepto $s$ y $t$) en algún orden.
- Recorremos TodosLosNodos en orden y descargamos a cada nodo $u$.
- Si altura$(u)$ cambió en la descarga, ponemos $u$ al frente de la lista y empezamos de nuevo. Ver ejemplo de wikipedia

# Empujar-Levantar al frente

- El algoritmo corre en tiempo $O(V^3)$.

# Empujar-Levantar al frente

- El algoritmo corre en tiempo $O(V^3)$.
- Por cada nodo, en el peor caso debemos intentar a todos los demás nodos $(V^2)$ para cada altura posible del nodo (otros $V$).

# Empujar-Levantar al frente

- El algoritmo corre en tiempo $O(V^3)$.
- Por cada nodo, en el peor caso debemos intentar a todos los demás nodos $(V^2)$ para cada altura posible del nodo (otros $V$).
- Se puede mejorar un poquito, como dijimos, utilizando estructuras de datos exóticas: árboles dinámicos.

# Empujar-Levantar al frente

- El algoritmo corre en tiempo $O(V^3)$.
- Por cada nodo, en el peor caso debemos intentar a todos los demás nodos ($V^2$) para cada altura posible del nodo (otros $V$).
- Se puede mejorar un poquito, como dijimos, utilizando estructuras de datos exóticas: árboles dinámicos.
- También se puede tratar de ordenar la lista de nodos en algún orden bueno. Usualmente los nodos cerca de $s$ es mejor ponerlos antes.

# Empujar-Levantar al frente

- El algoritmo corre en tiempo $O(V^3)$.
- Por cada nodo, en el peor caso debemos intentar a todos los demás nodos ($V^2$) para cada altura posible del nodo (otros $V$).
- Se puede mejorar un poquito, como dijimos, utilizando estructuras de datos exóticas: árboles dinámicos.
- También se puede tratar de ordenar la lista de nodos en algún orden bueno. Usualmente los nodos cerca de $s$ es mejor ponerlos antes.
- Al final de que corre el algoritmo, en alguna altura $h$ tendremos el mínimo corte como los nodos con altura $\geq k$ y aquellos con altura $\leq k - 1$.

# Índice:

# Flujos con Costos y Capacidades

- Sea $G$ una digráfica en donde en cada arista $(u, v)$ tiene asociados 3 números reales: $a(u, v)$, $b(u, v)$, $c(u, v)$, donde:

# Flujos con Costos y Capacidades

- Sea $G$ una digráfica en donde en cada arista $(u, v)$ tiene asociados 3 números reales: $a(u, v)$, $b(u, v)$, $c(u, v)$, donde:
  - $a(u, v)$ representa la cota por abajo del flujo.

# Flujos con Costos y Capacidades

- Sea $G$ una digráfica en donde en cada arista $(u, v)$ tiene asociados 3 números reales: $a(u, v)$, $b(u, v)$, $c(u, v)$, donde:
  - $a(u, v)$ representa la cota por abajo del flujo.
  - $b(u, v)$ representa la cota por arriba del flujo.

# Flujos con Costos y Capacidades

- Sea $G$ una digráfica en donde en cada arista $(u, v)$ tiene asociados 3 números reales: $a(u, v)$, $b(u, v)$, $c(u, v)$, donde:
  - $a(u, v)$ representa la cota por abajo del flujo.
  - $b(u, v)$ representa la cota por arriba del flujo.
  - $c(u, v)$ representa el costo de enviar una unidad de flujo por esa arista.

# Flujos con Costos y Capacidades

- Sea $G$ una digráfica en donde en cada arista $(u, v)$ tiene asociados 3 números reales: $a(u, v)$, $b(u, v)$, $c(u, v)$, donde:
  - $a(u, v)$ representa la cota por abajo del flujo.
  - $b(u, v)$ representa la cota por arriba del flujo.
  - $c(u, v)$ representa el costo de enviar una unidad de flujo por esa arista.
- El objetivo es encontrar una función de flujo $f(u, v)$ con $a(u, v) \leq f(u, v) \leq b(u, v)$ tal que se conserve el flujo:

$$\sum_{u} f(u, v) = \sum_{u} f(v, u) \quad \forall v \in V$$

# Flujos con Costos y Capacidades

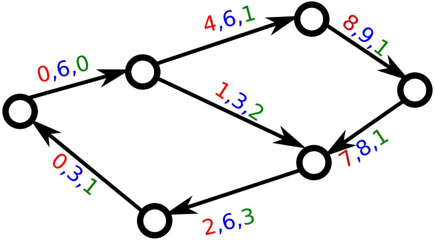
- Sea $G$ una digráfica en donde en cada arista $(u, v)$ tiene asociados 3 números reales: $a(u, v)$, $b(u, v)$, $c(u, v)$, donde:
  - $a(u, v)$ representa la cota por abajo del flujo.
  - $b(u, v)$ representa la cota por arriba del flujo.
  - $c(u, v)$ representa el costo de enviar una unidad de flujo por esa arista.
- El objetivo es encontrar una función de flujo $f(u, v)$ con $a(u, v) \leq f(u, v) \leq b(u, v)$ tal que se conserve el flujo:

$$\sum_{u} f(u, v) = \sum_{u} f(v, u) \quad \forall v \in V$$

- Si existe dicha $f$, queremos encontrar la $f$ que minimice

$$\sum_{u,v} f(u, v) c(u, v)$$

# Ejemplo de Super-Flujo



# Programa Lineal

- Claramente es un programa lineal. Pero observemos que es un programa lineal con mucha estructura:

# Programa Lineal

- Claramente es un programa lineal. Pero observemos que es un programa lineal con mucha estructura:
- Todas las entradas en la matriz son -1, 0 o 1!

# Programa Lineal

- Claramente es un programa lineal. Pero observemos que es un programa lineal con mucha estructura:
- Todas las entradas en la matriz son -1, 0 o 1!
- Para no hacer el cuento largo, resulta que este tipo de problemas se pueden resolver aún más rápido que con programación lineal normal.

# Programa Lineal

- Claramente es un programa lineal. Pero observemos que es un programa lineal con mucha estructura:
- Todas las entradas en la matriz son -1, 0 o 1!
- Para no hacer el cuento largo, resulta que este tipo de problemas se pueden resolver aún más rápido que con programación lineal normal.
- Es tan rápido que incluso muchos resolvedores lineales tratan de encontrar pedazos de programas lineales que se parezcan a esto (sólo -1,0,1 de coeficientes) para acelerarse mucho ahí.

# Programa Lineal

- Claramente es un programa lineal. Pero observemos que es un programa lineal con mucha estructura:
- Todas las entradas en la matriz son -1, 0 o 1!
- Para no hacer el cuento largo, resulta que este tipo de problemas se pueden resolver aún más rápido que con programación lineal normal.
- Es tan rápido que incluso muchos resolvedores lineales tratan de encontrar pedazos de programas lineales que se parezcan a esto (sólo -1,0,1 de coeficientes) para acelerarse mucho ahí.
- Teorema: Si las cotas inferiores, superiores y costos son todos enteros, la solución será entera! (si la matriz es “unimodular”)

# Índice:

# Máximo flujo

Consideremos otra vez el problema de encontrar el máximo flujo en una red con fuente y pozo:

# Máximo flujo

Consideremos otra vez el problema de encontrar el máximo flujo en una red con fuente y pozo:

- Podemos poner una arista "fantasma" de $t$ a $s$ para que se satisfaga que en todos los nodos el flujo que entra es igual al que sale.

# Máximo flujo

Consideremos otra vez el problema de encontrar el máximo flujo en una red con fuente y pozo:

- Podemos poner una arista "fantasma" de $t$ a $s$ para que se satisfaga que en todos los nodos el flujo que entra es igual al que sale.
- $a(u, v) = 0$ para todos $u, v$

# Máximo flujo

Consideremos otra vez el problema de encontrar el máximo flujo en una red con fuente y pozo:

- Podemos poner una arista "fantasma" de $t$ a $s$ para que se satisfaga que en todos los nodos el flujo que entra es igual al que sale.
- $a(u, v) = 0$ para todos $u, v$
- $b(u, v) =$ capacidad para todos $u, v$

# Máximo flujo

Consideremos otra vez el problema de encontrar el máximo flujo en una red con fuente y pozo:

- Podemos poner una arista "fantasma" de $t$ a $s$ para que se satisfaga que en todos los nodos el flujo que entra es igual al que sale.
- $a(u, v) = 0$ para todos $u, v$
- $b(u, v) =$ capacidad para todos $u, v$
- $c(u, v) = 0$ para todos $u, v$ EXCEPTO para la arista fantasma: $c(t, s) = -1$

# Problema del Transporte

- Pensemos que tenemos fábricas, tiendas y costos asociados de llevar producto de cada fábrica a cada tienda.

# Problema del Transporte

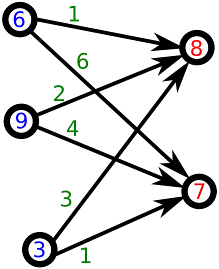
- Pensemos que tenemos fábricas, tiendas y costos asociados de llevar producto de cada fábrica a cada tienda.
- En cada fábrica podemos producir a lo más cierta cantidad de productos.

# Problema del Transporte

- Pensemos que tenemos fábricas, tiendas y costos asociados de llevar producto de cada fábrica a cada tienda.
- En cada fábrica podemos producir a lo más cierta cantidad de productos.
- Además, en cada tienda necesitamos cumplir con cierta demanda de flujo.

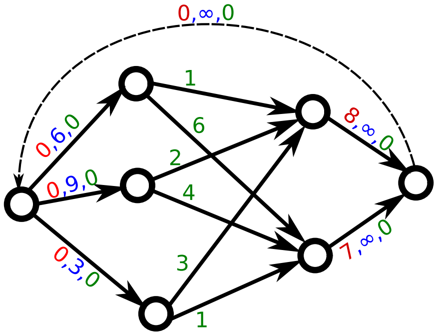
# Problema del Transporte

- Pensemos que tenemos fábricas, tiendas y costos asociados de llevar producto de cada fábrica a cada tienda.
- En cada fábrica podemos producir a lo más cierta cantidad de productos.
- Además, en cada tienda necesitamos cumplir con cierta demanda de flujo.



# Problema del Transporte

Para convertirlo en un problema de flujo, agreguemos dos nodos extra y aristas así:



# Problema de Asignación

- Tenemos personas y tareas y queremos asignar las tareas de manera óptima.

# Problema de Asignación

- Tenemos personas y tareas y queremos asignar las tareas de manera óptima.
- Cada persona tarda cierta cantidad de minutos en hacer cada tarea.

# Problema de Asignación

- Tenemos personas y tareas y queremos asignar las tareas de manera óptima.
- Cada persona tarda cierta cantidad de minutos en hacer cada tarea.
- A cada persona le podemos asignar sólo una tarea.

# Problema de Asignación

- Tenemos personas y tareas y queremos asignar las tareas de manera óptima.
- Cada persona tarda cierta cantidad de minutos en hacer cada tarea.
- A cada persona le podemos asignar sólo una tarea.
- Claramente este es el mismo que el problema de transporte.

# Camino más corto

- El problema de encontrar el camino más corto a un nodo objetivo también es uno de estos!

# Camino más corto

- El problema de encontrar el camino más corto a un nodo objetivo también es uno de estos!
- La gráfica dirigida será la misma.

# Camino más corto

- El problema de encontrar el camino más corto a un nodo objetivo también es uno de estos!
- La gráfica dirigida será la misma.
- Cada arista tendrá de cota inferior 0, cota superior infinito y costo igual al costo de la arista.

# Camino más corto

- El problema de encontrar el camino más corto a un nodo objetivo también es uno de estos!
- La gráfica dirigida será la misma.
- Cada arista tendrá de cota inferior 0, cota superior infinito y costo igual al costo de la arista.
- Crea un nuevo nodo que entra nodo inicial con cota inferior 1, cota superior 1 y costo 0.

# Camino más corto

- El problema de encontrar el camino más corto a un nodo objetivo también es uno de estos!
- La gráfica dirigida será la misma.
- Cada arista tendrá de cota inferior 0, cota superior infinito y costo igual al costo de la arista.
- Crea un nuevo nodo que entra nodo inicial con cota inferior 1, cota superior 1 y costo 0.
- Crea un nuevo nodo que sale del nodo final, con cota inferior 1, cota superior 1 y costo 0.

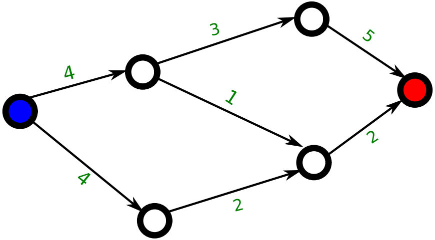
# Camino más corto

- El problema de encontrar el camino más corto a un nodo objetivo también es uno de estos!
- La gráfica dirigida será la misma.
- Cada arista tendrá de cota inferior 0, cota superior infinito y costo igual al costo de la arista.
- Crea un nuevo nodo que entra nodo inicial con cota inferior 1, cota superior 1 y costo 0.
- Crea un nuevo nodo que sale del nodo final, con cota inferior 1, cota superior 1 y costo 0.
- Un arco fantasma del final al inicial con cota inferior 0, superior infinito y costo 0.

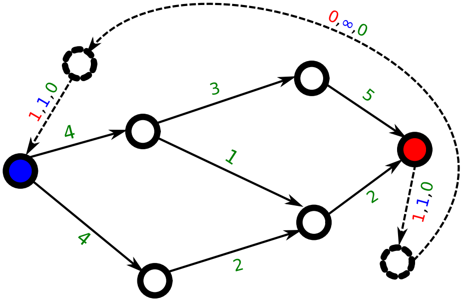
# Camino más corto

- El problema de encontrar el camino más corto a un nodo objetivo también es uno de estos!
- La gráfica dirigida será la misma.
- Cada arista tendrá de cota inferior 0, cota superior infinito y costo igual al costo de la arista.
- Crea un nuevo nodo que entra nodo inicial con cota inferior 1, cota superior 1 y costo 0.
- Crea un nuevo nodo que sale del nodo final, con cota inferior 1, cota superior 1 y costo 0.
- Un arco fantasma del final al inicial con cota inferior 0, superior infinito y costo 0.
- El programa lineal encontrará el camino más corto.

# Camino más corto



# Camino más corto



# Apareamientos Máximos en Gráficas Bipartitas

- Supongamos que $G$ es una gráfica bipartita con partes $X$ y $Y$. Queremos encontrar el apareamiento máximo.

# Apareamientos Máximos en Gráficas Bipartitas

- Supongamos que $G$ es una gráfica bipartita con partes $X$ y $Y$. Queremos encontrar el apareamiento máximo.
- Pondremos todas las aristas dirigidas de $X$ a $Y$.

# Apareamientos Máximos en Gráficas Bipartitas

- Supongamos que $G$ es una gráfica bipartita con partes $X$ y $Y$. Queremos encontrar el apareamiento máximo.
- Pondremos todas las aristas dirigidas de $X$ a $Y$.
- Agregamos dos nodos, $x$ y $y$, donde de $x$ salen todas las aristas a $X$ y de $Y$ salen todas las aristas a $y$.

# Apareamientos Máximos en Gráficas Bipartitas

- Supongamos que $G$ es una gráfica bipartita con partes $X$ y $Y$. Queremos encontrar el apareamiento máximo.
- Pondremos todas las aristas dirigidas de $X$ a $Y$.
- Agregamos dos nodos, $x$ y $y$, donde de $x$ salen todas las aristas a $X$ y de $Y$ salen todas las aristas a $y$.
- Las capacidades inferiores de todas las aristas será 0.

# Apareamientos Máximos en Gráficas Bipartitas

- Supongamos que $G$ es una gráfica bipartita con partes $X$ y $Y$. Queremos encontrar el apareamiento máximo.
- Pondremos todas las aristas dirigidas de $X$ a $Y$.
- Agregamos dos nodos, $x$ y $y$, donde de $x$ salen todas las aristas a $X$ y de $Y$ salen todas las aristas a $y$.
- Las capacidades inferiores de todas las aristas será 0.
- Las capacidades superiores de todas las aristas será 1.

# Apareamientos Máximos en Gráficas Bipartitas

- Supongamos que $G$ es una gráfica bipartita con partes $X$ y $Y$. Queremos encontrar el apareamiento máximo.
- Pondremos todas las aristas dirigidas de $X$ a $Y$.
- Agregamos dos nodos, $x$ y $y$, donde de $x$ salen todas las aristas a $X$ y de $Y$ salen todas las aristas a $y$.
- Las capacidades inferiores de todas las aristas será 0.
- Las capacidades superiores de todas las aristas será 1.
- El costo de todas las aristas será 0.

# Apareamientos Máximos en Gráficas Bipartitas

- Supongamos que $G$ es una gráfica bipartita con partes $X$ y $Y$. Queremos encontrar el apareamiento máximo.
- Pondremos todas las aristas dirigidas de $X$ a $Y$.
- Agregamos dos nodos, $x$ y $y$, donde de $x$ salen todas las aristas a $X$ y de $Y$ salen todas las aristas a $y$.
- Las capacidades inferiores de todas las aristas será 0.
- Las capacidades superiores de todas las aristas será 1.
- El costo de todas las aristas será 0.
- Pondremos arista fantasma de $y$ a $x$ con $(0, \infty, -1)$

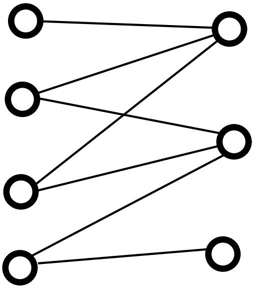
# Apareamientos Máximos en Gráficas Bipartitas

- Supongamos que $G$ es una gráfica bipartita con partes $X$ y $Y$. Queremos encontrar el apareamiento máximo.
- Pondremos todas las aristas dirigidas de $X$ a $Y$.
- Agregamos dos nodos, $x$ y $y$, donde de $x$ salen todas las aristas a $X$ y de $Y$ salen todas las aristas a $y$.
- Las capacidades inferiores de todas las aristas será 0.
- Las capacidades superiores de todas las aristas será 1.
- El costo de todas las aristas será 0.
- Pondremos arista fantasma de $y$ a $x$ con $(0, \infty, -1)$
- Por ser todo entero, la solución será entera.

# Apareamientos Máximos en Gráficas Bipartitas

- Supongamos que $G$ es una gráfica bipartita con partes $X$ y $Y$. Queremos encontrar el apareamiento máximo.
- Pondremos todas las aristas dirigidas de $X$ a $Y$.
- Agregamos dos nodos, $x$ y $y$, donde de $x$ salen todas las aristas a $X$ y de $Y$ salen todas las aristas a $y$.
- Las capacidades inferiores de todas las aristas será 0.
- Las capacidades superiores de todas las aristas será 1.
- El costo de todas las aristas será 0.
- Pondremos arista fantasma de $y$ a $x$ con $(0, \infty, -1)$
- Por ser todo entero, la solución será entera.
- En realidad lo convertimos al problema de máximo flujo.

# Apareamientos Máximos en Gráficas Bipartitas



# Apareamientos Máximos en Gráficas Bipartitas