











# Flows II: Efficient max flow algorithms

# Contenido

1. More algorithms	
2. More flow properties	
3. Edmonds Karp Algorithm	
4. Dinic's Algorithm	

# Contenido





<b>1. More algorithms</b>	
2. More flow properties	
3. Edmonds Karp Algorithm	
4. Dinic's Algorithm	

# Otros algoritmos

- ❑ Aunque Ford-Fulkerson es correcto, la complejidad  $O(EF)$  es muy lenta. Es muy común en los problemas de competencias en tener capacidades altas  $\geq 10^9$ , por lo tanto sería bueno tener algoritmos cuya complejidad sea independientes del flujo máximo.

Algoritmo	Complejidad
Ford Fulkerson	$O(EF)$
Ford Fulkerson - scaling	$O(E^2 \log U)$
Edmonds Karp	$O(E^2 V)$
Dinic	$O(EV^2)$
Dinic - scaling	$O(EV \log U)$
Push Relabel	$O(V^3)$
Compact Networks - Orlin	$O(VE)$

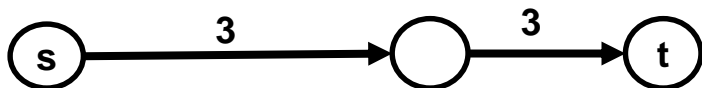
# Contenido

1. More algorithms	
<b>2. More flow properties</b>	
3. Edmonds Karp Algorithm	
4. Dinic's Algorithm	

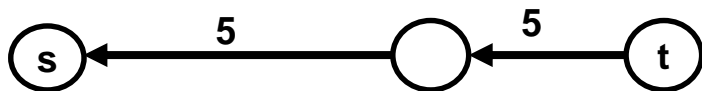
# Flow Decomposition Theorem

Para simplificar las cosas, en las siguientes diapositivas se asumirá que no existen anti-parallel edges.

- **Definición ((s-t) flow path):** Lo definiremos como un path  $P$  que va de  $s$  a  $t$  o de  $t$  a  $s$  tal que en cada arista del path haya una cantidad positiva flujo y que todas las aristas del path tengan la misma cantidad de flujo  $X$ . Denotaremos también el valor del flujo de este path como  $f(P)$  y será igual a  $X$  si el path va de  $s$  a  $t$ ; mientras que será igual a  $-X$  si va de  $t$  a  $s$ .

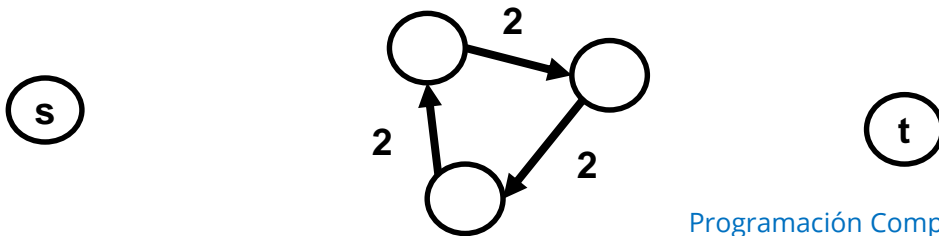


$$f(P) = 3$$



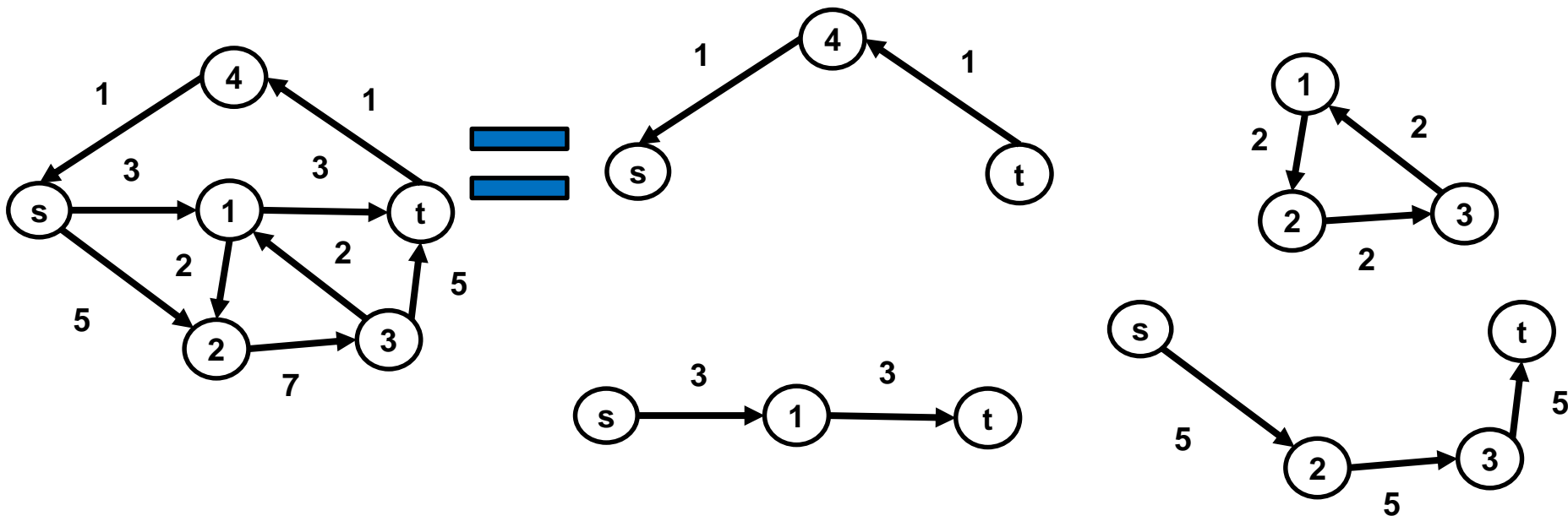
$$f(P) = -5$$

- **Definición (flow cycle):** Lo definiremos como un ciclo  $C$  tal que en cada arista del ciclo haya una cantidad positiva de flujo y que todas las aristas del ciclo tengan la misma cantidad de flujo  $X$ .



# Flow Decomposition Theorem

□ **Theorem:** Todo  $(s - t)$  flow  $f$  puede representarse como una combinación de  $(s - t)$  flow paths  $P_1, P_2, \dots, P_p$  y flow cycles  $C_1, C_2, \dots, C_c$ . Más aún, la cantidad de paths y ciclos es a lo más  $|E|$  y el valor del flujo  $|f|$  es igual a la suma de valores de flujo de cada uno de los paths, es decir  $|f| = \sum_i f(P_i)$

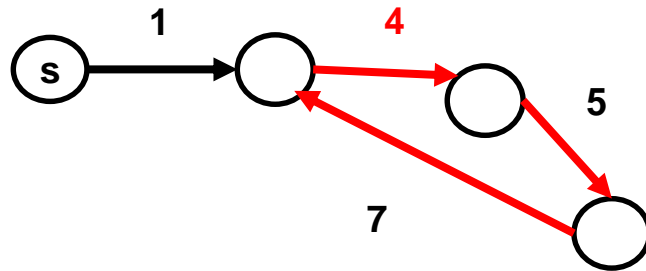
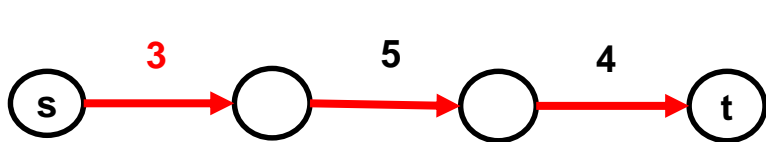


# Flow Decomposition Theorem

□ **Theorem:** Todo  $(s - t)$  flow  $f$  puede representarse como una combinación de  $(s - t)$  flow paths  $P_1, P_2, \dots, P_p$  y flow cycles  $C_1, C_2, \dots, C_c$ . Más aún, la cantidad de paths y ciclos es a lo más  $|E|$  y el valor del flujo  $|f|$  es igual a la suma de valores de flujo de cada uno de los paths, es decir  $|f| = \sum_i f(P_i)$

Incluso, podemos encontrar una posible combinación utilizando el siguiente algoritmo.

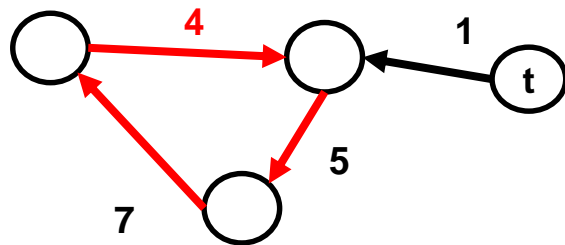
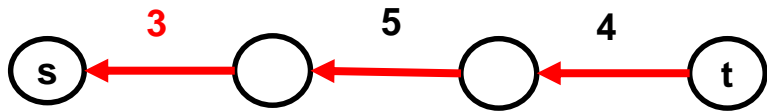
1) Mientras haya aristas salientes de  $s$  con flujo positivo, empiece a construir un *walk* usando las aristas con flujo positivo hasta que visites un nodo repetido o llegues a  $t$ . En caso llegues a  $t$ , habrás encontrado un  $(s - t)$  flow path utilizando el bottleneck de este path como su valor. En caso llegues a un nodo repetido, habrás encontrado algún flow cycle en alguno de los “sufijos” de este walk. Disminuya el bottleneck en este path o cycle.



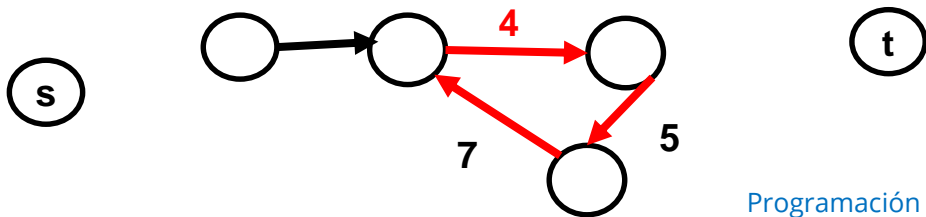


# Flow Decomposition Theorem

2) Mientras haya aristas salientes de  $t$  con flujo positivo, empiece a construir un *walk* usando las aristas con flujo positivo hasta que visites un nodo repetido o llegues a  $s$ . En caso llegues a  $s$ , habrás encontrado un  $(s - t)$  *flow path* utilizando el bottleneck de este path como su valor (en negativo). En caso llegues a un nodo repetido, habrás encontrado algún *flow cycle* en alguno de los “sufijos” de este *walk*. Disminuya el bottleneck en este path o cycle.



3) Mientras haya aristas con flujo positivo, escoja cualquier nodo  $u$  que tenga una arista con flujo positivo saliente y empiece a construir un *walk* recorriendo las aristas con flujo positivo a partir de ahí hasta hallar un nodo repetido que formará un *flow cycle*. Disminuya el bottleneck en este cycle.



# Flow Decomposition Theorem

□ **Theorem:** Todo  $(s - t)$  flow  $f$  puede representarse como una combinación de  $(s - t)$  flow paths  $P_1, P_2, \dots, P_p$  y flow cycles  $C_1, C_2, \dots, C_c$ . Más aún, la cantidad de paths y ciclos es a lo más  $|E|$  y el valor del flujo  $|f|$  es igual a la suma de valores de flujo de cada uno de los paths, es decir  $|f| = \sum_i f(P_i)$

## Demostración:

Demostremos primero que el algoritmo anterior es correcto. Solo debemos demostrar que nunca nos quedamos estancados.

- En cualquiera de los 3 pasos es imposible que te quedes estancado en algún nodo  $u$  distinto de  $s$  o  $t$  mientras haces la búsqueda. Esto es debido a que, de lo contrario, significaría que este nodo solo tiene flujo entrante y violaría el *flow conservation*.
- Si en el paso 1 o 2 llegas a  $s$  o  $t$ , tampoco te quedas estancado porque habrás encontrado en ese momento un ciclo o un  $(s - t)$  path.
- Finalmente, después del paso 2, tenemos que ni  $s$  ni  $t$  tienen aristas salientes con flujo positivo, por lo que su flujo neto no positivo,  $net(s) \leq 0$  y  $net(t) \leq 0$ . Pero, debido a la propiedad de  $net(s) = -net(t)$  solo podemos concluir que  $net(s) = net(t) = 0$ . Por lo tanto en este punto, todos los nodos tienen flujo neto 0. Por lo tanto, aún si llegas a  $s$  o  $t$ , no te podrás quedar debido a que esto violaría su flujo neto 0.

Esto demuestra que el algoritmo encuentra una representación válida de combinación de flow paths y flow cycles.

# Flow Decomposition Theorem

□ **Theorem:** Todo  $(s - t)$  flow  $f$  puede representarse como una combinación de  $(s - t)$  flow paths  $P_1, P_2, \dots, P_p$  y flow cycles  $C_1, C_2, \dots, C_c$ . Más aún, la cantidad de paths y ciclos es a lo más  $|E|$  y el valor del flujo  $|f|$  es igual a la suma de valores de flujo de cada uno de los paths, es decir  $|f| = \sum_i f(P_i)$

## Demostración:

En cada iteración estamos disminuyendo un path o ciclo en su bottleneck. Esto hará que al menos una arista tenga flujo 0 después de esta operación. Como solo disminuimos flujo en todo el algoritmo, el número de aristas con flujo positivo siempre disminuye en al menos 1 en cada iteración, por lo tanto habrá a lo más  $|E|$  iteraciones.

Finalmente el valor del flujo original  $|f|$  es el valor del flujo neto de  $s$  que está compuesto por la suma de flujo de las aristas salientes menos las entrantes.

- Los ciclos que no contienen a  $s$  no tienen ninguna arista que afecte a su flujo neto inicial de  $s$ .
- Los ciclos que contienen a  $s$  tienen una arista saliente de  $s$  y una entrante con el mismo flujo. Ambos valores se anulan en la fórmula por lo que este ciclo tampoco tiene efecto en el flujo neto inicial de  $s$ .
- Los paths de  $s$  a  $t$  contienen solo a una arista saliente de  $s$  por lo que contribuyen en  $f(P_i)$  en el flujo neto inicial de  $s$
- Los paths de  $t$  a  $s$  contienen solo una arista entrante de  $s$ , por lo que contribuyen en  $f(P_i)$  en el flujo neto inicial de  $s$ . ■

# Flow Decomposition Theorem

- ❑ **Definición (Circulation):** Se dice que un flujo  $f$  es una circulación si el *flow conservation* se cumple en todos los nodos. Es decir, es un flujo con  $|f| = 0$ .
- ❑ **Corollary (Circulation decomposition):** Un *circulation* puede ser descompuesto en un conjunto de *flow cycles*, sin  $(s - t)$  *flow paths*.
- ❑ **Corollary (Acyclic flow decomposition):** Un flujo acíclico solo puede ser descompuesto en un conjunto de  $(s - t)$  *flow paths*, sin *flow cycles*.

# Flow difference

Para simplificar las cosas asumamos por ahora que no existen anti parallel edges en los flujos.

□ **Definición:** Sean  $f$  y  $f'$  funciones de flujo válidas en  $G$ . Denotaremos la diferencia de flujo como  $f' - f$  tal que

$$(f' - f)(u, v) = \begin{cases} f'(u, v) - f(u, v), & \text{if } f'(u, v) \geq f(u, v) \\ f(v, u) - f'(v, u), & \text{if } f(v, u) > f'(v, u) \\ 0, & \text{otherwise} \end{cases}$$

□ **Lemma 1:**  $f' - f$  es una función de flujo válida en  $G_f$  (en el residual graph, no en el original)

**Demostración:**

Sea  $D = f' - f$

➤ **Capacity constraint**

- Si  $f'(u, v) \geq f(u, v)$  entonces  $D(u, v) = f'(u, v) - f(u, v) \geq 0$ . Además como  $f'(u, v) \leq c(u, v)$ , entonces  $D(u, v) \leq c(u, v) - f(u, v) = c_f(u, v)$
- Si  $f(v, u) > f'(v, u)$  entonces  $D(u, v) = f(v, u) - f'(v, u) \geq 0$ . Además como  $f'(v, u) \geq 0$ , entonces  $D(u, v) \leq f(v, u) = c_f(u, v)$

# Flow difference

□ **Lemma 1:**  $f' - f$  es una función de flujo válida en  $G_f$

**Demostración:**

Sea  $D = f' - f$

➤ **Flow conservation**

$$\begin{aligned} net_{f'}(u) &= \sum_{out} f'(u, out) - \sum_{in} f'(in, u) \\ net_f(u) &= \sum_{out} f(u, out) - \sum_{in} f(in, u) \end{aligned}$$

Si analizamos arista por arista, en el caso de las aristas de salida, tenemos 2 casos:

- i) Si  $f'(u, out) \geq f(u, out)$  entonces su contribución al flujo neto es de  $f'(u, out) - f(u, out)$
- ii) Si  $f'(u, out) < f(u, out)$ , se tendrá  $f(u, out) - f'(u, out)$  pero en el sentido inverso, por lo que su contribución sigue siendo  $f'(u, out) - f(u, out)$

Se puede demostrar algo similar para las aristas de entrada con su contribución  $f(in, u) - f'(in, u)$

$$\Rightarrow net_D(u) = \sum_{out} (f'(u, out) - f(u, out)) - \sum_{in} (f'(in, u) - f(in, u)) = net_{f'}(u) - net_f(u)$$

Entonces  $\forall u \in V - \{s, t\}$ ,  $net_D(u) = 0$



# Flow difference

□ **Lemma 2:**  $|f' - f| = |f'| - |f|$

**Demostración:**

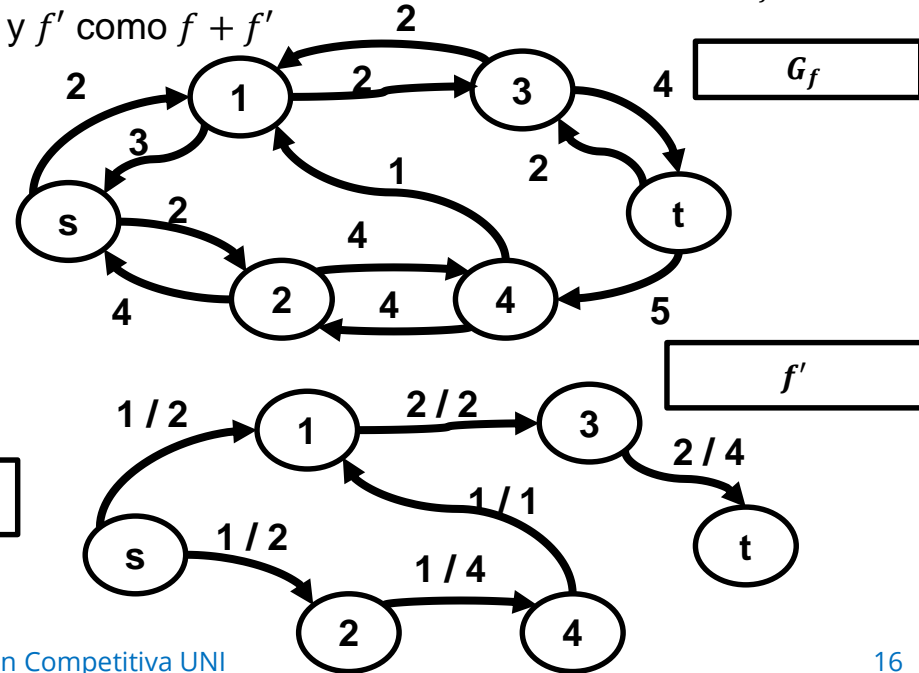
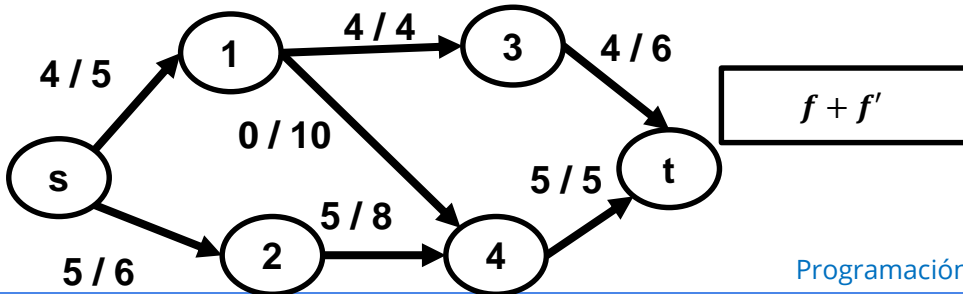
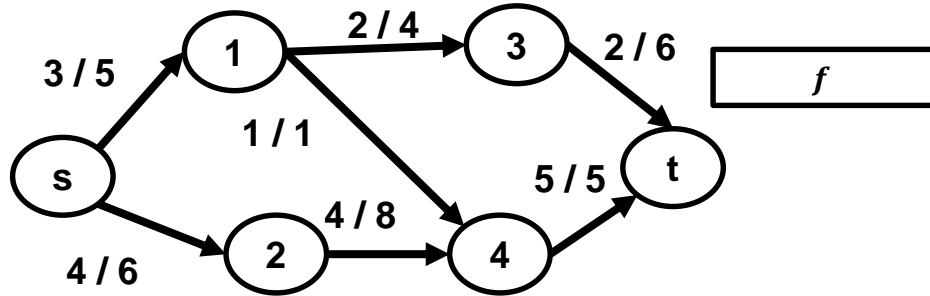
Sea  $D = f' - f$

En la demostración anterior obtuvimos que  $net_D(u) = net_{f'}(u) - net_f(u)$

Entonces  $net_D(s) = net_{f'}(s) - net_f(s) = |f'| - |f|$  ■

# General Flow augmentation

- Anteriormente habíamos definido un **augmentation** como la operación de actualizar el flujo a través de un **augmenting path**.
- Sin embargo, podemos generalizar esta operación en cualquier flujo sobre el grafo residual (no solo en paths).
- Definición:** Sea  $f$  una función de flujo en el grafo original  $G$  y sea  $f'$  un flujo en el residual graph  $G_f$ . Denotaremos el general augmentation como entre  $f$  y  $f'$  como  $f + f'$





# General Flow augmentation

□ **Lemma 3:**  $f + f'$  es también una función de flujo válida en  $G$

La demostración es similar a la que hicimos para un augmenting path.

□ **Lemma 4:**  $|f + f'| = |f| + |f'|$

La demostración es similar a la que hicimos para un augmenting path.

□ **Corollary 5:**  $f + (f' - f) = f'$

□ **Lemma 6:** Sea  $f$  una función de flujo en el grafo original  $G$ , sea  $f^*$  el máximo flujo en  $G$  y sea  $f'$  el máximo flujo en el residual graph  $G_f$ . Entonces  $|f + f'| = |f^*|$

**Demostración:**

Como  $f + f'$  es un flujo válido por el lemma 3, entonces  $|f + f'| \leq |f^*| \dots (1)$

Además por el lemma 1, sabemos que  $f^* - f$  es un flujo válido en  $G_f$ , entonces  $|f'| \geq |f^* - f|$

Por el lemma 2, tenemos que  $|f'| \geq |f^*| - |f| \Rightarrow |f| + |f'| \geq |f^*|$

Finalmente por el lemma 4 tenemos que  $|f + f'| \geq |f^*| \dots (2)$

Juntando (1) y (2) tenemos que  $|f + f'| = |f^*|$

■

# Max Flow Uniqueness

- ❑ **Definición:** Se dice que dos funciones de flujo  $f_1$  y  $f_2$  de un grafo  $G$  son distintas si existe al menos una arista  $(u, v)$  que cumpla  $f_1(u, v) \neq f_2(u, v)$
- ❑ **Lemma 7 (Unique max flow):** Un flujo máximo  $f$  es único si y solo si el grafo residual  $G_f$  no tiene ciclos (simples) de tamaño mayor que 2.

## Demostración:

( $\Rightarrow$ )

Demostremos el contra-positivo. Si el  $G_f$  tiene un ciclo  $C$ , cojamos la mínima capacidad residual de ese ciclo y denotémosla como  $c_f(C)$ . Si formamos un flow cycle con que contenga de flujo esa mínima capacidad residual, nosotros podemos hacer un **general augmentation** en ese ciclo, debido a que cumple con *capacity constraint* y *flow conservation*. La función  $f + C$  es distinta de  $f$ , debido a que el ciclo tiene longitud mayor que 2, por lo que cada arista del grafo original  $G$  será afectada por a lo más 1 arista del ciclo. Entonces  $f$  no es único

( $\Leftarrow$ )

Supongamos que  $G_f$  no tiene ciclos (simples) de tamaño mayor que 2. Por contradicción, asumamos que el flujo máximo no es único. Es decir existe otro flujo  $f'$  tal que  $|f'| = |f|$ . Entonces, definamos  $D = f' - f$ . Sabemos que  $|D| = 0$  por el lemma 2, por lo que es un **circulation**. Además, por el lemma 1 sabemos que  $D$  es una función de flujo válida en  $G_f$ . Por el corollary 5,  $f + D = f'$ . Por ser un circulation y por hipótesis,  $D$  solo puede contener ciclos de tamaño 2, pero estos no hacen ningún efecto en el augmentation, por lo tanto  $f = f'$  (contradicción) ■

# Min cut Uniqueness

□ **Definición:** Se dice que dos cortes  $C_1 = (S_1, T_1)$ ,  $C_2 = (S_2, T_2)$  son distintos si  $S_1 \neq S_2$

□ **Lemma 8:** Sea  $f$  cualquier max flow. En todo min cut  $C = (S, T)$  se cumple que, para toda arista  $(u, v)$  del corte ( $u \in S$ ,  $v \in T$ ),  $f(u, v) = c(u, v)$  y  $f(v, u) = 0$ . Es decir  $c_f(u, v) = 0$ .

**Demostración:**

Sabemos que en general  $f(S, T) \leq c(S, T)$ , pero un corte mínimo debe cumplir la igualdad por el MaxFlow-MinCut Theorem. Además como

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

Como  $f(u, v) \leq c(u, v)$ , la única forma de que se llegue a la igualdad del teorema es que  $f(u, v) = c(u, v)$  y  $f(v, u) = 0$  ■

□ **Corolario 9:** Sea  $f$  cualquier max flow. Sea  $R_s$  el conjunto de nodos alcanzables desde  $s$  en el grafo residual  $G_f$  y sea  $R_t$  el conjunto de nodos que pueden alcanzar a  $t$  en el grafo residual. Entonces  $C_s = (R_s, V - R_s)$  y  $C_t = (V - R_t, R_t)$  son min cuts.

# Min cut Uniqueness

□ **Lemma 10:** En todo min cut  $C = (S, T)$  se cumple que,  $R_s \subseteq S \subseteq V - R_t$

**Demostración:**

Por contradicción supongamos que  $R_s \not\subseteq S$  para cierto min cut  $C = (S, T)$  en donde por lo menos hay un vértice  $u \in R_s - \{s\}$ , tal que  $u \notin S$ . Elijamos el  $u$  más cercano a  $s$  (en términos de números de aristas) en  $G_f$ . Entonces existe un path  $s \rightsquigarrow u$  en el grafo residual  $G_f$ . Sea  $x$  el nodo predecesor de  $u$  en ese path. En primer lugar, tenemos que  $x \in R_s$  simplemente porque la fuente puede alcanzarlo. Además, por definición de  $u$  (ser el más cercano a  $s$ ), tenemos que  $x \in S$  y, por lo tanto,  $(x, u)$  es una arista del corte. Sin embargo, como  $u, x \in R_s$ , tenemos que  $c_f(x, u) = 0$ , y por el lemma 8, tenemos que  $C$  no es min cut (contradicción).

De forma similar se puede demostrar por contradicción que  $S \subseteq V - R_t$  ■

□ **Lemma 11 (Unique min cut):** Un grafo  $G$  tendrá min cut único si y solo si  $C_s = C_t$

**Demostración:**





( $\Rightarrow$ )

Por contrapositivo, si  $C_s \neq C_t$  tenemos dos min cuts distintos por lo que no es único.

( $\Leftarrow$ )

Si  $C_s = C_t$ , significa que  $R_s = V - R_t$ , y luego por el lemma 10 tenemos que el único min cut posible es  $C_s = C_t$  ■

# Contenido

1. More algorithms	
2. More flow properties	
<b>3. Edmonds Karp Algorithm</b>	
4. Dinic's Algorithm	

# Edmonds Karp's Algorithm

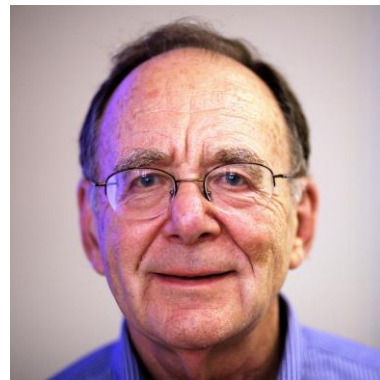
□ Jack Edmonds y Richard Karp publicaron en 1972 dos reglas adicionales para seleccionar un augmenting path en el algoritmo genérico de Ford-Fulkerson.

- 1) **Fattest augmenting path:** Tomar el augmenting path que maximice la capacidad del path (el bottle neck). Este algoritmo es  $O(E^2 \log F \log V)$
- 2) **Shortest path:** Tomar el augmenting path con menos aristas. Este algoritmo es  $O(E^2 V)$

Ambas reglas son correctas puesto que siguen siendo instancias de Ford-Fulkerson. Además para ambos algoritmos también es válido la cota  $O(EF)$



Jack Edmonds



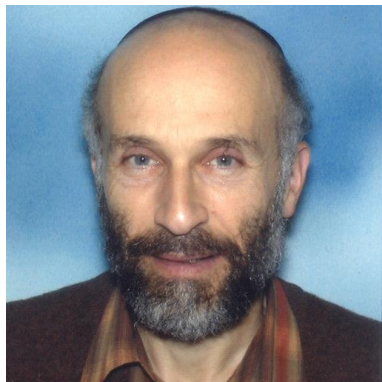
Richard Karp

# Contenido

1. More algorithms	
2. More flow properties	
3. Edmonds Karp Algorithm	
<b>4. Dinic's Algorithm</b>	

# Dinic's Algorithm

- ❑ En 1969 Yefim Dinitz llevaba una clase de algoritmos en la Unión Soviética (Rusia) e intentó resolver uno de los ejercicios dejado por su profesor, de lo cual salió su algoritmo para resolver el problema de max flow [7].
- ❑ En 1970 lo publicó y un personaje llamado Shimon Even estuvo muy interesado en su idea y estuvo muchos días estudiándola. Finalmente llevo la idea hasta occidente (USA) pero le hizo unos ajustes para que sea más entendible para él, y popularizó el algoritmo en sus clases como catedrático y lo llamaba “Dinic's Algorithm”, por una mala pronunciación del nombre de Dinitz [7].



Yefim Dinitz

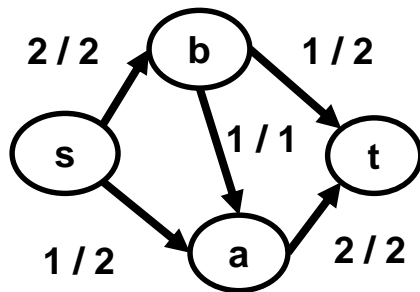


Shimon Even



# Dinic's Algorithm

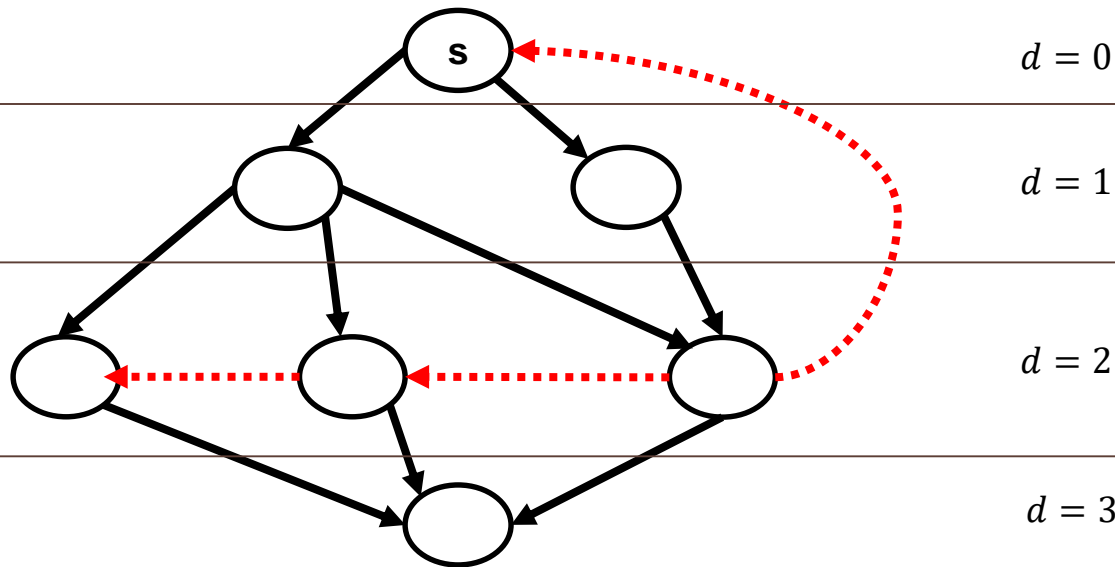
- ❑ El algoritmo de Dinic es parecido a un versión mejorada del algoritmo de Edmonds-Karp utilizando shortest path. Dinic se base en un concepto denominado **blocking flow**.
- ❑ **Definición (Blocking Flow):** El algoritmo que es aquel flujo en donde todos los caminos  $s - t$  tienen al menos una arista **saturada** (con flujo igual a la capacidad).
- ❑ Todo max flow es evidentemente un blocking flow, pero lo contrario no siempre se cumple. Sin embargo es más fácil hallar un blocking flow que un max flow.



Ejemplo de blocking flow

# Dinic's Algorithm

- ❑ El algoritmo de Dinic también hará uso del **level graph**.
- ❑ **Definición (level graph):** Para un grafo  $G(V, E)$ , sea  $d(u)$  el mínimo número de aristas que hay desde  $s$  hasta  $u$  (distancia), o  $+\infty$  en caso no exista un camino.  
El level graph es  $G^L = (V, E^L)$  con  $E^L = \{(u, v) \in E \mid d(v) = d(u) + 1\}$
- ❑ **Observación:** El **level graph** es un DAG



# Dinic's Algorithm

---

**Algorithm 4:** Dinic's algorithm

---

**input** :  $G(V, E, c)$  , *source* :  $s$ , *sink* :  $t$

**output:** max flow:  $f$

1 Initialize flow  $f$  to 0

2 **while** *there exists a path  $P$  in the residual graph  $G_f$  from  $s$  to  $t$*  **do**

3     Use BFS to build the level graph  $G^L$  from the residual network  $G_f$ , and use  $c_f(u, v)$  for the capacities

4     Find any blocking flow  $f'$  in  $G^L$

5      $f = f + f'$

6 **end**

7 **return**  $f$

---

# Dinic's Algorithm

- ❑ Es fácil demostrar la correctitud del algoritmo. Si el algoritmo termina, el resultado será un max flow, simplemente porque en ese momento no habrá ningún *augmenting path* por condición del bucle *while*.
- ❑ La complejidad dependerá de 2 factores
  - ❑ El número de fases en donde se construye el *level graph*
  - ❑ La complejidad para hallar un *blocking flow* en el *level graph*
- ❑ Nota que la condición del *while* para saber si existe un augmenting path, podemos también obtenerla del BFS de la última iteración.

Denotaremos cada iteración del algoritmo como fase. La fase inicial será la fase 0.

Las distancias al iniciar la fase  $i$  serán denotadas como  $d_i(u)$

El flujo al iniciar la fase  $i$  será denotado como  $f_i$ .

El grafo residual al iniciar la fase  $i$  será denotado como  $G_{f_i}$

# Dinic's Algorithm

□ **Lemma 13:** Para toda fase  $i > 0$ ,  $\forall u \in V$ ,  $d_i(u) \leq d_{i+1}(u)$

**Demostración:**

Para cada arista, al pasar de una fase a otra, la arista puede seguir presente en el siguiente grafo residual, como también puede aparecer su arista reversa.

Entonces, demostremos con inducción en la distancia.

- **Caso base:**  $D = 0$ . Se cumple trivialmente ya que  $d_i(s) = d_{i+1}(s) = 0$
- **Paso inductivo:**

Supongamos que para todos los nodos  $u$  que están a una distancia de  $s$  menor o igual que  $D$ , se cumple la hipótesis. Demostremos que también cumple para todos los nodos a una distancia  $D + 1$ .

Sea  $u$  un nodo cualquier en el nivel  $D + 1$ . Sea  $P$  un shortest path desde  $s$  hasta  $u$  en la iteración  $i + 1$  y sea  $q$  el nodo anterior en este shortest path.

# Dinic's Algorithm

□ **Lemma 13:** Para toda fase  $i > 0$ ,  $\forall u \in V$ ,  $d_i(u) \leq d_{i+1}(u)$

**Demostración:**

- **Paso inductivo:**

Entonces, por definición de shortest path,  $d_{i+1}(q) + 1 = d_{i+1}(u) \dots (1)$ .

Luego, por hipótesis inductiva  $d_i(q) \leq d_{i+1}(q) \dots (2)$ . Ahora analicemos la arista  $(q, u)$ , solo hay 2 casos.

❖ **La arista  $(q, u)$  también existe en  $G_{f_i}$ .**

$\Rightarrow d_i(u) \leq d_i(q) + 1$ , por triangle inequality

$\Rightarrow d_i(u) \leq d_{i+1}(q) + 1$ , usando hipótesis inductiva (2)

$\Rightarrow d_i(u) \leq d_{i+1}(u)$ , usando (1)

❖ **La arista  $(q, u)$  no existe en  $G_{f_i}$ .**

Entonces,  $(u, q)$  debe existir en  $G_{f_i}$  y debió haber sido afectada por el blocking flow para generar su reversa, por lo tanto debe ser parte del *level graph*

$\Rightarrow d_i(u) + 1 = d_i(q)$ , por ser parte del level graph

$\Rightarrow d_i(u) + 1 \leq d_{i+1}(q)$ , usando hipótesis inductiva (2)

$\Rightarrow d_i(u) + 1 \leq d_{i+1}(u) - 1$ , usando (1)

$\Rightarrow d_i(u) \leq d_{i+1}(u)$ . ■

# Dinic's Algorithm

□ **Lemma 14:** Para toda fase  $i > 0$ ,  $\forall u \in V$ ,  $d_i(t) < d_{i+1}(t)$

**Demostración:**

Por el lemma 13, tenemos que  $d_i(t) \leq d_{i+1}(t)$ . Por contradicción supongamos que  $d_i(t) = d_{i+1}(t)$ .

Sea  $P(s = v_0, v_1, \dots, v_L = t)$  un shortest path desde  $s$  a  $t$  en  $G_{f_{i+1}}$ .

Por serie telescópica debe cumplirse

$$d_i(t) = \sum_{k=0}^{L-1} (d_i(v_{k+1}) - d_i(v_k)) \dots (1)$$

Analizamos la arista  $(v_k, v_{k+1})$  en la fase  $i$ . Si la arista existe en  $G_{f_i}$ , entonces  $d_i(v_{k+1}) \leq d_i(v_k) + 1$  (*triangle inequality*),  $\Rightarrow d_i(v_{k+1}) - d_i(v_k) \leq 1 \dots (2)$ .

Sin embargo, si la arista no existiese en  $G_{f_i}$ , podemos usar el mismo argumento que en el lemma 13. La arista reversa  $(v_{k+1}, v_k)$  debe existir y ser parte del *level graph*

$$\Rightarrow d_i(v_{k+1}) + 1 = d_i(v_k) \Rightarrow d_i(v_{k+1}) - d_i(v_k) = -1 \dots (3)$$

Para que se cumpla la igualdad en (1), es imposible que suceda (3) ya que esto haría que la telescópica sea menor que  $d_i(t)$ . Por lo tanto, concluimos que todas las aristas deben existir en  $G_{f_i}$ .

Esto significa que  $P$  también es un shortest path en  $G_{f_i}$ ; sin embargo, aquí se produce una contradicción, debido a que luego del blocking flow en la fase  $i$ , ya no debería existir el path  $P$  en  $G_{f_{i+1}}$ .

$\Rightarrow d_i(t) < d_{i+1}(t)$  ■

# Dinic's Algorithm

□ **Corollary 15:** Para un grafo de  $V$  aristas, el número de fases en Dinic es  $O(V)$ . Para ser más exactos, a lo mucho hay  $|V| - 1$  fases

## **Demostración:**

El BFS solo genera distancias enteras.

Como  $s \neq t$  por definición de residual network, entonces  $1 \leq d(t)$

El máximo shortest path posible estaría formado por los  $|V|$  nodos y  $|V| - 1$  aristas, entonces  $d(t) \leq |V| - 1$

Por lo tanto, el máximo número de fases es  $|V| - 1$  ■



# Dinic's Algorithm – Blocking Flows

- ❑ Sabemos que la complejidad de cada fase toma en cuenta la complejidad de los bfs  $O(E)$  + la complejidad de hallar un blocking flow  $O(\text{blocking flow})$ . Usando el **corollary 15**, la complejidad total entre todas las fases sería  $O(EV + V * O(\text{blocking flow}))$ .
- ❑ ¿Cómo calculamos un blocking flow en el *level graph*?
- ❑ Una primera estrategia sería usar el *naive greedy flow* de buscar paths que mejoren el flujo hasta que ya no encuentres más. Como aquí ya no se crean aristas reversas y en cada iteración saturas por lo menos una arista, entonces la complejidad sería  $O(E^2)$  dando una complejidad total de  $O(EV + E^2V) = O(E^2V)$

# Dinic's Algorithm – Blocking Flows

- ❑ Podemos mejorar aún más la complejidad para hallar un blocking flow si aprovechamos que el *level graph* es un DAG. Cuando estemos buscando un path en el *dfs*, si estamos analizando *dfs(u)* y estamos revisando la arista  $(u, v)$ , y llamamos a *dfs(v)*, si no llegamos a encontrar ningún path desde *v*, podemos declarar la arista  $(u, v)$  como **inútil** y no volverla a recorrer **durante esta fase**.
- ❑ Esto es debido a que al no haber resultados en *dfs(v)*, y no haber ciclos en el DAG, nos indica que no existe ningún camino desde *v* a *t* y tampoco habrá en el futuro, debido a que no se agregan aristas durante esta fase de hallar un blocking flow, solo disminuyen estas.
- ❑ Para lograr esto, podemos guardar para cada nodo, un puntero que represente la arista que le toca visitar *nextEdge[u]* e ir avanzándola cada vez que se avance en el *dfs*.

```
Cap dfs(int u, int t, Cap f) {
    if (u == t) return f;
    for (int &i = nextEdge[u]; i < adj[u].size(); i++) {
        Edge &e = adj[u][i];
        int v = e.to;
        Edge &rev = adj[v][e.rev];
        Cap cf = e.cap - e.flow;
        if (cf == 0 || level[v] != level[u] + 1) continue;
        Cap ret = dfs(v, t, min(f, cf));
        if (ret > 0) {
            e.flow += ret;
            rev.flow -= ret;
            return ret;
        }
    }
    return 0;
}
```

# Dinic's Algorithm

□ **Lemma 16 (Blocking Flow Complexity):** La complejidad para hallar un blocking flow es  $O(EV)$

**Demostración:**

En cada llamada del dfs sucede lo siguiente

- i) Se encuentra un path  $P$  donde se pueda mandar flow (a excepción de la última iteración)
- ii) Se eliminan algunas aristas **inútiles** (se mueve el puntero).

En el caso (i), el path tiene  $O(V)$  aristas.

Para el caso (ii) supongamos que la  $k$  - th iteración se eliminan  $useless_k$  aristas.

Además, cada dfs satura al menos una arista (a excepción de la última iteración). Por lo tanto, a lo más hay  $|E|$  dfs que saturan aristas y 1 iteración final sin saturar.

Entonces el número total de operaciones es

$$O\left(EV + \sum_{k=1}^{E+1} useless_k\right) = O(EV + E) = O(EV) \blacksquare$$

□ **Corolario 17 (Dinic's algorithm Complexity):** El algoritmo de Dinic es  $O(EV^2)$

# Dinic's Algorithm

■ **Lemma 17 (Dinic's algorithm complexity):** Si  $G$  es un grafo con capacidades enteras y el max flow tiene valor  $F$ , entonces la complejidad de Dinic también es  $O(EV + VF)$

## Demostración:

Por el corolario 15 sabemos que hay  $O(V)$  fases. En cada fase se hace 1 bfs y se busca un blocking flow. La complejidad total de los bfs es  $O(EV)$ .

Para el blocking flow usaremos una fórmula similar al lemma 16. Nuevamente tenemos las dos operaciones principales en un dfs

- i) Se encuentra un path  $P$  donde se pueda mandar flow (a excepción de la última iteración)
- ii) Se eliminan algunas aristas **inútiles** (se mueve el puntero).

El total de complejidad de las operaciones para recorrer aristas inútiles y mover el puntero es  $O(E)$  por fase, que da un total de  $O(EV)$ .

Finalmente, en el caso de las operaciones encontrando paths, son  $O(V)$  y cada vez que se hace, el valor del flujo aumenta por lo menos en 1. Contabilizando todas las fases, da un total de  $O(VF)$

La complejidad total es  $O(EV) + O(EV) + O(VF) = O(EV + VF)$  ■

# Dinic's Algorithm – Unit Capacities

□ **Corollary 18 (Unit capacity complexity – Simple bound):** Si el grafo solo tiene capacidades unitarias, entonces Dinic corre en  $O(EV)$

**Demostración:**

Consecuencia directa del lemma 17, usando  $F = O(E)$

**Observación:** Si el grafo original solo tiene capacidades unitarias, entonces el grafo residual también tendrá capacidades unitarias en todo momento del algoritmo (asumiendo que no hay anti-paralel edges, o que se está usando un multigraph)

□ **Lemma 19:** Si el grafo solo tiene capacidades unitarias, entonces el algoritmo para hallar blocking flow es  $O(E)$

**Demostración:**

En cada llamada del dfs sucede lo siguiente

i) Se encuentra un path  $P$  donde se pueda mandar flow (a excepción de la última iteración). Como todas las capacidades son unitarias, en cada iteración se saturan todas las aristas del path. Entonces en total esta operación suma  $O(E)$  a lo largo de todas las llamadas de dfs.

ii) Se eliminan algunas aristas inútiles (se mueve el puntero). Se mantiene en  $O(E)$

La complejidad para hallar un blocking flow entonces es  $O(E) + O(E) = O(E)$  ■

# Dinic's Algorithm – Unit Capacities

□ **Lemma 20 (Unit capacity complexity – Bound 1):** Si el grafo solo tiene capacidades unitarias, entonces Dinic corre en  $O(E\sqrt{E})$

## Demostración:

Por el **lemma 19** basta con demostrar que el número de fases de Dinic es  $O(\sqrt{E})$ .

Supongamos que ya pasaron  $\lceil \sqrt{|E|} \rceil$  fases y el algoritmo no termina. Por el **lemma 14** sabemos que cada fase aumenta  $d(t)$  en al menos 1. Por lo tanto, luego de  $\lceil \sqrt{|E|} \rceil$  fases, tendremos  $d(t) \geq \sqrt{|E|}$ .

Sea  $f$  el flujo en ese momento y  $G_f$  el grafo residual en ese momento. Por el **lemma 6** sabemos que el max flow  $f'$  en el grafo residual nos dará el valor de flujo faltante. Además, por ser un grafo de capacidades enteras, también es una cota para el número de fases faltantes.

Luego, por el **flow decomposition theorem**, tenemos que este flujo puede ser dividido en  $|f'|$  *flow paths* (y algunos *flow cycles*).

Como  $f'$  solo tiene capacidades unitarias, entonces estos *flow paths* no pueden compartir aristas.

Considerando que hay  $|f'|$  flow paths y utilizando  $d(t) \geq \sqrt{|E|}$ . Sea  $T$  el número de aristas totales en todos estos flow paths, entonces  $\sqrt{|E|} * |f'| \leq T$ . Obviamente también  $T \leq |E|$ . Juntando ambas inecuaciones

$$\sqrt{|E|} * |f'| \leq |E|$$

$$\Rightarrow |f'| \leq \sqrt{|E|}$$

$$\Rightarrow \text{fases} \leq 2\sqrt{|E|} = O(\sqrt{E}) \blacksquare$$

# Dinic's Algorithm – Unit Capacities

■ **Lemma 21 (Unit capacity complexity – Bound 2):** Si el grafo solo tiene capacidades unitarias, entonces Dinic corre en  $O(EV^{2/3})$

## Demostración:

Por el **lemma 19**, basta con demostrar que el número de fases de Dinic es  $O(V^{2/3})$ .

Supongamos que ya pasaron  $\lceil |V|^{2/3} \rceil$  fases sin terminar, por el **lemma 14**, sabemos que  $d(t) \geq |V|^{2/3}$ .

Sea  $G_f$  el grafo residual en ese momento y sea  $A_i = \{u \mid d(u) = i \text{ en } G_f\}$ , entonces existen por lo menos hay  $|V|^{2/3} + 1$  conjuntos  $A_i$ .

Por el **lemma 6** sabemos que el max flow  $f'$  en el grafo residual nos dará el valor de flujo faltante. Además, por ser un grafo de capacidades enteras, también funciona como límite para el número de fases.

Podemos demostrar por contradicción que deben haber al menos dos conjuntos consecutivos  $A_i, A_{i+1}$  tal que  $|A_i|, |A_{i+1}| \leq 2|V|^{1/3}$ , ya que en caso no existieran, se necesitarían por lo menos la mitad de conjuntos con

$|A_x| > 2|V|^{2/3}$  lo que daría hasta ahí un total de nodos mayor que  $2|V|^{1/3} * \frac{(|V|^{2/3}+1)}{2} > |V|$  (contradicción).

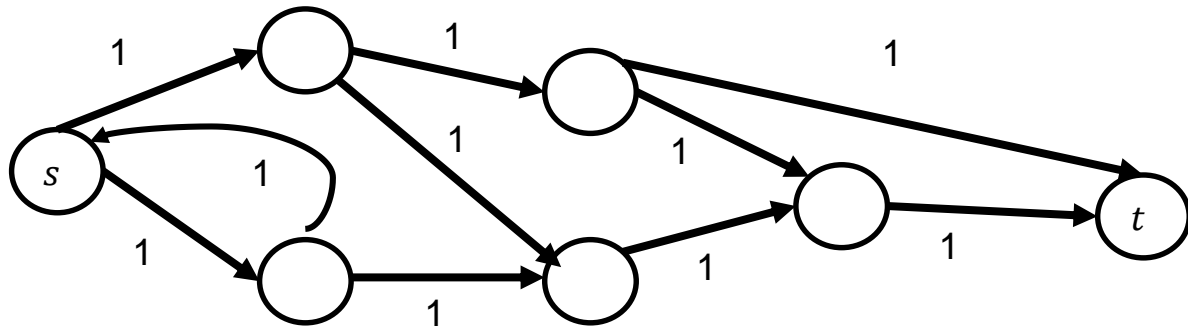
Podemos formar el corte  $(S, T)$  con  $S = A_0 \cup A_1 \cup \dots \cup A_i$ . Por propiedades del level graph, las únicas aristas del corte serían entre  $A_i$  y  $A_{i+1}$ , por lo tanto  $|\text{maxflow}| \leq |\text{mincut}| \leq |A_i| * |A_{i+1}| = 4|V|^{2/3}$ .

Entonces el max flow en  $G_f$  es como máximo  $4|V|^{2/3}$ , por lo tanto al algoritmo le queda ese número de fases como máximo.

$\Rightarrow \text{fases} \leq |V|^{2/3} + 4|V|^{2/3} = O(V^{2/3})$  ■

# Dinic's Algorithm – Unit Network

- **Definición (Unit network):** Es un grafo con capacidades unitarias y además, para todos los nodos  $u$  distintos de  $s, t$ , se cumple que solo tienen una arista de entrada o una arista de salida. Es decir  $\text{indegree}(u) = 1 \vee \text{outdegree}(u) = 1$ .



- **Observación:** Si el grafo  $G$  es un unit network, todos los grafos residuales  $G_f$  también lo serán luego de cada augmentation. Esto es debido a que cualquier flujo  $f'$  sobre el residual con el que se hará el siguiente agumentation, solo puede tener a lo más una arista entrante y una saliente para cada nodo  $u \neq s, t$ . Entonces el número de aristas entrantes y salientes para cada nodo  $u \neq s, t$  no va a cambiar.



# Dinic's Algorithm – Unit Network

□ **Lemma 22 (Unit network complexity):** Si el grafo es un unit network, entonces Dinic corre en  $O(E\sqrt{V})$ .

**Demostración:**

Por el **lemma 19** basta con demostrar que el número de fases de Dinic es  $O(\sqrt{V})$ .

Supongamos que ya pasaron  $\lceil \sqrt{|V|} \rceil$  fases y el algoritmo no termina. Por el **lemma 14** sabemos que cada fase aumenta  $d(t)$  en al menos 1. Por lo tanto, luego de  $\lceil \sqrt{|V|} \rceil$  fases, tendremos  $d(t) \geq \sqrt{|V|}$ .

Sea  $f$  el flujo en ese momento y  $G_f$  el grafo residual en ese momento. Por el **lemma 6** sabemos que el max flow  $f'$  en el grafo residual es una cota para el número de fases faltantes.

Luego, por el **flow decomposition theorem**, tenemos que este flujo puede ser dividido en  $|f'|$  *flow paths* (y algunos *flow cycles*).

Como  $f'$  solo tiene capacidades unitarias y **a lo más hay una arista de entrada o de salida**, entonces estos *flow paths* no pueden compartir nodos. Considerando que hay  $|f'|$  flow paths y utilizando  $d(t) \geq \sqrt{|V|}$ . Sea  $T$  el número de nodos totales en todos estos flow paths, entonces  $\sqrt{|V|} * |f'| \leq T$ . Obviamente también  $T \leq |V|$ .

Juntando ambas inecuaciones  $\sqrt{V} * |f'| \leq V \Rightarrow |f'| \leq \sqrt{V}$ .

$\Rightarrow$  fases  $\leq 2\sqrt{V} = O(\sqrt{V})$  ■

# Resumen complejidad

Algoritmo	Complejidad General			Unit capacity		Unit network
Ford Fulkerson	$O(EF)$			$O(EV)$		$O(EV)$
Dinic	$O(EV^2)$	$O(EV + VF)$	$O(EF)$	$O(E\sqrt{E})$	$O(EV^{2/3})$	$O(E\sqrt{V})$
Edmonds Karp	$O(E^2V)$	$O(EF)$		$O(EV)$		$O(EV)$

# Referencias

- ❑ [1] Jeff Erickson. Algorithms. Chapter 10: Maximum Flows and Minimum cuts.  
<https://jeffe.cs.illinois.edu/teaching/algorithms/book/10-maxflow.pdf>
- ❑ [2] cp-algorithms. Maximum Flow - Dinic. <https://cp-algorithms.com/graph/dinic.html>.
- ❑ [3] David Witner. Lecture 19: Max Flow II: Edmonds-Karp.  
<http://www.cs.cmu.edu/afs/cs/academic/class/15750-s17/ScribeNotes/lecture19.pdf>
- ❑ [4] <http://theory.stanford.edu/~trevisan/cs261/all-notes-2010.pdf>
- ❑ [5] Necessary and sufficient condition for determining unique max flow.  
<https://math.stackexchange.com/questions/4098777/necessary-and-sufficient-condition-for-determining-unique-max-flow>
- ❑ [6] Bocconi University. Problem set 2 Solutions. <https://lucatrevisan.github.io/30516/pset2-sol.pdf>
- ❑ [7] Dinitz's Algorithm: The Original Version and Even's Version. Yefim Dinitz.  
[https://rangevoting.org/Dinitz\\_alg.pdf](https://rangevoting.org/Dinitz_alg.pdf)
- ❑ [8] Lecture 11: Dinic's Algorithm. Gupta and Sleator.  
<http://www.cs.cmu.edu/afs/cs/academic/class/15451-f14/www/lectures/lec11/dinic.pdf>
- ❑ [9] Dinic' Max Flow Algorithm. Dominik Scheder.  
<https://basics.sjtu.edu.cn/~dominik/teaching/material/dinic-slides.pdf>