

UNIVERSIDAD NACIONAL DE INGENIERÍA

Análisis y Diseño de Algoritmos

Flujo máximo

Ford-Fulkerson, el teorema de flujo máximo y corte mínimo, y Edmonds-Karp

Gerson Garrido

Semana 11

Hoy

Sesión A

1. El problema: cuánto pasa
2. Un greedy que falla
3. La red residual
4. Ford-Fulkerson: integralidad y costo
5. El peor caso: el diamante
6. En C++: aristas en pares

Sesión B

7. Cortes: un techo para todo flujo
8. Flujo máximo y corte mínimo
9. Encontrar el corte
10. Edmonds-Karp
11. En la práctica

> **Lectura**
>
> Apuntes de la semana 11. Código en `code/week-11`.

SECCIÓN 01

El problema

01

Otra pregunta sobre el mismo objeto

Hasta ahora, los números de un grafo eran costos, y buscábamos *un* camino barato.

Ahora son **capacidades**: una tubería, un enlace, una carretera tienen un límite.

La pregunta: **¿cuánto puede pasar en total** de un punto a otro, repartido entre todos los caminos a la vez?

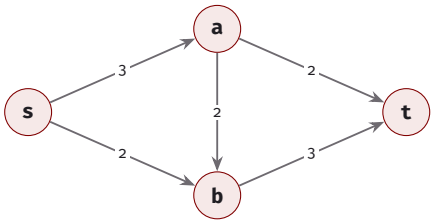
- Un greedy se atasca: lo veremos en cuatro nodos.
- No hay un subproblema pequeño a la vista: lo que pasa por un camino afecta a los que comparten sus aristas.

> **La idea nueva**
>
> Dejar que un greedy se equivoque, y **darle una forma de deshacer**.

Red de flujo *(flow network)*

Red de flujo

Grafo dirigido, con una **capacidad** $c(e) \geq 0$ en cada arista, una **fuente** s *(source)* y un **sumidero** t *(sink)*, con $s \neq t$.



Es la red de toda la clase. Cada número es una capacidad.

Flujo y valor

Flujo *(flow)*

Un número $f(e)$ en cada arista, con:

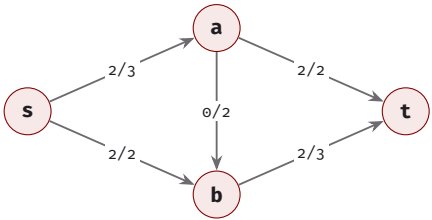
1. **Capacidad.** $0 \leq f(e) \leq c(e)$.
2. **Conservación.** En todo nodo distinto de s y t, lo que entra es igual a lo que sale.

Valor

$|f| =$ (lo que sale de s) $-$ (lo que entra a s). Es lo que sale **neto**.

La resta importa: una arista puede entrar a s, y lo que vuelve no se envió.
Flujo máximo *(maximum flow)*: un flujo de valor máximo.

Un flujo posible



Cada etiqueta es flujo/capacidad.

- Capacidades respetadas.
- En a: entran 2 y salen 2.
- En b: entran 2 y salen 2.
- Valor: $2 + 2 = 4$.
- De s solo salen capacidades $3 + 2 = 5$: nunca se pasará de 5.

Dos aclaraciones

Las aristas son dirigidas. Un tramo que va en los dos sentidos se escribe con dos aristas.

Se permiten aristas antiparalelas y paralelas. Antiparalelas: una de u a v, y otra de v a u.

- Algunos libros las prohíben: CLRS agrega un nodo intermedio.
- Aquí no hace falta: cada arista de la entrada tiene su propia capacidad y su propio flujo.
- Única consecuencia: en la red residual puede haber varios arcos entre dos nodos.

También se permiten aristas que entran a s o que salen de t. La definición de valor ya las contempla.

SECCIÓN 02

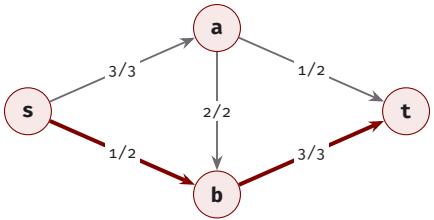
Un greedy que falla

02



La primera idea

Buscar un camino con espacio, empujar todo lo que se pueda, y repetir. Siempre el camino con más aristas.



$s \to b \to t$: empuja 1. Total 4. Quedan libres $s \to b$ y $a \to t$, pero no hay camino hacia adelante.

Se queda en 4, y el máximo es 5

terminal

```
$ build/greedy_demo
Sin volver atrás, siempre el camino con más aristas:
  s > a > b > t   empuja 2, total 2
  s > a > t   empuja 1, total 3
  s > b > t   empuja 1, total 4
  no queda camino: se queda en 4
```

Quedan libres $s \to b$ y $a \to t$, de 1 cada una. Desde b, la única salida es $b \to t$, saturada. A a no se llega: $s \to a$ también lo está.

¿Y si se eligen mejor los caminos?

build/greedy_demo, extracto

```
Sin volver atrás, todas las formas de elegir los caminos:
  termina en 4: 2 de 6 secuencias
  termina en 5: 4 de 6 secuencias
```

- Ningún camino está prohibido: el greedy no sabe cuál elegir.
- La información para decidir bien todavía no existe.

Idea clave

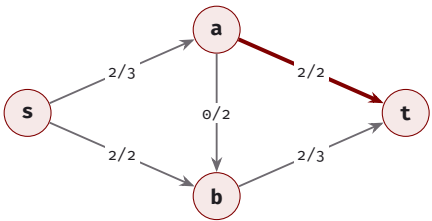
Una mala decisión de flujo no se evita, se deshace.

SECCIÓN 03

La red residual

03

Qué significa devolver flujo



- Después del primer camino pasan 2 por $a \to b$.
- Un nuevo camino llega a b por $s \to b$.
- En vez de seguir por $b \to t$, **cancela** 2 de las que iban de a a b.
- Esas unidades siguen en a, y salen por $a \to t$.
- Es como recorrer $a \to b$ al revés. Entraron 2 más, y llegaron a t.

La red residual *(residual network)*

Red residual G_f

Los mismos nodos. Por cada arista e de u a v, dos arcos:

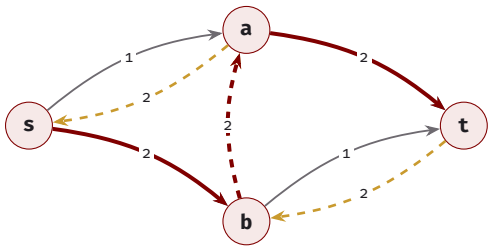
- **De ida** $u \to v$, con capacidad residual $c(e) - f(e)$: cuánto más se puede agregar.
- **Inverso** $v \to u$, con capacidad residual $f(e)$: cuánto se puede cancelar.

Solo están presentes los de capacidad residual positiva.

Arista	c	f	Ida	Inverso
$s \to a$	3	2	1	2
$s \to b$	2	0	2	0, no está
$a \to b$	2	2	0, no está	**2**
$a \to t$	2	0	2	0, no está
$b \to t$	3	2	1	2

Ahora existe $s \to b \Rightarrow a \to t$. El greedy no conocía el arco $b \Rightarrow a$.

La misma red residual, dibujada



Línea continua: arco de ida. Línea punteada: arco inverso.

Camino de aumento *(augmenting path)*: un camino simple de s a t en G_f.

Se empuja δ, la menor capacidad residual del camino. Aquí, 2.

Con aristas antiparalelas

Si $a \to b$ tiene capacidad 5 y flujo 3, y $b \to a$ tiene capacidad 4 y flujo 1:

Arco	Viene de	Capacidad residual
$a \to b$	la ida de $a \to b$	$5 - 3 = 2$
$b \to a$	la inversa de $a \to b$	3
$b \to a$	la ida de $b \to a$	$4 - 1 = 3$
$a \to b$	la inversa de $b \to a$	1

Entre a y b hay dos arcos en cada sentido, cada uno con su propia cuenta. Nada de lo que sigue necesita que haya uno solo.

Lema: aumentar da otro flujo

Lema. Si P es un camino de aumento con cuello de botella δ, lo que resulta, f', es un flujo, y $|f'| = |f| + \delta$.
Técnica. Comprobar cada condición. Para conservación, mirar el **saldo** de un nodo: lo que sale menos lo que entra.

- **Capacidad.** Ida: $\delta \leq c(e) - f(e)$, así que $f(e) + \delta \leq c(e)$.
- Inverso: $\delta \leq f(e)$, así que $f(e) - \delta \geq 0$.
- **Saldo.** Tomemos un arco del camino, de x a y.
- Si es la ida de $x \to y$: sale δ más de x, y entra δ más a y.
- Si es la inversa de $y \to x$: sale δ menos de y, y entra δ menos a x.

En los dos casos

El saldo de donde sale el arco sube δ. El saldo de adonde llega baja δ.

Lema: conservación y valor

El saldo de donde sale un arco sube δ, y el de adonde llega baja δ. P es simple.

- **Nodo intermedio de P:** un arco que llega y uno que sale. Baja δ y sube δ: no cambia.
- **Nodo que no está en P:** no cambia.
- **La fuente:** solo tiene un arco que sale. Su saldo sube δ, y $|f'| = |f| + \delta$.
- **El sumidero:** baja δ, y no se le exige conservación. ■

Como $\delta > 0$, cada aumento hace crecer el valor estrictamente.

SECCIÓN 04

Ford-Fulkerson

04

Ford-Fulkerson

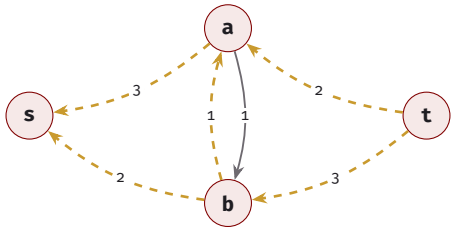
pseudocódigo

```
algoritmo ford_fulkerson(red, s, t)
    f[e] ← 0 en cada arista e
    mientras exista un camino de aumento P en la red residual
      hacer
        δ ← menor capacidad residual entre los arcos de P
        para cada arco de P hacer
            si es el arco de ida de e entonces
                f[e] ← f[e] + δ
            sino
                f[e] ← f[e] - δ
            fin si
        fin para
    fin mientras
    devolver f
fin algoritmo
```

No dice qué camino elegir: es una familia de algoritmos, uno por cada regla.

Por el lema, siempre hay un flujo. Que sea *máximo* se demuestra en la sesión B.

Un recorrido, en la red residual



Fin. De s no sale ningún arco residual: no hay camino de aumento. El valor es 5, el techo que habíamos visto.

El flujo de cada arista, vuelta por vuelta

	Camino	δ	$s \to a$	$s \to b$	$a \to b$	$a \to t$	$b \to t$	Valor
Capacidad			3	2	2	2	3	
Inicio			0	0	0	0	0	0
Vuelta 1	$s \to a \to b \to t$	2	2	0	2	0	2	2
Vuelta 2	$s \to b \Rightarrow a \to t$	2	2	2	**0**	2	2	4
Vuelta 3	$s \to a \to b \to t$	1	3	2	1	2	3	5

build/greedy_demo, extracto

```
Ford-Fulkerson, la misma regla con aristas inversas (< es recorrer una inversa):
  s > a > b > t   empuja 2, total 2
  s > b < a > t   empuja 2, total 4
  s > a > b > t   empuja 1, total 5
  no queda camino: termina en 5
```

La misma red y la misma primera mala decisión. La vuelta 2 la corrige.

Integralidad

Teorema. Con capacidades enteras, todos los $f(e)$ son enteros en todo momento, y hay a lo sumo f^* aumentos, donde f^* es el valor de un flujo máximo.
Técnica. Inducción sobre las vueltas. Invariante: todos los flujos son enteros.

- **Base.** Al inicio son 0.
- **Paso.** Si los $f(e)$ son enteros, $c(e) - f(e)$ y $f(e)$ también. δ es la menor de ellas, y es un entero: $\delta \geq 1$.
- Cada $f(e)$ cambia en $+\delta$, $-\delta$ o nada, y sigue siendo entero.
- **Terminación.** Cada vuelta sube el valor en al menos 1, y ningún flujo supera f^*. ■

Lo usaremos la próxima semana

Hay un flujo máximo entero: “una unidad” puede ser “una persona”. No dice que todo flujo máximo sea entero.

El costo: $O(E \cdot f)$

- Una búsqueda en la red residual: $O(V + E)$, con a lo sumo $2E$ arcos.
- Si cada nodo toca alguna arista, $V \leq 2E$, y la búsqueda es $O(E)$.
- Con capacidades enteras: a lo sumo f aumentos, más una búsqueda final.

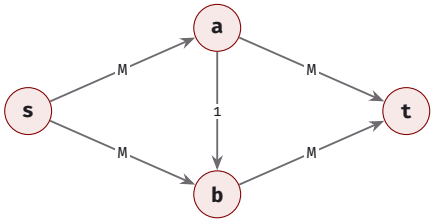
Costo de Ford-Fulkerson

$O(E \cdot f)$, donde f es el **valor del flujo máximo**, con capacidades enteras y cada búsqueda en $O(E)$.

f es un **número**, no un tamaño. Como $O(nW)$ en la mochila, es **pseudopolinomial**: duplicar los bits de las capacidades puede elevar f al cuadrado.

Con capacidades irracionales y una mala elección, puede no terminar. En un programa son enteros.

El peor caso: el diamante



Máximo: $2M$, con dos caminos bien elegidos.

Regla mala: siempre el camino con **más aristas**.

La arista del medio oscila entre llena y vacía, y su capacidad 1 limita cada vuelta a 1.

Contando las vueltas

make bench, extracto: -O2, una sola corrida; los milisegundos dependen de la máquina

capacidad	regla	flujo	iteraciones	ms
10	el más largo	20	20	0.0
10	primero de un DFS	20	4	0.0
10	BFS (Edmonds-Karp)	20	2	0.0
1000000	el más largo	2000000	2000000	360.3
1000000	primero de un DFS	2000000	4	0.0
1000000	BFS (Edmonds-Karp)	2000000	2	0.0

Con capacidades de 10^6 y una arista de 1 en medio: 2×10^6 **iteraciones** en cuatro nodos. La cota se alcanza.

Lo que dice el diamante

- Hay que dar la arista del medio con capacidad 1. Con capacidad 10^6, el primer camino empujaría 10^6.
- En esta red no puede hacer más: cada iteración empuja al menos 1, y el máximo es 2×10^6.
- DFS hace 4 aquí, pero eso no es una garantía: lo medimos más adelante.

Error común

"E es pequeño, así que es rápido." El peligro es f: miren las capacidades, no solo el tamaño del grafo.

SECCIÓN 05

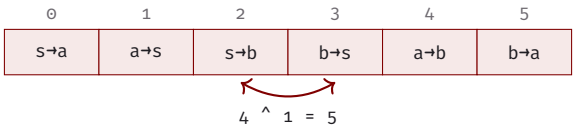
En C++

05



Aristas en pares: `e ^ 1`

Todos los arcos en un arreglo, **de a dos**: la arista en una posición par y su inversa justo después.



- El par del arco `e` es `e ^ 1`, en los dos sentidos: cambia el último bit.
- No se guarda el flujo: solo la **capacidad residual** de cada arco.
- El flujo de una arista se lee en su inversa. La capacidad es la suma de los dos.

Agregar una arista

flow.h

```cpp
int add_edge(int from, int to, long long capacity) {
    const int edge = static_cast<int>(to_.size());
    to_.push_back(to);
    residual_.push_back(capacity);
    arcs_from_[static_cast<std::size_t>(from)].push_back(edge);
    to_.push_back(from);
    residual_.push_back(0);
    arcs_from_[static_cast<std::size_t>(to)].push_back(edge + 1);
    return edge;
}
```

- Los dos arcos se crean **juntos**. Si no, los pares quedan corridos, y `e ^ 1` apunta a otra arista sin que nada avise.
- Dos aristas antiparalelas son dos pares independientes.

Empujar es una sola línea

flow.h

```cpp
long long flow_on(int edge) const { return residual(edge ^ 1); }
long long capacity(int edge) const { return residual(edge) + residual(edge ^ 1); }

void push(const std::vector<int>& path, long long amount) {
    for (int arc : path) {
        residual_[static_cast<std::size_t>(arc)] -= amount;
        residual_[static_cast<std::size_t>(arc ^ 1)] += amount;
    }
}
```

- Las dos ramas del pseudocódigo son la misma línea: quitarle al arco y darle a su par.
- `long long`: dos aristas de 2×10^9 caben en `int` una por una, pero su suma da overflow.

La regla es un parámetro

flow.h

```cpp
template <typename Rule>
FlowRun ford_fulkerson(FlowNetwork& net, int source, int sink, Rule rule,
                       std::vector<Augmentation>* log = nullptr) {
    long long iterations = 0;
    while (true) {
        const std::vector<int> path = rule(net, source, sink);
        if (path.empty()) break;
        const long long amount = net.bottleneck(path);
        net.push(path, amount);
        iterations++;
        if (log != nullptr) log->push_back({path, amount});
    }
    return {net.flow_value(source), iterations};
}
```

Cambiar de regla es cambiar un argumento. Es lo que hace `bad_capacities`.

Error común: la arista inversa

Olvidar crearla con capacidad 0

Sin arista inversa, es el greedy del principio: compila, termina, y a veces acierta. En nuestra red, con la regla del camino con más aristas, da 4 en vez de 5.

Tampoco se crea con capacidad *c*: eso deja pasar flujo por la arista al revés. Aquí, de *s* no se llega a *t*:

build/greedy_demo, extracto

```
Si la inversa se crea con capacidad c en lugar de 0, en s -> b, a -> b, a -> t, todas de 1:
  con capacidad 0: flujo 0
  con capacidad c: flujo 1
```

SECCIÓN 06

Cortes

06

Cómo certificar que no se puede más

Ya lo hicimos a ojo: de s salen $3 + 2 = 5$. Se puede hacer con cualquier separación.

Corte *(cut)*

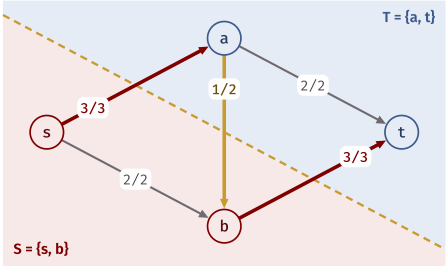
Una partición de los nodos en (S, T), con s en S y t en T.

- **Capacidad** $c(S, T)$: suma de capacidades de las aristas **de** S **a** T.
- **Flujo neto** $f(S, T)$: f en las aristas de S a T, menos f en las de T a S.

Las aristas de T a S **no cuentan** para la capacidad, y **sí restan** en el flujo neto.

Corte mínimo *(minimum cut)*: de capacidad mínima. Puede haber varios.

Un corte con una arista que vuelve



De S a T

s → a 3/3
b → t 3/3

De T a S

a → b 1/2

No cruzan

s → b y a → t

Capacidad: 3 + 3 = 6
a → b no suma.

Flujo neto: 3 + 3 − 1 = 5
a → b resta. Es |f|.

Error común

Sumar $a \to b$ en la capacidad daría 8. Una arista que entra a S no limita lo que sale.

Lema: todo corte ve el mismo número

Lema. Para todo flujo f y todo corte (S, T): $f(S, T) = |f|$.

Técnica. Sumar los saldos de los nodos de S, de dos formas.

- **Por nodos.** Los de S distintos de s tienen saldo 0, y t no está en S. La suma es $|f|$.
- **Por aristas.** Cada arista suma $f(e)$ a donde sale, y resta $f(e)$ a donde llega:

Extremos	Aporte
Los dos en S	$+f(e) - f(e) = 0$
Los dos en T	0
De S a T	$+f(e)$
De T a S	$-f(e)$

El total es $f(S, T)$. Es la misma suma, así que $f(S, T) = |f|$. ■

Dualidad débil: ningún flujo pasa un corte

Teorema. Para todo flujo f y todo corte (S, T): $|f| \leq c(S, T)$.

$$
\begin{aligned}
|f| &= f(S, T) && \text{por el lema} \\
&= \textstyle\sum_{S \to T} f(e) - \sum_{T \to S} f(e) \\
&\leq \textstyle\sum_{S \to T} f(e) && \text{porque } f(e) \geq 0 \\
&\leq \textstyle\sum_{S \to T} c(e) = c(S, T) && \text{porque } f(e) \leq c(e) \qquad \blacksquare
\end{aligned}
$$

Cuándo hay igualdad

$|f| = c(S, T)$ exactamente cuando las aristas de S a T están **saturadas** y las de T a S llevan **0**.

Un flujo y un corte con el mismo valor son los dos óptimos: es un **certificado**.

En nuestra red

terminal

```
$ build/cut_demo
La primera red, con su flujo máximo de valor 5:
  lado de s     capacidad   flujo neto
  {s}                   5            5
  {s, a}                6            5
  {s, b}                6            5
  {s, a, b}             5            5
```

- Cuatro cortes: *a* y *b* pueden caer de cualquier lado.
- En todos, el flujo neto es 5. La capacidad es 5 o más.
- Hay dos cortes mínimos, y los dos tienen capacidad igual al flujo.

SECCIÓN 07

Flujo máximo y corte mínimo

07

El teorema

Teorema de flujo máximo y corte mínimo

Sea f un flujo. Estas tres afirmaciones son equivalentes:

1. f es un flujo máximo.
2. La red residual G_f no tiene ningún camino de aumento.
3. Existe un corte (S, T) con $|f| = c(S, T)$.

En consecuencia, el valor del flujo máximo es igual a la capacidad del corte mínimo.

Técnica. El ciclo $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 1$. La parte difícil es $2 \Rightarrow 3$: hay que construir el corte, con lo alcanzable en la red residual.

Las dos implicaciones fáciles

(1 $\Rightarrow$ 2) Por contraposición.

- Si G_f tuviera un camino de aumento, el lema daría un flujo de valor $|f| + \delta$, con $\delta > 0$.
- Entonces f no sería máximo.

(3 $\Rightarrow$ 1) Sea (S, T) con $|f| = c(S, T)$, y sea g cualquier flujo.

- Por la dualidad débil, $|g| \leq c(S, T) = |f|$.
- Así que f es máximo.

(2 $\Rightarrow$ 3): construir el corte

G_f no tiene camino de s a t. Sea S lo alcanzable desde s en G_f, y T el resto. $s \in S$ y $t \in T$: es un corte.

Toda arista e de $u \in S$ a $v \in T$ está saturada.

- Si no, $f(e) < c(e)$, y el arco de ida $u \to v$ está en G_f.
- Como u es alcanzable, v también lo sería. Contradicción.

Toda arista e de $v \in T$ a $u \in S$ tiene flujo 0.

- Si no, $f(e) > 0$, y el arco inverso de e, de u a v, está en G_f.
- Otra vez v sería alcanzable. Contradicción.

Por lo tanto, $|f| = f(S,T) = \sum_{S \to T} c(e) - 0 = c(S,T)$. ■

Lo que queda

- **La consecuencia.** Un flujo máximo tiene, por $1 \Rightarrow 3$, un corte de su misma capacidad. Todo corte tiene al menos $|f|$. Luego ese corte es mínimo.
- **Ford-Fulkerson es correcto.** Se detiene sin camino de aumento: por $2 \Rightarrow 1$, tiene un flujo máximo. Vale con cualquier regla, si termina.
- **La demostración es un algoritmo.** $2 \Rightarrow 3$ dice cómo encontrar el corte.

> **Idea clave**
>
> Un problema de máximo y otro de mínimo, en apariencia distintos, con la misma respuesta. Al terminar, el algoritmo entrega los dos certificados.

SECCIÓN 08

Encontrar el corte

08

Encontrar el corte mínimo

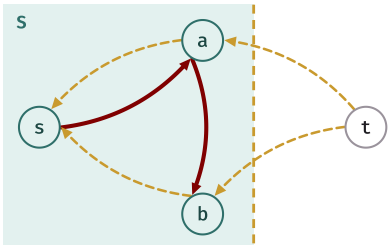
pseudocódigo

```
algoritmo corte_minimo(red, f, s)
    S ← nodos alcanzables desde s en la red residual de f, con BFS o DFS
    corte ← lista vacía
    para cada arista e de la red original hacer
        si e va de un nodo de S a un nodo que no está en S entonces
            agregar e a corte
        fin si
    fin para
    devolver corte
fin algoritmo
```

- **Se lee en la red original**, solo los arcos pares: un arco inverso de S a T es la vuelta de una arista de T a S.
- En nuestra red: $S = \{s\}$, y el corte es $\{s \to a,\ s \to b\}$, de capacidad 5.
- Comprobación barata: las capacidades del corte tienen que sumar el flujo.

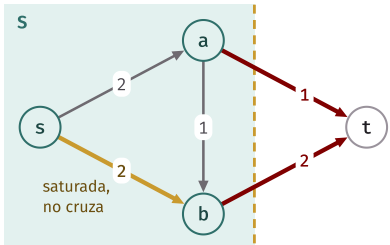
“El corte son las aristas saturadas”: no

En la red residual: lo alcanzable



Se llega a a y a b por s → a → b.
De S no sale ningún arco hacia t.

En la red original: el corte



a → t (1) + b → t (2) = 3, el flujo máximo.
Las saturadas suman 5, que no es el flujo.

S = {s, a, b}, con el flujo de BFS. Cada arista del corte va de S a T.

Toda arista del corte está saturada, pero no toda saturada está en el corte.

La misma red, corrida

build/cut_demo, extracto

```
flujo máximo: 3, cortes mínimos: 1
lado de s     capacidad   flujo neto
{s}                   4            3
{s, a}                4            3
{s, b}                4            3
{s, a, b}             3            3
con BFS, alcanzable en la red residual: {s, a, b}
aristas del corte:  a -> t (1)  b -> t (2)
```

aristas saturadas, con BFS

```
  s -> b (2)  no cruza
  a -> t (1)  cruza
  b -> t (2)  cruza
suma de las saturadas: 5
```

aristas saturadas, con otro flujo

```
  s -> a (2)  no cruza
  a -> b (1)  no cruza
  a -> t (1)  cruza
  b -> t (2)  cruza
suma de las saturadas: 6
```

Las saturadas que no cruzan dependen del flujo. Las del corte son las mismas.

¿Aumentar una arista del corte aumenta el flujo?

No siempre. Dos cortes mínimos en serie:

build/cut_demo, extracto

```
Dos cortes mínimos en serie (s -> a 5, a -> t 5):
  flujo máximo: 5, cortes mínimos: 2
  con s -> a de 6:          flujo 5
  con a -> t de 6:          flujo 5
  con las dos de 6:         flujo 6
```

Con capacidades enteras, sí es cierto que:

- El flujo **nunca sube más de 1**: el corte anterior aumentó a lo sumo 1, y sigue siendo un techo.
- Si el corte mínimo es **único** y la arista va de S a T, sube exactamente 1: todo otro corte tenía al menos una unidad más.

Las pruebas lo comprueban en 2000 redes al azar.

SECCIÓN 09

Edmonds-Karp

09

Edmonds-Karp: el camino con menos aristas

pseudocódigo

```
algoritmo edmonds_karp(red, s, t)
    f[e] ← 0 en cada arista e
    mientras verdadero hacer
        ejecutar BFS desde s en la red residual,
            guardando llegada[v]
        si t no se alcanzó entonces
            salir del ciclo
        fin si
        P ← los arcos de t hacia atrás hasta s,
            siguiendo llegada
        δ ← menor capacidad residual de P
        aumentar f por P en δ
    fin mientras
    devolver f
fin algoritmo
```

Teorema

Con cualquier capacidad, $O(VE)$ aumentos, y $O(VE^2)$ en total. Las capacidades no aparecen.

Las distancias nunca bajan

build/ek_trace

```
$ build/ek_trace
  distancias: s=0 a=2 b=1 c=1 d=2 t=3
vuelta 1: s > b > a > t   3 aristas, empuja 3
  críticas: a->t
  distancias: s=0 a=2 b=1 c=1 d=2 t=3
vuelta 2: s > b > d > t   3 aristas, empuja 1
  críticas: s->b
  distancias: s=0 a=2 b=2 c=1 d=3 t=4
vuelta 3: s > c > b > d > t   4 aristas, empuja 2
  críticas: c->b
  distancias: s=0 a=2 b=3 c=1 d=4 t=5
vuelta 4: s > c > a < b > d > t   5 aristas, empuja 2
  críticas: c->a b->d d->t
  distancias: s=0 a=- b=- c=1 d=- t=-
flujo máximo: 8, corte mínimo por fuerza bruta: 8
```

Red: $s \to b$ 4, $s \to c$ 6, $a \to t$ 3, $b \to a$ 6, $b \to d$ 5, $c \to a$ 2, $c \to b$ 2, $d \to t$ 5.

t pasa por 3, 3, 4, 5, y b por 1, 1, 2, 3. Ninguna distancia baja.

El BFS de la vuelta 4

Arcos con residual positivo ($\Rightarrow$ es inverso):

s	$s \to c$: 4
a	$a \Rightarrow b$: 3
b	$b \to a$: 3, $b \to d$: 2, $b \Rightarrow s$: 4, $b \Rightarrow c$: 2
c	$c \to a$: 2, $c \Rightarrow s$: 2
d	$d \to t$: 2, $d \Rightarrow b$: 3

Sale	Distancia	Nuevos
s	0	c
c	1	a
a	2	b
b	3	d
d	4	t

El camino es $s \to c \to a \Rightarrow b \to d \to t$, con capacidades 4, 2, 3, 2, 2. Empuja 2.

El flujo de $b \to a$ baja de 3 a 1: dos unidades que iban de b a a siguen por $b \to d$, y las que llegan a a desde c ocupan su lugar.

Después, solo se alcanzan s y c: $S = \{s, c\}$, y el corte es $s \to b$, $c \to a$, $c \to b$, con $4 + 2 + 2 = 8$.

La idea de la demostración, paso 1

Las distancias nunca bajan. Sea $d(x)$ la distancia en aristas desde s en la red residual actual.

- Por BFS, todo arco de x a y cumple $d(y) \leq d(x) + 1$.
- Al aumentar por un camino más corto, algunos arcos desaparecen, y otros aparecen: inversos de arcos del camino, que bajan del nivel $k + 1$ al k.
- En la red nueva, todos los arcos cumplen la desigualdad con las d **viejas**.
- Un camino avanza a lo sumo un nivel viejo por arco: para llegar a x necesita al menos $d(x)$ arcos.

Es la idea, no una demostración completa

Falta escribirla por inducción sobre los aumentos, con los nodos que dejan de alcanzarse.

La idea de la demostración, pasos 2 y 3

Arco crítico *(critical)*: su capacidad residual es igual a δ, y al empujar desaparece. Todo aumento tiene al menos uno.

- Si $x \to y$ es crítico: $d(y) = d(x) + 1$.
- Para volver a aparecer, un camino más corto posterior tiene que usar $y \to x$: $d'(x) = d'(y) + 1$.
- Como las distancias no bajan: $d'(x) = d'(y) + 1 \geq d(y) + 1 = d(x) + 2$.
- Entre dos veces crítico, $d(x)$ sube al menos 2, y es a lo sumo $V - 2$: a lo sumo $V/2$ veces.

Sumar. $2E$ arcos, $V/2$ veces cada uno: a lo sumo VE aumentos. Cada uno, un BFS de $O(E)$.

Total

$O(VE^2)$, sin depender de las capacidades.

Las capacidades no aparecen

En el diamante, BFS hace 2 iteraciones frente a 2×10^6: nunca toca la arista del medio.

build/ek_trace, extracto

```
Las mismas iteraciones si todas las capacidades se multiplican:
  por       1: flujo         8, iteraciones 4
  por    1000: flujo      8000, iteraciones 4
  por 1000000: flujo   8000000, iteraciones 4
```

BFS solo mira qué arcos tienen capacidad positiva, así que sigue exactamente las mismas vueltas.

Las pruebas comprueban en 3000 redes al azar: las distancias no bajan, cada arco es crítico a lo sumo $V/2$ veces, y las vueltas no pasan de VE.

Una medición, en redes por capas

build/iterations_bench: -O2, una corrida; los tiempos dependen de la máquina

V	E	regla	flujo	iteraciones	VE	ms
102	290	BFS	2942603981	60	29580	0.1
102	290	DFS	2942603981	1297	29580	1.9
202	590	BFS	2375625399	79	119180	0.3
202	590	DFS	2375625399	16299	119180	42.2
402	1190	BFS	2942603981	192	478380	1.3
402	1190	DFS	2942603981	1028098	478380	4576.3
La red de 402 nodos, con las aristas agregadas en orden al azar:						
402	1190	BFS	2942603981	198	478380	1.6
402	1190	DFS	2942603981	6292	478380	24.4

- BFS queda muy por debajo de *VE*. DFS hizo más de un millón; con otro orden de las aristas, 6292.
- Son pocas redes de una sola familia: no dicen cómo crece. Las garantías son las de las cotas.

Las dos, frente a frente

	Ford-Fulkerson, regla cualquiera	Edmonds-Karp
Camino	No lo dice	El de menos aristas, con BFS
Aumentos	A lo sumo f, con capacidades enteras	$O(VE)$, con cualquier capacidad
Costo	$O(E \cdot f)$	$O(VE^2)$
Depende de las capacidades	Sí	No
Capacidades irracionales	Puede no terminar	Termina
Diamante, $M = 10^6$	Hasta 2×10^6 vueltas	2 vueltas

Con 500 nodos, 5000 aristas y capacidades de hasta 10^9: $E \cdot f$ es del orden de $2{,}5 \times 10^{15}$, y VE^2 de $1{,}25 \times 10^{10}$.

SECCIÓN 10

En la práctica

10

Otros algoritmos

Edmonds-Karp es lo que programamos en este curso. No es lo mejor que se conoce.

- **Dinic.** También con distancias de BFS, aprovecha en cada fase todos los caminos más cortos. $O(V^2E)$.
- **Los de preflujo** *(push-relabel)*. No mantienen la conservación durante el proceso, y empujan exceso de un nodo a otro. Boost Graph trae `push_relabel_max_flow`.

En un proyecto real, con un grafo grande, lo razonable es usar una biblioteca.

> **Lo que no cambia**
>
> Qué es un flujo, qué certifica un corte, cómo se lee la respuesta. Y cómo se convierte un problema en una red: la semana 12.

Lo que hay que llevarse

- **Red residual:** ida con $c - f$, inverso con f. Permite deshacer.
- **Ford-Fulkerson:** mientras haya camino de aumento, empujar. Con enteros, flujo entero, en $O(E \cdot f)$.
- **Diamante:** capacidades de 10^6 y una de 1 en medio: 2×10^6 iteraciones con una mala regla.
- **En C++:** pares `e` y `e ^ 1`, creados juntos, la inversa con 0. `long long`.
- **Corte:** solo las aristas de S a T. Flujo neto por todo corte $= |f| \leq c(S, T)$.
- **Teorema:** máximo, sin camino de aumento y corte de igual capacidad son equivalentes.
- **Corte mínimo:** S alcanzable en la residual; se lee en la red original. No toda saturada cruza.
- **Edmonds-Karp:** BFS. Las distancias no bajan, $O(VE)$ aumentos, $O(VE^2)$.

Para la próxima clase

- Leer los apuntes de la semana 11, y hacer sus ejercicios.
- **Práctica 3** en la sesión de laboratorio: programación dinámica.
- Opcional: la capacidad de una red, en `lab/network_capacity`.

Preguntas

Corran Ford-Fulkerson a mano en una red de 4 nodos en la que un mal camino obligue a usar una arista inversa.
Demuestren que el valor de un flujo nunca supera la capacidad de un corte.
Con Ford-Fulkerson y capacidades de 1000000, ¿cuántas iteraciones puede hacer en el peor caso? Den la red.
Dado un flujo máximo, ¿cómo se encuentra un corte mínimo?
Si se aumenta en 1 la capacidad de una arista del corte mínimo, ¿siempre aumenta el flujo máximo? ¿Por qué?