

# Flot-max / coupe-min

CM n°1 — Mobilité (M2 IMPAIRS)

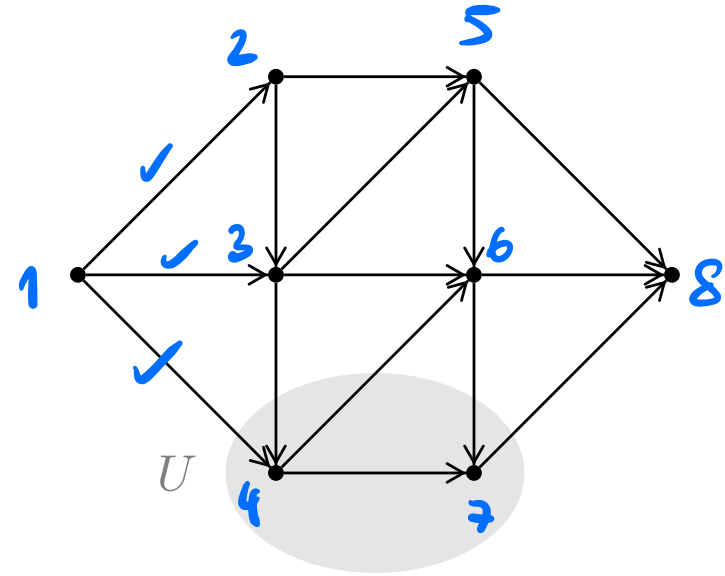
Matěj Stehlík

8/1/2024

## Arcs sortants et entrants

### Définition

Soient  $G = (V, E)$  un graphe orienté et  $U \subseteq V$ . Alors, on note  $\delta^+(U)$  l'ensemble des arcs dont le début est dans  $U$ . De même, on note  $\delta^-(U)$  l'ensemble des arcs dont la fin est dans  $U$ .

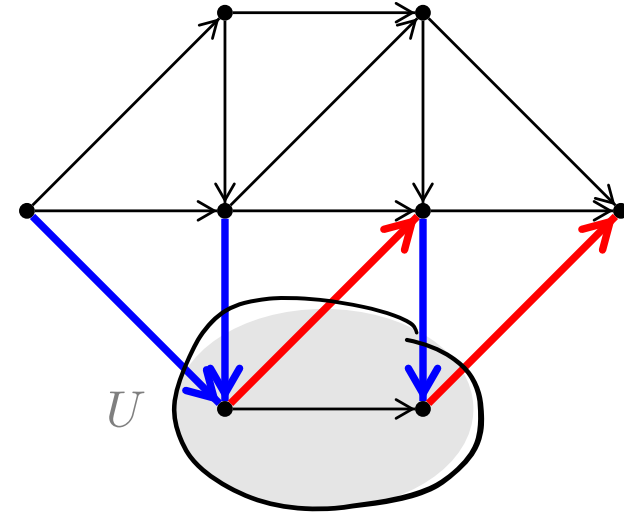


$(\{1, 2, \dots, 8\}, \{(1, 2), (1, 3), (1, 4), \dots\})$

## Arcs sortants et entrants

### Définition

Soient  $G = (V, E)$  un graphe orienté et  $U \subseteq V$ . Alors, on note  $\delta^+(U)$  l'ensemble des arcs dont le début est dans  $U$ . De même, on note  $\delta^-(U)$  l'ensemble des arcs dont la fin est dans  $U$ .



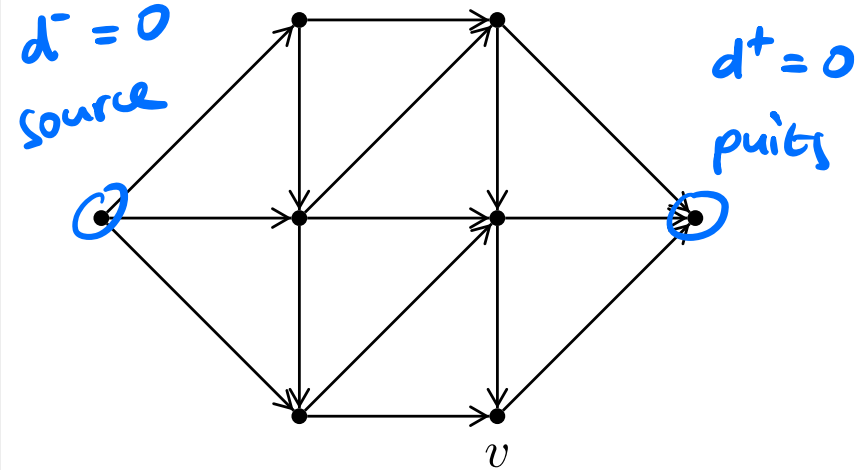
$\delta^+(U)$  arcs sortants

$\delta^-(U)$  arcs entrants

# Degré sortant et degré entrant

## Définition

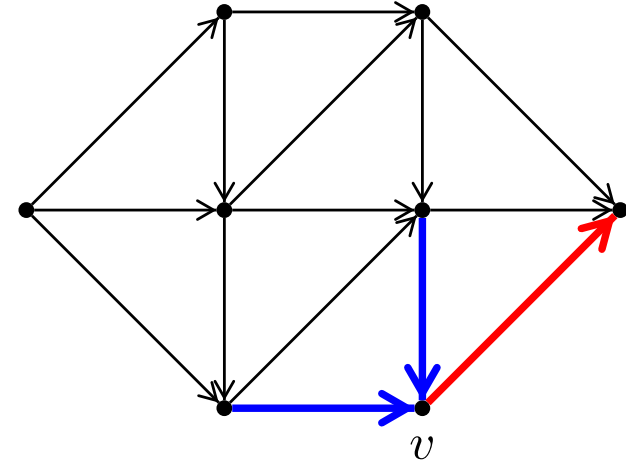
- le *degré entrant*  $d^-(v)$ , est le nombre d'arcs dont la fin est  $v$ .
- le *degré sortant*  $d^+(v)$ , est le nombre d'arcs dont le début est  $v$ .
- On a  $d^-(v) = |\delta^-(\{v\})|$  et  $d^+(v) = |\delta^+(\{v\})|$ .
- $v$  est une *source* si  $d^-(v) = 0$ .
- $v$  est un *puits* si  $d^+(v) = 0$ .



# Degré sortant et degré entrant

## Définition

- le *degré entrant*  $d^-(v)$ , est le nombre d'arcs dont la fin est  $v$ .
- le *degré sortant*  $d^+(v)$ , est le nombre d'arcs dont le début est  $v$ .
- On a  $d^-(v) = |\delta^-(\{v\})|$  et  $d^+(v) = |\delta^+(\{v\})|$ .
- $v$  est une *source* si  $d^-(v) = 0$ .
- $v$  est un *puits* si  $d^+(v) = 0$ .



$$d^+(v) = 1$$

$$d^-(v) = 2$$

# Flot max et coupe min

- Deux problèmes classiques de l'optimisation combinatoire.
- Exemple de la dualité mathématique.

## Quelques applications

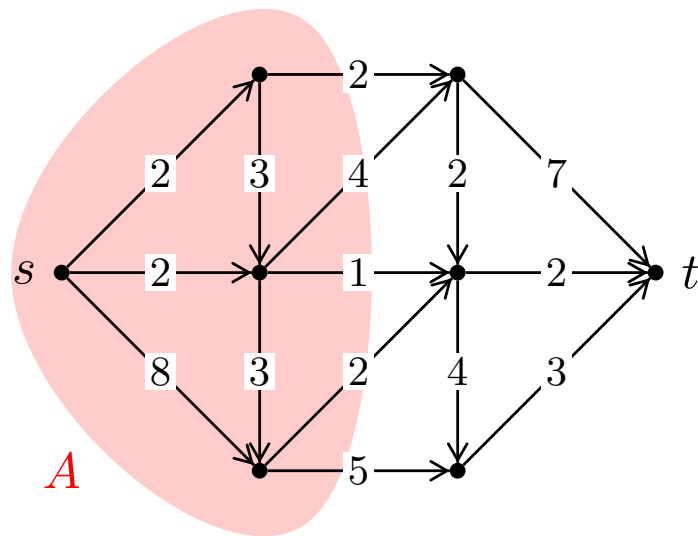
- Fouille de données.
- Sélection de projets.
- Ordonnancement (par exemple compagnies aériennes).
- Segmentation de l'image.
- Connectivité et fiabilité des réseaux.
- Calcul distribuée. . .

## Définition

Un *réseau* est un graphe orienté  $G = (V, E)$  avec :

- deux sommets spéciaux :
  - la source  $s$  tel que  $d^-(s) = 0$
  - le puits  $t$  tel que  $d^+(t) = 0$

- une fonction de *capacité*  
 $c : E \rightarrow \mathbb{R}^+$ . (les capacités sont réelles et positives)



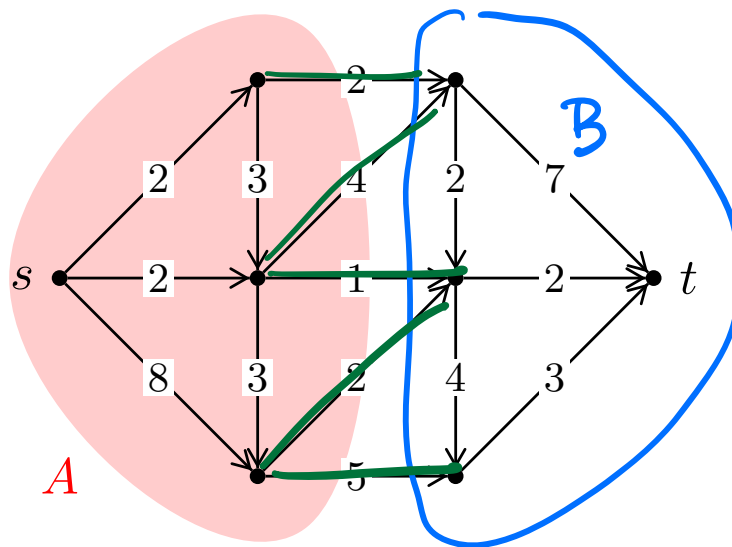
# Coupes

## Définition

Une  $s - t$  coupe est une partition  $(A, B)$  de  $V$  telle que  $s \in A$  et  $t \in B$

## Définition

La *capacité* d'une coupe  $(A, B)$  est  $\text{cap}(A, B) = \sum_{e \in \delta^+(A)} c(e)$ .



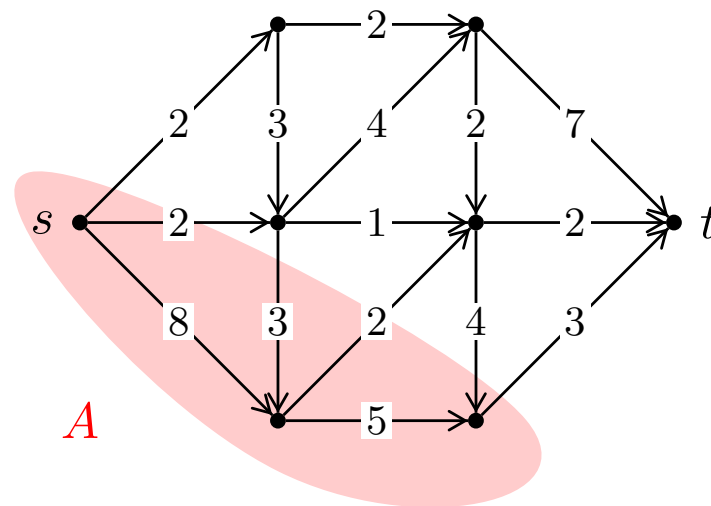
$$\text{cap}(A, B) = 2 + 4 + 1 + 2 + 5 = 14$$



# Le problème de la coupe minimum

## Problème

Trouver une  $s - t$  coupe de capacité minimum.



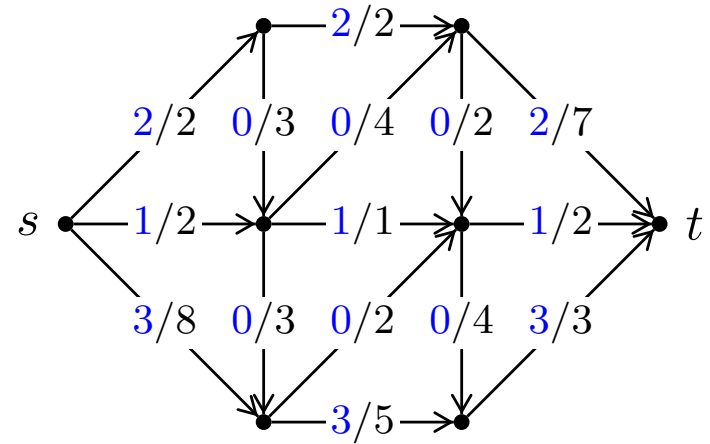
$$\text{cap}(A, B) = 2 + 2 + 2 + 3 = 9$$

# Flots

## Définition

Un  $s - t$  flot est une fonction  $f : E \rightarrow \mathbb{R}^+$  qui vérifie

- Pour tout  $e \in E$ ,  $0 \leq f(e) \leq c(e)$   
(contrainte de capacité)
- Pour tout  $v \in V - \{s, t\}$ ,  
 $\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e)$   
(conservation de flot).



$$\text{val}(f) = 2 + 1 + 3 = 6$$

## Définition

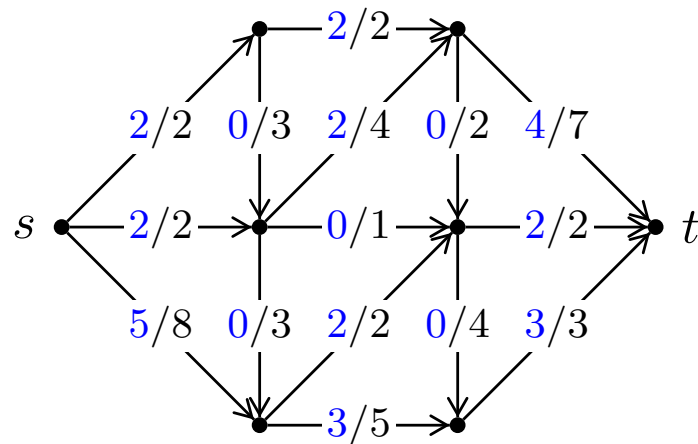
La *valeur* d'un flot  $f$  est

$$\text{val}(f) = \sum_{e \in \delta^+(s)} f(e).$$

# Le problème du flot maximum

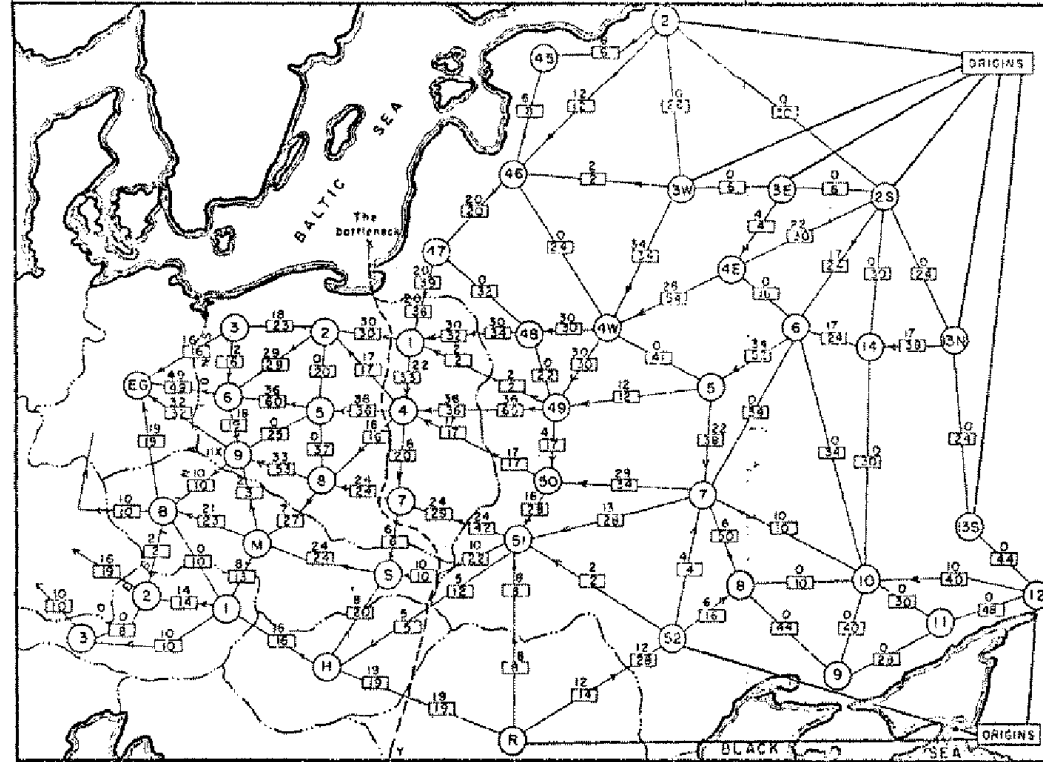
## Problème

Trouver un  $s - t$  flot de valeur maximum.



$$\text{val}(f) = 2 + 2 + 5 = 9$$

# Un peu d'histoire : le réseau ferroviaire du bloc de l'Est

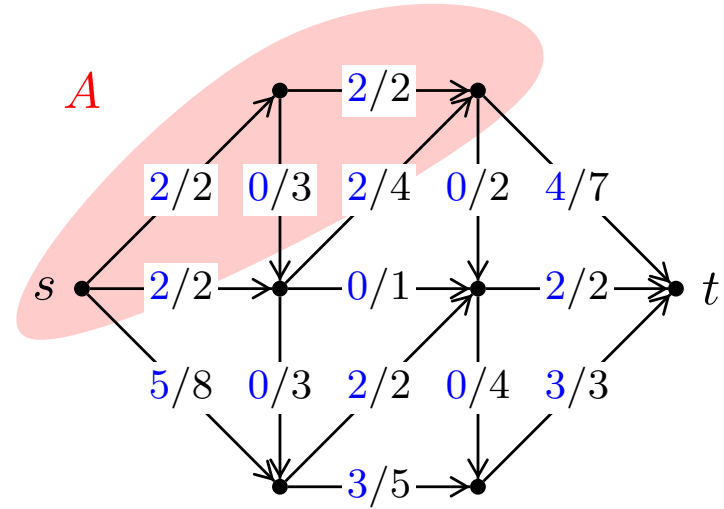


# Flots et coupes

## Lemme

Soit  $f$  un flot et  $(A, B)$  une  $s - t$  coupe. Alors, le flot qui traverse la coupe est égal au flot qui sort de  $s$ .

$$\sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e) = \text{val}(f).$$



$$\begin{aligned} \text{val}(f) &= 2 + 2 + 5 \\ &= 5 + 2 - 2 + 4 \end{aligned}$$

# Flots et coupes

## Démonstration

$$\text{val}(f) = \sum_{e \in \delta^+(s)} f(e)$$

déf. de  $\text{val}(f)$

$$= \sum_{e \in \delta^+(s)} f(e) - \underbrace{\sum_{e \in \delta^-(s)} f(e)}_{=0}$$

parce que  
 $\delta^-(s) = \emptyset$   
(s est une source)

$$= \sum_{v \in A} \left( \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) \right)$$

par la conservation  
de flot

$$= \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e).$$

# Dualité faible

## Lemme

Soit  $f$  un flot et  $(A, B)$  une  $s - t$  coupe quelconques. Alors, la valeur de  $f$  est inférieure ou égale à la capacité de la coupe.

## Démonstration

$$\begin{aligned} \text{val}(f) &= \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e) \\ &\leq \sum_{e \in \delta^+(A)} f(e) \\ &\leq \sum_{e \in \delta^+(A)} c(e) \\ &= \text{cap}(A, B). \end{aligned}$$

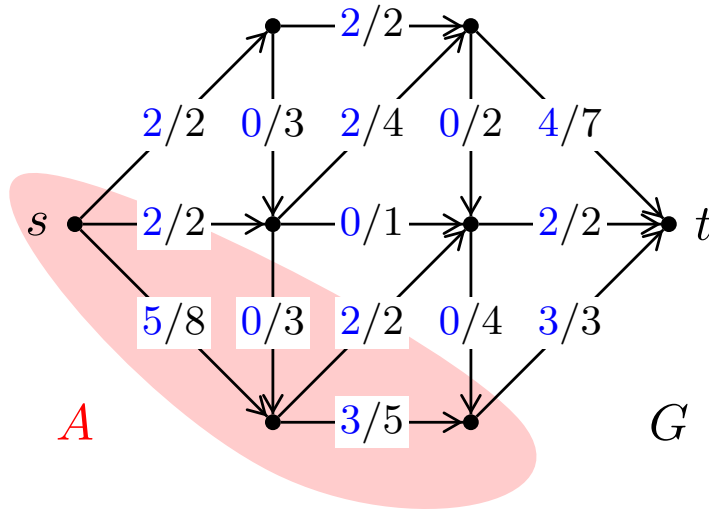
$\geq 0$  contrainte de capacité  $0 \leq f$

contrainte de capacité  $f \leq c$

# Certificat d'optimalité

## Corollaire

Soit  $f$  un flot et  $(A, B)$  une coupe. Si  $\text{val}(f) = \text{cap}(A, B)$ , alors  $f$  est un flot max et  $(A, B)$  est une coupe min.



$$\text{val}(f) = 9$$

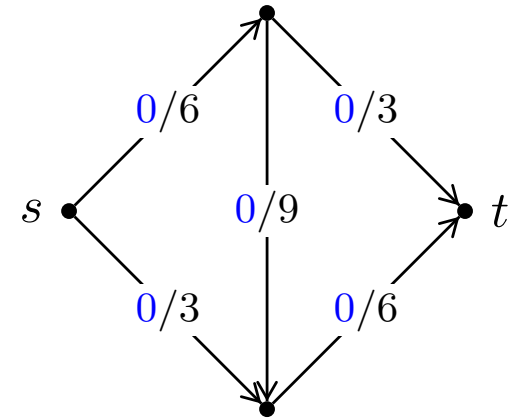
$$\text{cap}(A, B) = 9$$



# Vers un algorithme de flot max

## Un algorithme glouton

- Commencer par le flot nul, càd,  $f(e) = 0$  pour chaque arc  $e \in E$ .
- Trouver un  $s - t$  chemin  $P$  où tout arc vérifie  $f(e) < c(e)$ .
- Augmenter le flot le long le chemin  $P$ .
- Répéter jusqu'à devenir coincé.

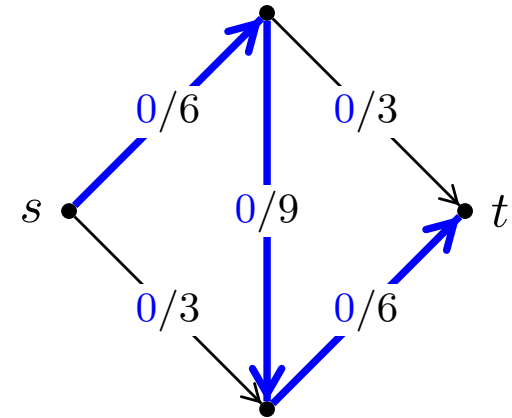


$$\text{val}(f) = 6$$

# Vers un algorithme de flot max

## Un algorithme glouton

- Commencer par le flot nul, càd,  $f(e) = 0$  pour chaque arc  $e \in E$ .
- Trouver un  $s - t$  chemin  $P$  où tout arc vérifie  $f(e) < c(e)$ .
- Augmenter le flot le long le chemin  $P$ .
- Répéter jusqu'à devenir coincé.

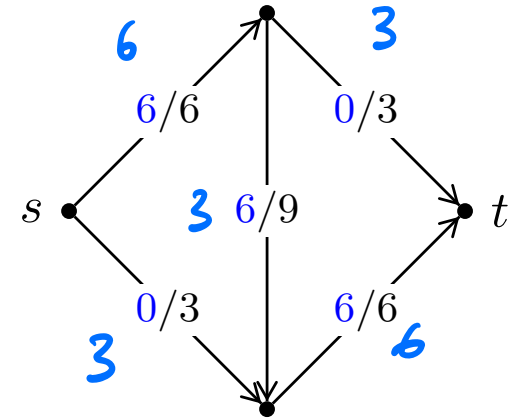


$$\text{val}(f) = 6$$

# Vers un algorithme de flot max

## Un algorithme glouton

- Commencer par le flot nul, càd,  $f(e) = 0$  pour chaque arc  $e \in E$ .
- Trouver un  $s - t$  chemin  $P$  où tout arc vérifie  $f(e) < c(e)$ .
- Augmenter le flot le long le chemin  $P$ .
- Répéter jusqu'à devenir coincé.



$$\text{val}(f) = 6$$

# Le graphe résiduel

Arc originel :

- $e = (u, v) \in E$ .
- Flot  $f(e)$ , capacité  $c(e)$ .

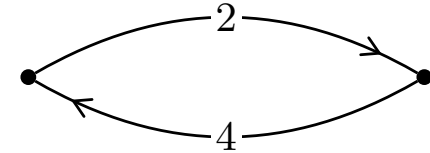
Arc résiduel :

- “annuler” le flot envoyé.
- $e = (u, v)$  et  $e^R = (v, u)$ .
- Capacité résiduelle :

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{si } e \in E \\ f(e) & \text{si } e^R \in E. \end{cases}$$



arc de  $G$



arcs de  $G_f$

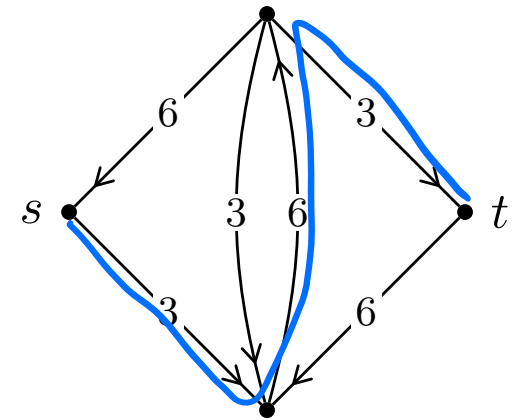
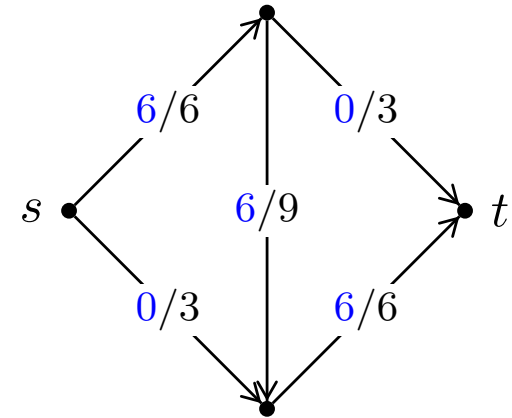
# Le graphe résiduel

Graphe résiduel :

- Arcs résiduels avec capacité résiduelle positive.
- $E_f = \{e \mid f(e) < c(e)\} \cup \{e^R \mid f(e) > 0\}$ .

Chemin augmentant :

- Un *chemin augmentant* est un  $s$ - $t$  chemin simple dans le graphe résiduel  $G_f$ .
- La *capacité*  $\text{cap}(G_f, P)$  d'un chemin augmentant  $P$  est le minimum des capacités résiduelles parmi tous les arcs de  $P$ .



## Propriété clé

- Soit  $f$  un flot et  $P$  un chemin augmentant dans  $G_f$ .
- En envoyant un flot de valeur  $\text{cap}(G_f, P)$  on obtient un nouveau flot  $f'$  de valeur  $\text{val}(f') = \text{val}(f) + \text{cap}(G_f, P)$ .

Augment( $f, c, P$ ) :

$\varepsilon \leftarrow \min\{c_f(e) : e \in E(P)\}$

**for**  $e \in E(P)$ :

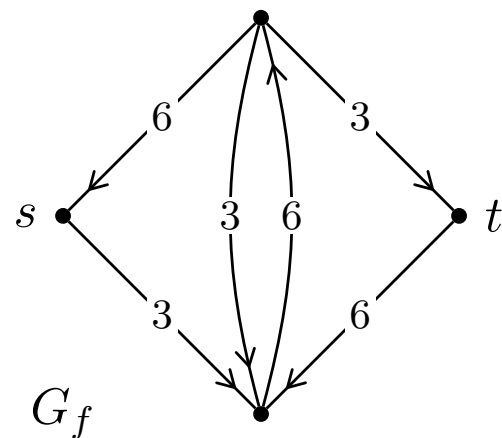
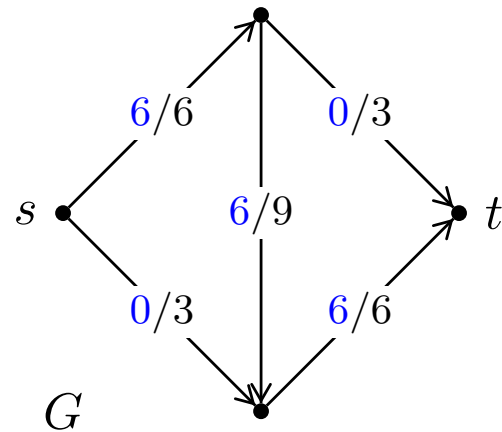
**if**  $e \in E$ :

$f(e) \leftarrow f(e) + \varepsilon$

**else**:

$f(e^R) \leftarrow f(e) - \varepsilon$

**return**  $f$



## Propriété clé

- Soit  $f$  un flot et  $P$  un chemin augmentant dans  $G_f$ .
- En envoyant un flot de valeur  $\text{cap}(G_f, P)$  on obtient un nouveau flot  $f'$  de valeur  $\text{val}(f') = \text{val}(f) + \text{cap}(G_f, P)$ .

Augment( $f, c, P$ ) :

$\varepsilon \leftarrow \min\{c_f(e) : e \in E(P)\}$

**for**  $e \in E(P)$ :

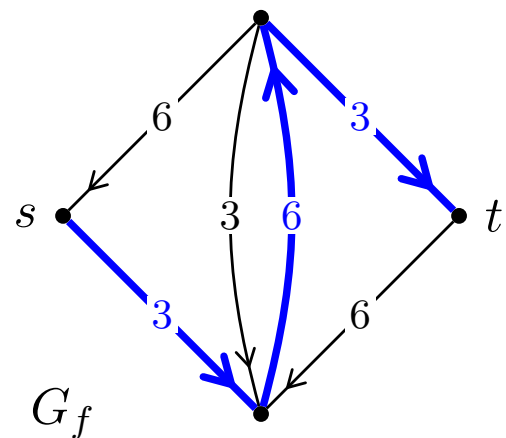
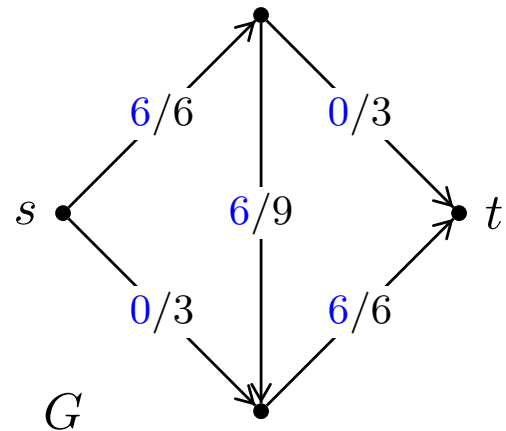
**if**  $e \in E$ :

$f(e) \leftarrow f(e) + \varepsilon$

**else**:

$f(e^R) \leftarrow f(e) - \varepsilon$

**return**  $f$



## Propriété clé

- Soit  $f$  un flot et  $P$  un chemin augmentant dans  $G_f$ .
- En envoyant un flot de valeur  $\text{cap}(G_f, P)$  on obtient un nouveau flot  $f'$  de valeur  $\text{val}(f') = \text{val}(f) + \text{cap}(G_f, P)$ .

Augment( $f, c, P$ ) :

$\varepsilon \leftarrow \min\{c_f(e) : e \in E(P)\}$

**for**  $e \in E(P)$ :

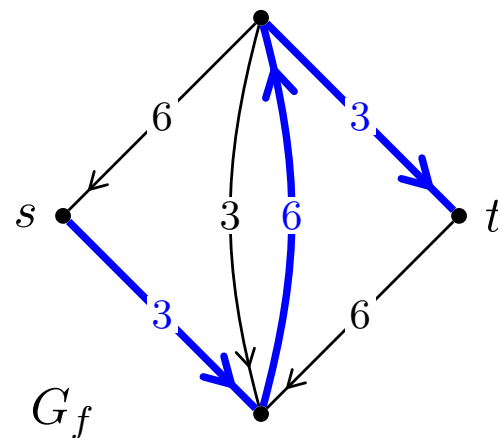
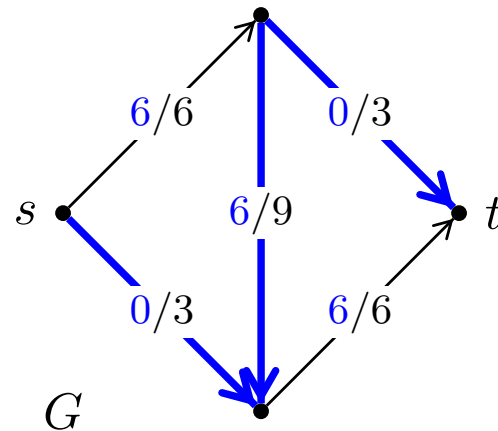
**if**  $e \in E$ :

$f(e) \leftarrow f(e) + \varepsilon$

**else**:

$f(e^R) \leftarrow f(e) - \varepsilon$

**return**  $f$





## Propriété clé

- Soit  $f$  un flot et  $P$  un chemin augmentant dans  $G_f$ .
- En envoyant un flot de valeur  $\text{cap}(G_f, P)$  on obtient un nouveau flot  $f'$  de valeur  $\text{val}(f') = \text{val}(f) + \text{cap}(G_f, P)$ .

Augment( $f, c, P$ ) :

$\varepsilon \leftarrow \min\{c_f(e) : e \in E(P)\}$

**for**  $e \in E(P)$ :

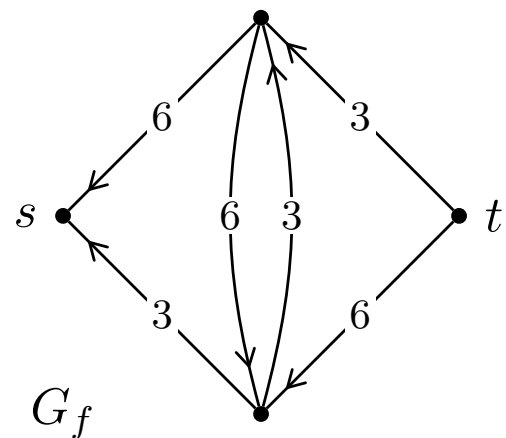
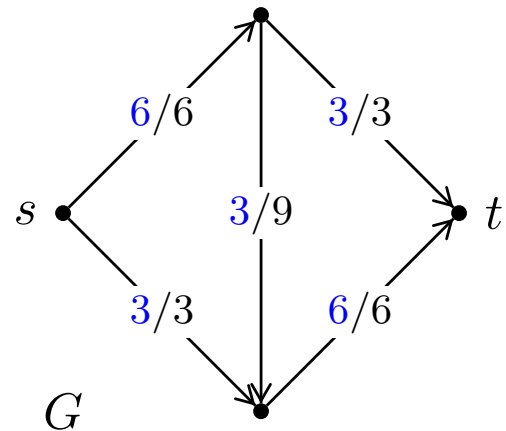
**if**  $e \in E$ :

$f(e) \leftarrow f(e) + \varepsilon$

**else**:

$f(e^R) \leftarrow f(e) - \varepsilon$

**return**  $f$



# Algorithme de Ford–Fulkerson

Ford–Fulkerson( $G, s, t, c$ ):

**for**  $e \in E(G)$ :

$f \leftarrow 0$

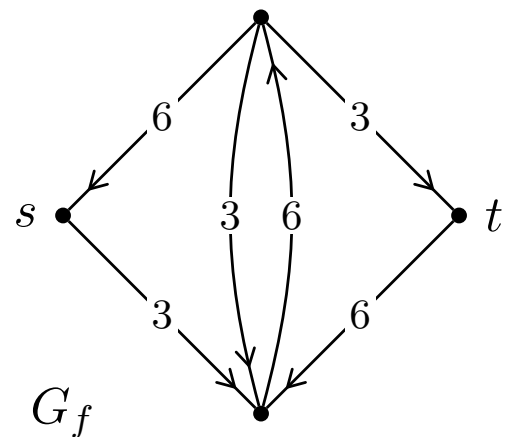
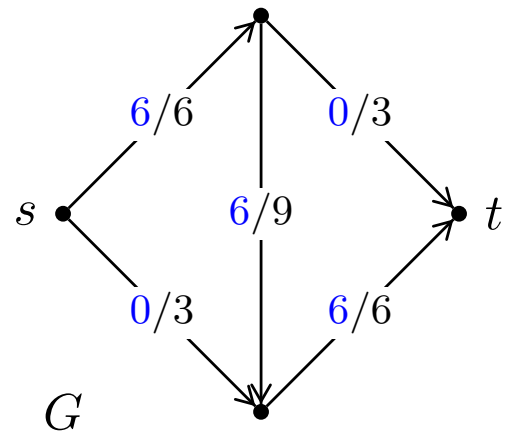
$G_f \leftarrow$  graphe résiduel

**while**  $\exists$  chemin augmentant  $P$ :

$f \leftarrow \text{Augment}(f, c, P)$

    mettre à jour  $G_f$

**return**  $f$



# Algorithme de Ford–Fulkerson

```
Ford-Fulkerson( $G, s, t, c$ ):
```

```
  for  $e \in E(G)$ :
```

```
     $f \leftarrow 0$ 
```

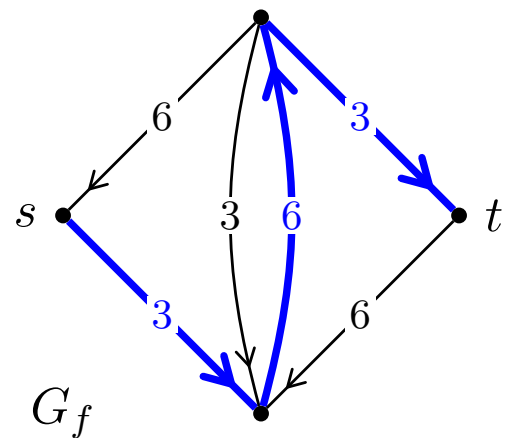
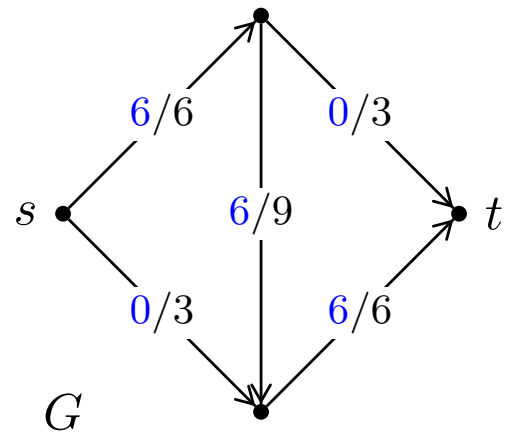
```
   $G_f \leftarrow$  graphe résiduel
```

```
  while  $\exists$  chemin augmentant  $P$ :
```

```
     $f \leftarrow \text{Augment}(f, c, P)$ 
```

```
    mettre à jour  $G_f$ 
```

```
  return  $f$ 
```



# Algorithme de Ford–Fulkerson

```
Ford-Fulkerson( $G, s, t, c$ ):
```

```
  for  $e \in E(G)$ :
```

```
     $f \leftarrow 0$ 
```

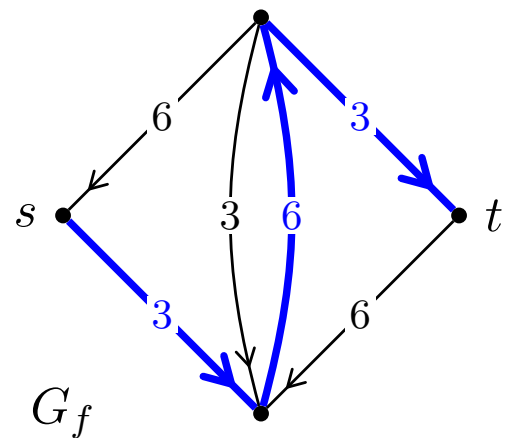
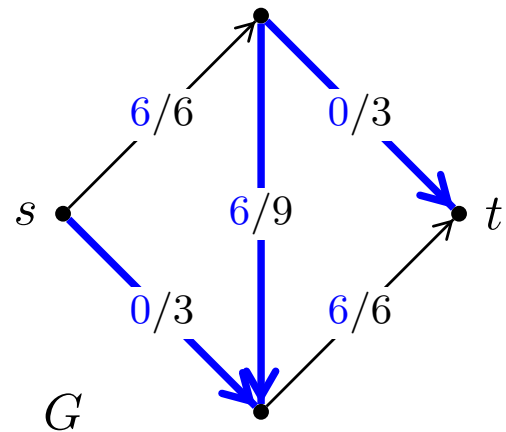
```
   $G_f \leftarrow$  graphe residuel
```

```
  while  $\exists$  chemin augmentant  $P$ :
```

```
     $f \leftarrow \text{Augment}(f, c, P)$ 
```

```
    mettre a jour  $G_f$ 
```

```
  return  $f$ 
```



# Algorithme de Ford–Fulkerson

```
Ford-Fulkerson( $G, s, t, c$ ):
```

```
  for  $e \in E(G)$ :
```

```
     $f \leftarrow 0$ 
```

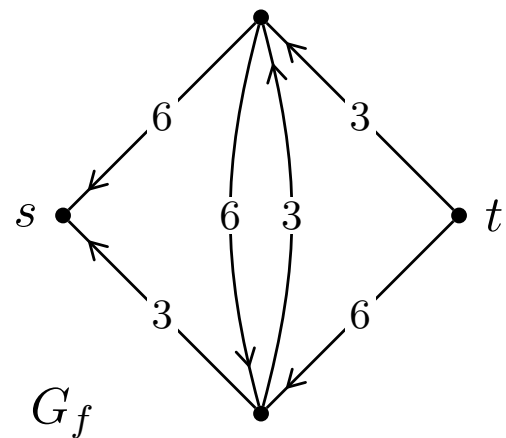
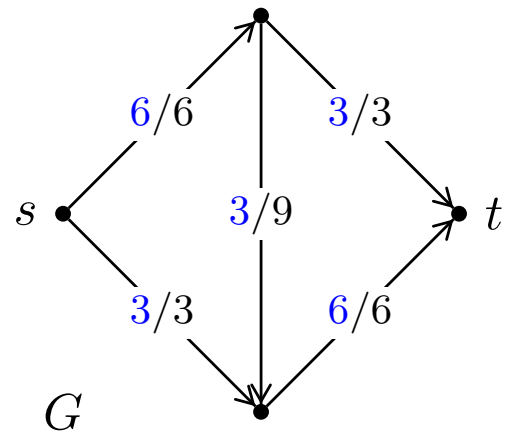
```
   $G_f \leftarrow$  graphe résiduel
```

```
  while  $\exists$  chemin augmentant  $P$ :
```

```
     $f \leftarrow \text{Augment}(f, c, P)$ 
```

```
    mettre à jour  $G_f$ 
```

```
  return  $f$ 
```



# Le théorème flot-max/coupe-min

## **Théorème des chemins augmentants**

Un flot  $f$  est maximum ssi il n'y a pas de chemin augmentant.

## **Théorème (Elias–Feinstein–Shannon 1956 ; Ford–Fulkerson 1956)**

La valeur maximum d'un flot est égale à la capacité minimum d'une coupe.

*Garantit l'existence d'un  
certificat d'optimalité.*

On va prouver les deux théorèmes en même temps en démontrant que les énoncés suivants sont équivalents :

- (1) Il existe une coupe  $(A, B)$  telle que  $\text{val}(f) = \text{cap}(A, B)$ .
- (2) Le flot  $f$  est maximum.
- (3) Il n'existe pas de chemin augmentant par rapport à  $f$ .

## Démonstration du théorème flot-max/coupe-min (1/2)

(1)  $\Rightarrow$  (2) Corollaire à la dualité faible.

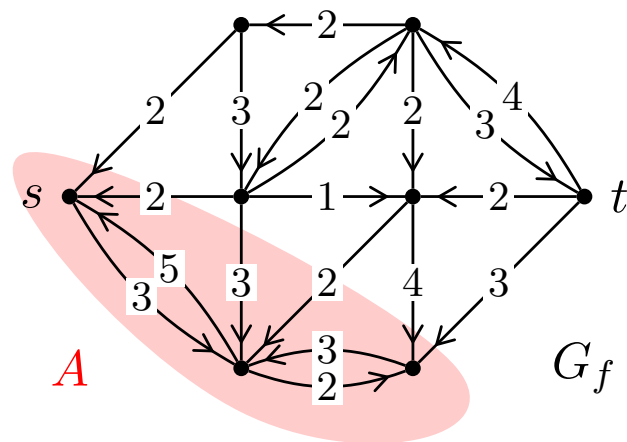
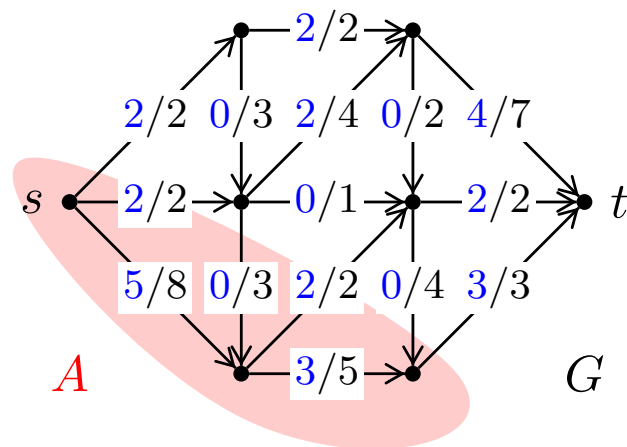
(2)  $\Rightarrow$  (3) Soit  $f$  un flot. S'il existe un chemin augmentant  $P$ , on peut augmenter  $f$  en envoyant un flot le long  $P$ .

(3)  $\Rightarrow$  (1)

- Soit  $f$  un flot sans chemin augmentant.
- Soit  $A$  un ensemble de sommets atteignables depuis  $s$  dans le graphe résiduel.  
*il existe un chemin (respectant le sens des flèches) de  $s$  au sommet*
- Par la définition de  $A$ ,  $s \in A$ .
- Par la définition de  $f$ ,  $t \notin A$ .

## Démonstration du théorème flot-max/coupe-min (2/2)

$$\begin{aligned}
 \text{val}(f) &= \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e) \\
 &= \sum_{e \in \delta^+(A)} c(e) \\
 &= \text{cap}(A, B)
 \end{aligned}$$





## Comment trouver une coupe minimum ?

- Il suffit de prendre  $A$  comme l'ensemble des sommets atteignables à partir de  $s$  dans le graphe résiduel  $G_f$ .
- C'est-à-dire,  $v \in A$  ssi il existe un chemin orienté dans  $G_f$  avec sommet de départ  $s$  et sommet d'arrivée  $v$ .
- Si vous trouvez que  $t \in A$ , alors il existe un chemin augmentant et  $f$  n'est pas maximum — dans ce cas, il faut encore faire tourner Ford–Fulkerson !

## Terminaison de l'algorithme de Ford–Fulkerson

- Si toutes les capacités sont entières, alors tous les flots intermédiaires sont entiers (récurrence).
- En particulier,  $\varepsilon \geq 1$  à chaque étape.
- Donc, l'algorithme termine au bout de  $C = \sum_{e \in E} c(e)$  itérations de la boucle `while`.
- L'algorithme termine aussi quand les capacités sont rationnelles (il suffit de les multiplier par le plus petit commun multiple).
- En général, pour les capacités réelles, il y a des cas où l'algorithme ne termine jamais. . .

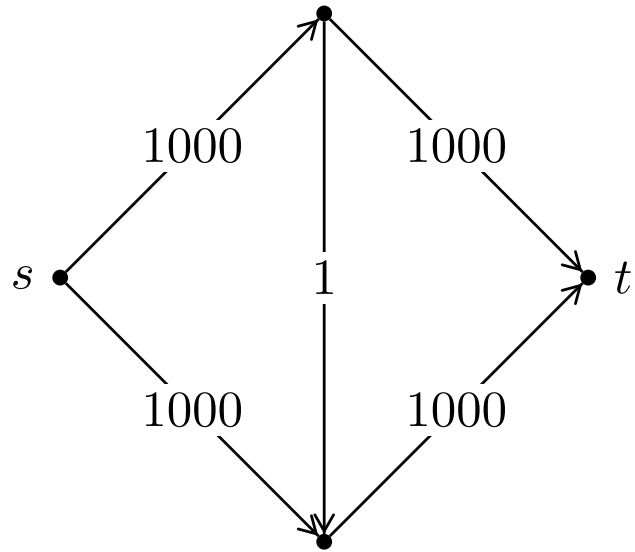
# Complexité de l'algorithme de Ford–Fulkerson (capacités entières)

## Théorème

Si toutes les capacités sont entières, la complexité de l'algorithme de Ford–Fulkerson est de  $O(mC)$ , où  $C = \sum_{e \in E} c(e)$ .

- Nous avons déjà remarqué qu'il y a  $O(C)$  itération de la boucle `while`.
- Dans chacune des itérations :
  - On peut trouver un chemin augmentant en temps  $O(m + n)$  (avec DFS ou BFS).
  - En supposant que chaque sommet de  $G$  est incident à au moins un arc, on a  $m \geq n/2$ ; par conséquent  $O(m + n) = O(m)$ .
  - La procédure `augment` est de complexité  $O(n)$ , la longueur du chemin  $P$  étant au plus  $n - 1$ .
  - Mettre à jour  $G_f$  est de complexité  $O(m)$ .

## Bien choisir les chemins augmentants



- Si l'algorithme de Ford–Fulkerson choisit toujours le chemin augmentant avec 3 arcs, alors l'algorithme termine au bout de **2**000 itérations.
- Par contre, si l'on choisit les chemins augmentants de longueur 2, alors l'algorithme se termine au bout de 2 itérations seulement !

## Une amélioration : l'algorithme de Edmonds–Karp

- Si l'on utilise un BFS pour trouver un chemin augmentant avec le nombre minimum d'arcs, on peut prouver que l'algorithme termine toujours (même pour les capacités irrationnelles).
- Cette version est appelée l'algorithme de Edmonds–Karp, et on peut prouver que sa complexité est de  $O(nm^2)$ .
- Peut-on faire mieux ?

# L'état de l'art

- Très récemment, Chen, Kyng, Liu, Peng, Probst Gutenberg et Sachdeva<sup>1</sup> ont donné un algorithme de flot-max de complexité  $m^{1+o(1)}$ , c'est-à-dire, d'une complexité quasi linéaire!
- Un vrai tour de force, utilise des techniques avancées de l'optimisation continue.

---

1. <https://arxiv.org/pdf/2203.00671.pdf>