

# Algorithmes de flots

Quentin Fortier

October 13, 2021

# Définitions

On considère un graphe orienté  $\vec{G} = (V, \vec{E})$  avec une **capacité**  $c : \vec{E} \longrightarrow \mathbb{R}^+$ .

# Définitions

On considère un graphe orienté  $\vec{G} = (V, \vec{E})$  avec une **capacité**  $c : \vec{E} \rightarrow \mathbb{R}^+$ .

## Notations

Si  $A \subseteq V$ , on note:

- $A^+ = \{(u, v) \in \vec{E} \mid u \in A, v \notin A\}$  (« arcs sortants de  $A$  »)

# Définitions

On considère un graphe orienté  $\vec{G} = (V, \vec{E})$  avec une **capacité**  $c : \vec{E} \rightarrow \mathbb{R}^+$ .

## Notations

Si  $A \subseteq V$ , on note:

- $A^+ = \{(u, v) \in \vec{E} \mid u \in A, v \notin A\}$  (« arcs sortants de  $A$  »)
- $A^- = \{(u, v) \in \vec{E} \mid u \notin A, v \in A\}$  (« arcs rentrants dans  $A$  »)

# Définitions

On considère un graphe orienté  $\vec{G} = (V, \vec{E})$  avec une **capacité**  $c : \vec{E} \rightarrow \mathbb{R}^+$ .

## Notations

Si  $A \subseteq V$ , on note:

- $A^+ = \{(u, v) \in \vec{E} \mid u \in A, v \notin A\}$  (« arcs sortants de  $A$  »)
- $A^- = \{(u, v) \in \vec{E} \mid u \notin A, v \in A\}$  (« arcs rentrants dans  $A$  »)
- Si  $v \in V$ ,  $v^+ = \{v\}^+$  et  $v^- = \{v\}^-$

# Définitions

On considère un graphe orienté  $\vec{G} = (V, \vec{E})$  avec une **capacité**  $c : \vec{E} \rightarrow \mathbb{R}^+$ .

## Notations

Si  $A \subseteq V$ , on note:

- $A^+ = \{(u, v) \in \vec{E} \mid u \in A, v \notin A\}$  (« arcs sortants de  $A$  »)
- $A^- = \{(u, v) \in \vec{E} \mid u \notin A, v \in A\}$  (« arcs rentrants dans  $A$  »)
- Si  $v \in V$ ,  $v^+ = \{v\}^+$  et  $v^- = \{v\}^-$

## Définition

Si  $f : \vec{E} \rightarrow \mathbb{R}^+$  et  $B \subseteq \vec{E}$  :

$$f(B) = \sum_{\vec{e} \in B} f(\vec{e})$$

## Flot

Soit  $s, t \in V$ .

Un **s-t flot** est une fonction  $f : \vec{E} \rightarrow \mathbb{R}^+$  telle que :

- $\forall \vec{e} \in \vec{E} : 0 \leq f(\vec{e}) \leq c(\vec{e})$

# Définitions

## Flot

Soit  $s, t \in V$ .

Un **s-t flot** est une fonction  $f : \vec{E} \rightarrow \mathbb{R}^+$  telle que :

- $\forall \vec{e} \in \vec{E} : 0 \leq f(\vec{e}) \leq c(\vec{e})$
- $\forall v \in V - \{s, t\} : f(v^-) = f(v^+)$

## Définition

$|f|$  est la **valeur** du flot  $f$



# Définitions

## Flot

Soit  $s, t \in V$ .

Un **s-t flot** est une fonction  $f : \vec{E} \rightarrow \mathbb{R}^+$  telle que :

- $\forall \vec{e} \in \vec{E} : 0 \leq f(\vec{e}) \leq c(\vec{e})$
- $\forall v \in V - \{s, t\} : f(v^-) = f(v^+)$

## Définition

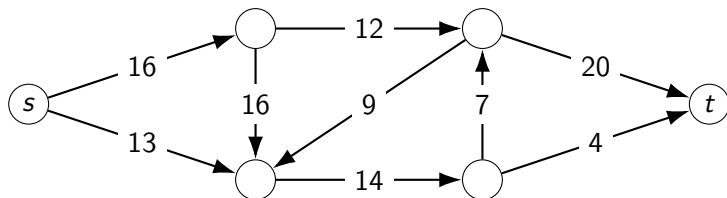
$|f|$  est la **valeur** du flot  $f$

## Problème

Trouver un flot dont la valeur est maximum.

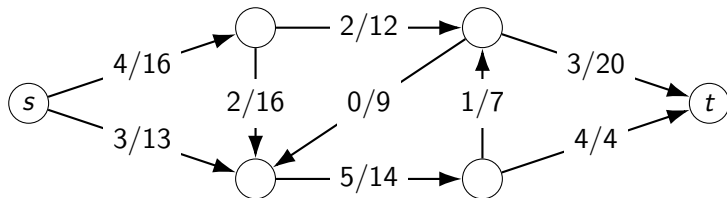
# Exemple

Exemple de graphe  $\vec{G}$  avec une capacité  $c$  sur les arcs :



# Définitions

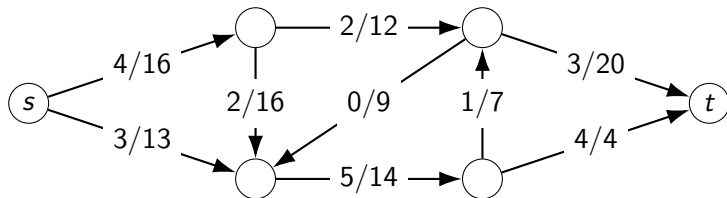
Exemple de flot :



Valeur du flot :

# Définitions

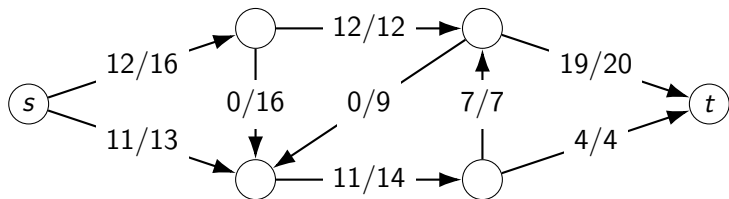
Exemple de flot :



Valeur du flot :  $4 + 3 = 7$

# Définitions

Exemple de flot de valeur maximum :



Valeur de ce flot : 23

# Algorithme de Ford-Fulkerson

Idée : partir d'un flot  $f$  non optimal (souvent le flot nul partout) et l'améliorer petit à petit.

# Algorithme de Ford-Fulkerson

Idée : partir d'un flot  $f$  non optimal (souvent le flot nul partout) et l'améliorer petit à petit.

## Graphe résiduel

La **capacité résiduelle** d'un arc est le flot que l'on peut encore y ajouter (capacité initiale moins flot)

# Algorithme de Ford-Fulkerson

Idée : partir d'un flot  $f$  non optimal (souvent le flot nul partout) et l'améliorer petit à petit.

## Graphe résiduel

La **capacité résiduelle** d'un arc est le flot que l'on peut encore y ajouter (capacité initiale moins flot)

Le **graphe résiduel** est obtenu en conservant la capacité résiduelle de chaque arc. Si un arc a une capacité résiduelle nulle, il est supprimé.



# Algorithme de Ford-Fulkerson

Idée : partir d'un flot  $f$  non optimal (souvent le flot nul partout) et l'améliorer petit à petit.

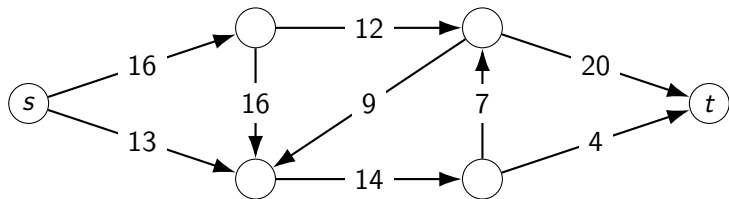
## Graphe résiduel

La **capacité résiduelle** d'un arc est le flot que l'on peut encore y ajouter (capacité initiale moins flot)

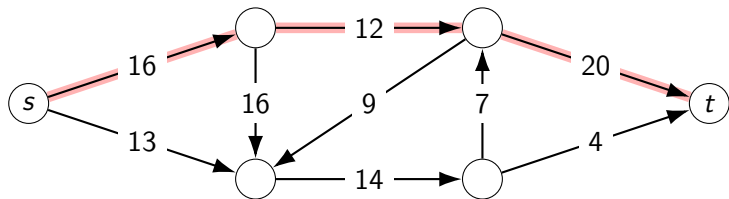
Le **graphe résiduel** est obtenu en conservant la capacité résiduelle de chaque arc. Si un arc a une capacité résiduelle nulle, il est supprimé.

Si on trouve un chemin  $\vec{P}$  de  $s$  à  $t$  dans le graphe résiduel, on peut augmenter le flot du minimum des capacités de  $\vec{P}$ .

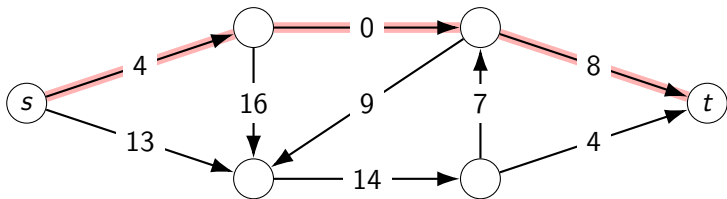
# Algorithme de Ford-Fulkerson



# Algorithme de Ford-Fulkerson



# Algorithme de Ford-Fulkerson



Flot augmenté de 12 (et capacité résiduelle diminuée de 12 le long du chemin).

On note sur chaque arc la capacité résiduelle (restante).

# Algorithme de Ford-Fulkerson

## Algorithme de Ford-Fulkerson

Tant que  $\exists$  un chemin  $\vec{P}$  de  $s$  à  $t$  dans le graphe résiduel :  
     $c \leftarrow$  minimum des capacités de  $\vec{P}$   
    Diminuer de  $c$  la capacité des arcs de  $\vec{P}$

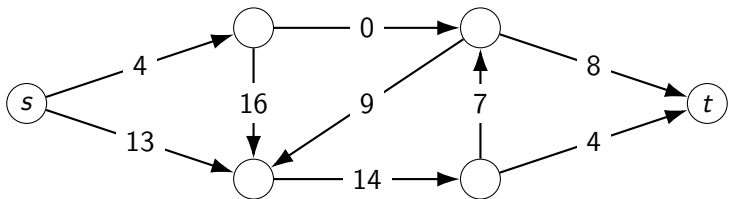
# Algorithme de Ford-Fulkerson

## Algorithme de Ford-Fulkerson

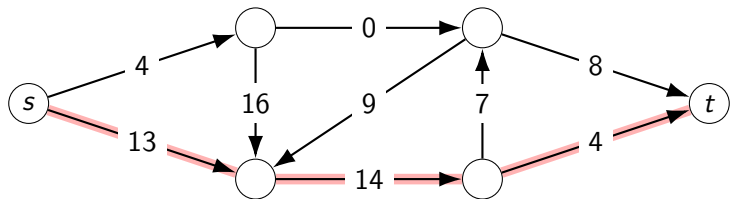
Tant que  $\exists$  un chemin  $\vec{P}$  de  $s$  à  $t$  dans le graphe résiduel :  
     $c \leftarrow$  minimum des capacités de  $\vec{P}$   
    Diminuer de  $c$  la capacité des arcs de  $\vec{P}$

À la fin, on peut connaître le flot sur chaque arc en retranchant la capacité résiduelle à la capacités initiale.

# Algorithme de Ford-Fulkerson

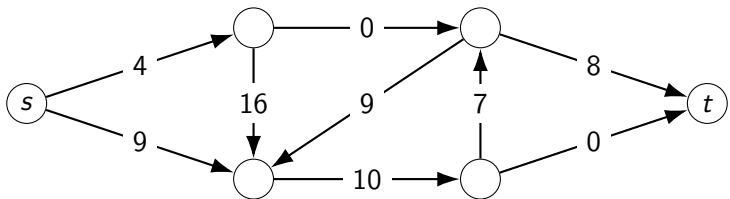


# Algorithme de Ford-Fulkerson

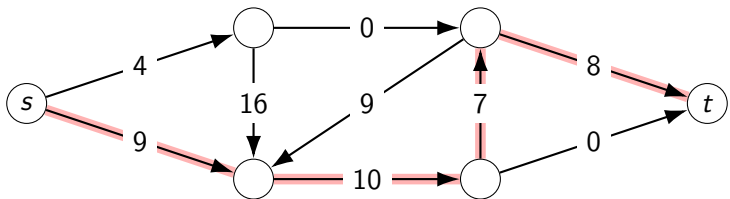




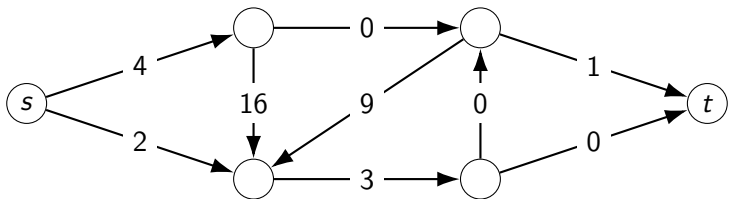
# Algorithme de Ford-Fulkerson



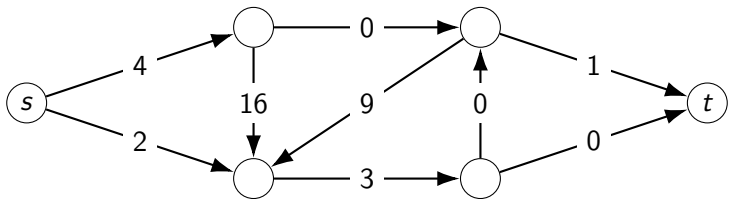
# Algorithme de Ford-Fulkerson



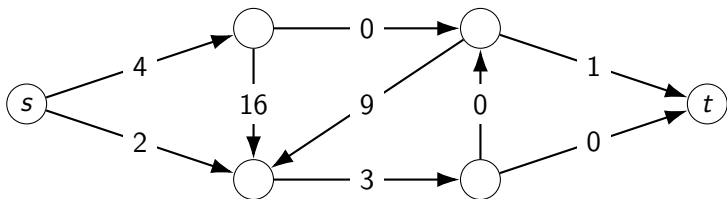
# Algorithme de Ford-Fulkerson



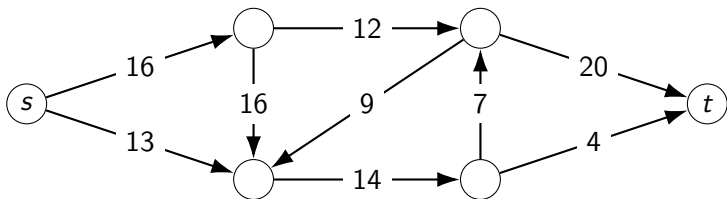
# Algorithme de Ford-Fulkerson



# Algorithme de Ford-Fulkerson



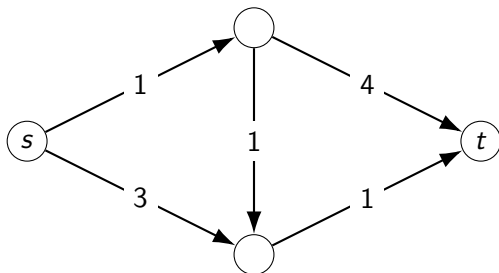
La comparaison avec le graphe initial permet de connaître le flot sur chaque arc :



Valeur du flot obtenu : 23

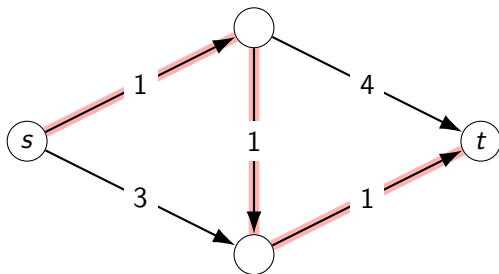
# Algorithme de Ford-Fulkerson

Lorsqu'on ajoute du flot sur un arc, on crée dans le graphe résiduel un arc dans le sens inverse (un **arc arrière**) dans la direction opposée :



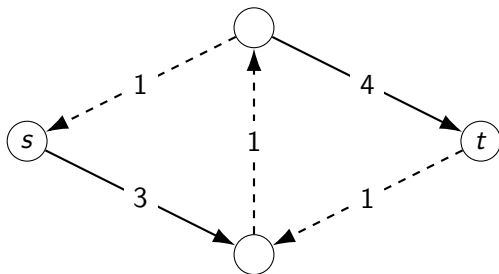
# Algorithme de Ford-Fulkerson

Lorsqu'on ajoute du flot sur un arc, on crée dans le graphe résiduel un arc dans le sens inverse (un **arc arrière**) dans la direction opposée :



# Algorithme de Ford-Fulkerson

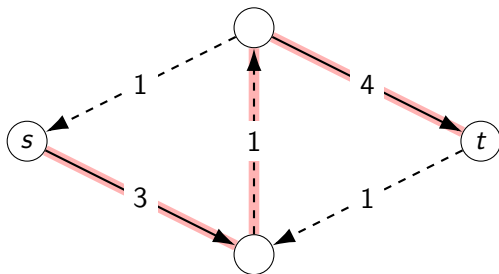
Lorsqu'on ajoute du flot sur un arc, on crée dans le graphe résiduel un arc dans le sens inverse (un **arc arrière**) dans la direction opposée :





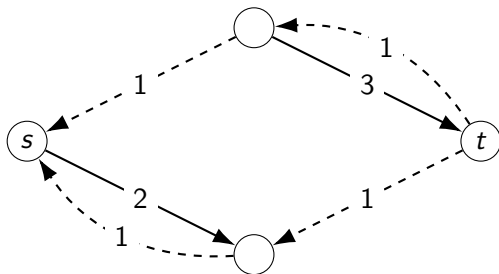
# Algorithme de Ford-Fulkerson

Lorsqu'on ajoute du flot sur un arc, on crée dans le graphe résiduel un arc dans le sens inverse (un **arc arrière**) dans la direction opposée :



# Algorithme de Ford-Fulkerson

Lorsqu'on ajoute du flot sur un arc, on crée dans le graphe résiduel un arc dans le sens inverse (un **arc arrière**) dans la direction opposée :



# Algorithme de Ford-Fulkerson

## Théorème

Si les capacités initiales sont toutes entières, l'algorithme de Ford-Fulkerson termine (ne fait pas boucle infinie).

Preuve :

# Algorithme de Ford-Fulkerson

## Théorème

Si les capacités initiales sont toutes entières, l'algorithme de Ford-Fulkerson termine (ne fait pas boucle infinie).

Preuve :

- Les capacités résiduelles restent toujours entières

# Algorithme de Ford-Fulkerson

## Théorème

Si les capacités initiales sont toutes entières, l'algorithme de Ford-Fulkerson termine (ne fait pas boucle infinie).

Preuve :

- Les capacités résiduelles restent toujours entières
- À chaque itération, on diminue au moins de 1 la somme de toutes les capacités résiduelles. Donc il ne peut pas y avoir plus de  $c(\vec{E})$  itérations.

# Algorithme de Ford-Fulkerson

## Théorème

Si les capacités initiales sont toutes entières, l'algorithme de Ford-Fulkerson termine (ne fait pas boucle infinie).

Preuve :

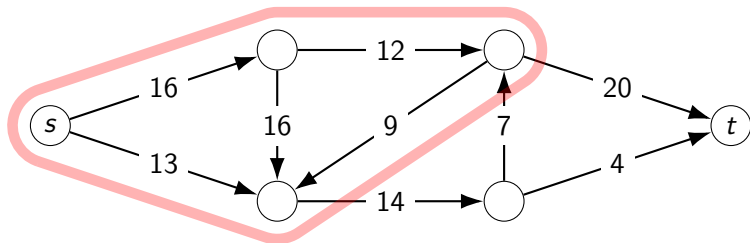
- Les capacités résiduelles restent toujours entières
- À chaque itération, on diminue au moins de 1 la somme de toutes les capacités résiduelles. Donc il ne peut pas y avoir plus de  $c(\vec{E})$  itérations.

On a de plus un majorant grossier de la complexité :  $O(c(\vec{E}))$  ou encore  $O(|f^*|)$ , où  $f^*$  est un flot de valeur maximum.

La complexité plus précise dépend de la façon dont on choisit les chemins.

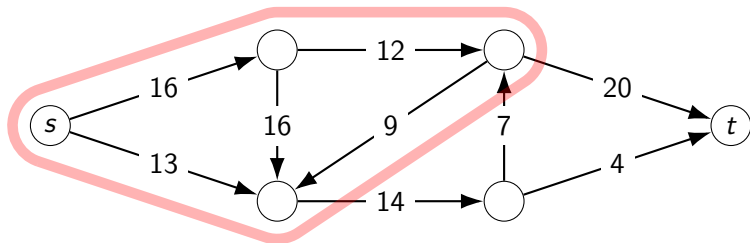
## Définition

Une **coupe** de  $\vec{G}$  est un ensemble  $S \subseteq V$  contenant  $s$  mais pas  $t$ .



## Définition

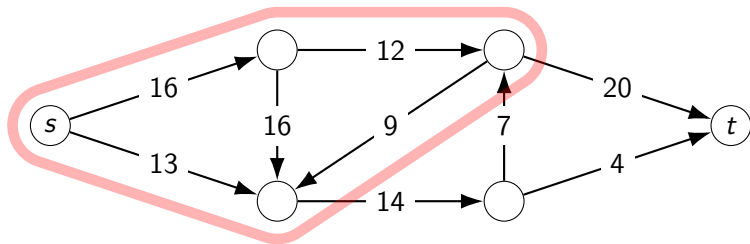
La capacité d'une coupe  $S$  est la somme  $c(S^+)$  des capacités des arcs sortant de  $S$  :





## Définition

La capacité d'une coupe  $S$  est la somme  $c(S^+)$  des capacités des arcs sortant de  $S$  :

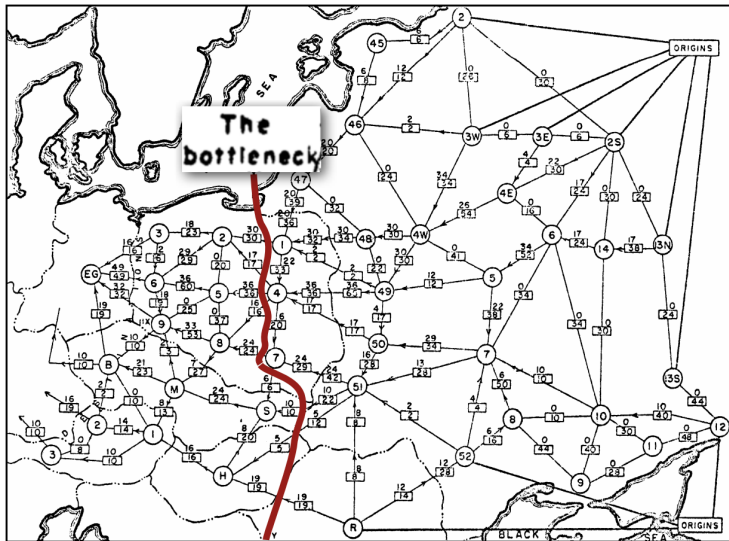


La capacité de cette coupe est  $20 + 14 = 34$ .

## Problème

Trouver une coupe de capacité minimum dans un graphe.

# Coupes

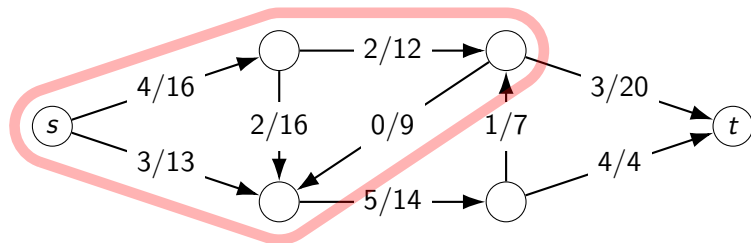


Plan américain de destruction d'un «min cut» des rails soviétiques

## Définition

Le flot sortant d'une coupe  $S$  est définie par :

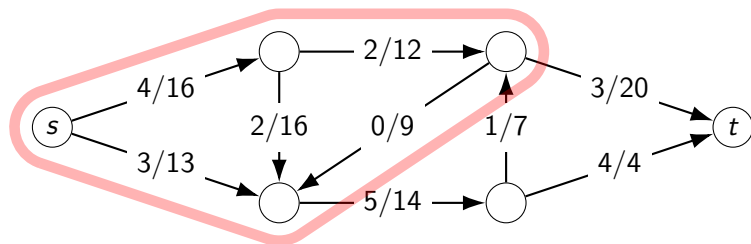
$$f(S) = |f| - f(S^-)$$



## Définition

Le flot sortant d'une coupe  $S$  est définie par :

$$f(S) = |f| - f(S^-)$$



Le flot sortant de cette coupe est  $5 + 3 - 1 = 7$ .

## Lemme 1

Si  $S$  est une coupe,  $f(S) \leq c(S^+)$ .

## Lemme 1

Si  $S$  est une coupe,  $f(S) \leq c(S^+)$ .

Preuve :

$$f(S) = |f| - f(S^-) \leq |f| \leq c(S^+)$$

## Lemme 2

Si  $S$  est une coupe,  $f(S) = |f|$ .

## Lemme 2

Si  $S$  est une coupe,  $f(S) = |f|$ .

Preuve :

Soit  $H_k$  : « Si  $S$  est une coupe avec  $k$  sommets alors  $f(S) = |f|$  ».



## Lemme 2

Si  $S$  est une coupe,  $f(S) = |f|$ .

Preuve :

Soit  $H_k$  : « Si  $S$  est une coupe avec  $k$  sommets alors  $f(S) = |f|$  ».

- $H_1$  est vrai car alors  $S = \{s\}$ .

## Lemme 2

Si  $S$  est une coupe,  $f(S) = |f|$ .

Preuve :

Soit  $H_k$  : « Si  $S$  est une coupe avec  $k$  sommets alors  $f(S) = |f|$  ».

- $H_1$  est vrai car alors  $S = \{s\}$ .
- Supposons  $H_k$  vraie pour un  $k \geq 1$ .

## Lemme 2

Si  $S$  est une coupe,  $f(S) = |f|$ .

Preuve :

Soit  $H_k$  : « Si  $S$  est une coupe avec  $k$  sommets alors  $f(S) = |f|$  ».

- $H_1$  est vrai car alors  $S = \{s\}$ .
- Supposons  $H_k$  vraie pour un  $k \geq 1$ .  
Soit  $S$  une coupe avec  $k + 1$  sommets.

## Lemme 2

Si  $S$  est une coupe,  $f(S) = |f|$ .

Preuve :

Soit  $H_k$  : « Si  $S$  est une coupe avec  $k$  sommets alors  $f(S) = |f|$  ».

- $H_1$  est vrai car alors  $S = \{s\}$ .
- Supposons  $H_k$  vraie pour un  $k \geq 1$ .  
Soit  $S$  une coupe avec  $k + 1$  sommets.  
Soit  $v \in S$ . Alors, en utilisant la définition :  
$$f(S) = f(S - \{v\}) + f(\{v\}).$$

## Lemme 2

Si  $S$  est une coupe,  $f(S) = |f|$ .

Preuve :

Soit  $H_k$  : « Si  $S$  est une coupe avec  $k$  sommets alors  $f(S) = |f|$  ».

- $H_1$  est vrai car alors  $S = \{s\}$ .
- Supposons  $H_k$  vraie pour un  $k \geq 1$ .

Soit  $S$  une coupe avec  $k + 1$  sommets.

Soit  $v \in S$ . Alors, en utilisant la définition :

$$f(S) = f(S - \{v\}) + f(\{v\}).$$

Or  $f(S \setminus \{v\}) = |f|$  par hypothèse de récurrence et  $f(\{v\}) = 0$ .

## Lemme 2

Si  $S$  est une coupe,  $f(S) = |f|$ .

Preuve :

Soit  $H_k$  : « Si  $S$  est une coupe avec  $k$  sommets alors  $f(S) = |f|$  ».

- $H_1$  est vrai car alors  $S = \{s\}$ .
- Supposons  $H_k$  vraie pour un  $k \geq 1$ .

Soit  $S$  une coupe avec  $k + 1$  sommets.

Soit  $v \in S$ . Alors, en utilisant la définition :

$$f(S) = f(S - \{v\}) + f(\{v\}).$$

Or  $f(S \setminus \{v\}) = |f|$  par hypothèse de récurrence et  $f(\{v\}) = 0$ .

Donc  $f(S) = |f|$  :  $H_{k+1}$  est vraie.

## Lemme 1

Si  $f$  est un flot et  $S$  une coupe,  $f(S) \leq c(S^+)$ .

## Lemme 2

Si  $f$  est un flot et  $S$  une coupe,  $f(S) = |f|$ .

## Théorème max flow - min cut

Si un flot  $f$  et une coupe  $S$  vérifient  $f(S) = c(S)$  alors :

- $f$  est un flot de valeur maximum
- $S$  une coupe de capacité minimum

Preuve :

## Lemme 1

Si  $f$  est un flot et  $S$  une coupe,  $f(S) \leq c(S^+)$ .

## Lemme 2

Si  $f$  est un flot et  $S$  une coupe,  $f(S) = |f|$ .

## Théorème max flow - min cut

Si un flot  $f$  et une coupe  $S$  vérifient  $f(S) = c(S)$  alors :

- $f$  est un flot de valeur maximum
- $S$  une coupe de capacité minimum

Preuve : Soit  $f^*$  un flot de valeur maximum.



## Lemme 1

Si  $f$  est un flot et  $S$  une coupe,  $f(S) \leq c(S^+)$ .

## Lemme 2

Si  $f$  est un flot et  $S$  une coupe,  $f(S) = |f|$ .

## Théorème max flow - min cut

Si un flot  $f$  et une coupe  $S$  vérifient  $f(S) = c(S)$  alors :

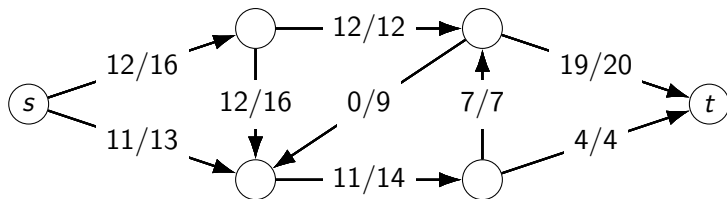
- $f$  est un flot de valeur maximum
- $S$  une coupe de capacité minimum

Preuve : Soit  $f^*$  un flot de valeur maximum.

$$f^*(s^+) \stackrel{2}{=} f^*(S) \stackrel{1}{\leq} c(S) = f(S) \stackrel{2}{=} |f|$$

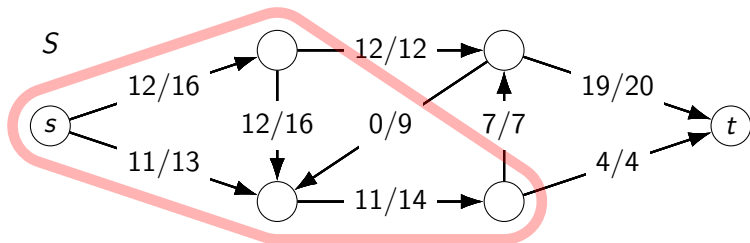
## Question

Comment prouver que ce flot est maximum ?



## Question

Comment prouver que ce flot est maximum ?



$$c(S) = 23 = f(S)$$

Donc  $f$  est un flot maximum et  $S$  une coupe minimum.

## Théorème

Si l'algorithme de Ford-Fulkerson termine, le flot obtenu est un flot maximum

Preuve :

## Théorème

Si l'algorithme de Ford-Fulkerson termine, le flot obtenu est un flot maximum

Preuve :

Soit  $S$  l'ensemble des sommets accessibles depuis  $s$  dans le graphe résiduel.

## Théorème

Si l'algorithme de Ford-Fulkerson termine, le flot obtenu est un flot maximum

Preuve :

Soit  $S$  l'ensemble des sommets accessibles depuis  $s$  dans le graphe résiduel.

Tout arc  $\vec{e}$  sortant de  $S$  a une capacité résiduelle nulle, donc  $c(\vec{e}) = f(\vec{e})$ . D'où :

$$c(S) = \sum_{\vec{e} \in \vec{E}} c(\vec{e}) = \sum_{\vec{e} \in \vec{E}} f(\vec{e}) = f(S)$$

## Décomposition de flot

Le flot renvoyé par l'algorithme de Ford-Fulkerson est une somme de chemins.

# Décomposition de flot

Le flot renvoyé par l'algorithme de Ford-Fulkerson est une somme de chemins. Inversement, tout flot peut s'obtenir comme somme de chemins :

## Théorème de décomposition de flot

Tout  $s - t$  flot peut se décomposer comme une somme de chemins de  $s$  à  $t$  et de cycles.

De plus, le nombre de chemins et cycles est au plus  $p$ .

Preuve :



# Décomposition de flot

Le flot renvoyé par l'algorithme de Ford-Fulkerson est une somme de chemins. Inversement, tout flot peut s'obtenir comme somme de chemins :

## Théorème de décomposition de flot

Tout  $s - t$  flot peut se décomposer comme une somme de chemins de  $s$  à  $t$  et de cycles.

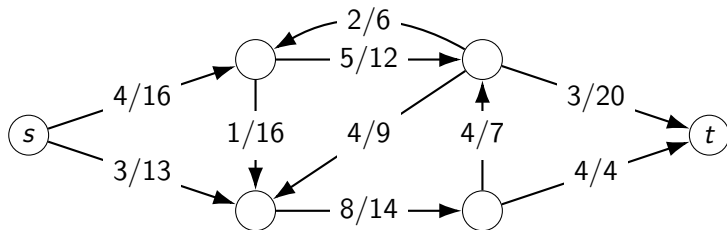
De plus, le nombre de chemins et cycles est au plus  $p$ .

Preuve : « Ford-Fulkerson à l'envers »

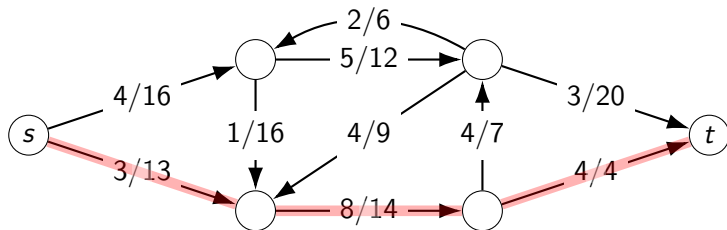
Tant que possible : on cherche un chemin de  $s$  à  $t$  en utilisant seulement les arêtes ayant un flot positif, puis on retire la capacité minimum le long de ce chemin.

Les propriétés de flot nous assurent que quand il n'y a plus de chemin, il ne reste que des cycles.

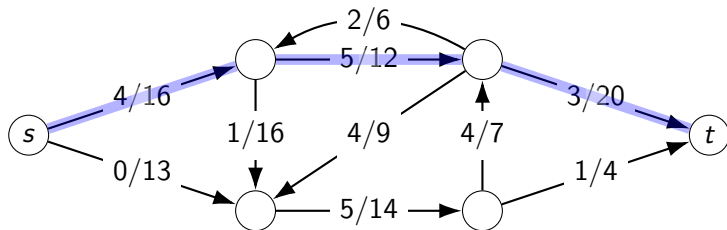
# Décomposition de flot



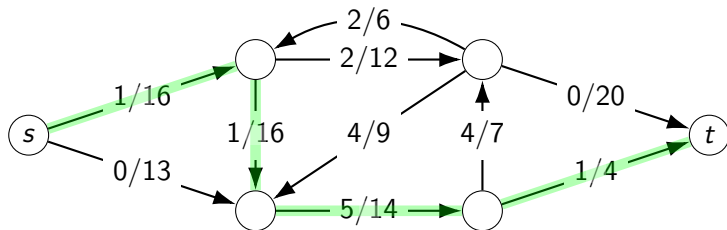
# Décomposition de flot



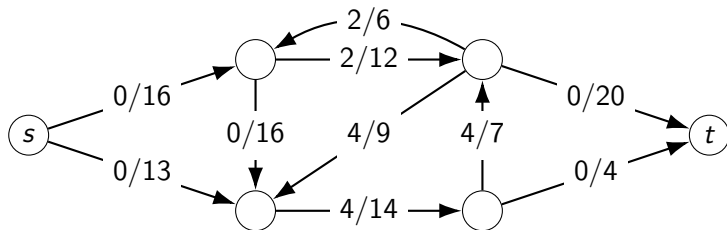
# Décomposition de flot



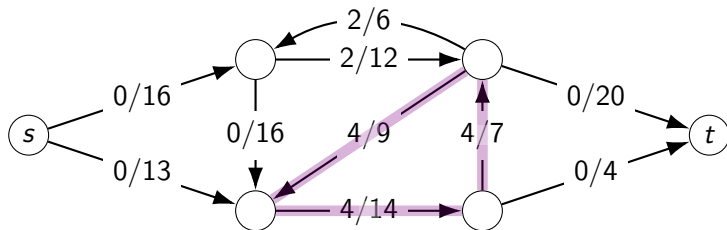
# Décomposition de flot



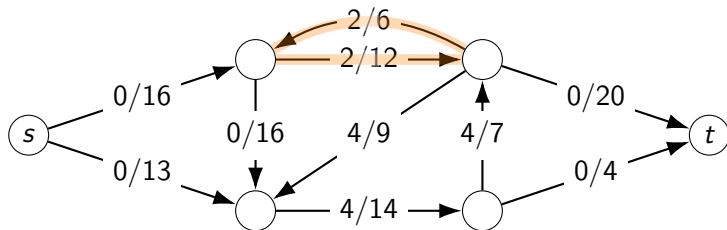
# Décomposition de flot



# Décomposition de flot

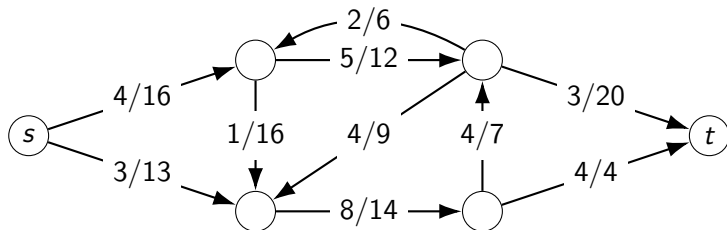


# Décomposition de flot



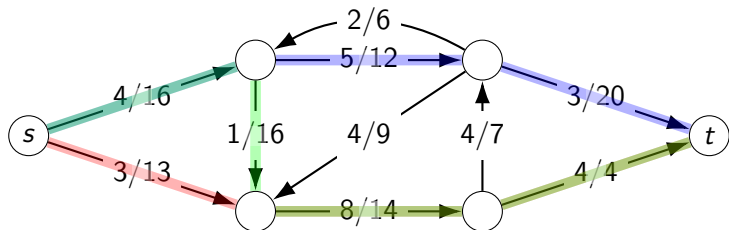


# Décomposition de flot



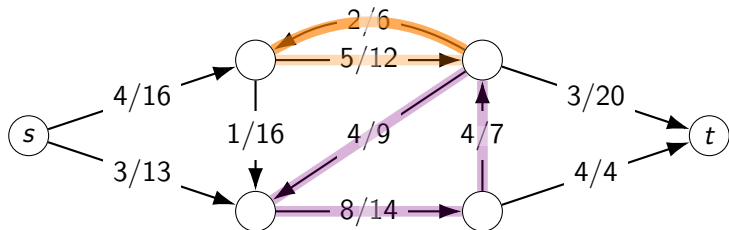
On a décomposé le flot en

# Décomposition de flot



On a décomposé le flot en 3 chemins

# Décomposition de flot



On a décomposé le flot en 3 chemins et 2 cycles

# Choix des chemins

On va voir plusieurs façon de trouver un chemin de  $s$  à  $t$  :

On va voir plusieurs façon de trouver un chemin de  $s$  à  $t$  :

- Parcours en profondeur
- Parcours en largeur (plus court chemin)
- Plus large chemin

## Exercice

Écrire une fonction Python `ford_fulkerson` telle que `ford_fulkerson(G, s, t, path)` renvoie la valeur maximum d'un flot, où :

- `G` est un graphe orienté avec une capacité sur les arcs.
- `path` est une fonction qui à un graphe résiduel associe un chemin de `s` à `t`

## Choix des chemins : parcours en profondeur

### Exercice

Implémenter le parcours en profondeur et l'utiliser avec `ford_fulkerson`.

Nombre d'itérations maximum de Ford-Fulkerson avec recherche des chemins par parcours en profondeur :  $|f^*|$ .

# Choix des chemins : parcours en profondeur

## Exercice

Implémenter le parcours en profondeur et l'utiliser avec `ford_fulkerson`.

Nombre d'itérations maximum de Ford-Fulkerson avec recherche des chemins par parcours en profondeur :  $|f^*|$ .

On note  $p = |\vec{E}|$  le nombre d'arcs et  $n = |V|$  le nombre de sommets.

## Théorème

La complexité de Ford-Fulkerson avec recherche des chemins par parcours en profondeur est  $O(p|f^*|)$

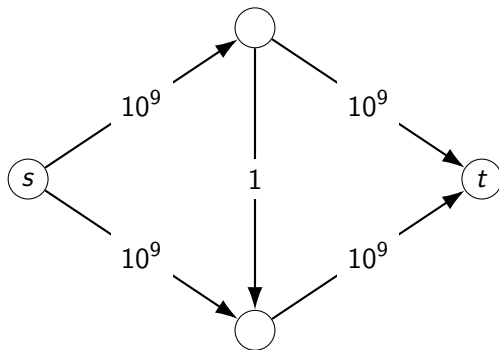


## Choix des chemins : parcours en profondeur

La complexité  $O(p \times |f^*|)$  est atteinte pour ce genre de graphe :

## Choix des chemins : parcours en profondeur

La complexité  $O(p \times |f^*|)$  est atteinte pour ce genre de graphe :



## Choix des chemins : plus large chemin

Il est logique de chercher à chaque itération le **chemin le plus large** (widest path), c'est à dire celui dont la capacité minimum est la plus grande (donc qui augmentera plus le flot).

## Choix des chemins : plus large chemin

Il est logique de chercher à chaque itération le **chemin le plus large** (widest path), c'est à dire celui dont la capacité minimum est la plus grande (donc qui augmentera plus le flot).

On peut trouver tous les plus larges chemins depuis  $s$  de différentes façons :

- Algorithme de Prim modifié en  $O(p \log(p))$
- Algorithme de Dijkstra modifié en  $O(p \log(p))$

# Choix des chemins : plus large chemin

## Algorithme de Prim modifié

$T \leftarrow \{s\}$

Tant que  $T$  n'est pas couvrant :  
Ajouter l'arête sortante de  $T$  de capacité maximum

## Théorème

$T$  est un « arbre des plus larges chemins » : le chemin de  $s$  à un autre sommet  $v$  dans  $T$  est un plus large chemin de  $s$  à  $v$

Preuve :

# Choix des chemins : plus large chemin

## Algorithme de Prim modifié

$T \leftarrow \{s\}$

Tant que  $T$  n'est pas couvrant :

Ajouter l'arête sortante de  $T$  de capacité maximum

## Théorème

$T$  est un « arbre des plus larges chemins » : le chemin de  $s$  à un autre sommet  $v$  dans  $T$  est un plus large chemin de  $s$  à  $v$

Preuve : Par récurrence sur  $|T|$ .

Quand on ajoute une arête vers  $v$ , d'après la façon de la choisir, il ne peut pas y avoir d'autres chemin plus large vers  $v$ .

## Choix des chemins : plus large chemin

### Algorithme de Prim modifié

$T \leftarrow \{s\}$

Tant que  $T$  n'est pas couvrant :

Ajouter l'arête sortante de  $T$  de capacité maximum

Complexité :  $O(p \log(p))$  en utilisant une file de priorité contenant les arêtes, avec extraction et ajout en complexité logarithmique

## Choix des chemins : plus large chemin

Soit  $f^*$  un flot maximum. D'après le théorème de décomposition,  $f^*$  s'écrit comme une somme d'au plus  $p$  chemins et cycles.



## Choix des chemins : plus large chemin

Soit  $f^*$  un flot maximum. D'après le théorème de décomposition,  $f^*$  s'écrit comme une somme d'au plus  $p$  chemins et cycles.

Au moins un de ces chemins transporte un flot  $\geq \frac{|f^*|}{p}$ .

## Choix des chemins : plus large chemin

Soit  $f^*$  un flot maximum. D'après le théorème de décomposition,  $f^*$  s'écrit comme une somme d'au plus  $p$  chemins et cycles.

Au moins un de ces chemins transporte un flot  $\geq \frac{|f^*|}{p}$ .

Donc le chemin de largeur maximum permet d'augmenter le flot d'au moins  $\frac{|f^*|}{p}$

## Choix des chemins : plus large chemin

Soit  $f^*$  un flot maximum. D'après le théorème de décomposition,  $f^*$  s'écrit comme une somme d'au plus  $p$  chemins et cycles.

Au moins un de ces chemins transporte un flot  $\geq \frac{|f^*|}{p}$ .

Donc le chemin de largeur maximum permet d'augmenter le flot d'au moins  $\frac{|f^*|}{p}$

La valeur du flot maximum dans le graphe résiduel passe alors de  $|f^*|$  à au plus  $|f^*| - \frac{|f^*|}{p} = |f^*|(1 - \frac{1}{p})$ .

## Choix des chemins : plus large chemin

Soit  $f^*$  un flot maximum. D'après le théorème de décomposition,  $f^*$  s'écrit comme une somme d'au plus  $p$  chemins et cycles.

Au moins un de ces chemins transporte un flot  $\geq \frac{|f^*|}{p}$ .

Donc le chemin de largeur maximum permet d'augmenter le flot d'au moins  $\frac{|f^*|}{p}$

La valeur du flot maximum dans le graphe résiduel passe alors de  $|f^*|$  à au plus  $|f^*| - \frac{|f^*|}{p} = |f^*|(1 - \frac{1}{p})$ .

Après  $p \ln(|f^*|)$  itérations, il reste donc un flot maximum d'au plus :

$$|f^*|(1 - \frac{1}{p})^{p \ln(|f^*|)} < |f^*|e^{-\ln(|f^*|)} = 1$$

## Choix des chemins : plus large chemin

Le nombre d'itération de Ford-Fulkerson avec recherche de plus large chemin est donc au plus  $p \ln(|f^*|)$ .

Comme chaque recherche de plus large chemin demande  $O(p \log(p))$  avec Prim :

### Théorème

La complexité de de Ford-Fulkerson avec recherche de plus large chemin est  $O(p^2 \log(p) \ln(|f^*|))$ .

## Choix des chemins : plus large chemin

Le nombre d'itération de Ford-Fulkerson avec recherche de plus large chemin est donc au plus  $p \ln(|f^*|)$ .

Comme chaque recherche de plus large chemin demande  $O(p \log(p))$  avec Prim :

### Théorème

La complexité de de Ford-Fulkerson avec recherche de plus large chemin est  $O(p^2 \log(p) \ln(|f^*|))$ .

Mieux qu'avec le parcours en profondeur en  $O(p|f^*|)$  mais il serait préférable de ne pas avoir cette dépendance en  $|f^*|$ .

## Choix des chemins : parcours en largeur (Edmond-Karp)

On note  $p = |\vec{E}|$  le nombre d'arcs et  $n = |V|$  le nombre de sommets.

### Théorème

Ford-Fulkerson avec recherche de plus court chemins (par parcours en largeur) termine en au plus  $\frac{np}{2}$  itérations.

## Choix des chemins : parcours en largeur (Edmond-Karp)

On note  $p = |\vec{E}|$  le nombre d'arcs et  $n = |V|$  le nombre de sommets.

### Théorème

Ford-Fulkerson avec recherche de plus court chemins (par parcours en largeur) termine en au plus  $\frac{np}{2}$  itérations.

Comme chaque parcours en largeur est en  $O(p)$  :

### Théorème

Ford-Fulkerson avec recherche de plus court chemins (par parcours en largeur) est en  $O(np^2)$ .



## Choix des chemins : résumé

Nombre d'itérations et complexité de Ford-Fulkerson suivant le choix des chemins :

	nombre d'itérations	complexité
parcours en profondeur	$ f^* $	$O(p f^* )$
plus large chemin	$\leq p \ln( f^* )$	$O(p^2 \log(p) \ln( f^* ))$
plus court chemin	$\leq \frac{np}{2}$	$O(np^2)$

$(p = |\vec{E}| \text{ et } n = |V|)$

# Choix des chemins : résumé

Exercice pour le 20 octobre : implémenter Ford-Fulkerson avec la recherche de chemin que vous préférez.

Bonus : faire toutes les recherches de chemins et les comparer sur des gros graphes aléatoires.