Algorithmique des graphes

7 — Flots et applications (1)

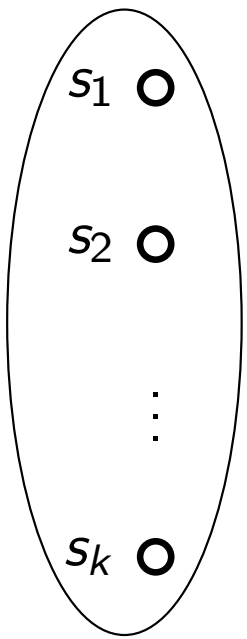
Anthony Labarre

17 mars 2021

Université Gustave Eiffel

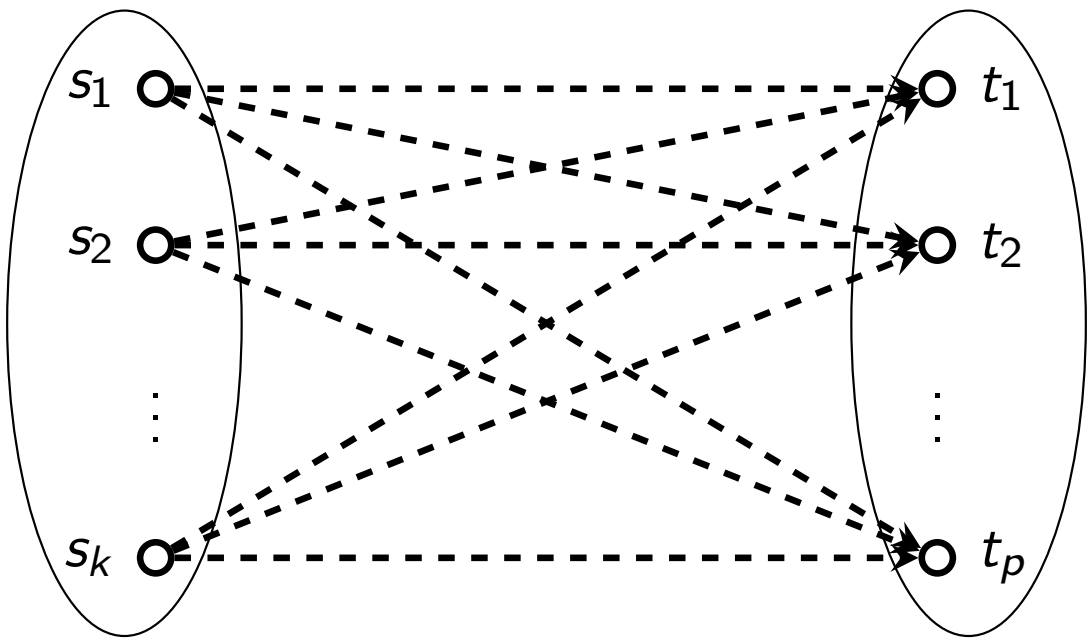
Motivations

- On dispose de *sources* fabriquant divers produits ;



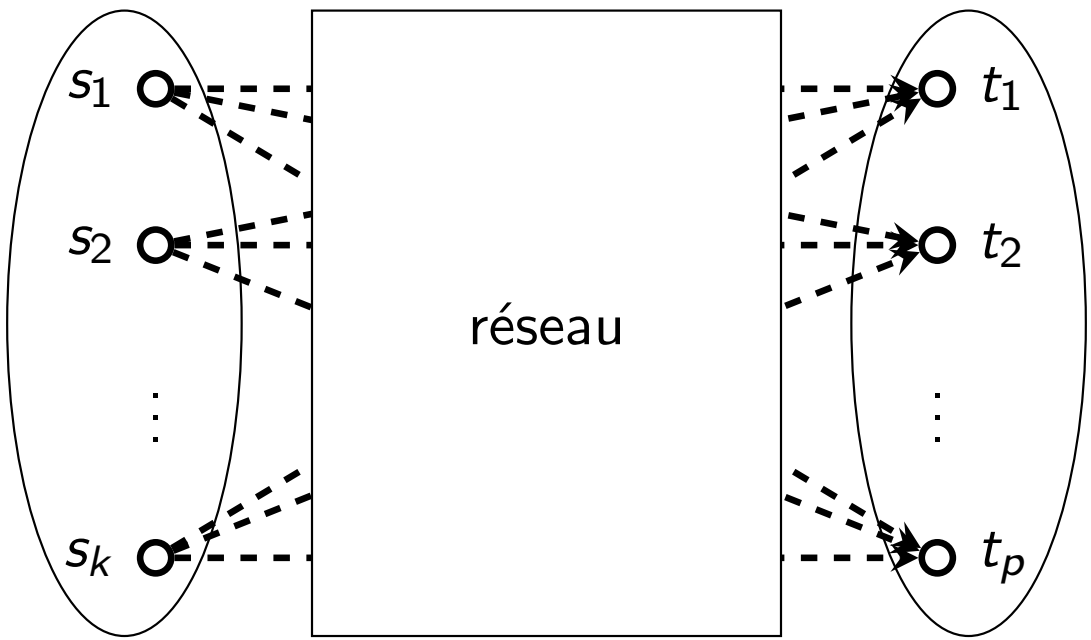
Motivations

- On dispose de *sources* fabriquant divers produits ;
- On veut acheminer ces ressources vers des *puits* ;



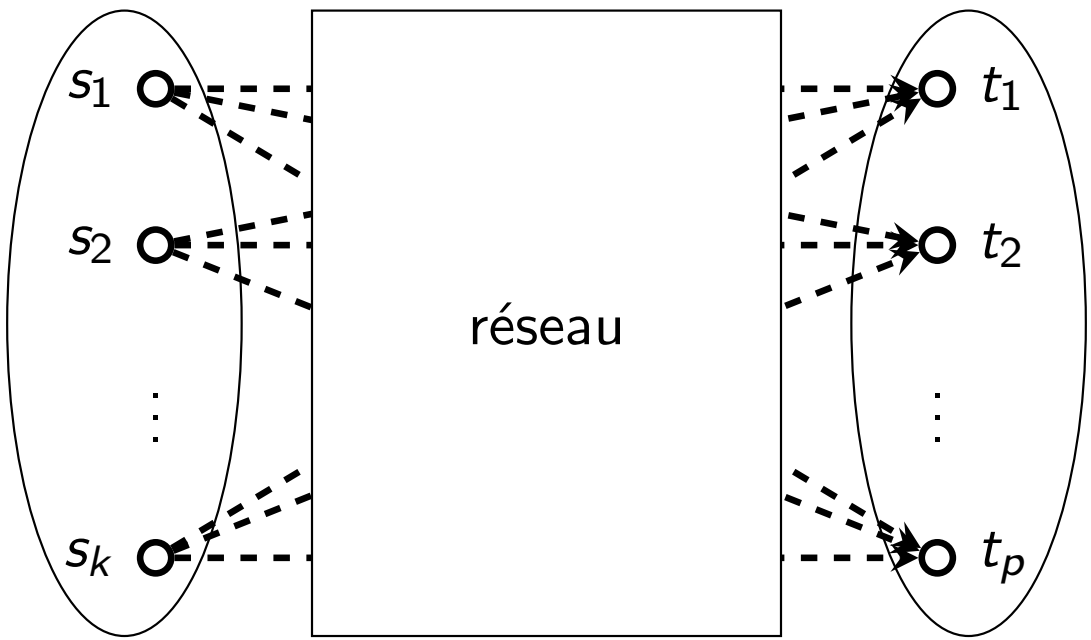
Motivations

- On dispose de *sources* fabriquant divers produits ;
- On veut acheminer ces ressources vers des *puits* ;
- On doit emprunter un réseau de transport dont les capacités sont limitées ;



Motivations

- On dispose de *sources* fabriquant divers produits ;
- On veut acheminer ces ressources vers des *puits* ;
- On doit emprunter un réseau de transport dont les capacités sont limitées ;
- On veut maximiser la quantité de ressources acheminées dans le réseau ;



Applications

- Le problème de flot maximum a des applications directes :

Applications

- Le problème de flot maximum a des applications directes :
  - acheminement de produits vers des destinations en maximisant la quantité, le profit, ...

Applications

- Le problème de flot maximum a des applications directes :
  - acheminement de produits vers des destinations en maximisant la quantité, le profit, ...
  - optimisation du trafic routier (cf. Waze) ;

Applications

- Le problème de flot maximum a des applications directes :
  - acheminement de produits vers des destinations en maximisant la quantité, le profit, ...
  - optimisation du trafic routier (cf. Waze) ;
- ... et un peu moins directes, développées plus loin :

Applications

- Le problème de flot maximum a des applications directes :
  - acheminement de produits vers des destinations en maximisant la quantité, le profit, ...
  - optimisation du trafic routier (cf. Waze) ;
- ... et un peu moins directes, développées plus loin :
  - calcul de chemins disjoints ;

Applications

- Le problème de flot maximum a des applications directes :
  - acheminement de produits vers des destinations en maximisant la quantité, le profit, ...
  - optimisation du trafic routier (cf. Waze) ;
- ... et un peu moins directes, développées plus loin :
  - calcul de chemins disjoints ;
  - calcul de couplages dans un graphe biparti ;

Réseaux de flot

Un **réseau de flot** est un graphe orienté et pondéré $G = (V, A, c)$ dont chaque arc (u, v) possède une **capacité** $c(u, v) \geq 0$, avec :

Réseaux de flot

Un **réseau de flot** est un graphe orienté et pondéré $G = (V, A, c)$ dont chaque arc (u, v) possède une **capacité** $c(u, v) \geq 0$, avec :

- une **source** s (un sommet de degré entrant nul) ;

Réseaux de flot

Un **réseau de flot** est un graphe orienté et pondéré $G = (V, A, c)$ dont chaque arc (u, v) possède une **capacité** $c(u, v) \geq 0$, avec :

- une **source** s (un sommet de degré entrant nul) ;
- un **puits** t (un sommet de degré sortant nul).

Réseaux de flot

Un **réseau de flot** est un graphe orienté et pondéré $G = (V, A, c)$ dont chaque arc (u, v) possède une **capacité** $c(u, v) \geq 0$, avec :

- une **source** s (un sommet de degré entrant nul) ;
- un **puits** t (un sommet de degré sortant nul).

On suppose que tous les autres sommets appartiennent à un chemin de s à t.

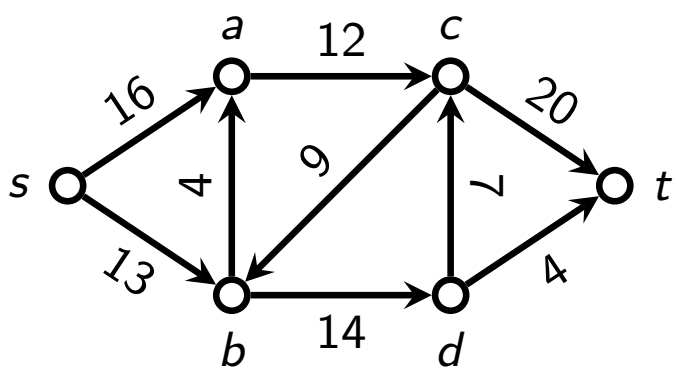
Réseaux de flot

Un **réseau de flot** est un graphe orienté et pondéré $G = (V, A, c)$ dont chaque arc (u, v) possède une **capacité** $c(u, v) \geq 0$, avec :

- une **source** s (un sommet de degré entrant nul) ;
- un **puits** t (un sommet de degré sortant nul).

On suppose que tous les autres sommets appartiennent à un chemin de s à t.

Exemple 1



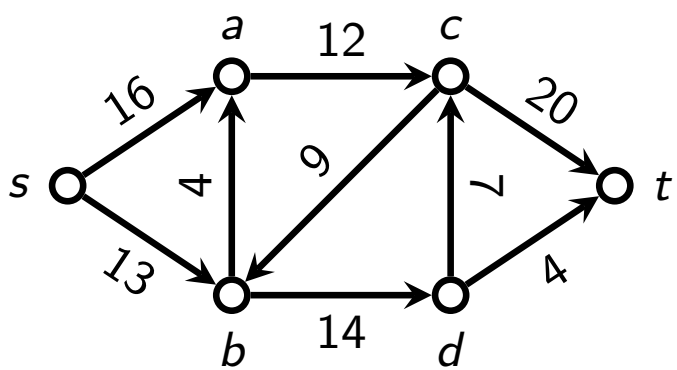
Réseaux de flot

Un **réseau de flot** est un graphe orienté et pondéré $G = (V, A, c)$ dont chaque arc (u, v) possède une **capacité** $c(u, v) \geq 0$, avec :

- une **source** s (un sommet de degré entrant nul) ;
- un **puits** t (un sommet de degré sortant nul).

On suppose que tous les autres sommets appartiennent à un chemin de s à t.

Exemple 1



Hypothèses supplémentaires :

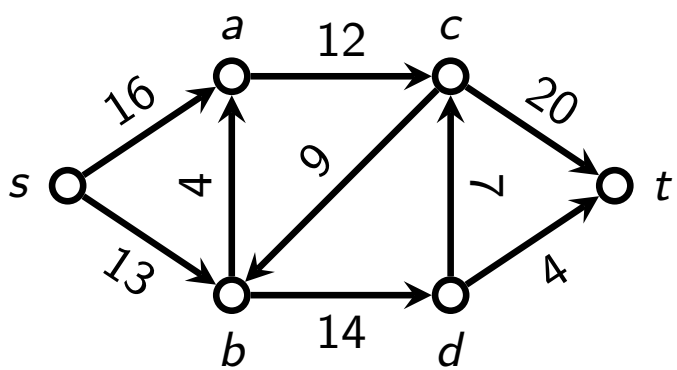
Réseaux de flot

Un **réseau de flot** est un graphe orienté et pondéré $G = (V, A, c)$ dont chaque arc (u, v) possède une **capacité** $c(u, v) \geq 0$, avec :

- une **source** s (un sommet de degré entrant nul) ;
- un **puits** t (un sommet de degré sortant nul).

On suppose que tous les autres sommets appartiennent à un chemin de s à t.

Exemple 1



Hypothèses supplémentaires :

1. toutes les *capacités* et tous les *flots* sont entiers ;

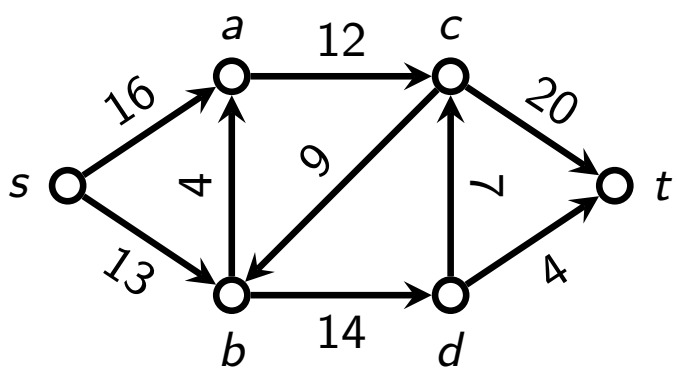
Réseaux de flot

Un **réseau de flot** est un graphe orienté et pondéré $G = (V, A, c)$ dont chaque arc (u, v) possède une **capacité** $c(u, v) \geq 0$, avec :

- une **source** s (un sommet de degré entrant nul) ;
- un **puits** t (un sommet de degré sortant nul).

On suppose que tous les autres sommets appartiennent à un chemin de s à t.

Exemple 1



Hypothèses supplémentaires :

1. toutes les *capacités* et tous les *flots* sont entiers ;
2. les arcs du réseau n'existent que dans un seul sens.

Flots

Un **flot** dans un réseau de flot $G = (V, A, c)$ est une fonction $f : V \times V \to \mathbb{R}$ satisfaisant les propriétés suivantes :

1. $\forall\ u, v \in V : 0 \leq f(u, v) \leq c(u, v)$; contrainte de capacité

Flots

Un **flot** dans un réseau de flot $G = (V, A, c)$ est une fonction $f : V \times V \to \mathbb{R}$ satisfaisant les propriétés suivantes :

1. $\forall\ u, v \in V : 0 \leq f(u, v) \leq c(u, v)$; contrainte de capacité
2. $\forall\ u \in V \setminus \{s, t\} : \sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$; conservation

Flots

Un **flot** dans un réseau de flot $G = (V, A, c)$ est une fonction $f : V \times V \to \mathbb{R}$ satisfaisant les propriétés suivantes :

1. $\forall\ u, v \in V : 0 \leq f(u, v) \leq c(u, v)$; contrainte de capacité
2. $\forall\ u \in V \setminus \{s, t\} : \sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$; conservation
3. $f(u, v) = 0$ si $(u, v) \notin A$.

Flots

Un **flot** dans un réseau de flot $G = (V, A, c)$ est une fonction $f : V \times V \to \mathbb{R}$ satisfaisant les propriétés suivantes :

1. $\forall\ u, v \in V : 0 \leq f(u, v) \leq c(u, v)$; contrainte de capacité
2. $\forall\ u \in V \setminus \{s, t\} : \sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$; conservation
3. $f(u, v) = 0$ si $(u, v) \notin A$.

La **valeur** du flot est $|f| = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t)$.

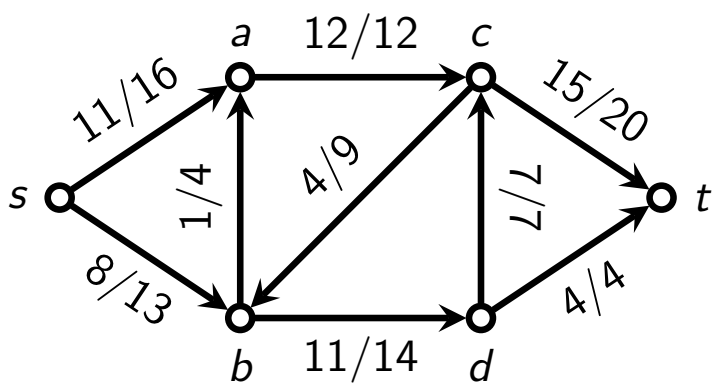
Flots

Un **flot** dans un réseau de flot $G = (V, A, c)$ est une fonction $f : V \times V \to \mathbb{R}$ satisfaisant les propriétés suivantes :

1. $\forall\ u, v \in V : 0 \leq f(u, v) \leq c(u, v)$; contrainte de capacité
2. $\forall\ u \in V \setminus \{s, t\} : \sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$; conservation
3. $f(u, v) = 0$ si $(u, v) \notin A$.

La **valeur** du flot est $|f| = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t)$.

Exemple 2 (x/y : flot x, capacité y)



Conservation

	entrant	sortant
a	12	12
b	12	12
c	19	19
d	11	11
s	0	19
t	19	0

Problème du flot maximum

- Le **problème du flot maximum** est de trouver, pour un réseau de flot G donné, un flot f de valeur maximum, c'est-à-dire que pour tout flot f' sur G, on a $|f| \geq |f'|$.

Problème du flot maximum

- Le **problème du flot maximum** est de trouver, pour un réseau de flot G donné, un flot f de valeur maximum, c'est-à-dire que pour tout flot f' sur G, on a $|f| \geq |f'|$.
- De nombreux algorithmes existent :

Problème du flot maximum

- Le **problème du flot maximum** est de trouver, pour un réseau de flot G donné, un flot f de valeur maximum, c'est-à-dire que pour tout flot f' sur G, on a $|f| \geq |f'|$.
- De nombreux algorithmes existent :
  - la *méthode* de Ford-Fulkerson, et la variante d'Edmonds-Karp ;

Problème du flot maximum

- Le **problème du flot maximum** est de trouver, pour un réseau de flot G donné, un flot f de valeur maximum, c'est-à-dire que pour tout flot f' sur G, on a $|f| \geq |f'|$.
- De nombreux algorithmes existent :
  - la *méthode* de Ford-Fulkerson, et la variante d'Edmonds-Karp ;
  - l'algorithme de Dinitz ;

Problème du flot maximum

- Le **problème du flot maximum** est de trouver, pour un réseau de flot G donné, un flot f de valeur maximum, c'est-à-dire que pour tout flot f' sur G, on a $|f| \geq |f'|$.
- De nombreux algorithmes existent :
  - la *méthode* de Ford-Fulkerson, et la variante d'Edmonds-Karp ;
  - l'algorithme de Dinitz ;
  - les préflots (voir [1]) ;

Problème du flot maximum

- Le **problème du flot maximum** est de trouver, pour un réseau de flot G donné, un flot f de valeur maximum, c'est-à-dire que pour tout flot f' sur G, on a $|f| \geq |f'|$.
- De nombreux algorithmes existent :
  - la *méthode* de Ford-Fulkerson, et la variante d'Edmonds-Karp ;
  - l'algorithme de Dinitz ;
  - les préflots (voir [1]) ;
- Le meilleur à l'heure actuelle est celui de James Orlin [2] (2013), en $O(|V||A|)$;

Présentation générale de la méthode

- La *méthode* de Ford-Fulkerson procède comme suit ; on démarre avec un flot f nul partout pour le réseau G, et à chaque itération :

Présentation générale de la méthode

- La *méthode* de Ford-Fulkerson procède comme suit ; on démarre avec un flot f nul partout pour le réseau G, et à chaque itération :
  1. on cherche un *chemin augmentant* dans un *réseau résiduel* G_f ;

Présentation générale de la méthode

- La *méthode* de Ford-Fulkerson procède comme suit ; on démarre avec un flot f nul partout pour le réseau G, et à chaque itération :
  1. on cherche un *chemin augmentant* dans un *réseau résiduel* G_f ;
  2. on augmente au maximum f sur les arcs de ce chemin ;

Présentation générale de la méthode

- La *méthode* de Ford-Fulkerson procède comme suit ; on démarre avec un flot f nul partout pour le réseau G, et à chaque itération :
  1. on cherche un *chemin augmentant* dans un *réseau résiduel* G_f ;
  2. on augmente au maximum f sur les arcs de ce chemin ;
  3. on met à jour G_f ;

Présentation générale de la méthode

- La *méthode* de Ford-Fulkerson procède comme suit ; on démarre avec un flot f nul partout pour le réseau G, et à chaque itération :
  1. on cherche un *chemin augmentant* dans un *réseau résiduel* G_f ;
  2. on augmente au maximum f sur les arcs de ce chemin ;
  3. on met à jour G_f ;
- La méthode se termine quand il n'existe plus de chemin augmentant ;

Présentation générale de la méthode

- La *méthode* de Ford-Fulkerson procède comme suit ; on démarre avec un flot f nul partout pour le réseau G, et à chaque itération :
  1. on cherche un *chemin augmentant* dans un *réseau résiduel* G_f ;
  2. on augmente au maximum f sur les arcs de ce chemin ;
  3. on met à jour G_f ;
- La méthode se termine quand il n'existe plus de chemin augmentant ;
- Le flot ainsi obtenu est maximum (on le prouvera plus loin) ;

Pseudocode de la méthode

Algorithme 1 : FordFulkerson(G)

Entrées : un réseau de flot G.
Sortie : un flot maximum pour G.

```
1  flot ← tableau associatif (clés = G.arcs(), valeurs = 0);
2  source ← unique sommet de degré entrant nul de G;
3  puits ← unique sommet de degré sortant nul de G;
4  G_f ← G;                          // au départ, réseau = résiduel
5  chemin ← CheminAugmentant(G_f, source, puits);
6  tant que chemin ≠ NIL faire
7  |   AugmenterFlot(flot, chemin);
8  |   MettreAJourRésiduel(G, G_f, chemin.arcs(), flot);
9  |   chemin ← CheminAugmentant(G_f, source, puits);
10 renvoyer flot;
```

Réseaux résiduels

Définition 1

Soit $G = (V, A, c)$ un réseau muni d'un flot f. La **capacité résiduelle** d'une paire de sommets $u, v \in V$ est

$$c_f(u, v) = \begin{cases}$$

Réseaux résiduels

Définition 1

Soit $G = (V, A, c)$ un réseau muni d'un flot f. La **capacité résiduelle** d'une paire de sommets $u, v \in V$ est

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{si } (u, v) \in A, \\ \end{cases}$$

Réseaux résiduels

Définition 1

Soit $G = (V, A, c)$ un réseau muni d'un flot f. La **capacité résiduelle** d'une paire de sommets $u, v \in V$ est

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{si } (u, v) \in A, \\ f(v, u) & \text{si } (v, u) \in A, \\ \end{cases}$$

Réseaux résiduels

Définition 1

Soit $G = (V, A, c)$ un réseau muni d'un flot f. La **capacité résiduelle** d'une paire de sommets $u, v \in V$ est

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{si } (u,v) \in A, \\ f(v,u) & \text{si } (v,u) \in A, \\ 0 & \text{sinon .} \end{cases}$$

Réseaux résiduels

Définition 1

Soit $G = (V, A, c)$ un réseau muni d'un flot f. La **capacité résiduelle** d'une paire de sommets $u, v \in V$ est

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{si } (u,v) \in A, \\ f(v,u) & \text{si } (v,u) \in A, \\ 0 & \text{sinon .} \end{cases}$$

Définition 2

Soit G un réseau muni d'un flot f. Le **réseau résiduel** G_f est le sous-graphe de G induit par les arcs de capacité résiduelle non nulle.

Réseaux résiduels

Définition 1

Soit $G = (V, A, c)$ un réseau muni d'un flot f. La **capacité résiduelle** d'une paire de sommets $u, v \in V$ est

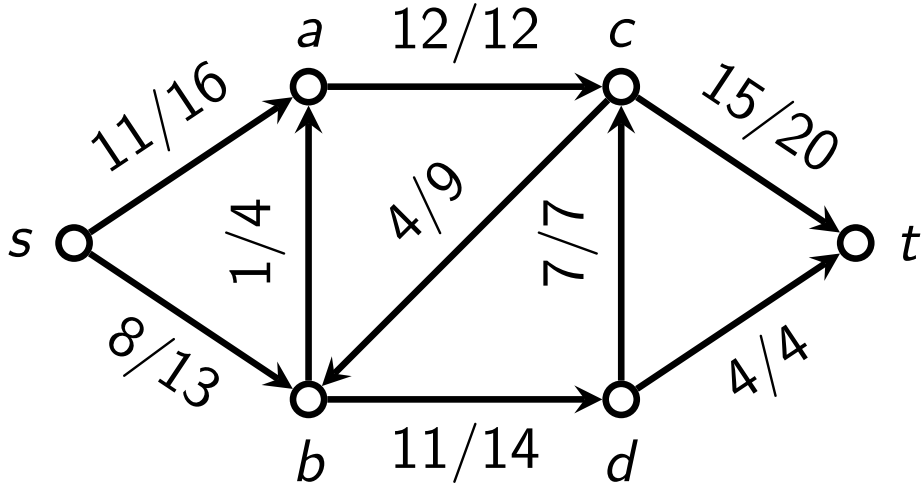
$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{si } (u,v) \in A, \\ f(v,u) & \text{si } (v,u) \in A, \\ 0 & \text{sinon .} \end{cases}$$

Définition 2

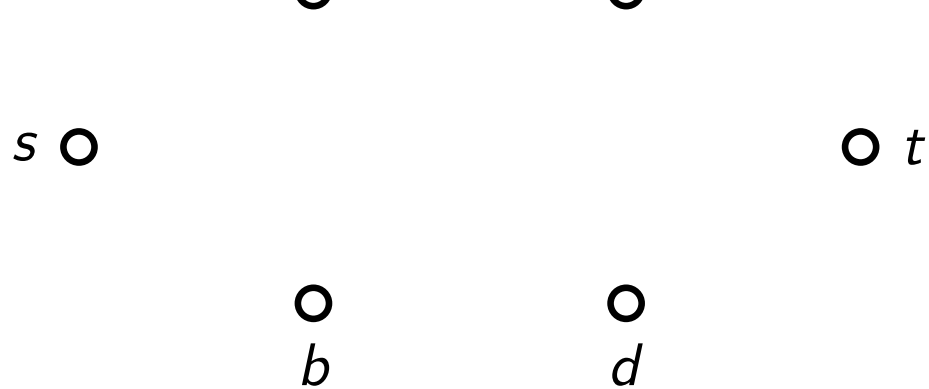
Soit G un réseau muni d'un flot f. Le **réseau résiduel** G_f est le sous-graphe de G induit par les arcs de capacité résiduelle non nulle. Autrement dit, il s'agit du graphe $G_f = (V, A_f, c_f)$ où $A_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$.

Réseaux résiduels : exemple

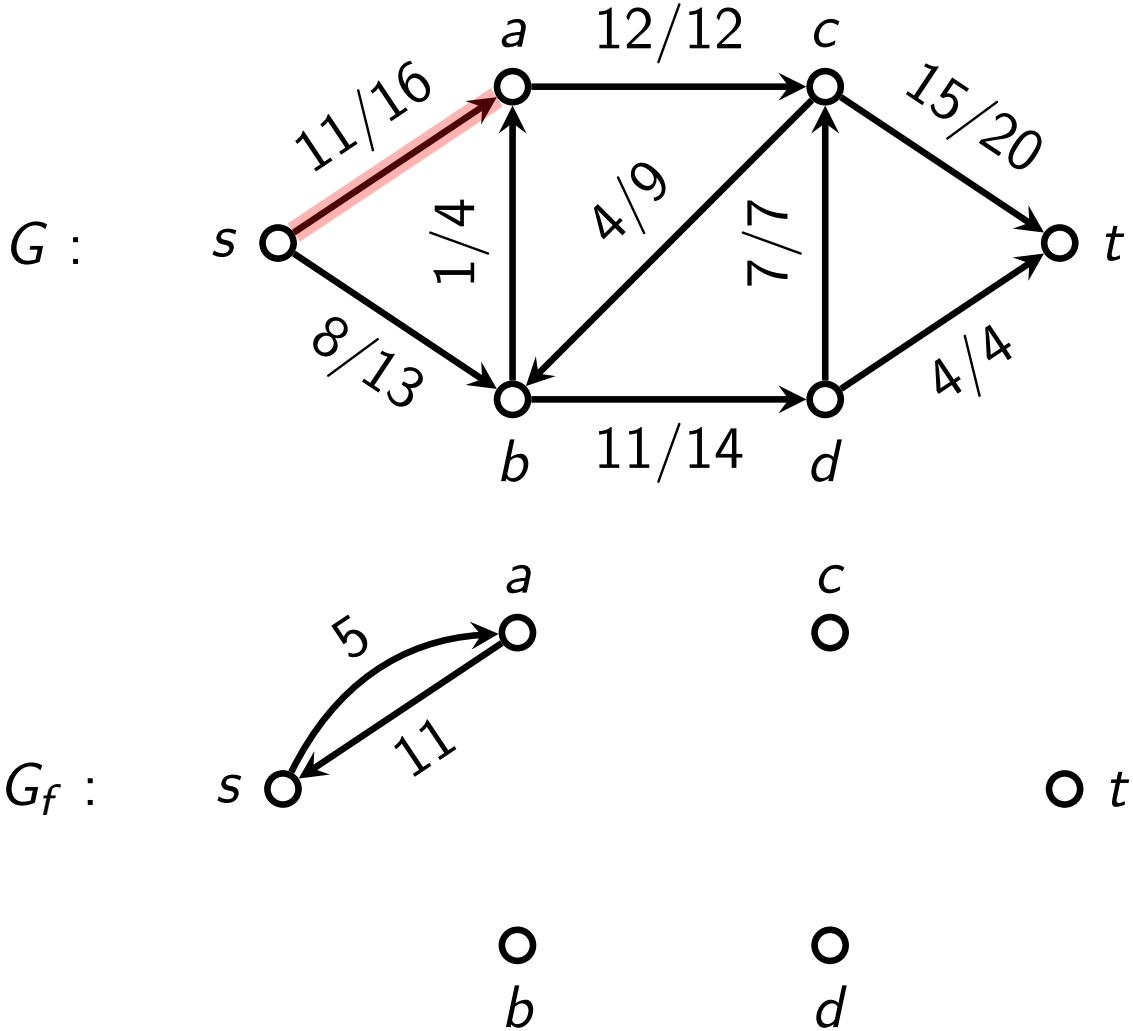
G :



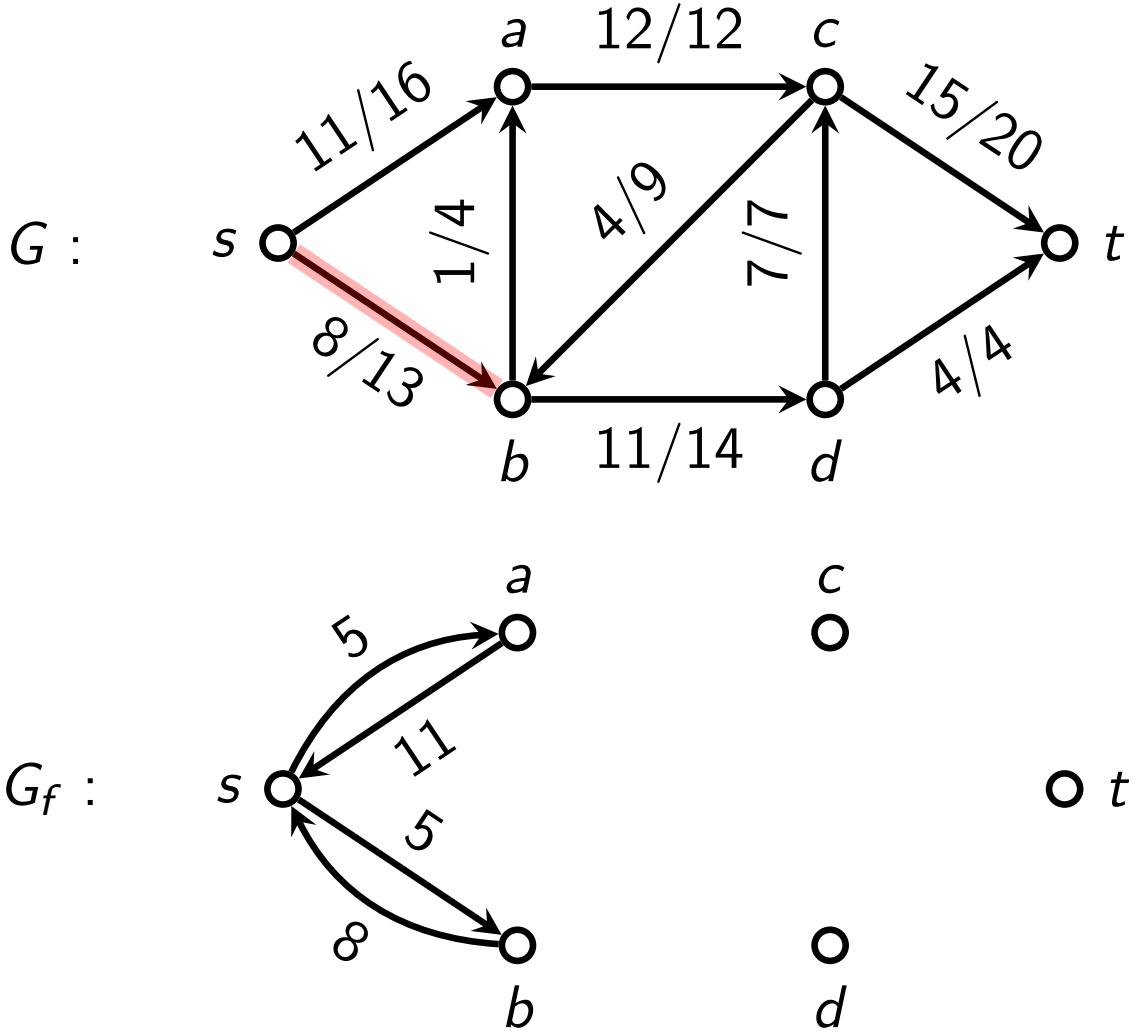
G_f :



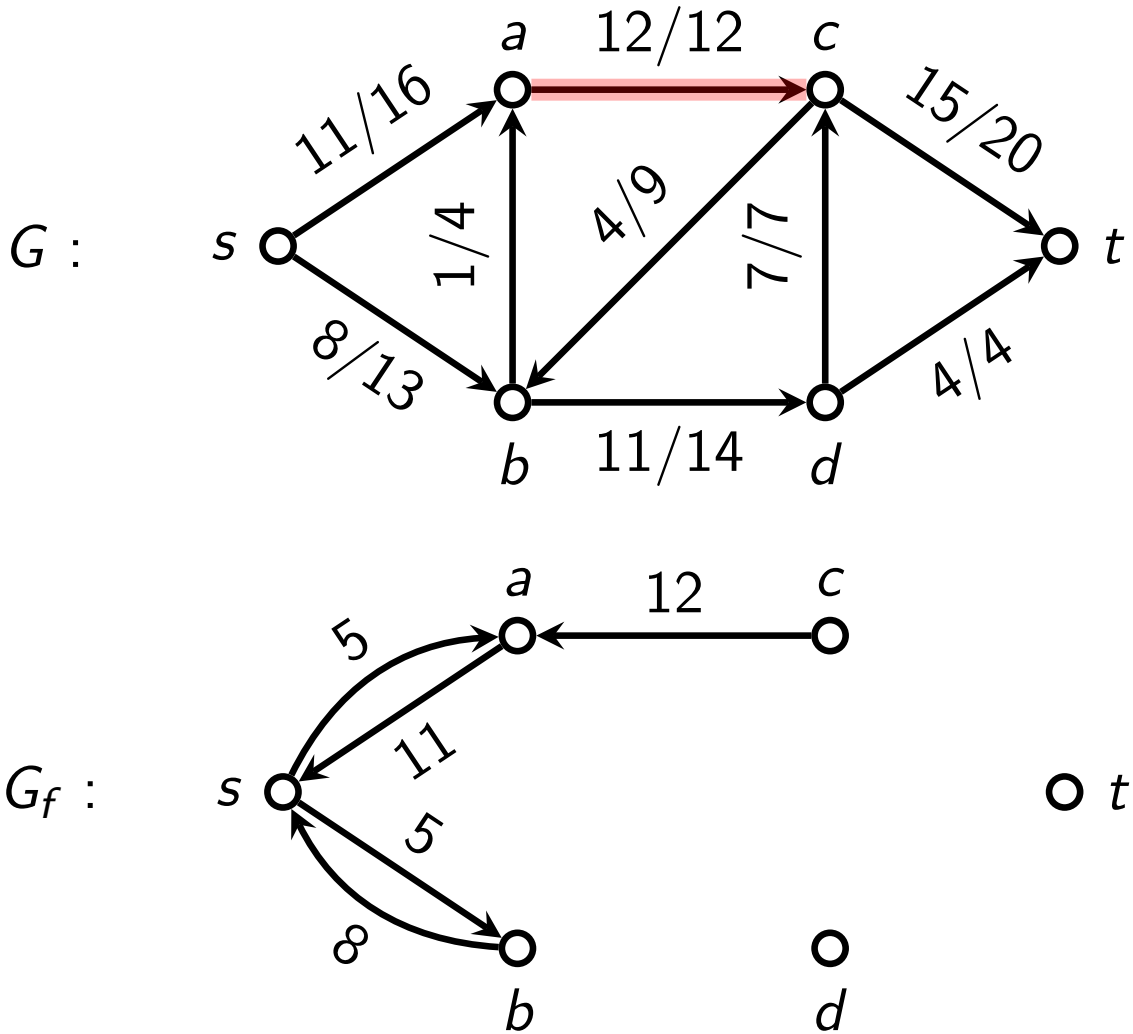
Réseaux résiduels : exemple



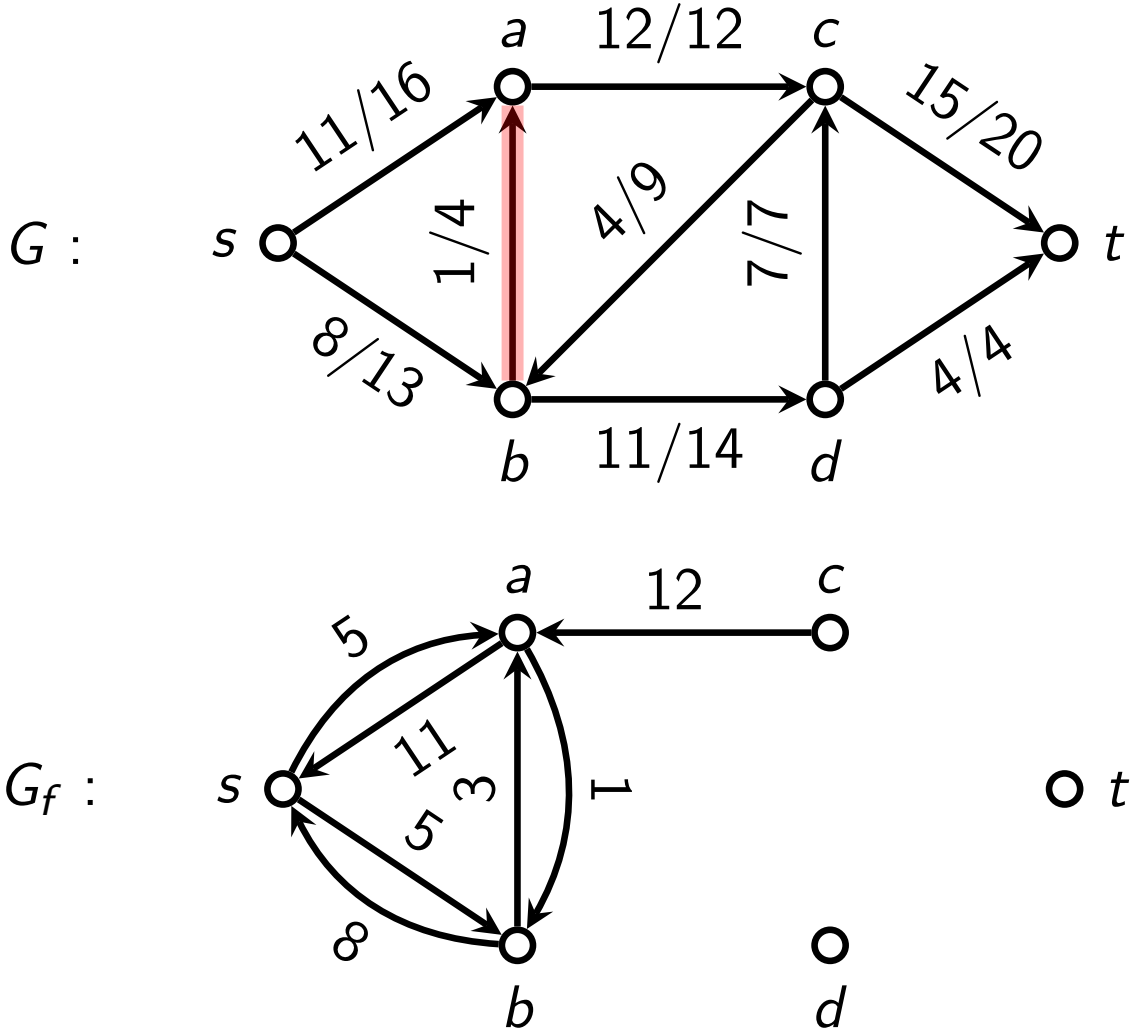
Réseaux résiduels : exemple



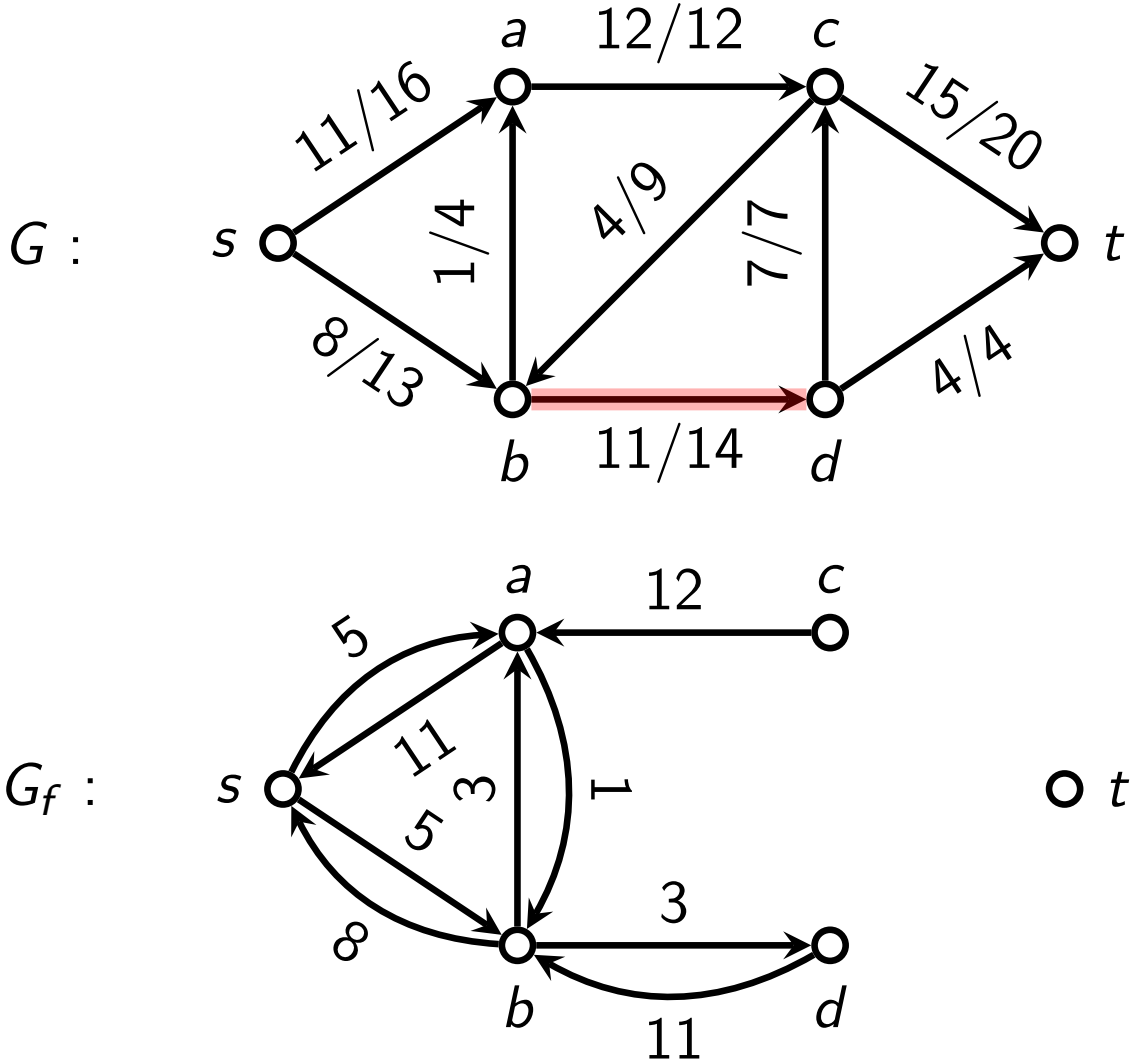
Réseaux résiduels : exemple



Réseaux résiduels : exemple

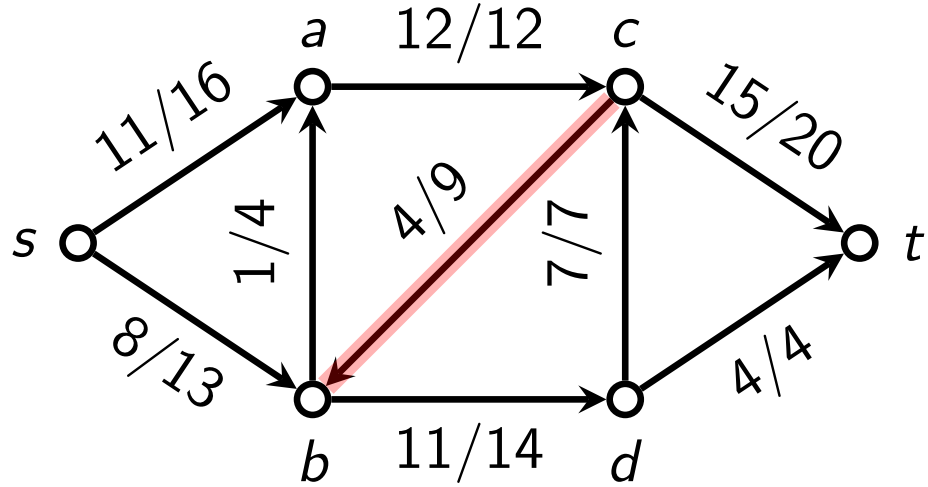


Réseaux résiduels : exemple

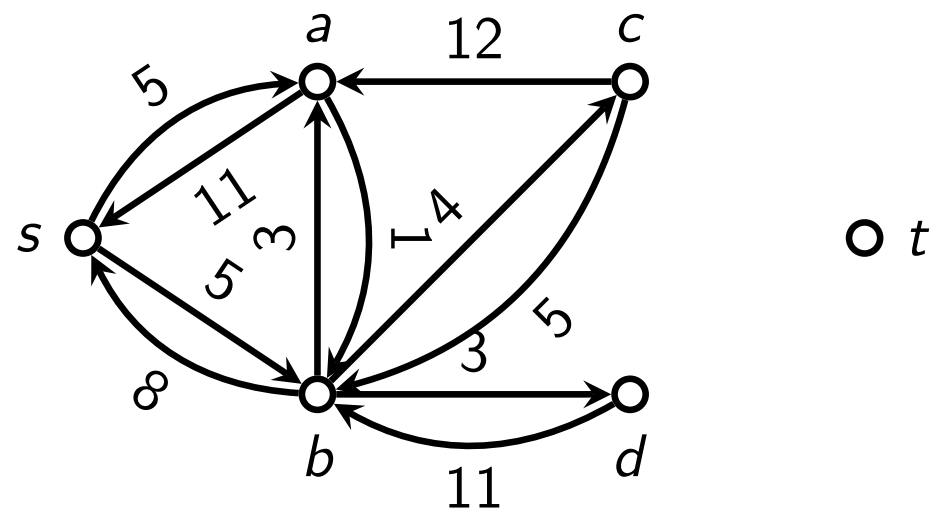


Réseaux résiduels : exemple

G :

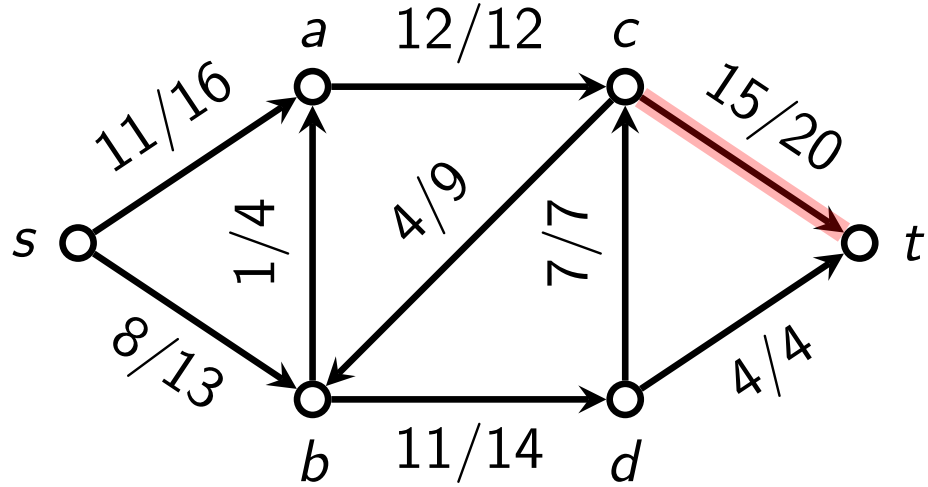


G_f :

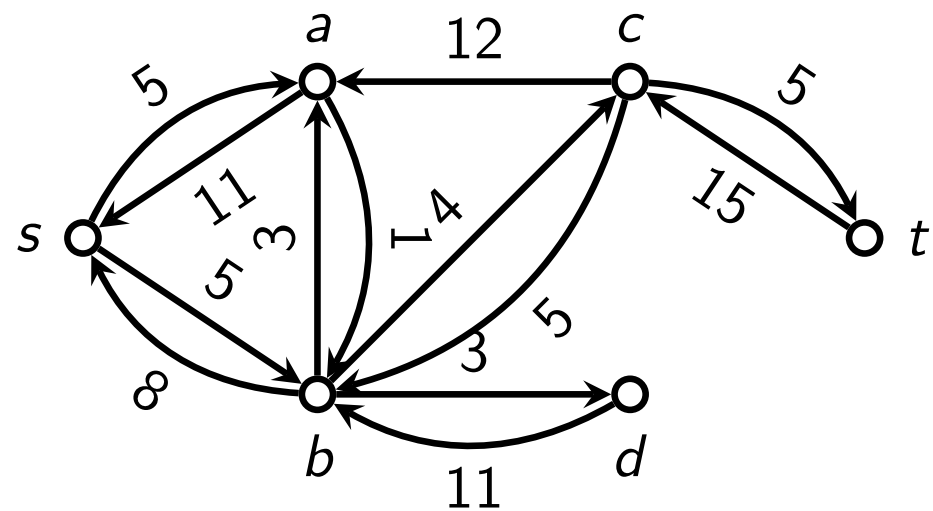


Réseaux résiduels : exemple

G :

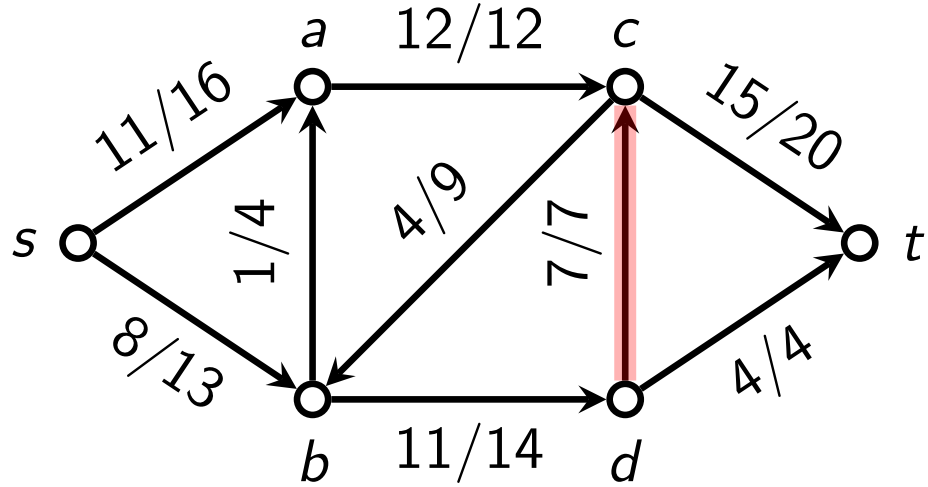


G_f :

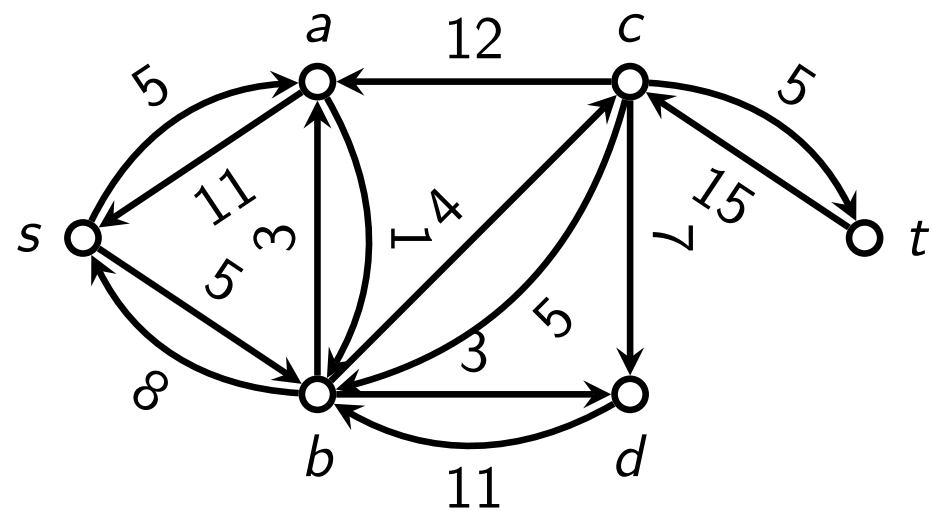


Réseaux résiduels : exemple

G :

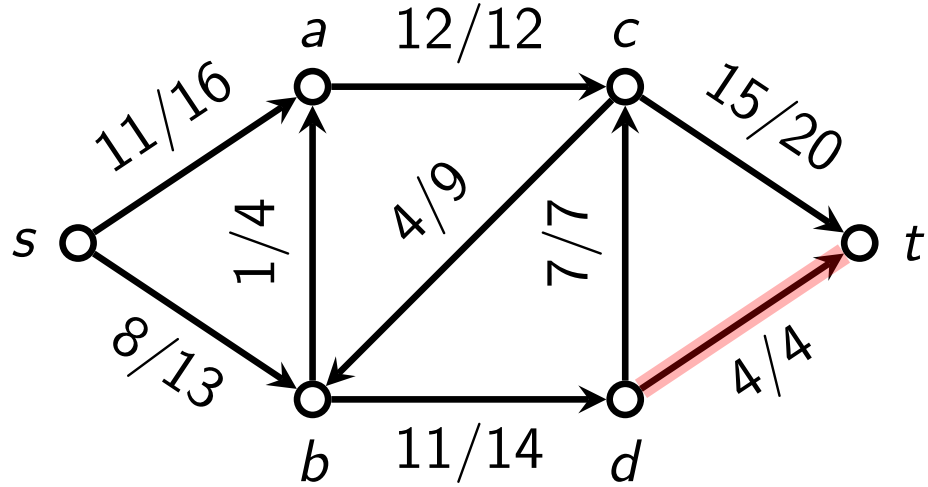


G_f :

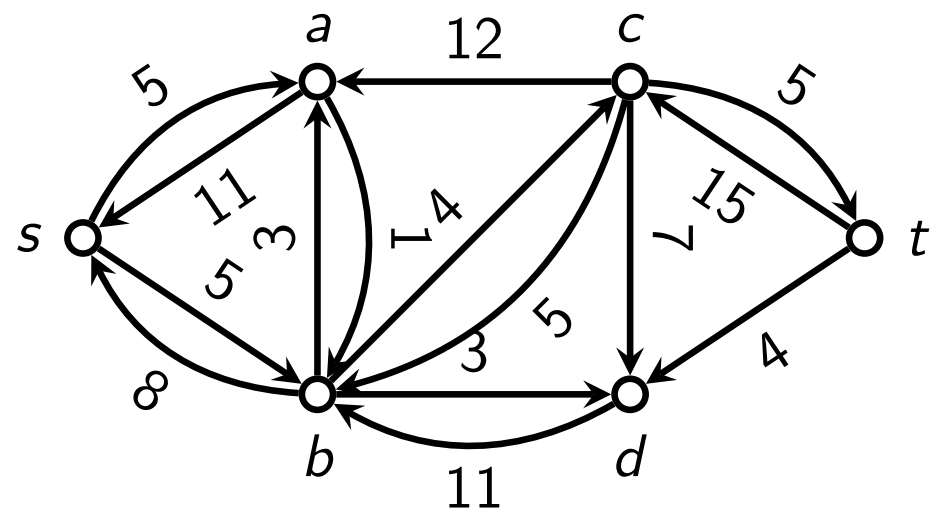


Réseaux résiduels : exemple

G :



G_f :



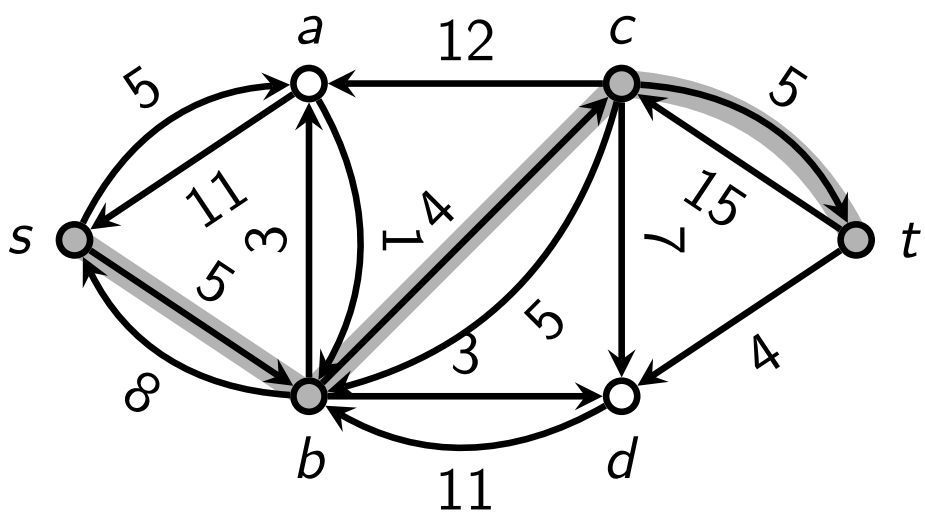
Chemins augmentants

Un **chemin augmentant** P est un chemin simple (sans répétition de sommets) de s à t dans le réseau résiduel G_f. Sa **capacité minimale** est $c_f(P) = \min_{(u,v)\in A(P)} c_f(u, v)$.

Chemins augmentants

Un **chemin augmentant** P est un chemin simple (sans répétition de sommets) de s à t dans le réseau résiduel G_f. Sa **capacité minimale** est $c_f(P) = \min_{(u,v)\in A(P)} c_f(u,v)$.

Exemple 3



Augmentation du flot par les chemins augmentants

- Une fois le chemin augmentant P trouvé, on augmente le flot *au maximum* sur tous ses arcs ;

Augmentation du flot par les chemins augmentants

- Une fois le chemin augmentant P trouvé, on augmente le flot *au maximum* sur tous ses arcs ;
- On ne peut pas faire mieux que $c_f(P)$, donc on augmente f de $c_f(P)$ sur tous les arcs de P ;

Augmentation du flot par les chemins augmentants

- Une fois le chemin augmentant P trouvé, on augmente le flot *au maximum* sur tous ses arcs ;
- On ne peut pas faire mieux que $c_f(P)$, donc on augmente f de $c_f(P)$ sur tous les arcs de P ;
- **Attention :** les arcs de P appartiennent au *résiduel* G_f !

Augmentation du flot par les chemins augmentants

- Une fois le chemin augmentant P trouvé, on augmente le flot *au maximum* sur tous ses arcs ;
- On ne peut pas faire mieux que $c_f(P)$, donc on augmente f de $c_f(P)$ sur tous les arcs de P ;
- **Attention :** les arcs de P appartiennent au *résiduel* G_f !
- Cela signifie que bien que $|f|$ augmente, f peut **diminuer** sur certains arcs du réseau G ;

L'algorithme d'Edmonds-Karp

- On parle de *méthode* (et non d'*algorithme*) de Ford-Fulkerson car aucune précision n'est donnée sur la manière de choisir le chemin augmentant ;

L'algorithme d'Edmonds-Karp

- On parle de *méthode* (et non d'*algorithme*) de Ford-Fulkerson car aucune précision n'est donnée sur la manière de choisir le chemin augmentant ;
- Il n'y a pas de consensus sur la manière optimale de choisir ce chemin (cf. TD) ;

L'algorithme d'Edmonds-Karp

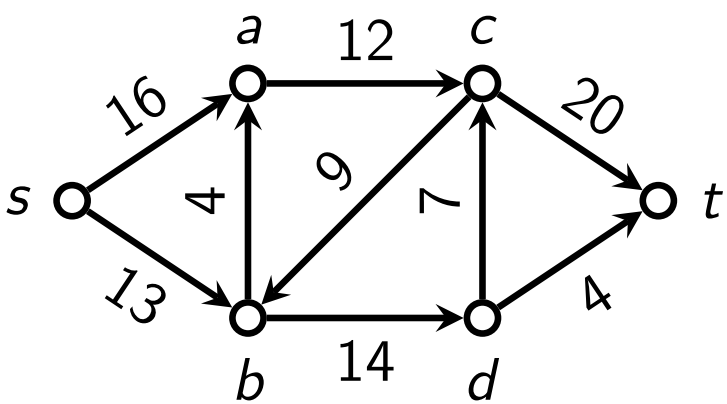
- On parle de *méthode* (et non d'*algorithme*) de Ford-Fulkerson car aucune précision n'est donnée sur la manière de choisir le chemin augmentant ;
- Il n'y a pas de consensus sur la manière optimale de choisir ce chemin (cf. TD) ;
- Algorithme d'Edmonds-Karp = méthode de Ford-Fulkerson avec sélection du plus court chemin comme chemin augmentant ;

L'algorithme d'Edmonds-Karp

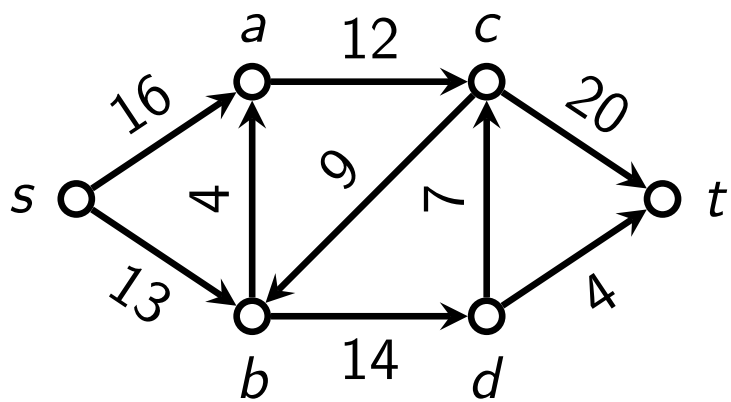
- On parle de *méthode* (et non d'*algorithme*) de Ford-Fulkerson car aucune précision n'est donnée sur la manière de choisir le chemin augmentant ;
- Il n'y a pas de consensus sur la manière optimale de choisir ce chemin (cf. TD) ;
- Algorithme d'Edmonds-Karp = méthode de Ford-Fulkerson avec sélection du plus court chemin comme chemin augmentant ;
- Pas nécessairement toujours le plus rapide, mais on peut démontrer que cette approche effectue au plus $O(|V||A|)$ itérations ;

L'algorithme d'Edmonds-Karp en action

Examinons les étapes de l'algorithme d'Edmonds-Karp sur un exemple.



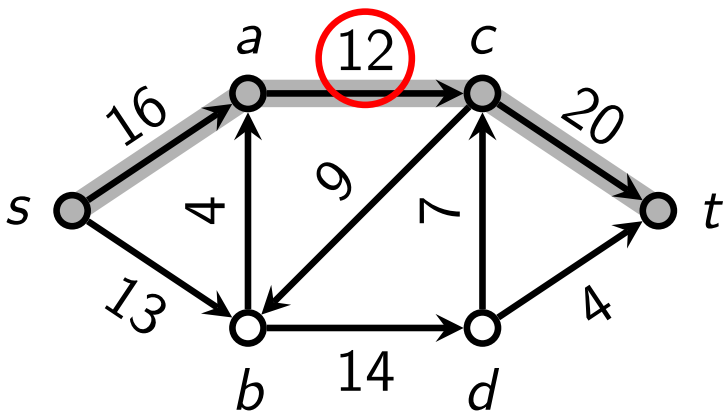
G_f



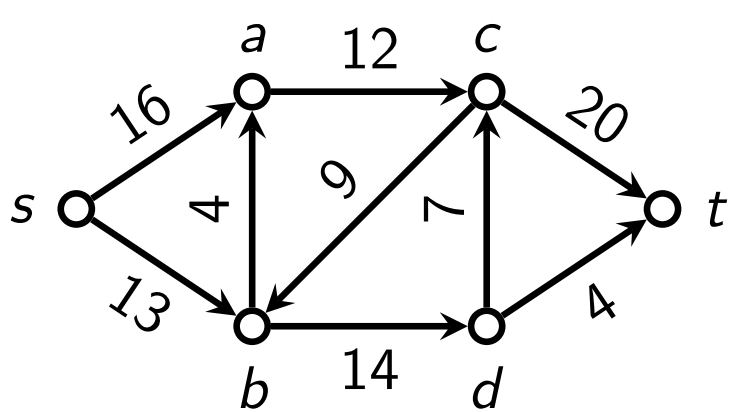
G

L'algorithme d'Edmonds-Karp en action

Examinons les étapes de l'algorithme d'Edmonds-Karp sur un exemple.



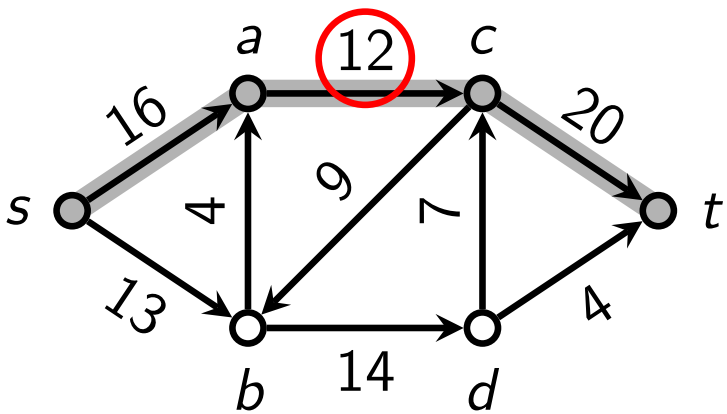
G_f



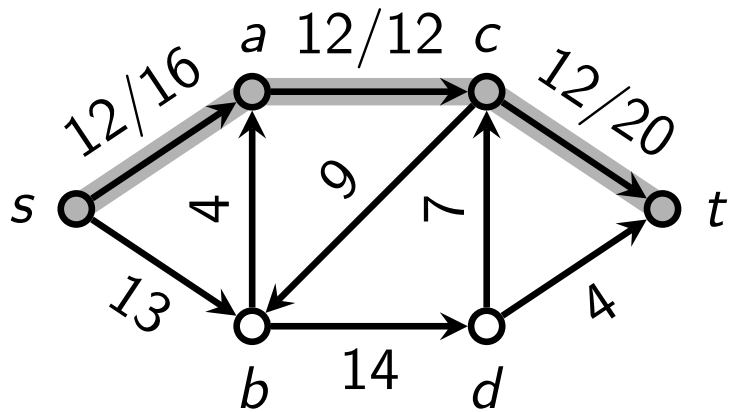
G

L'algorithme d'Edmonds-Karp en action

Examinons les étapes de l'algorithme d'Edmonds-Karp sur un exemple.



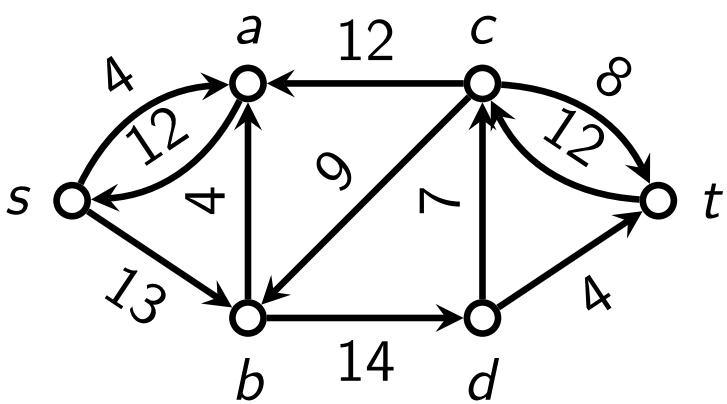
G_f



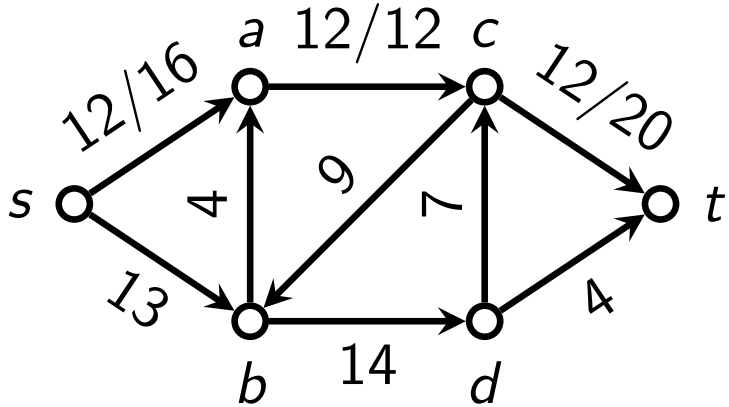
G

L'algorithme d'Edmonds-Karp en action

Examinons les étapes de l'algorithme d'Edmonds-Karp sur un exemple.



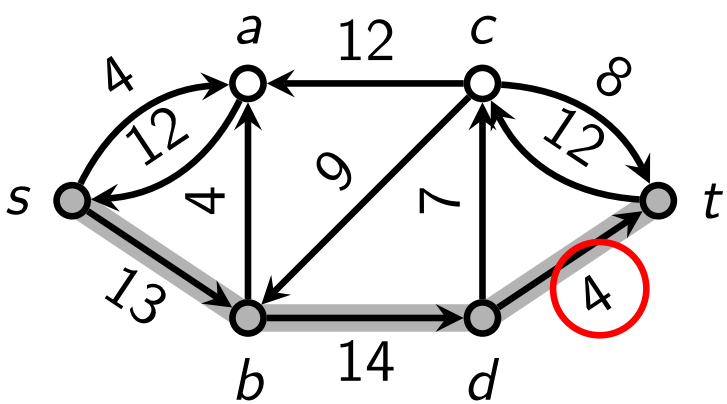
G_f



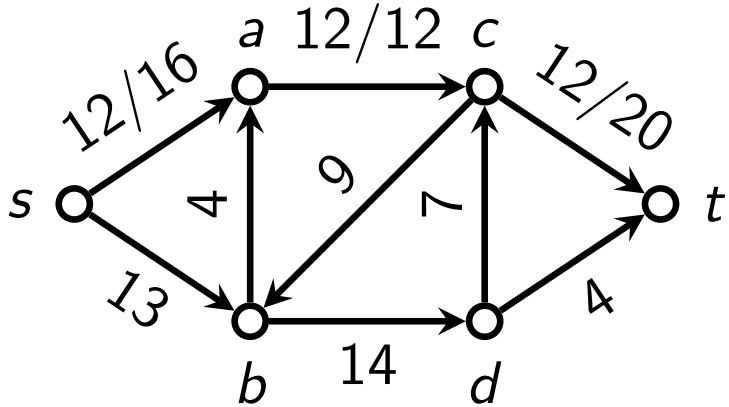
G

L'algorithme d'Edmonds-Karp en action

Examinons les étapes de l'algorithme d'Edmonds-Karp sur un exemple.



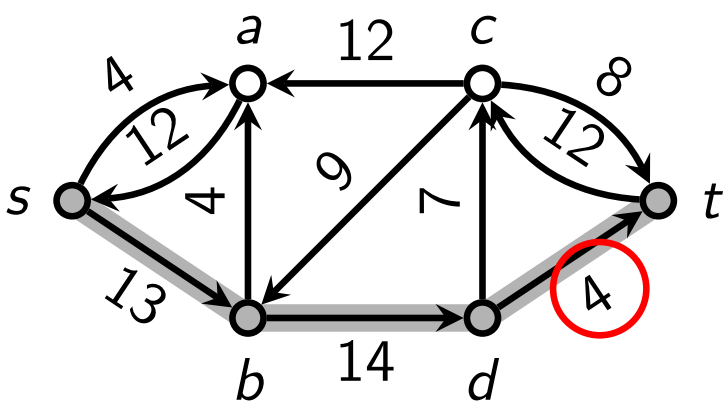
G_f



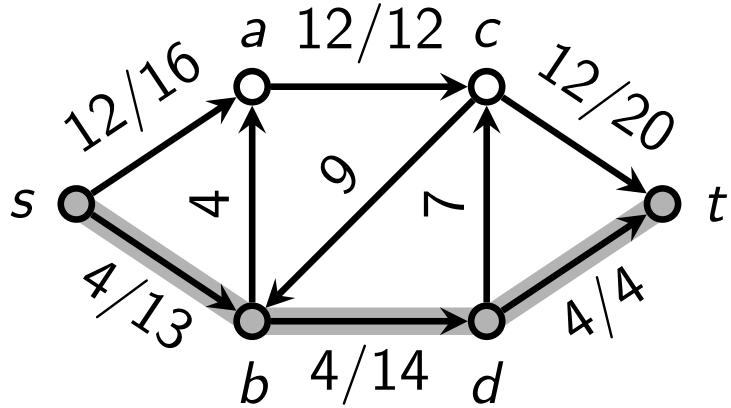
G

L'algorithme d'Edmonds-Karp en action

Examinons les étapes de l'algorithme d'Edmonds-Karp sur un exemple.



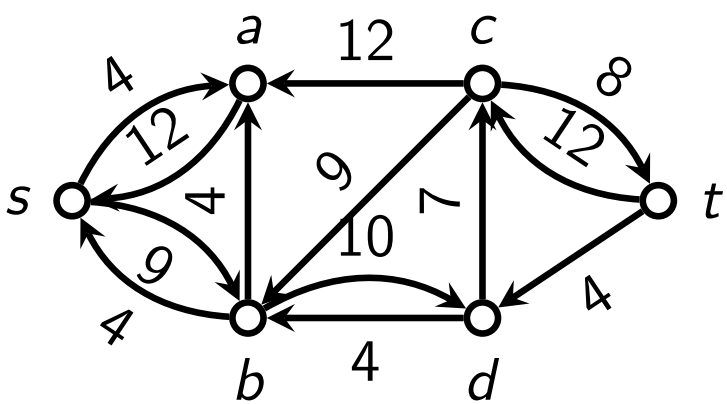
G_f



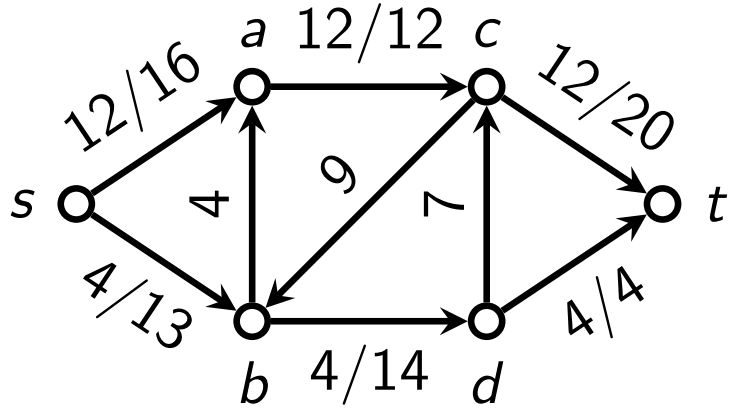
G

L'algorithme d'Edmonds-Karp en action

Examinons les étapes de l'algorithme d'Edmonds-Karp sur un exemple.



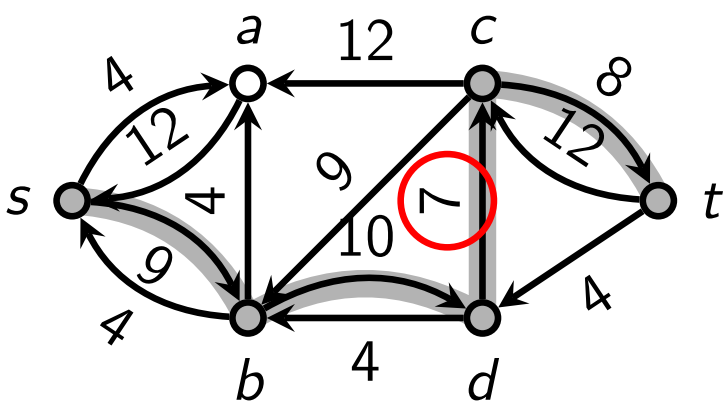
G_f



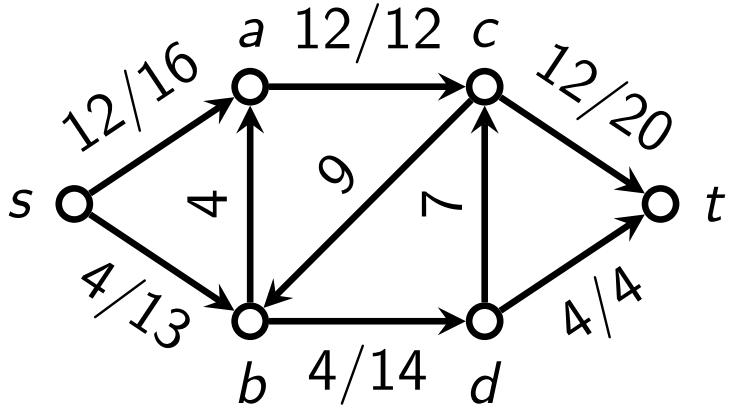
G

L'algorithme d'Edmonds-Karp en action

Examinons les étapes de l'algorithme d'Edmonds-Karp sur un exemple.



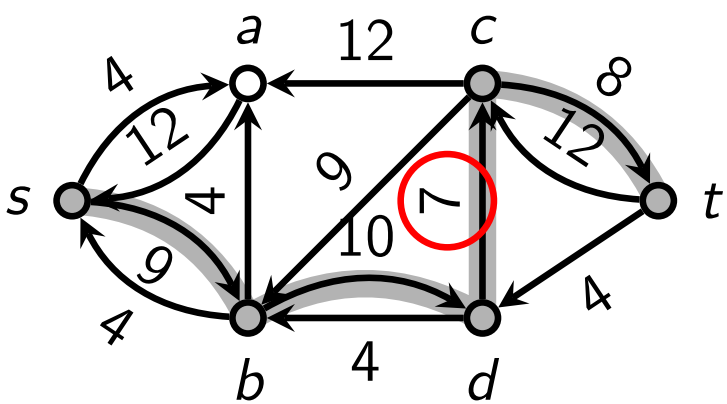
G_f



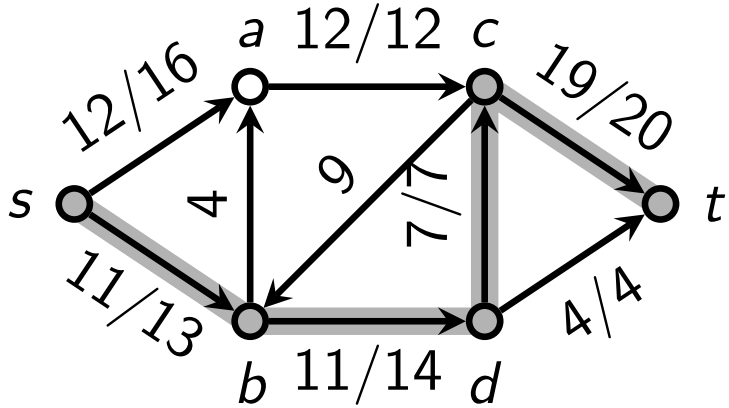
G

L'algorithme d'Edmonds-Karp en action

Examinons les étapes de l'algorithme d'Edmonds-Karp sur un exemple.



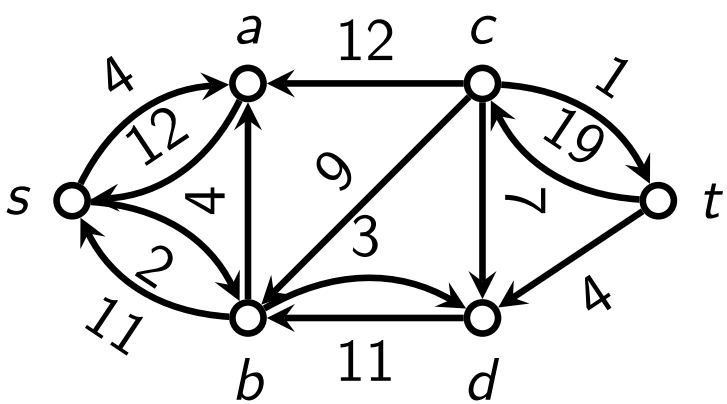
G_f



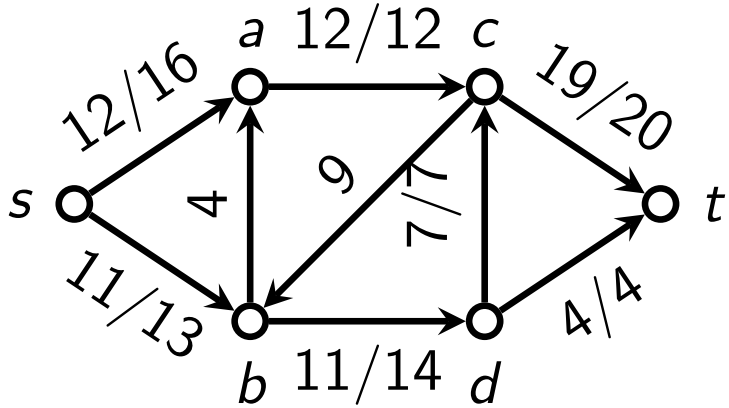
G

L'algorithme d'Edmonds-Karp en action

Examinons les étapes de l'algorithme d'Edmonds-Karp sur un exemple.



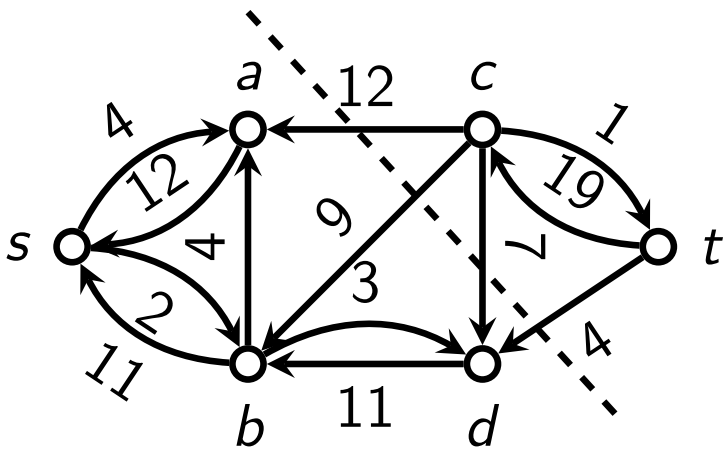
G_f



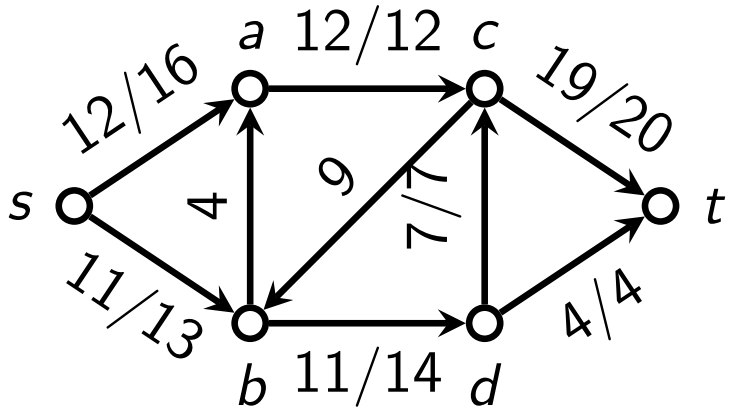
G

L'algorithme d'Edmonds-Karp en action

Examinons les étapes de l'algorithme d'Edmonds-Karp sur un exemple.



G_f



G

Coupes

La correction de la méthode de Ford-Fulkerson (quelle que soit la manière dont on choisit le chemin augmentant) découle d'un lien avec la notion de *coupe*.

Coupes

La correction de la méthode de Ford-Fulkerson (quelle que soit la manière dont on choisit le chemin augmentant) découle d'un lien avec la notion de *coupe*.

Définition 3

Une **coupe** (S, T) dans un réseau de flot avec source s et puits t est une partition de ses sommets en deux ensembles S et T tels que $s \in S$ et $t \in T$.

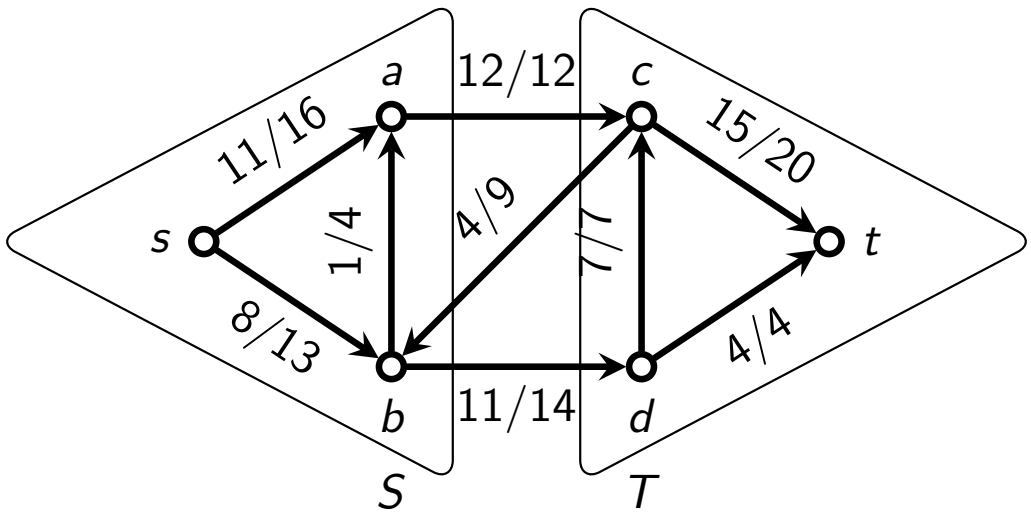
Coupes

La correction de la méthode de Ford-Fulkerson (quelle que soit la manière dont on choisit le chemin augmentant) découle d'un lien avec la notion de *coupe*.

Définition 3

Une **coupe** (S, T) dans un réseau de flot avec source s et puits t est une partition de ses sommets en deux ensembles S et T tels que $s \in S$ et $t \in T$.

Exemple 4



Capacité et flot associés à une coupe

Soit (S, T) une coupe ;

1. son **flot associé** est le flot "net" dans les arcs entre S et T :

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

Capacité et flot associés à une coupe

Soit (S, T) une coupe ;

1. son **flot associé** est le flot "net" dans les arcs entre S et T :

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

2. sa **capacité** est la somme des capacités des arcs de S à T :

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

Capacité et flot associés à une coupe

Soit (S, T) une coupe ;

1. son **flot associé** est le flot "net" dans les arcs entre S et T :

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

2. sa **capacité** est la somme des capacités des arcs de S à T :

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

3. (S, T) est **minimum** si pour toute coupe (S', T') du réseau, on a $c(S,T) \leq c(S',T')$.

Capacité et flot associés à une coupe

Soit (S, T) une coupe ;

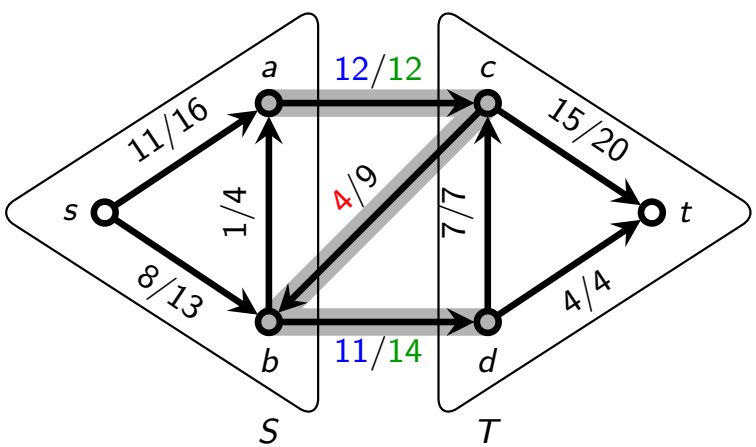
1. son **flot associé** est le flot "net" dans les arcs entre S et T :

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

2. sa **capacité** est la somme des capacités des arcs de S à T :

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

3. (S, T) est **minimum** si pour toute coupe (S', T') du réseau, on a $c(S,T) \leq c(S',T')$.



Capacité et flot associés à une coupe

Soit (S, T) une coupe ;

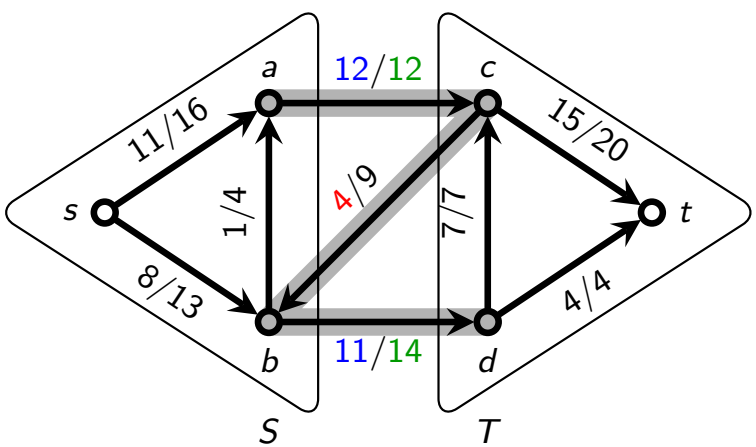
1. son **flot associé** est le flot "net" dans les arcs entre S et T :

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

2. sa **capacité** est la somme des capacités des arcs de S à T :

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

3. (S, T) est **minimum** si pour toute coupe (S', T') du réseau, on a $c(S,T) \leq c(S',T')$.



$$f(S,T) = 12 + 11 - 4 = 19$$

Capacité et flot associés à une coupe

Soit (S, T) une coupe ;

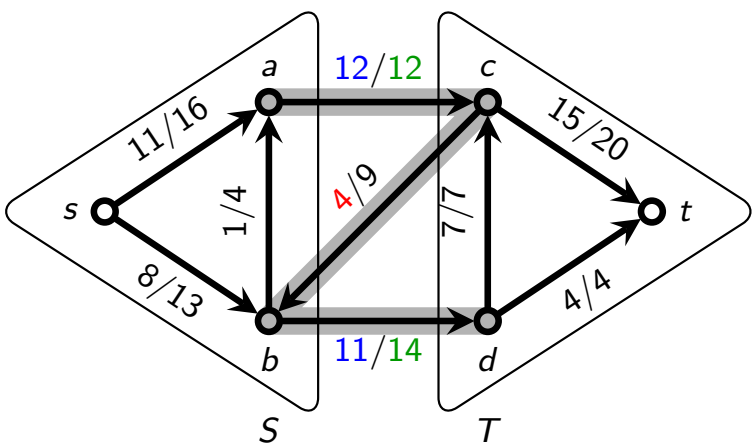
1. son **flot associé** est le flot "net" dans les arcs entre S et T :

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

2. sa **capacité** est la somme des capacités des arcs de S à T :

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

3. (S, T) est **minimum** si pour toute coupe (S', T') du réseau, on a $c(S,T) \leq c(S',T')$.



$$f(S,T) = 12 + 11 - 4 = 19$$
$$c(S,T) = 12 + 14 = 26$$

Lien entre les flots et les coupes

- La capacité $c(u, v)$ limite le flot pouvant circuler de u à v ;

Lien entre les flots et les coupes

- La capacité $c(u, v)$ limite le flot pouvant circuler de u à v ;
- De la même façon, la capacité de la coupe (S, T) limite le flot pouvant circuler de S à T ;

Lien entre les flots et les coupes

- La capacité $c(u, v)$ limite le flot pouvant circuler de u à v ;
- De la même façon, la capacité de la coupe (S, T) limite le flot pouvant circuler de S à T ;
- Plus formellement :

Lien entre les flots et les coupes

- La capacité $c(u, v)$ limite le flot pouvant circuler de u à v ;
- De la même façon, la capacité de la coupe (S, T) limite le flot pouvant circuler de S à T ;
- Plus formellement :

Lemme 4

Soit f un flot sur un réseau de flot G, et (S, T) une coupe sur G. Alors $|f| = f(S, T)$.

Lien entre les flots et les coupes

- La capacité $c(u, v)$ limite le flot pouvant circuler de u à v ;
- De la même façon, la capacité de la coupe (S, T) limite le flot pouvant circuler de S à T ;
- Plus formellement :

Lemme 4

Soit f un flot sur un réseau de flot G, et (S, T) une coupe sur G. Alors $|f| = f(S, T)$.

Corollaire 5

Pour tout flot f sur un réseau de flot G et pour toute coupe (S, T) sur ce même réseau, on a $|f| \leq c(S, T)$.

Théorème *max-flow min-cut*

Les résultats précédents interviennent dans la preuve du théorème suivant, qui prouve entre autres la correction de la méthode de Ford-Fulkerson.

Théorème 6 (Max-flow min-cut)

[1] Soit f un flot dans un réseau $G = (V, A, c)$ de source s et de puits t. Les conditions suivantes sont équivalentes :

Théorème *max-flow min-cut*

Les résultats précédents interviennent dans la preuve du théorème suivant, qui prouve entre autres la correction de la méthode de Ford-Fulkerson.

Théorème 6 (Max-flow min-cut)

[1] Soit f un flot dans un réseau $G = (V, A, c)$ de source s et de puits t. Les conditions suivantes sont équivalentes :

1. *f est un flot maximum ;*

Théorème *max-flow min-cut*

Les résultats précédents interviennent dans la preuve du théorème suivant, qui prouve entre autres la correction de la méthode de Ford-Fulkerson.

Théorème 6 (Max-flow min-cut)

[1] Soit f un flot dans un réseau $G = (V, A, c)$ de source s et de puits t. Les conditions suivantes sont équivalentes :

1. *f est un flot maximum ;*
2. *le réseau résiduel G_f ne contient pas de chemin augmentant ;*

Théorème *max-flow min-cut*

Les résultats précédents interviennent dans la preuve du théorème suivant, qui prouve entre autres la correction de la méthode de Ford-Fulkerson.

Théorème 6 (Max-flow min-cut)

[1] Soit f un flot dans un réseau $G = (V, A, c)$ de source s et de puits t. Les conditions suivantes sont équivalentes :

1. *f est un flot maximum ;*
2. *le réseau résiduel G_f ne contient pas de chemin augmentant ;*
3. *il existe une coupe (S, T) pour G telle que $|f| = c(S, T)$.*

Complexité de l'algorithme d'Edmonds-Karp

- Maintenant qu'on sait que l'algorithme d'Edmonds-Karp est correct, intéressons-nous à sa complexité ;
- Pour ce faire, examinons en détails les algorithmes utilisés et leurs complexités ;

Calcul du chemin augmentant

Le chemin augmentant s'obtient par un simple parcours en largeur de la source au puits :

Algorithme 2 : CheminAugmentant(G, source, puits)

Entrées : un graphe orienté pondéré G, et deux sommets source et puits.

Sortie : un chemin de source à puits dans G, ou NIL s'il n'en existe pas.

```
1  déjà_visités ← tableau(G.nombre_sommets(), FAUX);
2  a_traiter ← file();
3  a_traiter.enfiler(source);
4  parents ← tableau(G.nombre_sommets(), NIL);
5  tant que a_traiter.pas_vide() faire
6      sommet ← a_traiter.défiler();
7      si sommet = puits alors arrêter;
8      si ¬ déjà_visités[sommet] alors
9          déjà_visités[sommet] ← VRAI;
10         pour chaque v ∈ G.successeurs(sommet) faire
11             a_traiter.enfiler(v);
12             si parents[v] = NIL alors  parents[v] ← sommet ;
13 renvoyer ReconstruireChemin(G, source, puits, parents);
```

Reconstruction du chemin augmentant

Le chemin se reconstruit explicitement sur base des informations sur les parents du parcours en largeur :

Algorithme 3 : ReconstruireChemin(G, début, fin, parents)

Entrées : un graphe orienté pondéré G, deux sommets début et fin, et les parents des sommets du graphe.
Sortie : un chemin de début à fin dans G, ou NIL s'il n'en existe pas.

```
1 chemin ← GrapheOrientéPondéré();
2 v ← fin;
3 tant que v ≠ début faire
4 |   si parents[v] = NIL alors renvoyer NIL;
5 |   chemin.ajouter_arc(parents[v], v, G.poids_arc(parents[v], v));
6 |   v ← parents[v];
7 renvoyer chemin;
```

Augmentation du flot

L'augmentation du flot se fait le long du chemin augmentant, d'une quantité égale à la capacité résiduelle minimum :

Algorithme 4 : AugmenterFlot(f, P)

Entrées : un flot f, un chemin P.
Résultat : f augmente au maximum le long du chemin P.

```
1 capacité_min ← min {c | (u, v, c) ∈ P.arcs()};
2 pour chaque (u, v, c) ∈ P.arcs() faire
3 |   si (u, v) ∈ f alors  f[(u, v)] ← f[(u, v)] + capacité_min ;
4 |   sinon  f[(v, u)] ← f[(v, u)] − capacité_min ;
```

Mise à jour du résiduel

On met enfin à jour les arcs du chemin augmentant trouvé dans le réseau résiduel :

Algorithme 5 : MettreAJourRésiduel(G, G_f, arcs, f)

Entrées : un réseau G, le réseau résiduel correspondant G_f, un ensemble d'arcs dans G_f, et un flot f.

Résultat : modifie les arcs de G_f suivant l'augmentation de flot qu'ils ont subi.

```
   // calculer les capacités résiduelles
1  c_f ← tableau associatif;
2  pour chaque (u, v) ∈ arcs faire
3  |   si G.contient_arc(v, u) alors échanger u et v;
4  |   c_f[(u, v)] ← G.poids_arc(u, v) − f[(u, v)] ;
5  |   c_f[(v, u)] ← f[(u, v)] ;
   // remplacer les arcs concernés de G_f en supprimant ceux
      dont le poids devient nul
6  pour chaque (u, v) ∈ arcs faire
7  |   si c_f[(u, v)] > 0 alors  G_f.ajouter_arc(u, v, c_f[(u, v)]) ;
8  |   sinon  G_f.supprimer_arc(u, v) ;
```

Complexité de l'algorithme

Reprenons le pseudocode de la méthode de Ford-Fulkerson :

Algorithme 6 : EDMONDSKARP(G)

Entrées : un réseau de flot G.
Sortie : un flot maximum pour G.

```
1  flot ← tableau associatif (clés = G.arcs(), valeurs = 0);        // O(|A|)
2  s ← unique sommet de degré entrant nul de G;
3  t ← unique sommet de degré sortant nul de G;
4  G_f ← G
5  chemin ← CHEMINAUGMENTANT(G_f, s, t)
6  tant que chemin ≠ NIL faire
7  |   AUGMENTERFLOT(flot, chemin)
8  |   METTREAJOURRÉSIDUEL(G, G_f, chemin.arcs(), flot)
9  |   chemin ← CHEMINAUGMENTANT(G_f, s, t)
10 renvoyer flot;
```

Complexité de l'algorithme

Reprenons le pseudocode de la méthode de Ford-Fulkerson :

Algorithme 6 : EDMONDSKARP(G)

Entrées : un réseau de flot G.
Sortie : un flot maximum pour G.

```
1  flot ← tableau associatif (clés = G.arcs(), valeurs = 0);        // O(|A|)
2  s ← unique sommet de degré entrant nul de G;
3  t ← unique sommet de degré sortant nul de G;
4  G_f ← G;                                                         // O(|V| + |A|)
5  chemin ← CHEMINAUGMENTANT(G_f, s, t)
6  tant que chemin ≠ NIL faire
7  |   AUGMENTERFLOT(flot, chemin)
8  |   METTREAJOURRÉSIDUEL(G, G_f, chemin.arcs(), flot)
9  |   chemin ← CHEMINAUGMENTANT(G_f, s, t)
10 renvoyer flot;
```

Complexité de l'algorithme

Reprenons le pseudocode de la méthode de Ford-Fulkerson :

Algorithme 6 : EDMONDSKARP(G)

Entrées : un réseau de flot G.
Sortie : un flot maximum pour G.

```
1  flot ← tableau associatif (clés = G.arcs(), valeurs = 0);        // O(|A|)
2  s ← unique sommet de degré entrant nul de G;
3  t ← unique sommet de degré sortant nul de G;
4  G_f ← G;                                                         // O(|V| + |A|)
5  chemin ← CHEMINAUGMENTANT(G_f, s, t);                            // O(|V| + |A|)
6  tant que chemin ≠ NIL faire
7  |   AUGMENTERFLOT(flot, chemin)
8  |   METTREAJOURRÉSIDUEL(G, G_f, chemin.arcs(), flot)
9  |   chemin ← CHEMINAUGMENTANT(G_f, s, t)
10 renvoyer flot;
```

Complexité de l'algorithme

Reprenons le pseudocode de la méthode de Ford-Fulkerson :

Algorithme 6 : EDMONDSKARP(G)

Entrées : un réseau de flot G.
Sortie : un flot maximum pour G.

```
1  flot ← tableau associatif (clés = G.arcs(), valeurs = 0);        // O(|A|)
2  s ← unique sommet de degré entrant nul de G;
3  t ← unique sommet de degré sortant nul de G;
4  G_f ← G;                                                         // O(|V| + |A|)
5  chemin ← CHEMINAUGMENTANT(G_f, s, t);                            // O(|V| + |A|)
6  tant que chemin ≠ NIL faire
7  |   AUGMENTERFLOT(flot, chemin);                                 // O(|V|)
8  |   METTREAJOURRÉSIDUEL(G, G_f, chemin.arcs(), flot)
9  |   chemin ← CHEMINAUGMENTANT(G_f, s, t)
10 renvoyer flot;
```

Complexité de l'algorithme

Reprenons le pseudocode de la méthode de Ford-Fulkerson :

Algorithme 6 : EDMONDSKARP(G)

Entrées : un réseau de flot G.
Sortie : un flot maximum pour G.

```
1  flot ← tableau associatif (clés = G.arcs(), valeurs = 0);        // O(|A|)
2  s ← unique sommet de degré entrant nul de G;
3  t ← unique sommet de degré sortant nul de G;
4  G_f ← G;                                                         // O(|V| + |A|)
5  chemin ← CHEMINAUGMENTANT(G_f, s, t);                            // O(|V| + |A|)
6  tant que chemin ≠ NIL faire
7  |   AUGMENTERFLOT(flot, chemin);                                 // O(|V|)
8  |   METTREAJOURRÉSIDUEL(G, G_f, chemin.arcs(), flot);            // O(|V|)
9  |   chemin ← CHEMINAUGMENTANT(G_f, s, t)
10 renvoyer flot;
```

Complexité de l'algorithme

Reprenons le pseudocode de la méthode de Ford-Fulkerson :

Algorithme 6 : EDMONDSKARP(G)

Entrées : un réseau de flot G.
Sortie : un flot maximum pour G.

```
1  flot ← tableau associatif (clés = G.arcs(), valeurs = 0);        // O(|A|)
2  s ← unique sommet de degré entrant nul de G;
3  t ← unique sommet de degré sortant nul de G;
4  G_f ← G;                                                         // O(|V| + |A|)
5  chemin ← CHEMINAUGMENTANT(G_f, s, t);                            // O(|V| + |A|)
6  tant que chemin ≠ NIL faire
7  |   AUGMENTERFLOT(flot, chemin);                                 // O(|V|)
8  |   METTREAJOURRÉSIDUEL(G, G_f, chemin.arcs(), flot);            // O(|V|)
9  |   chemin ← CHEMINAUGMENTANT(G_f, s, t);                       // O(|V| + |A|)
10 renvoyer flot;
```

Complexité de l'algorithme

Reprenons le pseudocode de la méthode de Ford-Fulkerson :

Algorithme 6 : EDMONDSKARP(G)

Entrées : un réseau de flot G.
Sortie : un flot maximum pour G.

```
1  flot ← tableau associatif (clés = G.arcs(), valeurs = 0);        // O(|A|)
2  s ← unique sommet de degré entrant nul de G;
3  t ← unique sommet de degré sortant nul de G;
4  G_f ← G;                                                         // O(|V| + |A|)
5  chemin ← CHEMINAUGMENTANT(G_f, s, t);                            // O(|V| + |A|)
6  tant que chemin ≠ NIL faire
7  |   AUGMENTERFLOT(flot, chemin);                                 // O(|V|)
8  |   METTREAJOURRÉSIDUEL(G, G_f, chemin.arcs(), flot);            // O(|V|)
9  |   chemin ← CHEMINAUGMENTANT(G_f, s, t);                       // O(|V| + |A|)
10 renvoyer flot;
```

- Les seules opérations coûteuses sont en $O(|V| + |A|) = O(|A|)$ (car G est faiblement connexe) ;

Complexité de l'algorithme

Reprenons le pseudocode de la méthode de Ford-Fulkerson :

Algorithme 6 : EDMONDSKARP(G)

Entrées : un réseau de flot G.
Sortie : un flot maximum pour G.

```
1  flot ← tableau associatif (clés = G.arcs(), valeurs = 0);        // O(|A|)
2  s ← unique sommet de degré entrant nul de G;
3  t ← unique sommet de degré sortant nul de G;
4  G_f ← G;                                                         // O(|V| + |A|)
5  chemin ← CHEMINAUGMENTANT(G_f, s, t);                            // O(|V| + |A|)
6  tant que chemin ≠ NIL faire
7  |   AUGMENTERFLOT(flot, chemin);                                 // O(|V|)
8  |   METTREAJOURRÉSIDUEL(G, G_f, chemin.arcs(), flot);            // O(|V|)
9  |   chemin ← CHEMINAUGMENTANT(G_f, s, t);                       // O(|V| + |A|)
10 renvoyer flot;
```

- Les seules opérations coûteuses sont en $O(|V| + |A|) = O(|A|)$ (car G est faiblement connexe) ;
- On passe $O(|V||A|)$ fois dans la boucle [1] ;

Complexité de l'algorithme

Reprenons le pseudocode de la méthode de Ford-Fulkerson :

Algorithme 6 : EdmondsKarp(G)

Entrées : un réseau de flot G.
Sortie : un flot maximum pour G.

```
1  flot ← tableau associatif (clés = G.arcs(), valeurs = 0);        // O(|A|)
2  s ← unique sommet de degré entrant nul de G;
3  t ← unique sommet de degré sortant nul de G;
4  G_f ← G;                                                         // O(|V| + |A|)
5  chemin ← CheminAugmentant(G_f, s, t);                            // O(|V| + |A|)
6  tant que chemin ≠ NIL faire
7  |   AugmenterFlot(flot, chemin);                                 // O(|V|)
8  |   MettreAJourRésiduel(G, G_f, chemin.arcs(), flot);            // O(|V|)
9  |   chemin ← CheminAugmentant(G_f, s, t);                        // O(|V| + |A|)
10 renvoyer flot;
```

- Les seules opérations coûteuses sont en $O(|V| + |A|) = O(|A|)$ (car G est faiblement connexe) ;
- On passe $O(|V||A|)$ fois dans la boucle [1] ;
- L'algorithme est donc en $O(|V||A|^2)$;

Bibliographie

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.

Introduction to Algorithms.

MIT Press, 3ème edition, 2009.

[2] James B. Orlin.

Max flows in $O(nm)$ time, or better.

In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference (STOC)*, pages 765–774, Palo Alto, CA, June 2013. ACM.