

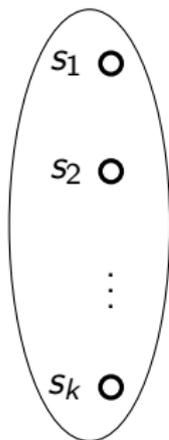
Algorithmique des graphes

7 — Flots et applications (1)

Marie-Pierre Béal (Cours d'Anthony Labarre)

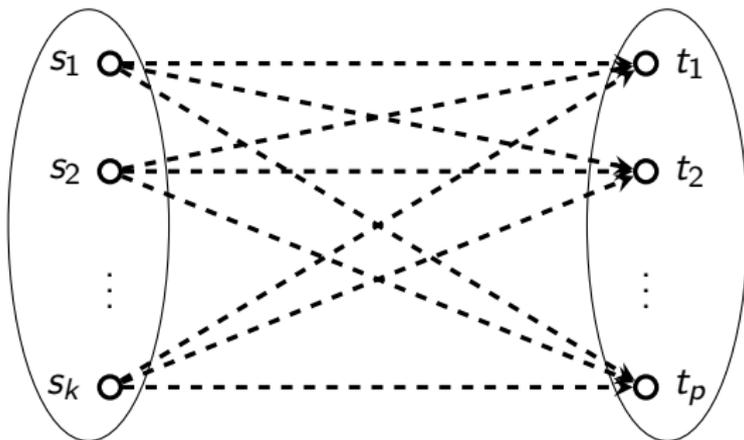
Motivations

- On dispose de *sources* fabriquant divers produits ;



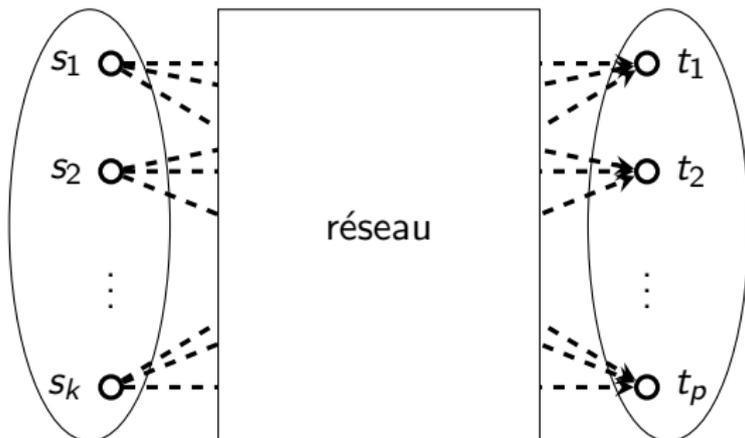
Motivations

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



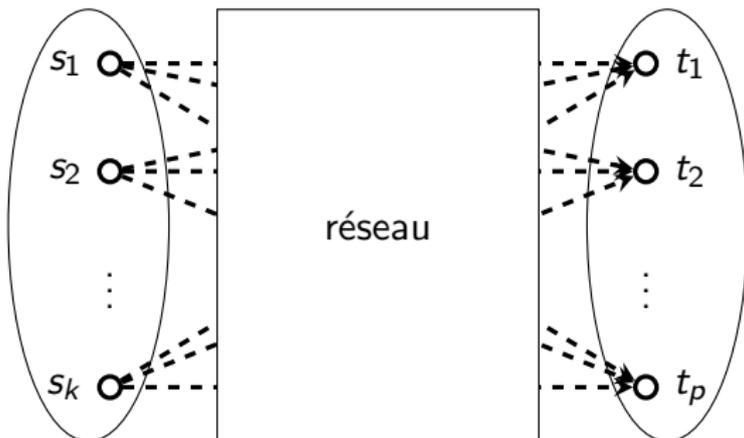
Motivations

- 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 ;

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 **puits** 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 **puits** t ;
- si $(u, v) \notin A$, alors $c(u, v) = 0$.

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 **puits** t ;
- si $(u, v) \notin A$, alors $c(u, v) = 0$.

On aura parfois plusieurs sources et plusieurs puits. On se ramènera à une source et un puits.

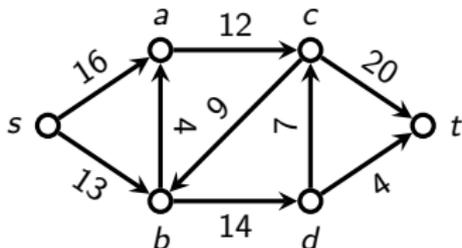
Réseaux de flot

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 **puits** t ;
- si $(u, v) \notin A$, alors $c(u, v) = 0$.

On aura parfois plusieurs sources et plusieurs puits. On se ramènera à une source et un puits.

Exemple 1



Réseaux de flot

Hypothèses supplémentaires :

- ① toutes les *capacités* et tous les *flots* sont ici entiers ;

Réseaux de flot

Hypothèses supplémentaires :

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

Flots

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

- 1 **Accessibilité** : tous les sommets sont sur un chemins de s à t .

Flots

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

- 1 **Accessibilité** : tous les sommets sont sur un chemins de s à t .
- 2 **Contrainte de capacité** : $\forall u, v \in V : f(u, v) \leq c(u, v)$;

Flots

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

- 1 **Accessibilité** : tous les sommets sont sur un chemins de s à t .
- 2 **Contrainte de capacité** : $\forall u, v \in V : f(u, v) \leq c(u, v)$;
- 3 **Anti-symétrie** : $\forall u, v \in V : f(u, v) = -f(v, u)$;

Flots

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

- 1 **Accessibilité** : tous les sommets sont sur un chemins de s à t .
- 2 **Contrainte de capacité** : $\forall u, v \in V : f(u, v) \leq c(u, v)$;
- 3 **Anti-symétrie** : $\forall u, v \in V : f(u, v) = -f(v, u)$;
- 4 **Conservation du flot** : $\forall u \in V \setminus \{s, t\} : \sum_{v \in V} f(u, v) = 0$
Ce qui rentre est égal à ce qui sort pour un sommet qui n'est pas la source ou le puits.

Flots

Conservation du flot : $\forall u \in V \setminus \{s, t\} : \sum_{v \in V} f(u, v) = 0$

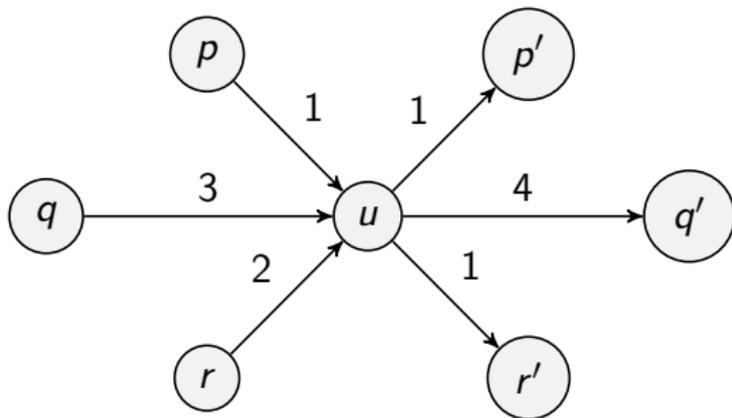


Figure – Conservation du flot : $u \neq s, t$

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

La valeur du flot est ce qui part (moins ce qui rentre) de la source.

Flots

Notation :

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

Propriétés :

- $\forall u \in V : f(u, u) = 0$;
- $\forall S \subseteq V : f(S, S) = 0$;
- $|f| = \sum_{u \in V} f(u, t)$.

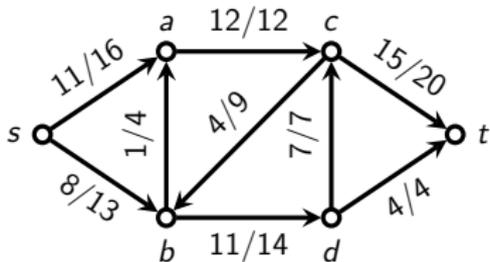
La valeur du flot est aussi ce qui rentre (moins ce qui part) de du puits.

Preuve :

$$\begin{aligned} f(V, V) &= 0 = f(s, V) + f(t, V) + f(V \setminus \{s, t\}, V) \\ &= f(s, V) + f(t, V) + 0 \\ &= |f| - f(V, t). \end{aligned}$$

Flots

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 ← un sommet de  $G$ ;
3 puits ← un autre sommet 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(flott, chemin);
8   | METTREAJOURRÉSIDUEL( $G$ ,  $G_f$ , chemin.arcs(), flott);
9   | chemin ← CHEMINAUGMENTANT( $G_f$ , source, puits);
10 renvoyer flott;
```

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

$$c_f(u, v) = \left\{ \right.$$

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

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

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

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

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

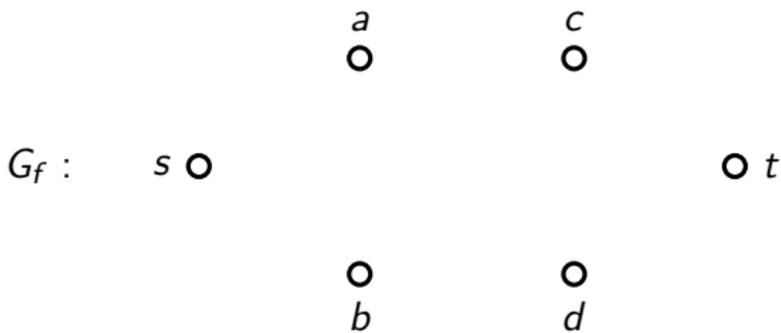
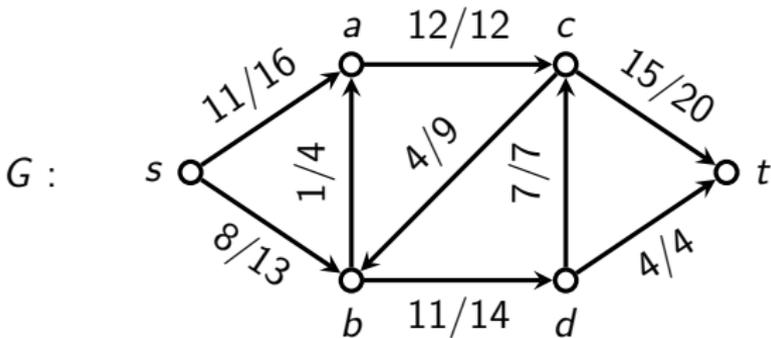
Soit $G = (V, A, c)$ un réseau muni d'un flot f . La **capacité résiduelle** d'une paire de sommets $u, v \in V$ est

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

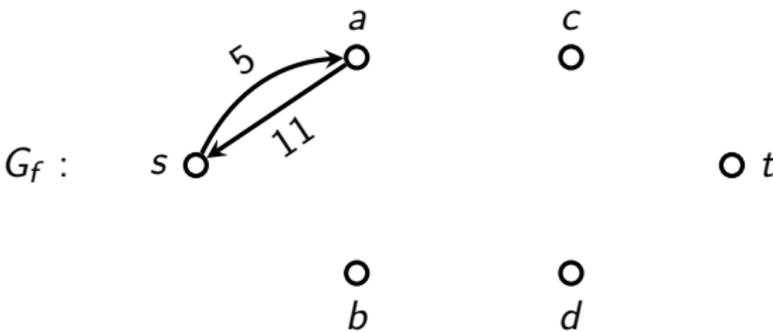
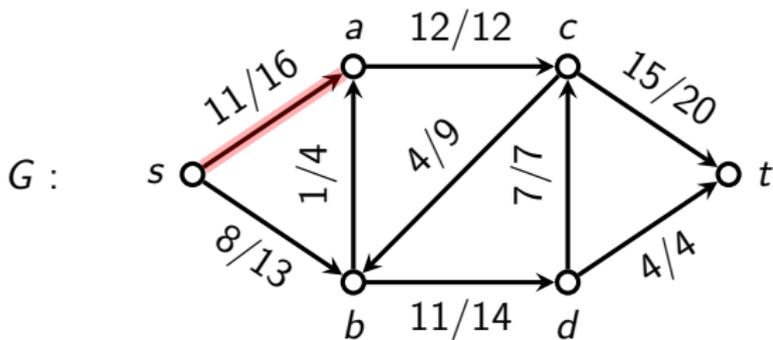
Définition 2

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 $A_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$.

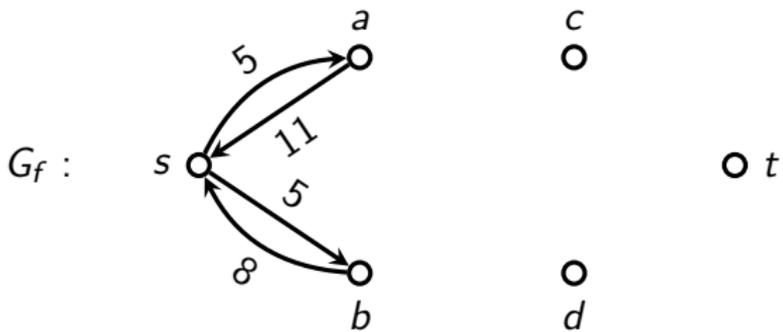
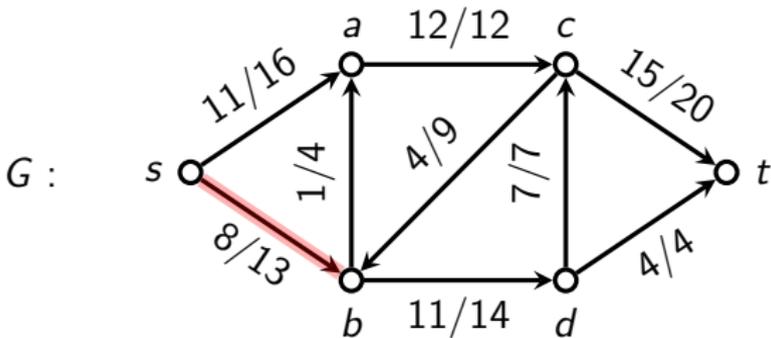
Réseaux résiduels : exemple



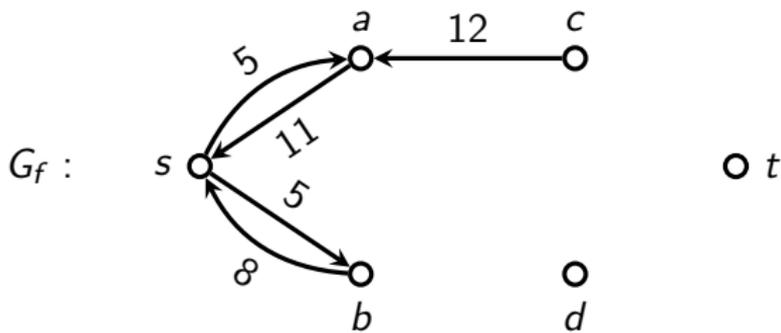
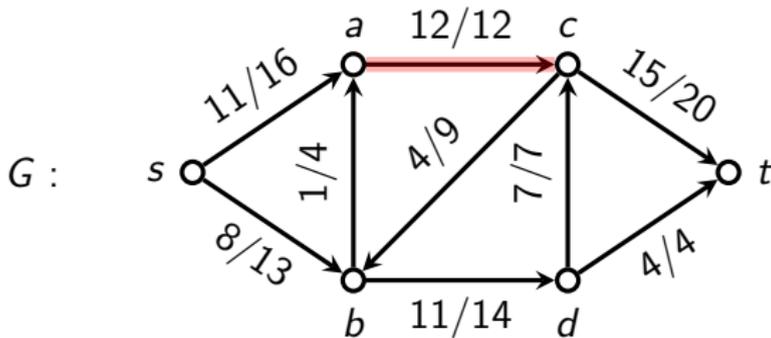
Réseaux résiduels : exemple



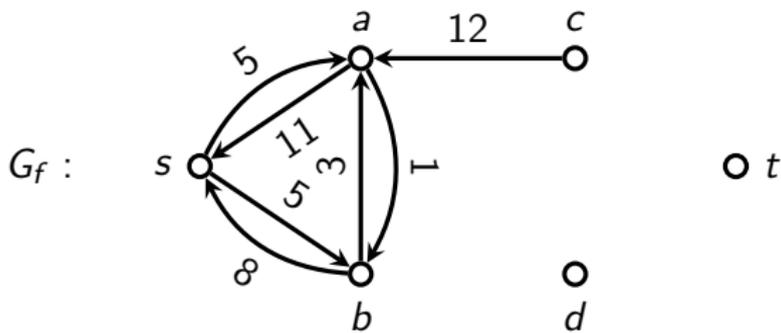
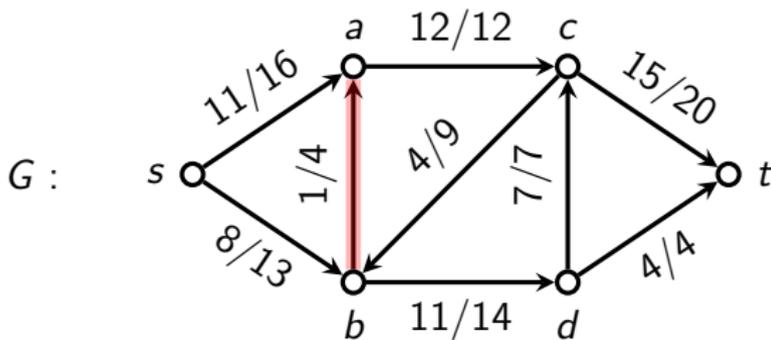
Réseaux résiduels : exemple



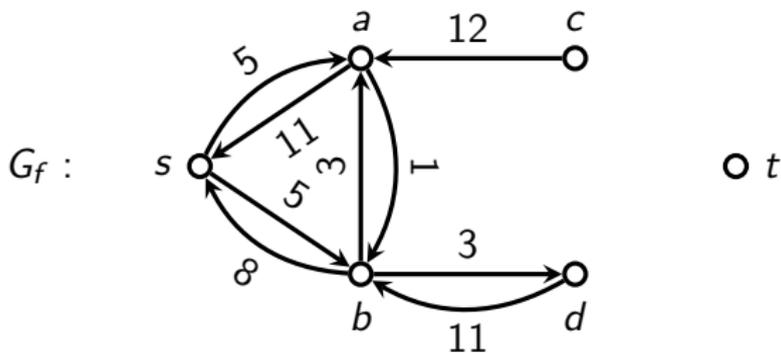
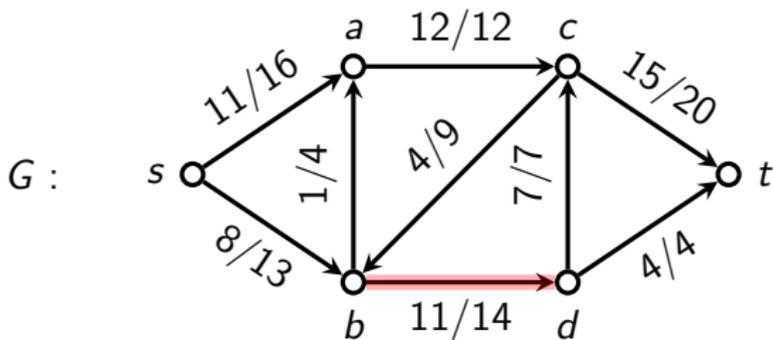
Réseaux résiduels : exemple



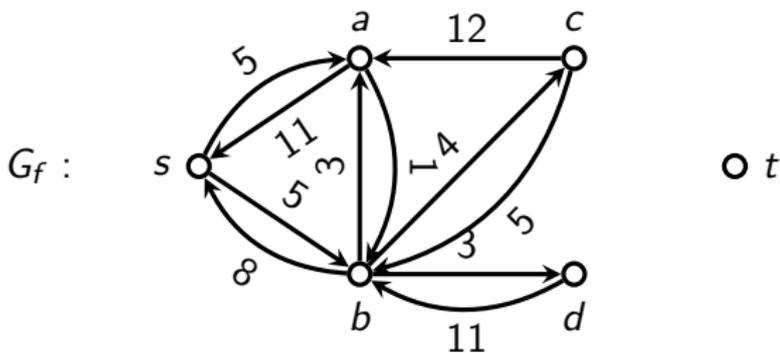
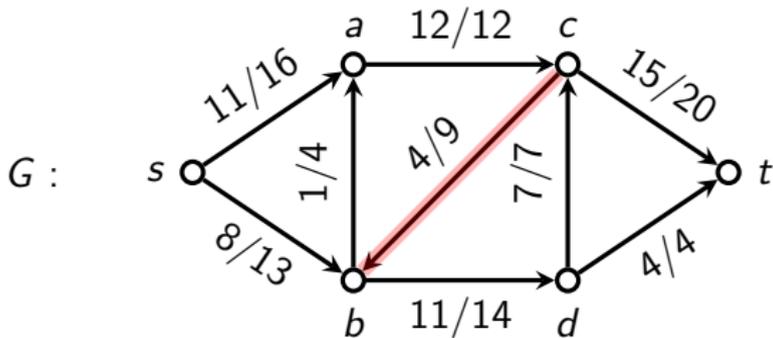
Réseaux résiduels : exemple



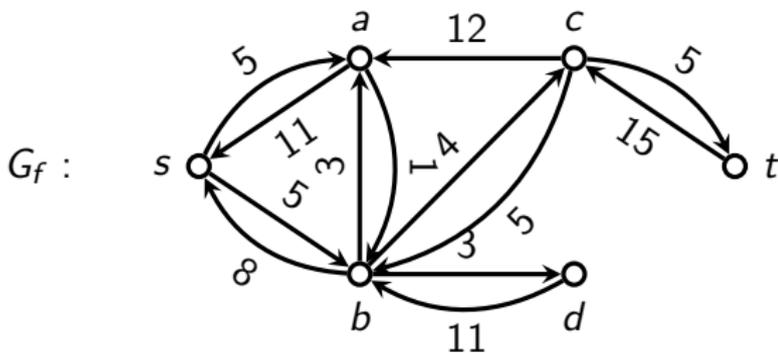
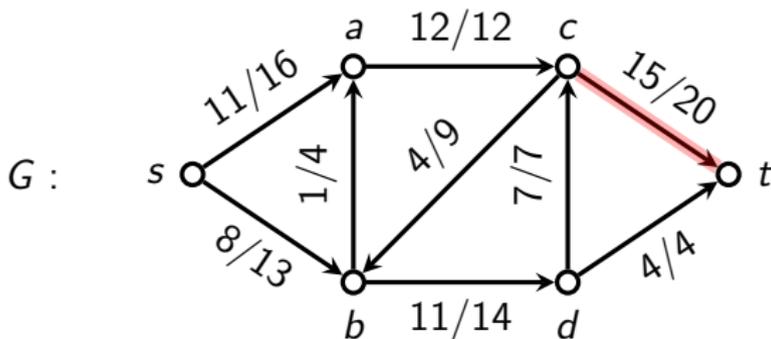
Réseaux résiduels : exemple



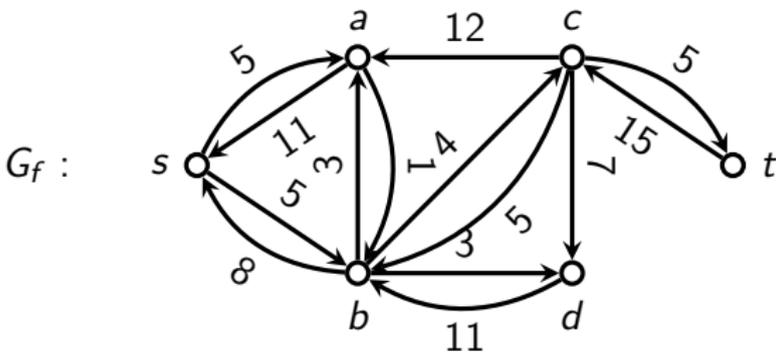
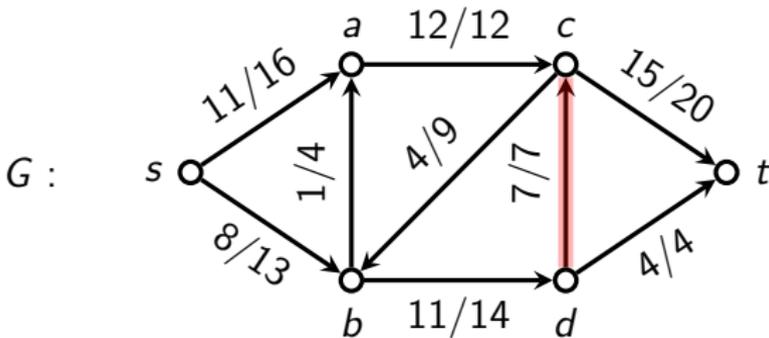
Réseaux résiduels : exemple



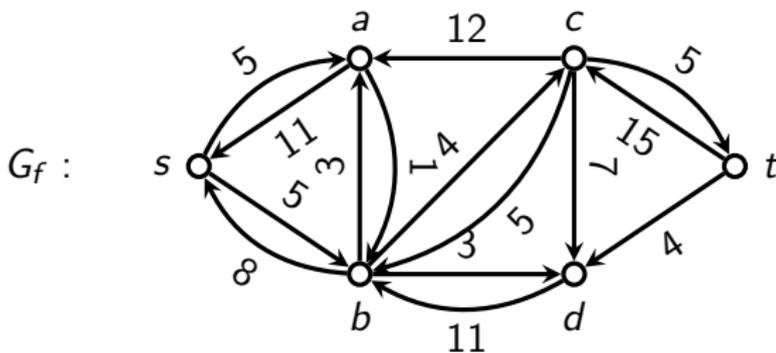
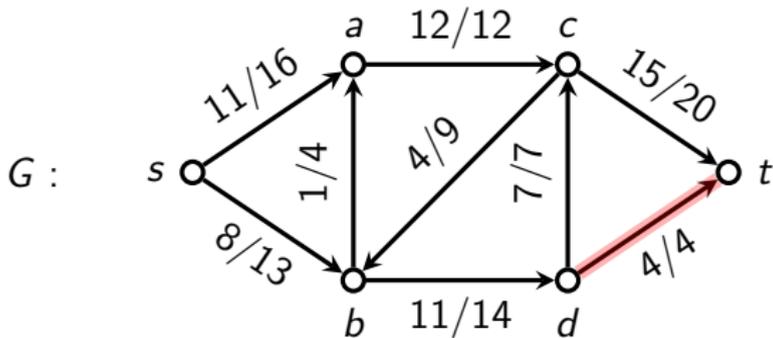
Réseaux résiduels : exemple



Réseaux résiduels : exemple



Réseaux résiduels : exemple



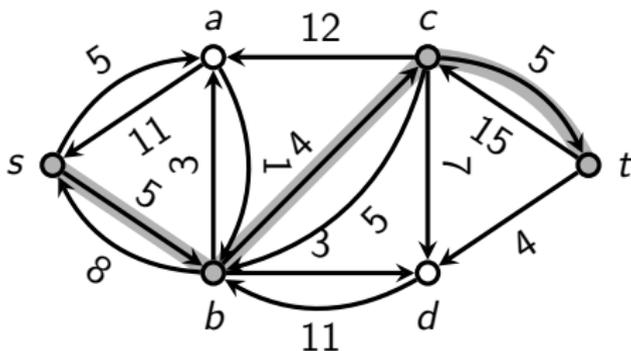
Chemins augmentants

Un **chemin augmentant** P est un chemin élémentaire (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 élémentaire (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

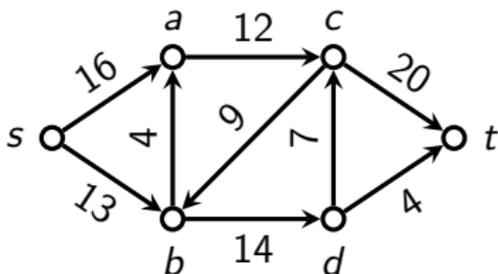
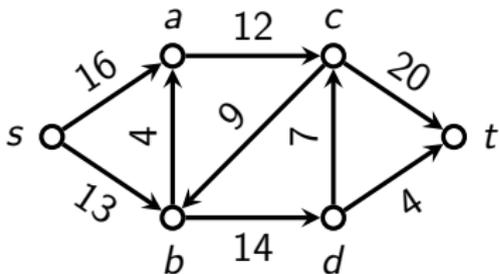
- On parle de *méthode* (et non d'*algorithme*) de Ford-Fulkerson car aucune précision n'est donnée sur la manière de choisir le chemin augmentant ;
- Il n'y a pas de consensus sur la manière optimale de choisir ce chemin (cf. TD) ;
- Algorithme d'Edmonds-Karp = méthode de Ford-Fulkerson avec sélection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

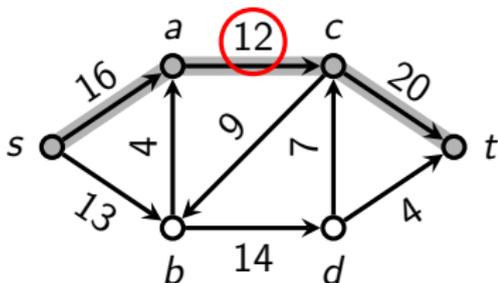
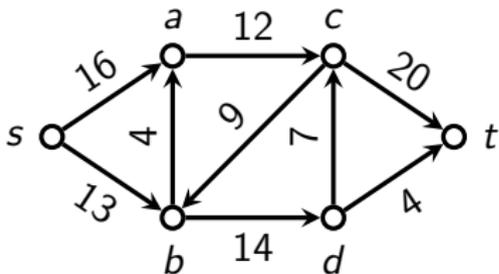
L'algorithme d'Edmonds-Karp en action

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

 G_f  G

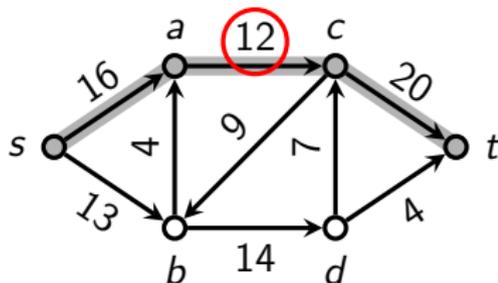
L'algorithme d'Edmonds-Karp en action

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

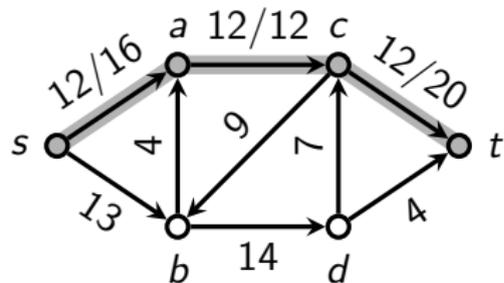
 G_f  G

L'algorithme d'Edmonds-Karp en action

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



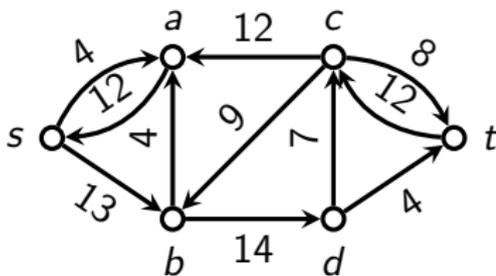
G_f



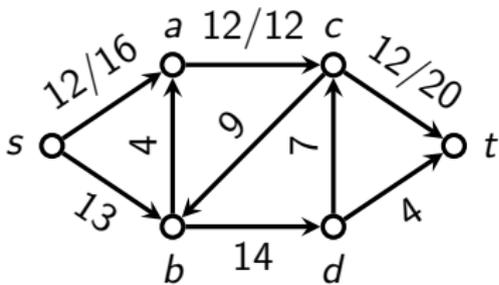
G

L'algorithme d'Edmonds-Karp en action

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



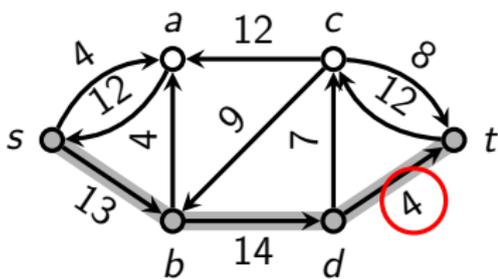
G_f



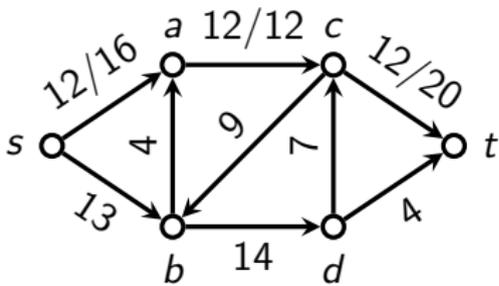
G

L'algorithme d'Edmonds-Karp en action

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



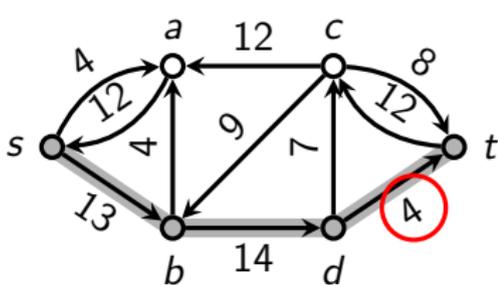
G_f



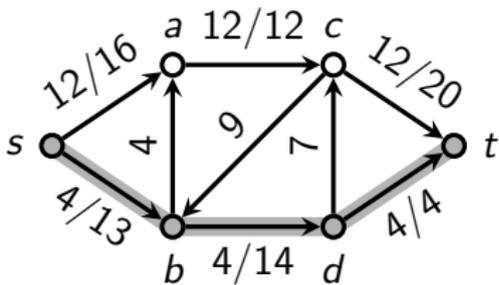
G

L'algorithme d'Edmonds-Karp en action

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



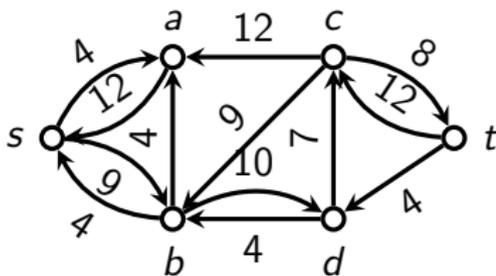
G_f



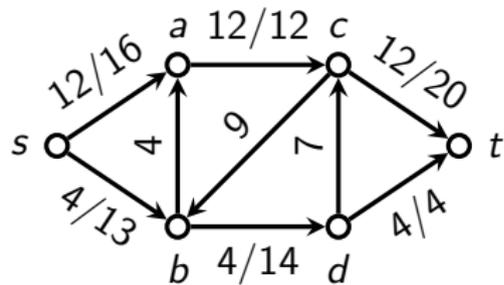
G

L'algorithme d'Edmonds-Karp en action

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



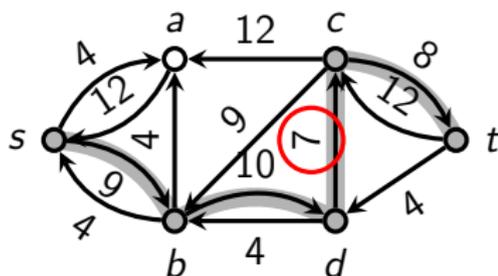
G_f



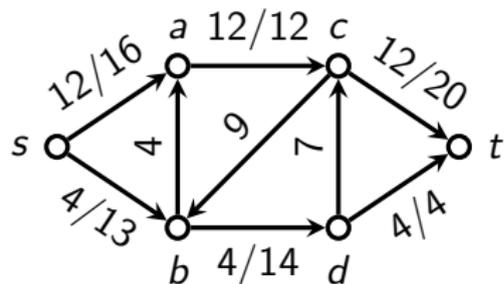
G

L'algorithme d'Edmonds-Karp en action

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



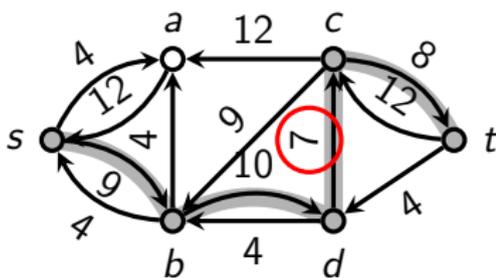
G_f



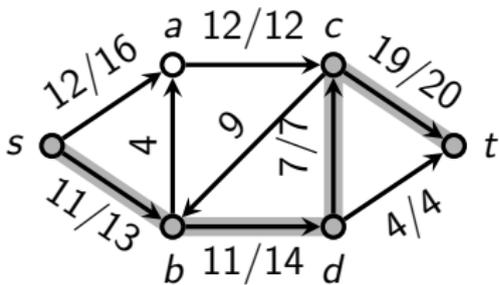
G

L'algorithme d'Edmonds-Karp en action

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



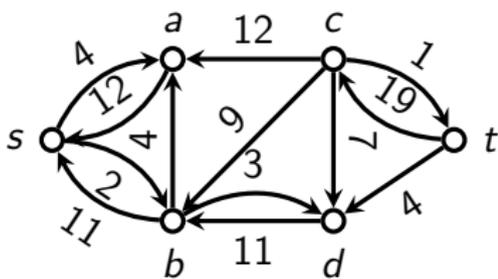
G_f



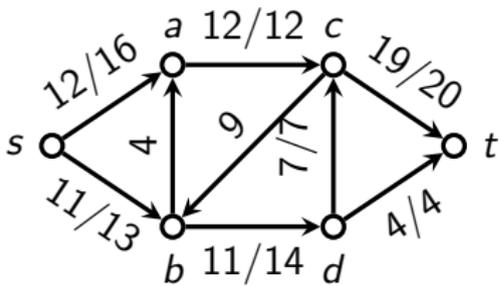
G

L'algorithme d'Edmonds-Karp en action

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



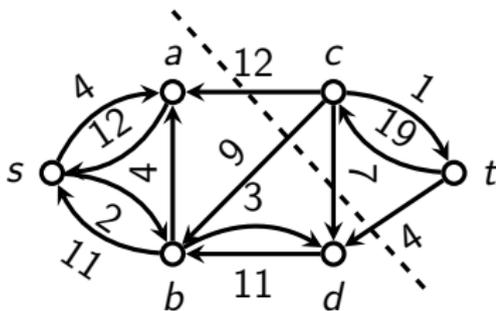
G_f



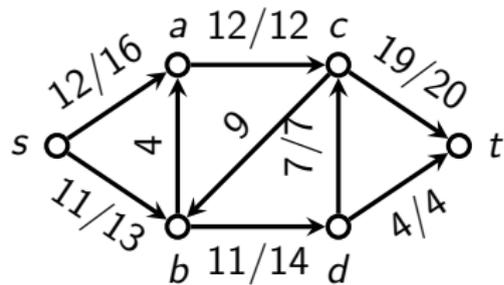
G

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 $t \in T$.

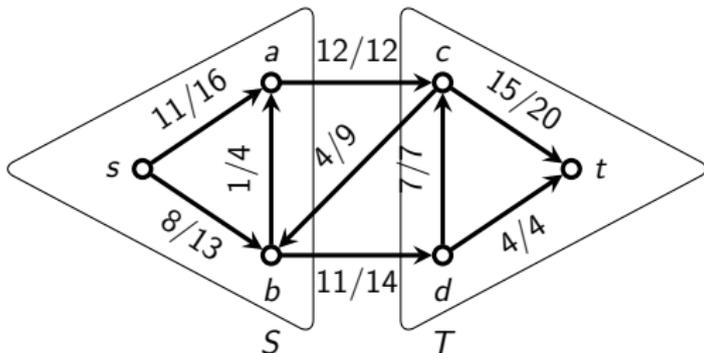
Coupes

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 :

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

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 :

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

- 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 :

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

- 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 ;

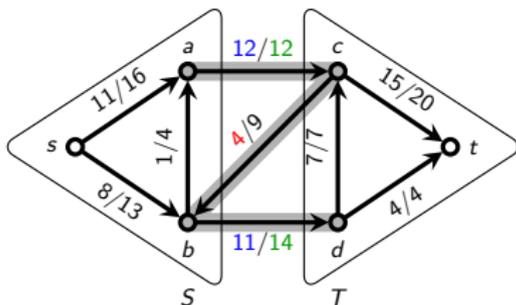
- ① son **flot associé** est le flot "net" dans les arcs entre S et T :

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

- 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 ;

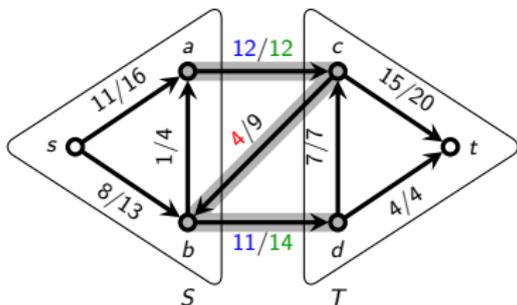
- ① son **flot associé** est le flot "net" dans les arcs entre S et T :

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

- 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 ;

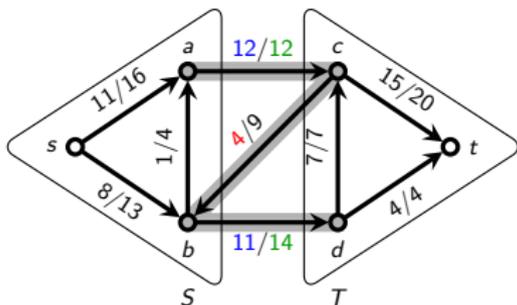
- ① son **flot associé** est le flot "net" dans les arcs entre S et T :

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

- 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

Lemme 4

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

Preuve

$$f(S, V) = f(s, V) + f(S \setminus s, V) = f(s, V) + 0 = |f|$$

$$f(S, V) = f(S, T) + f(S, V \setminus T)$$

$$= f(S, T) + f(S, S) = f(S, T) + 0 = f(S, T).$$

Lien entre les flots et les coupes

Lemme 5

Soit f un flot sur un réseau de flot G , et (S, T) une coupe sur G . Alors $f(S, T) \leq c(S, T)$.

Preuve

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

Lien entre les flots et les coupes

Corollaire 6

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 7 (Max-flow min-cut)

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 7 (Max-flow min-cut)

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 7 (Max-flow min-cut)

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 7 (Max-flow min-cut)

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 $|f| = c(S, T)$.*

Théorème *max-flow min-cut*

Preuve

- Pour tout flot f' et toute coupe (S, T) on a $|f'| = f'(S, T) \leq c(S, T)$.

Théorème *max-flow min-cut*

Preuve

- Pour tout flot f' et toute coupe (S, T) on a $|f'| = f'(S, T) \leq c(S, T)$.
- $1 \Rightarrow 2$: si f est maximum alors G_f ne contient pas de chemin augmentant ;

Théorème *max-flow min-cut*

Preuve

- Pour tout flot f' et toute coupe (S, T) on a $|f'| = f'(S, T) \leq c(S, T)$.
- $1 \Rightarrow 2$: si f est maximum alors G_f ne contient pas de chemin augmentant ;
- $2 \Rightarrow 3$: si G_f ne contient pas de chemin augmentant, alors soit S l'ensemble des sommets atteints lors de la dernière exploration de G_f et $T = V \setminus S$.

Alors pour cette coupe $|f| = f(S, T) = c(S, T)$. Sinon il existerait $u \in S, v \in T$ tels que $f(u, v) < c(u, v)$ et on aurait v dans S .

Théorème *max-flow min-cut*

Preuve

- Pour tout flot f' et toute coupe (S, T) on a $|f'| = f'(S, T) \leq c(S, T)$.
- $1 \Rightarrow 2$: si f est maximum alors G_f ne contient pas de chemin augmentant ;
- $2 \Rightarrow 3$: si G_f ne contient pas de chemin augmentant, alors soit S l'ensemble des sommets atteints lors de la dernière exploration de G_f et $T = V \setminus S$.

Alors pour cette coupe $|f| = f(S, T) = c(S, T)$. Sinon il existerait $u \in S, v \in T$ tels que $f(u, v) < c(u, v)$ et on aurait v dans S .

- $3 \Rightarrow 1$: si on a une coupe (S, T) pour G telle que $|f| = c(S, T)$. Si f' est un autre flot, $|f'| \leq c(S, T) = |f|$.

Obtention d'une coupe minimale

- On cherche un flot maximal. On note G_f le graphe résiduel final.

Obtention d'une coupe minimale

- On cherche un flot maximal. On note G_f le graphe résiduel final.
- On note S les sommets accessibles de s dans G_f lors de la dernière recherche infructueuse d'un chemin augmentant et $T = V \setminus S$.

Obtention d'une coupe minimale

- On cherche un flot maximal. On note G_f le graphe résiduel final.
- On note S les sommets accessibles de s dans G_f lors de la dernière recherche infructueuse d'un chemin augmentant et $T = V \setminus S$.
- Alors (S, T) est une coupe minimale.

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

Simple parcours en largeur de la source au puits (sera appliqué sur G_f) :

Algorithme 2 : CHEMINAUGMENTANT(G , source, puits)

Entrées : un graphe orienté pondéré G , une source et un 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  $\neg$  déjà_visités[sommet] alors
9         déjà_visités[sommet] ← VRAI;
10        pour chaque  $v \in G$ .successeurs(sommet) faire
11            si  $\neg$  déjà_visités[ $v$ ] alors
12                a_traiter.enfiler( $v$ );
13                si parents[ $v$ ] = NIL alors
14                    parents[ $v$ ] ← sommet;
15 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 (appliqué sur G_f , renvoie un chemin de G_f) :

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  $v \leftarrow$  fin;
3 tant que  $v \neq$  début faire
4   | si parents[ $v$ ] = NIL alors renvoyer NIL;
5   | chemin.ajouter_arc(parents[ $v$ ],  $v$ ,  $G$ .poids_arc(parents[ $v$ ],  $v$ ));
6   |  $v \leftarrow$  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 : AUGMENTER_FLOT(f, P)

Entrées : un flot f , un chemin P .

Résultat : f augmente au maximum le long du chemin P .

```
1 capacité_min ← min { $c \mid (u, v, c) \in P.\text{arcs}()$ };
2 pour chaque  $(u, v, c) \in P.\text{arcs}()$  faire
3   | si  $(u, v) \in A$  alors
4   |   |  $f(u, v) \leftarrow f(u, v) + \text{capacité\_min}$  ;
5   | sinon
6   |   |  $f(v, u) \leftarrow f(v, u) - \text{capacité\_min}$ ;
```

Mise à jour du résiduel

On met à jour 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 : met à jour les arcs de G_f .

```

1  $c_f \leftarrow$  tableau associatif;
2 pour chaque  $(u, v) \in \text{arcs}$  faire
3   | si  $G.\text{contient\_arc}(u, v)$  alors
4   |   |  $c_f(u, v) \leftarrow G.\text{poids\_arc}(u, v) - f(u, v)$  ;
5   |   |  $c_f(v, u) \leftarrow f(u, v)$  ;
6   | sinon
7   |   |  $c_f(u, v) \leftarrow f(v, u)$  ;
8   |   |  $c_f(v, u) \leftarrow G.\text{poids\_arc}(v, u) - f(v, u)$  ;
9 pour chaque  $(u, v) \in \text{arcs}$  faire
10 | si  $c_f(u, v) > 0$  alors  $G_f.\text{ajouter\_arc}(u, v, c_f(u, v))$  ;
11 | sinon  $G_f.\text{supprimer\_arc}(u, v)$  ;
12 | si  $c_f(v, u) > 0$  alors  $G_f.\text{ajouter\_arc}(v, u, c_f(v, u))$  ;
13 | sinon  $G_f.\text{supprimer\_arc}(v, u)$  ;

```

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  $s$  ← un sommet de  $G$ ;
3  $t$  ← un autre sommet de  $G$ ;
4  $G_f$  ←  $G$ 
5 chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ )
6 tant que chemin  $\neq$  NIL faire
7   | AUGMENTERFLOT(flott, chemin)
8   | METTREAJOURRÉSIDUEL( $G$ ,  $G_f$ , chemin.arcs(), flott)
9   | chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ )
10 renvoyer flott;
```

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  $s$  ← un sommet de  $G$ ;
3  $t$  ← un autre sommet de  $G$ ;
4  $G_f$  ←  $G$ ;                                                         //  $O(|V| + |A|)$ 
5 chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ )
6 tant que chemin  $\neq$  NIL faire
7   AUGMENTERFLOT(flots, chemin)
8   METTREAJOURRÉSIDUEL( $G$ ,  $G_f$ , chemin.arcs(), flots)
9   chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ )
10 renvoyer flots;
```

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  $s$  ← un sommet de  $G$ ;
3  $t$  ← un autre sommet de  $G$ ;
4  $G_f$  ←  $G$ ;                                                         //  $O(|V| + |A|)$ 
5 chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ );                             //  $O(|V| + |A|)$ 
6 tant que chemin  $\neq$  NIL faire
7   AUGMENTERFLOT(flott, chemin)
8   METTREAJOURRÉSIDUEL( $G$ ,  $G_f$ , chemin.arcs(), flott)
9   chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ )
10 renvoyer flott;
```

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  $s$  ← un sommet de  $G$ ;
3  $t$  ← un autre sommet de  $G$ ;
4  $G_f$  ←  $G$ ;                                                         //  $O(|V| + |A|)$ 
5 chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ );                             //  $O(|V| + |A|)$ 
6 tant que chemin  $\neq$  NIL faire
7   AUGMENTERFLOT(flots, chemin);                                     //  $O(|V|)$ 
8   METTREAJOURRÉSIDUEL( $G$ ,  $G_f$ , chemin.arcs(), flots)
9   chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ )
10 renvoyer flots;
```

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  $s$  ← un sommet de  $G$ ;
3  $t$  ← un autre sommet de  $G$ ;
4  $G_f$  ←  $G$ ;                                                         //  $O(|V| + |A|)$ 
5 chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ );                             //  $O(|V| + |A|)$ 
6 tant que chemin  $\neq$  NIL faire
7   AUGMENTERFLOT(flott, chemin);                                     //  $O(|V|)$ 
8   METTREAJOURRÉSIDUEL( $G$ ,  $G_f$ , chemin.arcs(), flott);           //  $O(|V|)$ 
9   chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ )
10 renvoyer flott;
```

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  $s$  ← un sommet de  $G$ ;
3  $t$  ← un autre sommet de  $G$ ;
4  $G_f$  ←  $G$ ;                                                           //  $O(|V| + |A|)$ 
5 chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ );                             //  $O(|V| + |A|)$ 
6 tant que chemin  $\neq$  NIL faire
7   AUGMENTERFLOT(flott, chemin);                                     //  $O(|V|)$ 
8   METTREAJOURRÉSIDUEL( $G$ ,  $G_f$ , chemin.arcs(), flott);           //  $O(|V|)$ 
9   chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ );                         //  $O(|V| + |A|)$ 
10 renvoyer flott;
```

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  $s$  ← un sommet de  $G$ ;
3  $t$  ← un autre sommet de  $G$ ;
4  $G_f$  ←  $G$ ;                                                         //  $O(|V| + |A|)$ 
5 chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ );                             //  $O(|V| + |A|)$ 
6 tant que chemin  $\neq$  NIL faire
7   AUGMENTERFLOT(flott, chemin);                                     //  $O(|V|)$ 
8   METTREAJOURRÉSIDUEL( $G$ ,  $G_f$ , chemin.arcs(), flott);           //  $O(|V|)$ 
9   chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ );                         //  $O(|V| + |A|)$ 
10 renvoyer flott;
```

- 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  $s$  ← un sommet de  $G$ ;
3  $t$  ← un autre sommet 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(flott, chemin);                                     //  $O(|V|)$ 
8   METTREAJOURRÉSIDUEL( $G$ ,  $G_f$ , chemin.arcs(), flott);           //  $O(|V|)$ 
9   chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ );                         //  $O(|V| + |A|)$ 
10 renvoyer flott;
```

- 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  $s$  ← un sommet de  $G$ ;
3  $t$  ← un autre sommet de  $G$ ;
4  $G_f$  ←  $G$ ;                                                         //  $O(|V| + |A|)$ 
5 chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ );                             //  $O(|V| + |A|)$ 
6 tant que chemin  $\neq$  NIL faire
7   AUGMENTERFLOT(flott, chemin);                                     //  $O(|V|)$ 
8   METTREAJOURRÉSIDUEL( $G$ ,  $G_f$ , chemin.arcs(), flott);           //  $O(|V|)$ 
9   chemin ← CHEMINAUGMENTANT( $G_f$ ,  $s$ ,  $t$ );                         //  $O(|V| + |A|)$ 
10 renvoyer flott;
```

- 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 $O(|V||A|^2)$;

Flots entiers

Si $G = (V, A, c)$ un réseau avec des capacités entières.

- L'algorithme de Ford-Fulkerson calcule des flots entiers.
- Il calcule un flot maximal même parmi tous les flots à valeurs réelles.
- On a donc le théorème d'intégrité :

Théorème 8 (Flow integrality theorem)

Soit $G = (V, A, c)$ un réseau de flot avec $c(u, v) \in \mathbb{N}$ pour tout u, v . Alors il existe un flot maximal qui satisfait $f(u, v) \in \mathbb{N}$ pour tout u, v .

Bibliographie

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

Introduction to Algorithms.

MIT Press, 3ème édition, 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.