Algorithmique des graphes

8 — Flots et applications (2)

Marie-Pierre Béal (Cours d'Anthony Labarre)

Université Gustave Eiffel

L'algorithme de Dinitz

- Parmi les algorithmes plus efficaces pour calculer des flots maximum, il y a celui de Dinitz ;

L'algorithme de Dinitz

- Parmi les algorithmes plus efficaces pour calculer des flots maximum, il y a celui de Dinitz ;
- Il réalise en $O(|V|^2|A|)$ ce que Edmonds-Karp fait en $O(|V||A|^2)$ (mieux puisque $|A| \geq |V|$ pour un réseau faiblement connexe) ;

L'algorithme de Dinitz

- Parmi les algorithmes plus efficaces pour calculer des flots maximum, il y a celui de Dinitz ;
- Il réalise en $O(|V|^2|A|)$ ce que Edmonds-Karp fait en $O(|V||A|^2)$ (mieux puisque $|A| \geq |V|$ pour un réseau faiblement connexe) ;
- On peut rabaisser sa complexité à $O(|V||A| \log |V|)$ grâce aux *link-cut trees* ou *dynamic trees* [**?**].

L'algorithme de Dinitz

- Parmi les algorithmes plus efficaces pour calculer des flots maximum, il y a celui de Dinitz ;
- Il réalise en $O(|V|^2|A|)$ ce que Edmonds-Karp fait en $O(|V||A|^2)$ (mieux puisque $|A| \geq |V|$ pour un réseau faiblement connexe) ;
- On peut rabaisser sa complexité à $O(|V||A| \log |V|)$ grâce aux *link-cut trees* ou *dynamic trees* [**?**].
- Il est assez similaire à l'algorithme d'Edmonds-Karp ;

Présentation de l'algorithme

L'algorithme de Dinitz suit la méthode de Edmonds et Karp, à deux différences près :

Présentation de l'algorithme

L'algorithme de Dinitz suit la méthode de Edmonds et Karp, à deux différences près :

1. on maintient une structure G_L contenant **tous** les plus courts chemins de s à t dans G_f ;

Présentation de l'algorithme

L'algorithme de Dinitz suit la méthode de Edmonds et Karp, à deux différences près :

1. on maintient une structure G_L contenant **tous** les plus courts chemins de s à t dans G_f ;
2. au lieu d'augmenter le flot sur un seul chemin de s à t dans G_f, on l'augmente sur **tous** les chemins trouvés dans G_L ;

Présentation de l'algorithme

L'algorithme de Dinitz suit la méthode de Edmonds et Karp, à deux différences près :

1. on maintient une structure G_L contenant **tous** les plus courts chemins de s à t dans G_f ;
2. au lieu d'augmenter le flot sur un seul chemin de s à t dans G_f, on l'augmente sur **tous** les chemins trouvés dans G_L ;
3. on met ensuite à jour G_f et G_L ;

Présentation de l'algorithme

L'algorithme de Dinitz suit la méthode de Edmonds et Karp, à deux différences près :

1. on maintient une structure G_L contenant **tous** les plus courts chemins de s à t dans G_f ;
2. au lieu d'augmenter le flot sur un seul chemin de s à t dans G_f, on l'augmente sur **tous** les chemins trouvés dans G_L ;
3. on met ensuite à jour G_f et G_L ;

Comme dans la méthode de Ford-Fulkerson, on s'arrête quand G_f ne permet plus d'augmentation (via G_L) ; le flot trouvé est alors maximum.

Le graphe de parcours en largeur

Définition 1

Soit $G = (V, A)$ un graphe orienté, et $s \in V$. Le **graphe de parcours en largeur de G au départ de s** est le graphe orienté G_L défini par :

- V_i = sommets de G à distance i de s ; d = distance du sommet le plus éloigné ;
- $V(G_L) = \cup_{i=0}^{d} V_i$;
- $A(G_L) = \cup_{i=1}^{d} A_i$, où $A_i = \{(u, v) \mid u \in V_{i-1}, v \in V_i\}$.

Attention : distance = nombre d'arcs, **pas** somme des poids.

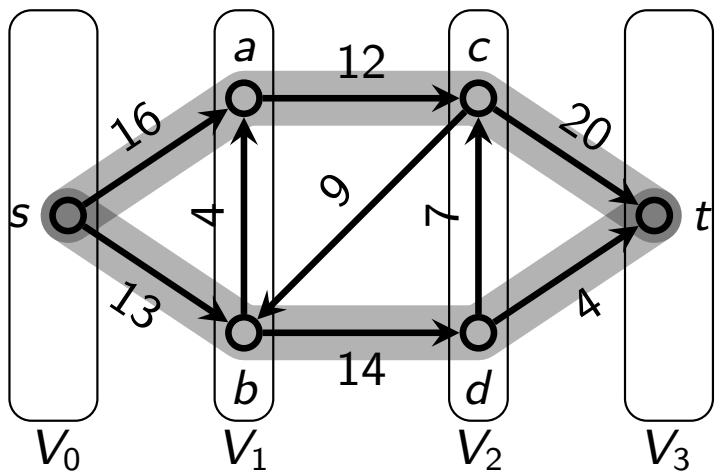
Le graphe de parcours en largeur

Définition 1

Soit $G = (V, A)$ un graphe orienté, et $s \in V$. Le **graphe de parcours en largeur de G au départ de s** est le graphe orienté G_L défini par :

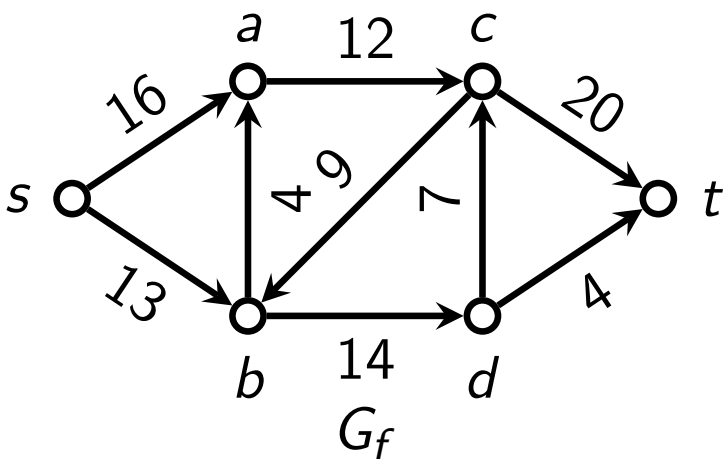
- V_i = sommets de G à distance i de s ; d = distance du sommet le plus éloigné ;
- $V(G_L) = \cup_{i=0}^{d} V_i$;
- $A(G_L) = \cup_{i=1}^{d} A_i$, où $A_i = \{(u, v) \mid u \in V_{i-1}, v \in V_i\}$.

Attention : distance = nombre d'arcs, **pas** somme des poids.

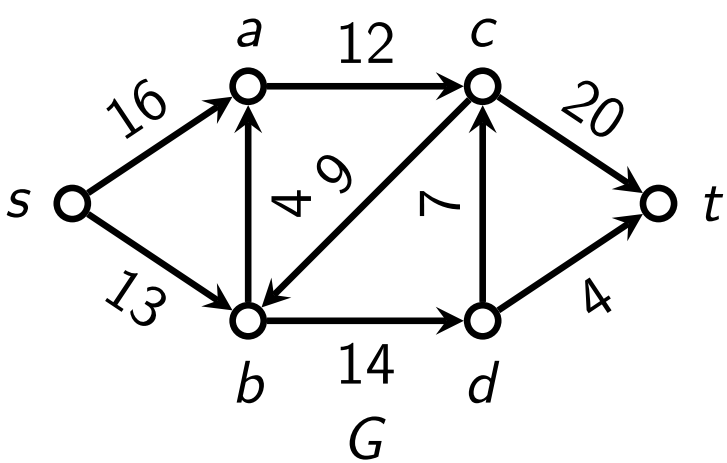


L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



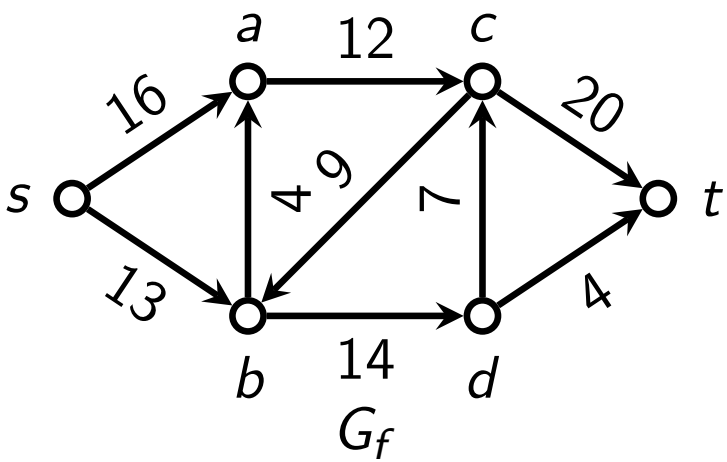
G_f



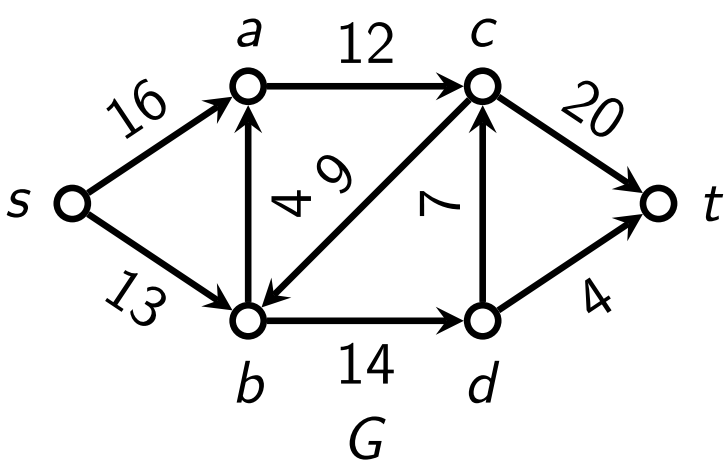
G

L'algorithme de Dinitz

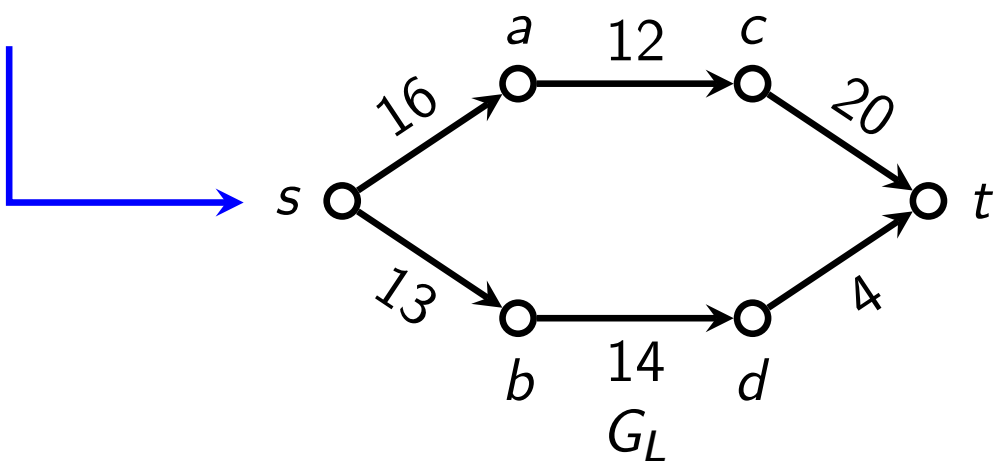
Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



G_f



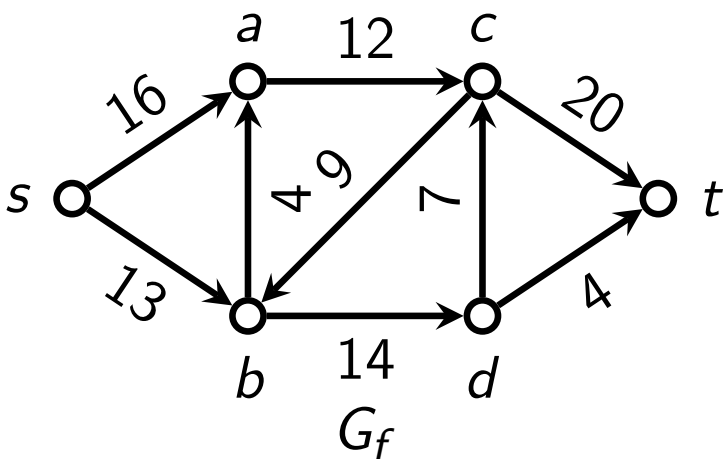
G



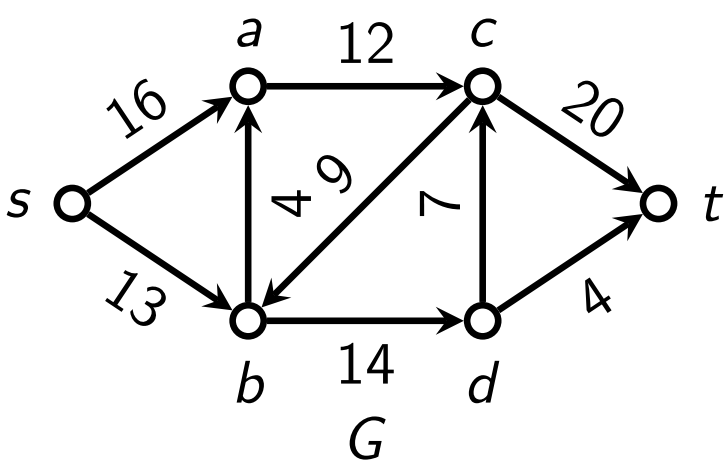
G_L

L'algorithme de Dinitz

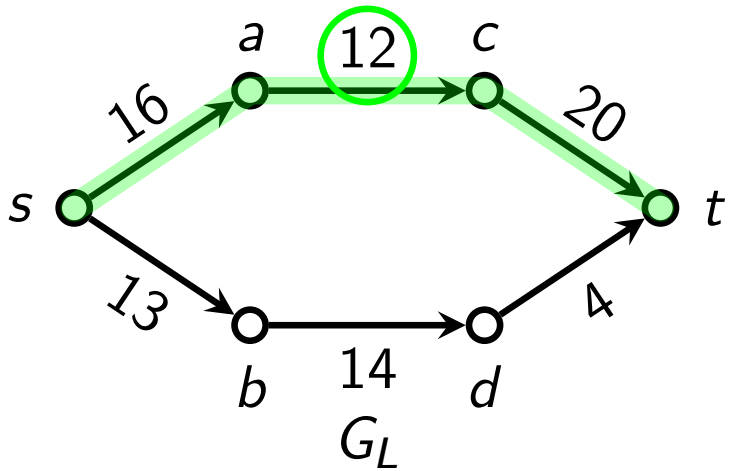
Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



G_f



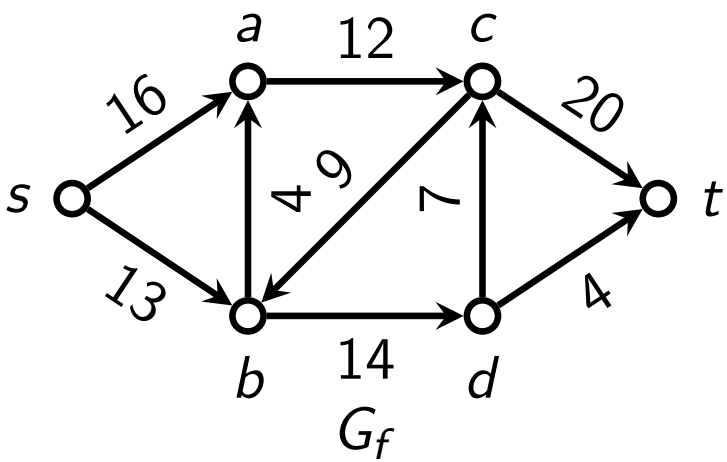
G



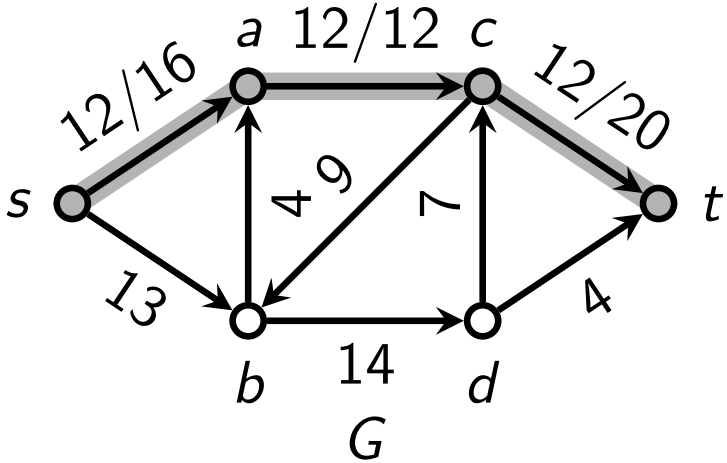
G_L

L'algorithme de Dinitz

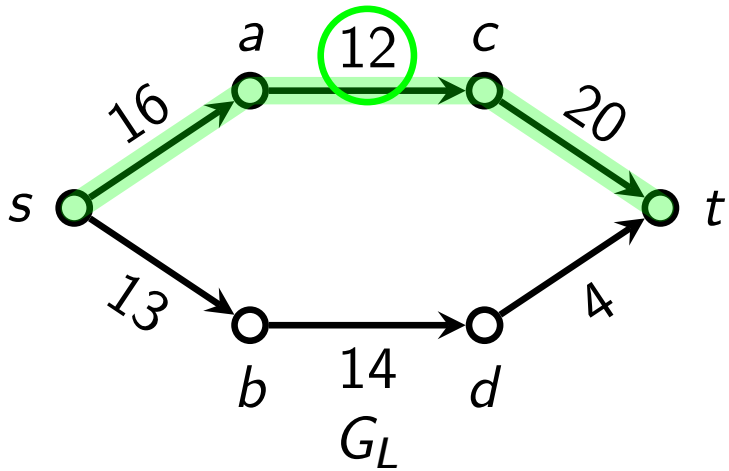
Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



G_f



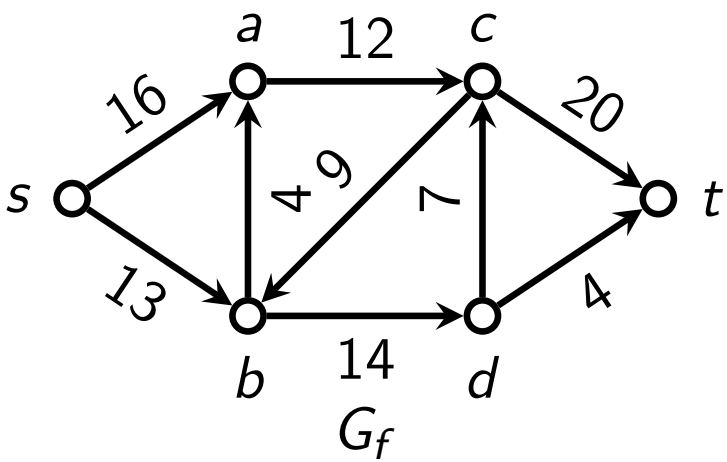
G



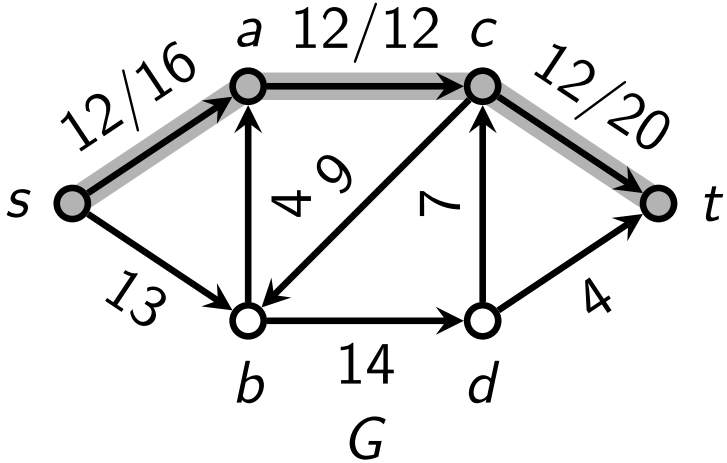
G_L

L'algorithme de Dinitz

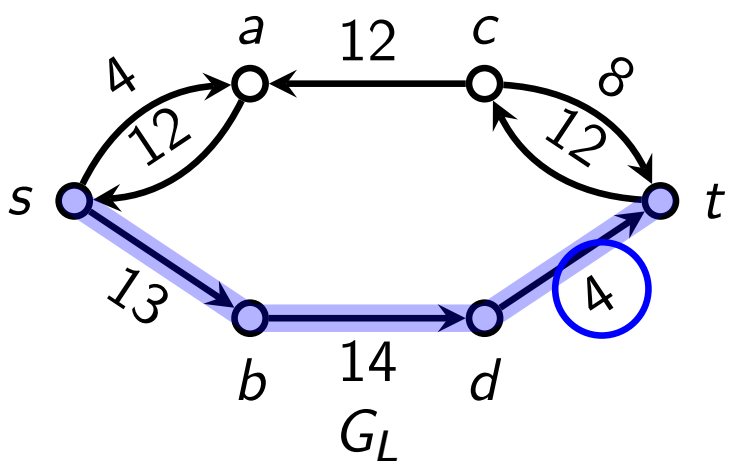
Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



G_f



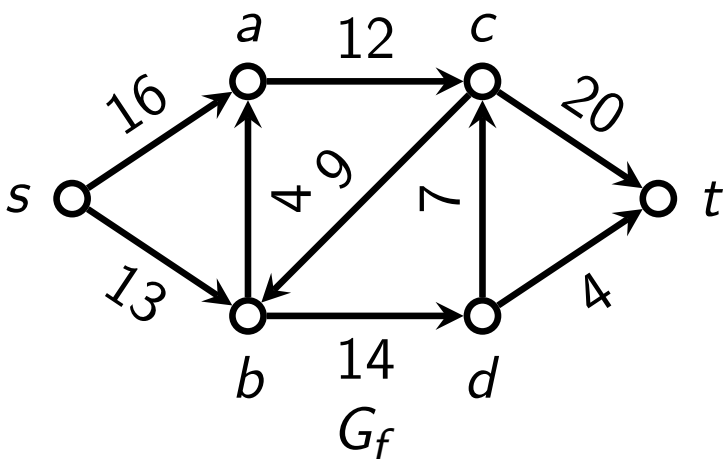
G



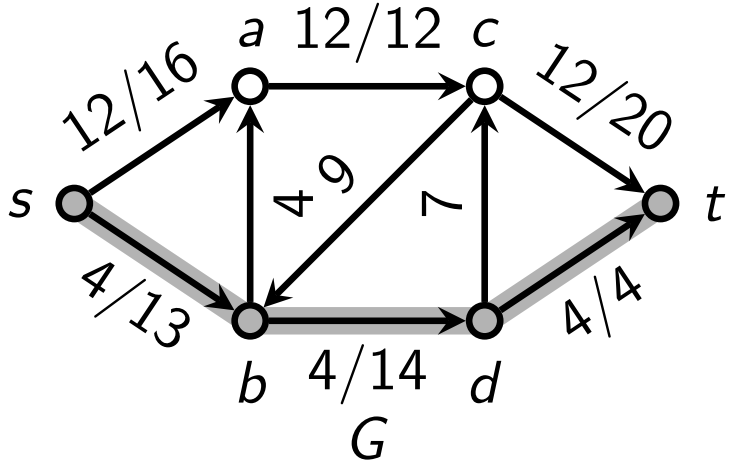
G_L

L'algorithme de Dinitz

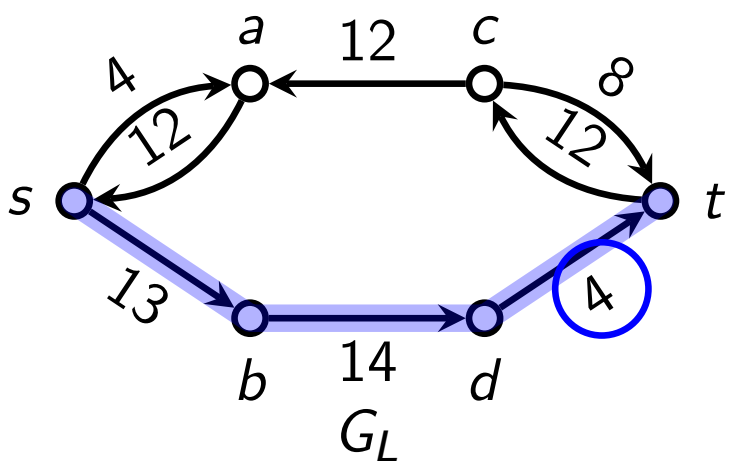
Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



G_f



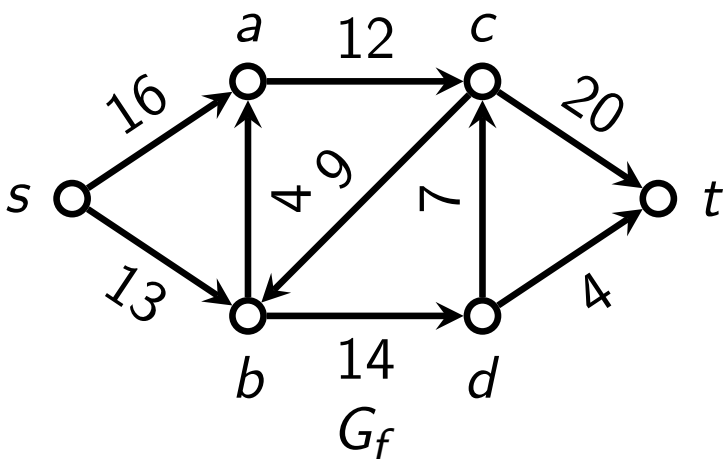
G



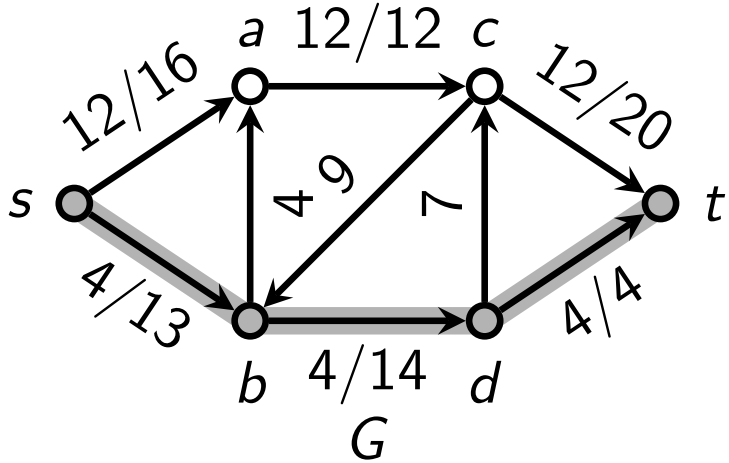
G_L

L'algorithme de Dinitz

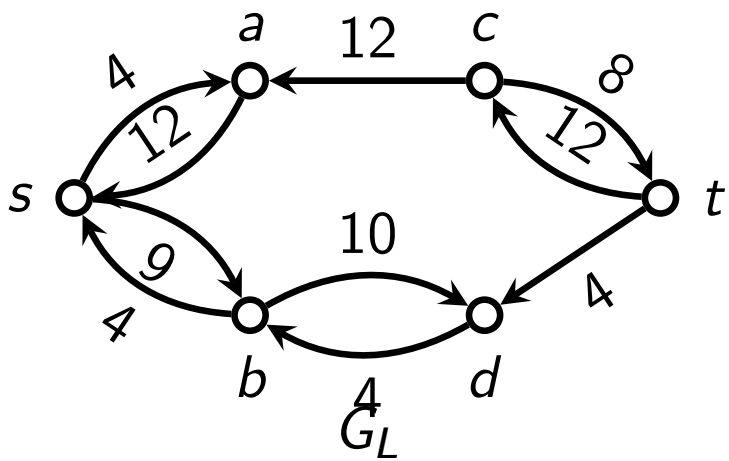
Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



G_f



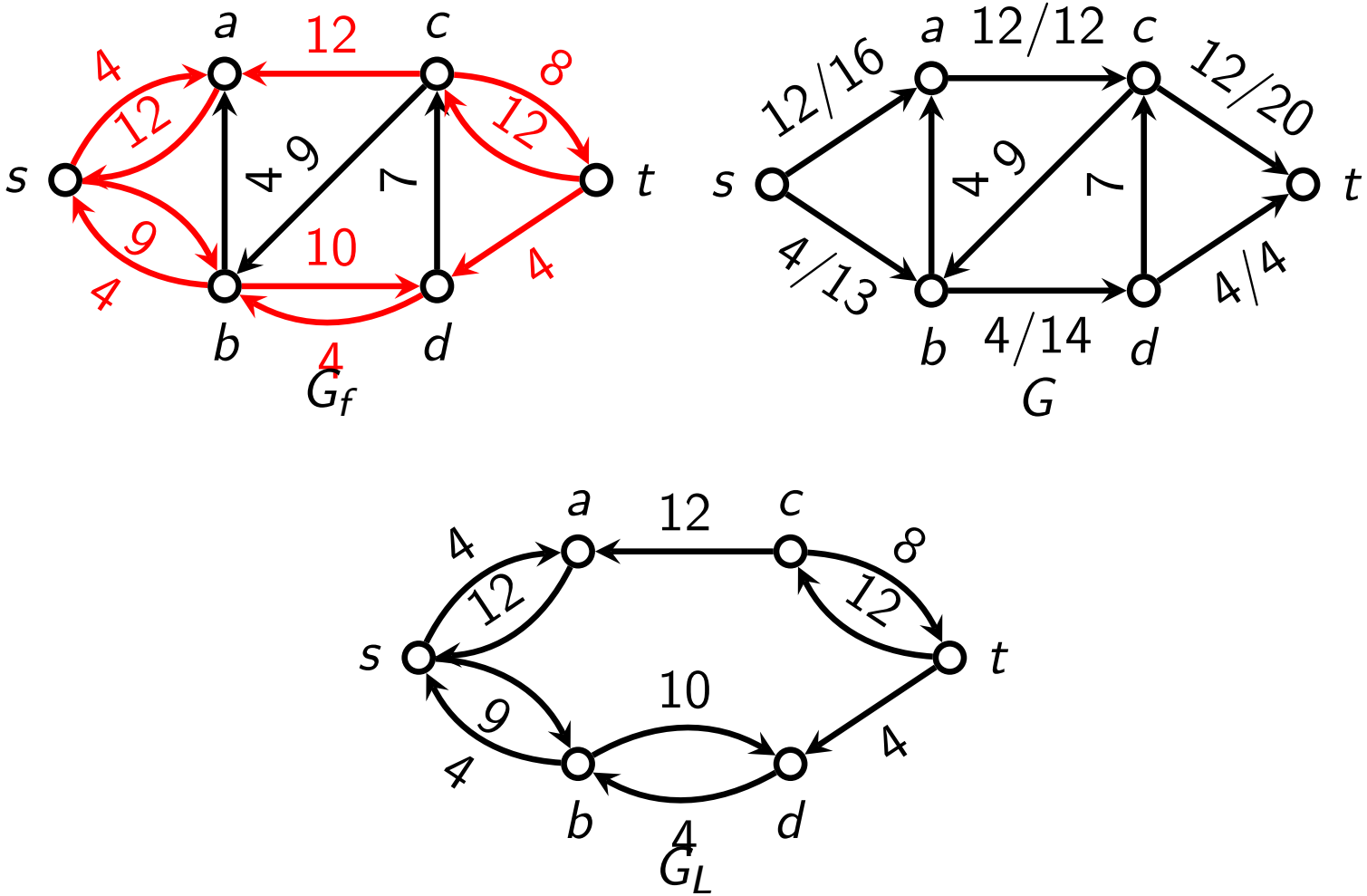
G



G_L

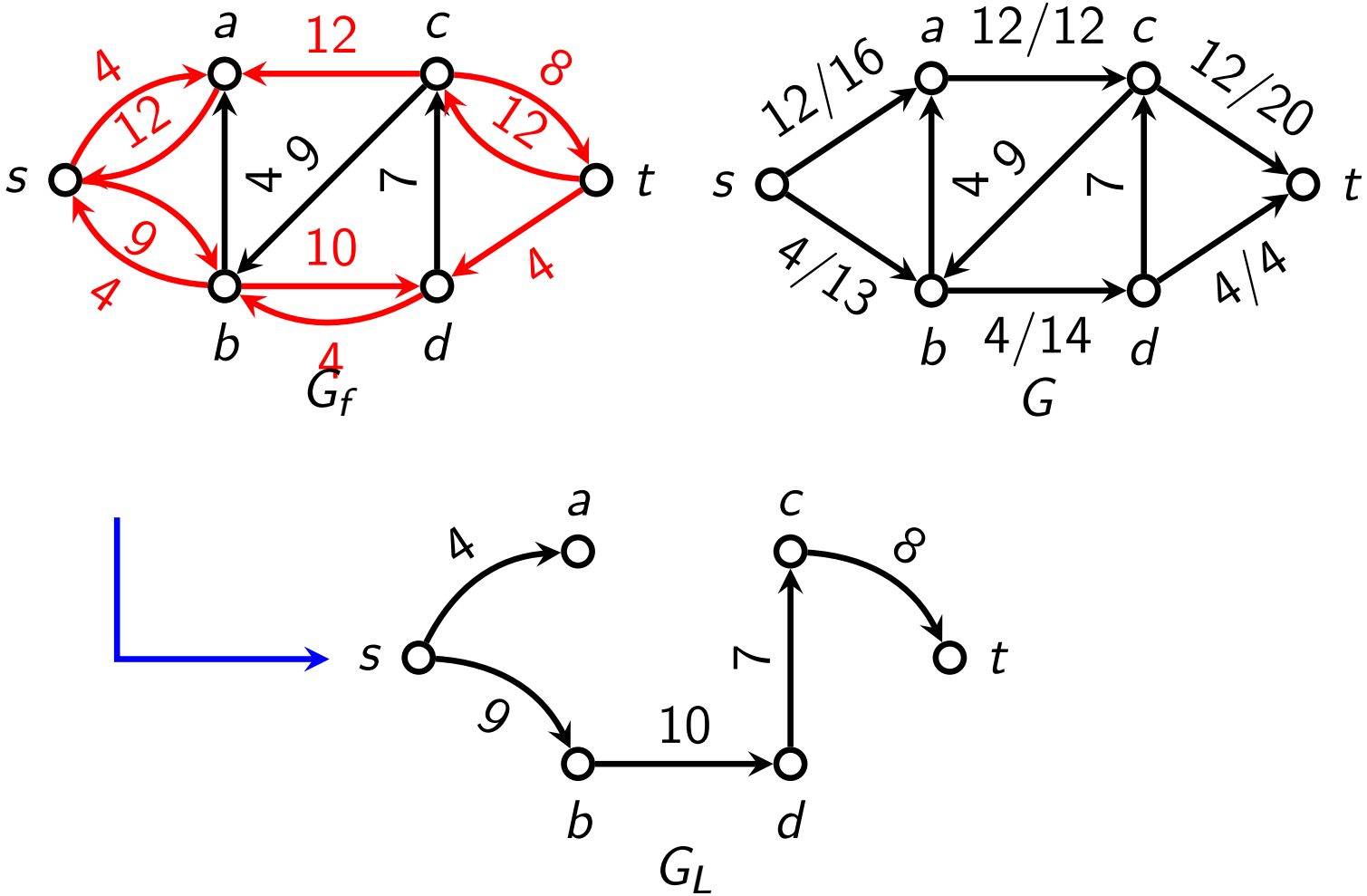
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



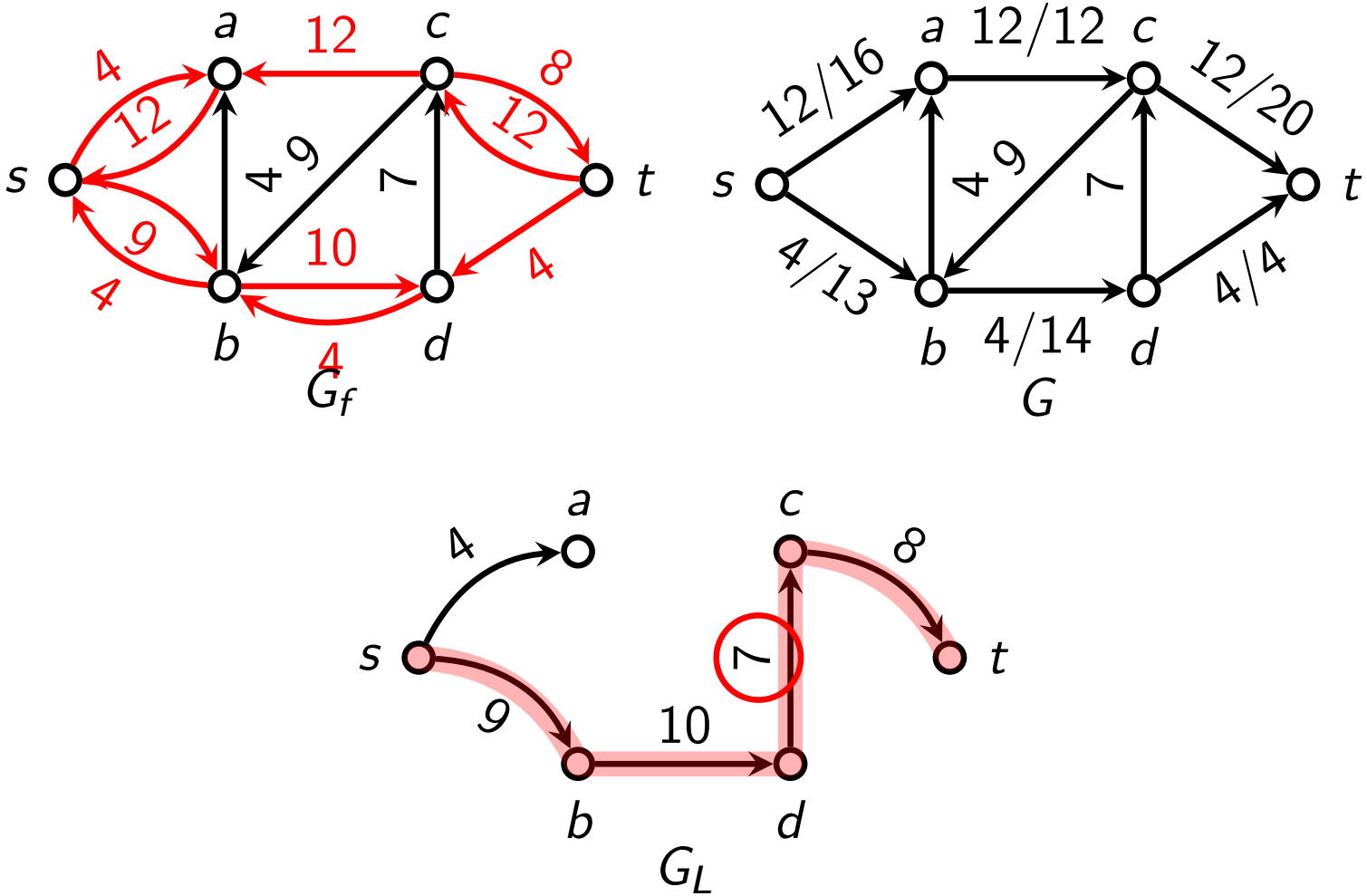
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



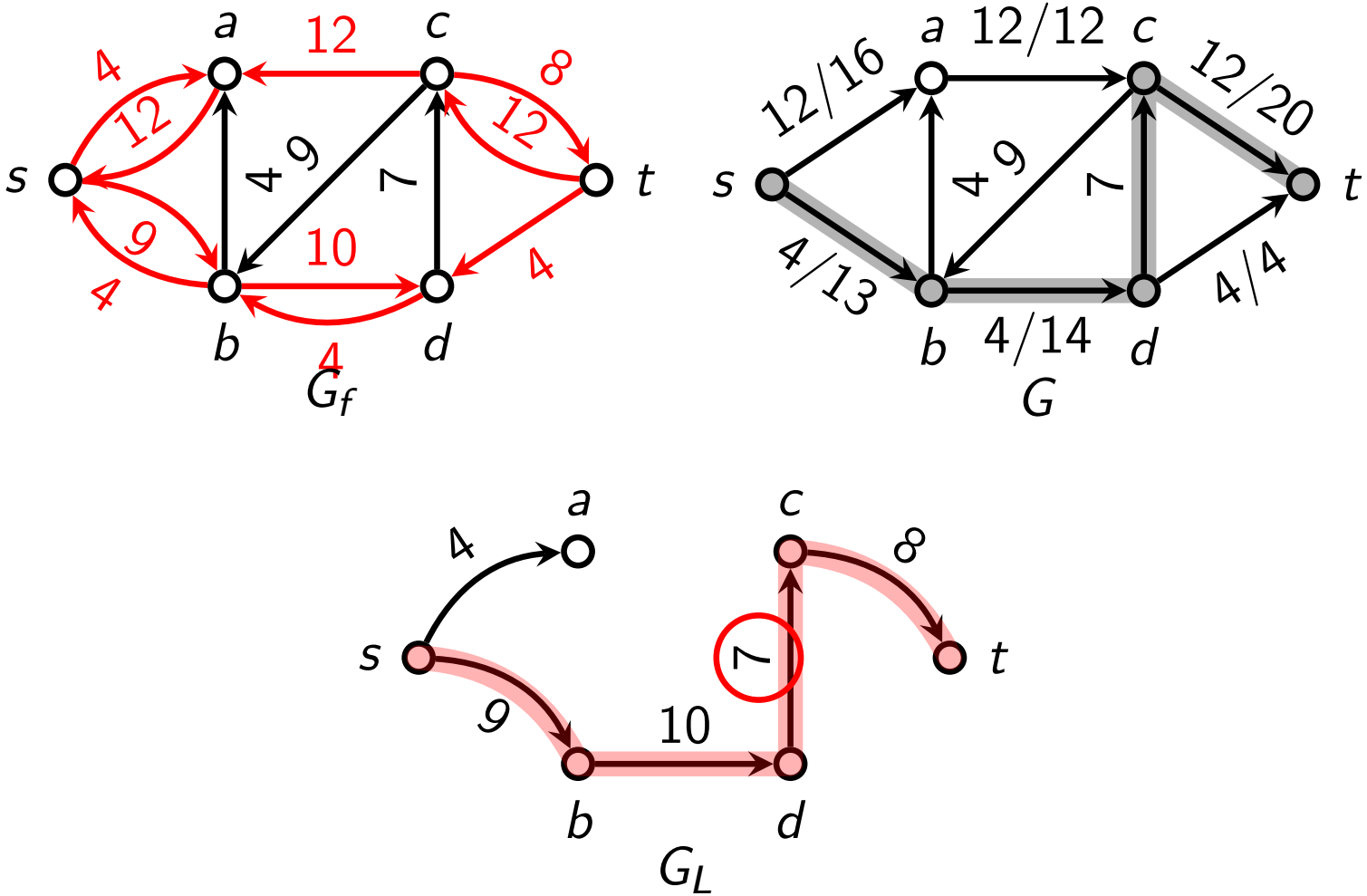
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



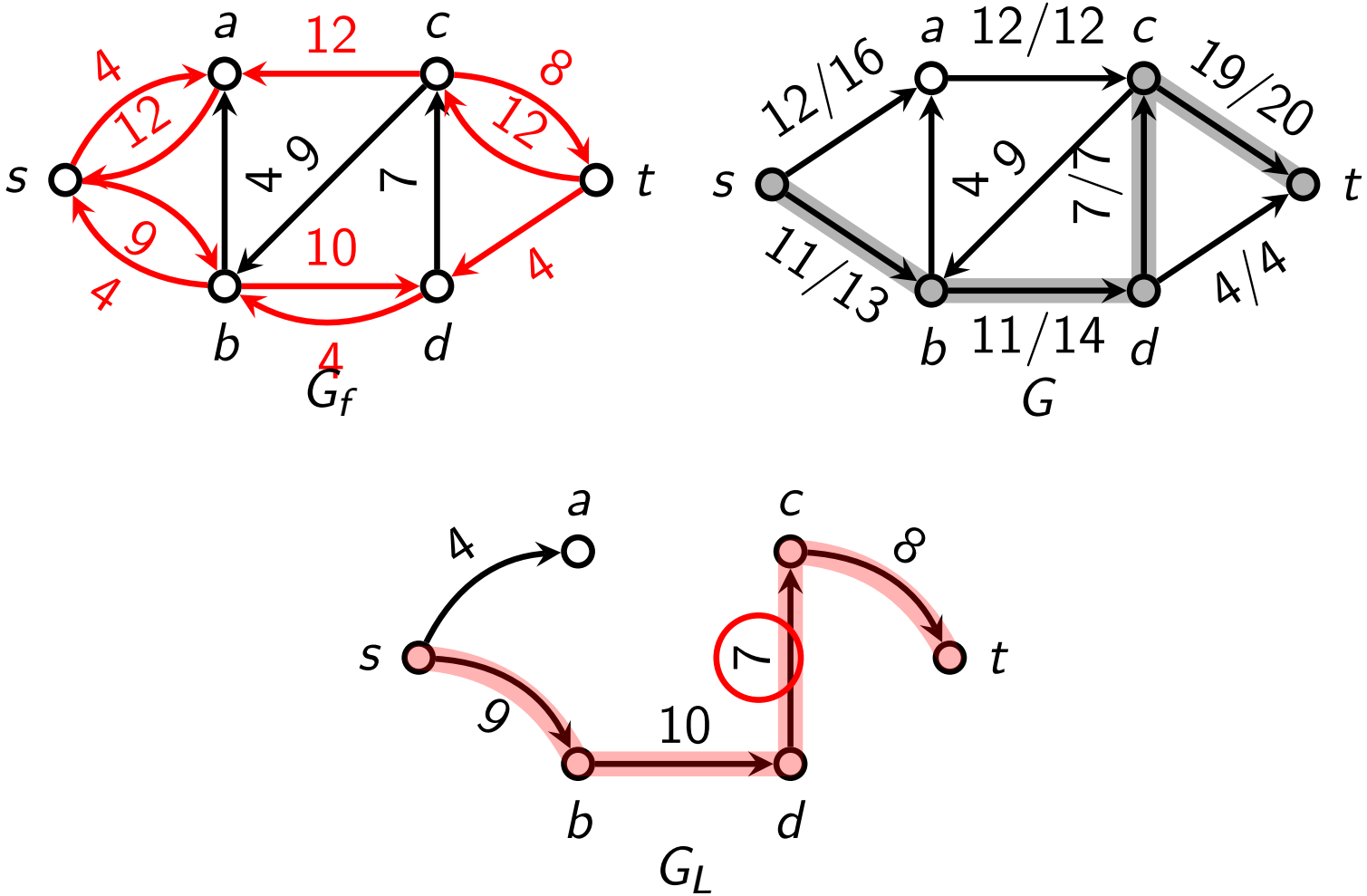
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



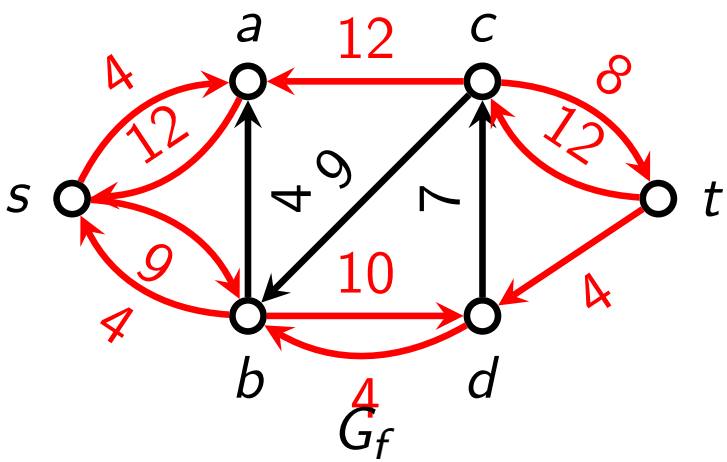
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.

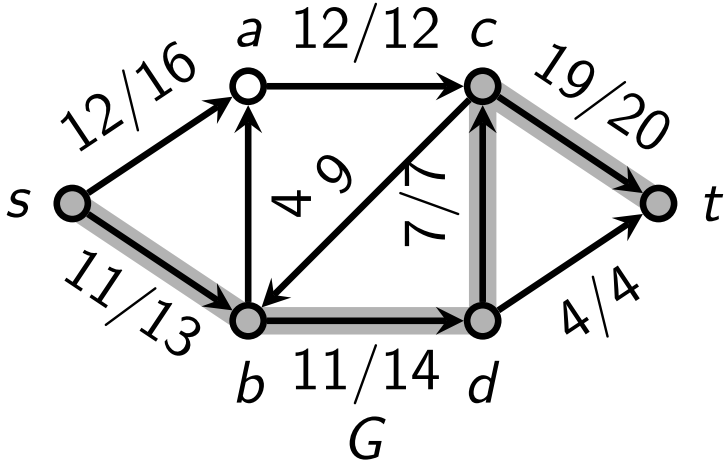


L'algorithme de Dinitz

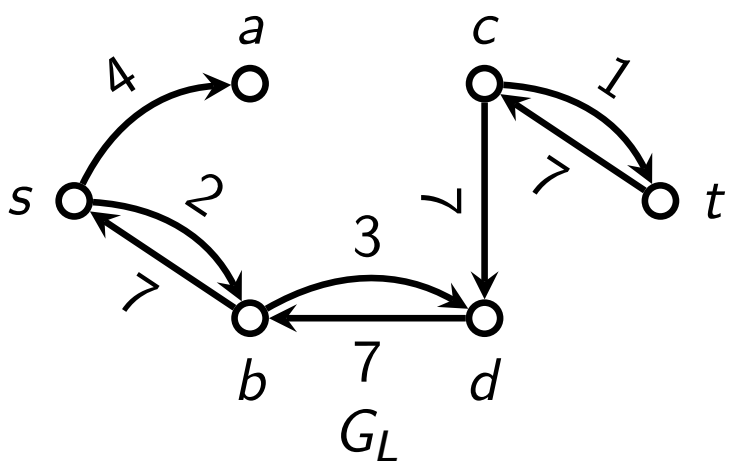
Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



G_f



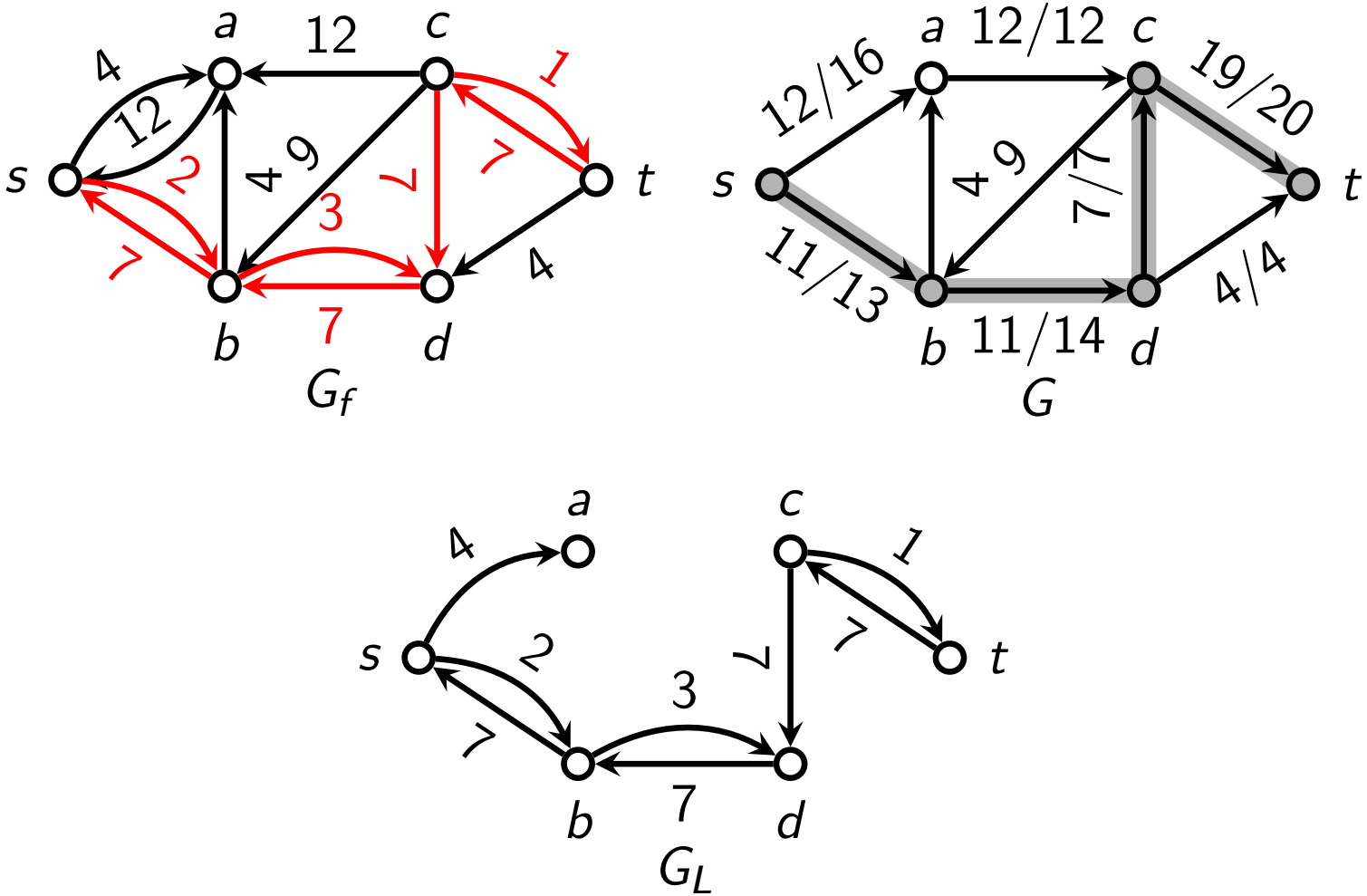
G



G_L

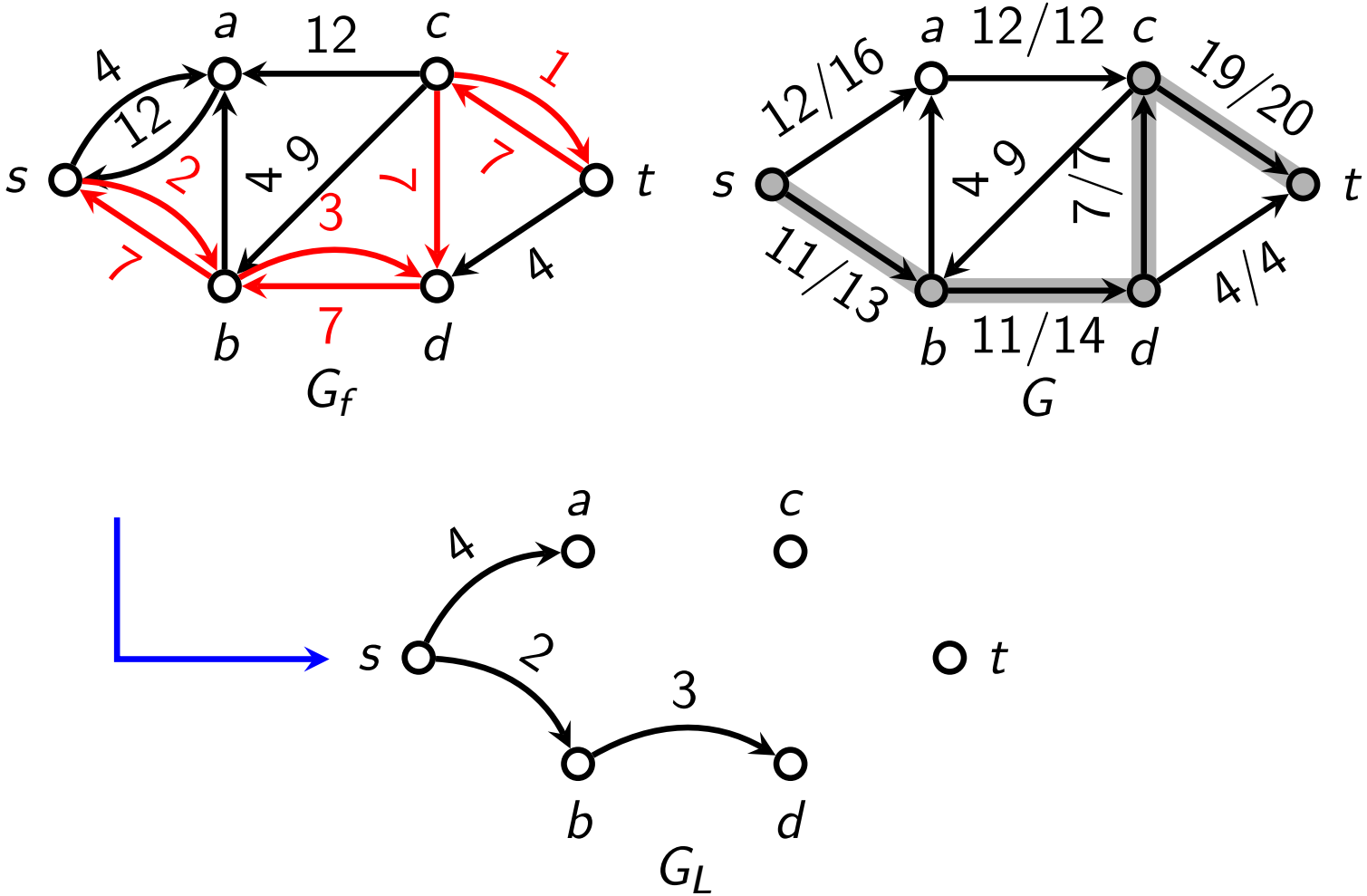
L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



L'algorithme de Dinitz

Examinons les étapes de l'algorithme de Dinitz sur le même exemple que pour Edmonds-Karp.



Pseudocode

On peut exprimer l'algorithme de Dinitz en pseudocode comme suit :

Algorithme 1 : DINITZ(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;
5  G_L ← DagLargeur(G, source, puits);
6  arcs ← FlotBloquant(G_L, f, source, puits);
7  tant que arcs ≠ ∅ faire
8  |   MettreAJourResiduel(G, G_f, arcs, flot);
9  |   G_L ← DagLargeur(G_f, source, puits);
10 |   arcs ← FlotBloquant(G_L, f, source, puits);
11 renvoyer flot;
```

Flots bloquants

Au lieu d'augmenter le flot sur un chemin dans le résiduel, on calcule un **flot bloquant** dans G_L :

Algorithme 2 : FlotBloquant(G_L, f, source, puits)

Entrées : un graphe orienté acyclique pondéré G_L, un flot f, une source et un puits.

Résultat : le flot f augmente au maximum le long de chaque chemin de G_L, qui est mis à jour au fur et à mesure ; renvoie l'ensemble des arcs qui ont subi un changement de flot.

```
1 arcs ← ∅;
2 chemin ← CheminAugmentant(G_L, source, puits);
3 tant que chemin ≠ NIL faire
4 |   AugmenterFlot(f, chemin);
5 |   MettreAJourDAGLargeur(G, G_L, chemin.arcs(), f);
6 |   arcs ← arcs ∪ chemin.arcs();
7 |   chemin ← CheminAugmentant(G_L, source, puits);
8 renvoyer arcs ;
```

Mise à jour de G_L

Et enfin, il faut mettre à jour tous les plus courts chemins

Algorithme 3 : METTREAJOURDAGLARGEUR(G, G_L, arcs, f)

Entrées : un réseau de flot G, un graphe orienté acyclique pondéré G_L, un ensemble d'arcs, et un flot f.
Résultat : le poids des arcs spécifiés de G_L est mis à jour

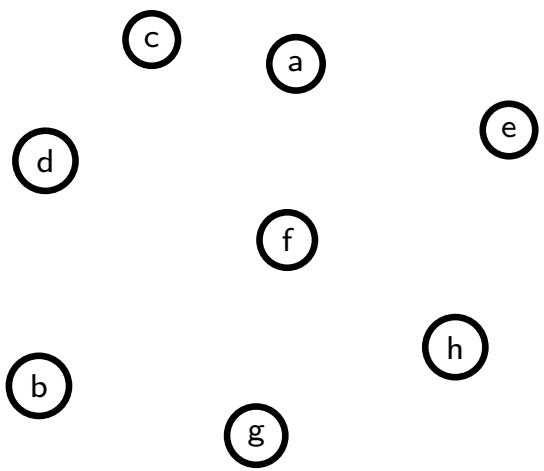
```
1  pour chaque (u, v, c) ∈ arcs faire
2      si G.contient_arc(u, v) alors
3          si G.poids_arc(u, v)−f(u, v) > 0 alors
4              G_L.ajouter_arc(u, v, G.poids_arc(u, v) −f(u, v))
5          sinon G_L.supprimer_arc(u, v) ;
6          si f(u, v) > 0 alors
7              G_L.ajouter_arc(v, u, f(u, v))
8          sinon  G_L.supprimer_arc(v, u) ;
9      sinon
10         si G.poids_arc(v, u) −f(v, u) > 0) alors
11             G_L.ajouter_arc(v, u, G.poids_arc(v, u) −f(v, u)))
12         sinon  G_L.supprimer_arc(v, u);
13         si f(v, u) > 0 alors
14             G_L.ajouter_arc(u, v, f(v, u))
15         sinon  G_L.supprimer_arc(u, v) ;
```

Conclusions sur Dinitz

- L'approche ressemble à celle de Ford-Fulkerson ;
- L'algorithme est plus efficace (on l'admettra sans preuve) ;
- L'implémentation présentée ici est plus pédagogique, mais on peut faire mieux ;

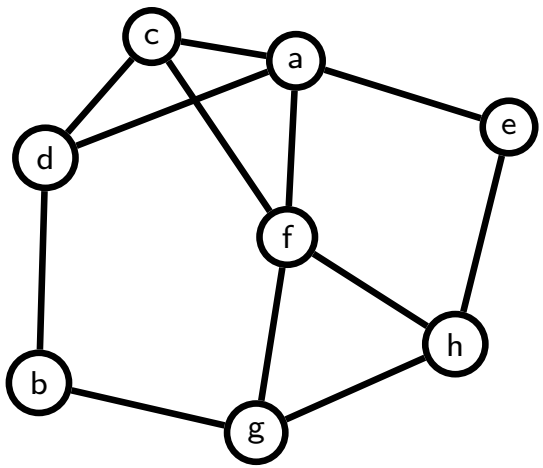
Motivations

- On doit grouper des étudiants en binômes pour un projet ;



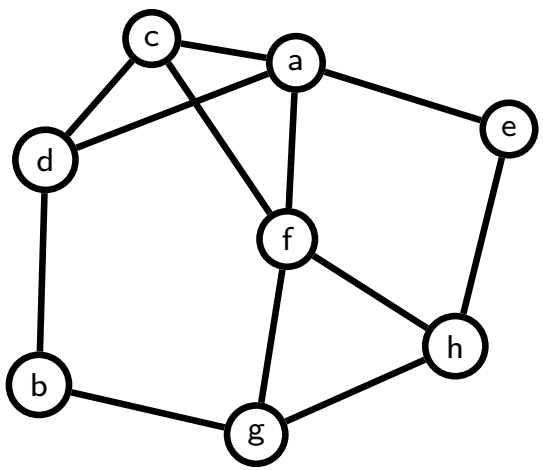
Motivations

- On doit grouper des étudiants en binômes pour un projet ;
- Les étudiants ne sont pas tous amis et communiquent donc leurs préférences, qu'on encode sous la forme d'un graphe ;



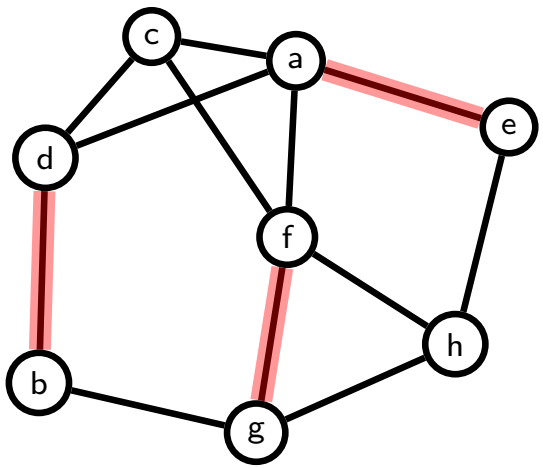
Motivations

- On doit grouper des étudiants en binômes pour un projet ;
- Les étudiants ne sont pas tous amis et communiquent donc leurs préférences, qu'on encode sous la forme d'un graphe ;
- On cherche ensuite à trouver un ensemble d'arêtes disjointes couvrant tous les étudiants :



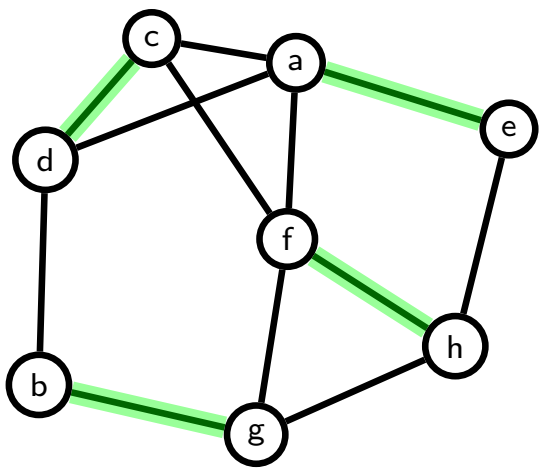
Motivations

- On doit grouper des étudiants en binômes pour un projet ;
- Les étudiants ne sont pas tous amis et communiquent donc leurs préférences, qu'on encode sous la forme d'un graphe ;
- On cherche ensuite à trouver un ensemble d'arêtes disjointes couvrant tous les étudiants :



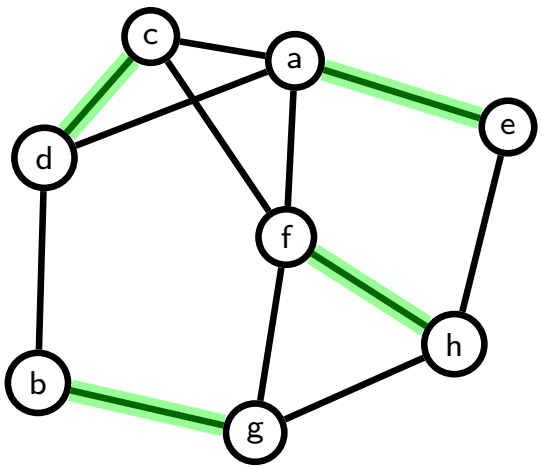
Motivations

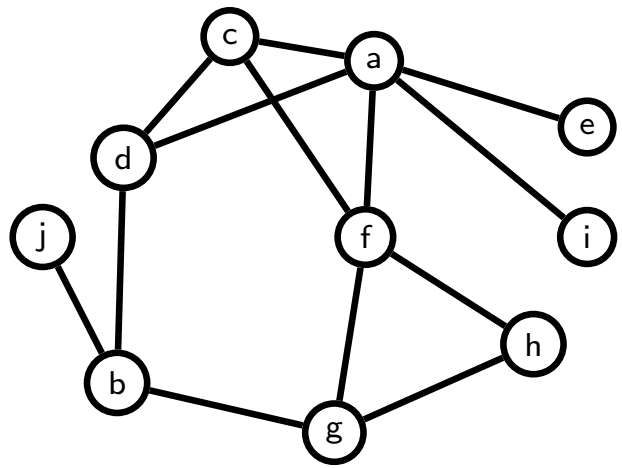
- On doit grouper des étudiants en binômes pour un projet ;
- Les étudiants ne sont pas tous amis et communiquent donc leurs préférences, qu'on encode sous la forme d'un graphe ;
- On cherche ensuite à trouver un ensemble d'arêtes disjointes couvrant tous les étudiants :



Motivations

- On doit grouper des étudiants en binômes pour un projet ;
- Les étudiants ne sont pas tous amis et communiquent donc leurs préférences, qu'on encode sous la forme d'un graphe ;
- On cherche ensuite à trouver un ensemble d'arêtes disjointes couvrant tous les étudiants :

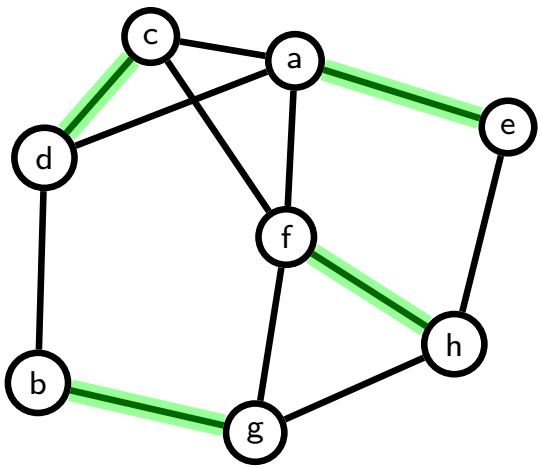


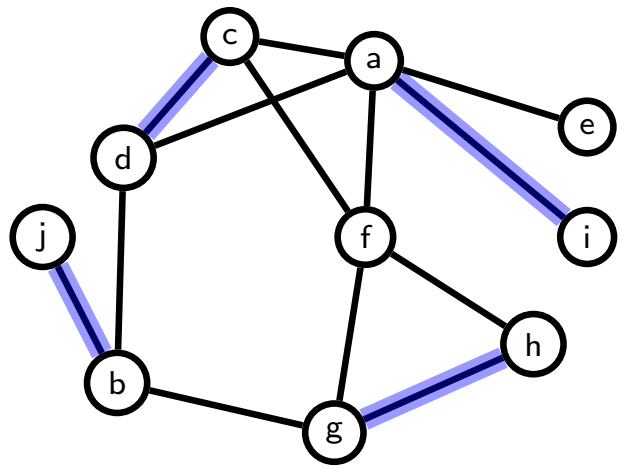


- *i* et *j* se sont réveillés un peu tard ; *e* et *h* se sont disputés ;

Motivations

- On doit grouper des étudiants en binômes pour un projet ;
- Les étudiants ne sont pas tous amis et communiquent donc leurs préférences, qu'on encode sous la forme d'un graphe ;
- On cherche ensuite à trouver un ensemble d'arêtes disjointes couvrant tous les étudiants :





- *i* et *j* se sont réveillés un peu tard ; *e* et *h* se sont disputés ;

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :
  - $V_1 =$ développeurs, $V_2 =$ tâches, $E =$ compétence de chaque développeur pour chaque tâche ;

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :
  - $V_1 =$ développeurs, $V_2 =$ tâches, $E =$ compétence de chaque développeur pour chaque tâche ;
  - $V_1 =$ entrepreneurs, $V_2 =$ travaux, $E =$ capacité de chaque entrepreneur pour chaque poste ;

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :
  - V_1 = développeurs, V_2 = tâches, E = compétence de chaque développeur pour chaque tâche ;
  - V_1 = entrepreneurs, V_2 = travaux, E = capacité de chaque entrepreneur pour chaque poste ;

  ⋮

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :
  - V_1 = développeurs, V_2 = tâches, E = compétence de chaque développeur pour chaque tâche ;
  - V_1 = entrepreneurs, V_2 = travaux, E = capacité de chaque entrepreneur pour chaque poste ;

  ⋮
- Dans ce cas-là, on peut alors résoudre notre problème à l'aide d'un calcul de flot maximum ;

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :
  - V_1 = développeurs, V_2 = tâches, E = compétence de chaque développeur pour chaque tâche ;
  - V_1 = entrepreneurs, V_2 = travaux, E = capacité de chaque entrepreneur pour chaque poste ;

  ⋮
- Dans ce cas-là, on peut alors résoudre notre problème à l'aide d'un calcul de flot maximum ;
- Dans le cas où le graphe n'est pas biparti, le problème reste soluble en temps polynomial mais l'algorithme est beaucoup plus complexe [**?**] ;

Motivations

- Dans certaines situations plus simples, le graphe $G = (V_1 \cup V_2, E)$ des préférences est biparti ; par exemple :
  - V_1 = développeurs, V_2 = tâches, E = compétence de chaque développeur pour chaque tâche ;
  - V_1 = entrepreneurs, V_2 = travaux, E = capacité de chaque entrepreneur pour chaque poste ;

  ⋮
- Dans ce cas-là, on peut alors résoudre notre problème à l'aide d'un calcul de flot maximum ;
- Dans le cas où le graphe n'est pas biparti, le problème reste soluble en temps polynomial mais l'algorithme est beaucoup plus complexe [**?**] ;
- Et si l'on veut des trinômes ? Le problème est NP-difficile [**?**] ;

Couplage

Un **couplage** dans un graphe $G = (V, E)$ est un ensemble $F \subseteq E$ d'arêtes deux à deux disjointes ; il est :

1. **maximal** si on ne peut pas lui rajouter d'arêtes,

Couplage

Un **couplage** dans un graphe $G = (V, E)$ est un ensemble $F \subseteq E$ d'arêtes deux à deux disjointes ; il est :

1. **maximal** si on ne peut pas lui rajouter d'arêtes,
2. **maximum** s'il n'existe aucun autre couplage possédant plus d'arêtes, et

Couplage

Un **couplage** dans un graphe $G = (V, E)$ est un ensemble $F \subseteq E$ d'arêtes deux à deux disjointes ; il est :

1. **maximal** si on ne peut pas lui rajouter d'arêtes,
2. **maximum** s'il n'existe aucun autre couplage possédant plus d'arêtes, et
3. **parfait** s'il couvre tous les sommets du graphe.

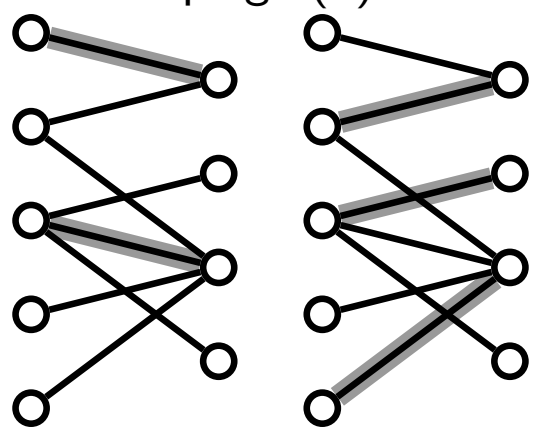
Couplage

Un **couplage** dans un graphe $G = (V, E)$ est un ensemble $F \subseteq E$ d'arêtes deux à deux disjointes ; il est :

1. **maximal** si on ne peut pas lui rajouter d'arêtes,
2. **maximum** s'il n'existe aucun autre couplage possédant plus d'arêtes, et
3. **parfait** s'il couvre tous les sommets du graphe.

Exemple 1

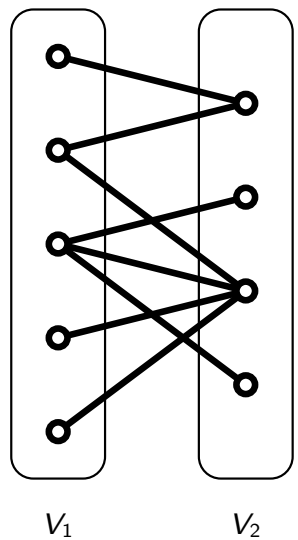
Voici un graphe biparti avec un couplage (a) maximal et (b) maximum.



(a) (b)

Lien entre couplages et flots

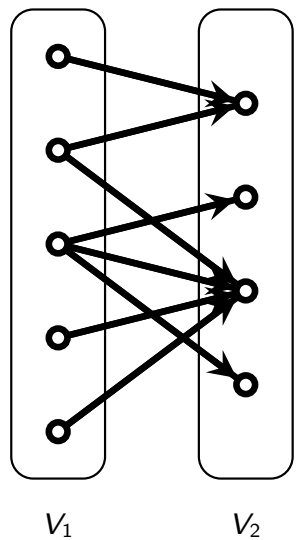
Si $G = (V_1 \cup V_2, E)$ **est biparti**, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :



Lien entre couplages et flots

Si $G = (V_1 \cup V_2, E)$ est biparti, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :

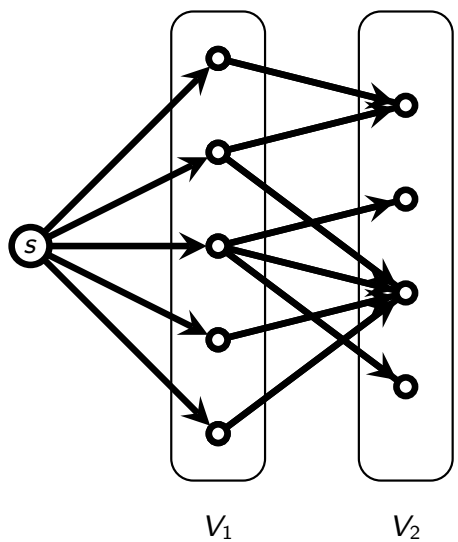
1. orienter toutes les arêtes de G de V_1 vers V_2 ;



Lien entre couplages et flots

Si $G = (V_1 \cup V_2, E)$ est biparti, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :

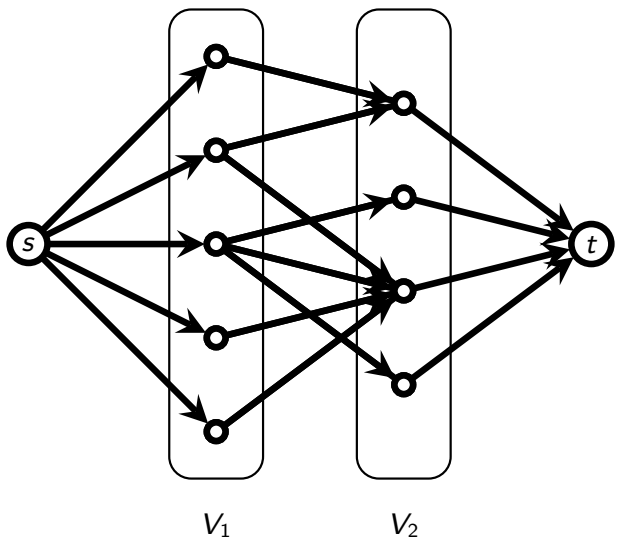
1. orienter toutes les arêtes de G de V_1 vers V_2 ;
2. rajouter une source s avec comme successeurs tout V_1 ;



Lien entre couplages et flots

Si $G = (V_1 \cup V_2, E)$ est biparti, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :

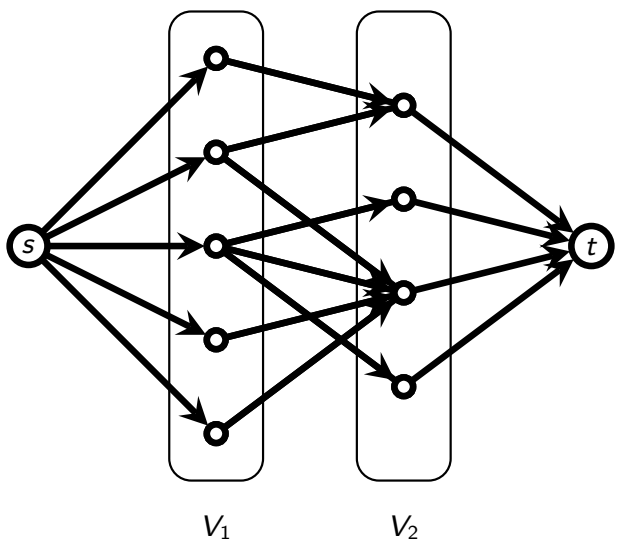
1. orienter toutes les arêtes de G de V_1 vers V_2 ;
2. rajouter une source s avec comme successeurs tout V_1 ;
3. rajouter un puits t avec comme prédécesseurs tout V_2 ;



Lien entre couplages et flots

Si $G = (V_1 \cup V_2, E)$ est biparti, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :

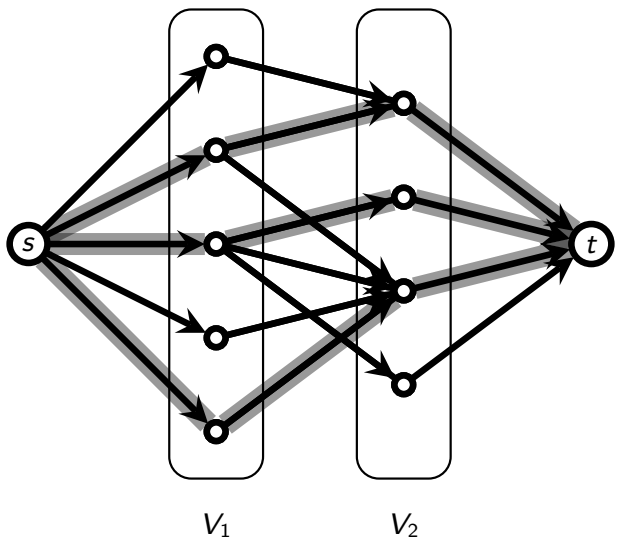
1. orienter toutes les arêtes de G de V_1 vers V_2 ;
2. rajouter une source s avec comme successeurs tout V_1 ;
3. rajouter un puits t avec comme prédécesseurs tout V_2 ;
4. affecter à tous les arcs une capacité de 1 ;



Lien entre couplages et flots

Si $G = (V_1 \cup V_2, E)$ est biparti, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :

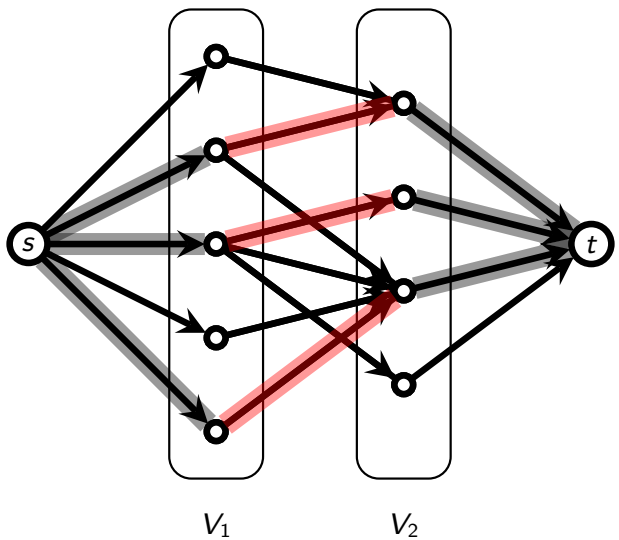
1. orienter toutes les arêtes de G de V_1 vers V_2 ;
2. rajouter une source s avec comme successeurs tout V_1 ;
3. rajouter un puits t avec comme prédécesseurs tout V_2 ;
4. affecter à tous les arcs une capacité de 1 ;
5. on note G' ce réseau résultant.



Lien entre couplages et flots

Si $G = (V_1 \cup V_2, E)$ est biparti, on peut trouver un couplage maximum à l'aide d'un flot maximum comme suit :

1. orienter toutes les arêtes de G de V_1 vers V_2 ;
2. rajouter une source s avec comme successeurs tout V_1 ;
3. rajouter un puits t avec comme prédécesseurs tout V_2 ;
4. affecter à tous les arcs une capacité de 1 ;
5. on note G' ce réseau résultant.
6. trouver un flot maximum dans le réseau résultant.



Les arêtes du couplage sont les arêtes de G saturées par le flot.

Correction

Proposition 2

Soit S les arcs auxquels un flot de 1 est affecté après le calcul d'un flot maximum f dans G' ; alors l'ensemble M des arcs de S dont les extrémités sont toutes deux dans G est un couplage maximum pour G.

Correction

Proposition 2

Soit S les arcs auxquels un flot de 1 est affecté après le calcul d'un flot maximum f dans G' ; alors l'ensemble M des arcs de S dont les extrémités sont toutes deux dans G est un couplage maximum pour G.

Démonstration.

On a deux propriétés à prouver :

Correction

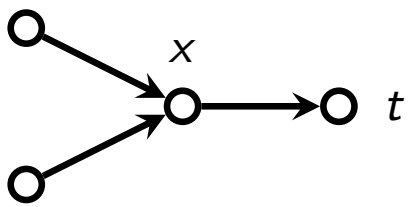
Proposition 2

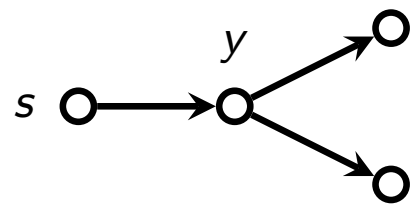
Soit S les arcs auxquels un flot de 1 est affecté après le calcul d'un flot maximum f dans G' ; alors l'ensemble M des arcs de S dont les extrémités sont toutes deux dans G est un couplage maximum pour G.

Démonstration.

On a deux propriétés à prouver :

1. *M* **est un couplage :** la conservation des flots empêche une entrée > 1 et une sortie > 1 :





Correction

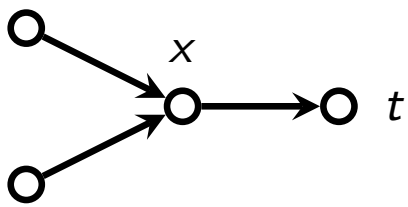
Proposition 2

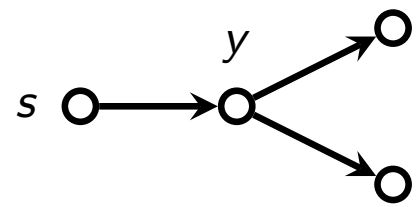
Soit S les arcs auxquels un flot de 1 est affecté après le calcul d'un flot maximum f dans G' ; alors l'ensemble M des arcs de S dont les extrémités sont toutes deux dans G est un couplage maximum pour G.

Démonstration.

On a deux propriétés à prouver :

1. M **est un couplage :** la conservation des flots empêche une entrée > 1 et une sortie > 1 :





2. M **est maximum :** par contradiction, s'il existe un couplage M' avec $|M'| > |M|$, alors les arêtes de M' correspondent à des arcs dans G' que l'on peut connecter à s et à t pour obtenir un flot f' de valeur supérieure à f, ce qui contredit l'hypothèse "f est maximum".

□

Complexité

Algorithme 4 : CouplageBiparti(G)

Entrées : un graphe biparti connexe $G = (V_1 \cup V_2, E)$.
Sortie : un couplage maximum pour G.

```
1 H ← GrapheOrientéPondéré();
2 H.ajouter_sommet(s);
3 H.ajouter_sommet(t);
4 pour chaque {u, v} ∈ G.aretes() avec u ∈ V1 et v ∈ V2 faire
5 |   H.ajouter_arc(s, u, 1) ; H.ajouter_arc(u, v, 1) ; H.ajouter_arc(v, t,
  |     1) ;
6 flot ← Dinitz(H);
7 renvoyer {{u, v} | {u, v} ∈ E et flot(u, v) = 1} ;
```

- Construction du réseau : $O(|V| + |E|)$

Complexité

Algorithme 4 : CouplageBiparti(G)

Entrées : un graphe biparti connexe $G = (V_1 \cup V_2, E)$.
Sortie : un couplage maximum pour G.

```
1 H ← GrapheOrientéPondéré();
2 H.ajouter_sommet(s);
3 H.ajouter_sommet(t);
4 pour chaque {u, v} ∈ G.aretes() avec u ∈ V1 et v ∈ V2 faire
5 |   H.ajouter_arc(s, u, 1) ; H.ajouter_arc(u, v, 1) ; H.ajouter_arc(v, t,
  |     1) ;
6 flot ← Dinitz(H);
7 renvoyer {{u, v} | {u, v} ∈ E et flot(u, v) = 1} ;
```

- Construction du réseau : $O(|V| + |E|)$
- La valeur d'un flot maximum f est $O(|V|)$ (couplage parfait)

Complexité

Algorithme 4 : CouplageBiparti(G)

Entrées : un graphe biparti connexe $G = (V_1 \cup V_2, E)$.
Sortie : un couplage maximum pour G.

```
1 H ← GrapheOrientéPondéré();
2 H.ajouter_sommet(s);
3 H.ajouter_sommet(t);
4 pour chaque {u, v} ∈ G.aretes() avec u ∈ V1 et v ∈ V2 faire
5 |   H.ajouter_arc(s, u, 1) ; H.ajouter_arc(u, v, 1) ; H.ajouter_arc(v, t,
  |     1) ;
6 flot ← Dinitz(H);
7 renvoyer {{u, v} | {u, v} ∈ E et flot(u, v) = 1} ;
```

- Construction du réseau : $O(|V| + |E|)$
- La valeur d'un flot maximum f est $O(|V|)$ (couplage parfait)
- Chaque chemin augmente le flot d'au moins 1 $\Rightarrow$ au plus $O(|V|)$ calculs de chemin

Complexité

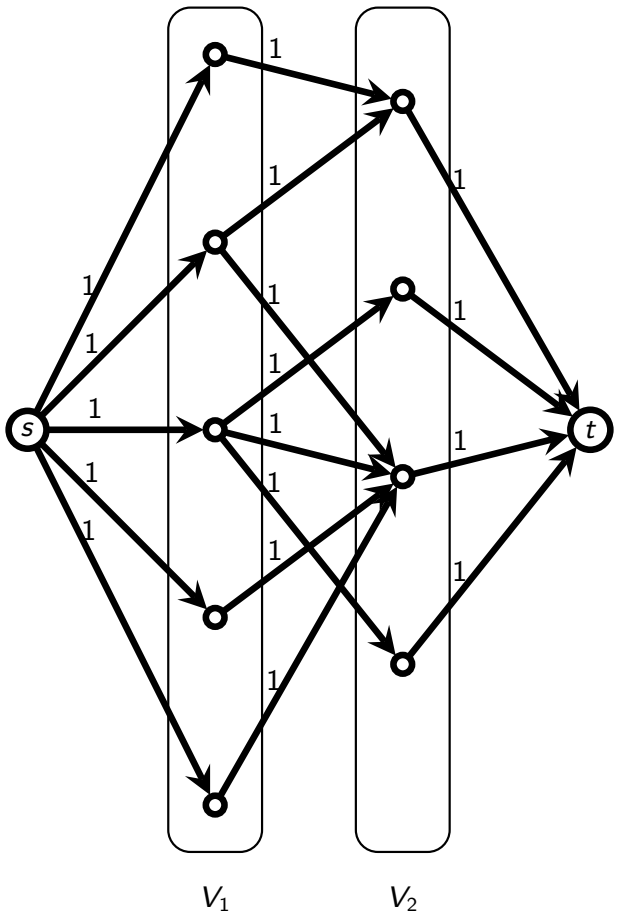
Algorithme 4 : CouplageBiparti(G)

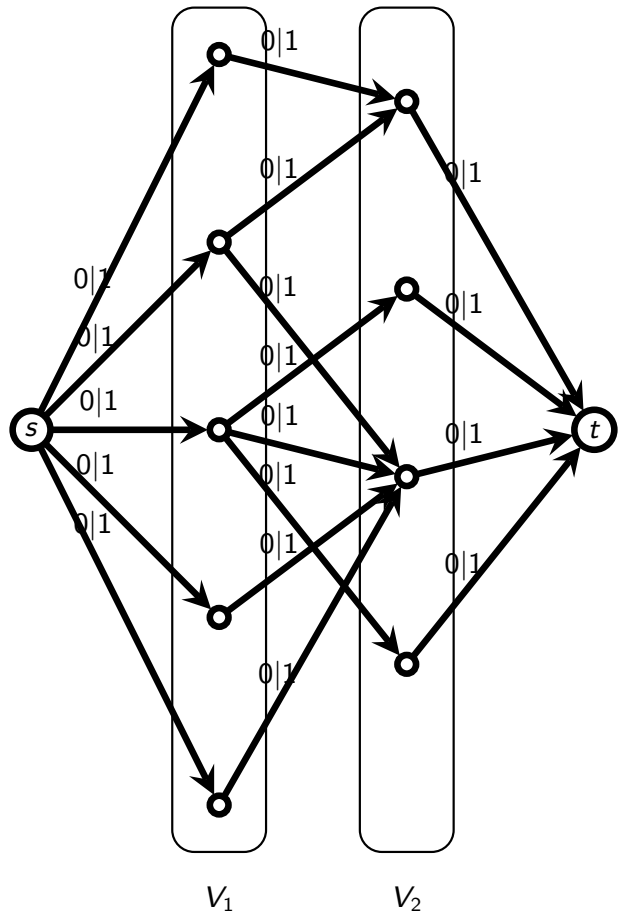
Entrées : un graphe biparti connexe $G = (V_1 \cup V_2, E)$.
Sortie : un couplage maximum pour G.

```
1 H ← GrapheOrientéPondéré();
2 H.ajouter_sommet(s);
3 H.ajouter_sommet(t);
4 pour chaque {u, v} ∈ G.aretes() avec u ∈ V1 et v ∈ V2 faire
5 |   H.ajouter_arc(s, u, 1) ; H.ajouter_arc(u, v, 1) ; H.ajouter_arc(v, t,
  |     1) ;
6 flot ← Dinitz(H);
7 renvoyer {{u, v} | {u, v} ∈ E et flot(u, v) = 1} ;
```

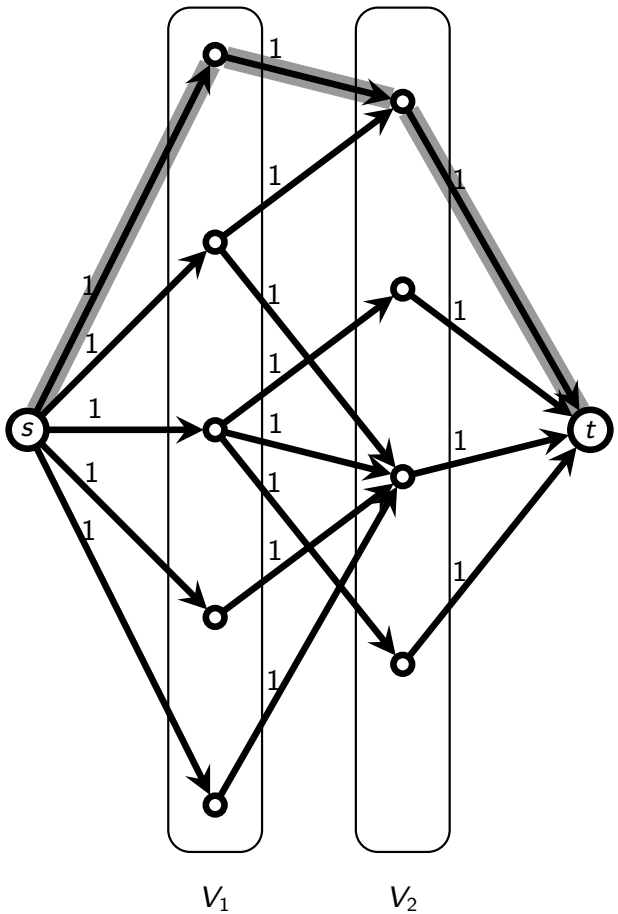
- Construction du réseau : $O(|V| + |E|)$
- La valeur d'un flot maximum f est $O(|V|)$ (couplage parfait)
- Chaque chemin augmente le flot d'au moins 1 $\Rightarrow$ au plus $O(|V|)$ calculs de chemin
- On peut donc obtenir du $O(|V||E|)$; Dinitz donne du $O(\sqrt{|V|}|E|)$;

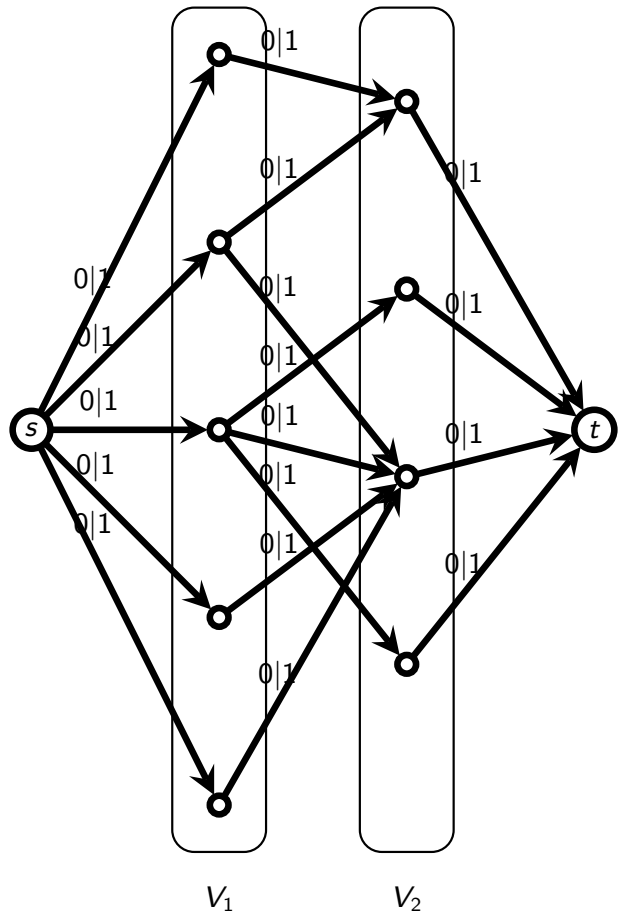
Exemple



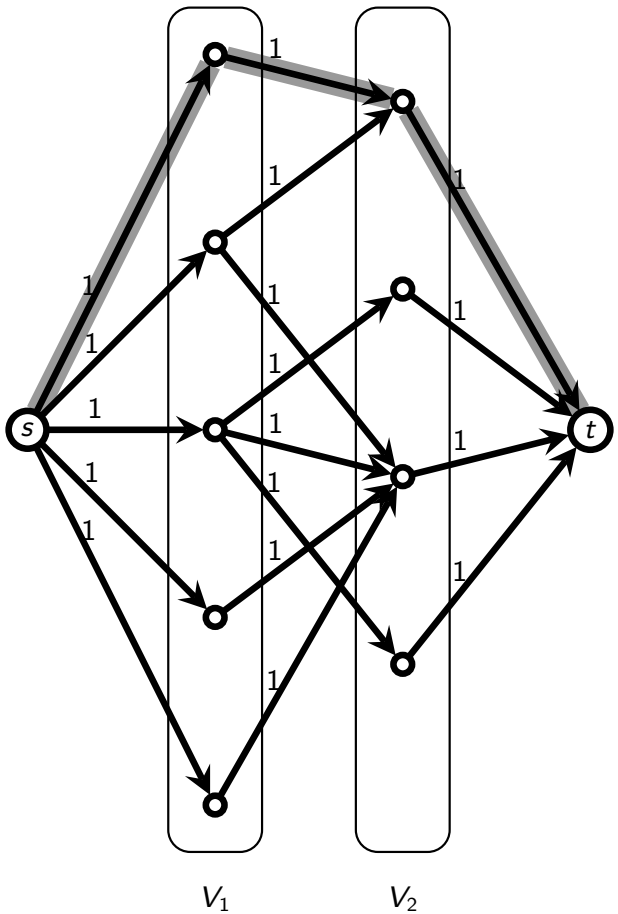


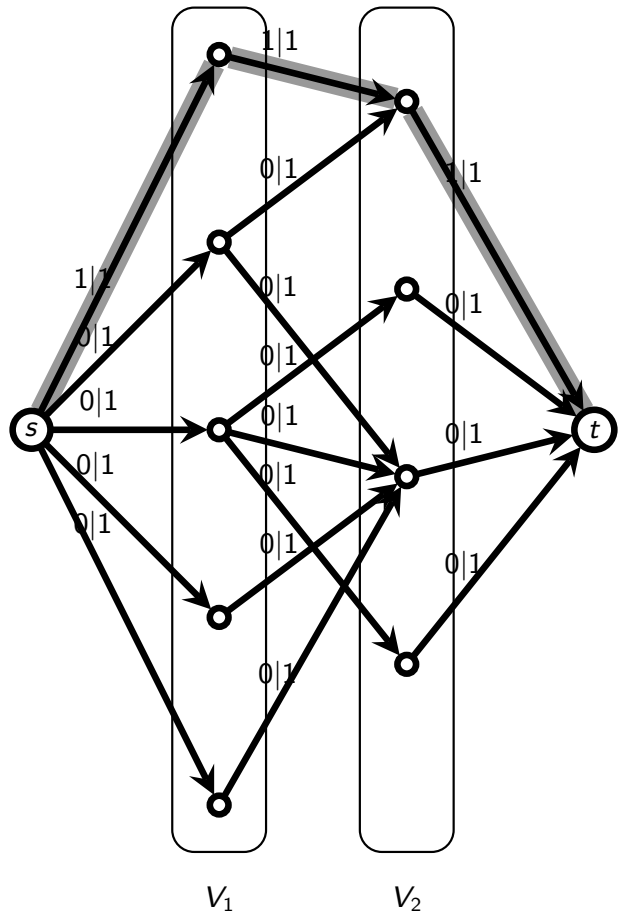
Exemple



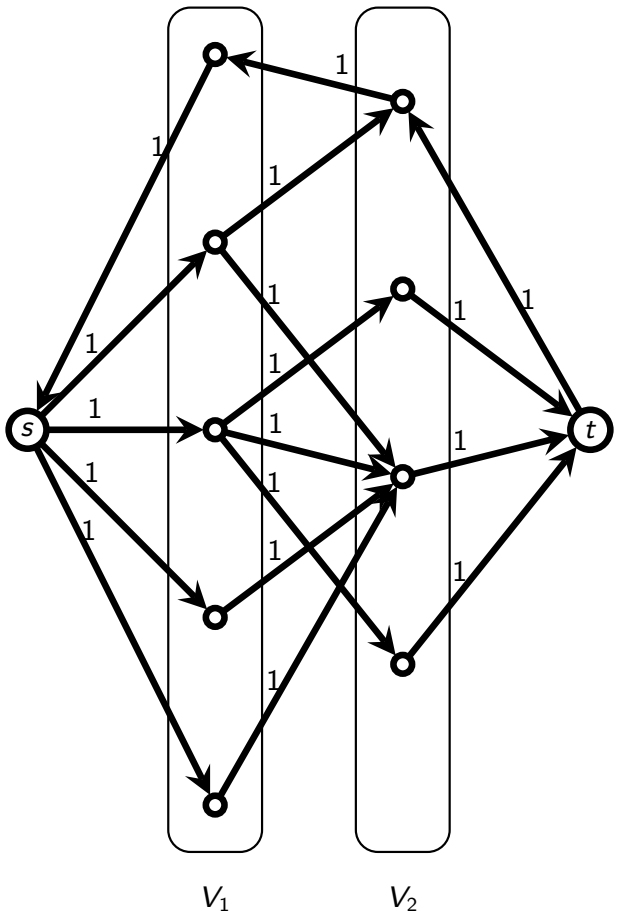


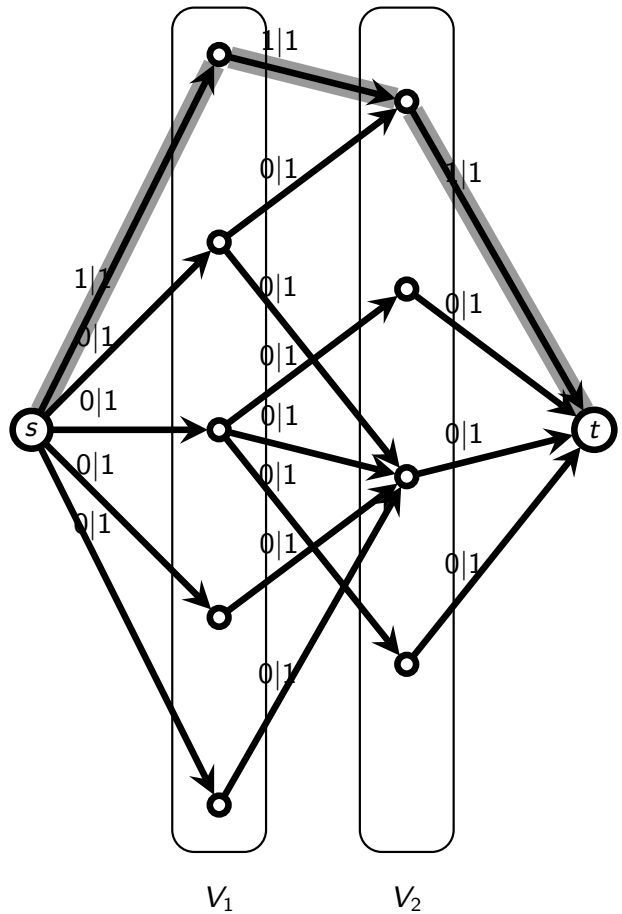
Exemple



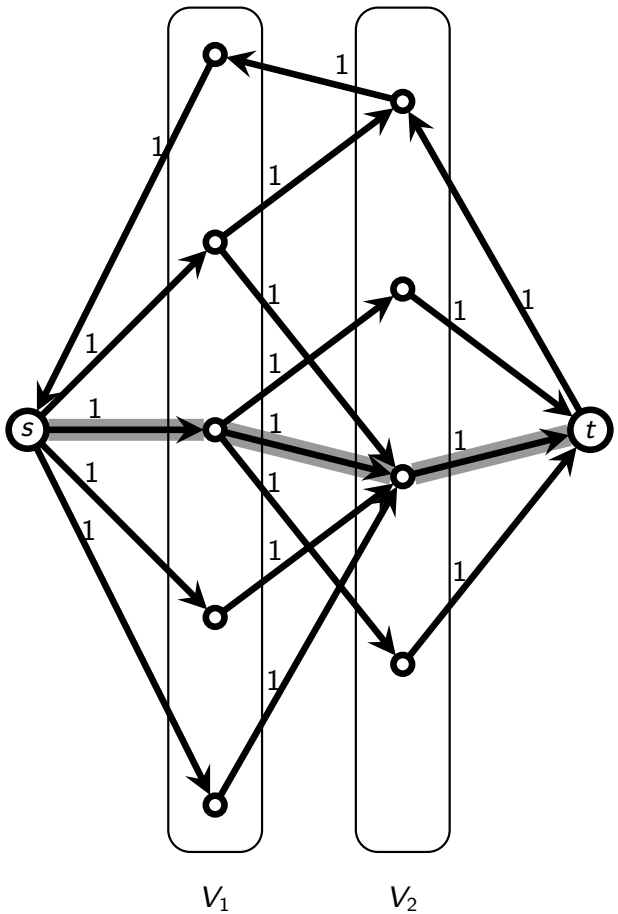


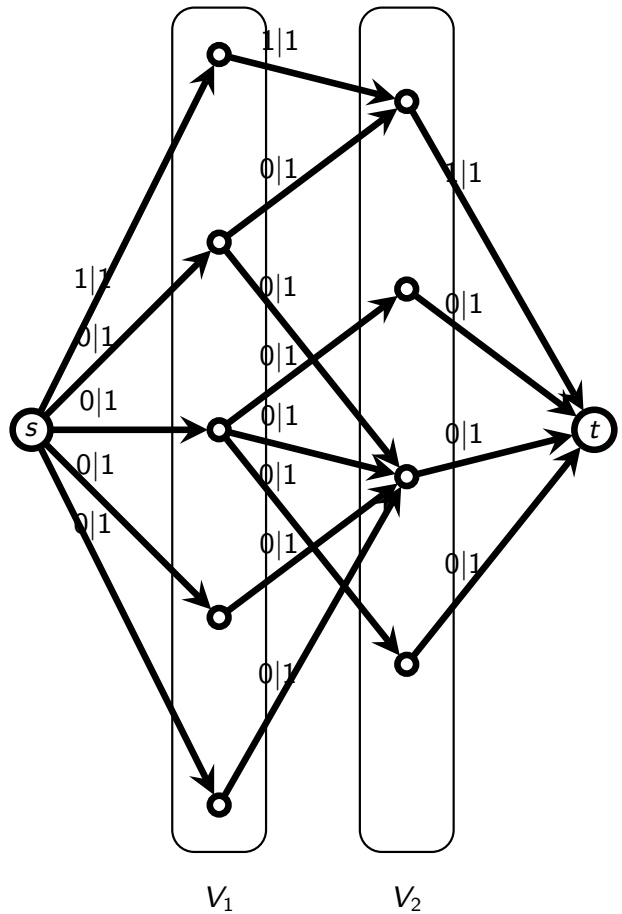
Exemple



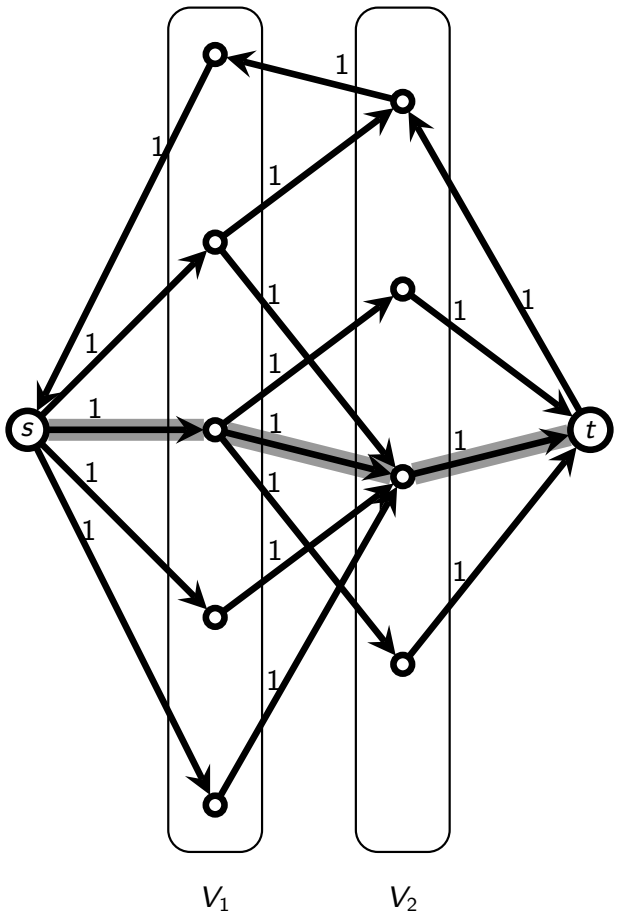


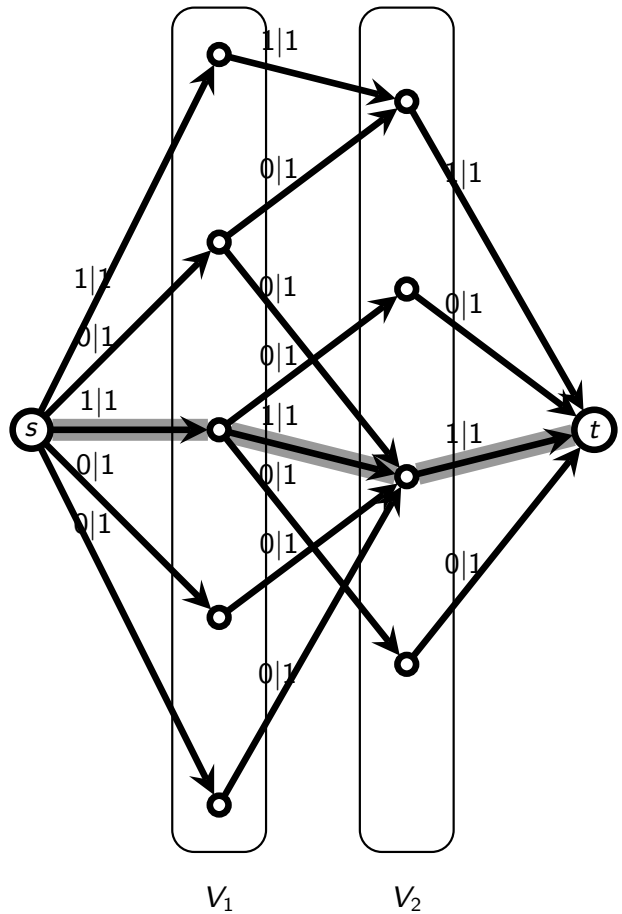
Exemple



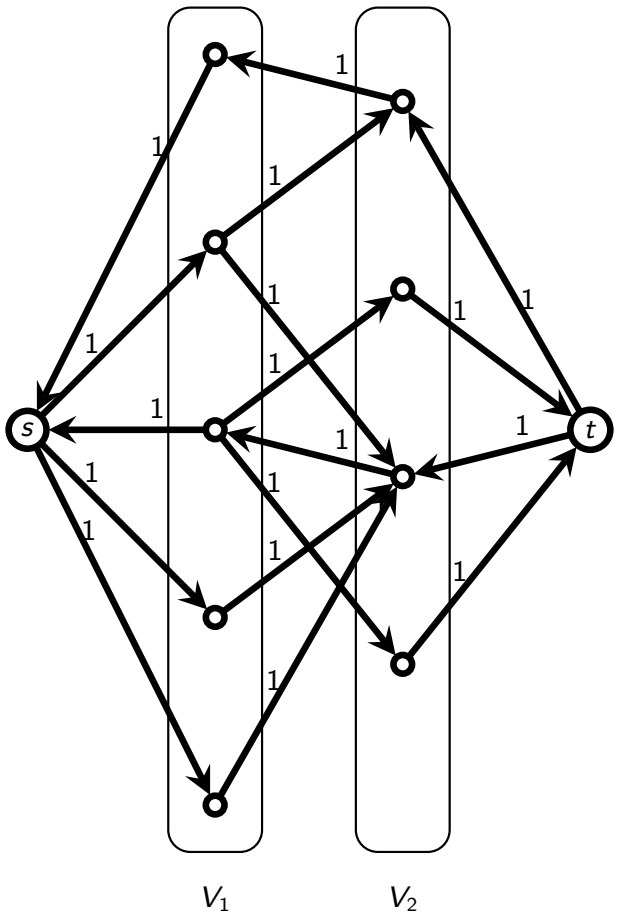


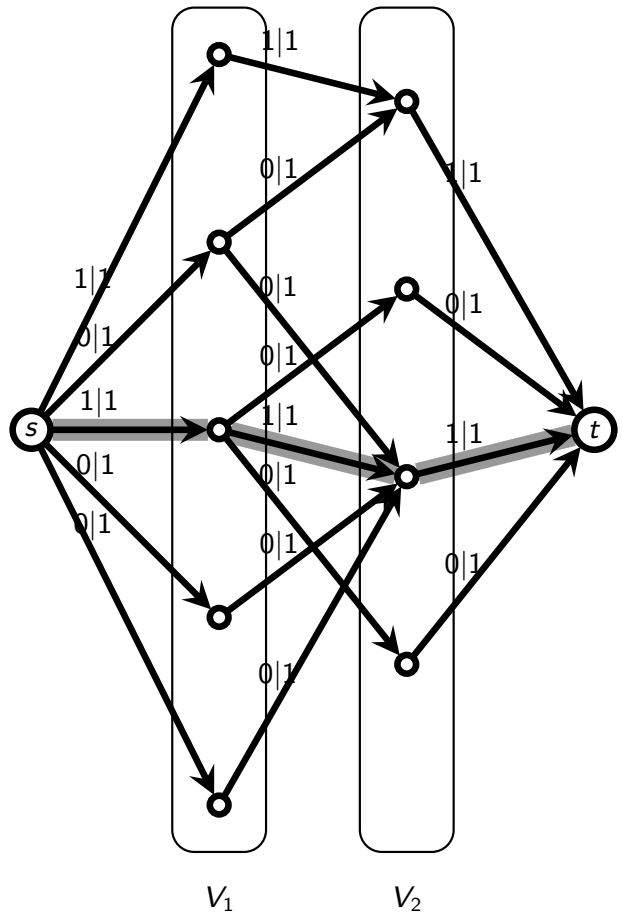
Exemple



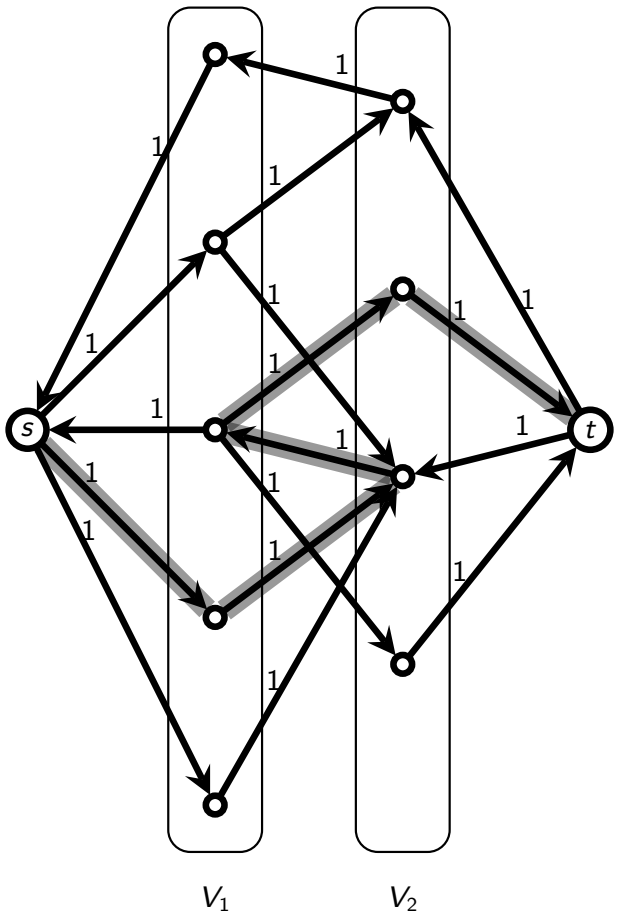


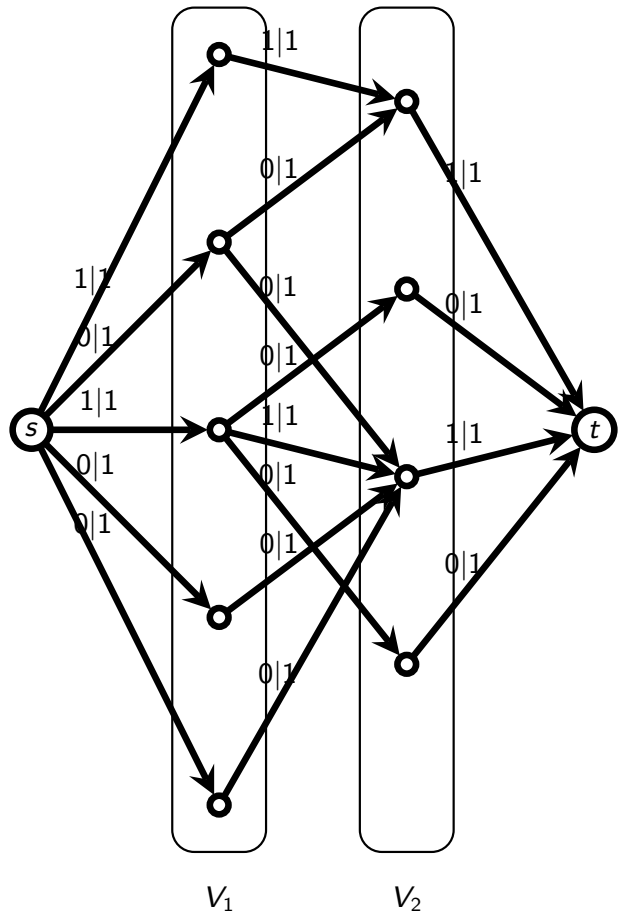
Exemple



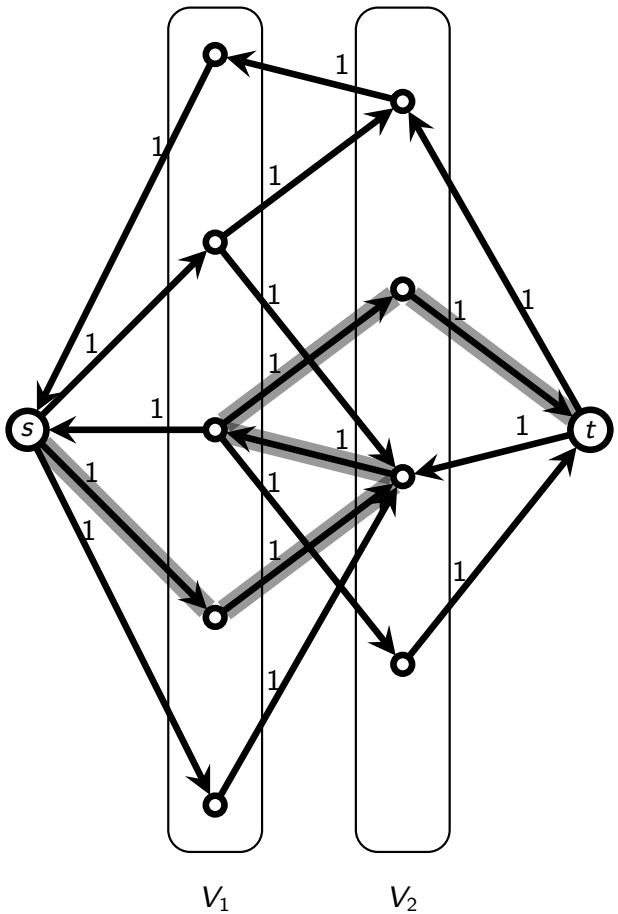


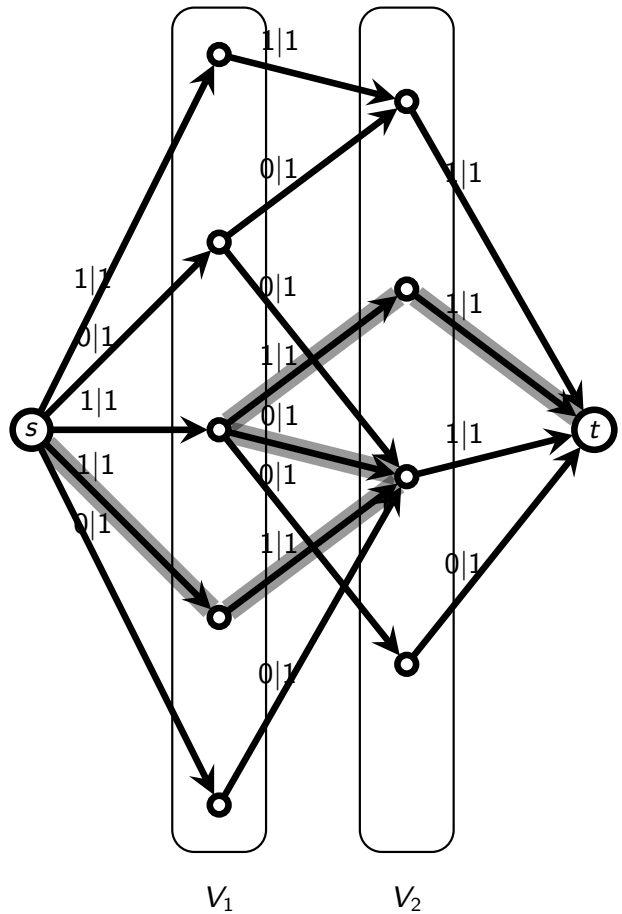
Exemple



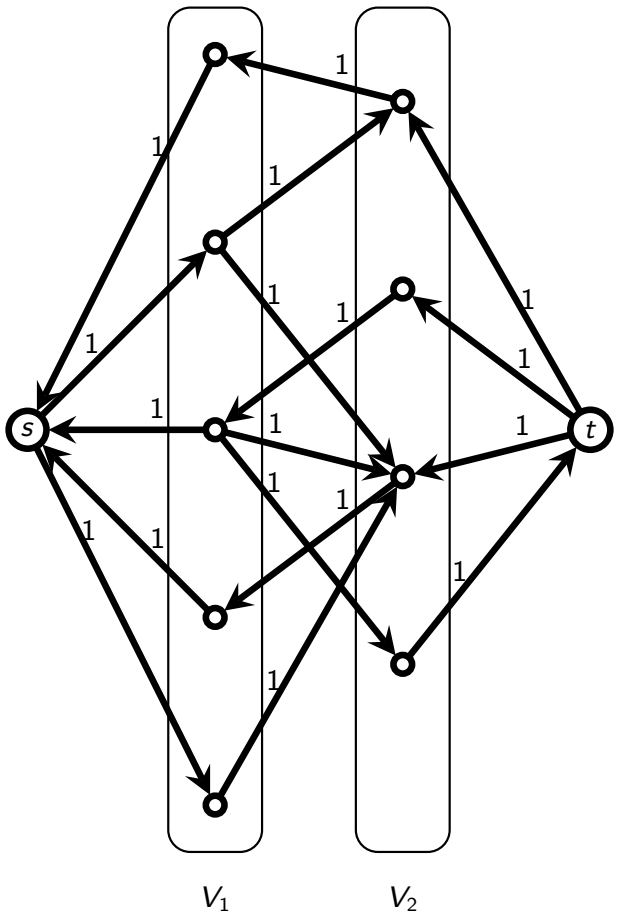


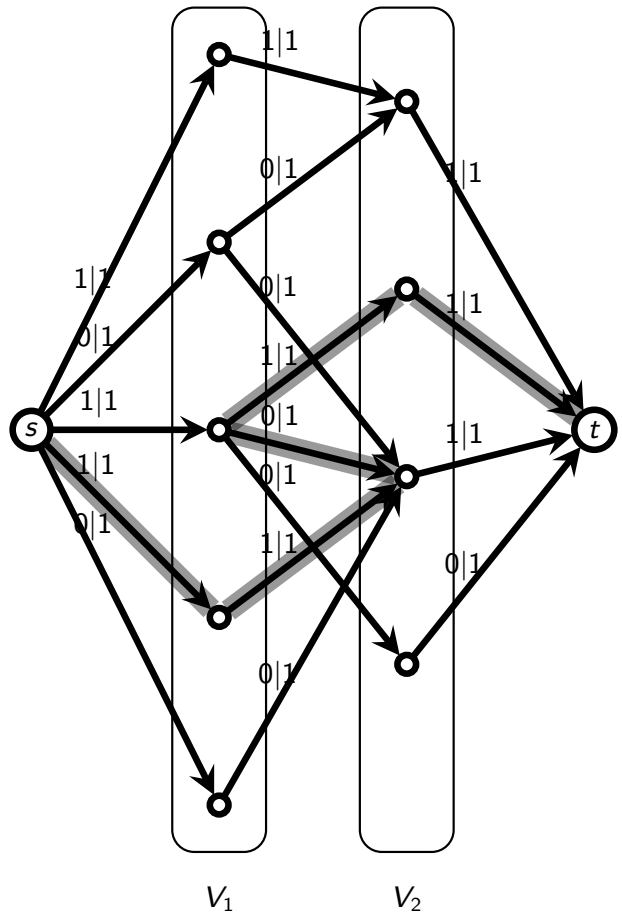
Exemple



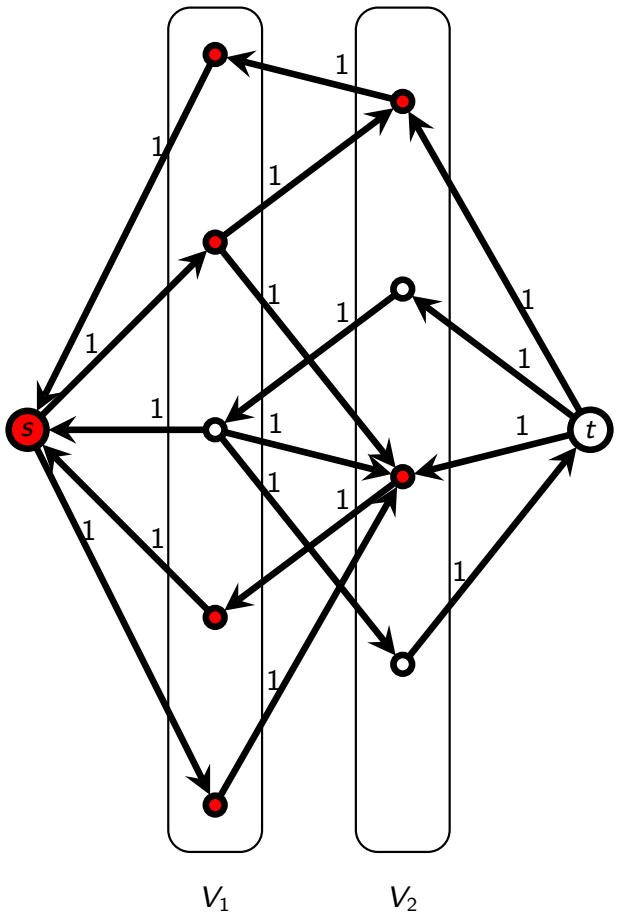


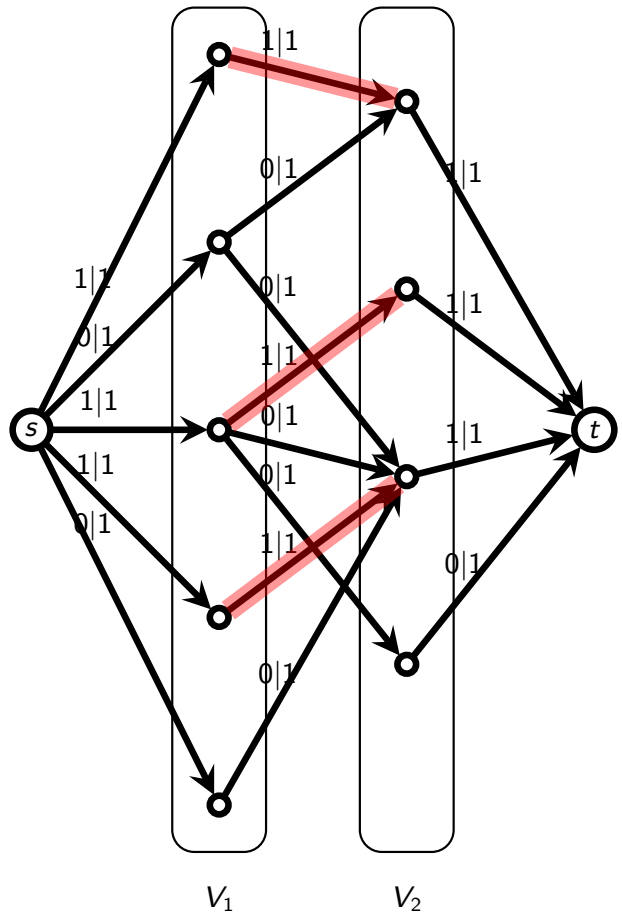
Exemple





Exemple





Bibliographie

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.
Introduction to Algorithms.
MIT Press, 3ème edition, 2009.

[2] Jack Edmonds.
Paths, trees, and flowers.
Canadian Journal of Mathematics, 17 :449–467, 1965.

[3] M. R. Garey and David S. Johnson.
Computers and Intractability : A Guide to the Theory of NP-Completeness.
W. H. Freeman, 1979.

[4] Daniel Dominic Sleator and Robert Endre Tarjan.
A data structure for dynamic trees.
Journal of Computer and System Sciences, 26(3) :362–391, 1983.