eFrei

PARIS PANTHÉON-ASSAS UNIVERSITÉ

Projet de Recherche opérationnelle

Département de Mathématiques Efrei Paris

Année 2024/2025 S5

I1 - NEW

1. Le problème à résoudre

1.1 Introduction

La question des problèmes de transport, tout comme de flot, est intimement liée aux questions sociales, économiques ou écologiques. A travers les algorithmes vus en cours, nous cherchons à comprendre comment mieux utiliser un réseau. La nature de ces coûts que l'on maximise ou minimise peut être humaine, monétaire ou liée à l'environnement.

1.2 Le projet

Dans ce projet, il est question de la rédaction d'un programme qui résout un problème appelé **problème de flot**. D'abord, nous vous invitons à comprendre son fonctionnement en lisant **un cours fourni en annexe 2**. Puis, il faudra construire le code qui résout ces problèmes. Nous vous demanderons alors de le tester sur les problèmes qui se trouvent en annexe 1 et dont il faudra nous fournir les traces d'exécution. Enfin, vous devez l'utiliser afin d'analyser la complexité générée.

2. Le code

2.1 Le travail demandé

Une fois le cours en annexe 2 lu, nous vous demandons d'effectuer le travail suivant. Pour tout réseau de flot (V,E,c,s,t), nous vous demandons de coder les deux algorithmes pour trouver un flot maximal : la méthode de Ford-Fulkerson et l'algorithme pousser-réétiqueter. Et pour tout réseau de flot pondéré (V,E,c,d,s,t), nous vous demandons de coder l'algorithme permettant de trouver un flot de coût minimal pour une valeur de flot donnée en entrée.

Dans ce cadre de ce projet, vous travaillerez avec le langage de programmation de votre choix : C, C++, Python, Java.

2.2 La table des réseaux de flot (V,E,c,s,t) et (V,E,c,d,s,t)

Pour chaque problème de flot, on notera $n = |V|$ le nombre de sommets avec s et t compris. Les sommets seront appelés v_i dont v_1 étant s et a_n étant t. La capacité de l'arête entre les sommets v_i et v_j est toujours notée $c(i,j)$. Et $d(i,j)$ le coût de l'arête entre les sommets v_i et v_j. Avant tout chose, il faudra créer un fichier .txt pour tout réseau de flot donné, organisé sous la forme suivante :

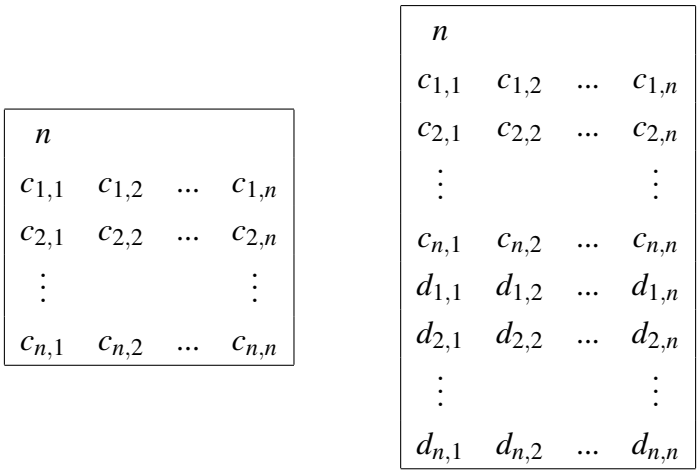


FIGURE 2.1 – Les tables demandées en .txt
Les problèmes de flot max étant à gauche et les problèmes de flot à coût min étant à droite.

En annexes, vous trouverez les graphes des dix problèmes de flot que l'on vous demande de résoudre avec votre programme. Il vous faudra éditer les dix tableaux dans 10 fichiers .txt différents de la façon ainsi définie. Ces fichiers devront être joints à votre rendu.

2.3 Le code

2.3.1 Les fonctions

Dans le programme que vous rédigerez, vous devrez mettre en place les fonctions suivantes :

1. Lecture des données issues du fichier texte (.txt) et son stockage en mémoire.
2. Affichage des tableaux suivants :
   - ⋆ matrice des capacités
   - ⋆ matrice des coûts pour les réseaux de flot avec des coûts
   - ⋆ la table issue de l'algorithme de Bellman pour les réseaux de flot avec des coûts

   Attention : la fonction affichage doit être absolument soignée. Toute table comportant des colonnes qui se décalent sera très lourdement sanctionné. La lisibilité des tables est fondamentale.
3. Algorithme pour résoudre le flot max.
   - ⋆ Pour F-F : le détail du parcours en largeur, la chaîne améliorante potentiellement trouvée avec le calcul de sa valeur de flot ainsi que les modifications sur le graphe résiduel.
   - ⋆ Pour "pousser-réétiqueter" : le détail des itérations.
4. Algorithme pour résoudre le flot à coût minimal.
   - ⋆ La table détaillée de Bellman.

- ⋆ Valeur de flot d'une chaîne améliorante potentiellement trouvée.
- ⋆ Les modifications sur le graphe résiduel.

Attention : Chaque fonction devra être mise en évidence et expliquée clairement à l'oral à l'aide d'un pseudo-code.

2.4 La structure globale

La structure globale de votre programme est illustrée par le pseudo-code suivant :

Début
- Tant que l'utilisateur décide de tester un *problème de flot*, faire :
  - ⋆ Choisir le numéro du problème à traiter
  - ⋆ Lire la table sur fichier et la stocker en mémoire
  - ⋆ Créer les matrices correspondantes représentant cette table et l'affichage
    - ⋆ S'il s'agit d'un problème de flot max, choisir l'algorithme
    - ⋆ S'il s'agit d'un problème de flot à coût min, choisir la valeur de flot
  - ⋆ Dérouler la méthode l'algorithme correspondant
  - ⋆ Affichage du flot max ou du flot à coût min selon les cas.

Fin

2.5 Les traces d'exécution

Les traces d'exécution pour les dix graphes fournis en annexes sont demandés dans le rendu. On appelle trace d'exécution ce qui est affiché par la console. Elles ne pourront pas être remplacées par des copies d'écran.

Pour les problèmes de flot max, il faudra exécuter avec votre programme avec les deux algorithmes proposés.

Les fichiers seront stockés de la façon suivante :

- ⋆ Groupe B - Equipe 4 - Problème 5 - Ford-Fulkerson : "B4-trace5-FF.txt"
- ⋆ Groupe D - Equipe 2 - Problème 5 - pousser-réétiqueter : "D2-trace5-PR.txt"
- ⋆ Groupe E - Equipe 3 - Problème 6 - flot à coût min : "E3-trace6-MIN.txt"

3. L'étude de la complexité

3.1 Introduction

Cette partie doit être abordée par toutes les équipe, à partir du moment où des fonctions marchent. Nous vous proposons maintenant d'étudier la *complexité* des algorithmes de ce projet.

Dans tout le cours donné dans ce projet, nous parlons de complexité. Mais au juste, qu'est-ce que la complexité d'un algorithme ? Il s'agit de l'évaluation des ressources nécessaires à exécution d'un algorithme (essentiellement la quantité de mémoire requise) et le temps de calcul à prévoir. Ces deux notions dépendent de nombreux paramètres matériels qui sortent du domaine de l'algorithmique : nous ne pouvons attribuer une valeur absolue ni à la quantité de mémoire requise ni au temps d'exécution d'un algorithme donné. En revanche, il est souvent possible d'évaluer l'*ordre de grandeur* de ces deux quantités de manière à identifier l'algorithme le plus efficace au sein d'un ensemble d'algorithmes résolvant le même problème.

C'est ce que nous nous proposons de faire ici en comparant les algorithmes codés.

3.2 Un peu de théorie

La plupart des algorithmes ont un temps d'exécution qui dépend non seulement de la taille des données en entrée mais des données elles-mêmes. Dans ce cas on distingue plusieurs types de complexités :

Définition 3.1 (complexité dans le pire des cas)
La **complexité dans le pire des cas** est un majorant du temps d'exécution possible pour toutes les entrées possibles d'une même taille. On l'exprime en général à l'aide de la notation O.

Définition 3.2 (complexité dans le meilleur des cas)
La **complexité dans le meilleur des cas** est un minorant du temps d'exécution possible pour toutes les entrées possibles d'une même taille. On l'exprime en général à l'aide de la notation Ω. Cependant cette notion n'est que rarement utilisée car souvent peu pertinente au regard des complexités dans le pire des cas et en moyenne.

Définition 3.3 (complexité en moyenne)
La **complexité en moyenne** est une évaluation du temps d'exécution moyen portant sur toutes les entrées possible d'une même taille supposées équiprobable.

Définition 3.4 (complexité spatiale)
La **complexité spatiale** évalue la consommation en espace mémoire. Le principe est le même sauf qu'ici on cherche à évaluer l'ordre de grandeur du volume en mémoire utilisé : il ne s'agit pas d'évaluer précisément combien d'octets sont consommés par un algorithme mais de préciser son taux de croissance en fonction de la taille n de l'entrée.

3.3 L'étude

Dans ce projet, on analysera la **complexité dans le pire des cas**. Pour cela, nous vous demandons de générer des problèmes de flot aléatoires. Puis de regarder les temps d'exécution des algorithmes.

3.3.1 Les problèmes de flot en entrée

Afin de simplifier le problème, vous travaillerez avec des problèmes de flot de taille n sommets. La matrice de capacité est toujours notée $C = (c_{i,j})_{(i,j)\in[\![1;n]\!]^2}$, ainsi que la matrice de la matrice de coûts $D = (d_{i,j})_{(i,j)\in[\![1;n]\!]^2}$.

Afin de générer un échantillonnage mimant toutes les entrées possibles d'une même taille n, vous écrirez une fonction pour éditer des problèmes de flot aléatoires. Elle se fera de la manière suivante :

1. On prend la valeur 0 pour tous les $c_{i,j}$.
2. Puis, pour $E(\frac{n^2}{2})$ couples (i, j) avec $i \neq j$, on génère un nombre aléatoire entier entre 1 et 100 inclus que l'on donnera à $c_{i,j}$, la fonction E étant la fonction partie entière.

Ainsi, la matrice C sera nulle sur la diagonale et comportera moins de la moitié de valeurs non nulles. Pour la matrice D, nous procéderons de la même façon. Bien évidemment, C et D n'ont aucune raison d'être égales, et bien évidemment, on donnera un coût aux arêtes ayant une capacité non nulle.

De plus, dans le cas de l'algorithme du flot à coût min, on ne demandera pas à l'utilisateur de fixer la valeur de flot. On prendra comme valeur la moitié de la valeur du flot max, que l'on calculera à chaque fois.

3.3.2 La mesure du temps

Une fois le problème généré, c'est à dire une fois que C et D sont fixés, il faudra stocker la valeur du temps de chaque portion de code qui nous intéresse. Par exemple en Python, il suffit simplement d'utiliser la fonction time.clock() qui renvoie le temps CPU en secondes et de stocker cette valeur. La différence entre 2 valeurs relevées donnera le temps d'exécution de la portion de code encadrée.

Avec ce problème de taille n généré, vous devrez mesurer le temps d'exécution de :

1. l'algorithme Ford-Fulkerson. On appellera ce temps $\theta_{FF}(n)$,
2. l'algorithme pousser-réétiqueter. On appellera ce temps $\theta_{PR}(n)$,
3. l'algorithme du flot à coût min. On appellera ce temps $\theta_{MIN}(n)$,

3.3.3 Le nuage de points

Pour chaque valeurs de n, vous ferez tourner 100 fois votre programme avec des valeurs aléatoires différentes pour le problème de flot. On obtiendra donc, pour n fixé, 100 valeurs de $\theta_{FF}(n)$ par exemple.

Valeurs de n à tester	10	20	40	10^2	4.10^2	10^3	4.10^3	10^4

Attention : pour effectuer cette tâche, vous travaillerez sur une seule machine à processeur unique. Les instructions seront exécutées l'une après l'autre, sans opération simultanées. Il ne faudra donc pas utiliser votre machine pour faire autre chose pendant toute l'exécution.

Une fois les valeurs stockées, vous tracerez en fonction de n, les nuages de points (les 100 valeurs pour une même abscisse) suivants :

- $\star$ $\theta_{FF}(n)$.
- $\star$ $\theta_{PR}(n)$.
- $\star$ $\theta_{MIN}(n)$.

3.3.4 La complexité dans le pire des cas par algorithme

On suppose que la complexité dans le pire des cas est l'enveloppe supérieure du nuage de points. Pour chaque valeur de n, déterminer cette valeur maximale au travers des 100 réalisations pour n fixé. Puis tracer en fonction de n la valeur maximale trouvée.

Pour θ_{FF}, θ_{PR} et θ_{MIN} identifier le type de complexité dans le pire des cas à l'aide du tableau 3.1 et comparez votre résultat avec le cours donné dans le projet :

O($log(n)$)	logarithmique
O(n)	linéaire
O($nlog(n)$)	quasi-linéaire
O(n^2)	quadratique
O(n^k) (k > 2)	polynomiale
O(k^n) (k > 1)	exponentielle

FIGURE 3.1 – Qualifications usuelles des complexités.

3.3.5 Comparaison de la complexité dans le pire des cas

Comparons maintenant les deux algorithmes résolvant le même problème pour n fixé en traçant :

$$\frac{\theta_{FF}}{\theta_{PR}}(n)$$

Tracez ensuite la valeur maximale trouvée pour chaque valeur de n et discutez des résultats.

4. Précisions sur le rendu

4.1 Le rendu

Le dépôt de votre projet se fera sur Moodle jusqu'au Samedi 14 décembre à 23h59. Aucun délais supplémentaire ne pourra être accepté. Le dimanche, vos enseignants étudieront vos projets et les premières soutenances sont le lundi.

Dans le dépôt, vous joindrez l'entièreté de vos programmes, les dix traces d'exécutions et les dix fichiers .txt des problèmes de flot à résoudre. La notation tiendra compte de la qualité des algorithmes et des traces présentées. Un rapport est demandé pour toutes les équipes sur la complexité et qui fera au maximum 5 pages.

Le dossier de rendu sera intitulé de la façon suivante : pour le groupe B et l'équipe 4 : "B4".

4.2 L'oral

Pour l'oral, il est attendu une présentation avec des diapositives. La pédagogie et la clarté sont le but de cette présentation de 10 min. Nous n'accepterons pas de code sur les diapositives : nous demandons seulement du pseudo-code. Les diapositives ne doivent évidemment pas comporter de texte écrit à la main et photographié. Toutes les formules et matrices doivent être tapées.

A l'oral, il est demandé de restituer avec précision les points les plus importants de chaque partie. Une diapositive qui résume le travail effectué à savoir les fonctions que vous avez réussies à faire fonctionner et celles non réalisées est attendue. Attention, si vous dépassez les 10 min de présentation, votre enseignant vous arrêtera.

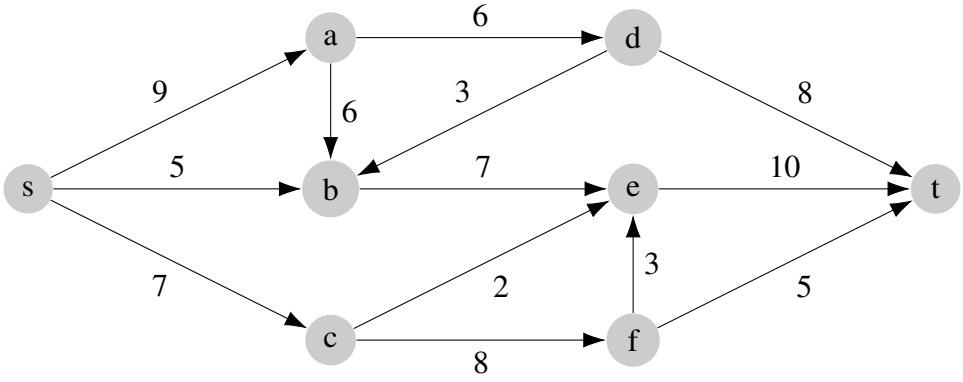
A l'issue de l'oral, des questions vous seront posées pendant 20 min : chaque étudiant sera interrogé individuellement sur le projet ou sur un point du cours. Le code devra être absolument maîtrisé par tous les membres de l'équipe.

Bon courage pour la rédaction de ce projet,

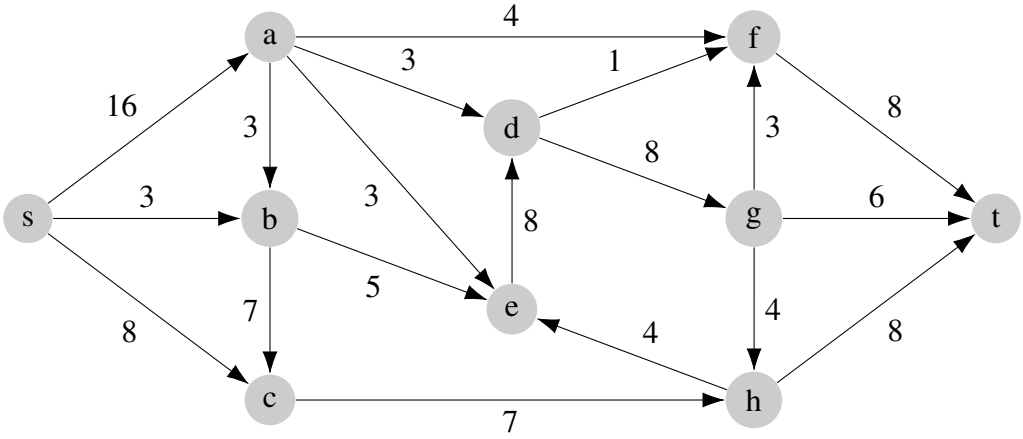
L'équipe des enseignants de Recherche Opérationnelle.

ANNEXE 1 : les dix propositions de flot

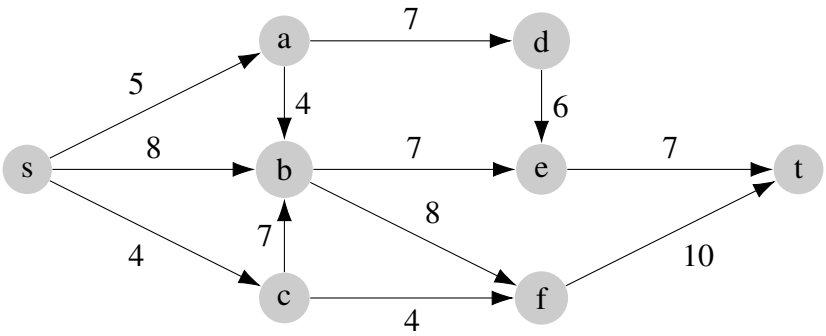
proposition 1



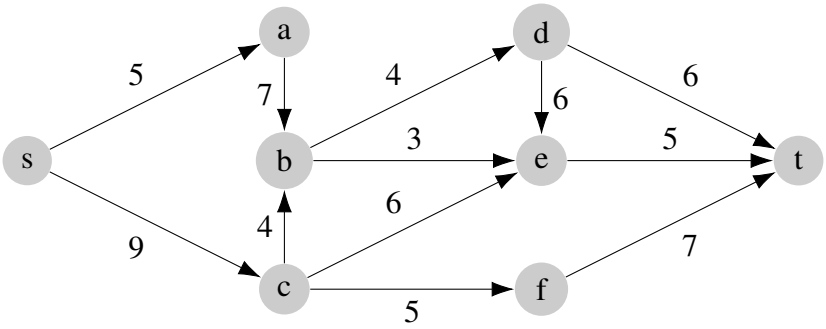
proposition 2



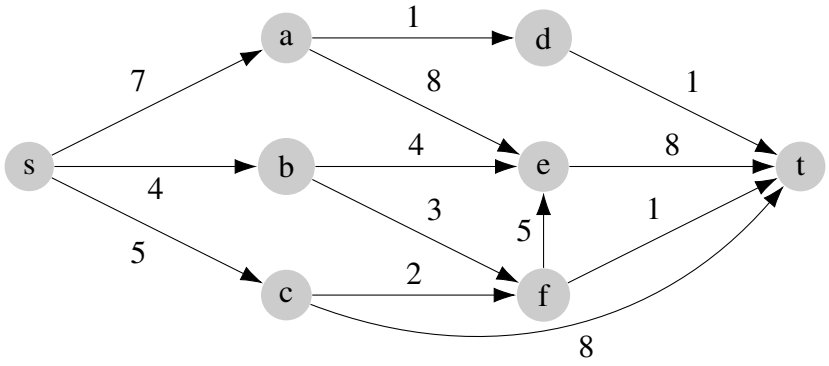
proposition 3



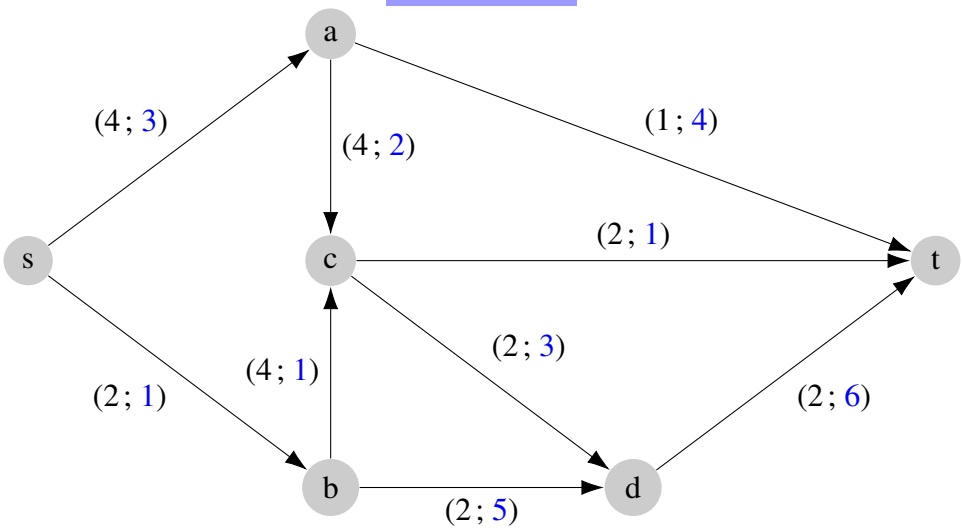
proposition 4



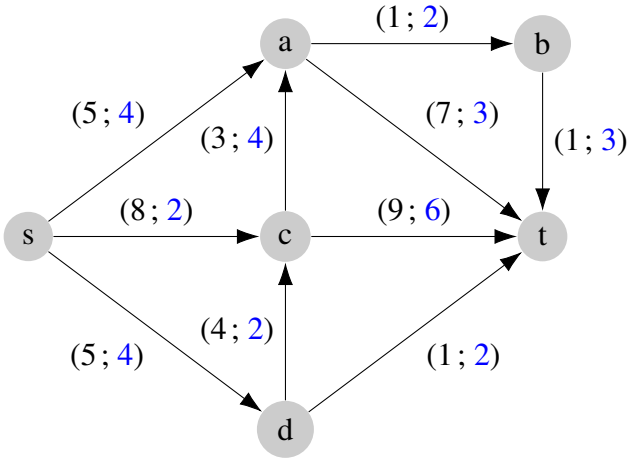
proposition 5



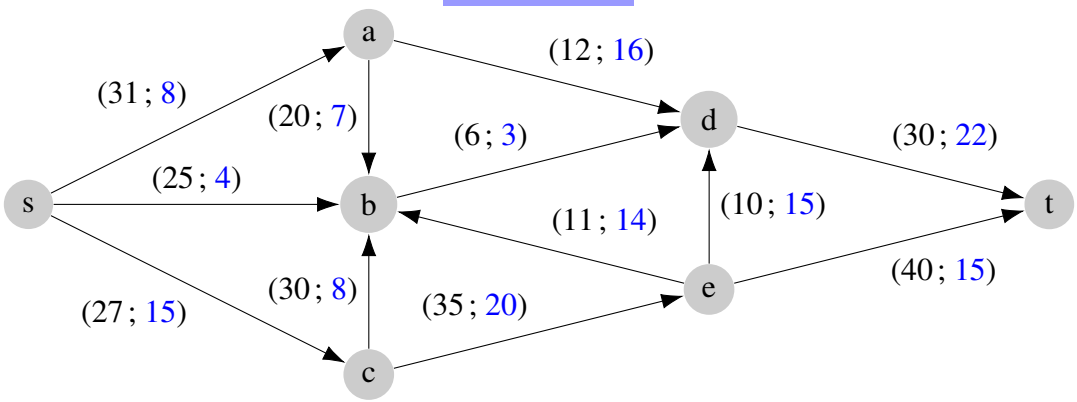
proposition 6



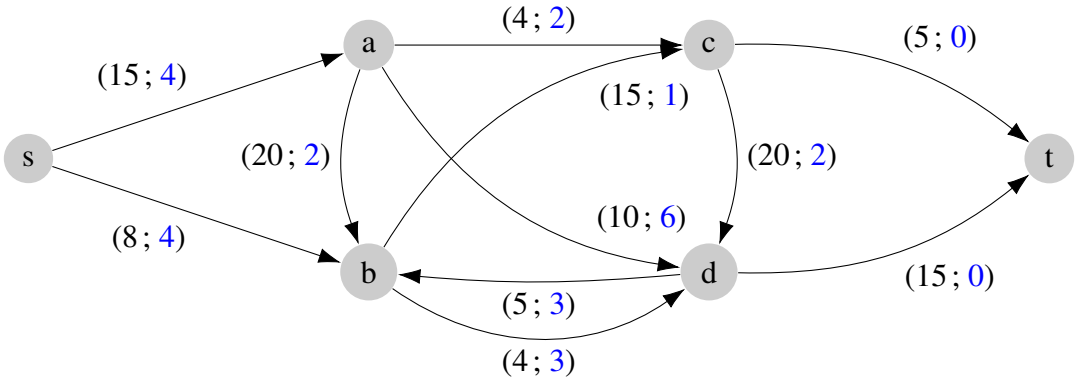
proposition 7

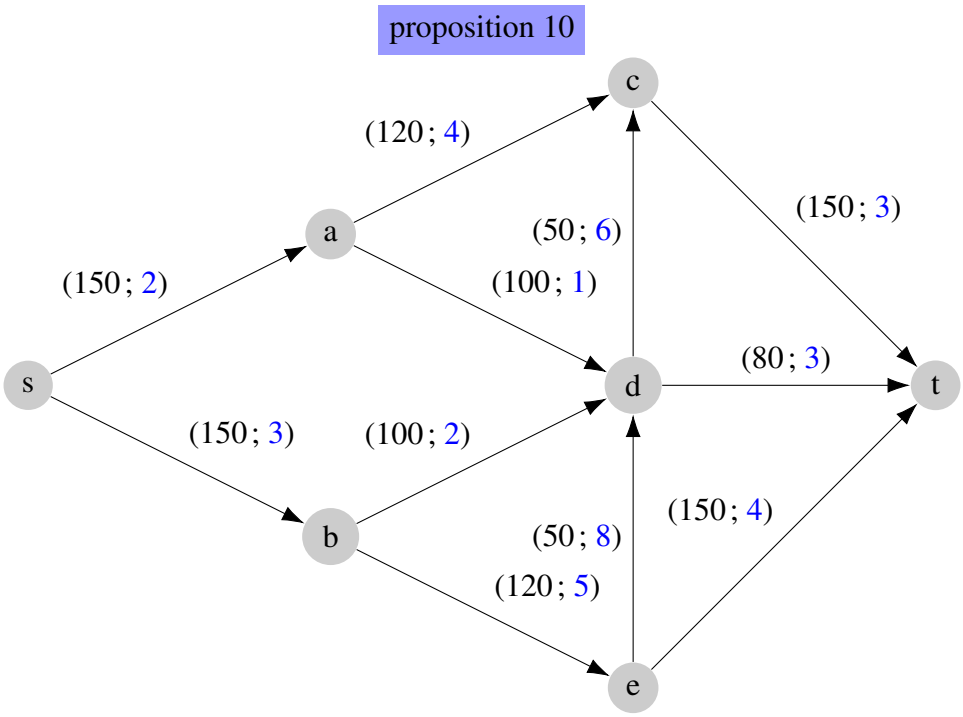


proposition 8



proposition 9

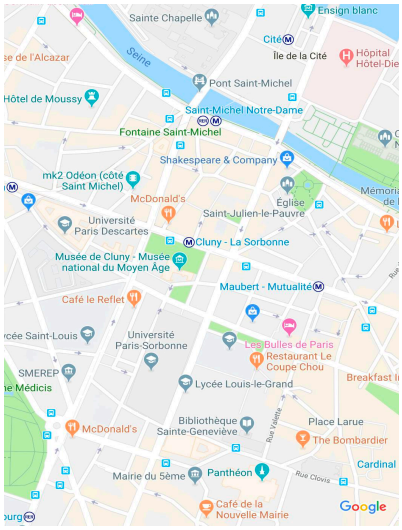



proposition 10
c
(120 ; 4)
(150 ; 3)
a
(50 ; 6)
(150 ; 2)
(100 ; 1)
(80 ; 3)
s
d
t
(150 ; 3)
(100 ; 2)
(150 ; 4)
b
(50 ; 8)
(120 ; 5)
e

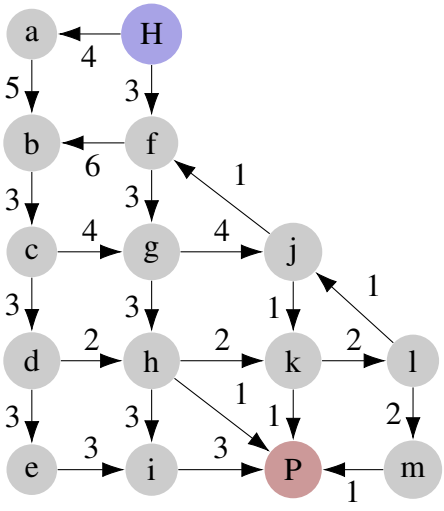
ANNEXE 2 : le cours sur les flots

5.1 Un exemple d'introduction

La directrice de l'Hôtel-Dieu de Paris veut étudier la capacité de réaction de son hôpital en cas de situations d'urgence. En particulier, elle souhaite déterminer le nombre maximal d'ambulances qui peuvent être envoyées simultanément au Panthéon (voir carte ci-dessous). Elle décide donc de modéliser la situation par un graphe orienté, où chaque arête représente une rue et chaque poids représente le nombre d'ambulances par unité de temps qui peuvent circuler à travers cette rue.



(a) Carte des rues de Paris entre l'Hôtel-Dieu et le Panthéon.



(b) Graphe modélisant le traffic dans les rues du quartier latin.

FIGURE 5.1 – Un exemple de problème de flot maximal.

Un flot pour le graphe de la Figure 5.1b est une façon particulière d'envoyer un certain nombre d'ambulances depuis le sommet H (Hôtel-Dieu) jusqu'au sommet P (Panthéon). La figure 5.2 ci-dessous montre un exemple d'un flot de 3 ambulances allant de H à P.

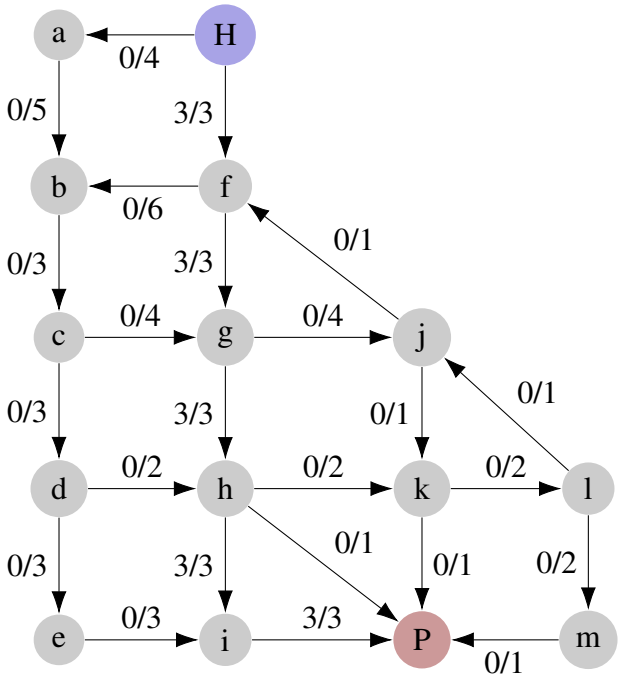


FIGURE 5.2 – Un exemple de flot.

Cependant, cet exemple n'est clairement pas *maximal*, dans le sens qu'il est possible d'envoyer davantage d'ambulances de H vers P. En effet, la figure 5.3 montre un nouvel exemple de flot, avec cette fois-ci un total de 5 ambulances. La question qui se pose maintenant est : avons-nous atteint le flot maximal grâce à ce nouveau graphe ?

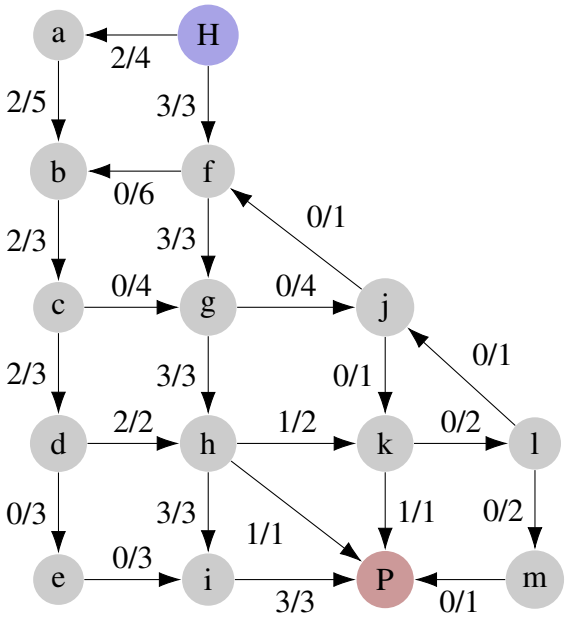


FIGURE 5.3 – Un exemple d'un flot plus important.

5.2 Le problème du flot maximal

Pour commencer, nous avons besoin d'un réseau de flot :

Définition 5.1 (Réseau de flot)
Un *réseau de flot* est un quintuplé (V,E,c,s,t) où :

i) V est un ensemble fini dont les éléments sont les *sommets* du réseau.

ii) $E \subset V \times V$ est l'ensemble des *arêtes orientées*.

iii) $c : E \longrightarrow \mathbb{R}^+ \cup \{+\infty\}$ est une application qui à chaque arête du réseau lui associe un nombre positif appelé la *capacité de l'arête*.

iv) $(s,t) \in V \times V$ sont deux sommets bien identifiés et appelés la *source* et le *puits*.

v) Pour tout sommet $v \in V$, il existe un chemin Γ depuis s vers t passant par v.

L'analogie avec un cours d'eau est très utile pour avoir une intuition du problème et sera utilisée tout le long de ce chapitre. La source s est le point duquel toute l'eau jaillit et le puits t est le point où l'eau est absorbée. La capacité $c(u,v)$ de l'arête entre les sommets u et v est la quantité maximale d'eau qui peut passer par le tuyau connectant u à v par unité de temps. Pour des raisons qui deviendront bientôt claires, nous incluons le cas où la capacité est infinie.

Etant donné un sommet u du graphe, on définit les deux ensembles suivants :

$$V_u^{in} = \big\{v \in V \,\big|\, (v,u) \in E\big\}, \qquad V_u^{out} = \big\{v \in V \,\big|\, (u,v) \in E\big\}.$$

V_u^{in} est l'ensemble des sommets des arêtes arrivant dans u, alors que V_u^{out} est l'ensemble des sommets des arêtes partant depuis u.

Définition 5.2 (Flot)
Etant donné un réseau de flot (V,E,c,s,t), un *flot* est une fonction $f : E \to \mathbb{R}^+$ telle que

i) $\forall (u,v) \in E,\ f(u,v) \leq c(u,v)$ (**contrainte de capacité**),

ii) $\forall u \in V \setminus \{s,t\},\ \displaystyle\sum_{v \in V_u^{out}} f(u,v) = \sum_{v \in V_u^{in}} f(v,u)$ (**conservation du flot**).

La **valeur totale** du flot est la quantité $|f| = \displaystyle\sum_{v \in V_s^{out}} f(s,v) = \sum_{v \in V_t^{in}} f(v,t)$. [1]

1. De façon équivalente, au lieu de définir un flot comme une fonction à valeurs positives sur les arêtes du graphe, un flot peut être défini comme une fonction à valeurs dans $\mathbb{R}$ sur tout couple de sommets. Plus précisément, on peut aussi définir un *flot* comme une fonction $f : V \times V \longrightarrow \mathbb{R}$ telle que

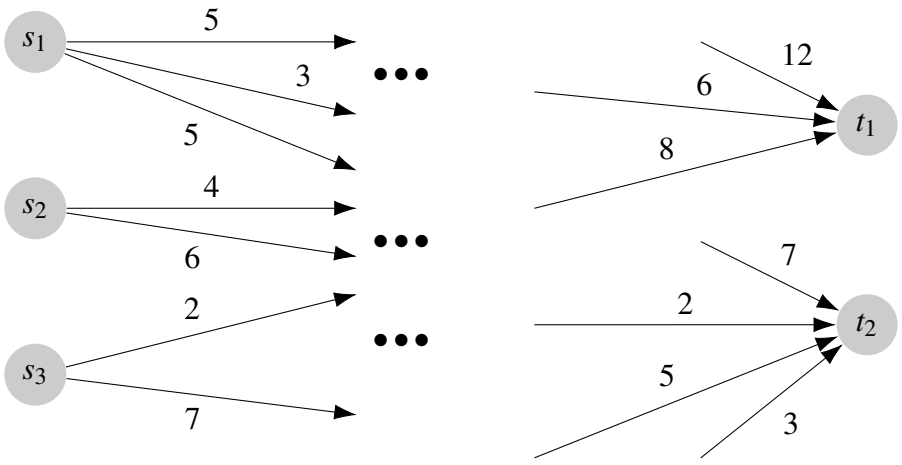
i) si $(u,v) \notin E$ et $(v,u) \notin E$, $f(u,v) = 0$,

La première condition traduit le fait que la quantité d'eau à travers un tuyau ne peut pas excéder sa capacité maximale. D'un autre côté, la deuxième condition dit que la quantité d'eau arrivant au sommet u est égale à la quantité d'eau partant de u. En d'autres mots, aucun sommet (mis à part la source et le puits) ne peut absorber ni créer de l'eau. Dans le contexte des circuits électriques, où le flot f est le courant électrique, la conservation du flot n'est rien d'autre que la *loi de Kirchhoff*.

Avec cette nouvelle terminologie, on peut maintenant formuler précisément le problème du flot maximal :

Problème du flot maximal : étant donné un réseau de flot (V,E,c,s,t), trouver un flot f tel que sa valeur totale $|f|$ soit maximale parmi tous les flots possibles.

Remarque : A première vue, on peut objecter à cette formulation du problème d'être trop spécifique et donc d'exclure des problèmes similaires qui ne rentrent pas dans la définition. En effet, dans les problèmes de flot que l'on peut rencontrer, il y a souvent plus d'une source et plus d'un puits (les ambulances peuvent partir de plusieurs hôpitaux différents ou bien vouloir arriver à plusieurs destinations différentes). Ainsi, le graphe typique d'un problème de flot ressemblera plutôt à



Cependant, avec une petite astuce, il est possible de reformuler ces problèmes de flot avec des multiples sources et puits comme des problèmes avec une seule source et un unique puits : il suffit tout simplement d'ajouter une *super-source*, un *super-puits* et des arêtes de capacités infinies reliant ces

---

ii) $\forall (u,v) \in V \times V,\ f(u,v) = -f(v,u)$ (**anti-symétrie**),

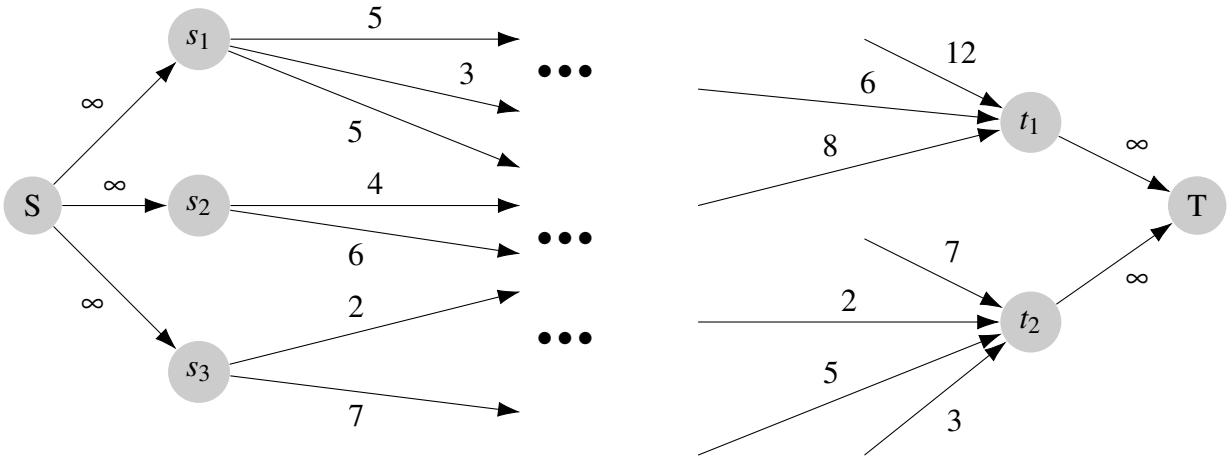
iii) $\forall (u,v) \in E,\ f(u,v) \leq c(u,v)$ (**contrainte de capacité**),

iv) $\forall u \in V \setminus \{s,t\},\ \sum_{v \in V} f(u,v) = 0$ (**conservation du flot**).

Dans ce cas, la valeur totale du flot est définie par $|f| = \sum_{v \in V} f(s,v) = \sum_{v \in V} f(v,t)$.

Cette deuxième définition plus abstraite et moins intuitive est pourtant plus utile pour prouver des propriétés générales des flots.

sommets artificiels aux vrais sources et puits.



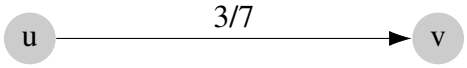
Un flot maximal pour ce deuxième réseau de flot sera une solution au problème de trouver un flot maximal pour le premier réseau.

5.3 La méthode de Ford-Fulkerson

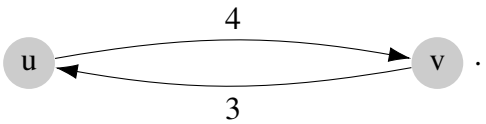
On passe maintenant à l'étude d'une des méthodes principales pour résoudre les problèmes de flot maximal.

5.3.1 Idées et outils principaux

La méthode de Ford-Fulkerson [2] est basée sur **deux outils clés : les réseaux résiduels et les chaînes améliorantes**. Etant donné un flot f sur un réseau de flot, le réseau résiduel associé encode toutes les modifications possibles que l'on pourrait décider de faire. Par exemple, si une arête (u,v) a une capacité de 7 et il y a 3 unités s'écoulant de u vers v, alors on peut encore décider d'envoyer jusqu'à 4 unités supplémentaires dans cette direction. Mais ce n'est pas tout : on peut aussi décider de diminuer ou annuler complètement le flot déjà existant. Cette possibilité de réduire le flot peut être représentée par une arête de v vers u. Ainsi, au flot



on associe le graphe résiduel suivant :



.

2. Cette méthode générale a été développée par L.R. Ford et D.R. Fulkerson en 1956.

Définition 5.3 (Réseau résiduel)

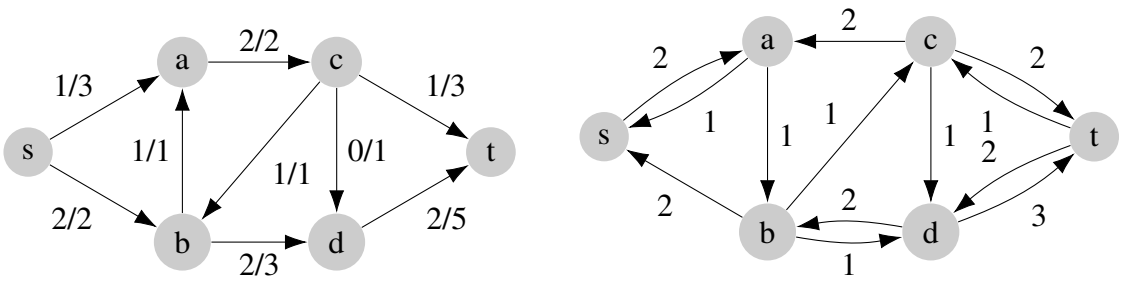
Etant donné un réseau de flot (V,E,c,s,t) et un flot $f : E \longrightarrow \mathbb{R}^+$, le réseau résiduel associé est le réseau de flot (V,E_f,c_f,s,t) défini par :

i) si $(u,v) \in E$ alors (u,v) et (v,u) sont tous les deux des éléments de E_f.

ii) $\forall (u,v) \in E_f$, $c_f(u,v) = \begin{cases} c(u,v) - f(u,v) \text{ si } (u,v) \in E \\ f(v,u) \qquad\qquad \text{si } (v,u) \in E \end{cases}$.

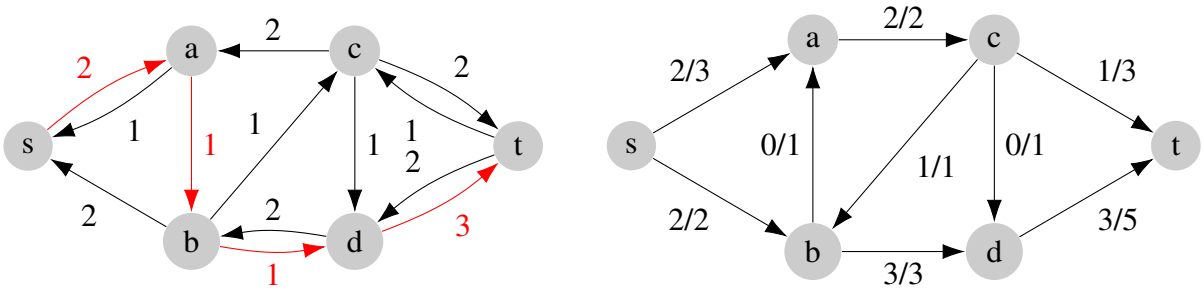
$c_f(u,v)$ est appelée la **capacité résiduelle** de l'arête (u,v).

Exemple : ci-dessous est représenté un réseau de flot avec un flot de valeur totale $|f| = 3$ (à gauche) et le réseau résiduel associé (à droite).



Regardons maintenant comment le réseau résiduel permet de déterminer si un flot donné est maximal. Le problème est équivalent à déterminer s'il est possible d'envoyer davantage d'eau depuis la source jusqu'au puits. Et puisque le réseau résiduel contient l'information de tout ce qui est encore possible de faire, tout chemin de s vers t dans le graphe résiduel nous fournira une façon d'augmenter la valeur totale du flot.

Dans l'exemple précédent, il y a un chemin de s vers t dans le graphe résiduel : **sabdt**. Il est donc possible d'envoyer plus d'eau de la source vers le puits en utilisant ce chemin. La quantité maximale que l'on peut envoyer en utilisant ce chemin est simplement la plus petite des capacités des arêtes qui le composent. Dans ce cas ci, il est possible d'envoyer une seule unité supplémentaire puisque les arêtes (a,b) et (b,d) ne permettent pas d'envoyer plus. Au passage, remarquez que envoyer une unité d'eau supplémentaire de a vers b veut en réalité dire annuler l'unité d'eau qui coulait initialement de b vers a. Ci-dessous, on représente la "chaîne améliorante" (à gauche) et le nouveau flot résultant de cette amélioration (à droite).



Définition 5.4 (Chaîne améliorante)
Etant donné un réseau de flot $G = (V,E,c,s,t)$ et un flot $f : E \longrightarrow \mathbb{R}^+$, on appelle *chaîne améliorante* tout chemin simple [3] de s vers t dans le réseau résiduel $G_f = (V,E_f,c_f,s,t)$.

Soient $G = (V,E,c,s,t)$ un réseau de flot et $f : E \longrightarrow \mathbb{R}^+$ un flot de valeur totale $|f|$ et supposons qu'il existe un chemin Γ de s vers t dans le graphe résiduel G_f. Alors, ce chemin peut être utilisé pour construire un nouveau flot f_Γ tel que $|f_\Gamma| > |f|$.

Démonstration. La preuve est juste une réécriture de l'exemple précédent dans le cas général :
Soit $c_\Gamma := \min\{c_f(u,v) \,|\, (u,v) \in \Gamma\}$, c'est-à-dire la capacité minimale des arêtes composant le chemin Γ. Puisque Γ est un chemin sans cycles,(u,v) et (v,u) ne peuvent pas être tous les deux des arêtes de Γ. On définit alors f_Γ par

$$f_\Gamma(u,v) = \begin{cases} f(u,v) + c_\Gamma \text{ si } (u,v) \in \Gamma \\ f(u,v) - c_\Gamma \text{ si } (v,u) \in \Gamma \\ f(u,v) \qquad \text{sinon.} \end{cases} \tag{5.1}$$

La fonction f_Γ ainsi définie est un flot sur G de valeur totale $|f_\Gamma| = |f| + c_\Gamma > |f|$. [4] □

Résumons la situation : étant donné un flot sur un réseau de flot, on construit le réseau résiduel. Si, dans ce nouveau réseau, il existe un chemin depuis la source vers le puits, alors nous savons construire un nouveau flot avec une valeur totale plus grande. Ainsi, dès lors qu'il existe une chaîne améliorante, un flot n'est pas maximal. Il est donc naturel de se poser la question inverse : la réciproque est-elle vraie ? Le théorème suivant répond par l'affirmative :

[Le théorème de la chaîne améliorante [5]] Etant donné un flot f sur un réseau de flot G, on a

$$\text{Le flot } f \text{ est maximal} \iff \text{Il n'existe pas de chemin de } s \text{ vers } t \text{ dans le réseau résiduel } G_f.$$

3. Un chemin est dit être *simple* s'il ne contient pas de cycles.
4. EXERCICE BONUS : Montrez explicitement que f_Γ vérifie les conditions d'un flot.

Démonstration. Voir subsection 5.3.4. □

Ce théorème affirme donc que rechercher des chaînes améliorantes est une bonne stratégie pour résoudre le problème du flot maximal. C'est le résultat fondamental sur lequel est construit la méthode de Ford-Fulkerson :

MÉTHODE DE FORD-FULKERSON (V,E,c,s,t)
1 Initialiser le flot à zéro.
2 **Tant que** il existe une chaîne améliorante Γ.
3 **Faire** augmenter le flot f en suivant le chemin Γ en utilisant l'équation (5.1).
4 **Retourner** f.

5.3.2 Mise en oeuvre : l'algorithme d'Edmonds-Karp

La méthode de Ford-Fulkerson fournit une solution au problème du flot maximal *pourvu que l'on sache trouver les chaînes améliorantes*. Cependant, la méthode pour trouver ces chemins de la source au puits dans les réseaux résiduels n'est pas explicitée et c'est pour cela que l'on parle de la *méthode* (plutôt que de l'algorithme) de Ford-Fulkerson. L'efficacité de cette méthode pour trouver un flot maximal dans de très grands réseaux dépend de l'algorithme utilisé pour trouver les chaînes améliorantes.

L'algorithme de Edmonds-Karp, introduit d'abord par Y. Dinic en 1970 et puis, de façon indépendante, par J. Edmonds et R. Karp en 1972, résout le problème du flot maximal encombinant la méthode de Ford-Fulkerson et l'algorithme de parcours en largeur qui permet de trouver le plus court chemin entre deux sommets dans un graphe orienté non pondéré. Ainsi, à chaque étape de la boucle "tant que" de la méthode Ford-Fulkerson, l'algorithme de Edmonds-Karp trouve comme une chaîne améliorante qui est aussi le plus court chemin entre s et t dans le graphe résiduel.

Algorithme de Edmonds-Karp	=	Méthode de Ford-Fulkerson	+	Algorithme de parcours en largeur

5. Ce théorème est plus souvent appelé le "max-flow min-cut theorem", mais comme la notion de "cut" (coupure) n'est pas au programme de ce cours, nous avons décidé de changer la terminologie. Pour les curieux, nous avons cependant inclus la version complète du théorème à la fin du chapitre (voir théorème 5.3.4 page 21).

Revenons rapidement sur le fonctionnement de l'algorithme de parcours en largeur appliqué au graphe résiduel (V, E_f) [6]. En partant du sommet s, l'algorithme trouve d'abord tous les sommets voisins de s (c'est-à-dire, tous les sommets tels que $(s,u) \in E_f$). Dans la deuxième itération, l'algorithme trouve tous les sommets non explorés qui sont voisins d'un voisin de s (c'est-à-dire, des sommets accessibles en deux étapes à partir de s). Puis, dans les itérations suivantes, l'algorithme trouve tous les sommets accessibles en trois étapes à partir de s. Et ainsi de suite jusqu'à atteindre le sommet t.

Plus précisément, l'algorithme se base sur deux ingrédients :

- La relation **ancêtre-descendant**.
- La **file** (premier arrivé, premier servi) [7].

Au début, aucun sommet est exploré et ils sont tous marqués comme ayant un parent inconnu ($\pi[v] = NIL$). Au fur et à mesure que l'algorithme progresse, le sommet u qui permet d'atteindre le sommet v pour la première fois est appelé le "parent de v", ce qui est noté $\pi[v] = u$:

PARCOURS EN LARGEUR – EDMONDS-KARP (V, E_f, s, t)

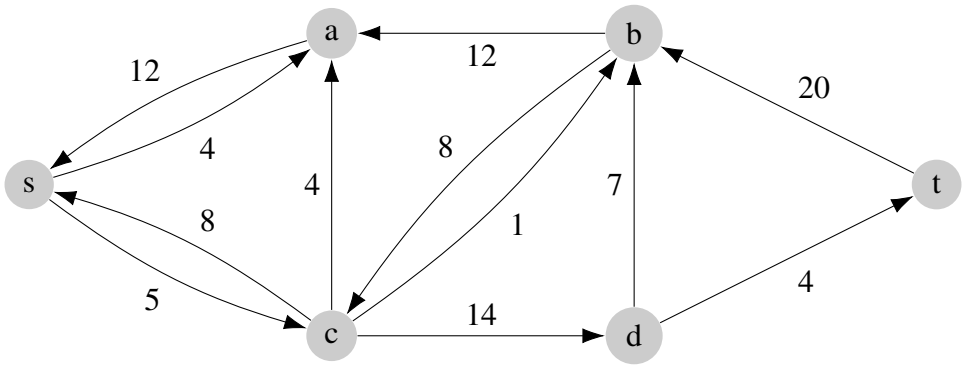
```
1 Pour chaque sommet u ∈ V.
2     faire π[u] ← NIL
3 Q ← ∅
4 ENFILE (Q, s)
5 Tant que Q ≠ ∅
6   Faire u ← DEFILE(Q)
7     Pour chaque v ∈ V tel que (u, v) ∈ E_f et π[v] =NIL
8       Faire π[v] ← u
9         Si v = t
10          Alors Retourner "Chaîne améliorante existe"
11        Sinon ENFILE (Q, v)
12 Retourner "Flot est maximal"
```

Dans la boucle de la ligne 7, on doit spécifier un ordre dans lequel les sommets de V vont être parcourus. La chaîne améliorante trouvée à la fin va en général dépendre de ce choix (bien que la taille de cette chaîne soit indépendante du choix). Dans tous les exemples que nous traiterons, les sommets du graphe seront indexés par des lettres ou des nombres, auquel cas **les sommets seront parcourus dans l'ordre alphabétique ou dans l'ordre croissant**.

6. On rappelle aussi que l'algorithme de parcours en largeur n'est rien d'autre que l'algorithme de Dijkstra appliqué au cas particulier d'un graphe orienté où les poids de toutes les arêtes sont égaux.

7. En opposition avec la pile : "dernier arrivé, premier servi".

Exemple : Regardons comment l'algorithme de parcours en largeur fonctionne pour le réseau résiduel (V, E_f, c_f, s, t) suivant :



Point de départ : $Q = s$
Première itération : $Q = ac$, $\pi(a) = \pi(c)$=s
Deuxième itération : $Q = c$, (pas de descendants de a)
Troisième itération : $Q = bd$, $\pi(b) = \pi(d) = c$
Quatrième itération : $Q = d$, (pas de descendants de b)
Cinquième itération : $Q = t$, $\pi(t) = d$
Une chaîne améliorante existe : **scdt**.

• Complexité de l'algorithme d'Edmonds-Karp

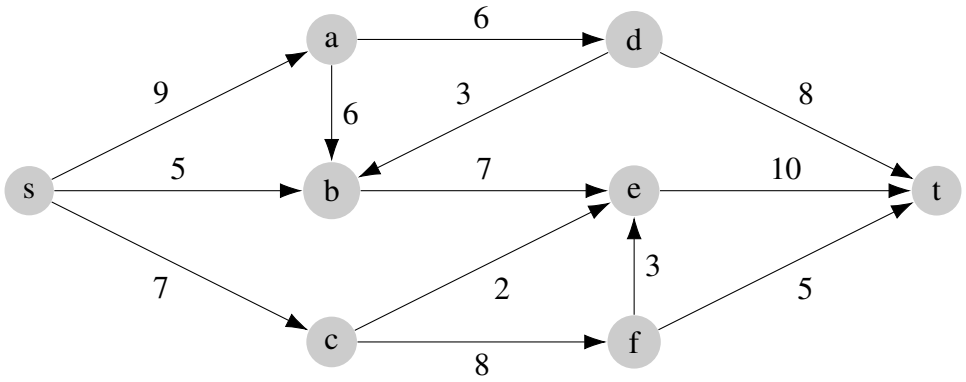
D'abord, on analyse la complexité de l'algorithme de parcours en largeur. La boucle pour des lignes 1–3 prend un temps $O(|V|)$. De plus, puisque chaque sommet peut être inséré au plus une fois dans la queue Q, le temps total pris par les opérations d'enfiler et défiler est $O(|V|)$. Enfin, la boucle pour des lignes 7–11 va parcourir tous les sommets de E_f et prendra donc un temps $O(|E_f|)$. Or, puisque $|E_f| \leq 2|E|$ et $|V| \leq |E| \leq |V|^2$, le temps total de l'algorithme est $O(2|V| + 2|E|) = O(|E|)$.

Ensuite, il faut déterminer le nombre de fois que l'algorithme de parcours en largeur est utilisé à l'intérieur de l'algorithme de Edmonds-Karp. En d'autre mots, il faut évaluer le nombre de chaînes améliorantes nécessaires pour atteindre le flot maximal. On admettra que ce nombre se comporte comme $O(|V||E|)$ [8]. Ainsi, l'on a [Complexité d'Edmonds-Karp] Etant donné un réseau de flot $G = (V, E, c, s, t)$, le temps total d'exécution de l'algorithme d'Edmonds-Karp se comporte comme $O(|V||E|^2)$.

8. Voir *Introduction to Algorithms*, p. 662–663, pour une preuve.

5.3.3 Exemple complet

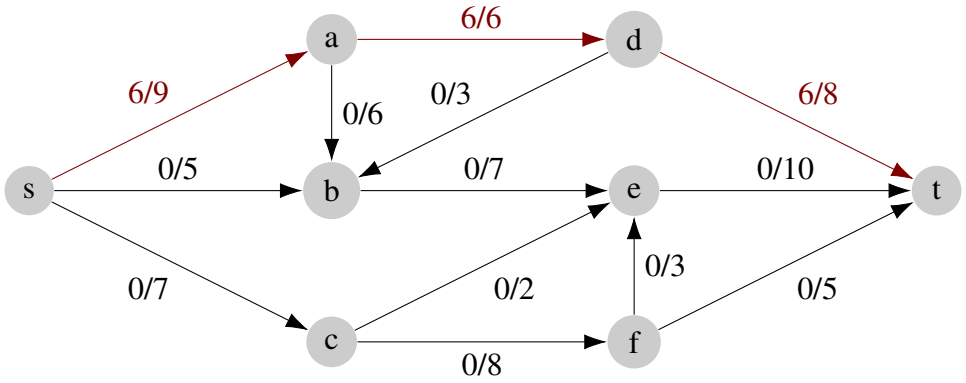
Considérons le réseau de flot ci-dessous et appliquons l'algorithme d'Edmonds-Karp pour trouver un flot maximal.



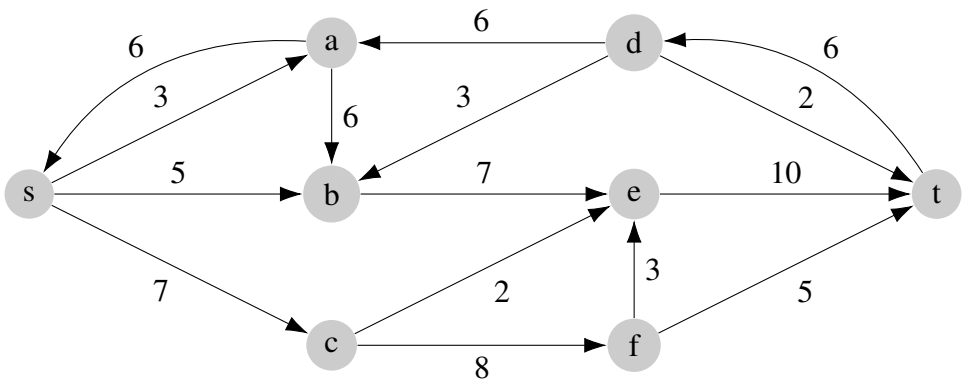
Puisque le flot initial est nul, le premier graphe résiduel est tout simplement le graphe de départ. Dans le recherche d'une première chaîne améliorante, l'algorithme produit :

- Q = s
- Q = abc, $\pi(a) = \pi(b) = \pi(c) = s$
- Q = bcd, $\pi(d) = a$
- Q = cde, $\pi(e) = b$
- Q = def, $\pi(f) = c$
- Q = eft, $\pi(t) = d$

La première chaîne améliorante est donc **sadt de flot 6**. Ci-dessous, on montre le réseau de flot qui en résulte ainsi que son graphe résiduel associé [9].



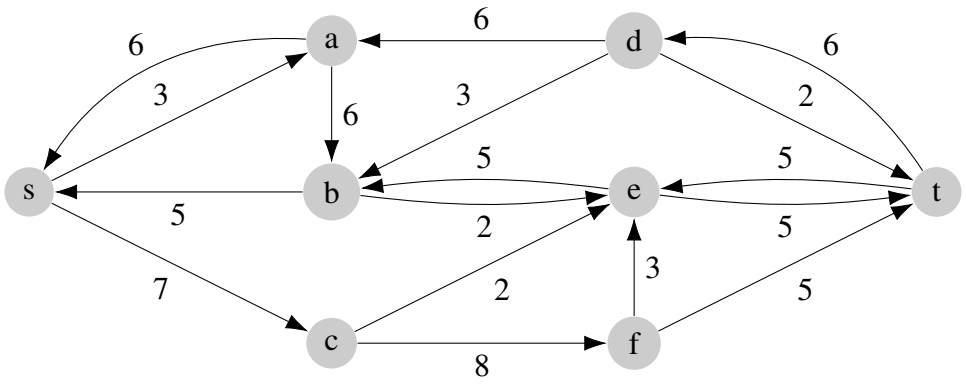
9. Pour simplifier, toues les arêtes ayant une capacité nulle dans le graphe résiduel ne sont pas représentées.



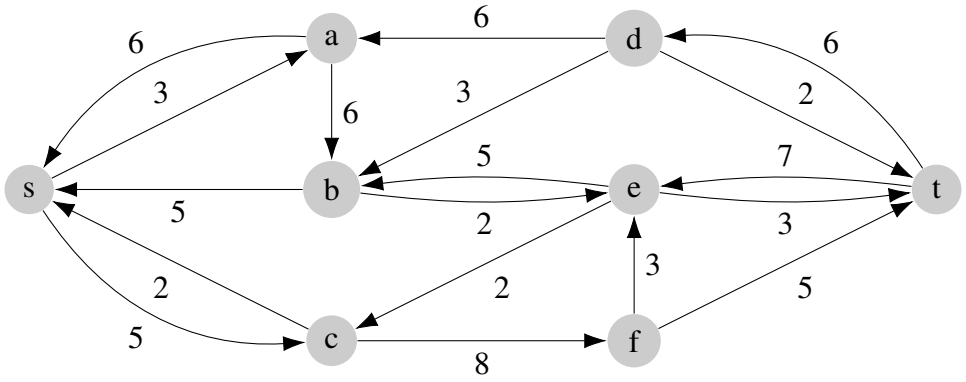
Dans la recherche d'une deuxième chaîne améliorante, l'algorithme d'Edmonds-Karp produit

- Q = s
- Q = abc, $\pi(a) = \pi(b) = \pi(c) = s$
- Q = bc,
- Q = ce, $\pi(e) = b$
- Q = ef, $\pi(f) = c$
- Q = ft, $\pi(t) = e$

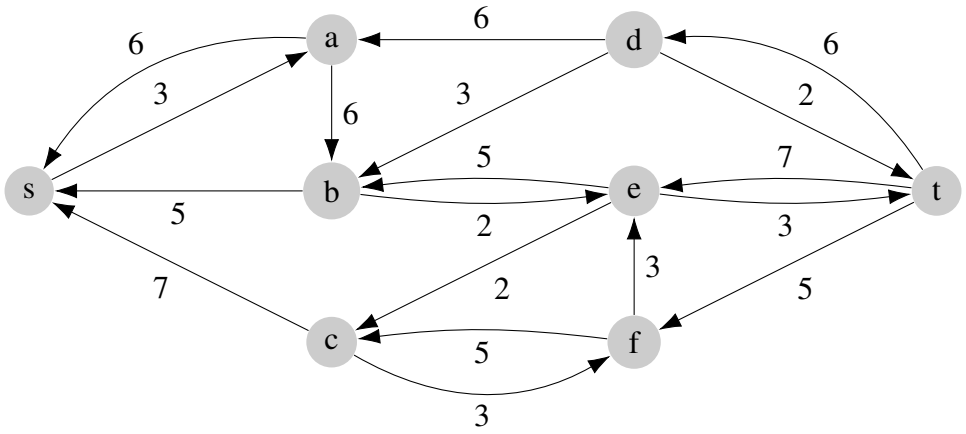
La deuxième chaîne améliorante est donc **sbet de flot 5**. Le graphe résiduel résultant de cette modification du flot est :



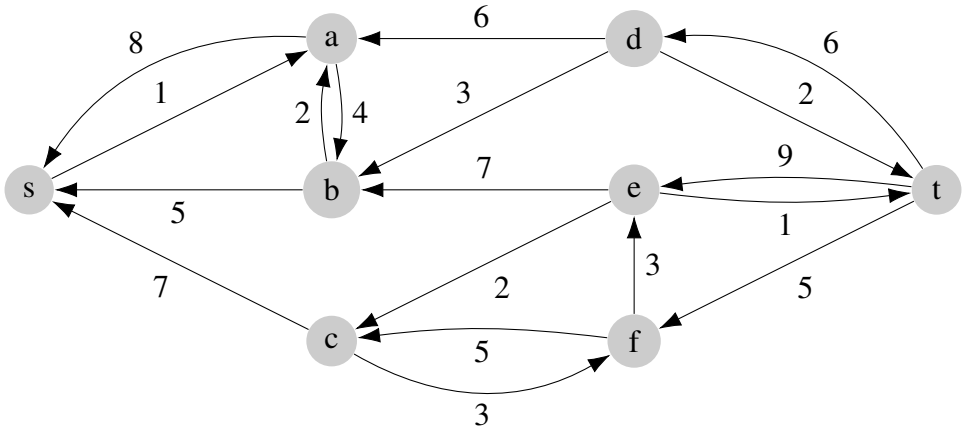
La troisième chaîne améliorante est **scet de flot 2**. Le graphe résiduel est donc modifié en :



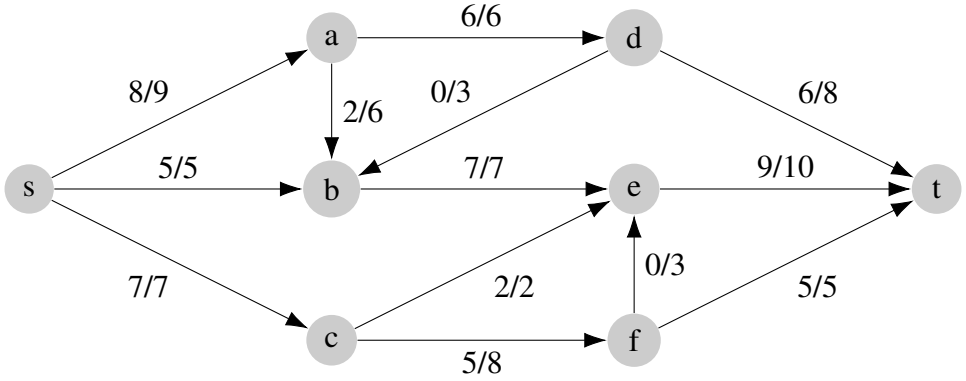
La quatrième chaîne améliorante est **scft de flot 5**. Le graphe résiduel est donc modifié en :



La cinquième chaîne améliorante est **sabet de flot 2**, ce qui mène au réseau résiduel ci-dessous



Enfin, dans ce dernier réseau résiduel il n'y a pas de chemins connectant la source s au puits t, donc l'algorithme s'achève. Ainsi, **la valeur maximale de flot est 20** et une solution permettant d'atteindre cette valeur est :



5.3.4 Preuve du théorème de la chaîne améliorante [10]

Dans toute cette section, on travaillera avec la deuxième définition d'un flot (voir note 1, page 9) et l'on fixe un réseau de flot $G = (V, E, c, s, t,)$ et un flot $f : V \times V \longrightarrow \mathbb{R}$. Pour simplifier les calculs, on adopte la notation suivante : pour tout sommet $v \in V$ et pour tout sous-ensemble de sommets $X \subset V$, on définit les quantités

$$f(u,X) := \sum_{x \in X} f(u,x) \quad \text{and} \quad c(u,X) := \sum_{x \in X} c(u,x).$$

De même, étant données deux sous-ensembles de sommets X et Y, on note $f(X,Y)$ et $c(X,Y)$ les quantités

$$f(X,Y) := \sum_{x \in X} \sum_{y \in Y} f(x,y) \quad \text{and} \quad c(X,Y) := \sum_{x \in X} \sum_{y \in Y} c(x,y).$$

Notons ici quelques propriétés générales des flots qui découlent de la définition 5.2.

Lemme 5.1

(1) $\forall X \subset V,\ f(X,X) = 0.$

(2) $\forall X, Y \subset V,\ f(X,Y) = -f(Y,X).$

(3) $\forall X, Y, Z \subset V$ tels que $X \cap Y = \emptyset,\ f(X \cup Y, Z) = f(X,Z) + f(Y,Z).$

Démonstration. EXERCICE BONUS ! [11] □

La preuve la plus simple du théorème de la chaîne améliorante utilise la notion de *coupure* :

Définition 5.5 (Coupure)
Etant donné un réseau de flot G, une *coupure* de G est une partition de V en deux sous-ensembles (S,T) tels que :
i) $S \cap T = \emptyset$, ii) $S \cup T = V$, iii) $s \in S$, iv) $t \in T$.

Dans la preuve le théorème, nous aurons besoin des deux propriétés suivantes des coupures :

Lemme 5.2

Pour toute coupure (S,T) du réseau de flot G, on a $f(S,T) = |f|$.

10. Cette section est hors-programme. Elle est incluse pour les étudiants intéressés qui voudraient comprendre plus en détail le cours.

11. Piste pour la première : utilisez une récurrence sur la taille de X.

Démonstration.

$$\begin{aligned} f(S,T) &= f(S,V) - f(S,S) && \text{puisque } V = S \cup T \text{ et } S \cap T = \emptyset \\ &= f(S,V) && \text{d'après le Lemme 5.1} \\ &= f(s,V) + f(S \setminus s, V) && \text{d'après le Lemme 5.1} \\ &= f(s,V) && \text{d'après la conservation du flot} \\ &= |f|. \end{aligned}$$

□

Lemme 5.3

Pour toute coupure (S,T), on a $|f| \leq c(S,T)$.

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

Nous avons maintenant tous les ingrédients nécessaires pour prouver le théorème de la chaîne améliorante, aussi appelé le "théorème coupure minimale / flot maximal".

[Le théorème flot maximal/coupe minimale] Etant donné un flot f sur un réseau de flot G, les trois affirmations suivantes sont équivalentes :

(1) Le flot f est maximal.

(2) Il n'existe pas de chemin de s vers t dans le réseau résiduel G_f.

(3) Il existe une coupure (S,T) telle que $|f| = c(S,T)$.

Démonstration. Les implications $(3) \Longrightarrow (1)$ et $(1) \Longrightarrow (2)$ sont maintenant évidents : la première est une conséquence directe du Lemme 5.3 puisque $c(S,T)$ est une borne supérieure pour *n'importe quel* flot, alors que la seconde est la Proposition 5.3.1.

Ainsi, le seul travail qu'il reste à faire est de prouver $(2) \Longrightarrow (3)$. Supposons donc qu'il n'existe pas de chemin dans G_f depuis s vers t, et définissons la coupure suivante de G :

$$S = \{v \in V \text{ tel qu'il existe un chemin dans } G_f \text{ de } s \text{ vers } v\}$$
$$T = \{v \in V \text{ tel qu'il n'existe pas de chemin dans } G_f \text{ de } s \text{ vers } v\}.$$

On a clairement $S \cap T = \emptyset$, $S \cup T = V$, $s \in S$ et $t \in T$. Donc (S,T) est bien une coupure de G.

Maintenant, pour tout $(u,v) \in S \times T$, puisque u est connecté à s dans G_f et v ne l'est pas, on a nécessairement $(u,v) \notin E_f$. Cela veut dire que $c_f(u,v) = 0$, donc, par définition de c_f, $f(u,v) = c(u,v)$. D'où

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

□

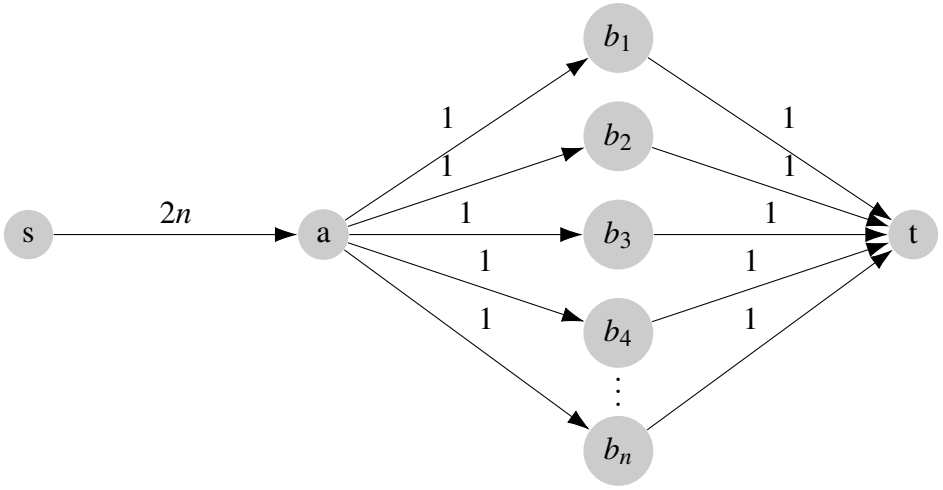
5.4 L'algorithme pousser-réétiqueter

L'algorithme pousser-réétiqueter (Push-Relabel en anglais) a été présenté pour la première fois en 1986 par A.V. Goldberg et R. Tarjan, en se basant sur des idées que A.V. Karzanov avait développé au milieu des années 1970. L'idée est peut-être moins intuitive que la méthode de Ford-Fulkerson mais l'algorithme pousser-réétiqueter est plus efficace que l'algorithme d'Edmonds-Karp puisque sa complexité est d'ordre $O(|V|^3)$.

5.4.1 Intuitions

Pour comprendre les idées principales derrière l'algorithme pousser-réétiqueter, il peut être utile de commencer par une analogie en physique. Le problème du flot maximal peut être formulé comme suit : le réseau de flot est un réseau de rivières de tailles différentes (la quantité maximale d'eau qui peut couler dans chaque rivière est différente) et chaque sommet peut être pensé comme un lac. A l'était initial, la source s produit autant d'eau que possible —c'est-à-dire que toutes les arêtes/rivières partant de s sont saturées. Or, puisque l'eau ne peut couler que vers le bas, la source doit être placée à une plus grande altitude que tous les lacs. Etant donnée cette situation initiale, le problème qui se pose est de savoir *comment gérer cette arrivée d'eau en évitant que les lacs ne débordent*. Inévitablement, cela nécessitera de *pousser* l'eau arrivant à un sommet/lac vers d'autres sommets/lacs, ce qui à son tour, nécessite que l'on puisse *réétiqueter*/relocaliser les lacs à des plus grandes altitudes pour que l'eau puisse bien couler vers le bas lorsqu'elle est poussée d'un lac vers un autre. De cette façon, le danger de débordement est déplacé de certains sommets vers d'autres, ce qui demandera à nouveau de pousser l'eau dans de nouvelles directions. Enfin, il peut arriver que le réseau de rivières et lacs soit incapable de gérer la quantité totale d'eau initialement envoyée par la source, auquel cas, *et seulement en dernière recours*, l'excédent d'eau sera retourné à la source.

D'un point de vue pratique, voici l'exemple paradigmatique de situations où l'inefficacité de l'algorithme d'Edmonds-Karp apparaît clairement :



Ici, la valeur maximale du flot est clairement n. L'algorithme de Edmonds-Karp arrivera à la solution en trouvant d'abord la chaîne améliorante sab_1t, puis sab_2t, sab_3t etc. A chaque itération, une unité supplémentaire de flot sera envoyée de s à a. Au total, le nombre d'opérations effectuées par Edmonds-Karp sera ici de l'ordre $O(n^2)$. Il est cependant beaucoup plus rapide (et probablement beaucoup plus proche de la façon qui nous fait "voir" immédiatement la solution) de d'abord envoyer $2n$ unités de s à a et puis se demander combien de ces $2n$ unités peuvent être envoyées de a à t. La solution sera ainsi trouvée en $O(n)$ opérations.

Ainsi, la différence fondamentale entre l'algorithme pousser-réétiqueter et la méthode de Ford-Fulkerson est que la *contrainte de conservation du flot est maintenant relâchée : à chaque étape intermédiaire, il y aura des sommets excédents ayant un flot en entrée plus grand que le flot en sortie*.

5.4.2 Cadre formel

Le théorème de la chaîne améliorante est le résultat fondamental qui ouvre la voie pour toute stratégie de résolution du problème du flot maximal. Pour rappel, ce théorème dit qu'un flot f sur un réseau de flot G est maximal si et seulement si s et t ne sont pas connectés dans le réseau résiduel G_f.

Etant donné ce résultat, la méthode de Ford-Fulkerson consistait à adopter la stratégie suivante :

Invariant de l'algorithme : à chaque itération, maintenir un flot f de s vers t.

Objectif :travailler pour progressivement déconnecter s de t dans le réseau residuel G_f.

A l'inverse, l'algorithme pousser-réétiqueter adopte la stratégie complémentaire :

Invariant de l'algorithme : à chaque itération, maintenir s déconnecté de t dans le réseau residuel G_f.

Objectif : travailler pour construire progressivement un flot f (i.e., récupérer les contraintes de conservation).

Pour mettre en oeuvre cette idée, la première étape est de définir un cadre dans lequel la contrainte de conservation peut être relachée :

Définition 5.6 (Pré-flot)
Etant donné un réseau de flot $G = (V, E, c, s, t)$, on appelle *pré-flot* une fonction $f : E \longrightarrow \mathbb{R}^+$ telle que :

i) $\forall (u,v) \in E,\ f(u,v) \leq c(u,v)$ (**contrainte de capacité**),

ii) $\forall u \in V \setminus \{s,t\},\ e(u) := \sum_{v \in V_u^{in}} f(v,u) - \sum_{v \in V_u^{out}} f(u,v) \geq 0.$

La quantité $e(u)$ est appelée **l'excédent** de u. [12]

La deuxième condition distingue les pré-flots des flots. Elle laisse ouverte la possibilité que la quantité d'eau sortant d'un sommet soit plus petite (ou égale) que la quantité d'eau arrivant au sommet. Néanmoins, en aucun cas (sauf si le sommet est la source s) la quantité d'eau sortant d'un sommet peut être supérieure à celle qui arrive.

Etant donné un pré-flot f sur un réseau de flot G, on construit le réseau résiduel associé exactement de la même manière qu'avec les flots.

L'étape suivante est de formaliser la notion d'"altitude" (de sorte que l'eau ne coule que vers le bas) :

Définition 5.7 (Hauteur)
Etant donné un pré-flot f sur un réseau de flot G, on appelle *hauteur* toute fonction $h : V \longrightarrow \mathbb{N}$ telle que :

i) $h(s) = |V|$,

ii) $h(t) = 0$,

iii) $\forall (u,v) \in E_f, h(u) - h(v) \leq 1$.

L'objectif principal de cette définition est de s'assurer qu'il n'y aura pas de chemin allant de s vers t. En effet, le chemin le plus court allant de s vers t (s'il existe) doit être de longueur au plus $|V| - 1$ (ce qui voudrait dire passer par tous les sommets du graphe avant d'atteindre t). Or la condition *iii)* assure que, en suivant une arête du graphe résiduel, la hauteur ne peut décroitre de plus de 1. Ainsi, si un chemin existait entre s et t, on devrait avoir $h(s) - h(t) \leq |V| - 1$ ce qui est exclu par les conditions *i)* et *ii)*.

On peut maintenant décrire les trois opérations fondamentales qui composent l'algorithme pousser-réetiqueter.

i) INITIALISATION : cette opération situe simplement tous les sommets à une hauteur nulle (sauf la source, forcée à être à une hauteur fixe de $|V|$) et définit le pré-flot initial en saturant toutes les arêtes qui partent de la source s.

12. De façon analogue à la note 1 page 9, la notion de pré-flot peut aussi être définie de façon équivalente comme une fonction $f : V \times V \longrightarrow \mathbb{R}$ telle que :

i) si $(u,v) \notin E$ et $(v,u) \notin E$, $f(u,v) = 0$,

ii) $\forall (u,v) \in E$, $f(u,v) \leq c(u,v)$ (**contrainte de capacité**),

iii) $\forall (u,v) \in V \times V$, $f(u,v) = -f(v,u)$ (**anti-symétrie**),

iv) $\forall u \in V \setminus \{s,t\}$, $e(u) := \sum_{v \in V} f(v,u) \geq 0$.

INITIALISER (V,E,c,s,t)

1 Initialiser le flot à zéro.
2 **Pour** chaque sommet $u \in V \setminus \{s\}$.
3 **Faire** $h[u] \leftarrow 0$
4 $e[u] \leftarrow 0$
5 $h[s] \leftarrow |V|$
6 **Pour** chaque sommet $u \in V$ tel que $(s,u) \in E$
7 **Faire** $f[s,u] \leftarrow c[s,u]$
8 $e[u] \leftarrow c[s,u]$
9 $e[s] \leftarrow e[s] - c[s,u]$

ii) POUSSER : Etant donné un sommet u avec un excédent non nul, on pousse autant d'eau que possible en suivant les arêtes sortant de u et allant vers une plus petite hauteur. Par conséquent, l'excédent de u diminue mais l'excédent des sommets voisins augmente.

POUSSER (u,v)

1 **Si** $e[u] \geq 0$, $c_f[u,v] > 0$ et $h[u] - h[v] = 1$
2 **Alors** $d_f[u,v] \leftarrow \min(e[u], c_f[u,v])$.
3 $f[u,v] \leftarrow f[u,v] + d_f[u,v]$
4 $e[u] \leftarrow e[u] - d_f[u,v]$
5 $e[v] \leftarrow e[v] + d[u,v]$

iii) RÉÉTIQUETER : Enfin, étant donné un sommet u avec un excédent non nul et sans voisins à une hauteur plus petite, changer la hauteur de u de sorte qu'au moins un de ses voisins dans G_f soit à une hauteur plus petite que $h(u)$.

RÉÉTIQUETER (u)

1 **Si** $e[u] \geq 0$ et $h[u] - h[v] < 1$ pour tout $(u,v) \in E_f$
2 **Alors** $h[u] \leftarrow 1 + \min\big(h[v] : (u,v) \in E_f\big)$.

Etant données ces opérations, l'algorithme est extrêmement simple :

POUSSER-RÉÉTIQUETER (V,E,c,s,t)

1 INITIALISER (V,E,c,s,t)
2 **Tant qu'** il soit possible d'effectuer une opération POUSSER ou RÉÉTIQUETER
3 **Faire** choisir une des ces opérations et l'effectuer.

En pratique, il faut aussi prescrire un ordre dans lequel les opérations POUSSER et RÉÉTIQUETER seront effectuées. Ici, on s'accorde de choisir les priorités suivantes :

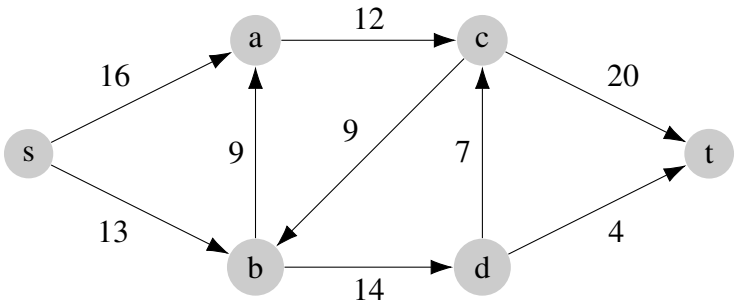
Priorité des opérations dans l'algorithme Pousser-réétiqueter

- S'il y a plusieurs sommets excédentaires, choisir celui avec la plus grande hauteur. En cas d'égalité, choisir par ordre alphabétique.
- Pour un sommet donné, s'il existe plusieurs opérations POUSSER possibles, choisir en priorité POUSSER vers t et, si cela n'est pas possible, choisir le sommet vers lequel on pousse par ordre alphabétique.

Bien que ce résultat soit ici admis, voici la complexité de l'algorithme pousser-réétiqueter, à comparer avec celle de l'algorithme d'Edmonds-Karp [13]. [Complexité de l'algorithme Pousser-Réétiqueter] Etant donné un flot $G = (V,E,c,s,t)$, le temps d'exécution total de l'algorithme pousser-réétiqueter se comporte comme $O(|V|^3)$.

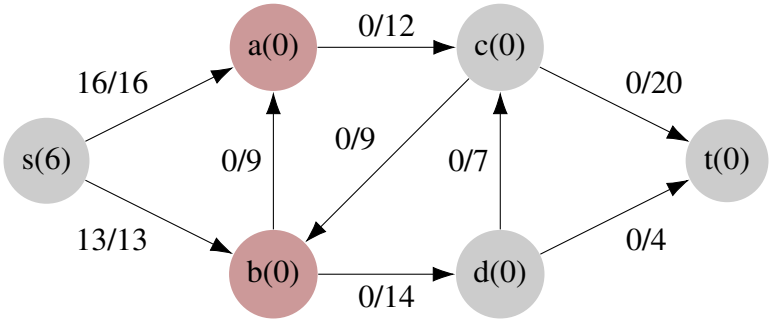
5.4.3 Exemple complet

Voyons comment l'algorithme pousser-réétiqueter permet de trouver un flot maximal pour le réseau de flot suivant :

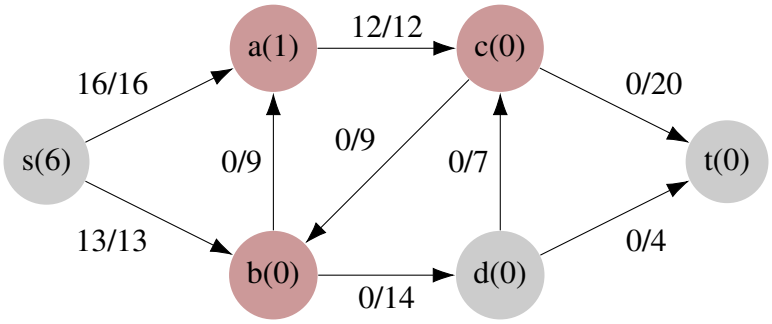


13. Pour la preuve, voir Roughgarden's Lecture Note #3 pp. 10–15.

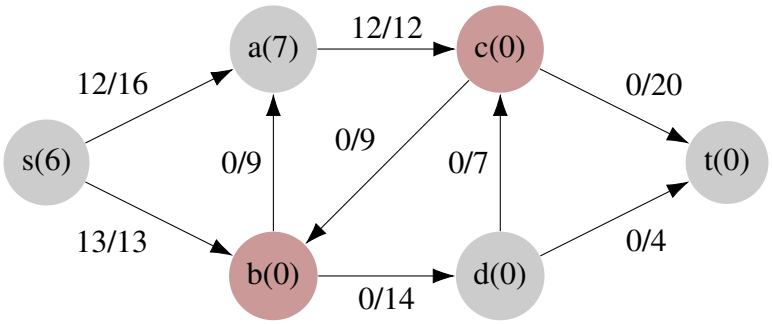
A chaque étape du processus, nous écrirons entre parenthèses la hauteur de chaque sommet et nous colorierons en rouge les sommets ayant un excédent non nul. L'opération d'initialisation donne :



Ici, a and b sont les sommets posant problème. D'après les priorités choisies, l'algorithme doit traiter d'abord le sommet a. Puisque tous ses voisins sont à la même hauteur, il est impossible de pousser de l'eau vers le bas à partir de a. Ainsi, la seule option est de réétiqueter a à une hauteur de 1. Puis l'on pousse 12 de a vers c :

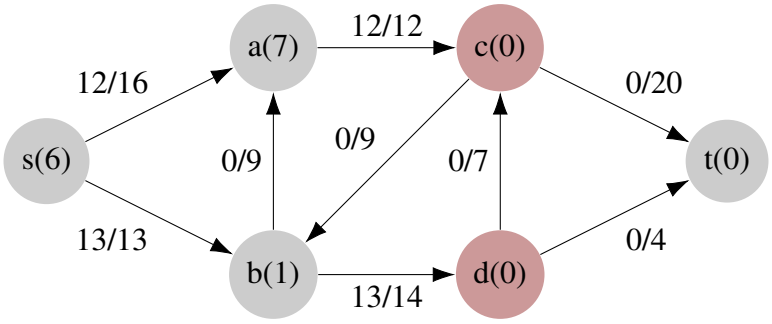


Malgré ces opérations, a reste excédentaire ($e(a) = 4$). Le seul voisin de a dans le graphe résiduel est maintenant s. La seule option pour régler l'excédent en a est donc d'élever a à une hauteur de 7 pour ainsi pouvoir pousser 4 de a vers s. Après ces opérations, on a :

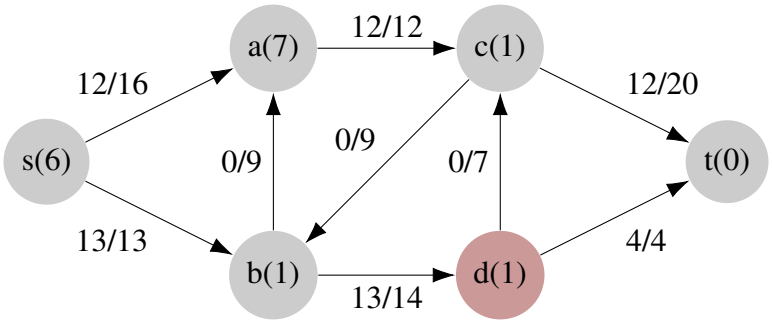


Maintenant, d'après les priorités choisies, l'algorithme s'attaque au sommet b. Ses trois voisins sont a, d et s et aucun d'eux est plus bas que b. Il faut donc réetiqueter b de sorte qu'il soit plus haut qu'au moins un de ses voisins. Puisque $h(d) = 0$, il suffit de prendre $h(b) = 1$ ce qui permet de pousser 13 de

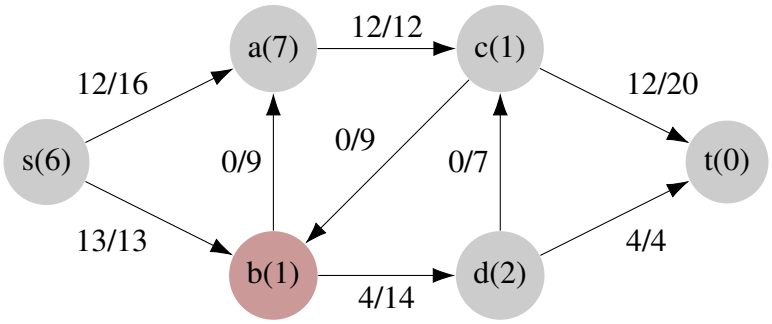
b vers d :



Ensuite, on réétiquette c à 1, on pousse 12 de c vers t, puis on réétiquette d à 1 et on pousse 4 de d vers t :

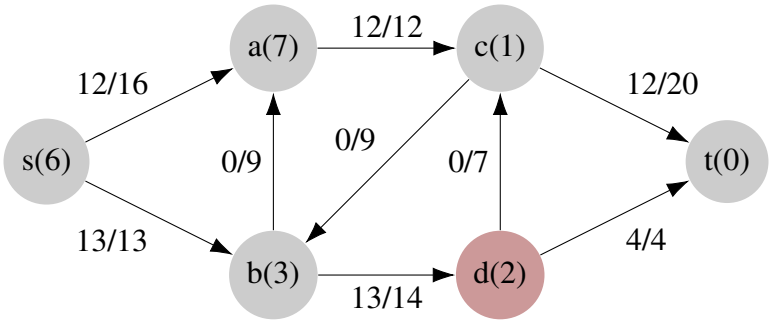


Malgré les opérations, le sommet d reste excédentaire. Ses voisins sont b et c et sont à la même hauteur que d. On doit donc réétiqueter d à 2, ce qui permet maintenant de pousser de l'eau vers b ou vers c. Selon les priorités choisies pour l'algorithme, il faut choisir par ordre alphabétique et donc pousser vers b autant que possible. On se retrouve donc dans la situation suivante :

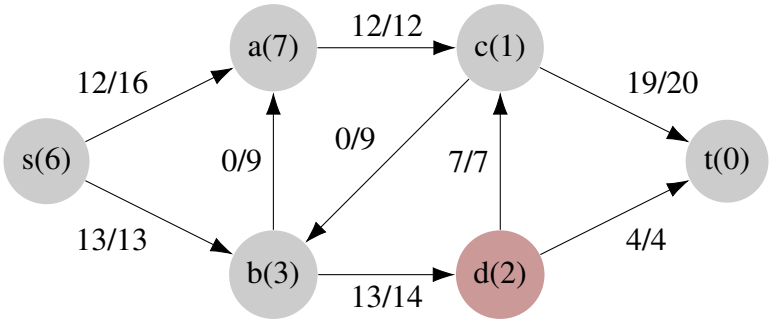


La dernière opération semble contre-productive puisque maintenant b est devenu à nouveau excédentaire. Pourtant, l'algorithme continue...Pour traiter le problème en b, on n'a pas d'autre choix que de

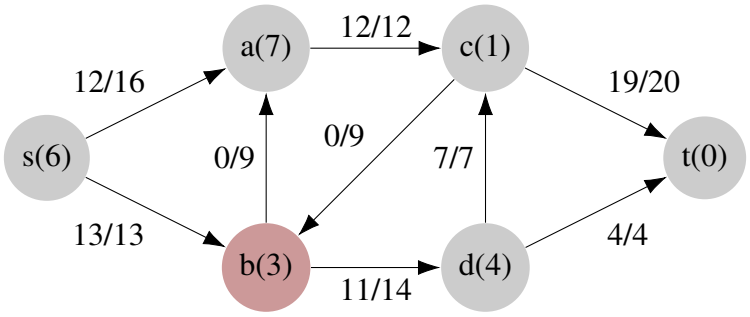
réétiqueter b à 3, ce qui permet de pousser (à nouveau !) 9 de b vers d :



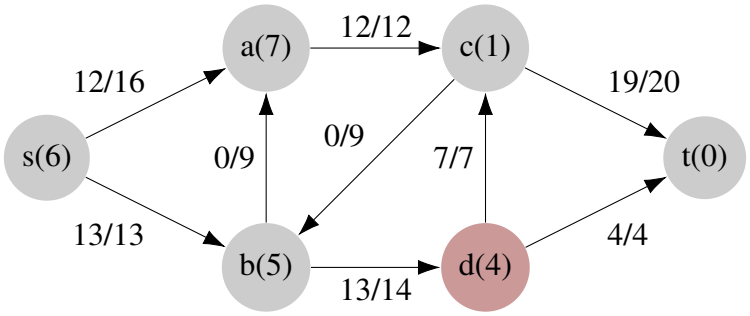
On semble être revenu à la situation de deux étapes auparavant, sauf que maintenant c se trouve plus bas que d et l'on peut donc pousser 7 de d vers c puis de c vers t :



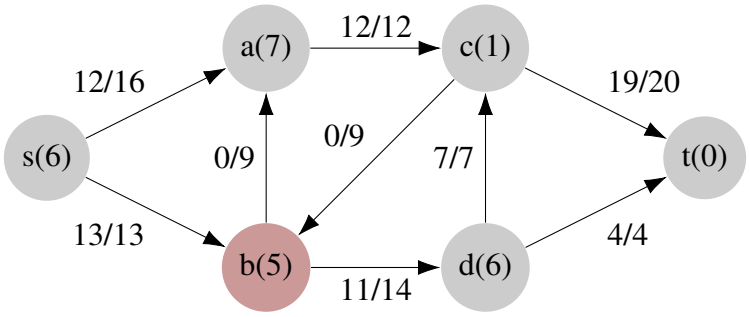
Mais le problème en d persiste...L'algorithme réétiquette d à 4 et pousse 2 de d vers b :



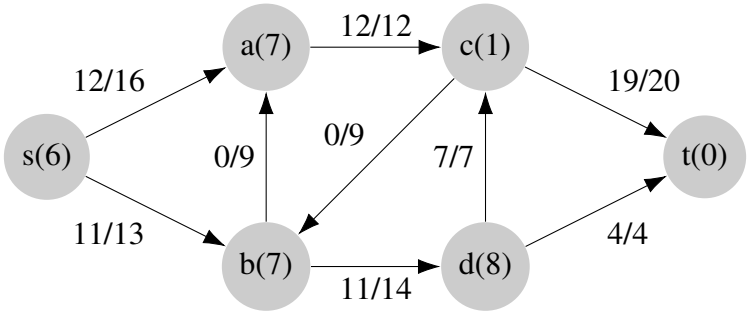
Puis il réétiquette b à 5 et pousse 2 de b vers d ...



… réétiquette d à 6 et pousse 2 de d vers b …



… réétiquette b à 7, pousse 2 de b vers d, réétiquette d à 8, pousse 2 de d vers b et (enfin !) pousse 2 de b vers s, ce qui achève l'algorithme. Le résultat final est donc :



La valeur maximale du flot pour ce réseau est donc 23.

5.5 Flot à coût minimal

5.5.1 Le problème

Une variante similaire au problème du flot maximal est le problème de flot à coût minimal. Ici, on considère un réseau de flot (V,E,c,s,t) muni d'une donnée additionnelle : le coût de transport d'une unité le long de chaque arête du graphe. Cette donnée peut être simplement formalisée par une fonction $d : E \to \mathbb{R}^+$ qui associe un poids positif à chaque arête.

Définition 5.8 (Réseau de flot pondéré)
Un *réseau de flot pondéré* est un sextuplet (V,E,c,d,s,t) où :

i) (V,E,c,s,t) est un réseau de flot.

ii) $d : E \longrightarrow \mathbb{R}^+ \cup \{+\infty\}$ est une application qui à chaque arête du réseau lui associe un nombre positif appelé le *coût unitaire le long de l'arête*.

Etant donné un flot f sur un réseau de flot pondéré, on peut donc définir le **coût du flot** par la quantité :

$$d(f) = \sum_{(u,v)\in E} d(u,v)f(u,v).$$

Dans ce cadre, il est naturel de considérer le problème suivant :

Problème du flot à coût minimal : étant donné un réseau de flot pondéré (V,E,c,d,s,t) et une nombre positif F, trouver un flot f de valeur totale $|f| = F$ et de coût minimal parmi tous les flots possibles de valeur totale F. [14]

5.5.2 La solution

L'idée la plus simple pour résoudre ce problème est de modifier la méthode de Ford-Fulkerson de sorte qu'elle prenne en compte les poids des arêtes. Ainsi, au lieu d'utiliser l'algorithme du parcours en largeur pour trouver les chaînes améliorantes (Edmonds-Karp), on va maintenant utiliser un algorithme de plus court chemin dans un graphe pondéré :

Algorithme pour résoudre Flot à coût minimal	=	Méthode de Ford-Fulkerson	+	Algorithme du plus court chemin (Bellman-Ford)

14. Bien sûr, pour que le problème puisse avoir une solution, il faut que la valeur F choisie au départ puisse être atteinte et donc soit plus petite (ou égale) que la valeur maximale de flot du réseau (V,E,c,s,t).

Pour être précis, il faut adapter la méthode de Ford-Fulkerson au nouveau cadre des graphes pondérés. En particulier, il faut modifier la construction du graphe résiduel :

Définition 5.9 (Réseau résiduel pondéré)
Etant donné un réseau de flot pondéré (V,E,c,d,s,t) et un flot $f : E \longrightarrow \mathbb{R}^+$, le réseau résiduel pondéré associé est le réseau de flot pondéré (V,E_f,c_f,d_f,s,t) défini par :

i) si $(u,v) \in E$ alors (u,v) et (v,u) sont tous les deux des éléments de E_f.

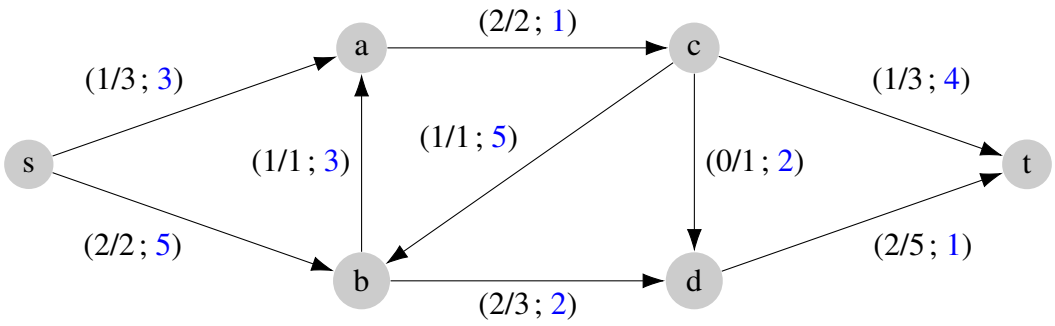
ii) $\forall (u,v) \in E_f,\ c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{si } (u,v) \in E \\ f(v,u) & \text{si } (v,u) \in E \end{cases}$.

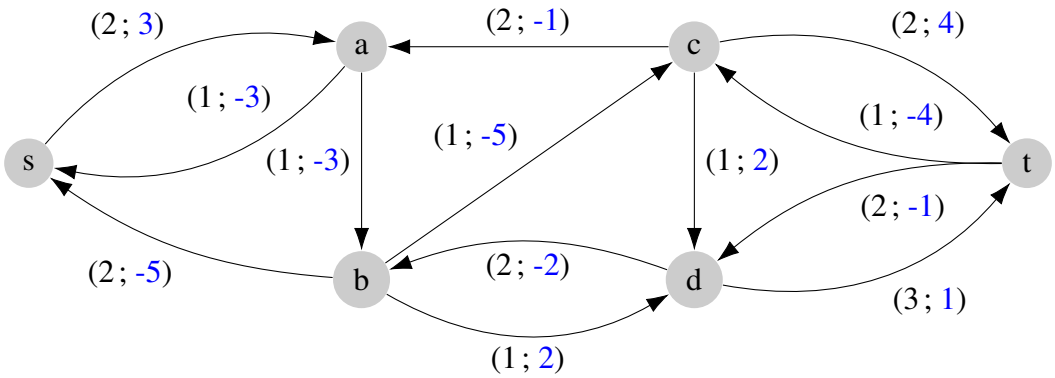
$c_f(u,v)$ est appelée la **capacité résiduelle** de l'arête (u,v).

iii) $\forall (u,v) \in E_f,\ d_f(u,v) = \begin{cases} +\infty & \text{si } c_f(u,v) = 0 \\ d(u,v) & \text{si } c_f(u,v) \neq 0 \text{ et } (u,v) \in E \\ -d(v,u) & \text{si } c_f(u,v) \neq 0 \text{ et } (v,u) \in E \end{cases}$.

$d_f(u,v)$ est appelée le **coût résiduel** le long de l'arête (u,v).

Exemple : On reprend l'exemple de la page 12 en rajoutant pour chaque arête un coût (en bleu). D'abord on représente le réseau de flot avec le flot, puis, en bas, le réseau pondéré résiduel. Pour simplifier, toutes les arêtes de capacité nulle (et donc de coût infini) ne sont pas représentées.





De plus, pour que la méthode de Ford-Fulkerson soit mise en oeuvre de façon algorithmique, il faut préciser l'algorithme permettant de trouver le plus court chemin entre la source s et le puits t dans le graphe résiduel pondéré. Puisque (V, E_f, d_f) est un graphe pondéré avec des poids négatifs, l'algorithme de Dijkstra ne fonctionne pas et il faut utiliser l'algorithme de Bellman-Ford.

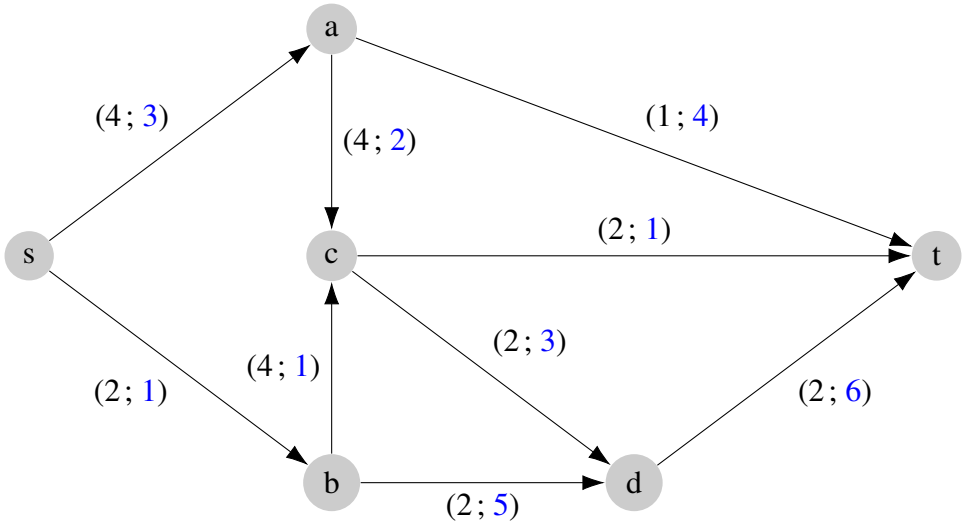
Rappelons de façon très succincte le fonctionnement de l'algorithme de Bellman-Ford : étant donné le graphe pondéré (V, E_f, d_f) avec la source s et le puits t

- ⋆ on initialise en mettant tous les sommets à l'infini et le sommet source à 0,
- ⋆ à l'itération k, pour tout sommet u autre que s, on choisit comme distance en cours, notée $\pi^k(u)$, le minimum entre sa distance à l'itération précédente $\pi^{k-1}(u)$ et la distance obtenue en passant par tous ses prédécesseurs dont la distance a changé à l'itération précédente.
- ⋆ l'algorithme s'arrête dès qu'il n'y a aucun changement à une itération donnée (ce qui arrive au bout d'au plus $|V|$ itérations) [15].

5.5.3 Exemple explicite

On considère le réseau de flot pondéré ci-dessous. Pour chaque arête, les capacités sont marquées en noir et les coûts unitaires en bleu. L'objectif est de trouver un flot de valeur 4, de coût minimal.

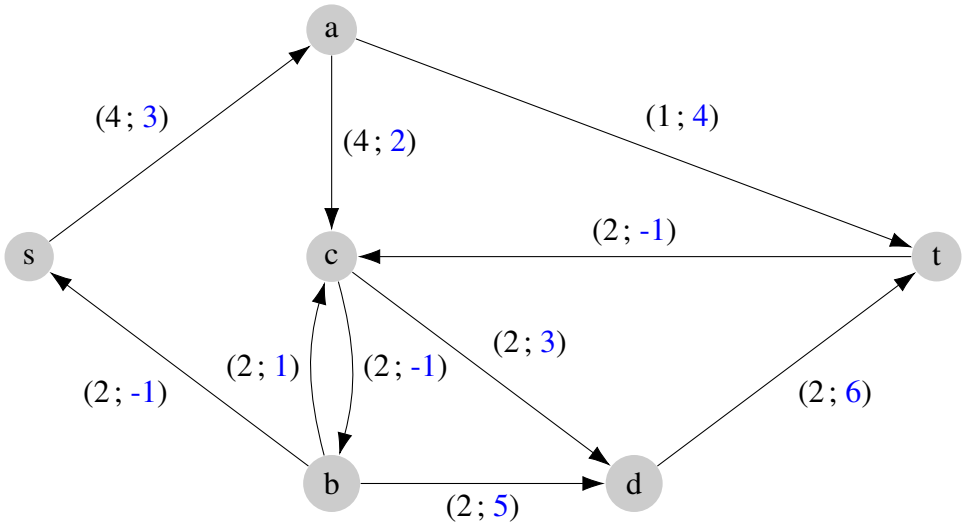
15. Dans notre cas, nous n'avons pas à nous préoccuper de l'existence de circuits absorbants puisque la construction particulière du graphe résiduel pondéré assure qu'il n'en existe pas.



L'algorithme commence par initialiser le flot à 0. Le graphe résiduel pondéré est donc le même que le graphe de départ. Ensuite, on applique l'algorithme de Bellman-Ford à ce graphe pondéré pour trouver le plus court chemin entre s et t (**attention à ne pas confondre les capacités et les poids !**) :

k	**s**	**a**	**b**	**c**	**d**	**t**
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	3_s	1_s	$+\infty$	$+\infty$	$+\infty$
2	0	3_s	1_s	2_b	6_b	7_a
3	0	3_s	1_s	2_b	5_c	3_c
4	0	3_s	1_s	2_b	5_c	3_c

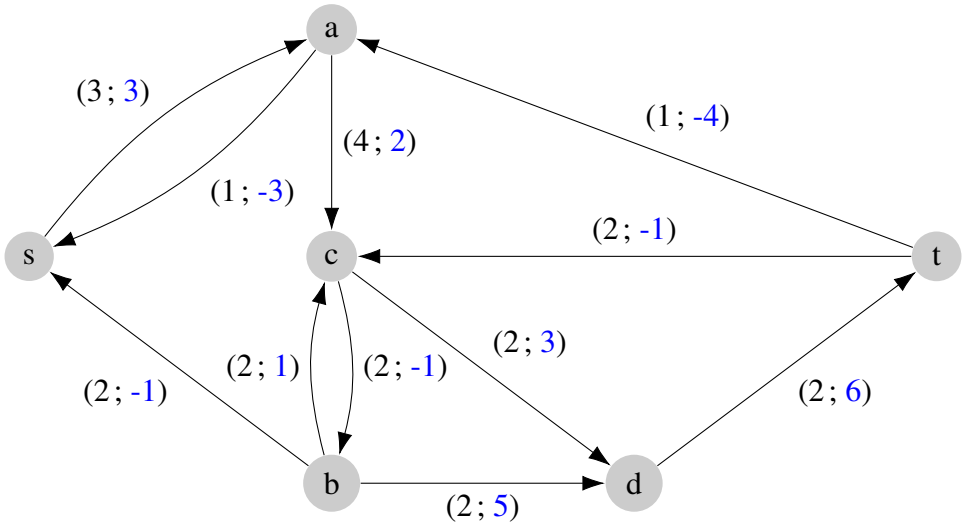
La première chaîne améliorante est donc **sbct de flot 2**. Le flot total est $2 < 4$ donc on continue. Le graphe résiduel pondéré associé à cette première modification est :



On applique Bellman-Ford à ce nouveau graphe :

k	s	a	b	c	d	t
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	3_s	$+\infty$	$+\infty$	$+\infty$	$+\infty$
2	0	3_s	$+\infty$	5_a	$+\infty$	7_a
3	0	3_s	4_c	5_a	8_c	7_a
4	0	3_s	4_c	5_a	8_c	7_a

D'où la deuxième chaîne améliorante : **sat de flot 1**. Le flot total est maintenant $3 < 4$, donc on continue. Le graphe résiduel pondéré devient :

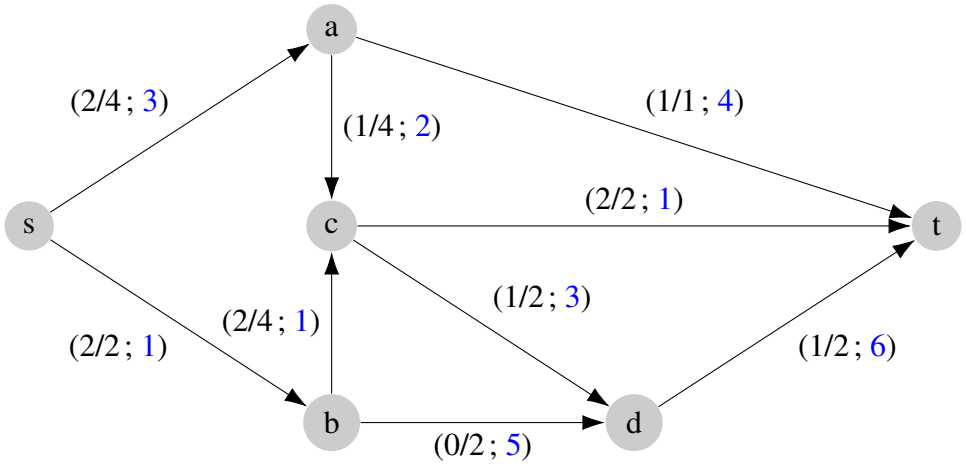


et l'algorithme de Bellman-Ford donne :

k	s	a	b	c	d	t
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	3_s	$+\infty$	$+\infty$	$+\infty$	$+\infty$
2	0	3_s	$+\infty$	5_a	$+\infty$	$+\infty$
3	0	3_s	4_c	5_a	8_c	$+\infty$
4	0	3_s	4_c	5_a	8_c	14_d
5	0	3_s	4_c	5_a	8_c	14_d

On trouve la chaîne améliorante sacdt où l'on peut faire passer un flot de 2. Mais comme on a déjà un flot total de 3 et que l'on cherche à avoir un flot total de 4, on décide de faire passer seulement un

flot de 1 par sacdt. On a donc trouvé comme solution au problème le flot suivant :



pour lequel le coût est

$$d(f) = 2\times 3+2\times 1+2\times 1+1\times 2+1\times 4+2\times 1+1\times 3+1\times 6 = 27.$$