PARIS PANTHÉON-ASSAS UNIVERSITÉ

Projet de Recherche opérationnelle

Département de Mathématiques Efrei Paris

Année 2024/2025 S6

I1

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 travailler sur son fonctionnement en regardant **les deux vidéos** sur Moodle puis 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 les vidéos visionnées et le cours en annexe 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 v_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 :

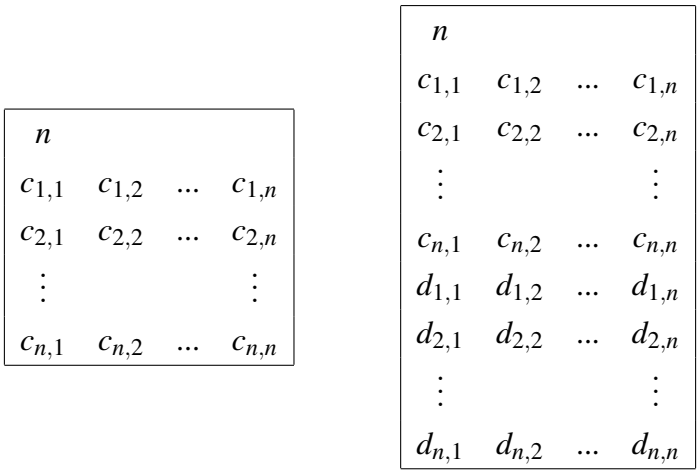


FIGURE 2.1 – Les tables demandées en .txt
Les problèmes de flot max étant à gauche et les problèmes de flot à coût min étant à droite.

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"

Exemple de trace pour la proposition 1 et FF :

⋆ Affichage de la table des capacités : .

8							
0	9	5	7	0	0	0	0
0	0	6	0	6	0	0	0
0	0	0	0	0	7	0	0
0	0	0	0	0	2	8	0
0	0	3	0	0	0	0	8
0	0	0	0	0	0	0	10
0	0	0	0	0	3	0	5
0	0	0	0	0	0	0	0

Le graphe résiduel initial est le graphe de départ.

⋆ Itération 1 :

Le parcours en largeur :

s

abc ; $\Pi(a) = s$; $\Pi(b) = s$; $\Pi(c) = s$

bcd ; $\Pi(d) = a$

cde ; $\Pi(e) = b$

def ; $\Pi(f) = c$

eft ; $\Pi(t) = d$

Détection d'une chaîne améliorante : *sadt* de flot 6.

Modifications sur le graphe résiduel :

	s	a	b	c	d	e	f	t
s	0	3	5	7	0	0	0	0
a	6	0	6	0	0	0	0	0
b	0	0	0	0	0	7	0	0
c	0	0	0	0	0	2	8	0
d	0	6	3	0	0	0	0	2
e	0	0	0	0	0	0	0	10
f	0	0	0	0	0	3	0	5
t	0	0	0	0	6	0	0	0

⋆ Itérations 1, 2, .. :

⋆ Affichage du flot max :

	s	a	b	c	d	e	f	t
s	0	8/9	5/5	7/7	0	0	0	0
a	0	0	2/6	0	6/6	0	0	0
b	0	0	0	0	0	7/7	0	0
c	0	0	0	0	0	2/2	5/8	0
d	0	0	0/3	0	0	0	0	6/8
e	0	0	0	0	0	0	0	9/10
f	0	0	0	0	0	0/3	0	5/5
t	0	0	0	0	0	0	0	0

Valeur du flot max = 20

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 3 mai 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 4 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.

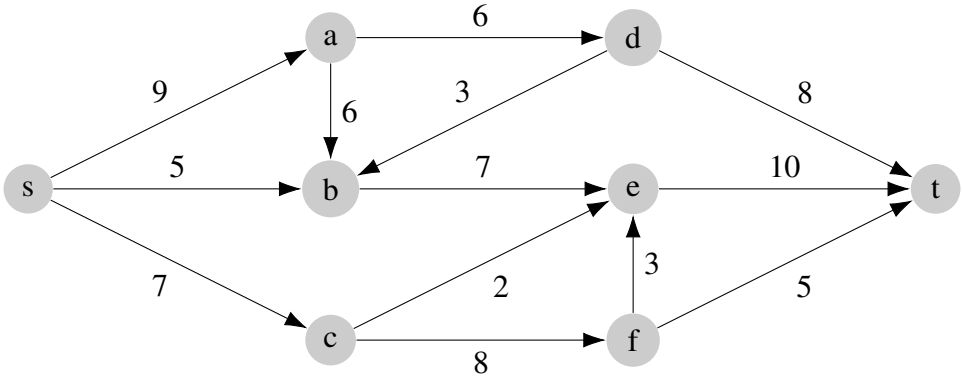
A l'issue de l'oral, des questions vous seront posées pendant 20 min : chaque étudiant sera interrogé individuellement sur le projet ou sur un point du cours. Le code devra être absolument maîtrisé par tous les membres de l'équipe.

Bon courage pour la rédaction de ce projet,

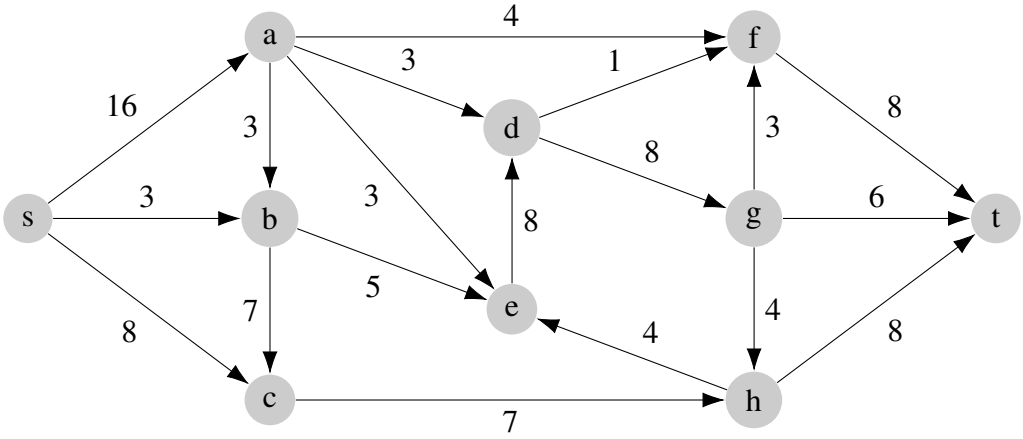
L'équipe des enseignants de Recherche Opérationnelle.

ANNEXE 1 : les dix propositions de flot

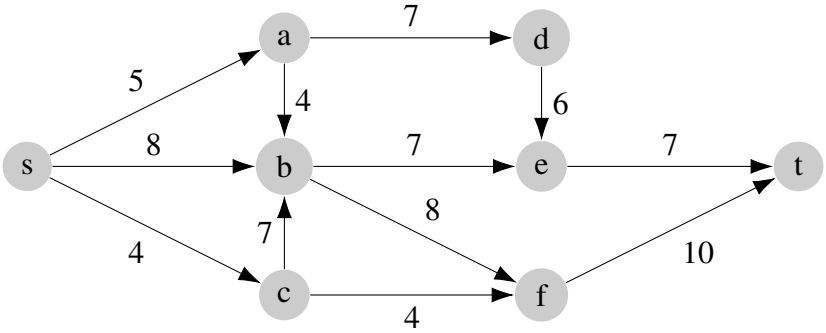
proposition 1



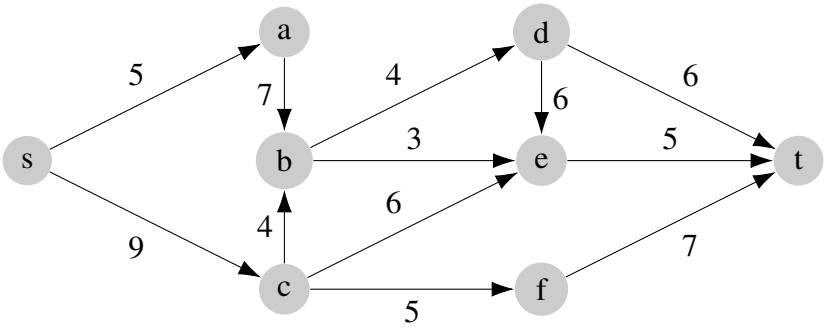
proposition 2



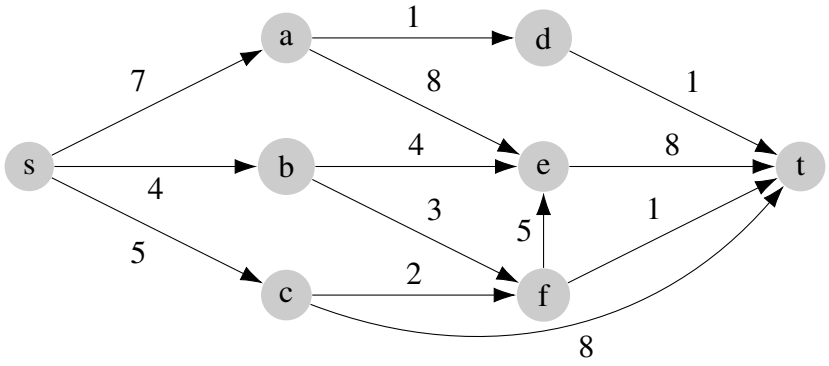
proposition 3



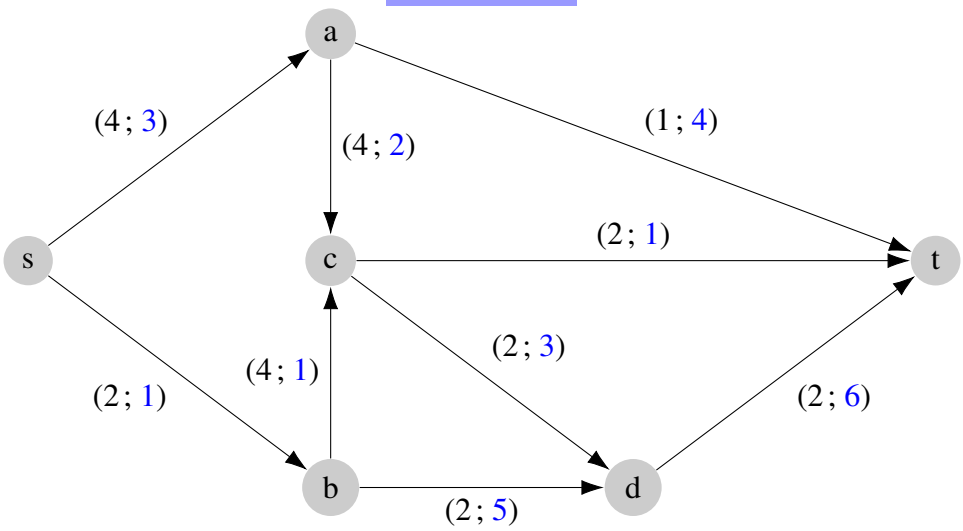
proposition 4



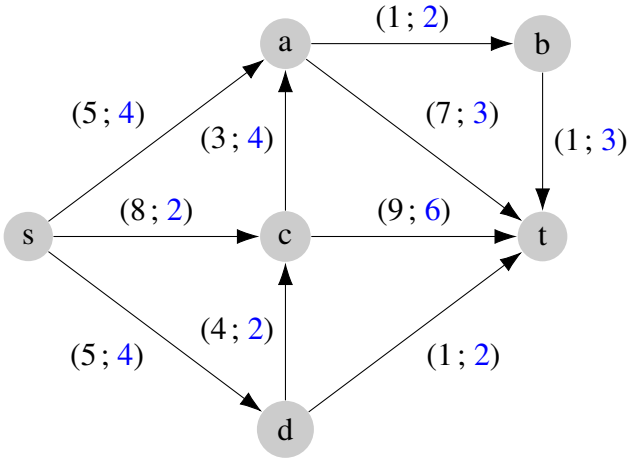
proposition 5



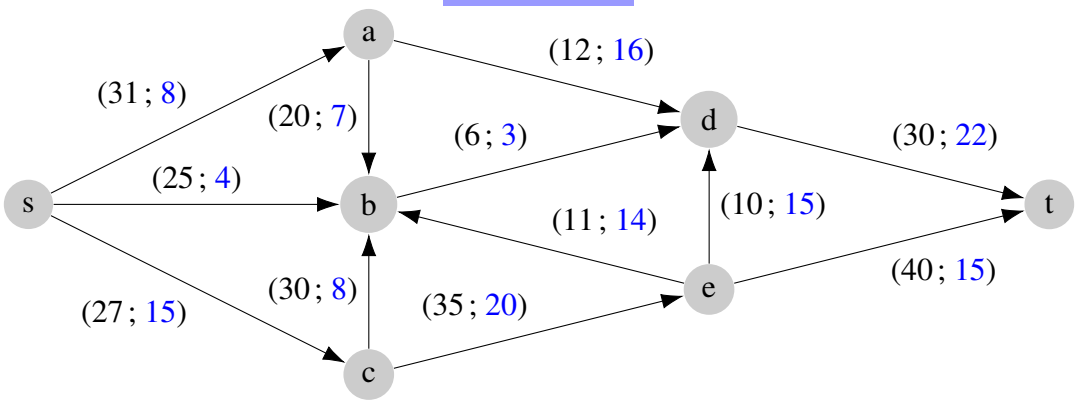
proposition 6



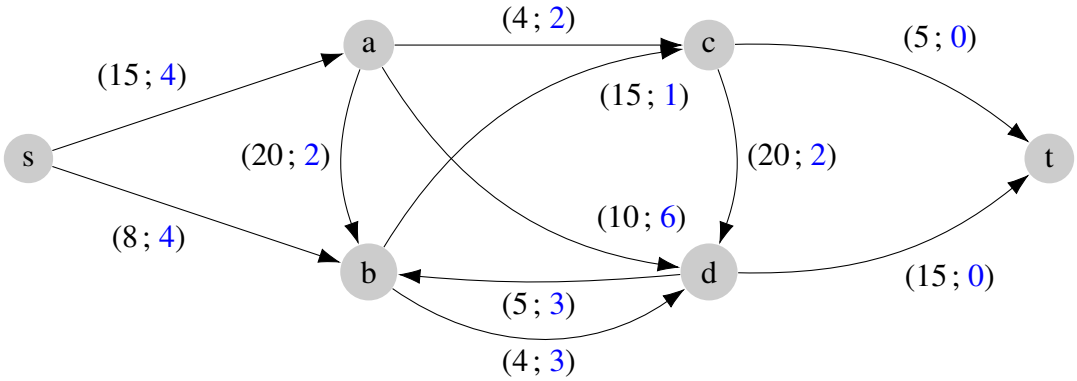
proposition 7

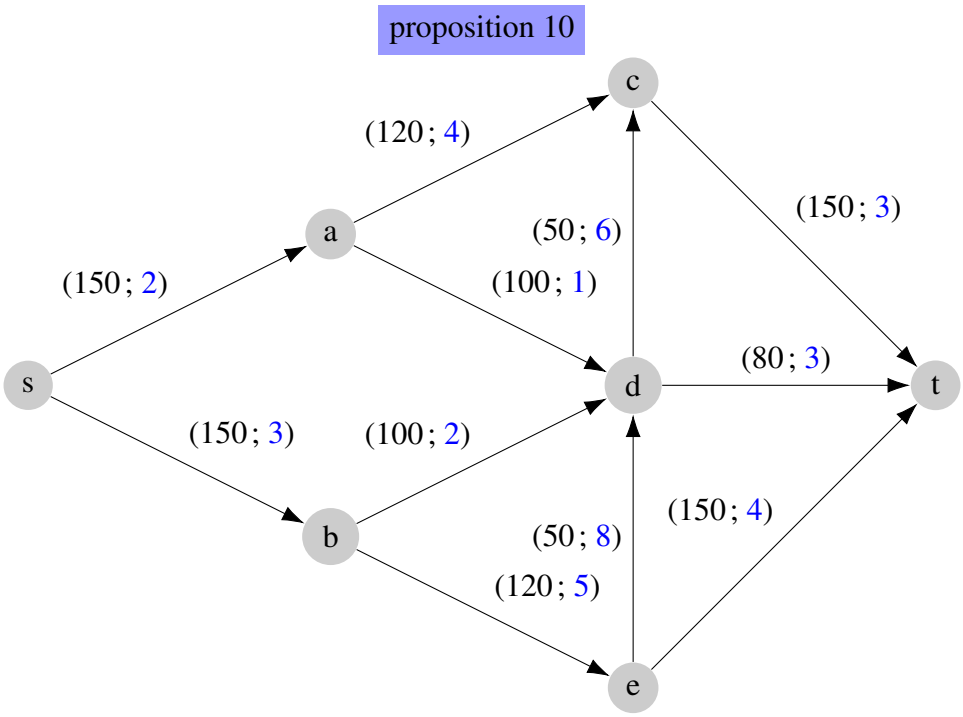


proposition 8



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