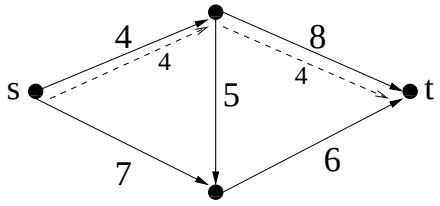
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
Section d'Informatique et de Systèmes de Communication

Corrigé de la série 10 30 Novembre 2007

1. Flux

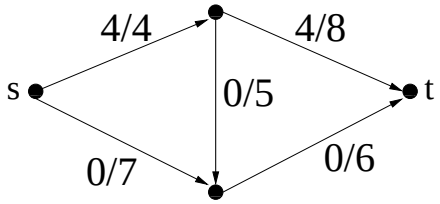
a) Non, ce n'est pas un flux puisqu'il y a un sommet intérieur pour lequel le bilan est non-nul : Dans le sommet en haut ce qui rentre est 2 alors qu'il sort 5 de ce sommet.

b) On utilise l'algorithme maxflow-mincut du cours. On commence par trouver un chemin quelconque de s à t :

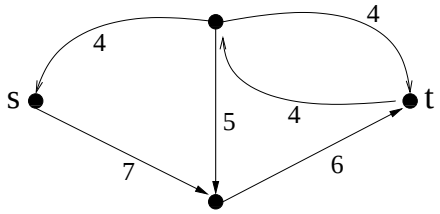


La valeur de ce chemin est 4 car on ne peut pas faire passer plus de 4 dans la première arête.

On obtient donc le flux suivant :



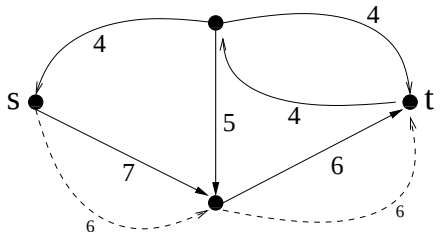
On construit à présent le graphe résiduel correspondant. On rappelle que le graphe résiduel indique comment peut être modifié un flux :



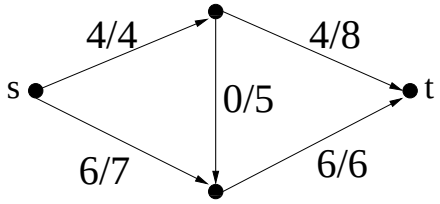
En effet, la première arête quittant s (celle de capacité 4) à une valeur égale à sa capacité, sa valeur ne peut donc pas être augmentée (donc pas de flèche dans le même sens). Par contre, la valeur peut être diminuée à 0 (donc diminuée de 4), on a donc une flèche de capacité 4 dans le sens inverse.

De même, la première arête arrivant vers t (celle de capacité 8) peut soit voir sa valeur augmentée de 4 (pour arriver à sa capacité) ou soit diminuée de 4 (pour revenir à 0), on a donc une flèche de capacité 4 dans chaque sens. etc.

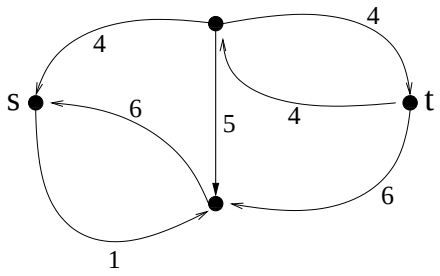
On choisit un chemin de s à t dans ce graphe résiduel :



On obtient donc le flux suivant :



On construit maintenant le graphe résiduel correspondant pour voir si ce flux peut encore être amélioré :



On voit qu'il n'y a aucun chemin de s à t, notre flux est donc maximal.

Sa valeur est égale à

$$|f| = \sum_{e \in E(s, V \setminus \{s\})} f(e) - \sum_{e \in E(V \setminus \{s\}, s)} f(e).$$

Donc dans ce cas

$$(4 + 6) \quad - \quad 0 = 10$$

c) On voit dans la question précédante que l'arrête de capacité 5 n'est pas utilisée dans le flux maximal, donc si elle est enlevée le flux maximal restera le même.

d) Dans ce réseau très simple, si la capacité de la première arête quittant s est augmentée de 4 à 10, alors le flux pourra être augmenté de 10 à 14.

De manière générale, il faut choisir une arête adéquate sur le min cut du graphe, ce qui permet de considérablement simplifier le problème pour les réseaux plus compliqués.

2. Augmenter les poids

a) Par la formule d'Euler on sait que tous les arbres couvrants ont $|V| - 1$ arêtes. Donc si T_1 est un arbre couvrant de poids $w(T_1)$, l'augmentation des poids sur les arêtes par une constante c fait que le nouveau poids de T_1 sera $w(T_1) + (|V| - 1) \cdot c$.

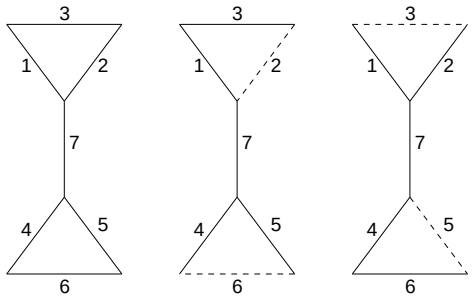
Soit T_2 un autre arbre couvrant. Comme on a $w(T_1) + (|V| - 1) \cdot c \leq w(T_2) + (|V| - 1) \cdot c$ si et seulement si $w(T_1) \leq w(T_2)$, l'ordre partiel défini par le poids est préservé et donc en particulier, les éléments minimaux restent les mêmes.

b) Oui. Si on diminue le poids de l'une des arêtes de T d'une constante c, son nouveau poids devient $w(T) - c$. Pour tout autre arbre couvrant T' le nouveau poids sera soit $w(T') - c$ (si l'arête dont on a diminué le poids appartient à T'), soit $w(T')$ (si elle n'appartient pas à T'). Dans les deux cas le nouveau poids de T est plus petit que le nouveau poids de T', car $w(T) \leq w(T')$.

3. Deuxième meilleur arbre couvrant

a) Supposons qu'on a deux arbres couvrants minimaux T_1 et T_2. Considerer les coûts des arêtes de T_1 et T_2 de manière croissante, soit e la première arête qui apparait dans un arbre mais pas dans l'autre. Si $e \in T_1$ on ajoute e dans T_2. Il y aura un cycle dans T_2, et dans ce cycle il existe une arête f telle que $w(f) > w(e)$ (sinon toutes les arête dans le cycle sont dans T_1 mais T_1 est un arbre). Si on enlève f de T_2 on aura un arbre T_3 de coût plus petit que le coût de T_2 ce qui est une contradiction puisque T_2 est un arbre couvrant minimum.

On montre que le deuxième meilleur arbre couvrant n'est pas unique à l'aide d'un exemple.



b) Nous montrons qu'un arbre qui n'est pas optimal peut toujours être transformé en un meilleur arbre en ajoutant une arête et enlevant une autre arête.

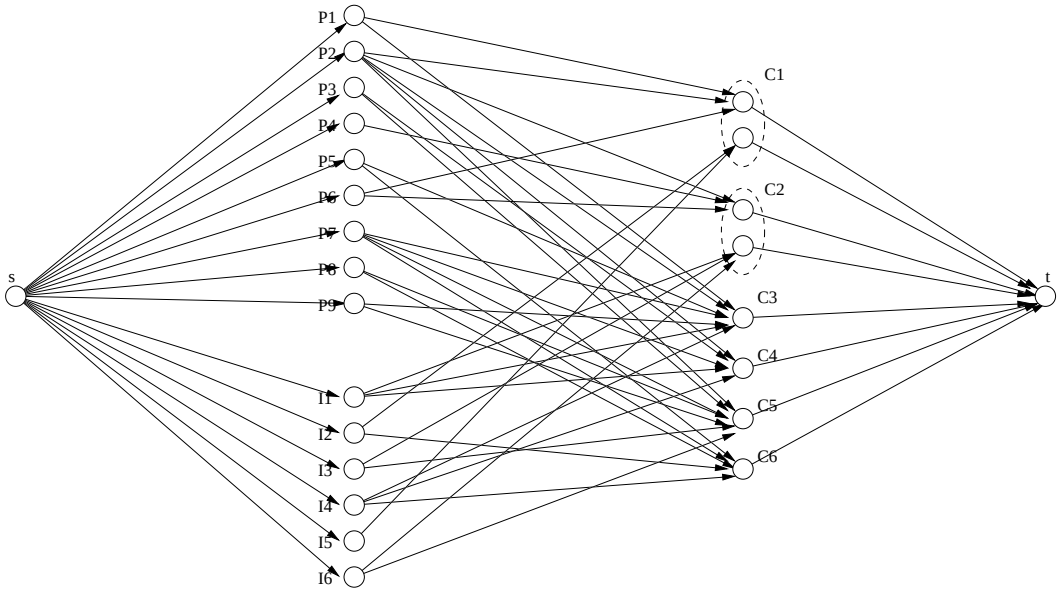
Comme n'importe quelle amélioration du deuxième meilleur arbre résulte en un arbre optimal, ceci permet de conclure.

Soit T un arbre couvrant non-optimal quelconque, et T_{opt} l'arbre optimal obtenu en appliquant l'algorithme de Kruskal. Soit $E = \{e_1, \ldots, e_m\}$ avec $e_i < e_{i+1}$ si $1 \leq i \leq m - 1$. Il faut montrer qu'on peut améliorer T en ajoutant une arête et enlevant une autre. Soit i le plus petit indice tel que soit $e_i \in T_{\text{opt}} \setminus T$ ou $e_i \in T \setminus T_{\text{opt}}$. Alors $e_i \in T \setminus T_{\text{opt}}$: si ce n'était pas le cas, par construction de T_{opt}, ajouter e_i à T_{opt} créera un cycle se constituant d'arêtes d'indice $\leq i$ dans T_{opt}. Comme les mêmes arêtes sont dans T, le même cycle serait aussi dans T, mais T est un arbre. Donc $e_i \in T_{\text{opt}} \setminus T$. En ajoutant e_i dans T, on obtient un cycle contenant au moins une arête j d'indice $> i$. On enlève cette arête, ce qui résulte en un arbre de poids $w(T) - w(e_j) + w(e_i) < w(T)$, comme $j > i$.

4. Attribution optimale

a) Le problème ressemble à un problème de bipartite matching, et donc on veut appliquer l'algorithme max-flow-min-cut pour le résoudre. La seule subtilité est celle des chambres doubles. On peut la résoudre en travaillant plutôt sur des "lits" qu'on met dans les chambres. Dans les chambres simples on met un lit simple, dans les chambres doubles on en met deux : un pour un informaticien et un pour un physicien.

b) On obtient le graphe suivant :



5. Connexité d'un graphe

a) On peut par exemple utiliser UnionFind. Dans l'algorithme ci-dessous t est un tableau associatif. On sait qu'un tableau associatif peut être réalisé de telle façon que les opérations "ajouter une clé" ainsi que "vérifier si une clé est dedans" sont $O(\log(n))$, si n objets sont dans le tableau. Par exemple, on peut utiliser un arbre AVL pour avoir cette garantie.

for $(x, y) \in E$ **do**
  Call Union(x, y)
$k \leftarrow 0$
for $v \in V$ **do**
  $x \leftarrow$ Représentant de v dans UnionFind
  if $x \notin t$ **then**
    Ajouter x à t
    $k \leftarrow k + 1$

La première boucle utilise $O(|E| \log(|V|))$ opérations, car l'opération Union coûte au plus $\log(|V|)$.

La deuxième boucle coûte $O(|V| \log(|V|))$ parce que les opérations dans UnionFind ainsi que sur t sont les deux $O(\log(|V|))$.

b) Oui, on peut faire mieux. On peut d'abord lier tous les sommets par une liste doublement liée.

Ensuite on prend le premier sommet de la liste x et on fait un DFS sur le graphe commençant en x. On enlève chaque sommet de la liste liée pendant que DFS le visite. A la fin de DFS on a enlevé une composante connexe de la liste. On répète le procédé jusqu'à ce que la liste soit vide.

En procédant de cette manière on touche chaque sommet et chaque arête au plus une fois, et l'opération à effectuer est à chaque fois $O(1)$, ce qui donne $O(|V| + |E|)$.

c) On peut utiliser max-flow-min-cut. On munit chaque arête du graphe d'une capacité 1. Ensuite on laisse parcourir tous les couples de sommets (s, t) en exécutant à chaque fois l'algorithme max-flow-min-cut. Le cut trouvé donne le nombre minimal d'arêtes à enlever de sorte que s et t soient dans deux composantes distinctes du graphe. Comme on fait le test avec tous les couples, on doit de cette façon trouver le plus petit cut qui marchera.