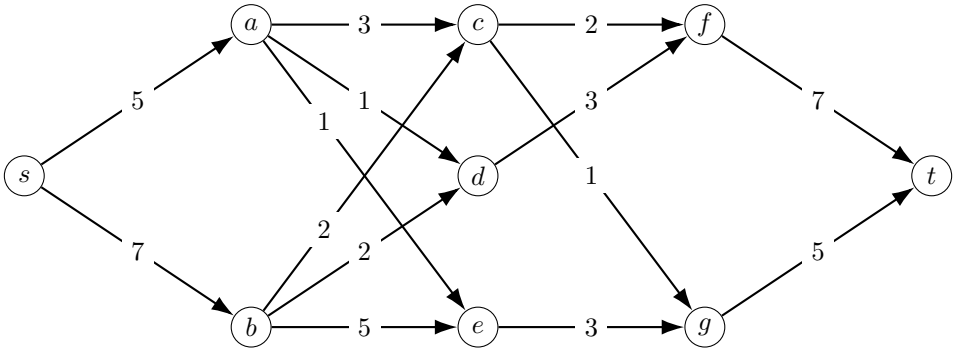
Partiel d'optimisation combinatoire corrigé
1h30

Les calculatrices, ordinateurs et documents de cours sont interdits.
Vous pouvez admettre le résultat d'une question pour passer à la suivante.

I Algorithme de Ford-Fulkerson (/4)

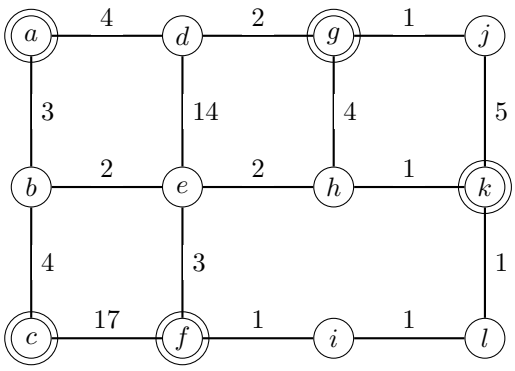
En utilisant l'algorithme de Ford-Fulkerson, donner un flot maximum de s à t et une coupe de capacité minimum sur le graphe ci-dessous. On détaillera chaque étape de l'algorithme.



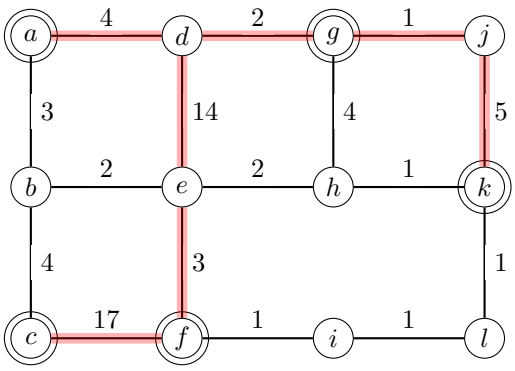
Solution : On trouve un flot max de valeur 9, et une coupe minimale $\{s, a, b, c, e\}$.

II Arbre de Steiner

Soit $G = (V, E)$ un graphe non-orienté pondéré par w et $R \subseteq V$ un sous-ensemble de sommets.
Un **arbre de Steiner** est un arbre T de G qui contient tous les sommets de R.
Remarque : T peut contenir des sommets qui ne sont pas dans R, mais tous les sommets de R doivent être dans T.
Le poids de T, noté $w(T)$, est la somme des poids des arêtes de T.
L'objectif est de trouver un arbre de Steiner de poids minimum.
Dans tout l'exercice, on utilise le graphe G_1 suivant comme exemple, avec $R = \{a, c, f, g, k\}$ (sommets doublement entourés) :



Un graphe G_1 non-orienté pondéré



Un arbre de Steiner T_1 (surligné)

À droite surligné, on a représenté un exemple d'arbre de Steiner T_1 de G_1.

1. (/1) T_1 est-il un arbre de Steiner de poids minimum ?

   Solution : Non car on peut remplacer les arêtes $\{c, f\}$ et $\{f, e\}$ par $\{c, b\}$ et $\{b, e\}$ pour obtenir un arbre de Steiner de poids strictement inférieur.

On peut montrer que trouver un arbre de Steiner de poids minimum est un problème NP-complet.
Dans la suite, on s'intéresse à une 2-approximation d'un arbre de Steiner de poids minimum.
La première étape consiste à construire un graphe complet (c'est à dire : contenant toutes les arêtes possibles) $\widetilde{G}$ dont l'ensemble des sommets est R. De plus, le poids d'une arête $\{u, v\}$ dans $\widetilde{G}$ est égal au poids d'un plus court chemin entre u et v dans G.

On rappelle l'algorithme de Bellman-Ford vu en cours :

Algorithme de Bellman-Ford

```
d[r] <- 0
Pour v ≠ r:
    Pour k = 0 à |V| − 2:
        d[v][k] <- ∞

Pour k = 0 à |V| − 2:
    Pour tout sommet v:
        Pour tout arc (u, v) entrant dans v:
            Si d[u][k] + w(u, v) < d[v][k + 1]:
                d[v][k + 1] ← d[u][k] + w(u, v)
```

2. (/2) Quel est le principe algorithmique utilisé par Bellman-Ford? Que permet-il d'obtenir ? Expliquer d'où vient la formule de récurrence `d[v][k + 1]` ← `d[u][k] + w(u, v)`.

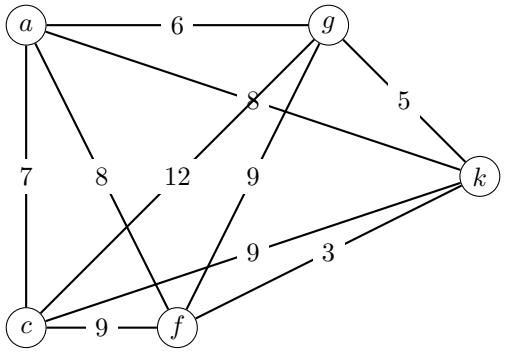
   Solution : Bellman-Ford est un algorithme de programmation dynamique et permet d'obtenir les distances depuis un sommet r vers tous les autres. Voir cours. (Nous l'avons vu sur des graphes orientés mais il marche aussi dans le cas non-orienté)

3. (/2) Expliquer comment utiliser l'algorithme de Bellman-Ford pour trouver toutes les distances dans un graphe, entre deux sommets quelconques. En donner la complexité en fonction du nombre de sommets n et du nombre d'arêtes p.

   Solution : Il suffit d'appliquer Bellman-Ford depuis chaque sommet r possible. Comme une application de Bellman-Ford est en $O(np)$, où $n = |V|$ et $p = |E|$, on obtient $O(n^2p)$ au total.

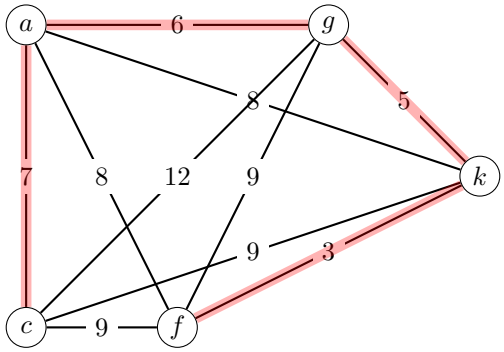
4. (/2) Dessiner $\widetilde{G_1}$, obtenu à partir de G_1.
   On écrira les poids sur chaque arête mais on ne demande pas d'expliquer comment vous avez trouvé ces poids.

   Solution :



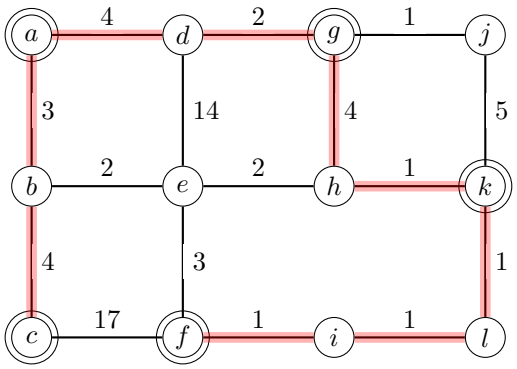
La deuxième partie consiste à trouver un arbre couvrant $\widetilde{T^*}$ de poids minimum de $\widetilde{G}$.

5. (/2.5) Donner un arbre couvrant $\widetilde{T_1^*}$ de poids minimum de $\widetilde{G_1}$, en utilisant, à votre choix, l'algorithme de Kruskal ou Prim.
   Vous précisérez quel algorithme vous avez utilisé ainsi que le détail de toutes les étapes pour obtenir un arbre couvrant de poids minimum.
   Solution : On obtient un arbre couvrant de poids 21

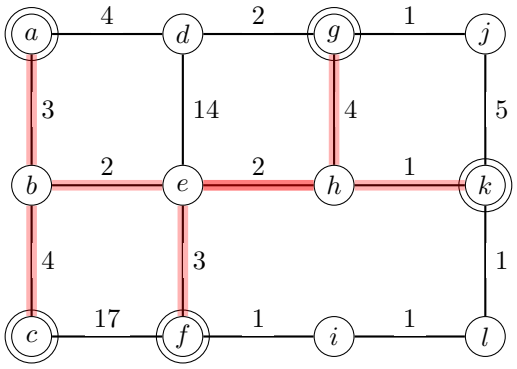


Enfin, on revient dans G en remplaçant chaque arête $\{u, v\}$ de $\widetilde{T^*}$ par le plus court chemin de u à v correspondant, dans G.

6. (/1) Dessiner G_1 en faisant apparaître les arêtes obtenues à partir des plus courts chemins de $\widetilde{T_1^*}$.
   Solution :



7. (/1) Si les arêtes obtenues à la question précédente ne forment pas un arbre, on supprime des arêtes dans des cycles jusqu'à obtenir un arbre. Est-ce que l'arbre T obtenu est un arbre de Steiner ? Est-il forcément de poids minimum ?
   Solution : On a bien obtenu un arbre de Steiner à la question précédente, de poids 21. Ce n'est pas optimal car l'arbre de Steiner suivant est de poids 19 :



Soit T_{opt} un arbre de Steiner de G. On veut montrer que T est une 2-approximation de T_{opt}, c'est à dire :

$$w(T) \leq 2w(T_{opt})$$

8. (/1) On traverse T_{opt} depuis un sommet fixé en faisant, par exemple, un parcours en profondeur.
   On obtient ainsi un cycle C de T_{opt} qui passe exactement 2 fois par chaque arête.
   Que vaut $w(C)$, en fonction de $w(T_{opt})$?

   Solution : $\boxed{w(C) = 2w(T_{opt})}$

9. (/1) On considère un cycle $\widetilde{C}$ dans $\widetilde{G}$ obtenu à partir de C en remplaçant les portions de chemins entre deux sommets r_1, r_2, de R par l'arête $\{r_1, r_2\}$. Par exemple, si C utilise les sommets $r_1, v_1, ..., v_k, r_2$ avec $v_1, ..., v_k \notin R$, $\widetilde{C}$ utilisera seulement l'arête de r_1 à r_2.
   Montrer que $w(\widetilde{C}) \leq w(C)$.

Solution : On remplace un chemin de r_1 à r_2 par une arête dont le poids est celui d'un plus court chemin de r_1 à r_2. Donc on n'augmente pas le poids.

10. (/1.5) En déduire que $w(T) \leq 2w(T_{opt})$.

Solution : $\widetilde{C}$ est connexe et contient tous les sommets de $\widetilde{G}$. Comme $\widetilde{T^*}$ est un arbre couvrant de poids minimum de $\widetilde{G}$, $w(\widetilde{T^*}) \leq w(\widetilde{C})$. Donc :

$$\boxed{w(T) \leq w(\widetilde{T^*}) \leq w(\widetilde{C}) \leq w(C) = 2w(T_{opt})}$$

11. (/1.5) Quelle est la complexité totale de cette méthode pour trouver une 2-approximation de l'arbre de Steiner de poids minimum ?

Solution : Soit $n = |V|$, $p = |E|$.
La recherche de tous les plus courts chemins demande $\mathrm{O}(n^2 p)$ avec Bellman-Ford ($\mathrm{O}(n^3)$ si on utilise Floyd-Warshall).
L'algorithme de Kruskal demande $\mathrm{O}(p \log(p))$ (complexité pour faire un tri des arêtes).
Revenir dans G demande seulement une complexité linéaire.
Au total la somme de ces complexité est $\boxed{\mathrm{O}(n^2 p)}$ (c'est le terme dominant).