L3, année 2010-11
Algorithmique et Programmation
Partiel du 6 Décembre 2010
Notes de cours autorisées, à l'exclusion de tout autre document.
Les algorithmes seront décrits informellement avant d'être écrit en pseudo-code.

1 Ensemble Indépendent Maximum

Soit $G = (S, A)$ un graphe non-orienté où S représente l'ensemble des sommets et $A \subseteq S \times S$ l'ensemble des arêtes de G. L'ensemble des sommets voisins d'un sommet v est noté $N(v)$ et contient v.
Un sous-ensemble de sommets $S' \subseteq S$ est dit *indépendant* s'il n'existe aucune arête entre les sommets de S'. Nous voulons trouver la taille maximale d'un sous-ensemble de sommets indépendants de G. Ce problème est NP-complet. Tout d'abord, nous allons améliorer la complexité d'algorithmes exponentiels en étudiant les degrés des sommets.

1. Montrer que l'algorithme récursif 1 est correct et donner sa complexité.

Algorithm 1 MaximumIndSet(G)

Require: $G = (S, A)$ un graphe
Ensure: Maximum de la taille de $S' \subseteq S$ indépendant
1: **if** $G = \emptyset$ **then**
2: **return** 0
3: **else**
4: $v \leftarrow G$
5: $avecv \leftarrow 1 + \text{MaximumIndSet}(G \setminus N(v))$
6: $sansv \leftarrow \text{MaximumIndSet}(G \setminus \{v\})$
7: **return** $\max\{avecv, sansv\}$
8: **end if**

2. Dans le cas où v n'a pas de voisin, montrer qu'un des deux appels récursif est inutile. En étudiant le cas où v a au moins un voisin, donner une variante de l'algorithme 1 et montrer que sa complexité est $O(\phi^n)$, où ϕ est racine de l'équation $x^2 - x - 1$.
3. En distinguant le cas où il existe un sommet v ayant exactement un voisin w et le cas complémentaire, donner une nouvelle variante algorithme et calculer sa complexité.

Si le graphe est un arbre, nous pouvons améliorer la complexité de ce problème.

4. Montrer que si $T = (S, A)$ est un arbre et v une feuille de l'arbre, alors il existe un ensemble indépendant de taille maximale contenant v. En déduire un algorithme glouton, expliquer sa correction et analyser sa complexité. (Attention, vous aurez peut-être besoin de maintenir une forêt plutôt qu'un arbre au cours de l'algorithme.)

Enfin, on considère le cas où les sommets de l'arbre ont un poids positif p_v et on recherche un ensemble indépendant de poids total maximal. Si on a une feuille, elle ne fait pas forcément partie d'un ensemble de poids maximal. Si v est une feuille,

$e = (u, v) \in A$, alors si $p_v \geq p_u$, on peut utiliser la même idée que précédemment, mais si $p_v < p_u$, on est face à un dilemme : inclure u pour augmenter le poids total, mais on garde plus d'option si on inclut v. On ne peut pas résoudre ce dilemme de façon locale sans prendre en compte tout l'arbre. On va utiliser une technique de programmation dynamique.

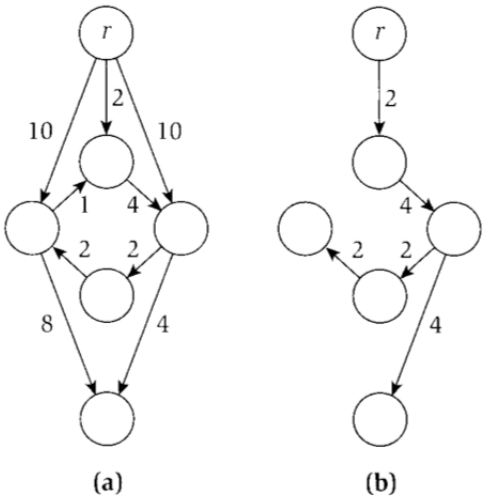
5. Il existe deux fonctions OPT_{avec} et OPT_{sans} de S vers $\mathbb{R}^{*+}$, qui à un sommet u détermine le poids de l'ensemble indépendant maximal suivant si u appartient ou non à l'ensemble. Définissez $\text{OPT}_{avec}(u)$ et $\text{OPT}_{sans}(u)$ en fonction des valeurs de ces fonctions aux fils de u. En déduire un algorithme en stockant ces valeurs aux sommets de l'arbre et évaluer sa complexité.

2 Arborescence de coût minimal d'un graphe orienté

Soit $G = (S, A)$ un graphe orienté connexe et r un sommet distingué. Les arcs possède un coût c'est-à-dire il existe une fonction c de A vers $\mathbb{R}^+$. Un *arbre couvrant* est un arbre T dont les arcs sont un sous-ensemble $F \subseteq A$ du graphe G et qui passe par tous les sommets. Une *arborescence* (par rapport à r) est un arbre couvrant orienté enraciné en r. Il s'agit d'un sous-graphe $T = (S, F)$ tel que T est un arbre couvrant de G si on ignore la direction et il existe un chemin dans T à partir de r vers tous les sommets $v \in S$ si on prend en compte la direction des arcs.

1. Montrer qu'un sous-graphe $T = (S, F)$ de G est une arborescence par rapport à r si et seulement si T n'a pas de cycles, et pour tout sommet $v \neq r$, il existe exactement un arc dans F qui entre dans v. En déduire qu'un graphe orienté G possède une arborescence enracinée en r si et seulement s'il existe un chemin orienté de r vers tous les autres sommets.

Ce problème est plus compliqué dans le cas orienté par rapport au cas non-orienté. Les techniques utilisées dans le cas de graphe non-orienté ne fonctionnent plus car un arc de coût minimal n'appartient pas forcément à l'arbre minimal et l'arc de coût maximal d'un cycle peut appartenir à l'arbre minimal comme le prouve les figures suivantes. Considérons l'ensemble F^* d'arcs formés pour chaque sommet $v \neq r$ de l'arc le plus léger entrant dans v.



2. Montrer que si (S, F^*) est une arborescence, alors c'est également un arbre couvrant minimal. Sinon, montrer que (S, F^*) contient un cycle C ne contenant pas r.

3. Pour tous les sommets v, diminuons le coût de chaque arc entrant au sommet v, du coût de l'arc minimal entrant en v. Appelons cette nouvelle fonction de coût c'. Montrer que T est un arbre optimal par rapport au coût c si et seulement s'il existe un arbre optimal par rapport au coût c'.

Considérons le graphe G avec la fonction c'. Comme tous les arcs de F^* ont un coût nul, si (S, F^*) contient un cycle C tous les arcs sont nuls et donc on peut prendre beaucoup d'arc de C pour construire l'arbre. On réduit le graphe G en un graphe G' en ajoutant un supersommet pour représenter le cycle C et en supprimant toutes les boucles.

Algorithm 2 Arborescence de coût minimal : MSTORIENTE(G, c, r)

Require: Graphe orienté $G = (S, A)$ avec fonction de coût c et un sommet r.
Ensure: Arbre couvrant de coût minimal $T = (S, F)$.

```
for chaque sommet v ≠ r do
    Soit y_v le cout minimal d'un arc entrant en v
    Modifier les couts de tous les arcs a entrant en v à c'_a = c_a − y_v
end for
Choisir un arc de coût 0 entrant chaque v ≠ r pour obtenir un ensemble F*
if F* forme une arborescence then
    return T = (S, F*)
else
    if il existe un cycle orienté C ⊆ F* then
        Contracter C en un seul supersommet pour obtenir le graphe G' = (S', A')
        return MSTORIENTE(G', c', r)
        Étendre (S', F') en une arborescence (S, F) de G en ajoutant tous les arcs sauf un seul de C
        return (S, F)
    end if
end if
```

4. Soit C un cycle dans G consistant en arc de coût 0, tel que $r \notin C$. Montrer qu'il existe un arbre couvrant optimal enraciné en r qui a exactement un arc entrant dans C.
5. Montrer que l'algorithme trouve un arbre couvrant minimal enraciné en r dans G.

3 Flots

Supposons qu'au lieu d'avoir une unique source s et un unique puits t, nous ayons un ensemble S de sources et un ensemble T de puits. Nous supposerons également que les capacités des arcs sont des entiers. Au lieu de maximiser un flot, on va considérer un problème où les sources sont des fournisseurs qui ont des *ressources* et les puits des clients qui ont des *demandes*. Le but est de transporter les ressources à ceux qui ont des demandes et nous voulons satisfaire toutes les demandes en utilisant les ressources disponibles.
Soit $G = (V, A)$ un graphe avec des capacités sur les arcs. On associe à chaque sommet $v \in V$ une *demande* d_v. Si $d_v > 0$, le sommet v a une demande de flot ; c'est un puits, et il veut recevoir d_v unités de plus qu'il renvoie. Si $d_v < 0$, le sommet v a une réserve de $-d_v$; c'est une source. Si $d_v = 0$, il ne s'agit ni d'une source, ni d'un puits. Un tel graphe s'appelle un réseau de circulation.
Une *circulation* avec demandes $\{d_v\}$ est une fonction f qui associe un réel positif à chaque arc et satisfait les deux conditions suivantes :

(Conditions de capacité) Pour chaque $a \in A$, $0 \leq f(a) \leq c_a$.

(Conditions de demande) Pour chaque $v \in S$, $f^{in}(v) - f^{out}(v) = d_v$, où $f^{in}(v)$, $f^{out}(v)$, représentent respectivement le flot entrant et sortant en v.

On s'intéresse au problème de *faisabilité* : existe-t-il une circulation.

1. Montrer que s'il existe une circulation possible avec demandes $\{d_v\}$, alors $\sum_v d_v = 0$. On notera D la valeur commune $\sum_{v:d_v>0} d_v = \sum_{v:d_v<0} -d_v$.
2. Montrer qu'on peut réduire le problème de trouver une circulation possible au problème de calculer un flot maximal dans un réseau de valeur D. Si toutes les capacités et les demandes dans G sont des entiers, et qu'il existe une circulation, alors il existe une circulation à valeurs entières.

Supposons maintenant qu'en plus des capacités, on impose des bornes inférieures $\ell_a \geq 0$ sur la circulation le long de certains arcs. Les conditions de capacités sont de la forme : $\ell_a \leq f(a) \leq c_a$.

3. Montrer comment réduire ce problème au problème de trouver une circulation dans un réseau. (Indication : comme il faut que la circulation fasse passer au moins ℓ_a unités dans ces arcs, considérer la circulation résultante. Cette circulation satisfait les conditions de capacité, mais pas de demandes. Construire un réseau de circulation G' pour lequel trouver une circulation est équivalent à trouver une circulation avec bornes inférieures sur les arcs.)
4. Montrer qu'il existe une circulation dans G si et seulement s'il existe une circulation dans G'. Si toutes les demandes, capacités et bornes inférieures de G sont des entiers, et qu'il existe une circulation, il existe une circulation à valeurs entières.

On va maintenant considérer le problème de maximiser le nombre de chemins à arcs disjoints (pas forcément sommets disjoints) dans un graphe. Ce problème se réduit à un problème de flot maximal quand on recherche les chemins entre deux sommets s et t. Si on cherche à relier le plus grand nombre de paires $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ avec des chemins à arcs disjoints, le problème, *Maximum Chemins Disjoints*, devient NP-complet. Il est même NP-difficile à approximer à moins d'un facteur $O(\sqrt{m})$ près où m est le nombre d'arcs à moins que P=NP.

Pour éviter le plus possible les collisions sur les arcs, l'algorithme glouton suivant sélectionne les chemins par ordre croissant de longueur.

Algorithm 3 Glouton-Disjoints-Chemins(G, L)

Require: Graphe orienté $G = (S, A)$ et une liste de paires de sommets $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$.
Ensure: Ensemble de chemins disjoints $\{P_i\}$ et les indices $i \in I$ des paires connectées.

```
I = ∅
while Aucun chemin nouveau peut être trouvé do
    P_i un chemin court (s'il en existe) qui est à arc disjoint des chemins précédents et qui connecte une paire (s_i, t_i) qui ne l'est pas déjà
    Ajoute i à I et sélectionne P_i pour connecter s_i à t_i.
end while
```

5. Montrer que l'algorithme Glouton-Disjoints-Chemins fournit une $(2\sqrt{m}+1)$-approximation pour le problème *Maximum Chemins Disjoints*. (Distinguer les chemins *courts* de longueurs $\leq \sqrt{m}$ et les chemins longs. Donner une borne supérieure sur le nombre de chemins long et une borne sur le nombre de chemins courts dans la solution optimale en fonction du nombre de chemins courts dans la solution retournée par l'algorithme précédent.)