POLYTECH MONTPELLIER

IG 4 - Algorithmique avancée

Contrôle sur la DP et les réductions : 1h30

- Durées recommandées et barème indicatif :
  Section 2 (questions de cours, 5 points) : 15 min,
  Section 3 (DP, 8 points) 40 min,
  Section 4 (Réduction, 7 points) : 35 min.
- Pour gagnez du temps, ne lisez les rappels de Section 1 que si besoin!
- Commencez par mettre votre prénom/nom/groupe sur vos feuilles

1 Rappels

1.1 Flots

Un **réseau** R est un triplet (G, s, t) où:

- $G = (V, A)$ est un graphe orienté avec une fonction de capacité c sur les arcs (chaque arc $a \in A$ a une capacité $c(a) > 0$)
- s (la source) et t (le puit) sont deux sommets particuliers de V

Soit $R = (G, s, t)$ un réseau. Un **flot** f dans le réseau R est une fonction $f : A \mapsto \mathbb{R}^+$ vérifiant les propriétés suivantes:

- **contrainte de capacité**: pour tout $a \in A$, $f(a) \leq c(a)$
- **conservation du flot**: pour tout u sauf s et t on a $f^+(v) = f^-(v)$,
  avec $f^+(v) = \sum_{vw \in A} f(vw)$ et $f^-(v) = \sum_{wv \in A} f(wv)$
- **La valeur d'un flot** f est $|f| = f^+(s) - f^-(s)$

On définit $\delta^+(X) = \{uv \in A, u \in X, v \notin X\}$, et $\Delta(v) = f^+(v) - f^-(v)$ pour tout $v \in V$. Les notations sont étendues aux sous ensembles.
Problème du FLOT :

- entrée : un réseau R
- sortie : un flot f de R
- objectif : maximiser $|f|$

1.2 Coupes

- **Une coupe** S dans un réseau de $R = (G, s, t)$ est un sous ensemble de sommets tel que $s \in S$ et $t \notin S$.
- Rappel : $\delta^+(S)$ arcs sortants de S, $\delta^-(S)$ arcs entrants de S
- **La valeur d'une coupe** est $c(S) = c(\delta^+(S))$ (et rappel, $c(\delta^+(S)) = \sum_{a \in \delta^+(S)} c(a)$)

Problème de COUPE :

- entrée : un réseau R
- sortie : une coupe S de R
- objectif : minimiser $c(S)$

1.3 Réduction en problèmes d'optimisation

Soit Π_1 et Π_2 deux problèmes de maximisation (avec c_i leurs fonctions de coût). On dit que Π_1 se S-réduit à Π_2 (et on note $\Pi_1 \leq_S \Pi_2$) ssi il existe deux algorithmes poly f, g tel que

- à partir d'une instance I_1 de Π_1, f crée une instance I_2 de Π_2
- pour tout t, $\exists s_1$ solution de I_1 avec $c_1(s_1) \geq t \Leftrightarrow \exists s_2$ solution de I_2 avec $c_2(s_2) \geq t$
- pour toute solution s_2, g calcule une solution s_1 tq $c_1(s_1) \geq c_2(s_2)$

2 Questions de cours

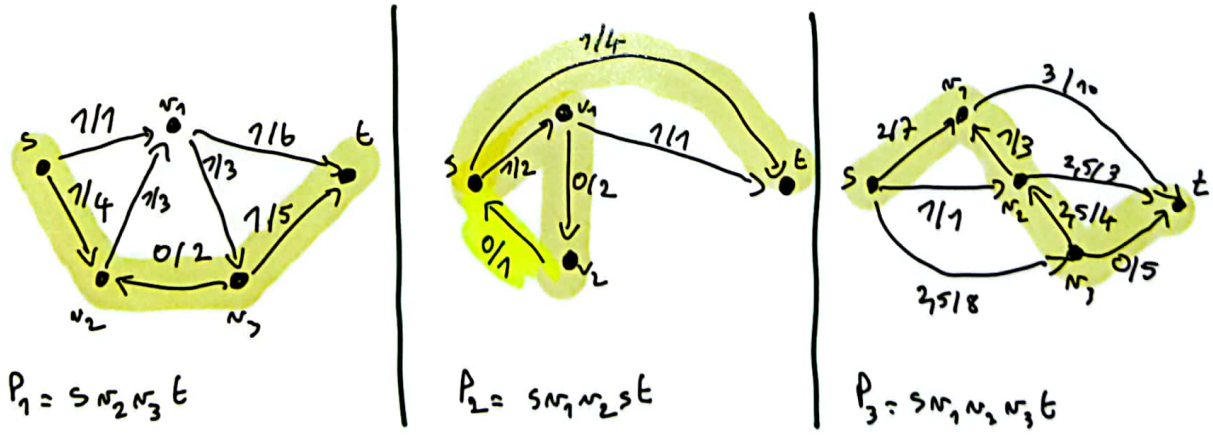
Exercice 1. *Question cours (2 points)*

1. Soit $R = (G, s, t)$ un réseau, f un flot sur R, et S une coupe ($s \in S$ et $t \notin S$) tels que $|f| < c(S)$. Peut on en déduire que forcement f n'est pas maximum ?

Exercice 2. *Question cours (3 points)*

1. Dans chacune des 3 situations suivantes, on considère un réseau R_i, un flot f_i sur ce réseau, et un P_i. Vous devez dire pour chaque situation si :

- oui ou non le chemin proposé P_i est bien un chemin améliorant (en justifiant si la réponse est non)
- si oui, alors donner sa valeur ϵ_{P_i}



3 Programmation dynamique

Exercice 3. *DP : plus grande sous suite croissante (8 points)*
On considère le problème suivant noté SUITE-CROISS de la plus grande sous suite croissante :

- entrée : un tableau d'entiers t à n cases
- sortie : une séquence d'indices du tableau $(i_1, i_2, .., i_k)$ (avec $i_l < i_{l+1}$, et $0 \leq i_l < t.length$) telle que $t[i_l] \leq t[i_{l+1}]$ pour tout l
- objectif : maximiser k, la longueur de la séquence

Par exemple, pour $t = [5, 6, 2, 5, 3, 7, 8, 9, 1, 11]$, on a $opt(t) = 6$ car la plus longue sous suite croissante est $(2, 4, 5, 6, 7, 9)$ (on rappelle qu'une sous suite contient les indices des cases, et pas les valeurs), et est donc de longueur 6.
Nous allons résoudre ce problème par programmation dynamique. Essayons à présent de donner l'intuition derrière le problème auxiliaire que nous allons définir. Supposons que l'on veuille calculer la plus grande sous suite croissante de $t[i..]$ (le sous tableau allant de la case i à la fin du tableau). Le problème est qu'à priori on ne sait pas si la meilleure sous suite contient la case i ou non ... on va donc essayer les deux ! Attention :

- si l'on ne prend pas la case i, alors on peut bien chercher récursivement la meilleure sous suite à partir de $i + 1$
- mais si l'on prend la case i, il faut faire attention à bien donner à l'appel récursif de quoi se souvenir de ce choix, afin qu'il choisisse des cases dont la valeur est supérieure à $v = t[i]$

On obtient donc le problème auxiliaire suivant noté SUITE-CROISS-AUX (dans lequel t est "fixé") :

- entrée : un indice i (avec $0 \leq i \leq t.length$) et un entier v
- sortie : une séquence d'indices du tableau $(i_1, i_2, .., i_k)$ (avec $i_l < i_{l+1}$, et $0 \leq i_l < t.length$) telle que $t[i_l] \leq t[i_{l+1}]$ pour tout l **et telle que :**
  - $i \leq i_1$
  - $v \leq t[i_1]$
- objectif : maximiser k, la longueur de la séquence

Autrement dit, dans SUITE-CROISS-AUX, on cherche donc $opt_{AUX}(t, i, v)$: la plus longue sous suite croissante à droite de i, et dont tous les éléments du tableau correspondants sont plus grands que v.

1. Ecrire l'algorithme récursif AUX (sans le transformer en DP) suivant :

```
int aux(int[]t,int i,int v)
 // prerequis : 0 <= i <= t.length
 // action : retourne opt_AUX(t,i,v)
```

Par exemple, pour $t = [8, 6, 3, 9, 4, 10]$, $i = 1$, $v = 7$, AUX(t,i,v) doit retourner 2 correspondant à la sous suite $(3, 5)$. On remarque que AUX(t,i,v) ne peut pas considérer la sous suite $(1, 3, 5)$ car $t[1] < v$.

2. Ajouter (dans une autre couleur, ou recopiez ailleurs) les instructions pour transformer AUX en une programmation dynamique AUXDP, et ajouter aussi une méthode "cliente" SOUSSUITE qui résout le problème principal. Plus précisément, on souhaite donc avoir :

- une méthode int $auxDp(t, i, v, tabMarq)$ où $tabMarq$ est un tableau dont vous préciserez le type
- une méthode int $suiteCroiss(t)$ qui, étant donné une instance de SUITE-CROISS donnée en paramètre, calcule la valeur optimale.

(Remarquez que la méthode aux de la première question n'est donc pas utilisée, elle a juste servi de version préliminaire de la méthode AUXDP.)

3. Déterminez la complexité (en temps) de la méthode *suiteCroiss* n, la taille du tableau.

4 Réduction

Exercice 4. *Réduction correcte ? (7 points)*
On rappelle le problème MAX-2-SAT :

- entrée : m clauses de tailles $\{C_i\}$ de taille au plus 2, avec $C_i = l_i^1$ si C_i est de taille 1, ou $C_i = l_i^1 \vee l_i^2$ si C_i est de taille 2 (où l_i^x dénote un littéral, c'est à dire une variable X_j ou sa négation $\bar{X}_j$
- sortie : une affectation des n variables
- objectif : maximiser le nombre de clauses satisfaites

Par exemple pour l'entrée $C_1 = \bar{X}_2, C_2 = X_2 \vee X_3, C_3 = X_1 \vee X_3$, la valeur optimale est 3 car l'affectation $X_1 = vrai$, $X_2 = faux$, $X_3 = vrai$ satisfait les 3 clauses.

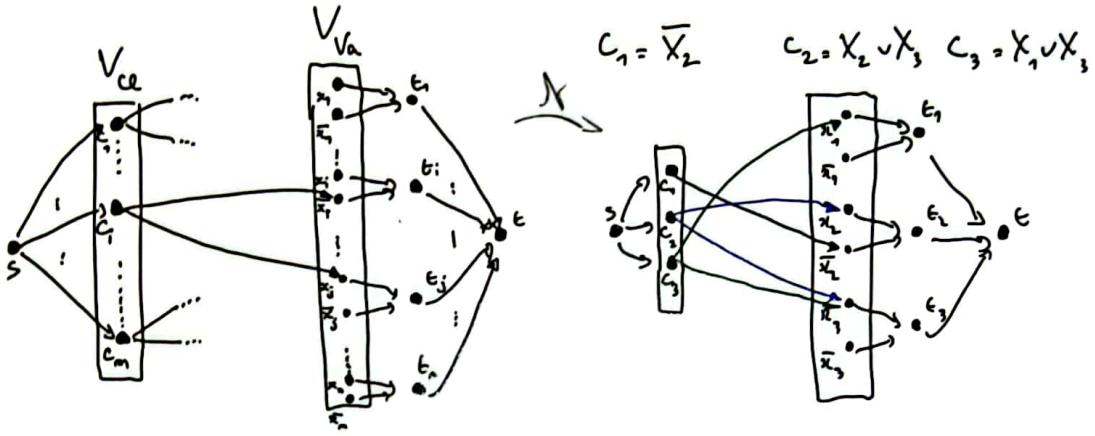


Figure 1: Définition de la réduction (toutes les capacités des arcs valent 1)

On souhaiterait montrer que MAX-2-SAT $\leq_S$ FLOT avec la réduction suivante (voir Figure 1). Etant donné une entrée de MAX-2-SAT, notre fonction f_R (que l'on évite de noter f ici pour ne pas la confondre avec un flot) va:

- créer d'abord deux ensembles de sommets, V_{Cl} et V_{Va} ainsi : pour chaque clause C_i, on crée un sommet $c_i \in V_{Cl}$, et pour chaque littéral deux sommets $x_j \in V_{Va}$ et $\bar{x}_j \in V_{Va}$
- ajouter pour tout $i \in \{1, \ldots, m\}$ un arc de c_i vers les (au plus deux) sommets correspondants aux littéraux qui composent C_i
- ajouter pour chaque $j \in \{1, \ldots, n\}$ un sommet t_j, et les arcs $(x_j t_j)$, $(\bar{x}_j t_j)$
- ajouter la source s avec les arcs (sc_i) pour tout $i \in \{1, \ldots, m\}$, et le puits t avec les arcs $(t_j t)$ pour tout $j \in \{1, \ldots, n\}$
- enfin, définir toutes les capacités des arcs égales à 1.

1. Est ce que le sens $\Rightarrow$ de l'équivalence (cf section rappels) que devrait vérifier f_R est respecté ? (répondre soit OUI avec une preuve, soit NON avec un contre exemple).

2. Est ce que le sens $\Leftarrow$ de l'équivalence (cf section rappels) que devrait vérifier f_R est respecté ? (répondre soit OUI avec une preuve, soit NON avec un contre exemple).