TD 4

Algorithmique

Exercice 1: Coupe minimale dans un graphe

Soit G un graphe connecté, non-orienté avec éventuellement plusieurs arêtes entre les n sommets. Une *coupe* dans G est un ensemble d'arêtes qui si on les enlève, G devient non-connexe. Une *coupe minimale* est une coupe de cardinalité minimale. Le problème de trouver une coupe maximale est NP-complet.

L'idée de l'algorithme est de choisir uniformément une arête et de fusionner les deux sommets en un seul sommet en mettant sur ce sommet les arêtes qui arrivaient aux deux sommets initiaux et en enlevant les boucles. On appelle cette opération une *contraction*. On voit que même si le graphe initial n'avait qu'une seule arrête entre chaque sommet, le graphe ayant subi une contraction peut en contenir au plus deux. Ce processus diminue d'une unité le nombre d'arête. L'algorithme effectue des contractions jusqu'à ce que le nombre de sommets soit égal à 2 et retourne comme valeur le nombre d'arêtes entre ces deux points.

1. Montrer qu'une contraction d'arête ne diminue pas la valeur d'une coupe minimale.
2. Soit k la valeur d'une coupe minimale. Montrer que G a au moins $kn/2$ arêtes.
3. Soit $\mathcal{E}_i$ l'événement de ne pas choisir une arête de C à la i-ième étape, pour $1 \leq i \leq n-2$.
   (a) Montrer que $\Pr[\mathcal{E}_1] \geq 1 - 2/n$.
   (b) Montrer que $\Pr[\mathcal{E}_2|\mathcal{E}_1] \geq 1-2/(n-1)$ et plus généralement que $\Pr[\mathcal{E}_i|\cap_{j=1}^{i-1}\mathcal{E}_j] \geq 1-2/(n-i+1)$.
   (c) Montrer que la probabilité trouver une coupe minimale par ce procédé est au moins $\frac{2}{n(n-1)}$.
4. Montrer comment obtenir un algorithme dont la probabilité d'échec soit $< 1/e$ et donner sa complexité.

Exercice 2: Clôture transitive dans un graphe

1. Rappeler l'algorithme de Floyd-Warshall pour calculer la clôture transitiveen $\Theta(n^3)$ en temps.
2. Montrer que s'il existe un algorithme qui calcule la clôture transitive d'une matrice $n \times n$ en $A(n)$ additions, multiplications d'éléments d'un semi-anneau et si $A(3n) \leq c \cdot A(n)$ pour un réel $c > 0$ et tout n entier positifs, alors il existe un algorithme de multiplication avec $M(n) = O(A(n))$ additions et multiplications.
3. Montrer que si le produit de deux matrices $n \times n$ peut être calculé en $M(n)$ additions et multiplications et si $4 \cdot M(n/2) \leq M(n)$ et $M(2n) \leq c \cdot M(n)$ pour un c et tout n, alors la clôture transitive se calcule en $A(n) = O(M(n))$ additions et multiplications.
4. Montrer que le semi-anneau booléen $B = (\{0,1\}, \vee, \wedge, 0, 1)$ n'est pas un anneau.
5. Soit A et B deux matrices $n \times n$ booléeennes. Le produit peut se calculer en $O(n^{\log 7} \cdot (\log n)^2) = O(n^{2.82})$ opérations binaires, ainsi que la clôture transitive.

Exercice 3: Chemins de longueur minimale

On utilise le semi-anneau des réels pour traiter les chemins de longueur minimal dans un graphe. Soit A et B deux matrices $n \times n$ à cœfficients dans $[0..M] \cup \{\infty\}$. On veut calculer la matrice C de cœfficients

$$c_{ik} = \min_{1 \leq j \leq n} (a_{ij} + b_{jk}).$$

1. Montrer que l'algorithme classique fonctionne en $O(n^3 \log M)$.
2. On peut améliorer un peu cet algorithme en utilisant le fait que pour $a \neq b$,

$$\lim_{x \to 0} (x^a + x^b)/x^{\min(a,b)} = 1$$

et donc $\min(a, b) = \log(x^a + x^b)/\log x$ pour x petit.

   (a) Soit $b_1, \ldots, b_n$, n entiers positifs, et soit $f(x) = \sum_{k=1}^{n} x^{b_k}$ et $a = \min(b_1, \ldots, b_n)$. Montrer alors que $a = \lceil -(1/m) \log f(2^{-m}) \rceil$ pour tout $m > \log n$.

   (b) Soit un réel $0 < x < 1$ et z le nombre de zéros en tête de la représentation binaire de x. Alors $\lceil -(1/m) \log x \rceil = 1 + \lfloor z/m \rfloor$.

   (c) Donner un algorithme qui calcule le $(\min, +)$-produit de deux matrices A et B avec des entrées dans $[0..M] \cup \{\infty\}$ en $O(n^{\log 7})$ opérations arithmétiques sur les réels ou $O(n^{\log 7} \cdot (M \log n)^2)$ opérations binaires.
3. Comparer les deux algorithmes et donner la valeur de M pour laquelle le premier algorithme est plus efficace.
4. Est-ce que cet algorithme est plus rapide quand il s'agit de matrices booléennes ?

Exercice 4: Problème 2-SAT

Soit $x_1, \ldots, x_n$ un ensemble de n variables booléennes indexées par les entiers $\leq n$. Un littéral sur cet ensemble de variables est une variable x_i $i \leq n$, ou sa négation $\neg x_i$. Une 2-clause sur cet ensemble de variable est une disjonction de deux littéraux $\ell \vee \ell'$. Une assignation τ est une fonction de l'ensemble $x_1, \ldots, x_n$ dans $\{0, 1\}$. La valeur d'un littéral ℓ relativement à une assignation τ, soit $V(\ell, \tau)$ est égale à $\tau(x_i)$ si $\ell = x_i$ et à $1 - \tau(x_i)$ si $\ell = \neg x_i$. La valeur d'une clause c relativement à une assignation τ, soit $V(c, \tau)$, est le maximum des valeurs de ses deux littéraux. Le problème 2-SAT s'énonce comme suit :

Étant donné un entier n et une liste de 2-clauses sur l'ensemble de variables $x_1, \ldots, x_n$, déterminer s'il existe une assignation τ qui donne à toutes les clauses de la liste la valeur 1 (problème de décision). Calculer une telle assignation si elle existe (calcul d'une solution).

1. On associe à une liste de 2-clauses sur l'ensemble de variables $x_1, \ldots, x_n$ un graphe orienté G dont les sommets sont les littéraux sur l'ensemble de variables $x_1, \ldots, x_n$. Pour chaque clause $\ell \vee \ell'$, on ajoute à G une arête allant de $\neg\ell$ à ℓ' et une arête allant de $\neg\ell'$ à ℓ (où on pose $\neg\neg x_i = x_i$).

   (a) Montrer que l'on peut calculer G en temps $O(n + m)$ où n est le nombre de variables et m le nombre de clauses. On explicitera un choix de structures de données et on proposera une description de l'algorithme en terme de pseudo-code.

(b) Montrer que, s'il existe dans G un chemin allant de u à v, il existe aussi un chemin allant de $\neg v$ à $\neg u$.

2. Soit τ une telle assignation. On étend τ à l'ensemble des sommets de G en posant $\tau(\neg x_i) = 1 - \tau(x_i)$. Montrer que $V(c, \tau)$ vaut 1 pour toutes les clauses de la liste donnée si et seulement si τ est croissante sur tout chemin γ du graphe G (ce qui signifie : si $\gamma = u_0, \ldots, u_k$, alors $\tau(u_i) \leq \tau(u_{i+1})$ pour $i = 0, \ldots, k-1$).
3. En utilisant l'algorithme qui détermine les composantes fortement connexes d'un graphe orienté ou un autre des algorithmes sur les graphes, donner un algorithme qui résout le problème de décision 2-SAT. Montrer sa correction et évaluer sa complexité.
4. Comment calculer une assignation lorsque la réponse au problème de décision est positive ? On évaluera la complexité de l'algorithme proposé.