École Normale Supérieure **2010-2011**
Département d'Informatique

Algorithmique et Programmation
Examen

Notes de cours autorisées à l'exception de tout autre document.

Le résultat d'une question peut être utilisé dans la suite du problème même si elle n'a pas été résolue.

Une justification est demandée pour toute réponse et sera prise en compte dans la notation.

Durée : 3 heures

Exercice 1. RECHERCHE DE MOTIFS

1. Rappeler la variante de l'algorithme de Aho-Corasick (vue en TD) de recherche de motifs qui détermine si un motif P de longueur m apparaît dans un texte T de longueur n lorsque le motif P contient k caractères spéciaux *jokers* (?), *i.e.* qui détermine l'ensemble

$$\mathcal{M}_? = \{i \in \{1,\dots,n\} \quad | \quad \forall j \in \{1,\dots,m\}, (p_j = t_{i+j-1}) \vee (p_j = ?)\}.$$

   Donner sa complexité.

Le but de cet exercice est de proposer des algorithmes *numériques* de recherche de motifs (classique, avec *jokers* et avec erreurs). Nous supposerons sans perte de généralité que l'alphabet $\Sigma = \{1,\dots,\sigma\} \subset \mathbb{N}$ est l'ensemble des σ premiers entiers positifs.

2. (a) En remarquant que le motif $P = (p_1,\dots,p_m) \in \Sigma^m$ apparaît à la position $i \in \{1,\dots,n\}$ de $T = (t_1,\dots,t_n) \in \Sigma^n$ si et seulement si

$$\sum_{j=1}^{m} (p_j - t_{i+j-1})^2 = 0$$

   proposer un algorithme de complexité $O(n \log n)$ opérations arithmétiques pour le problème de recherche de motifs (classique).

   (b) En découpant le texte T en n/m blocs de longueur $2m$ (se recouvrant partiellement), montrer comment adapter l'algorithme précédent pour avoir une complexité en $O(n \log m)$ opérations arithmétiques.

3. Adapter cet algorithme pour la recherche de motifs lorsque le motif P *et* le texte T contiennent des caractères spéciaux *jokers* (?), *i.e.* qui détermine l'ensemble

$$\mathcal{M}_{??} = \{i \in \{1,\dots,n\} \quad | \quad \forall j \in \{1,\dots,m\}, (p_j = t_{i+j-1}) \vee (p_j = ?) \vee (t_{i+j-1} = ?)\}.$$

4. Nous allons adapter l'algorithme de la question 2 pour qu'il détermine le nombre de lettres communes d'un motif à chaque position d'un texte (le motif et le texte ne contenant plus de caractère *joker*).

   (a) Donner un algorithme en $O(n\sigma \log m)$ opérations arithmétiques qui prenant en entrée P et T retourne un vecteur $h(P,T) = (h_1,\dots,h_{n-m+1}) \in \mathbb{N}^{n-m+1}$ tel que $h_i = \sum_{j=1}^{m} \mathrm{eq}(p_j, t_{i+j-1})$ où $\mathrm{eq}(a,a) = 1$ et $\mathrm{eq}(a,b) = 0$ si $a \neq b$.

   (b) Soit $f \in \mathbb{N}$. Une lettre de Σ est dite *f-fréquente* si elle apparaît au moins f fois dans P. En supposant que le motif P ne contient aucune lettre f-fréquente, donner un algorithme de complexité $O(nf)$ (et donc indépendant de σ) qui prenant en entrée P et T retourne le vecteur $H(P,T)$.

   (c) En appliquant ces deux algorithmes en fonction de la fréquence des lettres de P, donner un algorithme qui prenant en entrée un motif $P \in \Sigma^m$ arbitraire et un texte $T \in \Sigma^n$ retourne le vecteur $H(P,T)$ en $O(n\sqrt{m \log m})$ opérations arithmétiques.

5. Nous allons adapter cet algorithme pour qu'il détermine les occurences du motif P qui diffèrent de T en au plus k lettres, *i.e.*

$$\mathcal{M}_k = \{i \in \{1,\dots,n\} \quad | \quad \#\{j \in \{1,\dots,m\},\, p_j \neq t_{i+j-1}\} \leq k\}.$$

   (a) En posant $x_{i,j} = (p_j - t_{i+j-1})^2$ pour $(i,j) \in \{1,\dots,n\} \times \{1,\dots,m\}$, montrer que $i \in \mathcal{M}_k$ si et seulement si

$$C_i := \sum_{1 \leq j_1 \leq \cdots \leq j_{k+1} \leq m} x_{i,j_1} \dots x_{i,j_{k+1}} = 0.$$

   (b) Donner un algorithme qui calcule les valeurs

$$D_s(i) = \sum_{j=1}^{m} x_{i,j}^s$$

   pour $s \in \{1,\dots,k+1\}$ et $i \in \{1,\dots,n\}$ en $O(k^2 n \log m)$ opérations arithmétiques.

   (c) Posons pour s et t des entiers tels que $s + t \leq k + 2$

$$F_{t,1}(i) = \sum_{1 \leq j_1 < j_2 < \cdots < j_t \leq m} x_{i,j_1}^s x_{i,j_2} \dots x_{i,j_t}.$$

   et pour $s \geq 2$,

$$F_{t,s}(i) = \sum_{\substack{1 \leq j_1 \leq m \\ 1 < j_2 < \cdots < j_t \leq m \\ \forall \ell \neq 1, j_\ell \neq j_1}} x_{i,j_1}^s x_{i,j_2} \dots x_{i,j_t}.$$

   de sorte que $F_{1,s}(i) = D_s(i)$ et $F_{k+1,1}(i) = C(i)$. Montrer la récurrence suivante :

$$F_{t+1,s}(i) = \frac{1}{c_s}(F_{t,1}(i) \cdot F_{1,s}(i) - F_{t,s+1}(i))$$

   où $c_1 = t + 1$ et $c_s = 1$ si $s > 1$.

   (d) En déduire un algorithme pour calculer $\mathcal{M}_k$ en $O(nk^2 \log m)$ opérations arithmétiques.

   (e) Le calcul précédent manipule des grands entiers et la complexité en opérations arithmétiques est inférieure à la complexité réelle. Donner un algorithme probabiliste analogue qui ne manipule que des entiers de taille $O(\log \sigma)$.

Exercice 2. Régression linéaire

Soient

$$\vec{b} = \begin{pmatrix} b_1 \\ \dots \\ b_m \end{pmatrix} \in \mathbb{R}^n, A = \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix} \in \mathbb{R}^{n \times m}$$

Le but de l'exercice est de calculer les vecteurs x qui réalisent les minima

$$\min_{y \in \mathbb{R}^n} \|Ay - b\|_1 \quad \text{et} \quad \min_{y \in \mathbb{R}^n} \|Ay - b\|_2.$$

1. Un *pseudo-inverse* de A est une matrice $G \in \mathcal{M}_{n,m}$ vérifiant $AGA = A$, $GAG = G$, ${}^t(AG) = AG$, ${}^t(GA) = GA$. Nous allons montrer que toute matrice admet un pseudo-inverse que nous notons A^g et de donner un algorithme pour le calculer.

   (a) Montrer qu'une matrice A possède au plus un pseudo-inverse.

   (b) Soit A une matrice de rang n. Montrer que $({}^tAA)^{-1}{}^tA$ est le pseudo-inverse de A.

   (c) Soit A une matrice de rang r et supposons que A se mette sous la forme U concaténé avec UK où $U \in \mathcal{M}_{m,r}$ est de rang r et $K \in \mathcal{M}_{r,n-r}$.

   Montrer que $\begin{pmatrix} WU^g \\ {}^tKWU^g \end{pmatrix}$ où $W = (I_r + K{}^tK)^{-1}$ est le pseudo-inverse de A.

(d) En déduire que toute matrice possède un unique pseudo-inverse et donner un algorithme pour la calculer.

(e) Démontrer que x est solution de

$$\|Ax - b\|_2 = \min_{y \in \mathbb{R}^n} \|Ay - b\|_2. \tag{1}$$

si et seulement si ${}^tAAx = {}^tAb$. En déduire que $A^g b$ est solution de (1). Démontrer que si A est de rang n, alors (1) a une solution unique.

2. La valeur de x telle que

$$\|Ax - b\|_1 = \min_{y \in \mathbb{R}^n} \|Ay - b\|_1. \tag{2}$$

n'a pas de formule explicite mais nous allons proposer un algorithme pour la calculer.

(a) Résoudre le problème (2) lorsque $m = 1$.
Indication : On pourra se ramener au cas où $a_{1,1} = \cdots = a_{n,1} = 1$ (vu en TD).

(b) Montrer que le problème (2) s'écrit sous la forme du problème d'optimisation linéaire suivant :

$$\text{minimiser } \sum_{i=1}^{m}(u_i + v_i)$$
$$\text{sous la contrainte } \begin{cases} b_j = \sum_{i=1}^{n}(x_i - y_i)a_{j,i} + u_j - v_j, \quad j \in \{1, \ldots, n\} \\ x_i, y_i, u_i, v_i \geq 0, \quad i \in \{1, \ldots, n\} \end{cases}$$

(c) Donner la forme duale du problème d'optimisation linéaire précédent et montrer qu'elle est équivalente au problème suivant :

$$\text{maximiser } \sum_{i=1}^{m} w_i b_i$$
$$\text{sous la contrainte } \begin{cases} \sum_{i=1}^{n} a_{j,i} w_i = 0, \quad j \in \{1, \ldots, n\} \\ -1 \leq w_i \leq 1, \quad i \in \{1, \ldots, n\} \end{cases}$$

Exercice 3. MULTIPLICATION DE MATRICES CREUSES

La multiplication naïve de deux matrices $n \times n$ effectue $O(n^3)$ opérations dans l'anneau de base. Strassen a le premier montré que l'algorithme n'est pas optimal en donnant un algorithme en $O(n^{2.81})$ opérations. Souvent en pratique, les matrices à multiplier sont *creuses*, *i.e.* leur nombre d'éléments non-nuls est petit comparé à leur nombre de zéros. Malheureusement, les algorithmes de multiplication rapides ne peuvent pas utilisés le caractère "creux" des matrices multipliées. Le but de cet exercice est de montrer que les algorithmes de multiplication rapide peuvent quand même être utilisés pour accélérer le calcul du produit de matrices creuses.

Notation : Soient A et B deux matrices $n \times n$ sur un anneau R. A_{*k} désigne la k-ième colonne de A et B_{k*} désigne la k-ième ligne de B, pour $1 \leq k \leq n$. Nous notons a_k et b_k le nombre d'éléments non-nuls de A_{*k} et B_{k*} respectivement.

1. Montrer que le nombre de multiplications nécessaires pour calculer le produit AB par l'algorithme naïf est $\sum_{k=1}^{n} a_k b_k$.

2. Montrer que le nombre de multiplications effectuées pour calculer AB par l'algorithme de multiplication naïve est en $\Theta(mn)$ si $\sum_{k=1}^{n} a_k \leq m$ et $\sum_{k=1}^{n} b_k \leq m$.

Il est intéressant de noter que le calcul de AB dans le pire des cas peut être réduit au calcul d'un produit de matrices rectangulaires plus petit. Nous allons utiliser le fait que lorsque l'algorithme naïf doit effectuer beaucoup d'opérations, la multiplication rapide de matrices rectangulaires peut être utilisée pour accélerer le calcul.

Notation : Nous notons $M(a,b,c)$ le nombre minimal d'opérations algébriques nécessaires pour calcul le produit d'une matrice $a \times b$ par une matrice $b \times c$ et $\omega(r,s,t)$ l'exposant minimal pour lequel $M(n^r, n^s, n^t) = O(n^{\omega+o(1)})$. Les meilleures bornes (dues à Coppersmith et Winograd) connues sur $\omega(1,r,1)$ pour $0 \leq r \leq 1$ sont :

$$\omega := \omega(1,1,1) \leq 2,376 \tag{3}$$

$$\max_{\{0\leq r\leq 1 \quad |} \quad \omega(1,r,1) = 2\} =: \alpha > 0.294 \tag{4}$$

3. Montrer que
$$M(ap, bp, cp) \leq M(a,b,c)M(p,p,p)$$
pour tout $a, b, c, p \in \mathbb{N}$.

4. En déduire que
$$\omega(1,r,1) \leq \begin{cases} 2 & \text{si } 0 \leq r \leq \alpha \\ 2 + \beta(r-\alpha) & \text{sinon} \end{cases}$$
où $\beta = (\omega - 2)/(1-\alpha)$.

5. Soit $\pi \in \mathfrak{S}_n$ une permutation telle que $a_{\pi(1)}b_{\pi(1)} \geq a_{\pi(2)}b_{\pi(2)} \geq \cdots \geq a_{\pi(n)}b_{\pi(n)}$. Montrer que pour tout $1 \leq \ell \leq n$,
$$\sum_{k=\ell+1}^{n} a_{\pi(k)}b_{\pi(k)} \leq \frac{m_1 m_2}{\ell}$$
où $m_1 = \sum_{k=1}^{n} a_k$ et $m_2 = \sum_{k=1}^{n} b_k$.

6. Montrer la validité de l'algorithme suivant et donner sa complexité en fonction de n, m_1 et m_2.

Algorithm 1 Multiplication rapide de matrices creuses

Require: A et B deux matrices $n \times n$
Ensure: AB
Soit a_k le nombre d'éléments non nuls de A_{*k} la k-ième colonne de A, pour $1 \leq k \leq n$.
Soit b_k le nombre d'éléments non nuls de B_{k*} la k-ième ligne de B, pour $1 \leq k \leq n$.
Soit $\pi \in \mathfrak{S}_n$ une permutation telle que $a_{\pi(1)}b_{\pi(1)} \geq a_{\pi(2)}b_{\pi(2)} \geq \cdots \geq a_{\pi(n)}b_{\pi(n)}$
Trouver un ℓ qui minimise $M(n,\ell,n) + \sum_{k\geq\ell} a_{\pi(k)}b_{\pi(k)}$
Soit $I = \{\pi(1), \ldots, \pi(\ell)\}$ et $J = \{\pi(\ell+1), \ldots, \pi(n)\}$
Calculer $C_1 = A_{*I}B_{I*}$ en utilisant l'algorithme rapide de multiplication de matrices rectangulaires
Calculer $C_2 = A_{*J}B_{J*}$ en utilisant l'algorithme naïf de multiplication de matrices creuses
Retourner $C_1 + C_2$

En particulier, en utilisant les meilleures bornes connues sur ω et α, montrer que le nombre d'opérations effectuées par cet algorithme est au plus $O(m^{0,7}n^{1,2} + n^{2+o(1)})$ pour $m = m_1 = m_2$.

7. $(*)$ Comme le produit de deux matrices creuses n'est pas nécessairement creux, nous ne pouvons pas utiliser cet algorithme directement pour calculer efficacement le produit de plus de deux matrices creuses. Proposer un algorithme qui calcule le produit de trois matrices creuses plus rapidement que la multiplication naïve creuse et la multiplication rapide "pleine" pour $n^{\theta_1} < m < n^{\theta_2}$ (où m est une majoration du nombre d'éléments non nuls des trois matrices.