ÉCOLE NORMALE SUPÉRIEURE DE LYON

Concours d'admission session 2023

Filière universitaire : Second concours

COMPOSITION D'INFORMATIQUE

Durée : 3 heures

L'utilisation des calculatrices n'est pas autorisée pour cette épreuve.

⋆ ⋆ ⋆

Quelques objets combinatoires munis d'un ordre partiel

Le sujet comporte 4 pages, numérotées de 1 à 4.

On étudie dans ce problème quelques résultats combinatoires simples liés aux matrices ordonnées par lignes et par colonnes, et aux partitions d'un entier. La première partie concerne la recherche d'un élément dans une matrice ordonnée, la second partie traite de l'insertion et de la suppression dans une telle matrice. La troisième partie, indépendante, se concentre sur les partitions d'un entier.

Définitions, algorithmes, complexité

$\mathbb{N}$ désigne l'ensemble des entiers naturels, $\mathbb{Z}$ l'ensemble des entiers relatifs et $\mathbb{N}^*$ l'ensemble $\mathbb{N}$ privé de 0. Un tableau unidimensionnel à k éléments, où $k \in \mathbb{N}^*$, est représenté en mémoire par un tableau $array[1..k]$ dont les indices commencent à 1. Une matrice à m lignes et n colonnes, où $m, n \in \mathbb{N}^*$, est représentée en mémoire par un tableau à deux dimensions $array[1..m, 1..n]$ dont les indices commencent à 1.

Lorsqu'on demande l'écriture d'un algorithme, on pourra donner celui-ci soit en pseudo-code, soit dans le langage de programmation de votre choix (à préciser au début du sujet). Dans le cas de l'utilisation de pseudo-code, on s'assurera de donner suffisamment de détail, en particulier sur les structure de données utilisées.

La complexité d'un algorithme qui manipule un tableau unidimensionnel de taille k ou une matrice de taille $m \times n$ est définie comme le nombre d'opérations élémentaires que l'algorithme effectue **dans le pire cas**. Cette complexité est exprimée en ordre de grandeur, par exemple $O(k^2)$ ou bien $O(m \times \log(n))$. La notation $O(f(k))$ signifie qu'il existe une constante c telle que la complexité est majorée par $c \times f(k)$ pour tout tableau de taille k (et de même pour les matrices). Noter que la base du logarithme $\log(n)$ n'a pas d'importance dans ce contexte.

Partie I. Recherche dans une matrice partiellement ordonnée

On considère une matrice rectangulaire $\mathcal{A} = (a_{i,j})$ à m lignes et n colonnes avec $m, n \geq 1$. On suppose que les coefficients $(a_{i,j})$ sont dans $\mathbb{N}^*$. On suppose aussi la croissance au sens large sur chaque ligne et chaque colonne :

— $a_{i,j} \leq a_{i,j+1}$ pour $1 \leq i \leq m-1, 1 \leq j < n$ (croissance par lignes)

— $a_{i,j} \leq a_{i+1,j}$ pour $1 \leq i < m, 1 \leq j \leq n-1$ (croissance par colonnes).

On va considérer plusieurs algorithmes pour rechercher un entier $x \in \mathbb{N}$ dans la matrice $\mathcal{A}$.

Question I.1. Proposer un algorithme pour trouver un entier x dans la matrice $\mathcal{A}$, qui n'utilise pas l'information sur l'ordre dans les lignes et les colonnes. Donner et justifier sa complexité.

Question I.2. On utilise la recherche dichotomique dans cette question.

1. Soit $\mathcal{B} = (b_i)$ un tableau unidimensionnel à k coefficients avec $k \geq 1$. On suppose que les coefficients b_i sont dans $\mathbb{N}^*$ et croissants au sens large : $b_i \leq b_{i+1}$ pour $1 \leq i < k$. Donner un algorithme pour rechercher un entier $x \in \mathbb{N}^*$ dans le tableau $\mathcal{B}$ avec une complexité $O(\log(k))$, expliquer son fonctionnement et justifier sa complexité.
2. Expliquer comment utiliser l'algorithme précédent pour la recherche dans la matrice $\mathcal{A}$. En fonction des valeurs de m et n, faut-il procéder par lignes ou par colonnes ?

Question I.3. On cherche maintenant un algorithme de type *diviser-pour-régner*. Pour simplifier, on suppose dans cette question que $\mathcal{A}$ est carrée et de taille $m = n = 2^p + 1$ avec $p \geq 1$.

1. Soit $y = a_{\frac{n+1}{2},\frac{n+1}{2}}$ le coefficient du milieu de la matrice $\mathcal{A}$ (les valeurs de m et n permettent de définir le milieu sans ambiguïté). Quand on compare x et y (et que $x \neq y$), quel quadrant de la matrice $\mathcal{A}$ peut-on éliminer pour poursuivre la recherche ?
2. Donner un algorithme qui continue la recherche indépendamment dans les trois quadrants restants, en utilisant cette idée à chaque itération.
3. Soit c_q la complexité de cet algorithme pour une matrice carrée de taille $n_q = 2^q + 1$, avec $0 \leq q \leq p$. Montrer que $c_q = 1 + 3c_{q-1}$ pour $q \geq 1$.
4. En considérant la suite $c'_q = c_q + \frac{1}{2}$, montrer que $c_q = O(3^q)$, et en déduire la complexité de l'algorithme en fonction de n.
5. Comment étendre cet algorithme pour une matrice $\mathcal{A}$ carrée de taille $m = n$ arbitraire ? quelle est la complexité ?

Question I.4. On cherche un algorithme de complexité linéaire $O(m+n)$ dans cette question.

1. Soit $y = a_{1,n}$ le coefficient du coin supérieur droit de la matrice $\mathcal{A}$. En comparant x et y (et lorsque $x \neq y$), on peut éliminer une partie de la matrice pour la recherche de x. En utilisant cette idée, expliquer comment obtenir un algorithme plus efficace que les précédents pour trouver x.
2. Donner et justifier sa complexité.

Partie II. Tri avec une matrice partiellement ordonnée

Comme dans la partie précédente, on considère une matrice rectangulaire $\mathcal{A} = (a_{i,j})$ à m lignes et n colonnes avec $m, n \geq 1$. On suppose toujours que les coefficients $(a_{i,j})$ sont dans $\mathbb{N}^*$ et croissants au sens large sur chaque ligne et chaque colonne. Mais on suppose maintenant que la matrice A est partiellement remplie : seuls les k premiers coefficients sont significatifs, avec l'ordre ligne par ligne et de gauche à droite ; autrement dit, seuls les coefficients a_{ij} de numéros 1 à k sont significatifs, où le numéro de a_{ij} est $n \times (i-1) + j$. Les valeurs admissibles de k varient de 0 (matrice vide) à $m \times n$ (matrice pleine). On pose $a_{i,j} = \infty$ si le numéro de $a_{i,j}$ est supérieur à k, avec la convention que ∞ est plus grand que tout entier (pour respecter l'ordre sur les lignes et les colonnes). Voici un exemple avec $m = 4, n = 5, k = 13$:

$$\mathcal{A} = \begin{pmatrix} 1 & 3 & 4 & 12 & 17 \\ 4 & 4 & 4 & 18 & 18 \\ 5 & 7 & 8 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix}$$

Une matrice *Trimat*(m, n, k) est une matrice $\mathcal{A}$ à m lignes et n colonnes croissantes au sens large et avec k éléments significatifs.

Question II.1. On veut insérer un nouvel élément x dans une matrice *Trimat*(m, n, k), avec $k < m \times n$, pour en faire une matrice *Trimat*$(m, n, k+1)$. Expliquer comment procéder et donner un algorithme de complexité linéaire $O(m+n)$. *Indication : on pourra placer x en dernière position $a_{i,j}$ de numéro $k+1$, puis le comparer avec les éléments en positions $a_{i-1,j}$ et $a_{i,j-1}$ (s'ils existent) et le faire remonter dans la matrice via des échanges avec d'autres coefficients.*

Question II.2. On veut supprimer le coefficient minimum a_{11} d'une matrice *Trimat*(m, n, k), avec $k > 0$, pour en faire une matrice *Trimat*$(m, n, k-1)$. Expliquer comment procéder et donner un algorithme de complexité linéaire $O(m+n)$.

Question II.3. Proposer un algorithme pour trier $p \leq m \times n$ éléments en utilisant les deux algorithmes précédents. Donner et justifier sa complexité.

Partie III. Partitions d'un entier

Une partition d'un entier $n \in \mathbb{N}^*$ est un k-uplet $a = (p_1, p_2, \ldots, p_k)$, avec $p_i \in \mathbb{N}^*$ pour $1 \leq i \leq k$, $p_1 + p_2 + \ldots + p_k = n$ et $p_1 \geq p_2 \geq \ldots \geq p_k$; k est la longueur de la partition.

On définit l'ordre lexicographique inverse, noté $\rhd$, sur les partitions d'un entier : si $a = (p_1, p_2, \ldots, p_k)$ et $a' = (p'_1, p'_2, \ldots, p'_{k'})$ sont deux partitions de n, on pose $a \rhd a'$ si et seulement s'il existe un entier $0 \leq i \leq \min(k, k')$ tel que $p_j = p'_j$ pour $1 \leq j \leq i-1$ et $p_i > p'_i$.

Par exemple pour $n = 5$, $a = (3, 1, 1)$, $b = (2, 1, 1, 1)$, $c = (5)$ et $d = (2, 2, 1)$ sont des partitions, avec $c \rhd a \rhd d \rhd b$.

Question III.1.

1. Donner toutes les partitions de $n = 7$ dans l'ordre $\rhd$.
2. Quelle est la dernière partition de n pour l'ordre $\rhd$?
3. Étant donnée $a = (p_1, p_2, \ldots, p_k)$ une partition de n qui n'est pas la dernière pour l'ordre $\rhd$, déterminer (en justifiant le résultat) la partition suivante pour l'ordre $\rhd$.
4. On utilise un tableau B de taille n pour représenter une partition, puisque que $k \leq n$, en posant :
$$B[i] = \begin{cases} p_i & \text{si } i \leq k, \\ \infty & \text{sinon.} \end{cases}$$
Écrire un algorithme qui génère la partition suivant celle représentée par B dans l'ordre $\rhd$.

Question III.2.

1. Caractériser la dernière partition de n de longueur k, où $1 \leq k \leq n$, pour l'ordre $\rhd$.
2. Etant donnée $a = (p_1, p_2, \ldots, p_k)$ une partition de n de longueur k, et qui n'est pas la dernière des partitions de n de longueur k pour l'ordre $\rhd$, déterminer (en justifiant le résultat) la partition de n de longueur k qui est la suivante de a pour l'ordre $\rhd$.
3. Soit $p(n)$ le nombre de partitions de n, et $p_k(n)$ le nombre de partitions de n de longueur k. On étend la définition de $p_k(n)$ pour tout $k \in \mathbb{Z}$ en posant $p_k(n) = 0$ si $k \leq 0$ ou $k > n$. Montrer que
$$p_k(n) = p_{k-1}(n-1) + p_k(n-k).$$
4. En utilisant la relation $p(n) = \sum_{k=1}^{n} p_k(n)$, montrer que
$$p(n) \leq p(n-1) + p(n-2) \text{ pour } n \geq 3.$$
5. En déduire une majoration de $p(n)$ quand n tend vers l'infini.

Fin du sujet.