Exercice 1 : Médiane rapide

CCINP type A, 2025

On considère le problème $\mathscr{P}$ ci-dessous.

$$\begin{cases}\textbf{Entrée :} \text{ Un tableau } T \text{ contenant } n \text{ éléments deux à deux distincts et } k \in [\![0, n-1]\!] \\ \textbf{Sortie :} \text{ le } k+1 \text{♣-ième élément plus petit de } T.\end{cases}$$

Q. 1 Décrire, à l'oral, un algorithme résolvant $\mathscr{P}$. Quel est sa complexité ? On n'attend pas de code ou de pseudo-code.

On adopte une approche de diviser pour régner pour résoudre ce problème, avec l'algorithme suivant qui, de manière analogue au tri rapide, partitionne le tableau T selon un pivot x en $T_<$, où tous les éléments sont strictement inférieurs à x, et en $T_>$, où tous les éléments sont strictement supérieurs à x.

Algorithme 1 : QuickFind

Entrée : T un tableau de taille $n \in \mathbb{N}^\star$, $k \in [\![0, n-1]\!]$.

```
si n = 1 alors
    retourner T[0]
sinon
    Soit x un élément de T ;
    On considère des tableaux T_<, et T_> tels que T_< contient les éléments de T strictement
     inférieurs à x, et T_> les éléments de T strictement supérieurs à x ;
    si k < |T_<| alors
        retourner QuickFind(T_<, k)
    sinon si k = |T_<| alors
        retourner x
    sinon
        retourner QuickFind(T_>, k − 1 − |T_<|)
```

Q. 2 Expliquer *brièvement* les éléments assurant la correction de cet algorithme.

Q. 3 Que est le coût de la création des deux tableaux $T_<$ et $T_>$? Aucune justification n'est attendue.

Q. 4 Quelle est la complexité pire cas de l'algorithme 1 ?

La complexité dépend en fait grandement du choix du pivot, l'idéal étant de choisir la médiane des valeurs de T. On donne l'algorithme suivant, qui cherche à choisir un pivot qui proche de la médiane.

♣. Pour $k = 0$, on cherche le plus petit élément, pour $k = 1$ on cherche le deuxième plus petit élément, …

Algorithme 2 : MoMFind

Entrée : T un tableau de taille $n \in \mathbb{N}^\star$, $k \in [\![0, n-1]\!]$.
Sortie : Le $k+1$-ième élément le plus petit de T.

1 **si** $n \leqslant 5$ **alors**
2 | Résoudre par force brute ;
3 **sinon**
4 | Diviser le tableau T en $\lceil \frac{n}{5} \rceil$ blocs de taille 5 (sauf éventuellement le dernier) ;
5 | Fabriquer le tableau M de taille $\lceil \frac{n}{5} \rceil$ contenant les médianes des blocs fabriqués à la ligne précédente ;
6 | $x \leftarrow \mathsf{MoMFind}(M, \lfloor \frac{|M|-1}{2} \rfloor)$;
7 | On considère des tableaux $T_<$, et $T_>$ tels que $T_<$ contient les éléments de T strictement inférieurs à x, et $T_>$ les éléments de T strictement supérieurs à x ;
8 | **si** $k < |T_<|$ **alors**
9 | | **retourner** $\mathsf{MoMFind}(T_<, k)$
10 | **sinon si** $k = |T_<|$ **alors**
11 | | **retourner** x
12 | **sinon**
13 | | **retourner** $\mathsf{MoMFind}(T_>, k - 1 - |T_<|)$

On admet la propriété $(\mathcal{Q})$: Le pivot x calculé en ligne 6 est :
- supérieur strictement à au moins $\frac{3n}{10} - 3$ éléments du tableau T ;
- et inférieur strictement à au moins $\frac{3n}{10} - 3$ éléments du tableau T.

On note $C(n)$ le nombre d'opérations effectuées par l'algorithme 2 pour un tableau de taille n.

Q. 5 Montrer qu'il existe $\alpha \in \mathbb{R}^+$ tel que pour tout $n \geqslant 5$:

$$C(n) \leqslant C\left(\left\lceil \frac{n}{5} \right\rceil\right) + C\left(\left\lfloor \frac{7n}{10} \right\rfloor + 2\right) + \alpha n$$

Soit un tel α.

Q. 6 Soit $\beta \geqslant 310\alpha$, $n \geqslant 31$, on suppose que pour tout $k \in [\![1, n-1]\!], C(k) \leqslant \beta k$, montrer alors que $C(n) \leqslant \beta n$.

Q. 7 En déduire la complexité pire cas de l'algorithme 2.

Q. 8 Démontrer la propriété $(\mathcal{Q})$.

Exercice 2 : Matrice standard

CCINP type B, 2025

Cet énoncé est accompagné d'un code compagnon `matrice_standard.c` fournissant certaines des fonctions mentionnées dans l'énoncé : il est à compléter en y implémentant les fonctions demandées.
La ligne de compilation `gcc -o matrice_standard.exe -Wall *.c -lm` vous permet de créer un exécutable `matrice_standard.exe` à partir du ou des fichiers C fournis. Vous pouvez également utiliser l'utilitaire `make`. En ligne de commande, il suffit d'écrire `make`. Dans les deux cas, si la compilation réussit, le programme peut être exécuté avec la commande `./main.exe`.
Il est possible d'activer davantage d'avertissements et un outil d'analyse de la gestion de la mémoire avec la ligne de compilation `gcc -o main.exe -g -Wall -Wextra -fsanitize=address *.c -lm` ou en écrivant `make safe`. L'examinateur pourra vous demander de compiler avec ces options.
Si vous désirez forcer la compilation de tous les fichiers, vous pouvez au préalable nettoyer le répertoire en faisant `make clean` et relancer une compilation.

La compilation du code compagnon initial avec `make safe` provoque des warnings attendus qui seront résolus lors de l'implémentation des fonctions demandées par le sujet.

On prendra garde dans tout l'exercice à libérer l'espace mémoire alloué.
Une matrice $A \in M_{n,m}(\mathbb{N})$, avec $n, m \geqslant 1$, est dite *standard* si les trois conditions suivantes sont respectées.

(H_1) Toutes les lignes sont croissantes, *i.e.* $\forall i \geqslant 1, \forall j \geqslant 2, a_{i,j} \geqslant a_{i,j-1}$.

(H_2) Toutes les colonnes sont croissantes, *i.e.* $\forall i \geqslant 2, \forall j \geqslant 1, a_{i,j} \geqslant a_{i-1,j}$.

(H_3) Tous les entiers de $[\![1, nm]\!]$ apparaissent une et une seule fois dans A.

Q. 1 Identifier parmi les matrices suivantes lesquelles sont standard.

$$A_1 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad A_2 = \begin{pmatrix} 1 & 3 & 4 \\ 2 & 5 & 6 \\ 8 & 7 & 9 \end{pmatrix} \quad A_3 = \begin{pmatrix} 1 & 2 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

Q. 2 Donner les matrices standards de taille 2×2.

Q. 3 Coder la fonction `int** zeros(int nb_l, int nb_c)` qui renvoie une matrice nulle avec `nb_l` lignes et `nb_c` colonnes.

Le fichier `matrice_standard.c` fournit les fonctions `int** initialise(int* t, int nb_l, int nb_c)` et `void affiche(int** A, int nb_l, int nb_c)`, définies comme suit :

- `int** initialise(int* t, int nb_l, int nb_c)` crée une matrice à `nb_l` lignes et `nb_c` colonnes dont les éléments sont ceux de `t`, en remplissant ligne par ligne. Cette fonction repose sur la correction de la fonction `zeros`.
- `void affiche(int** A, int nb_l, int nb_c)` affiche la matrice A dans le terminal.

Q. 4 Écrire une fonction `bool ligne_croissante(int** A, int indice_l, int nb_c)` qui vérifie si la ligne `indice_l` de `A` est triée par ordre croissant.

Q. 5 Écrire une fonction `bool lignes_croissantes(int** A, int nb_c)` qui vérifie si toutes les lignes de `A` sont triées par ordre croissant.

On remarque que pour tester que les colonnes de A sont triées par ordre croissant, il suffit de tester que les colonnes de A^T la transposée sont triées par ordre croissant.

Q. 6 Écrire une fonction `int** transpose(int** A, int nb_l, int nb_c)` qui calcule la transposée de `A`.

Q. 7 À l'aide des fonctions précédentes, écrire une fonction `bool colonnes_croissantes(int** A, int indice_l, int nb_c)`.

Q. 8 Écrire une fonction `bool verif_H3(int** A, int nb_l, int nb_c)` qui vérifie si une matrice A vérifie la propriété (H_3).

Q. 9 Écrire une fonction `bool est_standard(int** A, int nb_l, int nb_c)` permettant de tester si une matrice est standard.

Q. 10 Donner la complexité temporelle de la fonction `est_standard`.