Examen terminal: Algorithmes

L1 MPCI

22 mai 2026 - Durée: 2h

On rappelle qu'aucun document, ni équipement électrique ou électronique n'est autorisé.
Cependant *l'usage d'un coussin péteur sera toléré s'il est placé sur le siège d'un autre étudiant que soit-même.*

Les exercices de cet examen :
- sont au nombre de 2 ;
- sont indépendants ;
- sont chacun constitués de parties de difficultés croissantes.

Rendez des copies séparées pour chaque exercice, ceci vous permettra de d'y revenir au cours de l'examen sans perdre le correcteur.

Exercice 1 – Inversions

On considérera dans cet exercice des tableaux d'entiers T de longueur n. On appelle *inversion* un couple d'indices (i, j) tel que :
- $0 \leq i < j < n$
- $T[i] > T[j]$

Le but de cet exercice est de compter le nombre d'inversions d'un tableau donné.

1.1 Algorithmes naïfs

Question 1.1.1 Combien d'inversions possède le tableau suivant : $T = [12, 2, 1, 2, 4, 1]$

Question 1.1.2 Pour un tableau d'entiers de longueur n quel est le nombre minimum et maximum d'inversions possibles ?

Question 1.1.3 Démontrez que l'algorithme suivant rend le nombre d'inversions du tableau passé en paramètre et donnez sa complexité :

```
algorithme INVERSIONS(T : [entier]) → entier :
    nb ≔ entier ← 0
    pour chaque (i ≔ entier) de [0..T.longueur[ :
        pour chaque (j ≔ entier) de [0..T.longueur[ :
            si (i ≤ j) et (T[i] > T[j]) :
                nb ← nb + 1
    rendre nb
```

Question 1.1.4 Donnez le pseudo-code et la complexité d'un algorithme récursif de signature `INVERSIONS_REC(T: [entier], k: entier) → entier` qui rend le nombre d'inversions (i, j) de T telles que $j \leq k$.

Question 1.1.5 Donnez le pseudo-code et la complexité d'un algorithme de signature `INVERSIONS2(T: [entier]) → entier` utilisant l'algorithme récursif de la question précédente et rendant le nombre d'inversions du tableau passé en paramètre.

1.2 Un petit détour

On suppose que l'on possède deux tableaux d'entiers T_1 et T_2 de longueurs respectives n_1 et n_2 et on cherche à déterminer le nombre de couples (i, j) tels que $T_1[i] > T_2[j]$ avec $0 \leq i < n_1$ et $0 \leq j < n_2$ (on peut avoir $j \leq i$ ici puisque l'on travaille sur 2 tableaux différents).

Question 1.2.1 Donnez le pseudo-code d'un algorithme naïf de signature `INVERSIONS_ENTRE_TABLEAUX(`T_1`: [entier], `T_2`: [entier]) → entier` et de complexité $\mathcal{O}(n_1 \cdot n_2)$ permettant de calculer ce nombre.

Question 1.2.2 Montrez que si les tableaux T_1 et T_2 sont tous deux triés par ordre croissant, on peut faire mieux et calculer le nombre d'inversions entre T_1 et T_2 avec un algorithme de complexité $\mathcal{O}(n_1 + n_2)$ dont vous donnerez le pseudo-code. Il sera de signature `INVERSIONS_ENTRE_TABLEAUX_TRIÉE(`T_1`: [entier], `T_2`: [entier]) → entier`.

1.3 Un algorithme plus efficace

Question 1.3.1 Donnez le pseudo-code et rappelez la complexité du tri fusion (pour simplifier, on supposera que la longueur n du tableau à trier est une puissance de 2 : $n = 2^p$).

Question 1.3.2 En reprenant l'idée du tri fusion donner le pseudo-code d'un algorithme récursif de complexité $\mathcal{O}(n \ln(n))$ (ce que, bien sûr, vous justifierez) permettant de rendre le nombre d'inversions d'un tableau T de longueur n (on supposera encore ici que n est une puissance de 2). Cet algorithme sera de signature `INDICES_REC(T: [entier]) → ([entier], entier)` et rendra un couple constitué d'une copie triée de T et du nombre d'inversions de T.

Exercice 2 – Skip list

Le but de cet exercice est d'étudier la structure de donnée nommée *skip list* définie en 1990 par William Pugh [1].

2.1 Recherche et insertion dans une liste triée

Question 2.1.1 Donnez le pseudo-code (et sa preuve bien sûr) et la complexité de l'algorithme de la recherche dichotomique dans un tableau d'entier trié. Votre algorithme devra être de signature : `RECHERCHE_DICHOTOMIQUE(`T`: [entier],` v`: entier)` $\rightarrow$ `booléen` et rendra vrai si v est dans T et faux sinon.

Question 2.1.2 On remplace souvent les tableaux par la structure de donnée **des listes**. Explicitez les différences entre les deux structures et pourquoi il est (presque) toujours plus avantageux d'utiliser des listes plutôt que des tableaux.

Question 2.1.3 Donnez le pseudo-code et la complexité d'un algorithme ajoutant un élément dans une liste triée. Votre algorithme modifiera la liste passée en entrée et elle devra rester triée. Enfin, la complexité spatiale de votre algorithme devra être en $\mathcal{O}(1)$ (vous ne comptez pas la complexité spatiale de la liste passée en paramètre).

Question 2.1.4 Plutôt que d'utiliser des listes on peut aussi utiliser des listes doublement chaînées définies comme une succession de maillons.

structure MAILLON **:**
> valeur : T $\leftarrow \varnothing$
> précédent : Maillon $\leftarrow \varnothing$
> suivant : Maillon$\leftarrow \varnothing$

Donnez le pseudo-code et la complexité minimale, maximale et en moyenne d'un algorithme ajoutant un élément dans une liste doublement chaînée triée avant et après l'ajout.

Question 2.1.5 Quels sont les avantages et inconvénients de ces deux structures pour rechercher/ajouter/supprimer des entiers dans une structure devant rester triée ?

2.2 *Skip list* régulière

La figure 1 montre une représentation d'une *skip list* régulière. Une *skip list* est une généralisation d'une liste chaînée

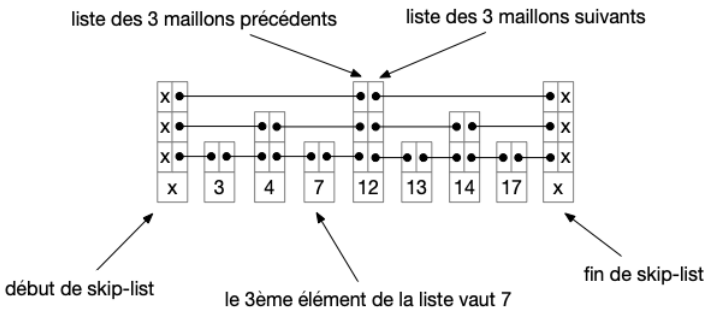

Figure 1 – Une skip-list régulière.

où chaque maillon possède une liste de maillons précédents et une liste de maillons suivants. Cette structure suit les règles suivantes :

- la longueur des listes suivants et précédents est la même pour un maillon donné et est appelée niveau (le deuxième élément de la figure 1 possède 2 niveaux et le troisième élément 1 seul niveau)
- pour un maillon donné, le $i$ème élément de sa liste suivant (respectivement de sa liste précédent) vaudra le prochain (*resp.* précédent) maillon de la liste de niveau i ou plus.
- une skip list possède un premier maillon qui débute la liste et un maillon qui termine la liste. Ces deux maillons ne possèdent pas de valeurs et sont de niveau maximum noté K.

Une *skip list* sera dite *régulière* si le $i$ème élément de la liste ($1 \leq i \leq n$) est de niveau $k+1$ avec k le plus grand entier tel que 2^k divise i.

1. William Pugh, *Skip lists : a probabilistic alternative to balanced trees*, Communications of the ACM, 33, 6 pp 668 - 676.

Question 2.2.1 Montrer que la figure 1 représente une *skip list* régulière.

Question 2.2.2 Modifiez la structure de maillon de la question 2.1.4 pour permettre de construire une structure MAILLON_SL d'un maillon de *skip list*.

Question 2.2.3 Utilisez la structure que vous avez créée à la question précédente pour donner le pseudo-code d'un algorithme de signature `CRÉE_MAILLON(`v`: entier,` k`: entier)` $\rightarrow$ `MAILLON_SL` qui rendre un maillon de *skip list* de niveau $k \geq 1$.

Question 2.2.4 Donnez le pseudo-code et la complexité d'un algorithme cherchant une valeur dans une *skip list*. Il devra être de signature `RECHERCHE(`slr`: MAILLON_SL,` v`: entier)` $\rightarrow$ `booléen` et rendra vrai si la valeur v est présente dans la *skip list* régulière *slr* et faux sinon.

Question 2.2.5 Donnez un argument qui justifie le fait que maintenir la structure de *skip list* régulière dans le cas de l'ajout (respectivement la suppression) d'un élément nécessite au minimum de parcourir tous les éléments après l'élément ajouté (*resp.* supprimé).

Question 2.2.6 Déduisez des deux questions précédentes la complexité de la recherche, de l'ajout et de la suppression d'un élément dans une *skip list* régulière.

2.3 *Skip list* quelconque

Une *skip list* ne spécifie pas le nombre de maillons suivants que doit posséder un élément. La figure 2 représente une *skip list* valide. Lors de l'ajout d'un élément dans la *skip list* son niveau k est déterminé par l'algorithme `NOMBRE` qui utilise la fonction `RANDOM()` $\rightarrow$ `réel` qui rend un réel aléatoire choisi uniformément dans $[0, 1]$.

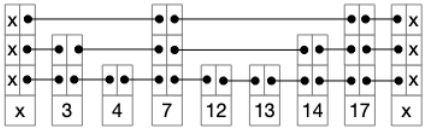

FIGURE 2 – Une skip-list.

algorithme NOMBRE(K : entier) $\rightarrow$ entier **:**
  $k :=$ entier $\leftarrow 1$
  tant que $(k < K)$ **et** RANDOM()$\leq 1/2$ **:**
    $k \leftarrow k + 1$
  rendre k

Question 2.3.1 Donnez la complexité minimale et maximale de l'algorithme de la question 2.2.4 appliqué à une *skip list* quelconque.

Question 2.3.2 Donnez le pseudo-code d'un algorithme insérant une valeur dans une *skip list*.

2.4 Estimation des complexités moyennes

Question 2.4.1 Pour une *skip list* donnée, combien y a-t-il en moyenne de maillons de niveau k ?

Question 2.4.2 Pour une *skip list* donnée, combien y a-t-il en moyenne de maillons de niveau strictement inférieur à k entre 2 maillons successifs de niveau k ou plus ?

Question 2.4.3 Estimez la complexité en moyenne de l'algorithme de la question 2.2.4 appliqué à une *skip list* quelconque.

Question 2.4.4 Si vous aviez n éléments à stocker dans une *skip list*, quelle valeur donneriez vous (et pourquoi) à son niveau maximum K ?