

L'utilisation de la calculatrice est interdite pour ce devoir.

Ce devoir est d'une durée de 2h, et est composé de 2 parties indépendantes :

- la partie 1 porte sur le problème de la somme partielle, et est à traiter dans le langage C ;
- la partie 2 porte sur l'algorithme de la médiane des médianes, et est à traiter dans le langage OCaml.

Le barème tient compte de la clarté et de la présentation de votre copie. On prendra garde à indiquer clairement le numéro des questions, et, autant que possible, à rédiger les questions dans l'ordre. Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat.

1 Somme et partition de multi-ensembles

Remarque : dans cette partie, les fonctions demandées sont à écrire dans le langage C.

Définition 1.1 : multi-ensemble

Un **multi-ensemble** est une sorte d'ensemble où chaque élément peut apparaître plusieurs fois.

La **multiplicité** (ou le **nombre d'occurrences**) d'un élément x d'un multi-ensemble S est le nombre de fois qu'apparaît x dans S .

Soit S_1, S_2, S trois multi-ensembles. On note :

- $S_1 \subseteq S$ si chaque élément de S_1 apparaît au moins autant de fois dans S que dans S_1 ;
- $S = S_1 \uplus S_2$ si $S_1 \subseteq S$ et $S_2 \subseteq S$ et que chaque élément de S apparaît autant de fois dans S qu'au cumulé dans S_1 et S_2 (dans ce cas, on dit que S_1 et S_2 forment une **partition** de S).

Exemple 1.1

On a $\{1, 0, 0, 2\} \subseteq \{0, 1, 3, 0, 2, 5, 3, 1, 0, 4\}$, et $\{0, 1, 3, 0, 2, 5, 3, 1, 0, 4\} = \{1, 0, 0, 2\} \uplus \{4, 0, 1, 5, 3, 3\}$

Dans cette partie, on s'intéresse à la résolution des problèmes PARTITION et SUBSET-SUM suivants :

PARTITION

Entrée : un multi-ensemble fini S d'entiers positifs

Question : Peut-on partitionner $S = S_1 \uplus S_2$ en deux multi-ensembles S_1 et S_2 tels que $\sum_{x \in S_1} x = \sum_{x \in S_2} x$?

SUBSET-SUM

Entrée : un multi-ensemble fini S d'entiers positifs, un entier $t \in \mathbb{N}$

Question : Existe-t-il $S_1 \subseteq S$ tel que $\sum_{x \in S_1} x = t$?

1.1 Fonctions préliminaires

Nous allons représenter en C un multi-ensemble S par la structure suivante :

```
struct multiens {
    int taille;
    int* elements;
};
typedef struct multiens multiens;
```

Si S est une variable de type `multiens` représentant un multi-ensemble S , alors :

- `S.taille` contient le nombre d'éléments de S (comptés avec leurs multiplicités);
 - `S.elements` pointe vers le début d'un tableau de taille `S.taille`, contenant les éléments de S .
1. Écrire une fonction `bool tous_positifs(multiens S)`; prenant en argument un multi-ensemble S et renvoyant `true` si tous les éléments de S sont positifs (et `false` sinon).
 2. Écrire une fonction `int somme(multiens S)`; prenant en argument un multi-ensemble S et renvoyant la somme des éléments de S .
 3. Écrire une fonction `int occurrences(multiens S, int x)`; prenant en arguments un multi-ensemble S et un entier x , et renvoyant le nombre d'occurrences de x dans S .
 4. Écrire une fonction `bool est_sous_ensemble(multiens S, multiens S1)`; prenant en arguments deux multi-ensembles S et S_1 et renvoyant `true` si $S_1 \subseteq S$ (et `false` sinon).
 5. Écrire une fonction `bool est_partition(multiens S, multiens S1, multiens S2)`; prenant en arguments trois multi-ensembles S , S_1 et S_2 et renvoyant `true` si $S = S_1 \uplus S_2$ (et `false` sinon).

1.2 Résolution naïve

Une manière naïve de résoudre SUBSET-SUM serait de tester tous les sous-ensembles possibles S_1 de S jusqu'à en trouver un tel que $\sum_{x \in S_1} x = t$. De plus, plutôt que de tester absolument tous les sous-ensembles S_1 possibles, on peut se restreindre à ceux dont les éléments du tableau `S1.elements` apparaissent dans le même ordre que dans `S.elements`.

6. (a) Si S est un multi-ensemble de taille n , combien y a-t-il de sous-tableaux possibles de `S.elements` dont les éléments apparaissent dans le même ordre que dans `S.elements`?
- (b) Quelle serait la complexité temporelle dans le pire des cas de cette approche naïve?

On fournit le code de la fonction suivante :

```

1 bool sous_somme(multiens S, int t, int n_max){
2     if (t == 0){
3         return true;
4     }
5     if (t < 0 || n_max < 0){
6         return false;
7     }
8     return sous_somme(S, t - S.elements[n_max], n_max-1) || sous_somme(S, t, n_max-1);
9 }
```

On introduit la notation suivante : si `tab` est un tableau et i et j sont deux indices, `tab[i:j]` désigne les éléments de `tab` dont les indices sont dans $\llbracket i, j-1 \rrbracket$.

7. Montrer que la fonction `sous_somme(S,t,n_max)` termine, pour tout multi-ensemble S d'entiers positifs et tous entiers $n_{\max} < S.taille$ et $t \in \mathbb{N}$.
8. Montrer que pour tout multi-ensemble S d'entiers positifs et tous entiers $n_{\max} < S.taille$ et $t \in \mathbb{N}$, `sous_somme(S,t,n_max)` renvoie `true` si et seulement s'il existe un sous-tableau de `S.elements[0:n_max+1]` dont la somme des éléments vaut t .
9. (a) À l'aide de la fonction `sous_somme`, écrire une fonction `bool partition(multiens S)`; prenant en argument un multi-ensemble S d'entiers positifs, et renvoyant `true` s'il est possible de partitionner $S = S_1 \uplus S_2$ en deux multi-ensembles tels que $\sum_{x \in S_1} x = \sum_{x \in S_2} x$ (et `false` sinon).
- (b) Quelle est la complexité temporelle de votre fonction `partition` dans le pire des cas?

1.3 Résolution par programmation dynamique

On cherche maintenant à améliorer la complexité dans le pire des cas de la fonction `sous_somme` en utilisant de la programmation dynamique. Pour cela, si S est un multi-ensemble de taille n , on va construire une matrice de booléens T de taille $(n + 1) \times (t + 1)$ telle que :

$$T[i][j] = \begin{cases} \text{true} & \text{s'il existe un sous-ensemble de } S.\text{elements}[0:i] \text{ dont la somme vaut } j; \\ \text{false} & \text{sinon.} \end{cases}$$

Cette matrice T sera implémentée par un tableau de type `bool**`.

10. Écrire une fonction `bool** init_false(int n, int m)`; prenant en arguments deux entiers positifs n et m , et renvoyant une matrice T de taille $n \times m$ dont toutes les cases sont initialisées à `false`.
 11. Écrire une fonction `void free_matrice(bool** T, int n, int m)`; libérant une telle matrice T (de taille $n \times m$) de la mémoire.
 12. Écrire une fonction `bool sous_somme_dyna(multiens S, int t)`; prenant en arguments un multi-ensemble S d'entiers positifs et une valeur $t \in \mathbb{N}$ et renvoyant `true` s'il existe un sous-ensemble de S dont la somme des éléments vaut t (et `false` sinon).
- Attention :** votre fonction devra avoir une complexité polynomiale en $S.\text{taille}$ et t .
13. En déduire une fonction `bool partition_dyna(multiens S)`; prenant en argument un multi-ensemble S d'entiers positifs, et renvoyant `true` s'il est possible de partitionner $S = S_1 \uplus S_2$ en deux multi-ensembles S_1 et S_2 tels que $\sum_{x \in S_1} x = \sum_{x \in S_2} x$ (et `false` sinon).

Attention : votre fonction devra avoir une complexité polynomiale en $S.\text{taille}$ et $\sum_{x \in S} x$.

2 Tri rapide et algorithme de la médiane des médianes

Remarque : dans cette partie, les fonctions demandées sont à écrire dans le langage OCaml.

Vous avez vu en cours que l'algorithme du tri rapide à une complexité en $\mathcal{O}(n^2)$ dans le pire des cas. On peut montrer que sa complexité en moyenne est en $\mathcal{O}(n \log n)$. Le but de ce devoir est de mieux choisir le pivot pour obtenir une complexité en $\mathcal{O}(n \log n)$ dans le pire des cas.

2.1 Le tri rapide

On rappelle le principe du tri rapide sur les tableaux. Soit t un tableau, et $g < d$ des indices de ce tableau, on note $t[g : d]$ la portion de t comprise entre les indices g inclus et d exclu. Par exemple, si t est de taille n , la portion totale de t est $t[0 : n]$. Pour cet algorithme de tri, on va commencer par **partitionner** une portion $t[g : d]$ de notre tableau selon un **pivot** (qui est une valeur p comparable aux éléments de notre tableau). À la fin de cette partition, on a échangé des cases du tableau pour qu'il existe deux indices i_s et i_e tels que :

- $\forall g \leq i < i_s, t.(i) < p$;
- $\forall i_s \leq i < i_e, t.(i) = p$;
- $\forall i_e \leq i < d, t.(i) > p$.

Remarque : contrairement à la version vue en cours, on partitionne ici en 3 morceaux.

14. Écrire une fonction `echange : 'a array -> int -> int -> unit` prenant en arguments un tableau t et deux entiers i et j , et qui échange par effets de bords les valeurs de $t.(i)$ et $t.(j)$.
15. Écrire une fonction `partition_en_3 : 'a array -> 'a -> int -> int -> int * int` prenant en arguments un tableau t , un pivot p , et deux indices g et d , et qui effectue la partition décrite ci-dessus (en modifiant le tableau t par effets de bords) et renvoie les valeurs des indices (i_s, i_e) .
Votre algorithme devra avoir une complexité **linéaire**.

16. (a) Écrire une fonction `tri_rapide1` : `'a array -> unit` qui trie par effets de bords un tableau pris en argument et ne renvoie rien. Votre algorithme devra reposer sur le tri rapide et choisir l'élément le plus à gauche de la portion à trier comme pivot.
- (b) Quelle est la complexité de votre algorithme dans le pire des cas ?

2.2 Calcul de la médiane d'un tableau

On cherche à améliorer la complexité du tri rapide dans le pire des cas. Pour cela, on va essayer de choisir un pivot garantissant de partitionner notre tableau en deux morceaux de même taille (à un élément près). Pour cela, on pourrait chercher la médiane du tableau, c'est à dire le k -ème plus petit élément du tableau, où k est la moitié de la taille du tableau.

17. Dans cette question, on suppose l'existence d'une fonction `mediane` : `'a array -> int -> 'a` telle que `mediane t k` renvoie la valeur du k -ème plus petit élément de `t` avec une complexité $f(n)$ où n est la taille de `t`. On s'intéresse ici à la complexité d'une version du tri rapide qui utiliserait la fonction `mediane` pour choisir le pivot.
- (a) Quelle serait la complexité de cet algorithme de tri rapide si $f(n) = \mathcal{O}(n)$?
- (b) Quelle serait la complexité de cet algorithme de tri rapide si $f(n) = \mathcal{O}(n^2)$?

Pour implémenter la fonction `mediane`, on utilise une approche qui s'inspire du tri rapide (mais qui n'effectue qu'un seul appel récursif au lieu de deux) :

- on fait une copie `t'` de `t` avec l'instruction `let t' = Array.copy t in ...` ;
- on partitionne `t'` à l'aide de la fonction `partition_en_3` ;
- après cette partition :
 - si $i_s = k$ ou $(i_s < k < i_e)$, alors `t'.`(`k`) est l'élément qu'on cherche ;
 - sinon, si $k < i_s$, alors l'élément qu'on recherche est dans la partie strictement à gauche de i_s ;
 - sinon, l'élément qu'on cherche est dans la partie à droite de i_e .

Dans ces deux derniers cas, on ne fait qu'un seul appel récursif (dans le morceau de `t'` qui nous intéresse).

18. (a) Implémenter une fonction `mediane` utilisant le principe précédent.
- (b) Quelle est la complexité de votre programme dans le pire des cas ?

2.3 L'algorithme de la médiane des médianes

On pourrait montrer que l'algorithme précédent a une complexité linéaire en moyenne, mais on va maintenant s'intéresser à une méthode fournissant une complexité linéaire dans le pire des cas. Pour cela, on va également optimiser le choix du pivot lors du calcul de la médiane. Le principe est le suivant (pour un tableau `t` de taille n) :

- (i) on regroupe les éléments de `t` en $\frac{n}{5}$ groupes de 5 éléments ;
 - (ii) on calcule (en temps constant) la médiane de chacun de ces groupes de 5 : $m_1, m_2, \dots, m_{\frac{n}{5}}$;
 - (iii) on calcule récursivement la médiane de ces médianes ;
 - (iv) on utilise cette valeur comme pivot pour partitionner `t`.
19. Avant d'implémenter l'algorithme, voyons pourquoi ce choix de pivot va améliorer la complexité du calcul de la médiane dans le pire cas.
- (a) Soit p le pivot obtenu (la médiane des médianes). Montrer qu'au moins 30% des éléments de `t` sont inférieurs à p , et qu'au moins 30% des éléments de `t` sont supérieurs à p .
- (b) Dans le pire des cas, quelle sera alors la taille de la portion de `t` où s'effectuera l'appel récursif ?
- (c) Notons $C(n)$ la complexité de l'algorithme `mediane` sur un tableau de taille n . Justifier que $C(n) \leq C(\frac{1}{5} \cdot n) + C(\frac{7}{10} \cdot n) + \alpha \cdot n$, où α est une constante qu'on n'essaiera pas d'exprimer.
- (d) Montrer que $\forall n \in \mathbb{N}, C(n) \leq 10 \cdot \alpha \cdot n$.

20. Écrire une fonction `partition5 : 'a array -> int -> int -> int` prenant en arguments un tableau `t` et deux indices `g` et `d`, et qui cherche la médiane des éléments `t.(g)`, `t.(g+1)`, ..., `t.(d)`, et renvoie l'indice où il se trouve (on aura le droit de modifier cette portion du tableau `t` par effets de bords).

Indication : en pratique, on n'utilisera cette fonction que si $0 \leq d - g \leq 5$, on ne se préoccupera donc pas de sa complexité asymptotique. On pourra par exemple trier la portion du tableau avec un algorithme de tri naïf (tri par sélection ou par insertion), et renvoyer l'indice du milieu de la portion.

On fournit le code des deux fonctions mutuellement récurives suivantes :

```

1  let rec pivot t g d = match d-g < 5 with
2    | true -> partition5 t g d
3    | false ->
4      let i = ref g in
5      while !i <= d do
6        let sd = min (!i+4) d in
7        let median5 = partition5 t !i sd in
8        let mi = g + (!i-g)/5 in
9        echange t median5 mi ;
10       i := !i + 5
11     done ;
12     let mid = g + (d-g)/10 + 1 in
13     let d' = g + (d-g)/5 in
14     select t g d' mid
15 and select t g d k = match g = d with
16   | true -> g
17   | false ->
18     let ip = pivot t g d in
19     let p = t.(ip) in
20     let i_s, i_e = partition_en_3 t p g d in
21     match i_s = k || (i_s < k && k < i_e) with
22     | true -> k
23     | false when k < i_s -> select t g (i_s-1) k
24     | _ -> select t i_e d k

```

21. (a) Quel est le type des fonctions `pivot` et `select` ?
 (b) Expliquer le rôle de chacune de ces fonctions vis-à-vis de l'algorithme de la médiane des médianes (sans prouver formellement leur correction).
 (c) Montrer que les fonctions `pivot` et `select` terminent.
 (d) Quelle est la complexité de la fonction `select` dans le pire des cas ?
22. Écrire une fonction `selection_rapide : 'a array -> int -> 'a` prenant en arguments un tableau `t` et en entier `k`, et qui renvoie le k -ème plus petit élément du tableau `t`.
Attention : on prendra garde à ne **pas** modifier `t` par effets de bords.
23. Écrire une fonction `tri_rapide2 : 'a array -> unit` qui trie par effets de bords un tableau pris en argument et ne renvoie rien.
 Votre algorithme devra reposer sur le tri rapide et choisir la médiane de la portion à trier comme pivot.
 Votre algorithme devra avoir une complexité en $\mathcal{O}(n \log n)$ dans le pire des cas.