Devoir Surveillé 6

3 mai 2025

INFORMATIQUE MP2I

DURÉE DE L'ÉPREUVE: **4 heures**

L'usage de la calculatrice et de tout dispositif électronique est interdit.

Ce sujet comporte huit pages numérotées de 1/8 à 8/8

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Tournez la page S.V.P.

Vue d'ensemble du sujet

Ce sujet est composé de 3 parties indépendantes, utilisant les langage de programmation C et OCaml.

Les différentes parties sont indépendantes et peuvent être traités dans n'importe quel ordre. Il n'est pas nécessaire d'avoir répondu aux questions d'une partie pour répondre aux questions d'une autre partie.

— La partie I, basée sur le sujet Centrale MP 2023 Option Info, s'intéresse à la représentation en machine d'ensemble de nombres, en analysant dans un premier temps 3 approches classiques : listes, vecteurs et arbres binaires de recherches, puis dans un second temps en proposant une approche basée sur les arbres binaires complets, et enfin dans un troisième temps une approche basée sur des arbres complets d'arité variable.

— La partie II s'intéresse au problème de la chaîne de multiplications de matrices, et en propose une résolution par programmation dynamique.

Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat. En langage C, il est inutile de rappeler que les entêtes `<assert.h>`, `<stdbool.h>`, etc. doivent être inclus. En OCaml, les noms des arguments des fonctions sont donnés avec des indications de type. Il n'est pas nécessaire de recopier ces indications de type sur votre copie.

Quand l'énoncé demande de coder une fonction, sauf indication explicite de l'énoncé, il n'est pas nécessaire de justifier que celle-ci est correcte ou de tester que des préconditions sont satisfaites.

Le barème tient compte de la clarté des programmes : nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrit le rôle.

1 Différentes implémentation d'ensembles de nombres – 35 + 40 pts

Dans cette section, on s'intéresse à des structures de données représentant des ensembles d'entiers naturels E contenus dans $[\![0, N-1]\!]$, « denses » au sens où $|E|$ est du même ordre de grandeur que N. On cherche à optimiser le temps d'exécution des opérations de test d'appartenance, d'insertion ou de suppression d'un élément, de calcul du minimum, de calcul de l'élément de E immédiatement supérieur à x (qu'on appellera successeur de x dans E, même si x n'appartient pas à E) ou d'opérations ensemblistes telles que l'union.

☐ 1 – Donner les successeur de 5 dans les ensembles suivants :

— $E_0 = \{1; 3; 5; 7\}$

— $E_1 = \{2; 3; 6; 8\}$

— $E_2 = \{10; 15; 20\}$

1.1 Avec une liste triée – OCaml

☐ 2 – Dans cette question uniquement, on implémente un ensemble d'entiers positifs par la liste de ses éléments, rangés dans l'ordre croissant. Écrire une fonction `succ_list` de signature `int list -> int -> int` prenant en arguments une liste d'entiers distincts dans l'ordre croissant et un entier x et renvoyant le successeur de x dans la liste, c'est-à-dire le plus petit entier strictement supérieur à x de la liste (-1 si cela n'existe pas). Donner sa complexité dans le pire cas.

1.2 Avec un vecteur trié – C

Soit N un entier naturel strictement positif, fixé pour toute cette partie. On choisit de représenter un ensemble d'entiers E de cardinal $n \leqslant N$ par un tableau `t` de taille $N+1$ dont la case d'indice 0 indique le nombre n d'éléments de E et les cases d'indices 1 à n contiennent les éléments de E rangés dans l'ordre croissant, les autres cases étant non significatives. Par exemple, le tableau `, 2, 5, 7, 9, 1, 143, 2, 5, 7, 9, 1, 14` représente l'ensemble à 3 éléments $\{2, 5, 7\}$.

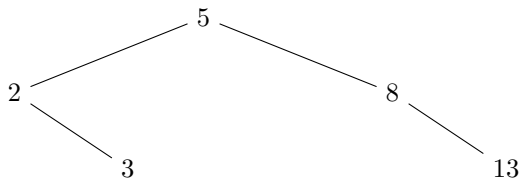

Figure 1: Un arbre binaire de recherche pour l'ensemble $\{2, 3, 5, 8, 13\}$

☐ 3 – Pour une telle implémentation d'un ensemble E, décrire brièvement des méthodes permettant de réaliser chacune des opérations ci-dessous (on ne demande pas d'écrire des programmes) et donner leurs complexités dans le pire cas :

— déterminer le maximum de E;

— tester l'appartenance d'un élément x à E;

— ajouter un élément x dans E (on suppose la taille du tableau suffisante et que x n'appartient pas à E).

☐ 4 – Par une méthode dichotomique, écrire une fonction `int succ_vect(int t[], int x)` prenant en arguments un tableau `t` codant un ensemble E comme ci-dessus et un entier `x` et renvoyant le successeur de `x` dans E (-1 si cela n'existe pas).

☐ 5 – Calculer la complexité dans le pire cas de la fonction `succ_vect` en fonction de n.

☐ 6 – Écrire une fonction `int *union_vect(int N, int t_1[], int t_2[])` prenant en arguments deux tableaux `t_1` et `t_2`, de taille N, codant deux ensembles E_1 et E_2 et renvoyant le tableau correspondant à $E_1 \cup E_2$. On supposera que $|E_1 \cup E_2| \leqslant N$.

1.3 Avec un arbre binaire de recherche – OCaml

On choisit maintenant de représenter un ensemble d'entiers à l'aide d'un arbre binaire de recherche, les nœuds étant étiquetés par les éléments de E. On définit le type :

```
type abr = Nil | Noeud of int * abr * abr
```

Pour un tel arbre, pour tout nœud x, les étiquettes de tous les nœuds du sous-arbre gauche de x, s'il y en a, doivent être inférieures à l'étiquette de x et les étiquettes de tous les nœuds du sous-arbre droit de x, s'il y en a, doivent être supérieures à l'étiquette de x. Par exemple, on peut représenter l'ensemble $\{2, 3, 5, 8, 13\}$ par l'arbre figure 1.

☐ 7 – Écrire une fonction `min_abr` de signature `abr -> int` prenant en argument un arbre binaire de recherche représentant un ensemble E et renvoyant son étiquette minimale (-1 si l'ensemble est vide).

☐ 8 – Écrire une fonction récursive `partitionne_abr` de signature `abr -> int -> (bool * abr * abr)` prenant en arguments un arbre binaire de recherche représentant un ensemble E et un entier x et renvoyant un triplet `(b, ag, ad)` où `b` vaut true si x appartient à E et false sinon, `ag` est un arbre binaire de recherche codant les éléments de E strictement plus petits que x et `ad` un arbre binaire de recherche codant les éléments de E strictement plus grands que x.

☐ 9 – Écrire une fonction `insertion_abr` de signature `abr -> int -> abr` prenant en arguments un arbre binaire de recherche représentant un ensemble E et un entier x et renvoyant un arbre binaire de recherche associé à l'ensemble $E \cup \{x\}$ et de racine étiquetée par x. Calculer sa complexité dans le pire cas en fonction de l'arbre reçu puis en fonction de E.

☐ 10 – Écrire une fonction `union_abr` de signature `abr -> abr -> abr` prenant en arguments deux arbres binaires de recherche représentant deux ensembles E_1 et E_2 et renvoyant un arbre binaire de recherche associé à l'ensemble $E_1 \cup E_2$. On expliquera brièvement la méthode choisie.

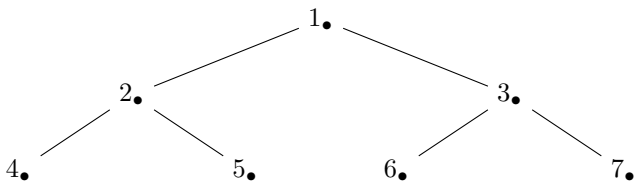

Figure 2: Numérotation des nœuds

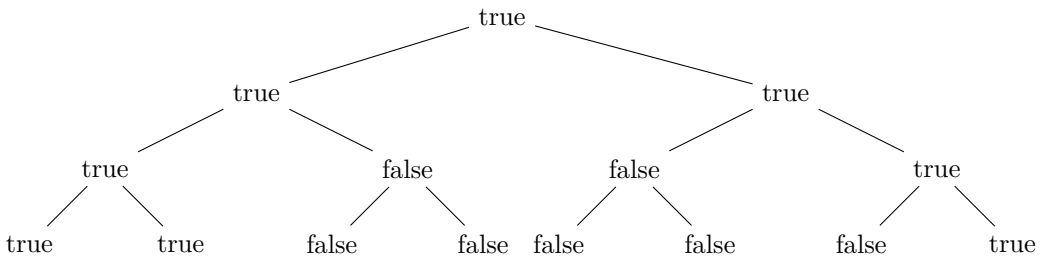

Figure 3: Arbre associé à l'ensemble $\{0, 1, 7\}$ pour $p = 3$

1.4 Avec un arbre binaire complet – C (40 pts)

On considère dans cette partie des arbres binaires complets dont les nœuds sont étiquetés par des booléens. On rappelle que la profondeur d'un nœud est égale à la longueur du chemin reliant la racine à ce nœud et la hauteur d'un arbre est la plus grande profondeur de ses feuilles (la racine est donc à la profondeur 0). Les nœuds seront numérotés à partir de la racine qui porte le numéro 1 dans l'ordre d'un parcours en largeur (de gauche à droite à chaque profondeur).

☐ 11 – Dans un arbre binaire complet de hauteur $p \in \mathbb{N}$, quel numéro peut avoir un nœud à la profondeur $k \in [\![0, p]\!]$? Combien le sous-arbre dont la racine a le numéro i a-t-il de feuilles ?

☐ 12 – Dans un arbre binaire complet de hauteur $p \in \mathbb{N}$, pour le nœud numéro i, donner, en les justifiant, les numéros de son fils gauche, de son fils droit et de son père.

Informatiquement, un tel arbre complet de hauteur p sera implémenté par un tableau de booléens de taille 2^{p+1}, la case d'indice $i \in [\![1, 2^{p+1} - 1]\!]$ contenant l'étiquette du nœud numéroté i. La case d'indice 0 n'est pas utilisée. La taille du tableau 2^{p+1} sera toujours passé en paramètre avec le tableau et notée `len`.
Soit $p \in \mathbb{N}$. On choisit de coder un ensemble $E \subset [\![0, 2^p - 1]\!]$ par un arbre binaire complet de hauteur p selon les règles suivantes:

— chaque feuille numérotée i est étiquetée par true si et seulement si $i - 2^p \in E$;

— chaque nœud interne est étiqueté par le « ou logique» des étiquettes de ses deux fils.

```
typedef bool * ensemble;
```

Par exemple, l'ensemble $\{0, 1, 7\}$ est représenté par l'arbre représenté figure 3 et par le tableau |true, true, true, true, false, fale, true, true, true, false, false, false, false, false, true|.

☐ 13 – Écrire une fonction `bool appartient(ensemble E, int len, int x)` qui détermine si un entier quelconque appartient ou non à un ensemble donné. Calculer sa complexité.

☐ 14 – Écrire une fonction `ensemble fabrique(int tab[], int size, int 2_to_p)` qui prend en arguments un tableau `tab` d'entiers positifs distincts de taille `size` et une valeur pour 2^p et renvoie l'arbre associé à l'ensemble E dont les éléments sont ceux de `tab`. On supposera que tous les éléments de `tab` appartiennent à $[\![0, 2^p - 1]\!]$. Cette fonction devra s'exécuter en $\mathcal{O}(2^p)$.

☐ 15 – Écrire une fonction `void insere(ensemble E, int len, int k)` qui ajoute un entier k à un ensemble E. On suppose k compatible avec la valeur de p associée à E. Cette fonction devra s'exécuter en $\mathcal{O}(1)$ dans le meilleur cas.

☐ 16 – Écrire une fonction `void supprime(ensemble E, int len, int k)` qui retire un entier k d'un ensemble E. On suppose k compatible avec la valeur de p associée à E. Calculer la complexité de cette fonction dans le pire cas.

☐ 17 – Écrire une fonction `int minlocal(ensemble E, int len, int i)` qui cherche l'élément de E minimal parmi ceux codés dans le sous-arbre de racine numérotée i dans l'arbre associé à E. Si un tel élément n'existe pas, cette fonction devra renvoyer -1. Calculer la complexité de cette fonction en fonction de p et i.

On veut maintenant écrire une fonction qui calcule le successeur d'un entier x dans un ensemble E pour une telle structure. On propose l'algorithme suivant, pour $x \in [\![0, 2^p - 1]\!]$:

— on part de la case numéro i codant l'entier x dans E ;

— tant que i n'est pas le nœud le plus à droite à sa profondeur et que la case $i + 1$ vaut `false`, on remplace i par son père ;

— on renvoie l'élément minimum du sous-arbre de racine $i + 1$ ou -1 si i était totalement à droite.

☐ 18 – Prouver l'algorithme décrit ci-dessus.

☐ 19 – Écrire une fonction `int successeur(ensemble E, int len, int x)` prenant en arguments l'ensemble E et un entier x positif et renvoyant son successeur dans E (-1 si cela n'existe pas).

☐ 20 – Montrer que si $x \in E$ admet bien un successeur dans E, il existe une constante $K > 0$ indépendante de E et p telle que la complexité de `successeur(E, len, x)` soit majorée par $K(\log_2(\texttt{successeur}(x) - x) + 2)$. On admet que le même type de justification montre que si x est le maximum de E, la complexité de `successeur(E, len, x)` est majorée par $K(\log_2(2^p - x) + 2)$.

☐ 21 – **En utilisant la fonction `successeur`**, écrire une fonction `int cardinal(ensemble E, int len)` prenant en argument un ensemble et renvoyant son cardinal.

☐ 22 – Déterminer la complexité de la fonction `cardinal` en fonction de p et $n = |E|$. On rappelle que la fonction $\log_2$ est concave.

☐ 23 – Quels sont les intérêts et inconvénients d'une telle structure ? Dans quels cas peut-elle s'avérer plus intéressante que des structures connues ?

1.5 Arbres de van Emde Boas

Soit p un entier positif et $N = 2^{2^p}$. On supposera que tous les entiers manipulés restent représentables par la structure d'entier OCaml ordinaire. On considère le type de structure suivant (appelé arbre *veb* par suite) implémentant un ensemble E d'entiers positifs strictement inférieurs à N.

```
type veb = {mutable mini : int; mutable maxi : int; table : veb array}
```

Les champs mutables d'un arbre *veb* son modifiables : par exemple, pour un arbre *veb* noté `v`, `v.mini <- 1` change la valeur du champ `v.mini`.

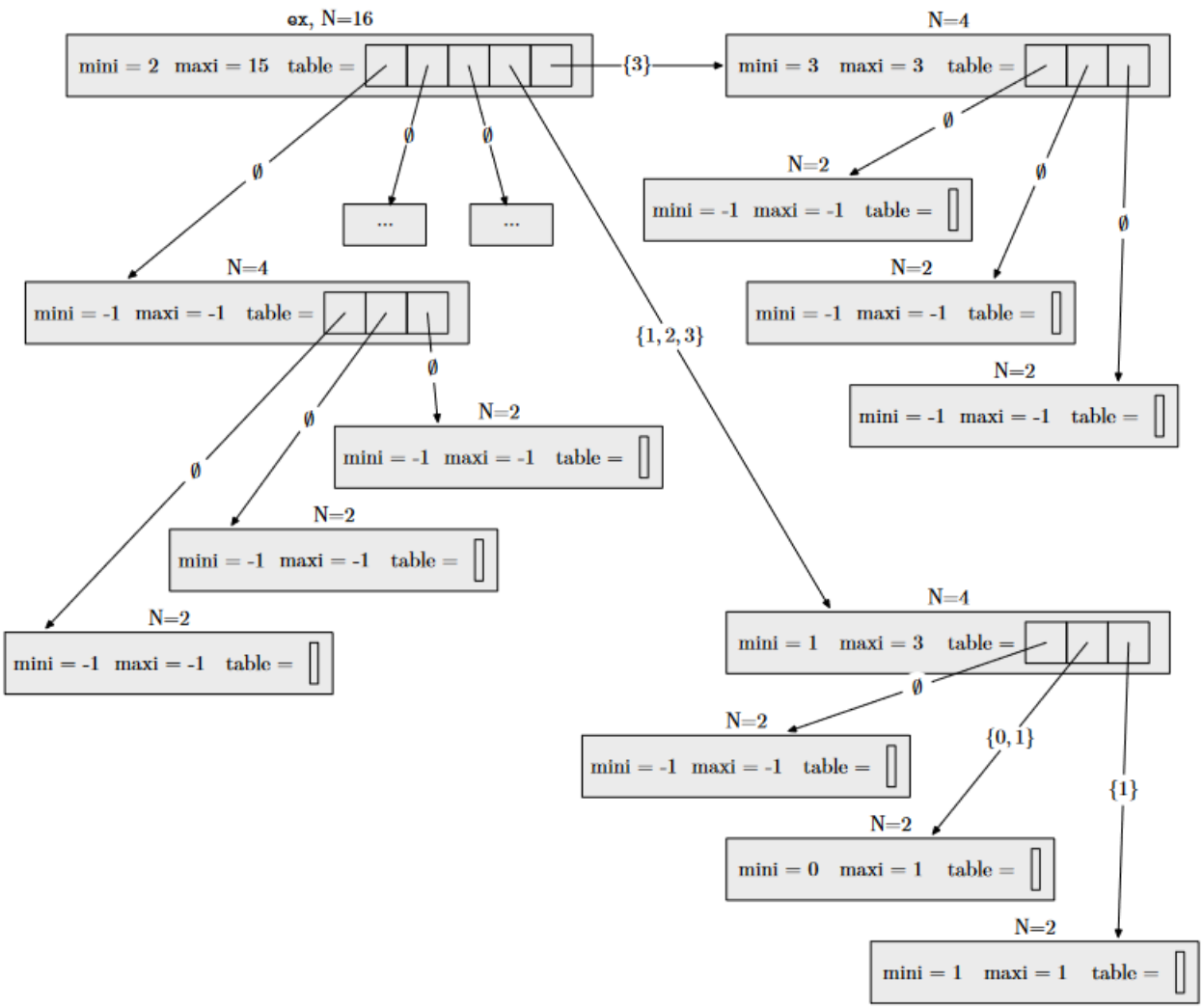

Figure 4: L'arbre *veb* ex associé à l'ensemble $\{2, 13, 14, 15\}$ avec $p = 2$

Le codage d'un ensemble $E \subset [\![0, N-1]\!]$ par un arbre *veb* dit d'ordre $N = 2^{2^p}$ suit les règles suivantes :

— Les champs `mini` et `maxi` représentent toujours la valeur minimale et maximale de E. Ils sont mis arbitrairement à -1 si l'ensemble est vide.

— Si $N = 2$, *i.e.* si l'arbre doit coder une partie de $[\![0, 1]\!]$, le champ `table` est un tableau vide (*i.e.* `[||]`), les champs `mini` et `maxi` suffisent à coder E. **On veillera à ce que les fonctions demandées traitent correctement ce cas particulier.**

— Pour $N > 2$, on note $\widehat{E} = E \setminus \{\min(E)\}$ (où $\min(E)$ désigne le minimum de E). On décompose $\widehat{E}$ en $\sqrt{N} = 2^{2^{p-1}}$ ensembles E_k définis par

$$\forall k \in [\![0, \sqrt{N}-1]\!], E_k = \left\{ x - k\sqrt{N} \;\middle|\; x \in \widehat{E} \cap [\![k\sqrt{N}, (k+1)\sqrt{N}-1 \right\}$$

Le champ `table` est alors un tableau de $\sqrt{N}+1$ arbres *veb* d'ordre $\sqrt{N}$:

- pour $k \in [\![0, \sqrt{N}-1]\!]$, l'abre *veb* stocké dans la case `table.(k)` code l'ensemble E_k ;
- l'abre *veb* stocké dans la case `table.`$(\sqrt{N})$ code l'ensemble $R = \left\{ k \in [\![0, \sqrt{N}-1]\!] \;\middle|\; E_k \neq \emptyset \right\}$

Par exemple, considérons l'arbre *veb* `ex` codant l'ensemble $E = \{2, 13, 14, 15\}$ avec $p = 2$ *i.e.* $N = 16$. On a $\widehat{E} = \{13, 14, 15\}$ puisque $\min(E) = 2$ et donc les cinq cases du tableau `ex.table` correspondent respectivement aux ensembles

$$E_0 = \emptyset \quad E_1 = \emptyset \quad E_2 = \emptyset \quad E_3 = \left\{13 - 3\sqrt{16}, 14 - 3\sqrt{16}, 15 - 3\sqrt{16}\right\} = \{1, 2, 3\} \quad R = \{3\}$$

On obtient la structure représentée en figure 4.

☐ 24 – On veut coder l'ensemble $\{0, 2, 3, 5, 7, 13, 14\}$ par un arbre *veb* d'ordre $N = 16$, noté `ex_veb`. Indiquer quel ensemble code chaque arbre *veb* stocké dans `ex_veb.table` puis préciser `ex_veb.table.(3)`.

☐ 25 – Écrire une fonction `creer_veb` de signature `int -> veb` prenant en argument un entier p et créant un arbre *veb* d'ordre $N = 2^{2^p}$ implémentant l'ensemble vide.

☐ 26 – Résoudre en fonction de q la récurrence $C(q) = C(\sqrt{q}) + \mathcal{O}(1)$ dans le cas où $q = 2^{2^s}$.

☐ 27 – Écrire une fonction `appartient_veb` de signature `veb -> int -> bool` qui teste l'appartenance d'un entier x quelconque à un ensemble E codé par un arbre *veb*. Calculer sa complexité dans le pire cas.

☐ 28 – Écrire une fonction `successeur_veb` de signature `veb -> int -> int` qui calcule le successeur d'un entier x quelconque dans E (-1 s'il n'y en a pas). Calculer sa complexité dans le pire cas.

☐ 29 – Écrire une fonction `insertion_veb` de signature `veb -> int -> unit` qui insère un entier x dans un arbre *veb*. On suppose que $x \in [\![0, N-1]\!]$. Cette fonction devra avoir une complexité en $\mathcal{O}(\log_2(\log_2(N)))$.

☐ 30 – Soit $M(N)$ la totalité de l'espace mémoire nécessaire pour stocker un ensemble par un arbre *veb* d'ordre N avec $N = 2^{2^p}$. Donner la relation de récurrence vérifiée par $M(N)$ puis déterminer l'ordre de grandeur de $M(N)$.

2 Multiplication de matrices – C (35 pts)

Étant donné deux matrices $A = (a_{i,j}) \in \mathbb{R}^{n\times k}$ et $B = (b_{i,j}) \in \mathbb{R}^{k\times m}$, le produit de ces matrices est la matrice $AB = (c_{i,j}) \in \mathbb{R}^{n\times m}$ telle que

$$c_{i,j} = \sum_{i=1}^{k} a_{i,k} b_{k,j}$$

Le but de cette section est de s'intéresser au produit de ℓ matrices $M_1 M_2 \dots M_\ell$.
Attention, la notation mathématique commence sa numérotation à 1 alors que le langage C impose que la numérotation des tableaux commence à 0.
Nous représenterons en C les matrices par des tableaux bidimensionnels de flottants :

```
typedef double ** fmatrix;
```

☐ 31 – Écrire une fonction `fmatrix new_matrix(int n1, int n2)` qui prend en entrée deux entiers n_1 et n_2 et qui renvoie une nouvelle matrice de $\mathbb{R}^{n_1\times n_2}$ ne contenant que des zéros.

☐ 32 – Écrire une fonction `void free_matrix(fmatrix m, int n1)` qui prend en entrée une matrice de $\mathbb{R}^{n_1\times n_2}$, et l'entier n_1 et qui libère la mémoire occupée par celle-ci.

☐ 33 – Écrire une fonction `fmatrix mult_matrix(fmatrix m1, fmatrix m2, int n1, int n2, int n3)` qui prend en argument deux matrices $M_1 \in \mathbb{R}^{n_1\times n_2}$ et $M_2 \in \mathbb{R}^{n_2\times n_3}$. La complexité de la fonction ne doit pas dépasser $O(n_1 n_2 n_3)$, on ne demande pas de le justifier. Dans la suite de l'énoncé, on suppose que notre multiplication de matrice possède comme complexité $n_1 n_2 n_3$.

☐ 34 – Si la matrice M_i a pour dimensions $n_i \times n_{i+1}$, quel est la dimension de la matrice $M_1 M_2 \dots M_\ell$?

☐ 35 – Étant donné trois matrices $A \in \mathbb{R}^{2\times 3}$, $B \in \mathbb{R}^{3\times 4}$, et $C \in \mathbb{R}^{4\times 2}$, on peut calculer ABC de deux manières différentes : $(AB)C$ ou $A(BC)$. Quelle est la méthode la plus efficace ?

☐ 36 – Étant donné 4 matrices A, B, C, D, combien y a-t-il de manières différentes d'obtenir le produit $ABCD$?

☐ 37 – On note C_ℓ le nombre de manières différentes de calculer le produit de ℓ matrices.
($C_0 = C_1 = C_2 = 1, C_3 = 2$)
En s'intéressant au dernier produit calculé, donner une relation de récurrence vérifiée par C_ℓ. Vérifier que $C_5 = 14$.

Passons à l'implémentation d'une solution. Soit une instance du problème que l'on souhaite résoudre : un tableau de ℓ matrices numérotés de 0 à $\ell - 1$: $M_0, \dots M_{\ell-1}$, avec M_i de dimension $n_i \times n_{i+1}$ (On a juste décalé les indices de 1 par rapport à la question 34).
On considère comme sous-problèmes la recherche des méthodes les plus efficaces pour calculer le produit des $j - i + 1$ matrices $M_i \dots M_j$ pour chaque couple (i, j).
Notons $C(i, j)$ le coût du calcul le plus efficace de $M_i \dots M_j$. Le problème initial est de calculer $C(0, \ell - 1)$.
Pour i allant de 0 à $\ell - 1$, on a $C(i, i) = 0$.

☐ 38 – Pour un k donné, quels sont les i pour lesquels $C(i, i + k)$ est défini ?

☐ 39 – Combien y a-t-il de sous-problèmes, quelle structure de données peut-on utiliser pour stocker leur résultat ?

☐ 40 – Pour i allant de 0 à $\ell - 1$, quelle est la complexité $C(i, i + 1)$?
On rapelle que pour multiplier deux matrices de dimension $n_1 \times n_2$ et $n_2 \times n_3$, la complexité est $n_1 n_2 n_3$.

☐ 41 – Donner une relation de récurrence permettant de calculer $C(i, j)$, pour $j > i + 1$.

☐ 42 – Dans quel ordre peut-on calculer les $C(i, j)$?

☐ 43 – Écrire une fonction `int **methode_optimale(fmatrix *m, int *n, int ell)` qui prend en entrée une liste de matrices de taille ℓ et une liste de dimensions de taille $\ell + 1$ et qui renvoie le tableau des $C(i, j)$.

☐ 44 – Expliquer comment retrouver l'ordre optimal des multiplications à effectuer à partir du tableau contenant les $C(i, j)$.

☐ 45 – Écrire une fonction `fmatrix mult_matrices(fmatrix *m, int *n, int ell, int **C, int i, int j)` qui prend en entrée ℓ matrices et le tableau calculé par la fonction précédente calcule le produit $M_i \dots M_j$ en utilisant la méthode optimale.

Fin de l'épreuve