MP2I – Concours blanc 2023-2024 – 4h

Pour chaque question, le langage imposé par l'énoncé doit être respecté. La rédaction doit être soignée.

Les fonctions doivent être documentées et commentées à bon escient. Le code doit être indenté et d'un seul tenant (pas de flèche pour dire qu'un bout de code se situe ailleurs par exemple). Le code illisible ne sera pas corrigé.

Sauf précision contraire, les réponses doivent être justifiées.

Tout document interdit. Calculatrice interdite. Téléphone éteint et rangé dans le sac. Pas de montre connectée.

Ce qui est illisible ou trop sale ne sera pas corrigé.
Soulignez ou encadrez vos résultats.

Toute fonction demandée dans une question peut être utilisée dans une question ultérieure, même si elle n'a pas été implémentée. Vous pouvez à tout moment choisir d'introduire des fonctions intermédiaires non explicitement demandées, mais elles doivent être précisément documentées. **Une fonction non demandée et non documentée ne sera pas lue.**
Sauf mention explicite contraire, toute réponse doit être justifiée.
Sauf mention explicite contraire, les fonctions demandées ne doivent pas modifier les informations fournies par leurs arguments.

Cet énoncé est composé de trois parties entièrement indépendantes.

PREMIÈRE PARTIE : LE PROBLÈME DU SAC-À-DOS

Ce problème est à traiter entièrement en C.

On considère le problème suivant : Madame X doit partir en voyage ; elle emporte un sac-à-dos qui peut supporter au maximum un poids Q, poids qui est un entier strictement positif. Par ailleurs, elle dispose de n objets numérotés par $0, 1, \ldots, n-1$. Pour tout i appartenant à $[\![0, n-1]\!]$, l'objet numéro i possède deux caractéristiques : son utilité notée u_i et son poids noté p_i, qui sont des entiers strictement positifs. Madame X désire emporter certains de ces objets dans son sac-à-dos ; le poids total des objets emportés ne doit pas dépasser Q ; elle cherche à maximiser la somme des utilités des objets qu'elle emporte parmi les contenus possibles ne dépassant pas au total le poids Q.

Les données seront représentées en C par les types suivants :

```
typedef int sac;

struct affaires {
  int n;
  int *u, *p; //tableaux de longueur n
};

typedef struct affaires affaires;
```

Notation. Si x est un nombre réel, $\lfloor x \rfloor$ désigne la partie entière par défaut de x (c'est-à-dire le plus grand entier inférieur ou égal à x) tandis que $\lceil x \rceil$ désigne sa partie entière par excès (c'est-à-dire le plus petit entier supérieur ou égal à x).

1 Sac-à-dos fractionnaire

Dans cette partie, les objets sont fractionnables. On note x_i la quantité de l'objet numéro i emportée par Madame X. Le problème s'écrit :

$$\text{Maximiser } z = \sum_{i=0}^{n-1} u_i x_i, \quad \text{avec les contraintes} \quad \begin{cases} \displaystyle\sum_{i=0}^{n-1} p_i x_i \leq Q \quad \text{et} \\ \forall i \in [\![0, n-1]\!],\ x_i \in [0,1]. \end{cases}$$

1.1 Avec une hypothèse sur l'ordre des données

Dans cette section 1.1, on fait l'hypothèse suivante :

$$\forall i \in [\![1, n-1]\!], \quad \frac{u_{i-1}}{p_{i-1}} \geq \frac{u_i}{p_i} \tag{1}$$

On définit un entier i^* compris entre 0 et n par :

- si $p_0 < Q$ et $\sum_{i=0}^{n-1} p_i \geq Q$, alors on définit i^* comme étant l'unique entier tel que

$$\sum_{i=0}^{i^*-1} p_i < Q \leq \sum_{i=0}^{i^*} p_i ,$$

- sinon :

$$i^* = \begin{cases} 0 & \text{si } p_0 \geq Q , \\ n & \text{si } \sum_{i=0}^{n-1} p_i < Q . \end{cases}$$

Question 1 : Montrer qu'une solution maximale pour le problème est donnée par :

$$x_i = \begin{cases} 1 & \text{si } 0 \leq i \leq i^*-1 , \\ \dfrac{Q - \sum_{k=0}^{i^*-1} p_k}{p_{i^*}} & \text{si } i = i^* \text{ et } i^* \neq n , \\ 0 & \text{si } i^*+1 \leq i \leq n-1 . \end{cases}$$

Question 2 : Résoudre le problème du sac-à-dos fractionnaire suivant :

$$\text{Maximiser } z = 16x_0 + 21x_1 + 19x_2 + 15x_3 + 13x_4 + 7x_5, \quad \text{avec les contraintes}$$

$$\begin{cases} 15x_0 + 22x_1 + 20x_2 + 17x_3 + 15x_4 + 9x_5 \leq 51 & \text{et} \\ \forall i \in [\![0,5]\!], \ x_i \in [0,1] . \end{cases}$$

On donnera la valeur du maximum de la fonction z et les valeurs x_0, x_1, x_2, x_3, x_4 et x_5 permettant d'atteindre ce maximum.

Question 3 : On revient au problème général du sac-à-dos fractionnaire. Écrire une fonction

```
bool ordre_respecte(affaires a);
```

qui teste si la condition (1) est bien vérifiée.

Question 4 : En supposant, sans avoir à le vérifier, que la condition (1) est vérifiée, écrire une fonction

```
int i_etoile(sac Q, affaires a);
```

qui calcule i^*.

1.2 Sans hypothèse sur l'ordre des données

On cherche maintenant à déterminer la solution du problème du sac-à-dos fractionnaire **sans faire l'hypothèse** (1). En revanche, on suppose que l'on a

$$\sum_{i=0}^{n-1} p_i \geq Q .$$

On cherche à déterminer i^* appartenant à $[\![0, n-1]\!]$ et une partition de l'ensemble $[\![0, n-1]\!] \setminus \{i^*\}$ en deux sous-ensembles I' et I'' de façon à avoir simultanément :

$$\begin{cases} \forall i \in I', \quad \dfrac{u_i}{p_i} \geq \dfrac{u_{i^*}}{p_{i^*}}, \\ \forall j \in I'', \quad \dfrac{u_{i^*}}{p_{i^*}} \geq \dfrac{u_j}{p_j}, \\ \displaystyle\sum_{i \in I'} p_i < Q \quad \text{et} \quad \left(\sum_{i \in I'} p_i\right) + p_{i^*} \geq Q. \end{cases}$$

On admet qu'on dispose d'un algorithme $\mathscr{A}$ qui, étant donné un ensemble E de m éléments ayant chacun une valeur, partitionne cet ensemble en deux sous-ensembles E' et E'' de cardinaux respectivement $\lfloor \frac{m}{2} \rfloor$ et $\lceil \frac{m}{2} \rceil$ de sorte que toutes les valeurs des éléments de E' soient supérieures ou égales à toutes les valeurs des éléments de E'' ; on suppose de plus que $\mathscr{A}$ est linéaire en m.

Question 5 : Expliciter, sans utiliser de langage de programmation, un algorithme récursif (qu'il ne sera pas la peine de prouver) utilisant l'algorithme $\mathscr{A}$ qui détermine i^*, I' et I'' ; cet algorithme devra être linéaire en n.

Question 6 : Prouver la linéarité en n de l'algorithme proposé ci-dessus. Pour simplifier cette preuve, on pourra se restreindre aux valeurs de n qui sont des puissances de 2 ; on admettra que l'algorithme est aussi linéaire lorsque n est quelconque.

Question 7 : Déduire des questions 1, 5 et 6 qu'on peut résoudre le problème du sac-à-dos fractionnaire portant sur n objets en un temps au plus linéaire en le nombre d'objets, même si la condition (1) n'est pas respectée.

2 Sac-à-dos en 0-1

Dans cette partie, les objets ne sont plus fractionnables : chaque objet est pris ou laissé. Le problème s'écrit :

$$\text{Maximiser } z = \sum_{i=0}^{n-1} u_i x_i, \quad \text{avec les contraintes} \quad \begin{cases} \displaystyle\sum_{i=0}^{n-1} p_i x_i \leq Q \quad \text{et} \\ \forall i \in [\![0, n-1]\!], \ x_i \in \{0,1\}. \end{cases}$$

On dit que $\overline{x} = (\overline{x_0}, \overline{x_1}, \ldots, \overline{x_{n-1}}) \in \{0,1\}^n$ est *réalisable* si on a $\displaystyle\sum_{i=0}^{n-1} p_i \overline{x_i} \leq Q$. On appelle *valeur* d'une telle solution la valeur correspondante de la fonction z ou, autrement dit, la quantité

$$\sum_{i=0}^{n-1} u_i \overline{x_i}.$$

Le problème consiste donc à déterminer la meilleure solution réalisable, c'est-à-dire celle qui a la plus grande valeur.

Question 8 : Montrer que la meilleure solution au problème du sac-à-dos en 0-1 est inférieure ou égale à la meilleure solution au problème du sac-à-dos fractionnaire ayant les mêmes données numériques.

On peut utiliser la méthode de type *diviser-pour-régner* suivante pour résoudre le problème du sac-à-dos en 0-1 : la meilleure solution réalisable peut être cherchée successivement parmi les solutions réalisables avec $x_0 = 0$, puis parmi celles avec $x_0 = 1$ (s'il en existe).

Question 9 : Écrire une fonction

```
int utilite_maximale(sac Q, affaires a);
```

qui renvoie l'utilité maximale qu'on peut obtenir avec les données fournies en argument pour le problème du sac-à-dos en 0-1 (n'hésitez pas à utiliser une ou plusieurs fonctions auxiliaires que vous spécifierez soigneusement).

Question 10 : Donner en la justifiant la complexité dans le pire des cas de la méthode par séparation utilisée dans la question précédente.

DEUXIÈME PARTIE : CODE BINAIRE DE GRAY

Dans toute cette partie, n désigne un entier naturel non nul.
On appelle *n-entier* un entier représentable en base 2 sur n chiffres, quitte à ajouter des 0 avant le chiffre de poids fort. Par exemple 2 est un 2-entier car on peut l'écrire sur 2 chiffres en base 2 ($\overline{10}^2$), mais c'est aussi un 3-entier : on peut l'écrire $\overline{010}^2$ en base 2.
On cherche à énumérer tous les n-entiers de différentes manières.

3 Ordre lexicographique

Toute cette section est à faire en C.
On s'intéresse tout d'abord à l'énumération des n-entiers dans l'ordre lexicographique.
Les n-entiers seront implémentés en C à l'aide du type

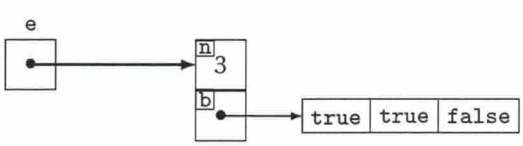
```
struct entier {
  int n; // longueur de la représentation
  bool *b; // écriture binaire : n cases (false <-> 0 / true <-> 1)
           // bit de poids faible dans la dernière case
};
```

où le booléen `false` représente le chiffre 0 et le booléen `true` représente le chiffre 1. Le chiffre de poids faible se trouve dans la dernière case du tableau (les premières cases pouvant contenir des `false`).

Question 11 : Écrire une fonction

```
void print(struct entier *e);
```

qui affiche l'écriture binaire associée à `e`.
Par exemple l'affichage obtenu pour



est 110.

Question 12 : Écrire une fonction

```
int valeur(struct entier *e)
```

qui renvoie l'entier représenté par `e`. La complexité de cette fonction doit être linéaire en la longueur de la représentation, sans avoir à la justifier.

Question 13 : Écrire une fonction

```
struct entier *suivant(struct entier e);
```

qui prend en argument une représentation d'un n-entier, et renvoie la liste représentant le n-entier suivant (pour l'ordre naturel sur $\mathbb{N}$). Si l'entier fourni en argument est le plus grand n-entier (que des 1 dans la représentation binaire), alors la fonction renvoie `NULL`.
La complexité temporelle dans le pire des cas de votre fonction doit être en $\Theta(\ell)$ où ℓ est la longueur de la représentation. Vous devez justifier brièvement cette complexité.

Question 14 : Écrire une fonction

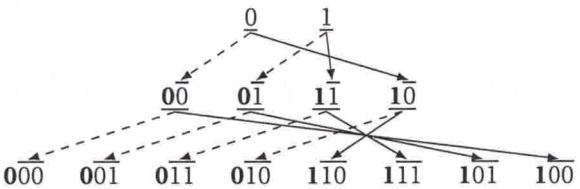
```
void affiche(int m);
```

telle que l'appel `affiche(m)` affiche les représentations binaires des entiers naturels encodables sur `m` bits, dans l'ordre lexicographique, un entier par ligne.

Question 15 : Donner en la justifiant la complexité temporelle dans le pire des cas de la fonction `affiche`.

4 Ordre de Gray

Toute cette section est à faire en `OCaml`.
Pour certaines applications, par exemple pour éviter des états de transitions intermédiaires dans les circuits logiques ou pour faciliter la correction d'erreur dans les transmissions numériques, on souhaite que le passage d'un entier au suivant ne modifie qu'un seul bit. Dans un brevet de 1953, Franck Gray définit un ordre possédant cette propriété.
L'algorithme proposé par Franck Gray est récursif : la liste des 1-entiers est (0, 1). Pour produire la liste des $(n+1)$-entiers dans l'ordre de Gray à partie de le liste des n-entiers, on les énumère dans l'ordre en ajoutant un 0 à gauche, puis dans l'ordre inverse en ajoutant un 1 à gauche :



4.1 Énumération

Dans cette partie, un n-entier est représenté par le type

```
type entier = bool list
```

où le booléen `false` représente le chiffre 0 et le booléen `true` le chiffre 1.

Question 16 : Écrire une fonction `rev : 'a list -> 'a list` qui renvoie une liste composée des mêmes éléments que la liste transmise en argument, mais dans le sens inverse. Cette fonction doit avoir une complexité linéaire en la taille de la liste passée en paramètre, à justifier brièvement.

On considère la fonction

```
1 let rec gray n =
2   if n <= 0 then [[]]
3   else let precedent = gray (n - 1) in
4        (ajout false precedent) @ (ajout true (rev precedent))
```

Question 17 : Donner le type de la fonction `gray`.

Question 18 : Prouver la terminaison de la fonction `gray`.

Question 19 : Prouver que si `n` est un entier positif, alors `gray n` est la liste des `n`-entiers dans l'ordre de Gray.

Question 20 : Prouver que la complexité de la fonction `gray` est en $\mathcal{O}(n^2)$ où n est la valeur du paramètre.

4.2 Récupération par indice

On va maintenant essayer d'obtenir le k-ème n-entier pour l'ordre de Gray à partir de k, sans fabriquer toute la suite de n-entiers.
Pour $n \in \mathbb{N}^*$ fixé, on définit $g_n : [\![0, 2^n-1]\!] \to \mathbb{N}$ telle que $g_n(k)$ est l'entier dont une représentation en base 2 est donnée par le k-ème n-entier dans l'ordre de Gray.
Ainsi, $g_3(7) = 4$ car la liste des 3-entiers dans l'ordre de Gray est $(\overline{000}^2, \overline{001}^2, \overline{011}^2, \overline{010}^2, \overline{110}^2, \overline{111}^2, \overline{101}^2, \overline{100}^2)$: l'élément d'indice 7 dans cette liste est $\overline{100}^2$ qui est une représentation en base 2 de l'entier 4.

Question 21 : Montrer que pour $n \in \mathbb{N}^*$ et $k \in [\![0, 2^n-1]\!]$, la valeur de $g_n(k)$ ne dépend pas de n.

Grâce au résultat de la question précédente, on se permet maintenant d'écrire $g(k)$ à la place de $g_n(k)$.

Question 22 : Soit $n \in \mathbb{N}^*$ et un entier $k \in [\![2^n, 2^{n+1}-1]\!]$.

1. Monter qu'il existe un entier $r \in [\![0, 2^n-1]\!]$ tel que $k = 2^n + r$.
2. Montrer que $g(k) = 2^n + g(2^n - 1 - r)$, où r est l'entier défini ci-dessus.

On note $\oplus$ l'opération de ou exclusif dont voici la table :

$\oplus$	0	1
0	0	1
1	1	0

On étend cette opération aux n-entiers en l'appliquant bit à bit, ainsi, si $x = 5 = \overline{101}^2$ et $y = 3 = \overline{011}^2$, on a

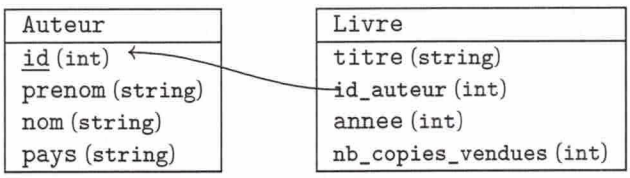
$$x \oplus y = \overline{101}^2 \oplus \overline{011}^2 = \overline{110}^2 = 6.$$

Question 23 : Déduire de la question précédente que si la représentation binaire de k est $b_{n-1}\dots b_0$ (avec b_{n-1} le chiffre de poids fort), et si on pose $b_n = 0$, alors la représentation binaire de $g(k)$ est $a_{n-1}\dots a_0$ où, pour tout $j \in [\![0, n-1]\!]$, on a : $a_j = b_j \oplus b_{j+1}$.

Question 24 : Exprimer à l'aide de $\oplus$ la valeur de $g(k)$ en fonction de k.

TROISIÈME PARTIE : BASE DE DONNÉES DE LIVRES

Toute cette partie est à faire en SQL.
Dans cette partie, on considère une base de données composée des deux tables suivantes :



- la table `Auteur` contient, pour chaque auteur, son prénom, son nom, son pays d'origine, ainsi qu'un entier `id` unique pour chaque auteur, servant de clé primaire pour cette table ;
- la table `Livre` contient, pour chaque livre, son titre, l'`id` de son auteur (clé étrangère), son année de sortie, ainsi que le nombre de copies du livre vendues.

Question 25 : Écrire une requête SQL renvoyant les couples (`prenom`, `nom`) des auteurs nés en France.

Question 26 : Écrire une requête SQL renvoyant les triplets (`prenom_auteur`, `nom_auteur`, `titre`) des livres sortis en 2023.

Question 27 : Écrire une requête SQL renvoyant le titre du livre ayant fait le plus de ventes (s'il y en a plusieurs ex-aequo, on veut tous les renvoyer).

Question 28 : Écrire une requête SQL renvoyant les triplets (`prenom`, `nom`, `nb_livres`), où `nb_livres` est le nombre de livres écrits par l'auteur en question.

Question 29 : Écrire une requête SQL renvoyant les pays dont les livres sont en moyennes vendus plus de 1000 copies.

Question 30 : Écrire une requête SQL renvoyant les couples (`prenom`, `nom`) des auteurs français qui ont vendus au total plus de 100 000 livres (au cumulé sur tous leurs livres).

Question 31 : Écrire une requête SQL renvoyant les couples (`titre_1`, `titre_2`) des livres ayant le même auteur.