Devoir en temps limité du Vendredi 28 Mars

Dans l'ensemble du devoir, toutes les fonctions seront :

- *écrites en langage OCaml,*
- *précédées d'une explication des variables utilisées,*
- *précédées d'une explication de l'algorithme.*

Exercice 1 (La suite de Fibonacci)

Considérons la suite de Fibonacci (F_n) définie par :

$$\begin{cases} F_0 = 0\,,\ F_1 = 1\,, \\ \forall n \in \mathbb{N}\,,\ F_{n+2} = F_{n+1} + F_n \end{cases}.$$

1. (a) Écrire en OCaml une fonction itérative `fib1 : int -> int` qui renvoie, pour tout entier naturel n, la valeur de F_n.
   (b) Écrire en OCaml une fonction récursive "naïve" `fib2 : int -> int` qui renvoie, pour tout entier naturel n, la valeur de F_n.
   (c) Écrire en OCaml une fonction récursive terminale `fib3 : int -> int` qui renvoie, pour tout entier naturel n, la valeur de F_n.
   (d) Écrire en OCaml une fonction récursive `fib4 : int -> int` utilisant le principe de mémoïsation qui renvoie, pour tout entier naturel n, la valeur de F_n.
   (e) Donner, sans justification, les complexités temporelles des fonctions `fib1`, `fib2`, `fib3` et `fib4`.
2. (a) Montrer par récurrence sur q que :
   $$\forall (p,q) \in \mathbb{N} \times \mathbb{N}^*, \quad F_{p+q} = F_{p+1}F_q + F_pF_{q-1}.$$
   (b) Exprimer F_{2n}, F_{2n+1} et F_{2n+2} en fonction de F_n et F_{n+1}.
   (c) A l'aide de la question précédente, écrire en OCaml une fonction récursive `fib5 : int -> int * int` qui renvoie, pour tout entier naturel n, la valeur du couple (F_n, F_{n+1}).
   On pourra distinguer deux cas suivant la parité de n.
   (d) Énoncer le théorème de complexité des algorithmes "diviser pour régner" puis en déduire la complexité temporelle de la fonction `fib5`.

Exercice 2 (Programmation récursive sur les listes)

Dans cet exercice, les fonctions seront récursives et manipuleront des listes en limitant les opérateurs sur les listes à l'ajout-en-tête `::`, la tête d'une liste `List.hd`, la queue d'une liste `List.tl` et la liste vide `[]`.
On rappelle que, par définition, un *ensemble* est un nombre fini d'éléments sans ordre et sans répétition. En OCaml, un ensemble sera représenté par une liste d'éléments sans ordre ni répétition. Par exemple, `[1;2;5;4;3]` est un ensemble, `[1;2;5;1;4;3]` n'est pas un ensemble.

1. Écrire une fonction `appartenance` qui permet de savoir si un élément appartient à un ensemble.
   Cette fonction pourra être utilisée dans la suite de cet exercice.

2. Écrire une fonction `test_ensemble` qui permet de savoir si une liste représente ou non un ensemble.
3. Écrire une fonction `union` qui effectue la réunion de deux ensembles. Par exemple :
```
union [1;2;5;4;3] [6;4;7;5] ;;
- : int list  = [1;2;5;4;3;6;7]  (* à l'ordre près *)
```
4. Écrire une fonction `intersection` qui effectue l'intersection de deux ensembles :
```
intersection [1;2;5;4;3] [6;4;7;5] ;;
- : int list  = [5;4]  (* à l'ordre près *)
```
5. Écrire une fonction `difference` qui effectue la différence de deux ensembles :
```
difference [1;2;5;4;3] [6;4;7;5] ;;
- : int list  = [1;2;3]  (* à l'ordre près *)
```

Exercice 3 (Système monétaire)

Nous disposons dans notre système monétaire, de billets et pièces de différentes valeurs, comme par exemple ... , 10€00 , 5€00 , 2€00 , 1€00 , 50cts, ... Pour la suite de l'exercice, nous nous limiterons à des billets d'une valeur inférieure ou égale à 50 €00.

Une somme d'un montant de $s = \dfrac{n}{100}$ €, $n \in \mathbb{N}$, nous est demandée.

La méthode la plus utilisée pour constituer cette somme à l'aide de nos billets et pièces est :

— de commencer par s'approcher au plus près de cette somme par valeur inférieure à l'aide de billets de 50€00 et notons a le nombre de billets ainsi déterminé (éventuellement $a = 0$ dans le cas d'une somme inférieure ou égale à 49€99 ...) ,

— puis de répéter cette opération sur la somme restante, c'est-à-dire $s - a \times 50$ € avec les billets de 20€00 et notons b le nombre de billets ainsi déterminé,

— puis de répéter cette opération sur la somme restante, c'est-à-dire $s - a \times 50 - b \times 20$ € avec les billets de 10€00,

— ..... jusqu'à obtenir la somme désirée.

Cette méthode (ou algorithme) qui suit le principe de faire, étape par étape, un choix optimum local, est connue sous le nom *d'algorithme glouton*.

Afin de rester dans le domaine entier, nous travaillerons avec comme unité le centime d'euro noté ct ou cts. Aussi, les variables manipulées seront toutes de type `int`, seules les opérations d'addition, de soustraction et de produit sont permises.

1. Soit $n \in \mathbb{N}$, $x \in \{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000\}$. Nous disposons d'un nombre illimité de billets ou pièces d'une valeur de x cts et une somme d'un montant de n cts nous est demandée.

   Écrire une fonction itérative ou récursive `compteur1 n x` qui détermine le nombre `a` de billets ou pièces d'une valeur de `x` cts tel que :

$$a \times x \leqslant n < (a+1) \times x\,.$$

   Reconnaissez-vous un schéma classique ?

2. Soit $n \in \mathbb{N}$, $x \in \{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000\}$. Nous disposons d'un nombre limité p de billets ou pièces d'une valeur de x cts et une somme d'un montant de n cts nous est demandée.

Écrire une fonction itérative ou récursive `compteur2 n x p` qui détermine le nombre maximal `a` de billets ou pièces d'une valeur de `x` cts tel que :

$$a \times x \leqslant n\,.$$

3. Soit $n \in \mathbb{N}$, $x \in \{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000\}$. Nous disposons d'un nombre limité p de billets ou pièces d'une valeur de x cts et une somme d'un montant de n cts nous est demandée.
   Écrire une fonction itérative ou récursive `compteur3 n x p` qui détermine le nombre `a` de billets ou pièces d'une valeur de `x` cts tel que :

$$a \times x \leqslant n < (a+1) \times x\,.$$

   dans le cas où le nombre de pièces ou billets n'est pas suffisant, la valeur finale de a sera -1.

4. x est un tableau indexé de 0 à 11 tel que :
   $x.(0) = 5000$, $x.(1) = 2000$, $x.(2) = 1000$, $x.(3) = 500$, $x.(4) = 200$,
   $x.(5) = 100$, $x.(6) = 50$, $x.(7) = 20$, $x.(8) = 10$, $x.(9) = 5$, $x.(10) = 2$, $x.(11) = 1$.
   Nous disposons d'un nombre illimité de billets ou pièces de chaque valeur $x.(i)$ cts, pour $i \in [\![0, 11]\!]$.
   n appartenant à $\mathbb{N}$, une somme de n cts nous est demandée.
   Nous voulons construire un tableau c indexé de 0 à 11 tel que pour tout $i \in [\![0, 11]\!]$, $c.(i)$ est le nombre de billets ou pièces de valeur $x.(i)$ cts utilisés afin de constituer la somme demandée.
   (a) Le problème a-t-il toujours une solution ? toujours une solution en suivant l'algorithme glouton ? Les réponses seront justifiées.
   (b) Écrire une fonction qui constitue la somme demandée en suivant l'algorithme glouton, autrement dit qui détermine le nombre de billets et pièces utilisés afin de constituer la somme demandée.

5. x est un tableau indexé de 0 à 11 tel que :
   $x.(0) = 5000$, $x.(1) = 2000$, $x.(2) = 1000$, $x.(3) = 500$, $x.(4) = 200$,
   $x.(5) = 100$, $x.(6) = 50$, $x.(7) = 20$, $x.(8) = 10$, $x.(9) = 5$, $x.(10) = 2$, $x.(11) = 1$.
   Nous disposons d'un nombre limité de billets ou pièces de chaque valeur $x.(i)$ cts, pour $i \in [\![0, 11]\!]$.
   p est un tableau indexé de 0 à 11 tel que pour tout $i \in [\![0, 11]\!]$, $p.(i)$ est le nombre de billets ou pièces de chaque valeur $x.(i)$ cts disponibles.
   n appartenant à $\mathbb{N}$, une somme de n cts nous est demandée.
   Nous allons, si possible, construire un tableau c indexé de 0 à 11 tel que pour tout $i \in [\![0, 11]\!]$, $c.(i)$ est le nombre de billets ou pièces de valeur $x.(i)$ cts utilisés afin de constituer la somme demandée.
   (a) Écrire une fonction qui calcule la somme maximale que nous pouvons constituer avec les pièces et billets disponibles.
   Toute somme inférieure à cette somme maximale peut-elle être constituée avec nos pièces et billets disponibles ? La réponse sera justifiée.

(b) Nous reprenons le schéma de l'algorithme glouton pour constituer la somme demandée et en cas de manque d'une catégorie de billets ou pièces d'une valeur $x.(i)$ cts, $i \in [\![0, 11]\!]$, nous décidons que nous sommes en situation d'échec.

Écrire une fonction qui détermine si nous pouvons constituer ainsi la somme demandée et dans l'affirmative détermine le nombre de billets et de pièces de chaque catégorie utilisée.

(c) Nous reprenons le schéma de l'algorithme glouton pour constituer la somme demandée et en cas de manque d'une catégorie de billets ou pièces d'une valeur $x.(i)$ cts, $i \in [\![0, 11]\!]$, nous décidons d'utiliser tous les billets de cette catégorie et de poursuivre en passant à la catégorie de billets ou pièces immédiatement inférieure (si elle existe ...).

Écrire une fonction qui détermine si nous pouvons constituer ainsi la somme demandée et dans l'affirmative détermine le nombre de billets et de pièces de chaque catégorie utilisée.

6. La confiance n'exclut pas le contrôle !

x est un tableau indexé de 0 à 11 tel que :

$x.(0) = 5000$, $x.(1) = 2000$, $x.(2) = 1000$, $x.(3) = 500$, $x.(4) = 200$,

$x.(5) = 100$, $x.(6) = 50$, $x.(7) = 20$, $x.(8) = 10$, $x.(9) = 5$, $x.(10) = 2$, $x.(11) = 1$.

Nous disposons d'un nombre limité de billets ou pièces de chaque valeur $x.(i)$ cts, $i \in [\![0, 11]\!]$.

p est un tableau indexé de 0 à 11 tel que pour tout $i \in [\![0, 11]\!]$, $p.(i)$ est le nombre de billets ou pièces de chaque valeur $x.(i)$ cts disponibles.

n appartenant à $\mathbb{N}$, une somme de n cts nous est demandée.

Nous supposons avoir construit un tableau c indexé de 0 à 11 tel que pour tout $i \in [\![0, 11]\!]$, $c.(i)$ est le nombre de billets ou pièces de valeur $x.(i)$ cts utilisés afin de constituer la somme demandée.

Écrire une fonction qui vérifie les résultats.

7. Dans notre système Euro, l'algorithme glouton est le plus efficace en termes de nombre de billets et pièces manipulés pour constituer une somme donnée.

Que pensez-vous de cet algorithme dans l'ancien système monétaire du Royaume-Uni qui utilisait des pièces de valeurs 30 pences (la demi-couronne), 24 pences (le florin), 12 pences (le shilling), 6 pences (le sixpence), 3 pences (le threepence) ou 1 penny ?