Devoir surveillé 2

3 Problèmes

Option informatique MPSI1 & MPSI2

Le devoir est composé de 3 problèmes qu'il n'est pas nécessaire de finir pour obtenir la note maximale.

I Problème 1 : sous ensembles, adapté de E3A 2012 ex. 3

Le but de l'exercice est de construire des fonctions de calculs sur des ensembles finis.
On considérera des sous-ensembles finis d'entiers.
Un ensemble A sera représenté par une liste **strictement croissante** donc sans doublon.
Par exemple $A = \{4, 5, 3, -1, 3, 7\}$ est représenté par `[-1; 3; 4; 5; 7]`.

Les fonctions seront récursives, toute fonction qui emploie des tableaux sera refusée.

Question 1 - Appartenance
Écrire une fonction `appartient` prenant en arguments un entier k et une liste représentant un ensemble A et qui renvoie `true` ou `false` selon que x appartient ou non à A.

```
appartient : int -> int list -> bool
appartient 7 [1; 4; 9] -> false
appartient 3 [2; 3; 11] -> true
```

Question 2 - Adjonction
Écrire une fonction `add` prenant en arguments un entier k et une liste représentant un ensemble A et qui renvoie la liste représentant $A \cup \{x\}$.

```
add : int -> int list -> int list
add 7 [1; 4; 9] -> [1; 4; 7; 9]
add 3 [2; 3; 11] -> [2; 3; 11]
```

Indication : on pourra s'inspirer de la fonction `inserer` du tri par insertion.

Pour calculer l'intersection de deux ensembles on propose la fonction suivante.

```
let rec union1 l1 l2 =
  match l1 with
  |[] -> l2
  |t1::q1 -> union1 q1 (add t1 l2);;
```

Question 3 - Beaucoup de comparaisons
Montrer que le nombre de comparaisons faites lors du calcul de l'union de deux ensembles A_1 et A_2 est majoré par $n_1.n_2$ où n_i est le cardinal de A_i.
Donner deux ensembles A_1 et A_2 de cardinal 3 pour lesquels le nombre de comparaisons est 9.

Cette fonction n'est pas satisfaisante car elle effectue beaucoup de comparaisons.

Question 4 - Une meilleure foncction
Écrire une fonction `union2` prenant en arguments deux listes représentant deux ensembles A et B et qui renvoie la liste représentant $A \cup B$. Cette fonction devra effectuer au plus $n_1 + n_2$ comparaisons où n_1 et n_2 sont les cardinaux des ensembles représentés par les listes.

```
union : int list -> int list -> int list
union [1; 4; 7; 9] [1; 7; 8] -> [1; 4; 7; 8; 9]
```

Indication : on pourra s'inspirer de la fonction `fusion` du tri fusion.

Question 5 - Intersection
Écrire une fonction `intersection` prenant en arguments deux listes représentant deux ensembles A et B et qui renvoie la liste représentant $A \cap B$.

```
intersection : int list -> int list -> int list
intersection [1; 4; 7; 9] [1; 7; 8] -> [1; 7]
```

Pour tout $n \in \mathbb{N}^*$ on définit $E_n = \{0, 1, 2, \ldots, n-1\}$ et on pose $E_0 = \emptyset$.

Question 6 - Parties
Écrire une fonction `parties` qui reçoit 2 entiers positifs n et p comme paramètres et qui renvoie la liste des parties de E_n à p éléments. Pour $n < p$ la fonction doit renvoyer une liste vide.

```
parties : int -> int -> int list list
parties 3 0 -> [[]]
parties 4 3 -> [[0; 1; 2]; [0; 1; 3]; [0; 2; 3]; [1; 2; 3]]
```

Indication : on pourra remarquer que les parties à p éléments de E_n sont
soit des parties à p éléments de E_{n-1},
soit des parties à $p-1$ éléments de E_{n-1} auxquelles on ajoute $n-1$. On pourra écrire une fonction qui ajoute un élément à toutes les listes d'une liste de listes ou utiliser `List.map`.

II Problème 2 : minimum local

Dans tout le problème on considère des tableaux de la forme $t = [|t_0; t_1; t_2; \ldots; t_{n-2}; t_{n-1}|]$ de longueur $n \geqslant 1$. On n'aura pas besoin de vérifier que les tableaux sont non vides.

On cherche un minimum local de ce tableau, c'est-à-dire la valeur t_i avec i compris entre 0 et $n-1$ telle que t_i est inférieur ou égal à ses voisins ; on remarquera que t_i peut avoir 2 voisins mais n'en a qu'un seul si $i = 0$ ou $i = n-1$ ou même aucun si $n = 1$.

- t_1 et t_5 sont les minimums locaux de de $[14; 5; 8; 7; 4; 2]$,
- t_1, t_2 et t_3 sont les minimums locaux de de $[14; 5; 5; 5; 8]$,

On peut remarquer que le minimum d'un tableau est un minimum local.

Question 7 - Minimum global
Écrire une fonction `minimum` qui reçoit un tableau pour paramètre et qui renvoie l'indice d'un minimum local en calculant l'indice d'un minimum.

```
minimum : 'a array -> 'a
```

Question 8 - Une propriété
Prouver que le premier minimum local d'un tableau t de longueur n est

- soit t_{n-1}
- soit t_i où i est le premier indice $i < n-1$ tel que $t_i \leqslant t_{i+1}$.

Dans quel cas le premier minimum local vaut-il t_{n-1} ?

Question 9 - Premier minimum local
En utilisant la question précédente écrire une fonction `premier_ml` qui renvoie le premier minimum local d'un tableau ; on devra utiliser une boucle **`while`**.

```
premier_ml : 'a array -> 'a
```

Question 10 - Cas des listes
Adapter l'algorithme précédent pour calculer un minimum local dans le cas d'une liste.

```
premier_ml_rec : 'a list -> 'a
```

Question 11 - Séparation
k est un indice tel que $0 < k < n-1$ et t_k n'est pas un minimum local. Prouver que
si $t_k > t_{k-1}$ alors un minimum local de $[|t_0; \ldots; t_{k-1}|]$ est un minimum local du tableau,
si $t_k > t_{k+1}$ alors un minimum local de $[|t_{k+1}; \ldots; t_{n-1}|]$ est un minimum local du tableau.

Question 12 - Algorithme rapide
En déduire une fonction `minimum_local` utilisant la méthode diviser-pour-régner qui calcule un minimum local d'un tableau avec une complexité logarithmique (on ne demande pas de justifier la complexité).

III Problème 3 : produit de matrices

Rappels OCaml

- On rappelle les opérations de base sur les nombres en virgule flottante (ou flottants), que l'on utilisera pour représenter des nombres réels dans OCaml :
  - `1.0` et `0.0` (ou `1.` et `0.`) représentent les nombres 1 et 0 respectivement.
  - L'addition entre deux flottants `a` et `b` s'écrit `a +. b`.
  - La multiplication entre deux flottants `a` et `b` s'écrit `a *. b`.
- On rappelle quelques opérations de base sur les tableaux :
  - `Array.length : 'a array -> int` donne la longueur d'un tableau.
  - `Array.make : int -> 'a -> 'a array` permet de créer un tableau : `Array.make m r` crée un tableau de taille m dont toutes les cases sont initialisées à r.
- On rutilisera des matrices : :
  - `Array.make_matrix : int -> int -> 'a -> 'a array array` permet de créer une matrice : `Array.make_matrix m n r` crée une matrice de taille $m \times n$ dont toutes les cases sont initialisées à r.
  - Si `a` est une matrice de taille $m \times n$, le nombre de lignes, m, est calculé par `Array.length a` et le nombre de colonnes, n, est calculé par `Array.length a.(0)`. Le termes d'indices i et j avec $0 \leqslant i < m$ et $0 \leqslant j < n$ est accessible par `a.(i).(j)`.
- Enfin, une opération de base sur les listes :
  - `List.rev : 'a list -> 'a list` renvoie le miroir de la liste passée en argument (cette fonction a une complexité linéaire en la taille de la liste).

Dans l'ensemble du sujet, il sera fréquemment question de matrices de flottants. Pour simplifier l'écriture du type des fonctions manipulant de telles matrices, nous définissons le type `matrix` :

```
type matrix = float array array;;
```

Exemples de fonctions matricielles

Deux matrices doivent avoir la même taille si on veut calculer leur somme. On peut tester la compatibilité avec la fonction `somme_possible : matrix -> matrix -> bool`

```
let somme_possible a b =
  let na = Array.length a in
  let ma = Array.length a.(0) in
  let nb = Array.length b in
  let mb = Array.length .(0) in
  (na = nb) && (ma = mb);;
```

On calcule la somme de deux matrices, sans vérification de compatibilité, par une fonction `somme : matrix -> matrix -> matrix`

```
let somme a b =
  let n = Array.length a in
  let m = Array.length a.(0) in
  let c = Array.make_matrix n m 0.0 in
  for i = 0 to (n-1) do
    for j = 0 to (m-1) do
      c.(i).(j) <- a.(i).(j) +. b.(i).(j) done done;
  c;;
```

III.1 Calcul matriciel

Question 13 - Matrice identité
Donner une fonction OCaml `identite : int -> matrix` prenant en entrée un entier $n \in \mathbb{N}^*$ et renvoyant la matrice identité de $\mathcal{M}_{n,n}(\mathbb{R})$.

Question 14 - Compatibilité
Donner une fonction OCaml `mul_possible : matrix -> matrix -> bool` prenant en entrée deux matrices A et B et renvoyant `true` si et seulement si les dimensions de A et B sont telles que $A \times B$ est bien définie.

Question 15 - Produit
Donner une fonction OCaml `mul : matrix -> matrix -> matrix` qui calcule le produit $A \times B$ des deux matrices A et B de flottants passées en argument. Si A est de dimension $m \times n$ et B de dimension $n \times p$, on impose que la fonction `mul` effectue $m \times n \times p$ multiplications de flottants. On ne vérifiera pas la compatibilité

Dans l'ensemble du sujet, on utilisera la fonction `mul` à chaque fois qu'il s'agit de calculer le produit de deux matrices, sans chercher à l'améliorer.

III.2 Produits de plusieurs matrices

Dans cette partie, on s'intéresse à la multiplication d'une suite finie de matrices réelles $\left(B^i\right)_{1 \leqslant i \leqslant n}$. Soit $k_0, \ldots, k_n$ la suite finie d'entiers tels que $B^i \in \mathcal{M}_{k_{i-1},k_i}(\mathbb{R})$.

Question 16 - Multiplication par la gauche
Donner une fonction OCaml `mul1 : matrix list -> matrix` qui prend en entrée une liste non vide de matrices $\left[B^1; \ldots; B^n\right]$ et renvoie leur multiplication (en utilisant `mul`) via un parenthésage à droite : $B^1 \times \left(B^2 \times (\ldots \times B^n)\right)$.
Dans le cas d'une liste vide la fonction renverra une erreur (`failwith`).

Question 17 - Multiplication par la droite
Donner une fonction OCaml `mul2 : matrix list -> matrix` qui prend en entrée une liste non vide de matrices $\left(B^1, \ldots, B^n\right)$ et renvoie leur multiplication (en utilisant `mul`) via un parenthésage à gauche : $\left((B^1 \times \ldots) \times B^{n-1}\right) \times B^n$.

Question 18 - Un exemple
On suppose, dans cette question, qu'on a $n = 3$, $k_0 = 2$, $k_1 = 4$, $k_2 = 2$ et $k_3 = 4$ donc $B^1 \in \mathcal{M}_{2,4}(\mathbb{R})$, $B^2 \in \mathcal{M}_{4,2}(\mathbb{R})$ et $B^3 \in \mathcal{M}_{2,4}(\mathbb{R})$.
Combien de multiplications de flottants demandent les calculs de `mul1 liste_b` et de`mul2 liste_b` si `liste_b` est la liste $[B^1; B^2; B^3]$?

III.3 Produit optimal

La question précédente montre que le nombre de multiplications de flottants varie selon l'ordre de multiplication des matrices, indiqué par le parenthésage.
On cherche à écrire un programme effectuant le parenthésage optimal sur les B^i pour un calcul efficace de leur multiplication $B^1 \times \cdots \times B^n$.
Pour chaque $1 \leqslant i \leqslant j \leqslant n$, on note $B^{i,j}$ le produit de matrices $B^i \times \cdots \times B^j$, et $p^{i,j}$ le nombre minimum de multiplications de flottants nécessaires pour calculer $B^{i,j}$.
En particulier on a $B^{i,i} = B^i$ et $p^{i,i} = 0$ et aussi $p^{i,i+1} = k_{i-1} \cdot k_i \cdot k_{i+1}$.
Les différentes manières de calculer $B^{i,j}$ doivent finir par un des produits

$$B^{i,i} \times B^{i+1,j},\ B^{i,i+1} \times B^{i+2,j},\ \ldots,\ B^{i,j-2} \times B^{j-1,j} \text{ ou } B^{i,j-1} \times B^{j,j}$$

Question 19 - Lemme
Si $B^{i,r}$ et $B^{r+1,j}$ avec $i \leqslant r < j$ ont été calculés de manière optimale, prouver que $B^{i,r} \times B^{r+1,j}$ est calculé avec $p^{i,r} + p^{r+1,j} + k_{i-1} \cdot k_r \cdot k_j$ multiplications.

Question 20 - Relation de récurrence
En déduire une relation permettant de calculer $p^{i,j}$ en fonction des k_r avec $i - 1 \leqslant r \leqslant j$ et des valeurs des $p^{i,r}$ et des $p^{r+1,j}$ avec $i \leqslant r < j$.

Question 21 - Algorithme brutal
Donner une fonction OCaml `min_mul: int array -> int` qui prend en entrée le tableau $k = [k_0; \ldots; k_n]$ des tailles de chacune des matrices, et renvoie $p^{1,n}$ en faisant des appels récursifs pour calculer les $p^{i,j}$, sans stockage d'information supplémentaire. Montrer que si, pour tout $i, k_i \geqslant 2$, la complexité en temps de `min_mul` est au moins exponentielle en n, c'est-à-dire en $\Omega(2^n)$.

Question 22 - Complexité (question difficile)
Montrer que si, pour tout $i, k_i \geqslant 2$, la complexité en temps de `min_mul` est au moins exponentielle en n, c'est-à-dire minorée par une fonction de la forme $K \cdot a^n$ avec $a > 1$..

Question 23 - Algorithme rapide
Donner une fonction OCaml `min_mul_opt : int array -> int` qui calcule le même nombre, mais où une matrice auxiliaire de taille $n \times n$, initialement nulle, permet de stocker les valeurs de $p^{i,j}$ déjà calculées.
Quelle est la complexité en temps de `min_mul_opt` en fonction de n ?

Solutions

Solution de la question 1 - Appartenance

```
let rec appartient x liste =
  match liste with
  |[] -> false
  |t::q when t = x -> true
  |t::q -> appartient x q;;
```

Solution de la question 2 - Adjonction
Pour la première méthode

```
let rec add x liste =
  if appartient x liste
  then liste
  else match liste with
       |[] -> [x]
       |t::q when x < t -> x::liste
       |t::q -> t::(add x qx);;
```

le seconde méthode est en fait plus simple

```
let rec add x liste =
  match liste with
  |[] -> [x]
  |t::q when t = x -> liste
  |t::q when x < t -> x::liste
  |t::q -> t::(add x q);;
```

Solution de la question 3 - Beaucoup de comparaisons
La fonction `add` peut comparer x à chaque élément de la liste ; c'est le cas quand x est supérieur à tous les élément de la liste.
Ainsi `union` peut comparer chaque élément de `l1` à chaque élément de `l2` ce qui fait $n_1.n_2$ comparaisons. Ce nombre est atteint quand tous les éléments de `l1` sont supérieurs aux éléments de `l2` : on peut choisir `l1 = [3; 4; 5]` et `l2 = [0; 1; 2]`.

Solution de la question 4 - Une meilleure foncction

```
let rec union2 l1 l2 =
  match l1, l2 with
  |[], _ -> l2
  |_, [] -> l1
  |t1::q1, t2::q2 when t1 = t2 -> t1::(union2 q1 q2)
  |t1::q1, t2::q2 when t1 < t2 -> t1::(union2 q1 l2)
  |t1::q1, t2::q2 -> t2::(union2 l1 q2);;
```

Solution de la question 5 - Intersection

```
let rec intersection l1 l2 =
  match l1, l2 with
  |[], _ -> []
  |_, [] -> []
  |t1::q1, t2::q2 when t1 = t2 -> t1::(intersection q1 q2)
  |t1::q1, t2::q2 when t1 < t2 -> intersection q1 l2
  |t1::q1, t2::q2 -> intersection l1 q2;;
```

Solution de la question 6 - Parties

```
let rec parties n p =
  if p = 0
  then [[]]
  else if n < p
       then []
       else let p1 = parties (n-1) p in
            let p2 = parties (n-1) (p-1) in
            p1@(List.map (add (n-1)) p2);;
```

```
let rec ajouter x liste =
  match liste with
  |[] -> []
  |t::q -> (add x t)::(ajouter x q);;

let rec parties n p =
  if p = 0
  then [[]]
  else if n < p
       then []
       else let p1 = parties (n-1) p in
            let p2 = parties (n-1) (p-1) in
            p1@(ajouter (n-1) p2);;
```

Solution de la question 7 - Minimum global

```
let minimum tab =
  let n = Array.length tab in
  let valmin = ref tab.(0) in
  for i = 1 to (n-1) do
    if tab.(i) < !valmin
    then valmin := tab.(i) done;
  !valmin;;
```

Solution de la question 8 - Une propriété

Si on a $t_0 \leqslant t_1$ alors t_0 est un minimum local, c'est le premier.
Si on a $t_0 > t_1 > \cdots > t_i$ et $t_i \leqslant t_{i+1}$ alors t_i est un minimum local et t_k n'en est pas un pour $k < i$, i est le premier minimum local.
Si on a $t_0 > t_1 > \cdots > t_{n-1}$, le tableau est strictement décroissant
et t_{n-1} est l'unique minimum local.

Solution de la question 9 - Premier minimum local

```
let premier_ml tab =
  let n = Array.length tab in
  let i = ref 0 in
  while !i < n-1 && tab.(!i) > tab.(!i+1) do i := !i + 1 done;
  tab.(!i);;
```

Solution de la question 10 - Cas des listes

```
let rec premier_ml_rec liste =
  match liste with
  |[] -> failwith "La liste est vide, pas de minimum local"
  |[a] -> a
  |a::b::q when a <= b -> a
  |t::q -> premier_ml_rec q;;
```

Solution de la question 11 - Séparation

Un minimum local de $[|t_0;\dots;t_{k-1}|]$ est un minimum local du tableau sauf s'il est atteint en t_{k-1}. Mais comme on a $t_{k-1} < t_k$, t_{k-1} reste un minimum local. De même pour l'autre cas.

Solution de la question 12 - Algorithme rapide

On écrit une fonction auxiliaire qui reçoit les bornes dans lesquelles chercher un minimum local. Les cas terminaux sont ceux où il reste un seul élément ou deux car on va chercher au milieu et celui-ci ne doit pas être une des bornes.

```
let minimum_local tab =
  let rec aux_ml i j =
    if j > i + 1
    then let k = (i+j)/2 in
         if tab.(k) > tab.(k-1)
         then aux_ml i (k-1)
         else if tab.(k) > tab.(k+1)
              then aux_ml (k+1) j
              else tab.(k)
    else if j = i + 1
         then min tab.(j) tab.(i)
         else tab.(i)
  in aux_ml 0 (Array.length tab -1);;
```

Solution de la question 13 - Matrice identité

```
let identite n =
  let m array.make_matrix n n 0.0 in
  for i = 0 to (n-1) do m.(i).(i) <- 1.0 done;
  m;;
```

Solution de la question 14 - Compatibilité

```
let mul_possible a b =
  Array.length a.(0) = Array.length b ;;
```

Solution de la question 15 - Produit

```
let mul a b =
  let m = Array.length a in
  let n = Array.length b in
  let p = Array.length b.(0) in
  let c = Array.make_matrix m p 0.0 in
  for i = 0 to (m-1) do
    for j = 0 to (p-1) do
      for k =0 to (n-1) do
        c.(i).(j) <- c.(i).(j) +. a.(i).(k) *. b.(k).(j) done
    done
  done;
  c;;
```

Solution de la question 16 - Multiplication par la gauche

```
let rec mul1 liste =
  match liste with
  |[] -> failwith "La liste ne doit pas être vide"
  |[m] -> m
  |m::reste -> mul m (mul1 reste);;
```

Solution de la question 17 - Multiplication par la droite

```
let mul2 liste =
  let aux_mul reste prod =
    mach reste with
    |[] -> prod
    |m::q -> aux_mul q (mul prod m) in
  match liste with
  |[] -> failwith "La liste ne doit pas être vide"
  |m::reste -> aux_mul reste m;;
```

Solution de la question 18 - Un exemple
On commence par calculer $B^2 \times B^3$ qui demande $4 \cdot 2 \cdot 4 = 32$ multiplications et donne une matrice de $B^{2,3} \in \mathcal{M}_{4,4}(\mathbb{R})$ puis $B^1 \times B^{2,3}$ qui demande $2 \cdot 4 \cdot 4 = 32$ multiplications d'où un total de 64 multiplications.
$(B^1 \times B^2) \times B^3$ demande $2 \cdot 4 \cdot 2 + 2 \cdot 2 \cdot 4 = 32$ multiplications.

Solution de la question 19 - Lemme
Les calculs de $B^{i,r}$ et $B^{r+1,j}$ demandent respectivement $p^{i,r}$ et $p^{r+1,j}$ multiplications et sont de taille $k_{i-1} \times k_r$ et $k_r \times k_j$. Leur produit demandera $k_{i-1} \cdot k_r \cdot k_j$ multiplications en plus.

Solution de la question 20 - Relation de récurrence
On a donc $p^{i,j} = \text{Min}\big(p^{i,r} + p^{r+1,j} + k_{i-1} \cdot k_r \cdot k_j \ ; \ i \leqslant r \leqslant j-1\big)$.

Solution de la question 21 - Algorithme brutal

```
let min_mul k =
  let n = Array.length k in
  let rec pm i j =
    if i = j
    then 0
    else begin
      let p0 = ref ((pm i (j-1)) + k.(i-1)*k.(j-1)*k.(j)) in
      for r = i to (j-2) do
        let p = (pm i r)+(pm (r+1) j)+k.(i-1)*k.(r)*k.(j) in
        p0 := min !p0 p done;
      !p0 end in
  pm 1 (n-1);;
```

Solution de la question 22 - Complexité (question difficile)
La complexité ne dépend que $h = j - i$, on le note $C(h)$.
On a $C(0) = 0$, $C(1) = 4$ (une affectation, une addition et 2 multiplications) et, pour $h \geqslant 2$
$C(h) = C(h-1) + 3 + \sum_{k=0}^{h-2} \big(C(k) + C(h-k-1) + 6\big) = 6h - 3 + 2\sum_{k=1}^{h-2} C(k) + C(h-1).$
On a donc $C(h+1) - C(h) = C(h) + C(h-1) + 6$ pour $h \geqslant 2$,
donc $C(h) = A \cdot (1+\sqrt{2})^h + B \cdot (1-\sqrt{2})^h + 3$ pour $h \geqslant 2$ avec $A > 0$ et $1 + \sqrt{2} > 2$.
La complexité est bien exponentielle.

Solution de la question 23 - Algorithme rapide

```
let min_mul k =
  let n = Array.length k in
  let p = Array.make_matrix n n 0 in
  for h = 1 to (n-2) do
    for i = 1 to (n-1-h) do
      let j = i + h in
      p.(i).(j) <- p.(i).(j-1) + k.(i-1)*k.(j-1)*k.(j);
      for r = i to (j-2) do
      let pp = p.(i).(r)+p.(r+1).(j)+k.(i-1)*k.(r)*k.(j) in
        p.(i).(j) <- min p.(i).(j) pp done done done;
  p.(1).(n-1);;
```

La complexité est cubique.