Colle 2 : Tableaux avec le langage Ocaml

28 septembre

1 Le module Array

Ocaml dispose d'un module `Array` qui permet de manipuler des tableaux.

Nous allons commencer par tester un certain nombre d'instructions afin de comprendre les règles d'utilisation de ces tableaux. Demandez-vous ce que va faire le langage avant d'exécuter.

```
[|5;7;2;3|];;
Array.length [|5;7;2;3|];;
let t1=[|5;4;5;7|];;
t1;;
t1.(0);;
t1.(-1);;
t1.(4);;
t1.(0) <- 3;;
t1;;

let t2 = t1 and t3 = [|3;4;5;6|];;
t1;;
t2;;
t2.(3) <- 9;;
t2;;
t1;;
t3;;
t2.(3)<-6;;
t2;;
t1;;
t3;;
t1=t2;;
t1=t3;;
t1==t2;;
t1==t3;;

(* la n\'egation de = est <> tandis que la n\'egation de == est != *)

t1<>t2;;
t1<>t3;;
t1!=t2;;
t1!=t3;;

let test t = t.(0) <- 9;;
t3;;
test t3;;
t3;;
t2;;
t1;;

test t2;;
t2;;
t1;;

let t4 = Array.make 5 42;;
t4;;
```

```
let t5 = [| [|1;2;3|]; [|4;5;6|];|];;

t5.(1).(2);;

let t6 = Array.copy t4;;
t6.(1)<-8;;
t6;;
t4;;
test t6;;
t6;;
t4;;

let t7 = Array.make 3 (t1);;
t7.(0)==t7.(1);;
t7.(0).(0)<- 12;;
t7;;

let t8=Array.make_matrix 2 3 0;;
t8.(1).(1)<-15;;
t8;;
```

On retiendra que :

1. Appeler des cases inexistantes du tableau crée des erreurs.
2. Une commande du type $t1 = t2$ ne crée pas de copie et toute action sur l'un des deux tableaux transforme le second. Il faut utiliser la commande :
3. Lorsque l'on passe un tableau en paramètre d'une fonction, la fonction peut le modifier. C'est ce qu'on appelle un effet de bord car on ne voit pas cette effet par la valeur de retour de la fonction.
4. Lorsque l'on travaille avec des tableaux à double entrée, une déclaration naive risque d'entrainer des problèmes de copie. Il faut utiliser la commande :

Notez ici les commandes que vous venez d'apprendre :

2 Quelques fonctions sur les tableaux :

▷ **Question 1.** Déclarer un tableau de deux manières différentes (en utilisant `Array.make` et en déclarant explicitement toutes les valeurs du tableau). Puis évaluer la taille de ces tableaux. ◁

▷ **Question 2.** Soit n un entier naturel non-nul. Ecrire une fonction qui prend en entrée un tableau d'entiers de taille $2n$ et renvoie un tableau de taille n dont la case i contient l'élément de la case $2i$ du tableau donné en entrée.
◁

▷ **Question 3.** Ecrire une fonction `appartient` qui prend en entrée un tableau T et un entier n et qui renvoie true si n appartient à T et false sinon. La signature de la fonction sera `'a array -> 'a -> bool = <fun>` ◁

▷ **Question 4.** Ecrire une fonction `maximum` qui prend en entrée un tableau d'entiers non vide et renvoie le maximum des éléments contenus dans ce tableau. La signature de la fonction sera `'a array -> 'a = <fun>`
On écrira ensuite une fonction `minimum` sur le même modèle. ◁

▷ **Question 5.**
Ecrire une fonction de signature `echange : 'a array -> int -> int -> unit = <fun>`, qui étant donné un tableau et deux indices i et j échange le contenu des cases d'indices i et j du tableau.
◁

▷ **Question 6.**
Ecrire une fonction de signature `affiche_tab : int array -> unit = <fun>`, dont l'appel à `affiche_tab t` aura pour seul effet d'afficher à l'écran les éléments du tableau `t` séparés par un espace et sur une même ligne. On terminera la fonction par `print_newline()` pour forcer CAML à aller à la ligne. On pourra utiliser la fonction `print_int` permettant l'affichage d'un nombre entier, ainsi que la fonction `print_string` pour afficher une chaîne de caractères.
◁

▷ **Question 7.**
Ecrire une fonction de signature `retourne : 'a array -> unit = <fun>`, qui étant donné un tableau le modifie de sorte que ses éléments soient rangés dans l'ordre inverse. On pourra utiliser la fonction `echange` préalablement définie.
◁

▷ **Question 8.**
Ecrire une fonction de signature `miroir : 'a array -> 'a array = <fun>`, qui étant donné un tableau non vide retourne un nouveau tableau composé des mêmes éléments mais rangés dans l'ordre inverse. Quelle est la différence avec la question précédente ? ◁

▷ **Question 9.** Ecrire une fonction de signature `map_tab : ('a -> 'b) -> 'a array -> 'b array = <fun>`, qui étant donné une fonction f et un tableau t retourne le tableau des images par f des éléments de t rangées dans le même ordre. On suppose t non vide. ◁

Voici une liste de fonctions que l'on pourrait vous demander d'utiliser après vous en avoir rappeler les spécificités (les fonctions/commandes vues dans la première partie doivent quand à elles être connues.) Tester ces fonctions et notez ce qu'elles font.

```
Array.mem : 'a -> 'a array -> bool

Array.exists : ('a -> bool) -> 'a array ->bool

Array.for_all : ('a->bool) -> 'a array -> bool

Array.init : int -> (int -> 'a) -> 'a array

Array.map : ('a -> 'b) -> 'a array -> 'b array

Array.iter : ('a -> unit) -> 'a array -> unit
```

3 Autour des permutations :

On décide à présent de représenter les permutations σ de $\{1,\dots n\}$ par un tableau tel que sa i-ieme case contienne la valeur de $\sigma(i)$. Ainsi la permutation $\begin{pmatrix}1&2&3&4\\3&2&4&1\end{pmatrix}$ sera représentée par le tableau `[|3;2;4;1|]`.

▷ **Question 10.**
Ecrire une fonction de signature `identite : int -> int array = <fun>` qui à tout entier n associe la permutation de $\{1,\dots,n\}$ vérifiant $\forall i \in \{1,\dots,n\}, \sigma(i) = i$. On pourra chercher à utiliser la fonction `Array.init`. ◁

▷ **Question 11.**
Ecrire une fonction de signature `est_permutation : int array -> bool = <fun>` qui retourne true si le tableau de n entiers passé en paramètre représente une permutation et false sinon. On pourra chercher à utiliser certaines des fonctions présentées dans la partie précédente. ◁

Nous allons maintenant chercher à générer toutes les permutations de l'ensemble $\{1,\dots,n\}$ pour un entier naturel non-nul n fixé.
On muni l'ensemble des permutations de l'ordre lexicographique (comme dans un dictionnaire), ainsi les permutations de $\{1,2,3\}$ sont rangées dans cet ordre :
`[|1;2;3|] < [|1;3;2|] < [|2;1;3|] < [|2;1;3|] < [|3;1;2|] < [|3;2;1|]`.

Méthode pour générer toutes les permutations :
La première permutation est toujours l'application identité et on cherche à passer d'une permutation t à la suivante.
Pour cela nous pouvons effectuer les 4 étapes suivantes :

1. Déterminer le plus grand indice i tel que $t.(i) < t.(i+1)$.
2. Déterminer l'indice $j > i$ de la case de plus petite valeur parmi celles ayant une valeur supérieure à $t.(i)$.
3. Echanger le contenu des cases d'indice i et j.
4. Ranger les cases d'indice strictement supérieur à i dans l'ordre croissant de leurs valeurs.

En appliquant l'algorithme à la main, déterminer la permutation suivant `[|4;5;3;6;2;1|]`. Quelle est la dernière permutation ?

▷ **Question 12.** Ecrire une fonction de signature `indice_etape1 : 'a array -> int = <fun>` qui étant donné une permutation retourne l'indice i de la première étape. ◁

▷ **Question 13.** Ecrire une fonction de signature `indice_etape2 : 'a array -> int = <fun>` qui étant donné une permutation t retourne l'indice de la case ayant la plus petite valeur supérieure à `t.(i)` (où i est l'indice renvoyé par la fonction `indice_etape1`) parmi les cases numérotées de $i+1$ à $n-1$. ◁

▷ **Question 14.** Ecrire une fonction de signature `indice_mini : 'a array-> int -> int = <fun>` qui étant donné une permutation t et i l'indice d'une case retourne l'indice de la case de plus petite valeur parmi les cases numérotées de $i+1$ à $n-1$. ◁

▷ **Question 15.** Ecrire une fonction de signature `tri_etape4 : 'a array -> int -> unit = <fun>` qui étant donné une permutation t et un indice i range dans l'ordre croissant les cases d'indice strictement supérieur à i. On pourra chercher l'indice du plus petit élément d'indice strictement supérieur à i pour le mettre en première place et continuer de la même manière avec les cases restantes. Cette fonction pourra être récursive. ◁

▷ **Question 16.** Ecrire une fonction de signature suivante : `'a array -> unit = <fun>` qui transforme une permutation en sa suivante. ◁

▷ **Question 17.** Ecrire une fonction de signature `permutations : int -> unit = <fun>` qui affiche les tableaux de toutes les permutations de $\{1,\dots,n\}$. On partira de l'identité et on appliquera la fonction précédente un certain nombre de fois. Il y a deux approches pour savoir quand s'arrêter : compter le nombre total de permutations ou identifier la dernière permutation selon l'ordre considéré ici. ◁

4 Application au problème du voyageur de commerce :

Un commercial dispose d'une liste de villes dans lesquelles il doit se rendre pour affaires avant de revenir chez lui. L'objet du problème est de déterminer dans quel ordre il doit visiter ces villes de façon à minimiser la distance totale parcourue.

On considère n villes numérotées de 0 à $n-1$ et on suppose (sans perte de généralité) que le commercial démarre de la ville numérotée 0. On représente alors un ordre de parcours des villes 1 à $n-1$ au moyen d'un tableau contenant leur numéro selon leur ordre de visite. Ce tableau représente en fait une permutation de l'ensemble $\{1,\dots,n\}$.

On se donne aussi un tableau D dont l'élément en ligne i et en colonne j (en commençant la numérotation à zéro) représente la distance (en kilomètres) à parcourir pour rejoindre la ville numéro j depuis la ville numéro i. On représentera ce tableau à double entrée sous OCAML au moyen d'un tableau à deux dimensions de type `int array array`.

Pour tester nos fonctions, on considèrera le tableau des distances ci-dessous :

```
let dist_data =
        [|
          [|   0 ; 124 ; 120 ; 144 ; 335 ; 315 ; 347 ; 353 ; 408 ; 465 ; 679 ; 584 |];
          [| 124 ;   0 ; 219 ; 291 ; 495 ; 441 ; 410 ; 311 ; 540 ; 596 ; 799 ; 661 |];
          [| 120 ; 219 ;   0 ; 266 ; 397 ; 315 ; 217 ; 302 ; 300 ; 410 ; 555 ; 449 |];
          [| 144 ; 291 ; 266 ;   0 ; 196 ; 280 ; 474 ; 483 ; 532 ; 476 ; 813 ; 724 |];
          [| 335 ; 495 ; 397 ; 196 ;   0 ; 212 ; 604 ; 644 ; 524 ; 406 ; 860 ; 860 |];
          [| 315 ; 441 ; 315 ; 280 ; 212 ;   0 ; 490 ; 599 ; 327 ; 195 ; 649 ; 649 |];
          [| 347 ; 410 ; 217 ; 474 ; 604 ; 490 ;   0 ; 295 ; 279 ; 429 ; 414 ; 257 |];
          [| 353 ; 311 ; 302 ; 483 ; 644 ; 599 ; 295 ;   0 ; 592 ; 740 ; 704 ; 472 |];
          [| 408 ; 540 ; 300 ; 532 ; 524 ; 327 ; 279 ; 592 ;   0 ; 164 ; 376 ; 375 |];
          [| 465 ; 596 ; 410 ; 476 ; 406 ; 195 ; 429 ; 740 ; 164 ;   0 ; 539 ; 556 |];
          [| 679 ; 799 ; 555 ; 813 ; 860 ; 649 ; 414 ; 704 ; 376 ; 539 ;   0 ; 246 |];
          [| 584 ; 661 ; 449 ; 724 ; 860 ; 649 ; 257 ; 472 ; 375 ; 556 ; 246 ;   0 |];
        |]
;;
```

▷ **Question 18.** Ecrire une fonction qui étant donné un parcours et une matrice des distances renvoie la distance totale à parcourir pour le commercial.

La signature de la fonction sera `distance : int array -> int array array -> int = <fun>`.

On rappelle que le voyageur doit, à la fin de son périple, revenir dans sa ville de départ (la distance parcourue en visitant les villes dans leur ordre de numérotation doit donner 3927). ◁

▷ **Question 19.** Ecrire une fonction de signature `meilleure_solution : int array array -> int=<fun>` qui étant donné un tableau de distances renvoie la distance du plus petit parcours parmi tous les parcours possibles (avec le tableau donné ici le meilleur parcours aura une distance de 2730km). Compte tenu du nombre très important de parcours possibles à tester (combien ?), votre calcul pourra prendre quelques (10/1s5) minutes. ◁

5 Chaînes de caractères :

En OCAML les chaînes de caractères sont assimilables à des tableaux de caractères. Une chaîne de caractères peut être saisie sous forme d'une suite de caractères délimitée par ". Pour accéder au caractère d'indice i d'une chaîne s on écrira `s.[i]`. On notera que l'on ne peut pas modifier un carcatère d'une chaîne de caractères, on dit que c'est un type immuable. La fonction `String.length` permettra d'obtenir le nombre de caractères d'une chaîne et la commande `String.make n c` retournera une chaîne de longueur n du caractère c. On prendra garde à bien distinguer une chaîne de caractères ne contenant qu'un seul caractère du caractère lui même : `'c'` vs `"c"`.

Les chaînes de caractères sont munies de la relation d'ordre lexicographique.

L'opérateur `^` permet de concaténer les chaînes de caractères.

On pourra tester les commandes suivantes afin d'illustrer ce propos :

```
let s = "bonjour";;
s<"chat";;
s.[3];;
s;;
String.length s;;
s>"ananas";;
s<"abandon";;
"Bonjour"^"!";;
let s = 'c' in
s="c";;
print_string("Hello");;
```

▷ **Question 20.** Écrire une fonction `rigolo` qui prend en entrée une chaîne de caractères C et qui renvoie la chaîne de caractères obtenue à partir de C en remplaçant toutes les lettres 'a' en lettre 'o'. (on pourra utiliser `String.map`). ◁

▷ **Question 21.** Ecrire une fonction de signature `melange : string -> int array -> string = <fun>` qui étant donné un mot de n caractères $m_1 m_2 \dots m_n$ et une permutation σ de l'ensemble $\{1, \dots, n\}$ retourne le mot $m_{\sigma(1)} m_{\sigma(2)} \dots m_{\sigma(n)}$. (On pourra utiliser la fonction String.init). ◁

▷ **Question 22.** Ecrire une fonction de signature `anagrammes : string -> int = <fun>` qui étant donné un mot composé de caractères différents deux à deux en affiche les anagrammes (au sens mathématique du terme). ◁

▷ **Question 23.** Ecrire une fonction qui prend en entrée une chaîne de caractères ne contenant que les caractères '(' et ')' et qui renvoie `true` si l'expression est bien parenthésée (au sens usuel) et `false` sinon.

Par exemple, (( )(( )) est bien parenthésée et (( )) est mal parenthésée.

La signature de la fonction sera : `string -> bool = <fun>`. ◁