Chapitre 8 : Tableaux en CAML- corrigé des exercices

EXERCICE 1

Écrire une fonction `element : int -> 'a list -> 'a` qui renvoie l'élément à une position donnée. Quelle est sa complexité ?

```
let rec element indice liste =
  match liste with
  | [] -> failwith "Indice non valable"
  | tete :: _ when indice = 0 -> tete
  | _ :: queue -> element (indice - 1) queue
;;
element : int -> 'a list -> 'a = <fun>
```

La complexité (au pire) est linéaire en la taille de la liste.

EXERCICE 2

Écrire une fonction `min_vect : 'a vect -> 'a` qui recherche le plus petit élément d'un tableau supposé non vide.

```
let min_vect tab =
  let mini = ref tab.(0) in
  for i = 1 to vect_length tab - 1 do
    mini := min !mini tab.(i)
  done;
  !mini
;;
min_vect : 'a vect -> 'a = <fun>

min_vect [|3; 2; 1; 10; 2|];;
- : int = 1
```

Remarque 1. *On peut aussi utiliser la programmation récursive.*

```
let min_vect tab =
  let rec parcours i =
    if i = vect_length tab - 1 then
      tab.(i)
    else
      min tab.(i) (parcours (i + 1))
  in
  parcours 0
;;
min_vect : 'a vect -> 'a = <fun>
```

La version récursive terminale est encore plus proche de la structure de la boucle (est-ce étonnant ?).

```
let min_vect tab =
  let rec parcours i acc =
    if i = vect_length tab then
      acc
    else
      parcours (i + 1) (min acc tab.(i))
  in
  parcours 0 tab.(0)
;;
min_vect : 'a vect -> 'a = <fun>
```

C'est peut-être un peu moins naturel et la boucle inconditionnelle est sans doute plus explicite. Mais ce type de construction s'avère nécessaire dès que l'on a besoin de manipuler également des listes (cf. DM) et il faut la connaître.

EXERCICE 3

Comment modifier la fonction précédente pour renvoyer plutôt l'indice du premier élément minimum rencontré ?

```
let index_min_vect tab =
  let mini = ref tab.(0) in
  let index = ref 0 in
  for i = 1 to vect_length tab - 1 do
    if tab.(i) < !mini then begin
      mini := tab.(i);
      index := i
    end
  done;
  !index
;;
index_min_vect : 'a vect -> int = <fun>

index_min_vect [|3; 2; 1; 10; 2|];;
- : int = 2
```

EXERCICE 4

Écrire une fonction `nb_occurrences` qui compte le nombre d'occurrences d'un élément dans un tableau.

```
let nb_occurences elem tab =
  let n = vect_length tab in
  let compte = ref 0 in
  for i = 0 to n - 1 do
    if elem = tab.(i) then incr compte;
  done;
  !compte
;;
nb_occurences : 'a -> 'a vect -> int = <fun>

nb_occurences 2 [|3; 2; 1; 10; 2|];;
- : int = 2
```

Voici également les exemples que nous avons étudiés en cours. Il est utile de revoir ces exemples pour être sûr d'avoir bien compris.

```
let v = [|0; 1; 2; 3|] in
    let w = v in
v.(0) <- 9;
w;;
- : int vect = [|9; 1; 2; 3|]

let v = [|0; 1; 2; 3|] in
let w = v in
w.(0) <- 9;
v;;
- : int vect = [|9; 1; 2; 3|]

let v = [|0; 1; 2; 3|] in
let w = copy_vect v in
w.(0) <- 9;
v;;
- : int vect = [|0; 1; 2; 3|]

let v = [|0; 1; 2; 3|] in
let w = v in
let w = [| 5; 6; 7|] in
v;;
- : int vect = [|0; 1; 2; 3|]

let v = [|0; 1; 2; 3|] in
let w = v in
let v = [| 5; 6; 7|] in
w;;
- : int vect = [|0; 1; 2; 3|]
```

```
let matrice = make_vect 2 (make_vect 2 1);;
matrice : int vect vect = [|[|1; 1|]; [|1; 1|]|]

(* Équivalent à *)

let matrice =
  let ligne = make_vect 2 1 in
  make_vect 2 ligne
;;
matrice : int vect vect = [|[|1; 1|]; [|1; 1|]|]

matrice.(0).(0);;
- : int = 1

matrice.(0).(0) <- 0;;
- : unit = ()

matrice;;
- : int vect vect = [|[|0; 1|]; [|0; 1|]|]
```

```
let matrice n v_init =
  let res = make_vect n [||] in
  for i = 0 to n - 1 do
    res.(i) <- make_vect n v_init
  done;
  res
;;
matrice : int -> 'a -> 'a vect vect

let m = matrice 2 1;;
m : int vect vect = [|[|1; 1|]; [|1; 1|]|]
m.(0).(0) <- 1; m;;
- : int vect vect = [|[|0; 1|]; [|1; 1|]|]
```