OCaml - Tableaux

Aubin SIONVILLE

MPI Clemenceau - 2021-2023

1 Bases

1.1 Premier zéro

Fonction calculant l'indice du premier zéro d'un tableau d'entiers avec une boucle **while**.

```
let premier_zero (t: int array): int =
    let n = Array.length t in
    let i = ref 0 in
    while !i < n && t.(!i) <> 0 do
        incr i
    done;
    if !i = n then -1 else !i;;
```

Fonction calculant l'indice du premier zéro d'un tableau d'entiers avec une boucle **for** arrêtée par une exception.

```
exception Zero of int;;
let premier_zero (t: int array): int =
    let n = Array.length t in
    try
        for i = 0 to n - 1 do
            if t.(i) = 0 then raise (Zero i)
        done; -1
    with Zero i -> i;;
```

Dernier zéro

Fonction calculant l'indice du dernier zéro d'un tableau d'entiers avec une boucle **while**.

```
let dernier_zero (t: int array): int =
    let n = Array.length t in
    let i = ref (n - 1) in
    while !i >= 0 && t.(!i) <> 0 do
        decr i
    done;
    if !i = -1 then -1 else !i;;
```

Existence d'un zéro

Fonction testant l'existence d'un zéro dans un tableau d'entiers avec une boucle **while**, arrêtée dès que possible sans utiliser d'exception.

```
let existe_zero (t: int array): bool =
    let res = ref false in
    let n = Array.length t in
    let i = ref 0 in
    while !i < n && not !res do
        if t.(!i) = 0 then res := true;
        incr i
    done;
    !res;;
```

Fonction testant l'existence d'un zéro dans un tableau d'entiers avec **Array.iter**, arrêtée par une exception.

```
exception Zero

let existe_zero (t: int array): bool =
    try
        Array.iter (fun x -> if x = 0 then raise Zero) t;
        false
    with Zero -> true;;
```

Somme des éléments d'un tableau

Fonction calculant la somme des éléments d'un tableau d'entiers avec une boucle **for**.

```
let somme (t: int array): int =
    let n = Array.length t in
    let s = ref 0 in
    for i = 0 to n - 1 do
        s := !s + t.(i)
    done;
    !s;;
```

Fonction calculant la somme des éléments d'un tableau d'entiers avec **Array.fold_left**.

```
let somme (t: int array): int =
    Array.fold_left (fun s x -> s + x) 0 t;;
```

Minimum et maximum

Fonction calculant l'indice du minimum d'un tableau d'entiers avec une boucle **for**. On lève une exception si le tableau est vide.

```
let indice_min (t: int array): int =
    let n = Array.length t in
    if n = 0 then raise (Invalid_argument "indice_min");
    else
        let i_min = ref 0 in
        let v_min = ref t.(0) in
        for i = 1 to n - 1 do
            if t.(i) < !v_min then
            begin
                i_min := i;
                v_min := t.(i)
            end
        done;
        !i_min;;
```

Fonction calculant l'indice du maximum d'un tableau d'entiers avec **Array.fold_left**. On lève une exception si le tableau est vide.

```
let indice_min (t: int array): int =
    let aux = (fun (i_max, v_max,i) ->
        if t.(i) > v_max then (i, t.(i), i + 1)
        else (i_max, v_max, i + 1))
    in let (i_max, _, _) = Array.fold_left aux (-1, -1, 0) t
    in i_max;;
```

Sommes cumulées

Fonction prenant en argument un tableau d'entier t et renvoyant le tableau des sommes cumulées de t.

```
let cumul (t: int array): int array =
    let n = Array.length t in
    if n = 0 then [||]
    else begin
        let cumul = Array.make n t.(0) in
        for i = 1 to n - 1 do
            cumul.(i) <- cumul.(i - 1) + t.(i)
        done;
        cumul
    end;;
```

Tableau de listes

Fonction prenant en argument un tableau de listes d'entiers t et le modifiant en ajoutant à chaque liste l'indice de la liste dans le tableau.

```
let ajoute_indice (t: int list array): unit =
    let n = Array.length t in
    for i = 0 to n - 1 do
        t.(i) <- i :: t.(i)
    done;;
```

Fonction prenant en argument un tableau de listes d'entiers t et un entier x et renvoyant un tableau de listes d'entiers obtenu en ajoutant x en tête de chaque liste de t.
On utilise **Array.map**.

```
let ajoute_x (t: int list array) (x: int): int list array =
    Array.map (fun l -> x :: l) t;;
```

Element majoritaire

Fonction prenant en argument un entier m et un tableau t d'entiers entre 0 et $m-1$ et renvoyant l'élément majoritaire de t.
On n'utilise pas de boucle **for** ou **while** et on a une complexité en $O(n+m)$.

```
let element_majoritaire (m: int) (t: int array): int =
    let tab_occurences = Array.make m 0 in
    Array.iter (fun x -> tab_occurences.(x) <- tab_occurences.(x) + 1) t;
    let aux = (fun (i_max, v_max, i) ->
        if x > v_max then (i, x, i + 1)
        else (i_max, v_max, i + 1))
    in let (i_max, _, _) = Array.fold_left aux (-1, -1, 0) tab_occurences
    in i_max;;
```