Tableaux

2 octobre 2023

Plan

Retours sur la semaine précédente

Tableaux

Algorithmes sur les tableaux

Matrices

Chaînes de Caractère

Retours sur la semaine précédente

Administratif

Il me manque toujours quelques élèves pour les e-mails d'information.

TP

Les consignes doivent être comprises, s'il vous manque des concepts (typage, signature, arguments...), il ne faut pas hésiter à poser la question.
Il faut *tester* autant que possible des choses pour voir quelles syntaxes sont acceptées.

Parenthèses et conditions

Attention, dans le cas d'un `if` et d'une séquence, vous devez mettre des parenthèses :

```
1 if 1= 0
2     then
3         print_int 1;
4         print_int 2
5     else
6         print_int 3
```

Parenthèses et conditions

Attention, dans le cas d'un `if` et d'une séquence, vous devez mettre des parenthèses :

```
1 if 1= 0
2     then
3         print_int 1;
4         print_int 2
5     else
6         print_int 3
```

```
1 if 1 = 0
2     then
3         (print_int 1;
4         print_int 2)
5     else
6         print_int 3
```

Séparateur (1)

On utilise le point-virgule pour séparer deux expression

```
1 a:= 0
2 b:= 1
```

Séparateur (1)

On utilise le point-virgule pour séparer deux expression

```
1 a:= 0
2 b:= 1
```

```
1 a:= 0;
2 b:= 1
```

Séparateur (2)

Pas de point-virgule à la fin d'une expression sans raison : la séquence sert exclusivement à séparer deux expressions, pas à mettre fin à une opération.

```
1 let echanger a b =
2     let r = !a in
3     b:= !a;
4     a:= r;
```

Séparateur (2)

Pas de point-virgule à la fin d'une expression sans raison : la séquence sert exclusivement à séparer deux expressions, pas à mettre fin à une opération.

```
let echanger a b =
    let r = !a in
    b:= !a;
    a:= r;
```

```
let echanger a b =
    let r = !a in
    b:= !a;
    a:= r
```

Séparateur (3)

Attention à ne pas mettre pas mettre le point-virgule aux mauvaises endroits :

```
1 let f r y= let a = ref !r ;
2 r:= y ; !r
```

Séparateur (3)

Attention à ne pas mettre pas mettre le point-virgule aux mauvaises endroits :

```
1 let f r y= let a = ref !r ;
2 r:= y ; !r
```

```
1 let f r y= let a = ref !r in
2 r:= y ; !r
```

Séparateur (4)

Pas de point-virgule avant le done, mais après :

```
let f n =
    for i=0 to n do
        print_int i;
        print_newline ();
    done
    n
```

Séparateur (4)

Pas de point-virgule avant le done, mais après :

```
let f n =
    for i=0 to n do
        print_int i;
        print_newline ();
    done
    n
```

```
let f n =
    for i=0 to n do
        print_int i;
        print_newline ()
    done ;
    n
```

Boucle

```
let fibonacci n =
let a = ref 0 in
let b = ref 1 in
for _ = 0 to n-1 do
    let c = !a + !b in
    a:= !b;
    b:= c
done ; !a
```

Boucle

```
let fibonacci n =
let a = ref 0 in
let b = ref 1 in
for _ = 0 to n-1 do
    let c = !a + !b in
    a:= !b;
    b:= c
done ; !a
```

```
let fibonacci n =
let a = ref 0 in
let b = ref 1 in
for _ = 0 to n-1 do
    b:= !a + !b;
    a:= !b - !a
done ; !a
```

Affichage (1)

```
let afficher_entre str1 x str2 =
    print_string str1;
    print_int x;
    print_string str2
```

```
afficher_entre "Un texte" 1 "Un autre texte"
```

```
Un texte1Un autre texte
```

Affichage (2)

```
let afficher_liste l =
    let rec aux l = match l with
    | [] -> ()
    | [a] -> print_int a
    | p::q -> print_int p ; print_string "; "; aux q
    in print_string "[" ;
    aux l ;
    print_string "]"
```

Affichage (3)

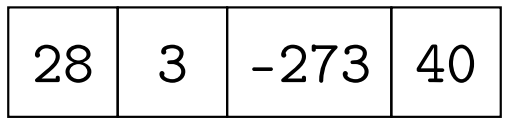
Solution impérative

```
let afficher_liste l =
    let vu_elem = ref false in
    let actuel = ref l in
    print_string "[";
    while !actuel <> [] do
        let p::q = !actuel in
        if !vu_elem then print_string "; ";
        print_int p;
        vu_elem:= true;
        actuel:= q
    done ;
    print_string "]"
```

Tableaux

Tableau

Un tableau est une structure de données qui permet de stocker une séquence de donnée mutable.



On peut accéder et modifier efficacement partout dans le tableau, mais on ne peut pas modifier la taille du tableau.

Définition d'un tableau en OCaml

On peut définir un tableau avec les délimiteurs ;, la différence avec les listes est que le tableau commence par [| et termine par |].

```
let t = [|0;1;2;3|]
```

```
val t : int array = [|0; 1; 2; 3|]
```

Tout comme les listes, tous les objets à l'intérieur d'un tableau doivent d'être d'un même type.

Accès à un tableau

On peut accéder à un élément d'un tableau avec la syntaxe `t.(k)` où `t` est le tableau et `k` est l'indice de l'élément.

```
let t = [|2;3;4;5|]
```

```
t.(2)
```

```
4
```

Attention : on commence à compter les indices à partir de 0 en OCaml comme dans la plupart des langages.

Modification d'un tableau

On peut modifier un élément d'un tableau avec la syntaxe suivante :

```
let t = [|2;3;4;5|]
```

```
t.(2) <- 6
```

```
t
```

```
[|2;3;6;5|]
```

Modules en OCaml

Un module est une bibliothèque en OCaml qui contient des fonctions (entre autre) que l'on peut utiliser sans avoir à les implémenter.
On accède à un module avec le nom du module (qui commence toujours par une majuscule) suivi de la fonction séparée par un point :

```
1 List.length [1;2;3]

3
```

Beaucoup de Module en OCaml

On peut citer quelques modules que nous aurions pu utiliser :

- Float [1] : fonctions sur des flottants ;
- List [2] : utilitaires pour les listes ;
- Array [3] : utilitaires pour les tableaux ;
- Printf [4] : utilitaires pour afficher et formater des chaînes de caractère.

Beaucoup des fonctions codée en TP sont déjà présentes dans la bibliothèque standard d'OCaml. On veillera à ne pas trop les utiliser quand l'objectif est justement leur implémentation.

---

1. https://v2.ocaml.org/api/Float.html
2. https://v2.ocaml.org/api/List.html
3. https://v2.ocaml.org/api/Array.html
4. https://v2.ocaml.org/api/Printf.html

Quelques fonctions utiles du module Array

- `Array.length t` renvoie la taille du tableau `t`

```
Array.length [|0;2;5|]
```

```
3
```

- `Array.make n x` renvoie un tableau de taille `n` dont tous les éléments sont égaux à `x`

```
Array.make 3 0
```

```
[|0;0;0|]
```

Quelques fonctions utiles du module Array

- `Array.length t` renvoie la taille du tableau `t`

```
Array.length [|0;2;5|]
```

```
3
```

- `Array.make n x` renvoie un tableau de taille `n` dont tous les éléments sont égaux à `x`

```
Array.make 3 0
```

```
[|0;0;0|]
```

Quelles sont les signatures de ces fonctions ?

Quelques fonctions utiles du module Array

- `Array.length t` renvoie la taille du tableau `t`

```
Array.length [|0;2;5|]
```

```
3
```

- `Array.make n x` renvoie un tableau de taille `n` dont tous les éléments sont égaux à `x`

```
Array.make 3 0
```

```
[|0;0;0|]
```

Quelles sont les signatures de ces fonctions ?

```
'a array -> int
```

```
int -> 'a -> 'a array
```

Ce seront les deux principales fonctions du module Array que nous utiliserons.

Algorithmes sur les tableaux

Programmation impérative sur les tableaux

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

Récursivités sur les tableaux

On ne peut pas faire de filtrages sur les tableaux en eux-même, mais on peut tout de même utiliser des fonctions récursives.

```
let somme t =
    let n = Array.length t in
    let rec aux i =
        if i = n then 0
        else t.(i) + aux (i+1)
    in aux 0
```

Exercice

Proposer une fonction de signature `int -> int array` qui, sur une entrée n, renvoie un tableau de taille n avec les entiers rangés par ordre croissant de 0 à $n-1$.
On pourra utiliser `Array.make n 0` pour initialiser un tableau d'entiers égaux à 0 de taille n.

Exercice

Proposer une fonction de signature `int -> int array` qui, sur une entrée n, renvoie un tableau de taille n avec les entiers rangés par ordre croissant de 0 à $n-1$.
On pourra utiliser `Array.make n 0` pour initialiser un tableau d'entiers égaux à 0 de taille n.

```
let ncroissant n =
    let reponse = Array.make n 0 in
    for i = 0 to n-1 do
        reponse.(i)<-i
    done ; reponse
```

Recherche Dichotomique

L'algorithme de la **recherche dichotomique** est un algorithme de recherche d'un élément dans un trié d'éléments.
À chaque étape de l'algorithme : on prend l'élément au centre du tableau, et on vérifie si l'élément est plus grand ou plus petit que l'élément que l'on cherche.

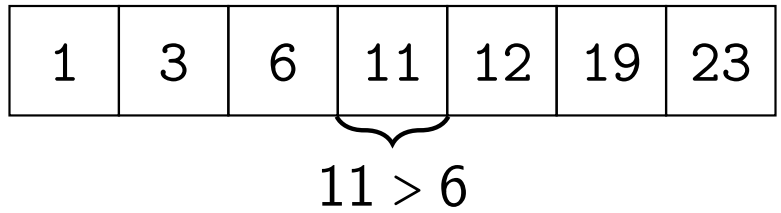
Exemple (1)

Recherche de 6 dans le tableau suivant :

1	3	6	11	12	19	23

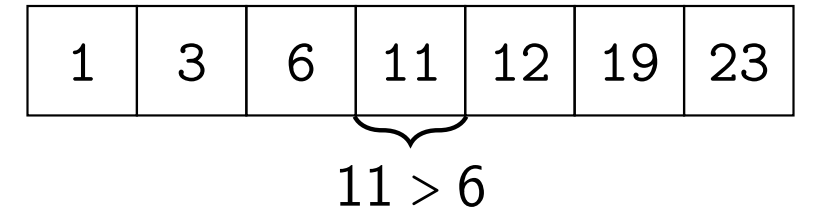
Exemple (2)

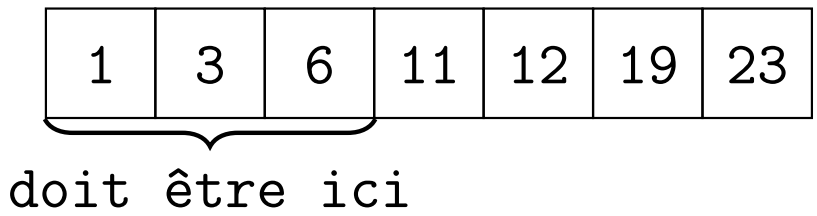
Recherche de 6 :



Exemple (2)

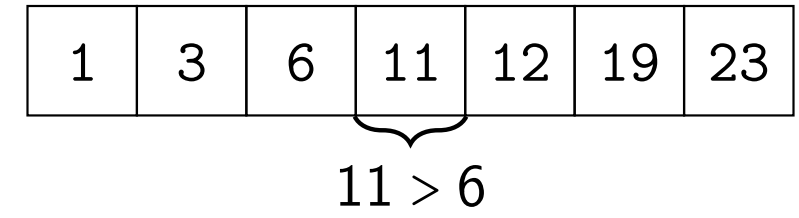
Recherche de 6 :

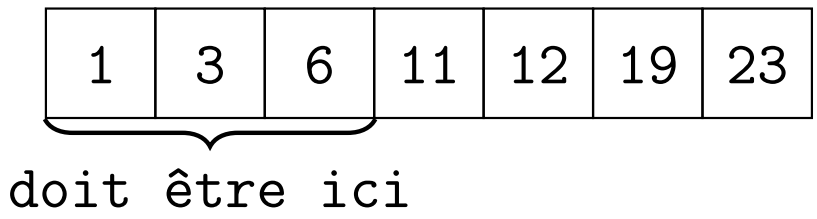


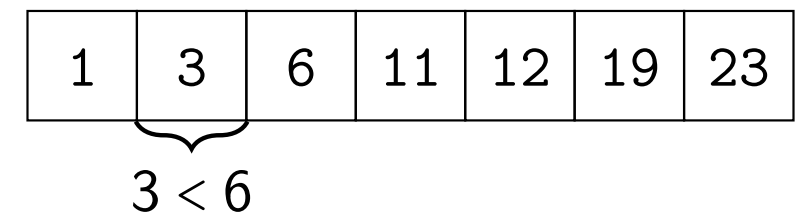


Exemple (2)

Recherche de 6 :

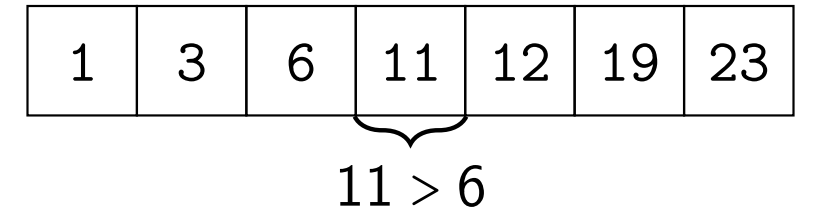


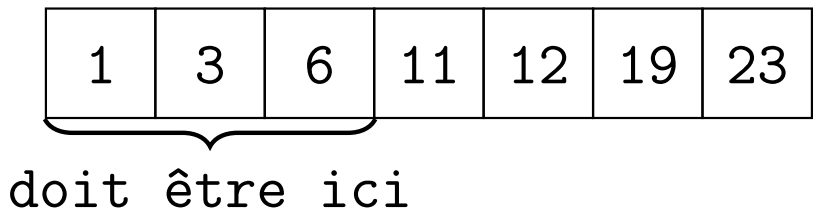


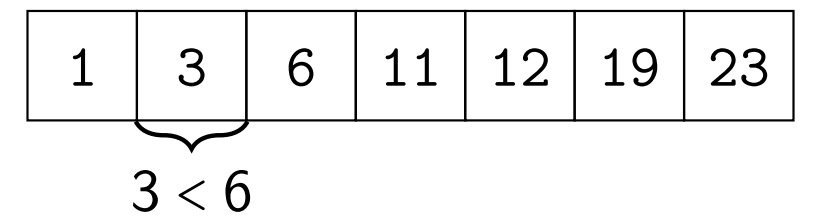


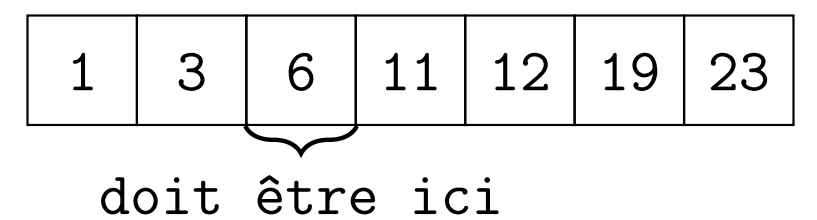
Exemple (2)

Recherche de 6 :



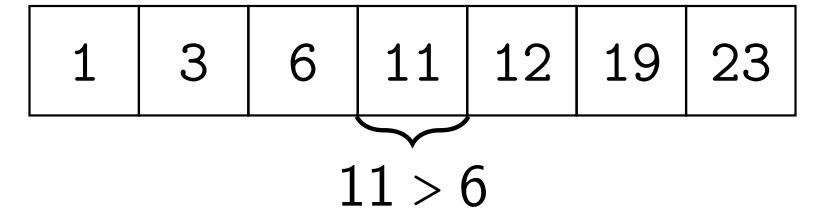


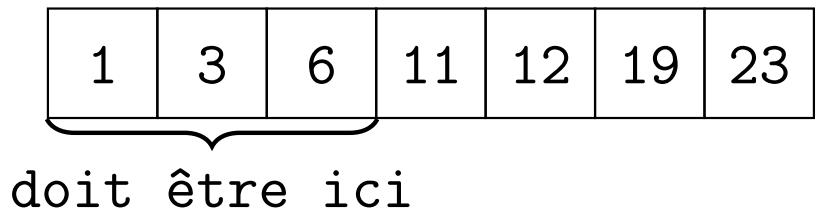


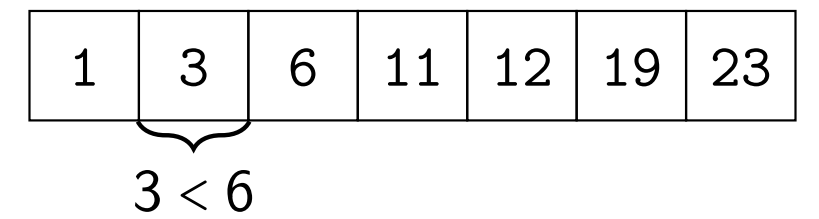


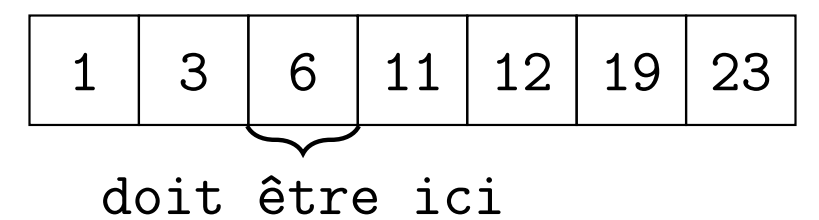
Exemple (2)

Recherche de 6 :









Que se passe-t-il lors de la recherche de 15 ?

Implémentation de la recherche dichotomique

```
1  let recherche t x =
2      let n = Array.length t in
3      let rec aux i j =
4          if i>j then false
5          else let k = (i+j)/2 in
6              if x = t.(k) then true
7              else
8              if x> t.(k) then aux (k+1) j
9              else aux i (k-1)
10     in aux 0 (n-1)
```

Matrices

Matrice

Définition 1 : Matrice

Une **matrice** est un tableau de tableaux dont tous les sous-tableaux sont de même taille.

Bien sûr, il ne s'agit que de matrices de dimension 2, et on pourrait aller plus loin vers une dimension 3, 4, ou plus.

Matrices en OCaml

En OCaml, les matrices sont des tableaux de tableaux, de type `'a array array` et on peut donc les manipuler comme tels :

```
let m = [|[|1;2|];
         [|3;4|]|]
```

```
m.(1).(0)
```

```
3
```

```
m.(0).(1) <- 5
```

```
m
```

```
[|[|1;5|];
[|3;4|]|]
```

Une fonction supplémentaire :

On peut utiliser Array.make_matrix pour construire une matrice.

```
1 Array.make_matrix 2 3 0
```

```
[|[|0;0;0|];
[|0;0;0|]|]
```

Exercice

Proposer une fonction de signature `float array array -> float` qui renvoie le produit des éléments d'une matrice.

Exercice

Proposer une fonction de signature `float array array -> float` qui renvoie le produit des éléments d'une matrice.

```
let produit_matrice mat =
    let n = Array.length mat in
    if n= 0 then 1. else
    let m = Array.length mat.(0) in
    let res = ref 1. in
    for i = 0 to n-1 do
        for j = 0 to m-1 do
            res:= !res * mat.(i).(j)
        done
    done ; !res
```

Chaînes de Caractère

Chaîne de caractère

Les chaînes de caractères sont des tableaux non-mutables de caractères. Les chaînes sont délimitées par des guillemets droits doubles, tandis que les caractères sont délimités par des guillemets droits simples.

```
1 "Bonjour !"
```

'B'	'o'	'n'	'j'	'o'	'u'	'r'	' '	'!'

Opération sur les chaînes de caractère.

On accède à un caractère d'une chaîne de caractère avec la syntaxe chaine .[indice] :

```
1 "Bonjour !".[3]
```

```
'j'
```

Attention, cet élément est un caractère et on une chaîne de caractère. On peut par ailleurs concaténer deux listes grâce à l'opérateur ^ :

```
1 "Bonjour" ^ " tout le monde !
```

```
"Bonjour tout le monde !
```

Attention à la distinction

```
let f str = str.[0]
```

```
string -> char
```

```
"abc".[0]="a"
```

Attention à la distinction

```
let f str = str.[0]
```

```
string -> char
```

```
"abc".[0]="a"
```

```
"abc".[0]='a'
```

Exercice

Proposer une fonction `est_dans` de signature `char -> string -> bool` qui vérifie si une chaîne de caractère passée en argument contient le caractère passé en argument.

Exercice

Proposer une fonction est_dans de signature char -> string -> bool qui vérifie si une chaîne de caractère passée en argument contient le caractère passé en argument.

```
let est_dans c s =
    let n = String.length s in
    let rec aux i =
        if i = n then false
        else if s.[i]=c then true
             else aux (i+1)
    in aux 0
```

Récapitulatif

	N-Uplet	Liste	Tableau	Chaîne de caractères
Type	'a * 'b *...	'a list	'a array	string
Taille	dans le type	par fonction récursive	Array.length	String.length
Accès	let a, b, c = nuplet	par fonction récursive	t.(i)	t.[i]
Modification	non	non	t.(i)<-x	non
Construction	(x,y,...)	[... ;... ;...] p : :q	Array.make n x [|... ;... ;...|]	"...."