TP4 : Tableaux, Matrices, Chaînes de caractère

Semaine du 2 octobre 2023

Exercice 1. Calcul sur des tableaux

1. Proposer une fonction de signature `int array -> int` qui renvoie la somme des éléments dans un tableau.
2. Proposer une fonction de signature `float array -> float` qui renvoie le produit des éléments dans un tableau.
3. Proposer une fonction de signature `int array -> int` qui renvoie l'élément le plus grand dans un tableau.
4. Proposer une fonction `recherche`de signature `'a array -> 'a -> int option` de sorte à ce que `recherche t x` renvoie le plus petit indice d'un élément de `t` égal à `a`.

   On veillera à avoir une complexité logarithmique dans le pire des cas.
5. Un tableau palindrome est un tableau qui a les mêmes valeurs dans un sens et dans l'autre, ainsi `[|1; 2; 4; 2; 1|]` et `[|2; 3; 3; 2|]`sont des palindromes mais `[|2; 3; 4; 2|]` n'en est pas un.

   Proposer une fonction de signature `'a array -> bool` qui détermine si un tableau est palindrome.

Exercice 2. Modification de tableaux

1. Proposer une fonction de signature `int array -> unit` qui mets à 0 tous les éléments d'un tableau en entrée.
2. Proposer une fonction de signature `'a array -> 'a -> unit` qui modifie les valeurs d'un tableau en entrée pour être toutes égales à un élément en entrée.
3. Proposer une fonction `copie` de signature `'a array -> 'a array -> unit` qui copie les éléments d'un premier tableau en entrée dans le second tableau en entrée. Dans le cas où les deux tableau ne sont pas de même taille, on copiera ce qu'il est possible de copier.
4. Proposer une fonction `copie_partielle` de signature `'a array -> 'a array -> int -> unit` de sorte à ce que `copie_partielle t1 t2 n` copie les n premières valeures de t_1 dans t_2.

Exercice 3. Construction de tableaux

1. Proposer une fonction de signature `int -> int array` qui, avec une entrée n, renvoie un tableau de taille n qui contient les entiers de 0 jusqu'à $n-1$ dans l'ordre croissant.
2. Proposer une fonction de signature `int -> int array` qui, avec une entrée n, renvoie un tableau de taille n qui contients les puissances de 2 de 1 jusqu'à 2^{n-1}.
3. Proposer une fonction de signature `int -> int array` qui calcule le tableau des diviseurs de son entrée n.

4. Proposer une fonction de signature `int -> bool array` qui renvoie le tableau de l'écriture binaire de l'entrée $n \geqslant 0$.
5. Proposer une fonction de signature `int -> bool array` qui renvoie le tableau de l'écriture binaire de l'entrée $n \in \mathbb{Z}$.
6. Proposer une fonction de signature `'a array -> 'a array` qui crée un nouveau tableau dont les éléments sont égaux au tableau en entrée.
7. Proposer une fonction `debut` de signature `'a array -> int -> 'a array` de sorte à ce que `debut t n` renvoie un tableau de taille n dont les éléments sont égaux aux n premiers éléments de t.
8. Proposer une fonction `partie` de signature `'a array -> int -> int -> 'a array` de sorte à ce que, pour $n < m$, `partie t n m` renvoie un tableau de taille $m-n+1$ dont les éléments sont égaux aux éléments de t d'indices n à m.

Exercice 4. Conversions

1. Proposer une fonction de signature `'a list -> 'a array` qui construit un tableau avec les mêmes éléments que la liste en entrée.
2. Proposer une fonction de signature `'a array -> 'a list` qui construit une liste avec les mêmes éléments que la liste en entrée.
3. Proposer une fonction de signature `string -> char list` qui construit une liste dont les éléments sont les caractères de la chaîne de caractère en entrée.
4. Proposer une fonction de signature `string -> char array` qui construit un tableau dont les éléments sont les caractères de la chaîne de caractère en entrée.
5. Proposer une fonction de signature `'a array list -> 'a list array` qui convertie une liste de tableaux en tableau de listes.

Exercice 5. Crible d'Ératosthène

Le crible d'Ératosthène est un algorithme qui permet de trouver tout les nombres premiers jusqu'à un nombre fixe N.

Le principe est le suivant, on représente tous les nombres dans un tableau jusqu'à ce nombre N. On commence par rayer le nombre 1, puis on procède comme suit jusqu'à traverser tout le tableau :

— On prend le premier nombre qui n'est pas rayé : ce nombre est premier, et on peut l'ajouter à la liste des nombres premiers ;
— On raye ensuite tout les multiples de ce nombre à part lui.

À l'aide du crible d'Ératosthène, proposer une fonction de type `int -> unit` qui, avec une entrée, affiche les nombres premiers inférieurs ou égaux à n.

On utilisera un tableau de booléens pour se souvenir de l'état actuel du crible.

Exercice 6. Matrices

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

3. Proposer une fonction de signature `int array array -> int array` qui renvoie un tableau des sommes des éléments des lignes d'une matrice donnée en argument.

```
somme_ligne [|[|1; 2|]; [|3; 4|]|] (* [|3; 7|] *)
```

4. Proposer une fonction `transposer`de signature `'a array array -> unit` qui transpose les éléments de la matrice en argument, c'est à dire qui, pour tout $i < j$ inverse la valeurs de `m.(i).(j)` et `m.(j).(i)`.
5. La fonction `Array.make_matrix` de signature `int -> int -> 'a -> 'a array array` permet, avec l'expression `Array.make_matrix n m x`, de construire une matrice de dimensions n, m initialisée à x.

   Proposer une implémentation de `make_matrix` à l'aide de `Array.make`.

   On prendra soin de construire un nouveau tableau pour chaque ligne de la matrice.
6. Proposer une fonction de signature `int -> int -> int array array` qui renvoie une matrice de dimensions données par les entrées et qui renvoie une matrice `m` de sorte à ce que `m.(i).(j)` ait pour valeur `i * j`.
7. Proposer une fonction `trace` de signature `float array array -> float` qui renvoie la trace d'une matrice passée en entrée.
8. Proposer une fonction `produit` de signature `float array array -> float array array -> float array array` qui renvoie la matrice produit des deux matrices en entrée.

Exercice 7. Création de matrice

1. Sans utiliser `Array.make_matrix`, mais en utilisant `Array.make`, proposer une implémentation de `Array.make_matrix`.
2. On propose le code suivant :

```
let make_matrix n m x =
    let resultat = Array.make n [||] in
    let ligne = Array.make m x in
    for i = 0 to n - 1 do
        resultat.(i) <- ligne
    done ; resultat
```

   Pourquoi ce code ne convient pas ?

   On pourra modifier le tableau résultant et observer un problème.

Exercice 8. Ordre supérieur

1. Proposer une fonction `init` de signature `int -> (int -> 'a) -> 'a array` de sorte à ce que `init n f` construise le tableau `[| f 0; f 1; f 2; ... ; f (n-1)|]`.
2. Proposer une fonction `init_matrice` de signature `int -> int -> (int -> int -> 'a) -> 'a array array` de sorte à ce que `init_matrice n m f` construise la matrice suivante :

```
[| [| f 0 0; f 0 1;... ; f 0 m|];
   [| f 1 0; f 1 1;... ; f 1 m|];
   ...
   [| f n 0; f n 1;... ; f 0 m|] |]
```

3. Proposer une fonction `map` de signature `'a array -> ('a -> 'b) -> 'b array` de sorte à ce que `map [|a0 ; a1; ...; an|] f` renvoie le tableau `[| f a0; f a1; ...; f an|]`.
4. Proposer une fonction `remplace` de signature `'a array -> ('a -> 'a) -> unit` de sorte à ce que `remplace [|a0; a1; ...; an|] f` modifie le tableau en entrée de sorte à ce qu'il soit égal à `[| f a0; f a1; ...; f an|]`.

Exercice 9. Tableaux de listes

1. Proposer une fonction chercher de signature 'a list array -> 'a -> bool qui vérifie si un élément est dans l'une des listes d'un tableau donné en argument.

```
chercher [|[5];[];[1;2]|] 2 (* true *)
```

```
chercher [|[5];[];[1;2]|] 8 (* false *)
```

2. Proposer une fonction chercher de signature 'a list array -> 'a -> int qui renvoie l'indice de la première liste dans le tableau passé en argument dont un élément est égal à l'élément passé en argument, et −1 à défaut.

```
chercher [|[5];[];[1;2]|] 2 (* 2 *)
```

```
chercher [|[5];[];[1;2]|] 8 (* -1 *)
```

3. Proposer une fonction taille_totale de signature 'a list array -> int qui renvoie la somme des tailles des listes dans un tableau donné en argument.

```
taille_totale [|[5];[];[1;2]|] (* 3 *)
```

```
taille_totale [|[5];[];[1;2]; [4;5;5]|] (* 6 *)
```

4. Proposer une fonction ajouter_a_tous de signature 'a list array -> 'a -> unit qui ajoute en tête de toutes les listes du tableau en argument l'élément en argument.

```
let t = [|[5];[];[1;2]|];;
let () = ajouter_a_tous t 1;;
t (*[|[1;5];[1];[1;2]|] *)
```

5. Proposer une fonction somme_tableau de signature int list array -> int array qui crée un tableau dont les éléments sont les sommes des listes du tableau passé en argument.

```
let somme_tableau [|[5];[];[1;2]|] (*[|5; 0; 3|]*)
```

Exercice 10. Tapis de Sierpiński

On cherche à construire une matrice qui corresponde au tapis de Sierpiński. Chaque case de la matrice est soit égale à 0 si la case est vide dans le tapis, soit égale à 1 si la case est remplie.

Ainsi, pour l'étape 1, la matrice qui représente le tapis est :

```
[|  [|1;1;1|];
    [|1;0;1|];
    [|1;1;1|]|]
```

Pour l'étape 2, la matrice est :

```
[|  [|1;1;1;  1;1;1;  1;1;1|];
    [|1;0;1;  1;0;1;  1;0;1|];
    [|1;1;1;  1;1;1;  1;1;1|];

    [|1;1;1;  0;0;0;  1;1;1|];
    [|1;0;1;  0;0;0;  1;0;1|];
    [|1;1;1;  0;0;0;  1;1;1|];

    [|1;1;1;  1;1;1;  1;1;1|];
    [|1;0;1;  1;1;1;  1;0;1|];
    [|1;1;1;  1;1;1;  1;1;1|]  |]
```

1. À l'étape n, quelles sont les dimensions de la matrice du tapis ?
2. Proposer une fonction `met_a_zero` de type `int array array -> int -> int -> int -> int -> unit` de sorte à ce que `met_a_zero t x1 y1 x2 y2` assigne `0` à tout les éléments dans la matrice `t` d'indices $x_1 \leqslant x \leqslant x_2$ et $y_1 \leqslant y \leqslant y_2$.
3. En déduire une fonction récursive `retire_centres` de type `int array array -> int -> int -> int -> int -> unit` de sorte à ce que `retire_centres t x1 y1 x2 y2` découpe la zone d'indices $x_1 \leqslant x \leqslant x_2$, $y_1 \leqslant y \leqslant y_2$ en 9 zones, met à zéro la partie centrale, et s'applique récursivement sur les parties extérieures.
4. En déduire une fonction `sierpinski` de signature `n -> int array array` qui renvoie le tapis de Sierpiński pour l'étape n.

Exercice 11. Jeu de la vie

Le jeu de la vie est un automate cellulaire en deux dimensions qui fait évoluer par génération une grille.
Les cellules sont organisées sur une grille de dimension fixe. Chaque cellule est soit vivante, soit morte.
Pour passer d'une étape à une autre, pour chaque cellule, on compte le nombre de ses 8 voisines directes qui sont vivante :

— Si une cellule a exactement 3 voisines vivantes, elle est vivante à l'étape suivante ;
— Si une celllule a exactement 2 voisines vivantes, elle conserve sont état précédent à l'étape suivante ;
— Si une cellule a un autre nombre de voisines vivantes (1 ou moins, 4 ou plus), elle est morte à l'étape suivante.

On représente une étape par une grille de type `bool array array` où un cellule est vivante si la valeur à ses coordonées vaut `true` et `false` sinon.

1. Proposer une fonction `acces` de signature `bool array array -> int -> int -> int -> bool` de sorte à ce que `acces m i j` renvoie l'élément d'indice i, j dans m si i, j correspond à une case du tableau, et `false` si i ou j sont en dehors des dimensions du tableau.

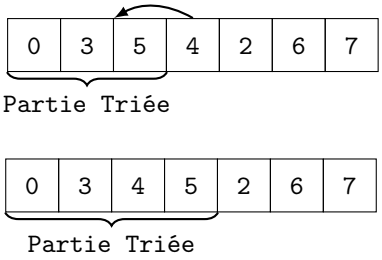
   Ainsi, grâce à cette fonction, on peut accéder à un élément qui est en dehors des dimensions de m : on considère que les cellules qui sont en dehors du domaine sont mortes.
2. En déduire une fonction `nombre_voisines` de signature `bool array array -> int -> int -> int` de sorte à ce que `nombre_voisines m i j` renvoie le nombre de voisines vivantes de la cellule de cordonnées i, j dans la matrice m.
3. En déduire une fonction `etape` de signature `bool array array -> unit` qui met à jour la matrice donnée en entrée pour une étape du jeu de la vie.

   On pourra utiliser une matrice intermédiaire de même dimensions.
4. En déduire une fonction `jeu_de_la_vie` de signature `bool array array -> int -> unit` qui met à jour la matrice donnée en argument pour un nombre d'étape donné en argument.

Exercice 12. Tri par insertion

Le tri par insertion est un tri simple qui trie le tableau au fur et à mesure en prenant les éléments dans l'ordre et en l'ajoutant à la bonne place dans la partie qui est déjà triée, quitte à redécaler tout les éléments déjà trié qui étaient plus grand que lui.

On commence donc avec le premier élément qui est bien placé, puis on prend le deuxième élément et on le met avant ou après le premier élément selon leur relation d'ordre.

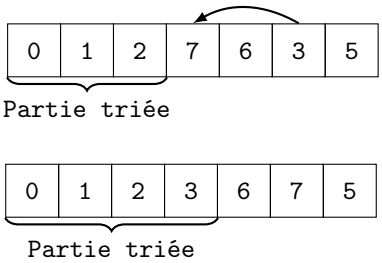


1. Quelles sont les étapes pour trier le tableau [|0; 5; 3; 4; 2; 6; 7|] ?
2. Proposer une fonction `echange` de signature `'a array -> int -> int -> unit` de sorte à ce que `echange t i j` échange les valeurs des cases `i` et `j` dans le tableau `t`.
3. Proposer une fonction `inserer` de signature `'a array -> int -> unit` de sorte à ce que `inserer t k` insère l'élément situé à la position `k` dans la partie triée du tableau qui fini à la position `k-1` dans le tableau `t`.
4. Proposer une fonction `tri_insertion` de signature `'a array -> unit` qui trie un tableau par insertion.
5. Combien d'appels à `echanger` doit-on faire dans le pire des cas ?

Exercice 13. Tri par sélection

Le tri par sélection est un tri simple qui, à chaque étape, prend l'élément le plus petit dans la partie non triée du tableau, et qui l'ajoute à la partie triée du tableau en l'échangeant avec l'élément le premier élément pas encore trié du tableau.

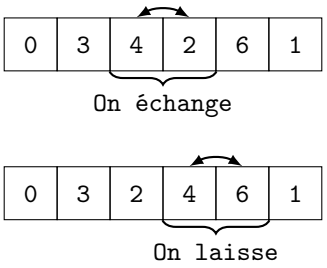
On commence donc à prendre le plus petit élément du tableau pour le mettre en première position, puis on prend le plus petit élément du tableau restant, c'est-à-dire le deuxième plus petit du tableau pour le mettre en deuxième position.



1. Quelles sont les étapes pour trier le tableau [|0; 7; 6; 1; 2; 3; 5|] ?
2. Proposer une fonction `echange` de signature `'a array -> int -> int -> unit` de sorte à ce que `echange t i j` échange les valeurs des cases `i` et `j` dans le tableau `t`.
3. Proposer une fonction `plus_petit_restant` de signature `'a array -> int -> int` de sorte à ce que `plus_petit_restant t k` donne l'indice du plus petit élément dans le tableau dont l'indice est supérieur à `k`.
4. Proposer une fonction `tri_selection` de signature `'a array -> unit` qui trie un tableau par sélection.
5. Combien d'appels à `echanger` doit-on faire dans le pire des cas ? Combien de fois doit on accéder à un élément avec `plus_petit_restant` ?

Exercice 14. Tri à bulle

L'opération de base du tri à bulle est la comparaison de deux éléments adjacents dans le tableau : on compare deux éléments adjacents, et on les inverse s'ils ne sont pas dans le bonne ordre.



Le tri à bulle consiste à répéter cette opération du début jusqu'à la fin du tableau.

1. Proposer une fonction `comparer` de signature `'a array -> int -> unit` de sorte à ce que `comparer t k` compare les éléments `k` et `k+1` et les échange si nécessaire.
2. Proposer une fonction `comparer_ligne` de signature `'a array -> int -> int -> unit` de sorte à ce que `comparer_ligne t a b` (avec $a < b$) applique successivement `comparer t a (a+1)`, puis `comparer t (a+1) (a+2)`, ..., jusqu'à `comparer t (b-1) b`.
3. Que peut on dire du dernier élément après l'application de `comparer_ligne t 0 (Array.length t - 1)` ?
4. Proposer un algorithme de tri en utilisant cette propriété.
5. Proposer une fonction `tri_bulle` de signature `'a array ->unit` qui trie le tableau en entrée avec le tri à bulle.
6. Combien d'appels à `comparer` doit-on faire ? Combien d'échanges ont lieu dans le pire et dans le meilleur des cas ?
7. Une variante de tri à bulle, le tri cocktail, alterne le sens des passages. Proposer une implémentantion du tri cocktail.
8. Pour ce tri, combien d'appels à `comparer` doit-on faire ? Combien d'échanges ont lieu dans le pire et dans le meilleur des cas ?

Exercice 15. Plus de mémoire

Dans cet exercice, on cherche à sauvegarder les résultats qu'on a déjà calculer pour accélérer leur utilisation par la suite. On utilise un tableau pour se souvenir des valeurs qu'on a déjà calculer, et les utiliser pour le calcul d'autres valeurs.

1. On cherche à calculer les termes de la suite de Fibonacci, et à retrouver facilement les termes qui ont déjà été calculés à l'aide d'un tableau.

   Dans cette question, on suppose qu'on accès à deux variables mutables :

```
let n = 100
```

```
let valeurs = Array.make n 0
```

```
let derniere_valeur = ref (-1)
```

   `valeurs` est un tableau rempli de 0 à partir de la case indiquée par `derniere_valeur`, mais qui contient dans tous les cases dont l'indice est inférieur ou égal la valeur de `fibo` pour cet indice.

   Ainsi, si on a calculé les 5 première valeurs de la suite de Fibonacci, notre tableau de valeurs est égal à `[|0; 1; 1; 2; 3; 0; 0; ...; 0|]`, et `derniere_valeur` est égal à 4.

   Proposer une fonction `fibo` de signature `int -> int` qui calcule le k-ième terme de la suite de Fibonacci en mettant à jour `valeurs` et `derniere_valeur` en utilisant le plus possible ces variables.

2. On cherche désormais à calculer la suite définie sur $\mathbb{N}*$ par récurrence suivante :

$$u_1 = 1$$
$$\forall n > 1, u_n = \sum_{k|n, k\neq n} u_k^2$$

Lors du calcul de u_n, on ne calcule pas nécessairement tous les u_k pour $k \leqslant n$, en on veut donc avoir une manière de savoir quelles valeurs on a déjà calculer. Que pourrait-on utiliser comme solution ?

Implémenter votre solution.

On pourra utiliser une autre variable mutable, ou bien utiliser une propriété sur les valeurs de u_n.

3. Proposer une fonction memoire de signature (int −> 'a) −> int −> (int −> 'a) qui prend en argument une fonction f et un entier n et qui sauvegarde les valeurs déjà calculées de f pour les n premiers entiers. La fonction renvoie une fonction g qui fait appel si possible au tableau avec la mémoire des valeurs de f, et qui sinon appelle f et modifie la mémoire des valeurs déjà calculées.

On prendra soin de construire la référence du tableau dans le bon contexte de sorte à ce qu'il periste entre les appels de g, mais pas entre les appels f.

```
let f x = Printf.printf "Calcul de %d\n" x ; x
let g = memoire f 10
```

```
g 0 (* affiche 0, et renvoie 0 *)
```

```
g 1 (* affiche 1, et renvoie 1 *)
```

```
g 0 (* renvoie seulement 0*)
```

```
let g2 = memoire f 10
```

```
g2 0 (* affiche 0, et renvoie 0 *)
```

Correction :

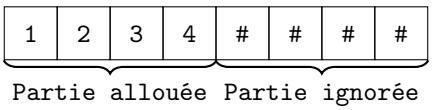
1.
```
let memoire f n =
    let mem = Array.make n false in
    let vals = Array.make n (f 0) in
    let g x = match mem.(x) with
        | true -> vals.(x)
        | false -> let res = f x in
            mem.(x)<- true;
            vals.(x)<- res;
            res
    in g
```

Exercice 16. Des Tableaux de taille variable

On considère un tableau à taille variable par un couple de type 'a array * int. Le premier élément du couple est le tableau dans lequel on peut allouer des valeurs, et le deuxième élément est l'entier qui correspond au nombre d'éléments qui ont déjà été alloués dans le tableau. On fera attention à faire la distinction entre cet entier, et la taille du tableau qui a été alloué.

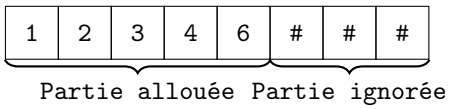
Ainsi, on a un tableau plus grand que le nombre de valeur qu'on veut stocker, et on se souvient de combien de valeurs sont importantes grâce au deuxième éléments du couple. Les éléments d'indices inférieur strictement au nombre de valeur stockée sont les éléments stockés, tandis que ceux dont l'indice est supérieur sont des valeurs sans importance auxquelles on ne veut pas accéder. La valeur exacte des éléments du tableau dont l'indice est supérieur à cette valeur n'est pas importante : on sait qu'on n'aura pas besoin de celles-ci.

De cette manière, le couple `([|1;2;3;4;0;0;0;0|], 4)` correspond au tableau suivant :



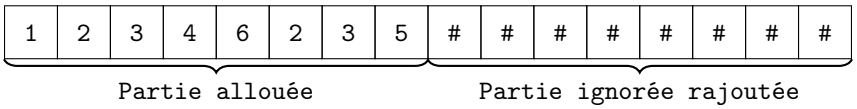
L'idée est de, lors de l'ajout d'une variable dans le tableau, l'ajouter si possible à la suite dans le tableau, et de sinon remplacer le tableau par un tableau deux fois plus grand, puis d'ajouter l'élément.

Ainsi si on ajoute un élément 6 dans le tableau précédent, on obtient le tableau suivant :

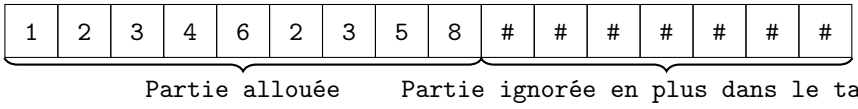


La valeur du couple serait donc `([|1;2;3;4;6;0;0;0|], 5)`.

En revanche, si on veut ajouter un élément dans le tableau `([|1;2;3;4;6;2;3;5|], 8)`, il faut procéder à un redimensionnement du tableau, et on obtient le tableau :



On peut ensuite rajouter l'élément, 8 par exemple et on obtient :



Cela nous donne le couple `([|1;2;3;4;6;2;3;5;8;0;0;0;0;0;0;0|], 9)`.

On remarquera qu'étant donné qu'un couple est un type non mutable, il faut renvoyer un couple lors des modifications du tableau à taille varia

1. Proposer une fonction `creer` de signature `'a -> 'a array * int` qui initialise un couple de tableau à taille variable à l'aide d'une valeur qui servira à remplir le tableau (on pourra prendre 0 pour initialiser un tableau d'entier, mais n'importe quel entier nous suffit).
2. Proposer une fonction `reaffecter` de signature `'a array -> 'a array -> unit` qui copie les éléments du premier tableau dans les premiers éléments du second.
3. Quelle est la complexité de `reaffecter` en fonction de n la taille du premier tableau ?
4. Proposer une fonction `ajouter_a_la_fin` de signature `'a array * int -> 'a -> 'a array * int` de sorte à ce que `ajouter (t, n) x` renvoie le couple du tableau et de l'entier modifié.
5. Proposer une fonction `supprimer` de signature `'a array * int -> 'a array * int` qui retire le dernier élément d'un tableau à taille variable.
6. Lors de l'ajout de n valeurs dans le tableau, combien de réaffectations ont lieux ? Quelle est la complexité total de l'ajout de ces n valeurs ?

   On pourra supposer que n est une puissance de 2.
7. Proposer une fonction `acceder` de signature `'a array * int -> int -> 'a` de sorte à ce que `acceder (t, n) k` renvoie l'élément d'indice k dans le tableau à taille variable en entrée.

   On pourra lever une erreur dans le cas où l'indice accède à une partie du tableau à laquelle on n'est pas sensé accéder.

8. Proposer une fonction `modifier` de signature `'a array * int -> int -> 'a -> 'a array * int` de sorte à ce que `modifier (t, n) k x` renvoie le tableau à taille variable dont l'élément d'indice k a été modifié pour la valeur x.

Exercice 17. Relation d'ordre

On peut comparer la plupart des types de base avec les comparateurs usuels (`>`, `<`, `>=`, `<>` et `<=`)

1. Tester sur machine la comparaison de chaînes de caractère, et proposer une définition de la relation d'ordre qui correspond à la relation d'ordre d'OCaml pour les chaînes de caractère.
2. Tester sur machine la comparaison de tableaux de mêmes types, et proposer une définition de la relation d'ordre qui correspond à la relation d'ordre d'OCaml pour les
3. Tester sur machine la comparaison de n-uplets de mêmes types, et proposer une définition de la relation d'ordre qui correspond à la relation d'ordre d'OCaml pour les n-uplets.

Exercice 18. FizzBuzz

Le FizzBuzz a été un grand classique des entretients d'embauche. Sa consigne est simple : afficher les entiers de 1 à 100, mais au lieu d'afficher les multiples de 3, afficher « Fizz », au lieu des multiples de 5 afficher « Buzz ».

L'idée de cet exercice en entretien n'est pas seulement de faire des vérifications algorithmiques sur le niveau de la candidate ou du candidat, mais de vérifier la capacité de la personne à proposer un code clair, commenté, et qui soit facile à modifier dans le futur. Par ailleurs, parfois, le cas d'un nombre mutiple de 15 n'est pas spécifié (comme ici), et le candidat ou la candidat doit proposer une solution.

Proposer une implémentation du FizzBuzz.

Exercice 19. Nombres Romains

Écrire une fonction `romain` de signature `int -> string` qui associe à un entier n la chaîne de charactère qui représente le nombre en chiffre romain.

On pourra utilesr l'opérateur `^` qui permet de concaténer deux chaînes de caractère.