

Exercice 1 : Nombre d'occurrences dans une liste

- Q. 1 Donner une fonction permettant le calcul du nombre d'occurrences d'un élément dans une liste.

Exercice 2 : Nombre d'occurrences du minimum

- Q. 1 Donner une fonction permettant le calcul, à partir d'une liste l de la paire (x, nb) où x est le plus petit élément de l et nb est son nombre d'occurrence dans l .

Exercice 3 : Première occurrence satisfaisant un prédicat

- Q. 1 Donner une fonction prenant en argument une fonction f et une liste l et calculant x le premier élément de l tel que $f(x)$ est vrai.

Exercice 4 : Position des occurrences

- Q. 1 Définir une fonction prenant en paramètres une liste l et un élément x et donnant la liste des indices auxquels x se trouve dans la liste l .

Exercice 5 : Sous-séquence

- Q. 1 Définir une fonction calculant, à partir d'une liste l , la plus longue sous liste d'éléments consécutifs identiques dans l . Votre fonction calculera un triplet : indice de départ de la sous-séquence en question, longueur de la sous-séquence en question, valeur des éléments de la sous-séquence en question.

Exercice 6 : Suppression du n -ième élément d'une liste

- Q. 1 Définir une fonction prenant en argument un entier n et une liste l et calculant la liste l privée de son n -ième élément.

Exercice 7 : Regroupement en une liste d'association

- Q. 1 Définir une fonction prenant en argument une liste l et calculant une liste de paires représentant dont le premier élément de la paire est un élément de l et le second son nombre d'occurrences dans l .

Exemples :

| regroupe [1; 2; 3; 2; 4; 3] = [(1, 1); (2, 2); (3, 2); (4, 1)]

Exercice 8 : Deuxième plus grand

Q. 1 Donner une fonction donnant le deuxième plus grand élément d'une liste.

Exercice 9 : Un sur deux

Q. 1 Donner une fonction prenant en argument une liste l et calculant la liste obtenue en prenant un élément sur deux de l .

Exercice 10 : Représentation d'une fonction au moyen d'une paire de liste

On représente une fonction par deux listes de même tailles : la liste de ses antécédents, la liste de ses images.

Exemples :

| ([1; 3; 2], [5; 5; 7]) représente la fonction ($1 \mapsto 5, 2 \mapsto 7, 3 \mapsto 5$).

Q. 1 Définir une fonction repr prenant en argument une liste l et une fonction f et calculant la représentation par paire de listes de la fonction f sur l'espace de définition formé par l . On proposera un type pour cette fonction avant de la définir.

Q. 2 Définir une fonction derepr prenant en argument une liste l et une fonction f et calculant la fonction représentée par la liste. La fonction ainsi calculée devra lever une exception si la liste ne précise pas son comportement.

Exercice 11 : Fonctionnelle 1

Q. 1 Définir le type et une implémentation de la fonction list_filter prenant en paramètres une fonction f à valeurs booléennes et une liste l et retournant la liste des éléments x de l tels que $f(x)$ vaut vrai (dans l'ordre dans lequel ils apparaissent dans l).

Exercice 12 : Fonctionnelle 2

Q. 1 Définir le type et une implémentation de la fonction list_map prenant en paramètres une fonction f à valeurs booléennes et une liste l et retournant la liste des éléments $f(x)$ pour $x \in l$ (dans l'ordre dans lequel ils apparaissent dans l).

Exercice 13 : Anagramme

Donner une fonction permettant de tester si deux listes sont l'anagramme l'une de l'autre.

Exercice 14 : Énumération des listes de booléens

- Q. 1 Donner une fonction `generate : int -> bool list list` prenant en argument un entier n et générant la liste de toutes les listes de booléens de taille n .
- Q. 2 Donner une fonction `enumerate : int -> (int -> bool list)` prenant en argument un entier n et calculant une fonction f réalisant une bijection de $\llbracket 0, 2^n - 1 \rrbracket$ dans l'ensemble des listes de booléens de longueur n .

Exercice 15 : Liste 1

- Q. 1 Implanter la fonction `partition : ('a -> bool) -> 'a list -> 'a list * 'a list` prenant en argument un prédicat et une liste et calculant la paire de la liste des éléments satisfaisant le prédicat et ceux ne le satisfaisant pas.

On définit le type `type 'a option = None | Some of 'a`

- Q. 2 Implanter une fonction `filter_map : ('a -> 'b option) -> 'a list -> 'b list` telle que `filter_map f l` applique f à chaque élément de l , ôte de la liste les éléments pour lesquels l'appel produit `None` et calcule la liste des éléments se trouvant dans le `Some` sinon.

Exercice 16 : Liste 2

- Q. 1 Implanter une fonction `combine : 'a list -> 'b list -> ('a * 'b) list` prenant en argument deux listes de même longueur et les combinant : `combine [a1; ...; an] [b1; ...; bn] = [(a1, b1); ...; (an, bn)]`.
- Q. 2 Implanter une fonction `fold_left2 : ('a -> 'b -> 'c -> 'a) -> 'a -> 'b list -> 'c list -> 'a` telle que `fold_left2 f init [a1; ...; an] [b1; ...; bn] = f (... (f (f init a1 b1) a2 b2) ...) an bn`

Exercice 17 : Liste 3

- Q. 1 Définir une fonction `compare_l : 'a list -> 'a list -> int` comparant les deux listes pour l'ordre lexicographique (l'ordre du dictionnaire). Ainsi `compare_l l1 l2 = -1` si $l1 < l2$, `compare_l l1 l2 = 1` si $l1 > l2$ et finalement `compare_l l1 l2 = 0` si $l1 = l2$.
- Q. 2 Comment rendre cette fonction paramétrique en l'opérateur de comparaison des éléments dans les listes ?
- Q. 3 Définir une fonction `map2 : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list` telle que `map2 f [a1; ...; an] [b1; ...; bn] = [f a1 b1; ...; f an bn]`.

Exercice 18 : Mélange

Dans cet exercice, on s'autorise à utiliser la fonction `List.rev : 'a list -> 'a list` calculant le miroir d'une liste.

- Q. 1 Définir une fonction `mélange : 'a list -> 'a list` prenant en paramètre une liste `[x0; x1; x2; ...; xn-2; xn-1]` et retournant la liste `[x0; xn-1; x1; xn-2; x2; ...]`

- Q. 2** Si la réponse à la première question est de complexité quadratique en la taille de la liste initiale, même question mais avec une complexité linéaire.

Exercice 19 : Sous-listes continues

- Q. 1** Écrire une fonction `ss_listes_continues : 'a list -> 'a list list` prenant en paramètre une liste `l` et calculant l'ensemble de ses sous-listes non vides d'éléments contigus dans la liste `l`.

```
1 | ss_listes_continues [1; 2; 3];;
2 | - : int list list = [[1]; [1; 2]; [1; 2; 3]; [2]; [2; 3]; [3]]
```

Exercice 20 : Produit cartésien

- Q. 1** Écrire une fonction `produit_cartesien : 'a list -> 'b list -> ('a * 'b) list` prenant en paramètres une liste `[x0; x1; x2; ...; xn-2; xn-1]` et une liste `[y0; y1; x2; ...; ym-2; ym-1]` et retournant la liste `[(x0, y0); (x0, y1); ...; (x0, ym-1); (x1, y0); (x1, y1); ...; (x1, ym-1); ...; (xn-1, y0); (xn-1, y1); ...; (xn-1, ym-1);]`.

Exercice 21 : Suppression des occurrences consécutives

- Q. 1** Écrire une fonction `compresse : 'a list -> 'a list` qui élimine les éléments consécutifs identiques d'une liste. Par exemple :

```
1 | # compresse [1; 1; 2; 3; 3; 3; 4; 4];;
2 | - : int list = [1; 2; 3; 4]
```

Exercice 22 : Liste d'éléments à tableau indicateur

Soit une constante entière $M \in \mathbb{N}$ fixée ($M = 5$ dans les exemples). On considère deux représentations en OCAML des sous-ensembles de $\llbracket 0, M - 1 \rrbracket$ (par exemple l'ensemble $\{0, 2, 3\}$).

- La première au moyen d'une liste finie des éléments de l'ensemble (`[0; 2; 3]` ou `[3; 2; 0]` pour l'exemple ci-dessus).
 - La seconde au moyen d'un tableau de taille M de booléen dont la case $i \in \llbracket 0, M - 1 \rrbracket$ indique si oui ou non l'élément est dans l'ensemble (`[true; false; true; true; false]` pour l'exemple ci-dessus).
- Q. 1** Donner des fonctions de conversion `a_to_b : int -> int list -> bool array` et `b_to_a : bool array -> int list` d'un type vers l'autre. La fonction `a_to_b` prend en paramètres non seulement la liste en question mais aussi l'entier M de l'énoncé.

Exercice 23 : Liste de tableaux

On considère une représentation des ensembles fini d'entiers au moyen d'une liste de tableaux de booléens. Une telle liste sera de la forme : $[t_{p-1}; t_{p-2}; \dots; t_1; t_0]$ où le tableau t_i est un tableau de 2^i booléen indiquant dans sa case $j \in \llbracket 0, 2^{i-1} \rrbracket$ si oui ou non l'élément $2^i - 1 + j$ est, ou non dans l'ensemble représenté. Ainsi l'exemple ex ci-dessous représente l'ensemble d'entiers : $\{0, 2, 4, 5, 6, 8, 9, 14\}$.

```
1 let ex =
2   (* 7      8      9      10      11      12      13      14 *)
3   [|false; true ; true ; false; false; false; false; true |];
4   (* 3      4      5      6 *)
5   [|false; true ; true ; true |];
6   (* 1      2 *)
7   [|false; true |];
8   (* 0 *)
9   [|true |];
10  ]
```

- Q. 1 Définir une fonction `mem (i: int) (s: bool array list): bool` permettant de tester si un élément i est dans l'ensemble s représenté.
- Q. 2 Définir une fonction `ajout (i: int) (s: bool array list): bool array list` permettant l'ajout d'un élément i à l'ensemble s . Par exemple l'appel `ajout 5 [|true|]` devra produire `[|false; false; true; false|]; [|false; false|]; [|true|]`.

Exercice 24 : Remplissage d'un tableau

- Q. 1 Définir une fonction `remplissage : int array -> int list -> unit` prenant en paramètres un tableau d'entiers t et une liste *non vide* l et remplaçant, dans le tableau t les occurrences du nombre -1 par des valeurs piochées dans la liste l . Les valeurs de la liste l devront être utilisées dans l'ordre, si on atteint la fin de la liste on recommence. Par exemple si t est le tableau `[|1; 2; -1; 0; -2; -1; -1|]` l'appel `(remplissage t [4; 5; 6; 8])` l'appel devra modifier le tableau t en la valeur `[|1; 2; 4; 0; -2; 5; 6|]`. `(remplissage t [4; 5])` devra modifier le tableau t en la valeur `[|1; 2; 4; 0; -2; 5; 4|]`.

Exercice 25 : Fusion de listes en tableau

- Q. 1 Définir une fonction `fusion : 'a list -> 'a list -> 'a array` prenant en paramètres deux listes *triées par ordre croissant* et retournant un tableau t , *trié* par ordre croissant et dont les éléments sont ceux se trouvant dans l'union des listes en paramètres.

Exercice 26 : Nombre d'occurrences

- Q. 1 Définir une fonction `histogramme (donnees: int list) (pas: int) (max: int): int array` prenant en paramètres une liste d'entiers (*donnees*) dont les éléments sont tous dans l'intervalle entier $\llbracket 0, \text{max} - 1 \rrbracket$, un pas entier (*pas*) et une valeur entière *max* et retournant un tableau t de taille $\lceil \frac{\text{max}}{\text{step}} \rceil$ contenant dans sa case d'indice i un entier indiquant le nombre d'entiers de la liste *donnees* se trouvant dans l'intervalle entier $\llbracket i \times \text{pas}, (i + 1) \times \text{pas} - 1 \rrbracket$.

Exercice 27 : Sous-séquences monotones

Q. 1 Définir une fonction `sseq_monotones : 'a array -> int list` prenant en paramètres un tableau d'éléments et retournant la liste des longueurs des sous-séquences monotones du tableau en paramètre. La longueur d'une séquence monotone est le nombre d'éléments se trouvant dans la séquence décrémente de 1. Par exemple le tableau `[1; 5; 8; 4; 6; 9; 8; 7; 6; 1; 9]` contient les séquences monotones : `[1; 5; 8]`, `[8; 4]`, `[4; 6; 9]`, `[9; 8; 7; 6; 1]` et finalement `[1; 9]` de longueurs respectives : 2, 1, 2, 4, 1. Ainsi le résultat attendu pour cette entrée est la liste `[2; 1; 2; 4; 1]`.

Exercice 28 : Transposition

On considère des tableaux de taille n de listes d'entiers de $\llbracket 0, n-1 \rrbracket$. On appelle *transposé* d'un tel tableau t , un tableau t' de taille n , contenant dans sa case $i \in \llbracket 0, n-1 \rrbracket$ une liste telle que $\forall j \in \llbracket 0, n-1 \rrbracket, j \in T[i] \Leftrightarrow i \in T[j]$.

Q. 1 Définir une fonction `transpose : int list array -> int list array` calculant la transposée du tableau qui les est passé en argument.

Exercice 29 : Matrice creuse

On considère dans cet exercice des matrices carrées d'entiers relatifs : $M \in \mathcal{M}_n(\mathbb{Z})$. Par exemple :

$$\begin{pmatrix} 0 & 3 & 5 \\ 0 & 0 & 7 \\ -1 & 0 & 0 \end{pmatrix}$$

On représente de telles matrices de deux manières :

- Ou bien par un tableau de tableau d'entiers relatifs, par exemple : `[ [0; 3; 5]; [0; 0; 7]; [-1; 0; 0] ]`.
- Ou bien par une liste de triplets : (i, j, v) indiquant que dans la case d'indice (i, j) de la matrice se trouve la valeur v . Si une case n'est pas mentionnée par une telle liste, c'est que la matrice représentée contient 0. Par exemple pour la matrice ci-dessus `[(0, 1, 3); (0, 2, 5); (1, 2, 7); (2, 0, -1)]`.

Q. 1 Donner des fonctions de conversion d'une représentation dans l'autre.

Exercice 30 : Cycle

Soit $n \in \mathbb{N}$. On dit d'une liste l d'entiers deux-à-deux distincts de $\llbracket 0, n-1 \rrbracket$ que c'est une liste *de cycle*. On dit qu'un tableau t' est le permuté d'un tableau t selon une liste de cycle $l = [l_0; l_1; \dots; l_{p-1}]$ lorsque t' est obtenu à partir de t en déplaçant en case $l_{(i+1) \bmod p}$ l'élément se trouvant en case l_i , et ce pour tout $i \in \llbracket 1, p \rrbracket$.

Q. 1 Écrire une fonction `permute : int array -> int list -> int array` prenant en paramètres un tableau t de taille n et une liste de cycle l et calculant le permuté du tableau t selon la liste l .

Q. 2 Écrire une fonction `est_permute : int array -> int array -> int list` prenant en paramètres deux tableaux t et t' de taille n et retournant une liste de cycle l telle que t' est le permuté de t selon la liste l . Si une telle liste n'existe pas, on lèvera un message d'erreur.

Exercice 31 : Permutations

Dans cet exercice on considère uniquement des tableaux T de taille $n \geq 0$, contenant une et une seule fois chaque entier de $\llbracket 0, n-1 \rrbracket$. Étant donné un tableau et un entier $i \in \llbracket 0, n-1 \rrbracket$, on note $o_T(i)$ (on l'appelle l'ordre de i dans le tableau T) le plus petit entier $p > 0$ tel que $\underbrace{T[T[T[\dots[T[i]]]]]}_{p \text{ fois}} = i$.

- Q. 1** Définir une fonction `o : int array -> int -> int` prenant en paramètres un tableau T et un entier i et calculant $o_T(i)$.
- Q. 2** Définir une fonction `max_o : int array -> int` prenant en paramètres un tableau T et retournant l'élément d'ordre maximal dans le tableau. On pourra supposer le tableau non vide.

Exercice 32 : Polynômes creux

On considère dans cet exercice des polynômes de $\mathbb{Z}[X]$. On représente de tels polynômes en OCAML au moyen d'une liste de couples d'entiers : la liste `[(a0, d0); (a1, d1); (a2, d2); ...; (ap, dp)]` avec pour tout $i \in \llbracket 0, p \rrbracket$, $a_i \in \mathbb{Z}^*$ et $d_i \in \mathbb{N}^*$. Une telle liste représente alors le polynôme :

$$X^{d_0}(a_0 + X^{d_1}(a_1 + X^{d_2}(a_2 + \dots X^{d_{p-1}}(a_{p-1} + a_p X^{d_p}))))$$

On définit donc le type suivant.

```
1 | type poly_1 = (int * int) list
```

On considère par ailleurs la représentation "classique" d'un polynôme $\sum_{i=0}^n b_i X^i$ au moyen d'un tableau de ses $n+1$ coefficients : `[b0; b1; ...; bn]`.

On définit donc le type suivant.

```
1 | type poly_2 = int array
```

- Q. 1** Donner des fonctions de conversion permettant la traduction d'une représentation vers l'autre. Discuter de la pertinence de l'une vis-à-vis de l'autre.

Exercice 33 : Énumérations

Dans cet exercice on s'intéresse au problème d'énumérer toutes les combinaisons possibles d'éléments choisis parmi des listes prédéfinies. Par exemple, étant donné les listes $l_0 = [0; 1]$, $l_1 = [1; 3; 5]$ et $l_2 = [9; 8]$, on souhaite pouvoir parcourir facilement l'ensemble des tableaux de taille 3 (le nombre de listes) dont la première case est choisie dans l_0 , la seconde dans l_1 et la troisième dans l_2 . Pour notre exemple cela donne les 12 tableaux de taille 3 suivants.

- | | | | |
|-------------|-------------|-------------|-------------|
| • [0; 1; 9] | • [1; 1; 8] | • [0; 1; 9] | • [1; 1; 8] |
| • [1; 3; 8] | • [0; 3; 9] | • [1; 3; 8] | • [0; 3; 9] |
| • [0; 5; 9] | • [1; 5; 8] | • [0; 5; 9] | • [1; 5; 8] |

Étant donné p listes $l_0, \dots, l_{p-1}$ d'entiers on souhaite donc énumérer tous les tableaux de taille p qu'il est possible de construire en plaçant dans ses cases d'indice i des éléments de l_i . On parlera d'énumérations des tuples des $(l_0, l_1, \dots, l_{p-1})$. On appelle p la dimension de l'énumération.

On définit pour cela le type `enumeration` suivant :

```
1 | type enumeration = (int list * int list) array
```

Ainsi une énumération est un tableau de couples de listes $[(k_0, l_0); (k_1, l_1); \dots; (k_{p-1}, l_{p-1})]$. La taille du tableau est la dimension de l'énumération.

Si l'on souhaite énumérer les tuples des $(l_0, l_1, \dots, l_{p-1})$ on crée un tableau de taille p dont la case d'indice i contient initialement le couple (l_i, l_i) . Durant l'énumération on ne modifie pas la seconde composante des couples (k_i, l_i) se trouvant dans les cases du tableau, ainsi celle-ci est invariante et vaut la liste l_i initiale.

L'état de l'énumération est obtenu en lisant les premiers entiers de chaque liste k_i du tableau d'énumération. Par exemple l'état de l'énumération $[([0; 1], [0; 1]); ([3; 5], [1; 3; 5]); ([9; 8], [9; 8])]$ est $[0; 3; 9]$. On modifie une énumération pour qu'elle représente l'état suivant en :

- trouvant le plus petit entier i_0 tel que k_{i_0} n'est pas une liste de taille 1,
- elle est alors de la forme $x_{i_0} :: r_{i_0}$,
- on modifie alors :
 - k_{i_0} en r_{i_0} ,
 - toutes les listes k_j en l_j pour $j < i_0$.

Q. 1 Définir en OCAML les fonctions :

- `init : int list array -> enumeration` permettant la création d'une énumération;
- `lit : enumeration -> int array` permettant la lecture de l'état d'une énumération;
- `suivant : enumeration -> bool` permettant de passage d'un état de l'énumération au suivant.

La fonction `suivant` retournera un booléen indiquant si les itérations sont terminées.

```
1 | # let e = init [| [0; 1] ; [1; 3; 5] ; [9; 8] |];;
2 | val e : enumeration = [| ([0; 1], [0; 1]); ([1; 3; 5], [1; 3; 5]); ([9; 8], [9; 8]) |]
3 | # lit e;;
4 | - : int array = [|0; 1; 9|]
5 | # suivant e;;
6 | - : bool = true
7 | # e;;
8 | - : enumeration = [| ([1], [0; 1]); ([1; 3; 5], [1; 3; 5]); ([9; 8], [9; 8]) |]
9 | # lit e;;
10 | - : int array = [|1; 1; 9|]
11 | # suivant e; suivant e; suivant e;;
12 | - : bool = true
13 | # e;;
14 | - : enumeration = [| ([0; 1], [0; 1]); ([5], [1; 3; 5]); ([9; 8], [9; 8]) |]
15 | # lit e;;
16 | - : int array = [|0; 5; 9|]
17 | # suivant e; suivant e; suivant e; suivant e; suivant e;;
18 | - : bool = true
19 | # lit e;;
20 | - : int array = [|1; 3; 8|]
21 | # suivant e; suivant e;;
22 | - : bool = true
23 | # lit e;;
24 | - : int array = [|1; 5; 8|]
25 | # suivant e;;
26 | - : bool = false
```

Exercice 34 : Représentation d'arbres au moyen de tableaux

Soit $n \in \mathbb{N}$. Dans cet exercice, on s'intéresse à des arbres ayant n sommets, étiquetés par les entiers de l'intervalle $\llbracket 0, n - 1 \rrbracket$ de sorte que chaque entier de $\llbracket 0, n - 1 \rrbracket$ apparaît une et une seule fois. On représente un tel arbre en OCAML au moyen d'un tableau de taille n contenant des entiers de $\llbracket 0, n - 1 \rrbracket$. Pour tout $i \in \llbracket 0, n - 1 \rrbracket$:

- si $t.(i) = i$ alors i est la racine de l'arbre représenté,
- si $t.(i) \neq i$ alors le sommet étiqueté par $t.(i)$ est le père du sommet i .

```
1 | type arbre = int array
```

- Q. 1** Dessiner l'arbre représenté par le tableau `[|1; 1; 1; 2; 3|]`.
- Q. 2** Donner une fonction `racine : arbre -> int` prenant en paramètre un arbre représenté par tableau et retournant sa racine.
- Q. 3** Donner une fonction `fabrique_branche : arbre -> int -> int list` prenant en paramètres un arbre et un sommet s et retournant la branche reliant s à la racine de l'arbre. Cette branche sera présentée sous la forme de la liste de ses sommets.
- Q. 4** Donner une fonction `hauteur : arbre -> int` retournant la hauteur de l'arbre passé en paramètre.
- Q. 5** Donner une fonction `plus_longue_branche : arbre -> int list` prenant en paramètre un arbre et retournant une branche de longueur maximale de l'arbre.