Entraînement à la programmation OCaml

Note : on s'efforcera de ne pas convertir les tableaux en listes et les listes en tableau, tout particulièrement lorsqu'il s'agit d'écrire deux fonctions similaires, l'une sur les listes et l'autre sur les tableaux.

Il est en revanche permis, pour répondre à une question, d'utiliser une fonction d'une autre question, mais si cela vous semble bienvenu, n'hésitez pas à spontanément décomposer le problème de la même façon la prochaine fois que vous le rencontrerez !

---

1. `somme_chiffres n` (`int -> int`)

Prend en argument un entier n positif ou nul, et renvoie la somme de ses chiffres dans son écriture décimale. On évitera de passer par des chaînes de caractères.

Complexité $O(\log n)$.

---

2. `nombre_bits_1 n` (`int -> int`)

Prend en argument un entier n positif ou nul, et renvoie le nombre de ses bits à 1 dans son écriture binaire. On évitera de passer par des chaînes de caractères.

Complexité $O(\log n)$.

---

3. `retourne n` (`int -> int`)

Prend en argument un entier n positif ou nul, et renvoie le nombre correspondant au retournement de ses chiffres dans son écriture décimale (par exemple, 123 donnera 321, 240 donnera 42). On évitera de passer par des chaînes de caractères.

Complexité $O(\log n)$.

---

4. `decompose b n` (`int -> int -> int list`)

Prend en argument une base b (entier $\geqslant 2$) et un entier n positif ou nul, et renvoie la liste $[a_0, a_1, \ldots, a_n]$ telle que les $a_k \in [\![0 \,..\, b-1]\!]$, $n = \sum_k a_k b^k$ et $a_n \neq 0$ si $n > 0$ (on renverra `[]` si $n = 0$).

Complexité $O(\log n)$.

5. `recompose b lst` (`int -> int list -> int`)

Effectue l'opération inverse de la fonction précédente. On n'utilisera pas de flottants, et on respectera la complexité (ce qui signifie ne pas calculer à chaque itération les b^k).

Complexité $O(|lst|)$.

---

6. `carre f` (`('a -> 'a) -> 'a -> 'a`)

Prend en argument une fonction f et renvoie la fonction $f \circ f$.

Complexité $O(1)$.

---

7. `compose f g` (`('b -> 'c) -> ('a -> 'b) -> 'a -> 'c`)

Prend en argument deux fonction f et g et renvoie leur composée $f \circ g$.

Complexité $O(1)$.

---

8. `puissance n f` (`('a -> 'a) -> int -> 'a -> 'a`)

Prend en argument une fonction f et renvoie la fonction $f^n = f \circ f \ldots f$. On pourra chercher une solution de complexité $O(\log n)$ (plus difficile).

Complexité $O(n)$ (ou $O(\log n)$).

---

9. `applique x` (`'a -> ('a -> 'b) -> 'b`)

Prend en argument un objet x et renvoie une fonction prenant en argument une fonction f et retournant $f(x)$.

Complexité $O(1)$.

---

10. `depasse f g` (`(int -> 'a) -> (int -> 'a) -> int`)

Prend en argument deux fonctions f et g et renvoie le plus petit entier positif k tel que $f(k) > g(k)$.

Complexité en $O(k)$.

11. `avant_dernier lst` (`'a list -> 'a`)

Renvoie l'avant-dernier élément d'une liste (et une erreur si la liste contient moins de deux éléments).

Complexité $O(|\texttt{lst}|)$.

---

12. `premiers n lst` (`int -> 'a list -> 'a list`)

Renvoie la liste des `n` premiers éléments de `lst` (on renverra l'intégralité de la liste si `n` est supérieur à la longueur de `lst`, et une liste vide si `n` est négatif).

Complexité $O(n)$.

---

13. `saute n lst` (`int -> 'a list -> 'a list`)

Renvoie la liste `lst` privée de ses `n` premiers éléments (on renverra une liste vide si `n` est supérieur à la longueur de `lst`, et la liste originale si `n` est négatif).

Complexité $O(n)$.

---

14. `retire i lst` (`int -> 'a list -> 'a list`)

Renvoie une liste correspondant à `lst` privée de l'élément à la position `i` (si `i` est strictement négatif ou si la liste contient moins de `i` éléments, on renverra la liste originale).

Complexité $O(i)$.

---

15. `index_pairs lst` (`'a list -> 'a list`)

Renvoie la liste des éléments de `lst` situés à des positions d'indices pairs.

Complexité $O(|\texttt{lst}|)$.

---

16. `separe lst` (`'a list -> 'a list * 'a list`)

Renvoie la liste des éléments de `lst` situés à des positions d'indices pairs et celle des éléments situés à des positions d'indice pair. On s'efforcera de ne parcourir la liste qu'une seule fois.

Complexité $O(|\texttt{lst}|)$.

17. `entremele lst1 lst2` (`'a list -> 'a list -> 'a list`)

Effectue l'opération inverse de la fonction précédente. Si une des deux listes est plus longue que l'autre, on rassemblera les éléments surnuméraires en fin de liste.

Complexité $O(|\texttt{lst1}| + |\texttt{lst2}|)$.

---

18. `derniers n lst` (`int -> 'a list -> 'a list`)

Renvoie la liste des `n` derniers éléments de `lst` (on renverra l'intégralité de la liste si `n` est supérieur à la longueur de `lst`, et une liste vide si `n` est négatif). On s'efforcera de trouver une solution qui ne nécessite pas de retourner la liste.

Complexité $O(|\texttt{lst}|)$.

---

19. `retourne_l lst` (`'a list -> 'a list`)

Renvoie une liste contenant les éléments de `lst` mais dans l'ordre inverse, sans utiliser `List.rev`.

Complexité $O(|\texttt{lst}|)$.

---

20. `retourne_t tab` (`'a array -> unit`)

Renverse l'ordre des éléments à l'intérieur d'un tableau.

Complexité $O(|\texttt{tab}|)$.

---

21. `compte_l x lst` (`'a -> 'a list -> int`)

Renvoie le nombre d'éléments dans la liste `lst` qui sont égaux à `x`.

Complexité $O(|\texttt{lst}|)$.

---

22. `compte_t x tab` (`'a -> 'a array -> int`)

Renvoie le nombre d'éléments dans le tableau `tab` qui sont égaux à `x`.

Complexité $O(|\texttt{tab}|)$.

23. `symetrique tab` `('a array -> bool)`

Détermine si les valeurs d'un tableau sont symétriques (le premier élément est égal au dernier, le second à l'avant-dernier, etc.).

Complexité $O(|\texttt{tab}|)$.

---

24. `duplique lst` `('a list -> 'a list)`

Renvoie une liste où chaque élément est doublé (par exemple, une liste `[1; 2; 3; 2; 2; 1]` doit donner `[1; 1; 2; 2; 3; 3; 2; 2; 2; 2; 1; 1]`).

Complexité $O(|\texttt{lst}|)$.

---

25. `repete lst1 lst2` `('a list -> int list -> 'a list)`

Renvoie une liste chaque élément x_i de `lst1` est répété y_i fois, y_i étant l'élément correspondant de `lst2` (par exemple, avec les listes `['a'; 'b'; 'a'; 'c']` et `[2; 1; 3; 1]`, on doit obtenir `['a'; 'a'; 'b'; 'a'; 'a'; 'a'; 'c']`). On suppose les listes de même longueur, on pourra lever une erreur si ce n'est pas le cas.

Complexité $O(|\texttt{lst1}|)$.

---

26. `decompose lst` `('a list -> 'a list * int list)`

Effectue la transformation inverse de la fonction précédente.

Complexité $O(|\texttt{lst}|)$.

---

27. `longueur_plateau_t tab` `('a array -> int)`

Prend en argument un tableau et renvoie la longueur de la plus longue séquence de valeurs consécutives égales deux à deux.

Complexité $O(|\texttt{tab}|)$.

---

28. `longueur_plateau_l lst` `('a list -> int)`

Prend en argument une liste et renvoie la longueur de la plus longue séquence de valeurs consécutives égales deux à deux.

Complexité $O(|\texttt{lst}|)$.

---

29. `retire_repetitions lst` `('a list -> 'a list)`

Renvoie une liste où les répétitions ont été supprimées (par exemple, une liste `[1; 2; 2; 1; 1; 1; 3; 2; 2; 1; 3]` doit donner `[1; 2; 1; 3; 2; 1; 3]`).

Complexité $O(|\texttt{lst}|)$.

---

30. `cherche f a b` `((int -> bool) -> int -> int -> int option)`

Prend en argument une fonction `f` et deux entiers `a` et `b` et renvoie `Some x` où x est un entier dans $[\![a\,..\,b]\!]$ tel que $f(x)$ est vrai s'il existe un tel x, et `None` sinon. On renverra `None` si $a > b$.

Complexité $O(|\max(1, b-a)|)$.

---

31. `trouve f a b` `((int -> bool) -> int -> int -> int list)`

Prend en argument une fonction `f` et deux entiers `a` et `b` et renvoie la liste des x entiers dans $[\![a\,..\,b]\!]$, triés par ordre croissant, tels que $f(x)$ est vrai.

Complexité $O(|\max(1, b-a)|)$.

---

32. `anti_filtre f lst` `(('a -> bool) -> 'a list -> 'a list)`

Renvoie une liste contenant les éléments de `lst` qui ne vérifient pas le prédicat `f` (les éléments `x` pour lesquels $f(x)$ est `false`). Les éléments doivent conserver leur ordre original.

Complexité $O(|\texttt{lst}|)$.

---

33. `denombre_l f lst` `(('a -> 'a -> bool) -> 'a list -> int)`

Dénombre le nombre de couples d'éléments distincts x_i et x_j de la liste tels que $f(x_i, x_j)$ est vrai.

Complexité $O(|\texttt{lst}|^2)$.

---

34. `denombre_t f tab` `(('a -> 'a -> bool) -> 'a array -> int)`

Dénombre le nombre de couples d'éléments distincts x_i et x_j du tableau tels que $f(x_i, x_j)$ est vrai.

Complexité $O(|\texttt{tab}|^2)$.

35. `doublon_l lst1 lst2` (`'a list -> 'a list -> 'a option`)

Renvoie `Some x` où `x` est un élément présent dans les deux listes, et `None` s'il n'en existe aucun.

Complexité $O(|\texttt{lst1}| \times |\texttt{lst2}|)$.

---

36. `doublon_t lst1 lst2` (`'a list -> 'a list -> 'a option`)

Même question que précédemment, mais on suppose `lst1` et `lst2` toutes deux triées par ordre croissant.

Complexité $O(|\texttt{lst1}| + |\texttt{lst2}|)$.

---

37. `egaux lst` (`'a list -> bool`)

Renvoie un booléen indiquant si les éléments de `lst` sont égaux deux à deux. On renverra `true` pour une liste vide ou à un seul élément.

Complexité $O(|\texttt{lst}|)$.

---

38. `images_egales f lst` (`('a -> 'b) -> 'a list -> bool`)

Renvoie un booléen indiquant si les $f(x_i)$, où les x_i sont les éléments de `lst`, sont égaux deux à deux. On renverra `true` pour une liste vide ou a un seul élément.

Complexité $O(|\texttt{lst}|)$.

---

39. `tri_l lst` (`'a list -> 'a list`)

Renvoie une liste contenant les éléments de `lst` triés par ordre croissant. On ne demande pas une méthode de tri particulière.

Complexité $O\big(|\texttt{lst}|^2\big)$, et si possible $O\big(|\texttt{lst}| \log |\texttt{lst}|\big)$.

---

40. `tri_t tab` (`'a array -> unit`)

Trie les éléments de `tab` par ordre croissant (en place). On ne demande pas une méthode de tri particulière.

Complexité $O\big(|\texttt{tab}|^2\big)$, et si possible $O\big(|\texttt{tab}| \log |\texttt{tab}|\big)$.

41. `eval_poly coeffs x` (`int list -> int -> int`)

Calcule $P(x)$ où P est un polynome dont les coefficients sont donnés dans la liste `coeff` par ordre croissant de degrés ($a_2x^2 + a_1x + a_0$ est représenté par `[a0; a1; a2]`). On n'utilisera pas de flottants, et on s'efforcera de respecter la complexité indiquée.

Complexité $O(|\texttt{coeffs}|)$.

---

42. `produit lst n` (`'a list -> int -> 'a list list`)

Prend un ensemble `lst` d'éléments et un entier positif `n` et renvoie la liste de toutes les listes de `n` éléments de `lst` avec remise (`['a'; 'b'; 'c']` et `2` donne `[['a'; 'a']; ['a'; 'b']; ['a'; 'c']; ['b'; 'a']; ['b'; 'b']; ['b'; 'c']; ['c'; 'a']; ['c'; 'b']; ['c'; 'c']]`). L'ordre des listes n'est pas imposée.

Complexité $O(|\texttt{lst}|^n)$.