

Exercice 1.

1. a. Ecrire une fonction `possible_ligne` prenant en argument une grille `g` de taille 9×9 , une position `i, j` d'entiers de $\llbracket 1, 9 \rrbracket$ et un chiffre `k` renvoyant `true` si et seulement si le chiffre `k` n'est pas présent dans la ligne `i` dans la grille de sudoku représentée par `g`.

b. Ecrire une fonction `possible_colonne` analogue à la fonction précédente pour les colonnes.

c. Ecrire une fonction `possible_bloc` vérifiant si un chiffre `k` est déjà présent dans le sous-bloc à neuf chiffres de l'indices `i, j` d'une grille de Sudoku `g`.

2. Ecrire une fonction de retour sur trace `completer` prenant en argument une grille de Sudoku partiellement remplie, renvoyant `true` si la grille possède une solution au problème du Sudoku et modifiant la grille en une solution, `false` sinon.

Indication : On pourra essayer sa solution sur la grille

```
let g=[|
[15; 3; 0; 0; 7; 0; 0; 0; 0];[16; 0; 0; 1; 9; 5; 0; 0; 0];[10; 9; 8; 0; 0; 0; 0; 6; 0];
[18; 0; 0; 0; 6; 0; 0; 0; 3];[14; 0; 0; 8; 0; 3; 0; 0; 1];[17; 0; 0; 0; 2; 0; 0; 0; 6];
[10; 6; 0; 0; 0; 0; 2; 8; 0];[10; 0; 0; 4; 1; 9; 0; 0; 5];[10; 0; 0; 0; 8; 0; 0; 7; 9]
|];;
```

Exercice 2.

On considère un ensemble E de nombres entiers et S un nombre entier. On cherche à savoir s'il existe un sous-ensemble $X \subset E$ tel que $\sum_{x \in X} x = S$.

1. Ecrire une fonction de type retour sur trace prenant en argument une liste de nombres `e` et un entier `s` renvoyant `true` si et seulement si il existe une sous-liste d'éléments de `e` dont la somme vaut `s`.

2. Modifier la fonction précédente afin de renvoyer une sous-liste sommant à l'entier `s` lorsqu'elle existe, renvoyant une erreur sinon.

3. Que dire de la complexité de ces fonctions ? 4. Adapter la réponse aux questions 1 et 2 lorsque l'ensemble E est représenté par un tableau.

Exercice 3.

Un **alphabet** est un ensemble fini Σ de symboles. Un mot est une suite finie $l_1 \cdots l_n$ d'éléments de l'alphabet. Σ^* est l'ensemble de tous les mots. Un langage sur Σ est une partie $L \subset \Sigma^*$.

Une **segmentation** d'une chaîne de caractères consiste en l'ajout d'espaces dans cette chaîne afin que chaque sous-chaîne séparée par deux espaces soit un mot du langage considéré.

1. Ecrire une fonction de type "retour sur trace" prenant en argument une chaîne de caractère `texte` renvoyant `true` si et seulement si cette chaîne admet une segmentation.

On suppose disposer d'une fonction `est_mot` prenant en argument une chaîne de caractères renvoyer vrai si et seulement si la chaîne est un mot du langage étudié.

2. Adapter la fonction précédente afin de renvoyant, si possible, une segmentation du texte.

Exercice 4.

1. Ecrire une fonction résolvant, si possible, le problème des n cavaliers, analogue au problème des n reines.

2. Etudier la complexité de votre algorithme.

Exercice 5.

On dispose de n dominos représentés par des couples (m, n) de chiffres. On dispose donc d'une liste $(m_i, n_i)_{1 \leq i \leq n}$.

Le but est de permuter ces dominos afin que pour tout $1 \leq i \leq n - 1$, $n_i = m_{i+1}$.

En revanche, les dominos ne peuvent pas être retournés (m_i et n_i ne peuvent être échangés).

1. Ecrire une fonction de type retour sur trace renvoyant, si possible, une solution du problème.

2. Evaluer la complexité dans le pire des cas.

Exercice 6. Un carré magique normal est une grille de taille n sur n dont les cases contiennent les entiers de 1 à n^2 et telles que la somme des nombres des cases d'une ligne, d'une colonne ou d'une des deux diagonales vaut $\frac{n(n^2+1)}{2}$.

Ecrire une fonction qui prend en entrée une matrice carrée de taille n partiellement remplie (les cases non remplies étant des 0) qui complète celle-ci en un carré magique normal.

Exercice 7. 1. Ecrire une fonction `somme : int array -> int -> int` qui renvoie la somme des éléments d'indices 0 à i d'un tableau d'entiers (c'est une somme à $i + 1$ éléments).

Un tableau t de $n > 0$ éléments qui sont $\llbracket 0, n - 1 \rrbracket$ est dit autoréférent si, pour tout $0 \leq i < n$, $t.(i)$ est le nombre d'occurrences de i dans t , à savoir, pour tout $i \in \llbracket 0, n - 1 \rrbracket$, $t.(i) = \text{Card}\{k \in \llbracket 0, n - 1 \rrbracket / t.(k) = i\}$.

Par exemple, `[1 ; 2 ; 1 ; 0]` est autoréférent.

2. Justifier qu'il n'existe aucun tableau autoréférent pour $1 \leq n \leq 3$. Trouver un autre tableau autoréférent pour $n = 4$.

3. Ecrire une fonction `est_auto : int array -> bool` qui vérifie si un tableau est autoréférent.

On attend une complexité de $O(n)$.

4. A l'aide d'une méthode de retour sur trace, écrire une fonction renvoyant un tableau autoréférent de taille n .

5. Ecrire une fonction renvoyant la liste de tous les tableaux autoréférents de taille n . L'essayer pour $n = 15$.

Pour accélérer cette méthode, il faut "élaguer" l'arbre de recherche en repérant le plus rapidement possible les branches sans solution.

6. Que peut-on dire de la somme des éléments d'un tableau autoréférent ? En déduire une stratégie d'élagage. On pourra utiliser la fonction "somme" pour interrompre l'exploration lorsque celle-ci dépasse déjà la valeur maximale possible.

7. Après avoir affecté la case i d'un tableau t , que peut-on dire s'il y a déjà strictement plus d'occurrences d'une valeur $0 \leq k \leq i$ que la valeur de t en k ? En déduire une stratégie d'élagage supplémentaire. Essayer cette nouvelle méthode pour $n = 15$.

8. Après avoir affecté la case i de t , combien de cases reste-t-il à remplir ? Combien de ces cases sont complétées par une valeur non nulle ? A quelle condition est-on alors certain que la somme dépassera la valeur maximale possible à la fin ? En déduire une stratégie d'élagage supplémentaire et la mettre en oeuvre. Essayer votre nouvelle méthode pour $n = 30$.

9. Montrer qu'il existe un tableau autoréférent pour tout $n \geq 7$. On pourra conjecturer la forme de ce tableau en testant empiriquement différentes valeurs de $n \geq 7$.