# TP : Algo récursif et interface Graphique

Vincent Berry, Christophe Fiorio, Tanmoy Modal

L'objectif ici est de résoudre une grille de sudoku, par exemple celles-ci

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 5 | 0 | 4 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 0 | 3 | 8 | 0 | 0 |
| 0 | 0 | 3 | 0 | 1 | 0 | 0 | 0 | 7 |
| 0 | 0 | 0 | 0 | 4 | 0 | 0 | 3 | 0 |
| 7 | 0 | 1 | 6 | 0 | 8 | 4 | 0 | 2 |
| 0 | 2 | 0 | 0 | 3 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 9 | 0 | 5 | 0 | 0 |
| 0 | 0 | 5 | 8 | 0 | 0 | 0 | 6 | 9 |
| 0 | 0 | 0 | 0 | 0 | 4 | 0 | 2 | 0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 3 | 0 | 0 | 4 | 1 | 0 | 0 | 8 | 7 |
| 0 | 0 | 9 | 0 | 0 | 5 | 0 | 6 | 0 |
| 4 | 0 | 0 | 7 | 9 | 0 | 5 | 0 | 3 |
| 0 | 7 | 3 | 2 | 4 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 7 | 8 | 2 | 4 | 0 |
| 6 | 0 | 2 | 0 | 8 | 3 | 0 | 0 | 5 |
| 0 | 5 | 0 | 1 | 0 | 0 | 3 | 0 | 0 |
| 1 | 3 | 0 | 0 | 2 | 4 | 0 | 9 | 6 |

Résoudre une grille de sudoku consiste à remplacer les 0 par un chiffre de 1 à 9 de telle sorte qu'il soit unique dans sa *ligne*, sa *colonne* et sa *région*. On appelle *région* les 9 sous-grilles de $3 \times 3$ encadrées.

Le principe de résolution est simple, et peut se résumer ainsi :

1. prendre la prochaine case à 0
2. si au moins une valeur est possible, on essaie une valeur et on continue de résoudre avec la prochaine case à 0
3. si on remplit toutes les cases à 0, le problème est résolu
4. sinon si on arrive à un blocage, on reprend au point 2. avec la prochaine valeur possible en ayant au préalable effacé toutes les valeurs inscrites depuis
5. si il n'y a plus de valeurs possibles, alors c'est un échec et il n'y a pas de solution

C'est exactement ainsi que l'on procède « à la main » en essayant d'être malin dans le choix des cases afin d'aller au plus vite à la solution.

La difficulté pour écrire un algorithme reprenant cette méthode est le point 4. Comment revenir en arrière simplement, sans avoir à mémoriser toutes les opérations pour pouvoir les annuler ?

## 1 Récursivité

Par la suite on supposera que chaque ligne est une liste de 9 entiers, et donc que la grille est une liste de 9 lignes.

### * Exercice 1 valeurs disponibles

Écrire une fonction qui

— prend en paramètre une liste de 9 entiers de 0 à 9, le 0 indiquant une position vide et pouvant être en plusieurs exemplaires, les autres chiffres apparaissant dans n'importe quel ordre et au plus une fois,

— retourne un tableau de booléens indiquant quelles sont les valeurs disponibles, c'est à dire celles absentes de la liste (on choisira un tableau de 10 booléens), l'indice indiquant la valeur dont on veut savoir si elle est disponible ou pas

### * Exercice 2 valeurs disponibles sur une ligne

Écrire une fonction qui rend la liste des valeurs disponibles pour une ligne sur le modèle du tableau de booléens de la question 1 ; la grille et la ligne sont données en paramètre et la ligne sera indiquée par son numéro de 0 à 8.

### * Exercice 3 valeurs disponibles sur une colonne

Écrire une fonction qui rend la liste des valeurs disponibles pour une colonne sur le modèle du tableau de booléens de la question 1 ; la grille et la colonne sont données en paramètre et la colonne sera indiquée par son numéro de 0 à 8.

### * Exercice 4 valeurs disponibles dans une région

Écrire une fonction qui rend la liste des valeurs disponibles dans une région sur le modèle du tableau de booléens de la question 1 ; la grille et la région sont passsées en paramètre et la région sera indiquée par le numéro de ligne et de colonne de la première case de la région.

### ** Exercice 5 valeurs possibles à une position

En vous servant des fonctions précédentes, écrire une fonction qui, pour une position donnée de la grille (ligne, colonne), donne la liste des valeurs possibles à cette position, toujours sur le modèle de la question 1.

### * Exercice 6 positions à remplir

Écrire une fonction `listePositionsNonRemplies` qui renvoie la liste des positions de la grille pour lesquels il faut trouver une valeur ; chaque position est indiquée par un couple $(l, c)$ ou $l$ et $c$ sont respectivement les numéros de lignes et de colonnes de la position.

### *** Exercice 7 remplir grille

Soit la fonction suivante :
`grilleFinie` : $[[\mathsf{int}]] \times [\mathsf{int}] \times \mathsf{int} \longrightarrow [[\mathsf{int}]] \times \mathsf{bool}$
qui à partir d'une grille, d'une liste des positions à remplir, de l'index à partir duquel les positions à remplir doivent être examinées [1], renvoie la grille complétée et `true` si elle a été entièrement remplie.
L'algorithme de résolution pourra s'écrire comme la fonction :

```
func sudoku(grille: [[int]]) → ([[int]], bool)
    var lpos: [(int, int)] = listePositionsNonRemplies(grille)
    if len(listePositionsNonRemplies) > 0 then
        return grilleFinie(grille,lpos,0)
    else
        return (grille,false)
    endif
endfunc
```

**Algorithme 1 :** Résolution de sudoku

Écrivez l'algorithme récursif de la fonction `grilleFinie`.

## 2 Interface Graphique

L'objectif désormais est de fournir une interface graphique pour saisir une grille à remplir et afficher le résultat du calcul

### ** Exercice 8 saisir grille

Définissez une interface graphique permettant de saisir une grille de sudoku à remplir et permettant de lancer la résolution.

Pour simplifier l'interface, on considèrera que l'on saisi chaque ligne comme un csv, par exemple $3, 0, 0, 4, 1, 0, 0, 8, 7$ pour la première ligne de la grille donnée en exemple.

Vous pouvez bien sûr proposer une interface plus riche et pratique.

1. Les positions précédant l'index sont considérées comme n'appartenant pas à la liste : si on avait une vraie liste, on les aurait supprimées de la liste, ce qui n'est pas possible avec un tableau.

### ** Exercice 9 afficher grille

Définissez une fenêtre permettant de visualiser une grille à remplir. Celle-ci devra maintenant s'afficher une fois la saisie de la question 8 validée.

### ** Exercice 10 grille résolue

Définissez une fenêtre permettant de visualiser une grille remplie. Celle-ci devra s'afficher dès que la résolution est terminée.

### * Exercice 11 programme complet

Faite en sorte que

— si la grille proposée n'est pas valide [2], un message d'avertissement s'affiche et qu'il ne soit pas possible de lancer la résolution.

— si la grille n'a pas de solution, un message d'avertissement s'affiche et qu'on puisse corriger la grille à résoudre proposée.

## 3 Pour ceux qui ont le temps....

### ** Exercice 12 sauvegarde

Rajoutez les possibilités de sauvegarder, au format csv, la grille saisie et la grille résolue.

### ** Exercice 13 saisie améliorée

Faites une interface plus conviviale où on peut rentrer les valeurs directement dans la grille.

Profitez-en pour ajouter la fonctionnalité suivante : à chaque saisie d'une valeur, pour entrer la grille à résoudre, une vérification est faite sur la validité de l'ajout de cette valeur et un message d'avertissement s'affiche si la nouvelle entrée entraîne la non validité de la grille.

2. Par valide on entend que les valeurs saisies ne respectent pas la contrainte d'unicité sur les lignes, colonnes et dans les régions.