Informatique - Master 1 IL
Option IA
TP 1 - Résolution de grilles de Sudoku

Vincent Drevelle, sujet original de Tassadit Bouadi

---

31 janvier 2025

Lors de ce TP, vous allez vous familiariser avec la recherche de solutions avec retour arrière (*backtracking*). Vous allez appliquer ces méthodes à la résolution de grilles de Sudoku.

Pour rappel, le but du jeu du Sudoku est de compléter une grille à partir de chiffres déjà placés initialement. Nous nous intéresserons ici aux grilles de taille 9×9 ($n = 9$). Les contraintes sur le remplissage de la grille sont alors les suivantes :

- chaque ligne doit contenir une seule fois chaque chiffre de 1 à 9 ;
- chaque colonne doit contenir une seule fois chaque chiffre de 1 à 9 ;
- chaque région (de taille 3×3) doit contenir une seule fois chaque chiffre de 1 à 9.

Le jeu est représenté par une grille contenant une matrice de cases.

1 Recherche de solution par retour arrière

Vous commencerez par implémenter les règles du Sudoku dans *Grille.java*. La classe proposée permet de s'adapter aux différentes grilles de taille $n \times n$, où $n = k^2, k \in \{1, 2, \ldots\}$.

Complétez ensuite dans le fichier *Sudoku.java* la fonction *resoudreSudoku(Stack<Grille> pileGrilles)* qui effectue la résolution de la grille placée en tête de pile, en utilisant un algorithme de recherche par retour arrière (l'algorithme est donné en annexe). La pile permet de gérer le parcours de l'espace de recherche. Si une solution est trouvée, la fonction retournera *true* et la grille complétée sera au sommet de la pile.

Le choix d'une case vide à traiter (fonction *getCasePossible()* de *Grille*) s'effectue ici en choisissant la première case non remplie, en considérant les cases selon leurs indices croissants de ligne et de colonne, dans la grille.

Plusieurs grilles vous sont proposées afin de tester votre programme, dont un Sudoku très difficile, et une grille sans solution. Instrumentez la fonction *resoudreSudoku* afin de compter le nombre de nœuds explorés.

2 Ajout d'une heuristique sur le choix des variables

Modifiez l'algorithme précédent pour pouvoir choisir, comme case vide à traiter, la case la plus contrainte, c'est-à-dire celle pour laquelle le nombre de choix de valeurs possibles est le plus petit.

Quel est l'impact sur le nombre de nœuds explorés pour la résolution ?

3 Filtrage des valeurs de variables (*forward-checking)*

Modifiez à nouveau l'algorithme afin de propager les contraintes sur les valeurs possibles des cases appartenant à la même ligne ou à la même colonne ou à la même région que la case à laquelle une valeur vient d'être affectée.

Vous pouvez modifier la classe *Case* afin d'y stocker le domaine des valeurs possibles (par exemple dans un *BitSet*).

Algorithme de retour arrière appliqué au Sudoku

```
booléen backtrack(Pile<Grille> pile)
  soit G la grille en sommet de pile (mais pas ôtée de la pile)

  si toutes les cases de G sont remplies  alors
     retourner vrai;
  sinon
     soit C une case vide de G;
     pour toutes les valeurs V de la case C faire
        si affecter la valeur V à la case C conserve toutes les
             contraintes du Sudoku sur G alors
           créer une copie G' de G;
           donner la valeur V à la case C sur la grille G';
           empiler G';
           si backtrack(pile) = vrai alors
               retourner vrai;  // solution au sommet de la pile
           fsi
           dépiler G';  // retour-arrière
        fsi
     fait
     retourner faux;  // pas de solution trouvée en partant de G
  fsi
```