TP-algorithmique

Solver de Sudoku

Jérémy Grisé, décembre 2020

22/12/2020

I - Modifications apportées à Grille.java

I.a. - Economie d'accès mémoire

Les fonctions testColonne(int j), testLigne(int i) et testCarre(int i0, int j0) utilisent à l'origine un tableau int[ ] test de 10 valeurs, or nous n'en avons besoin que de 9 et donc ainsi on peut faire une économie de 3 accès à la mémoire.

Pour réaliser cette optimisation, il suffit de soustraire 1 dans les conditions else if de chacune de ces fonctions et dans l'opération effectuée à la suite de ces conditions.

I.b. - Ajout de la fonction boolean isSolvable()

Si un Sudoku contient moins de 17 valeurs remplis, alors il n'y a pas de solution unique pour le résoudre, mais plusieurs, ainsi nous le considèreront comme faux.

Voici un lien pour plus d'informations concernant cette propriété :

https://www.science-et-vie.com/questions-reponses/dans-un-sudoku-y-a-t-il-un-minimum-de-cases-preremplies-5100

II - Conception du solver dans Solver.java

Ce solver met en place une approche récursive pour résoudre le Sudoku.

Tout d'abord, on utilise la fonction isSolvable() pour vérifier qu'il y'a au moins 17 valeurs.

A présent on va parcourir la grille de Sudoku de ligne i, en ligne i et de colonne j, en colonne j.

Si la case courante est vide (la valeur est 0), on va tester les valeurs de 1 à 9 pouvant être misent dans la case.

Dès qu'une valeur peut être placée on la place.

On fait de nouveau appel au solver, c'est-à-dire en usant d'une approche récursive.

On va ainsi à partir de la première hypothèse concernant la première case vide, faire d'autre hypothèses pour chacune des autres cases vides.

Si durant la récursion il est impossible de placer une valeur dans une case vide alors on va remettre à 0 toutes les cases posant problèmes,

Puis on continuera à tester les valeurs suivantes de la case à partir de laquelle la valeur placée précédemment n'entrainait pas de solution viable au Sudoku.

Une fois toutes les opérations réalisées et donc les cases vides remplis en respectant les règles du Sudoku. La fonction Solver() renverra finalement le booléen true.

III - Implémentation dans Main.java

Pour tester le solver, une grille complétée de référence et sa version non complète ont été placées.

On y utilise la fonction nanoTime(), car la fonction currentTimeMillis() n'est pas suffisamment précise.

Puis on utilise le solver.

Enfin, une double boucle for permet de parcourir la grille résolu et la grille de référence, afin de vérifier que le solver a bien retrouver la bonne grille et qu'il n'y pas d'erreur dans le code.