Projet Algo. S.D. (2014/2015) - Grilles de Sudoku

A rendre le mercredi 4 février 2015

Formation Ingénieurs ITII - CNAM Alsace

Ce projet est à réaliser en binôme. Le langage de programmation à utiliser est le C. Il faudra rendre un rapport imprimé avec une description du travail réalisé, des résultats obtenus ainsi que de l'adéquation des résultats obtenus avec les résultats attendus. Ce rapport contiendra 5 pages maximum et le code complet sera fourni en annexe. On s'attachera à soigner la présentation du code afin qu'il soit le plus lisible possible. Il ne suffit pas d'écrire le code d'une fonction, il faut expliquer (i.e. commenter) les choix que ce soit au niveau de la structure de données ou de l'algorithme choisi(e).

On s'intéresse à la modélisation de grilles de Sudoku. Une grille de Sudoku comporte 81 chiffres, qui sont rangés dans une grille 9×9. La grille est elle-même subdivisée en 9 sous-régions de taille 3×3. Une grille est considérée comme correcte si elle respecte les trois conditions suivantes :

— Chaque ligne ne contient qu'une seule occurrence des chiffres de 1 à 9.
— Chaque colonne ne contient qu'une seule occurrence des chiffres de 1 à 9.
— Chaque sous-région de taille 3×3 ne contient qu'une seule occurence des chiffres de 1 à 9.

1 Génération de grilles de Sudoku

Dans un premier temps, on s'intéresse à la génération de grilles bien formées (à trous, mais pour lesquelles il existe au moins une solution). Pour cela, il s'agit de générer des grilles complètes vérifiant toutes les conditions et ensuite de retirer certaines des valeurs. Une grille complète est présentée dans la figure 1. Pour que le jeu soit intéressant, il faut au moins retirer la moitié des valeurs des cases.

Les grilles ainsi créées pourront être enregistrées dans des fichiers en codant les 81 chiffres ligne par ligne, les cases vides pourront être représentées par le caractère "_".

Une première approche pourra consister à générer aléatoirement des grilles et ensuite à vérifier qu'elle vérifie bien les conditions pour être un sudoku valide. Dans un second temps,

6	3	2	9	1	4	7	8	5
1	9	8	7	5	6	4	2	3
7	4	5	8	3	2	6	1	9
2	8	7	4	6	9	3	5	1
3	5	4	1	2	8	9	6	7
9	6	1	3	7	5	8	4	2
8	1	9	5	4	7	2	3	6
4	2	3	6	9	1	5	7	8
5	7	6	2	8	3	1	9	4

FIGURE 1 – Un exemple de grille correcte

on pourra expérimenter avec des algorithmes un peu plus efficaces pour construire la grille. On évitera par exemple de mettre plusieurs fois la même valeur sur la même ligne.

2 Résolution de grilles de Sudoku

Dans cette partie, on cherche à développer un algorithme pour résoudre une grille de sudoku partiellement complétée. Il s'agit de trouver les valeurs à positionner dans les cases non remplies.

Une méthode simple consistera à choisir une case vide et à déterminer les valeurs possibles pour cette case en éliminant toutes les valeurs interdites par les 3 conditions énoncées au début du sujet. Si l'on trouve une seule valeur, on peut la mettre immédiatement dans la case. Sinon il faut mémoriser les différentes valeurs possibles et poursuivre l'instantiation des différentes cases vides, jusqu'à ce que toutes les cases soient remplies ou bien jusqu'à ce qu'une case ne puisse pas recevoir de valeurs. L'idéal sera donc de gérer une liste de valeurs possibles (comprises entre 1 et 9) pour chaque case de la grille. Une fois la configuration initiale connue, il suffira alors d'éliminer pour chaque case vide, les valeurs interdites jusqu'à n'avoir plus qu'une seule valeur possible par case.

Si le temps le permet, on pourra étudier d'autres configurations et rechercher des algorithmes plus efficaces pour résoudre ces grilles de sudoku. Nous vous fournissons 2 exemples de grille (voir Fig. 2 et 3) que votre programme devrait être capable de résoudre. Vous pouvez bien sûr choisir des grilles supplémentaires à résoudre. Pour chacune des 2 grilles fournies, il faudra évaluer le temps de calcul nécessaire pour produire une solution.

	2			3		9		7
	1							
4		7				2		8
		5	2				9	
			1	8		7		
	4				3			
				6			7	1
	7							
9		3		2		6		5

Figure 2 – Exemple 1

							1	
4								
	2							
				5		4		7
		8				3		
				1	9			
3			4			2		
	5		1					
			8		6			

Figure 3 – Exemple 2