Programmation structurée **Sudoku et Backtracking**

Le but de ce travail est de réaliser un programme capable de compléter des grilles de Sudoku.

Sudoku (d'après Wikipedia : http://fr.wikipedia.org/wiki/Sudoku) :

La grille de jeu est un carré de neuf cases de côté, subdivisé en autant de carrés identiques, appelés régions (voir figure). La règle du jeu est simple : chaque ligne, colonne et région ne doit contenir qu'une seule fois tous les chiffres de un à neuf. Formulé autrement, chacun de ces ensembles doit contenir tous les chiffres de un à neuf.

Exemple :

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

La plupart du temps, le jeu est proposé sous la forme d'une grille de 9×9, et composé de sous-grilles de 3×3, appelées « régions ». Quelques cellules contiennent des chiffres, dits « dévoilés ». Le but est de remplir les cellules vides, un chiffre dans chacune, de façon à ce que chaque rangée, chaque colonne et chaque région soient composées d'un seul chiffre allant de 1 à 9. En conséquence, chaque chiffre dans la solution apparaît une seule fois selon les trois « directions », d'où le nom « chiffre unique ». Lorsque qu'un chiffre peut s'inscrire dans une cellule, on dit qu'il est candidat.

Programmation par Backtracking

1) Programmer une première version permettant de résoudre le problème de la manière la plus « bête » possible, c'est à dire en ne vérifiant les contraintes d'acceptabilité qu'une fois la grille complètement remplie.
2) Programmer une deuxième version plus rapide en vérifiant les contraintes dès que possible.
3) Une bonne grille de Sudoku doit pouvoir se résoudre uniquement par déduction et donc sans utiliser de méthode de type backtracking.

   Programmer une troisième version accélérant davantage la résolution en exploitant au maximum les informations déjà déterminées :

(a) quand une valeur est placée, elle interdit le placement de la même valeur dans la même ligne-colonne-région, en combinant toutes ces contraintes sur une case il se peut qu'il n'y ait plus qu'une valeur possible pour une case donnée,

(b) de même, on peut s'apercevoir qu'une seule place est possible dans une ligne-colonne-région pour une valeur donnée.

Dans une phase initiale, on répétera ce processus chaque fois qu'une valeur est placée. Lorsqu'on ne pourra plus rien déduire, on lancera dans une seconde phase la procédure de recherche par backtracking dans sa version 2) qui sera ainsi appliquée aussi peu que possible.

Application sur l'exemple règle (a) :

la case centrale ne peut contenir

- 1, 2, 6, 7, 8, 9 présents sur la verticale,
- 1, 3, 4, 8 présents sur l'horizontale,
- 2, 3, 6, 8 présents dans la région.

La seule possibilité est donc 5.

Application sur l'exemple règle (b) :

la première colonne a 4 cases indéterminées ([2,0], [6,0], [7,0], [8,0]), par la règle (a) on déduit que :

- [2,0] ne peut contenir que 1, 2,
- [6,0] ne peut contenir que 1, 3, 9,
- [7,0] ne peut contenir que 2, 3,
- [8,0] ne peut contenir que 1, 2, 3,

La seule possibilité pour ranger le 9 est [6,0].

Programmation et structures de données :

Pour les questions 1) et 2) vous utiliserez une matrice 9x9 d'entiers pour représenter le jeu. Pour la question 3) vous ajouterez un tableau à 3 dimensions de booléens pour représenter les diverses possibilités de valeurs pour chaque case. La case i, j, k contiendra true si la valeur k + 1 est possible pour la case i, j et false sinon.

Résultats :

Comparez les temps et les nombres de grilles / valeurs envisagées par chacune des techniques.

A rendre :

une archive **de type tar (ET PAS AUTRE CHOSE)** nommée prenom-nom-sud.tar contenant un répertoire nommé prenom-nom-sud contenant les .h, les .cpp, un makefile et un fichier texte commentant les résultats de comparaison en temps et en nombre d'essai) des trois versions.