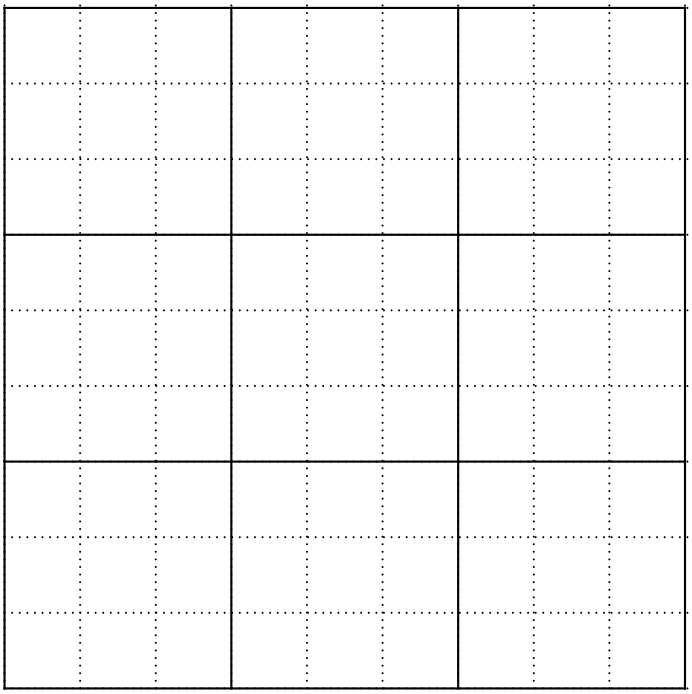
Sudoku

cours ENSTA MA206

Il s'agit d'implémenter un algorithme de résolution du Sudoku.
On rappelle les règles du jeu du Sodoku : considérons un carré de 9 cases de côté. Ce carré est lui-même subdivisé en 9 sous carrés de 3 cases de côté. Chaque ligne, chaque colonne et chaque sous carré doivent contenir une et une seule fois tous les entiers $\{1, \cdots, 9\}$.



1 Un algorithme de base

La'algorithme naïf de résolution du Sudoku consiste à tester à parcourir la grille (par exemple de gauche à droite et de haut en bas) en testant dans l'ordre les valeurs de 1 à 9 pour la cellule courante. Si aucune valeur n'est acceptable, alors on recule d'une case dans la grille et on propose la valeur suivante puis on reprend la marche normale.

Algorithme (Backtracking)**.** *On commence par la première case .*
Tant que l'on n'a pas atteint la dernière case, Faire

1. *On teste les valeurs de* 1 *à* 9 *dans l'ordre, dès qu'une valeur est acceptable on s'arrête.*
2. *Si aucune valeur n'est acceptable, reculer d'une case.*
3. *Si après recul, on se retrovue sur la première case et qu'elle contient la valeur* 9*, alors l'algorithme s'arrête et la grille n'admet pas de solution.*

4. *Si on a atteint la dernière case, l'algorithme se termine et on a trouvé une solution.*

On définira une classe **Sudoku** comme un tableau d'entiers contenant :

- les indices de ligne et colonne, son numéro de carré ;
- un fonction booléenne testant si une valeur est acceptable ou non, on séparera ce point en un test pour la ligne, la colonne et le carré ;
- Compte tenu du nombre important de fois que les tests devront être réalisés, on réfléchira à une méthode pour éviter de recalculer à chaque fois les indices des cases appartenant au même sous-carré ;
- une fonction de lecture de la grille depuis un fichier texte, dont le format est à définir ;
- une fonction d'impression d'une grille à l'écran.

2 Un algorithme plus évolué

L'algorithme proposé ici consiste à déterminer pour chaque case la liste des possibilités sans violer les règles du jeu. Une fois ces listes crées, on recherchera la plus courte. Si elle est de taille 1, on met la dite case à l'unique valeur de la liste et on répercute cette modification aux listes des valeurs possibles pour les cases alentours (la ligne, la colonne et le sous-carré). Si la plus petite liste est de taille strictement supérieure à 1, pour chaque entrée de la liste on duplique le contexte et on poursuit l'algorithme avec ce choix. L'algorithme s'arrête lorsque le carré est entièrement rempli ou lorsqu'une case ne peut être remplie sans violer les règles.

Algorithme. *Algorithme par liste chaînée*

1. *construire pour chaque cellule la liste des possiblités*
2. *on cherche les liste de plus courte longueur (on veillera à minimiser le nombre de parcours de la grille)*
3. *si la plus courte liste est de taille* 1*, on choisit cette valeur. On met à jour les listes des possibilités pour les cases alentours.*
4. *si la plus courte liste est de taille* > 1*, on crée une deuxième grille (copie de la première). On choisit la première valeur de la liste des possiblités pour cette nouvelle grille créée.*
5. *Revenir à l'étape 2 pour chacune des grilles*
6. *L'algorithme s'arrête dès qu'une grille est remplie.*

On définira une classe **Sudoku** comme un tableau de cellules (une classe **cell**). Une cellule devra contenir

- une liste des valeurs acceptables : on utilisera l'implémentation des listes fournit par la STL. De nombreuses opérations telles que l'ajout ou la suppression sont déjà définies ;
- la longueur de la liste pour éviter de la recalculer à chaque fois
- son numéro de ligne, de colonne, de sous-carré ;

- des fonctions de test d'une valeur sur une ligne, une colonne, un carré.

La classe **Sudoku** devra offrir la possibilité de lire une grille de Sudoku à partir d'un fichier texte.

3 Travail demandé

Dans un premier temps, on implémentera un algorithme naïf, de type *backtracking*. Une seconde étape consistera à coder l'algorithme plus évolué décrit dans la Partie 2. Finalement, vous comparerez les 2 algorithmes implémentés. Compte tenu du temps imparti à la réalisation des ce projet, il n'est pas demandé pas de créer d'interface graphique pour le jeu. Néanmoins il faudra mettre au point une syntaxe permettant de définir un jeu par un fichier texte. Le programme pourra alors prendre en entrée un tel fichier et affichera la solution dans la console.
On comparera cet algorithme avec l'approche naïve développée précédemment. En particulier, à l'aide de la fonction `clock` on s'intéressera aux temps de calcul respectifs sur différents exemples de difficulté croissante.
Si la grille que vous considérez n'admet pas une solution unique, il est fort probable que cet algorithme et le précédent ne rendent pas la même réponse. Il peut alors être intéressant de chercher toutes les solutions d'une grille avec cet algorithme plus évolué.
Chacun des deux algorithmes sera codé par un binome bien identifié.

4 Aller plus loin

Quelques pistes supplémentaires :

- considérer des grilles de taille $n^2 \times n^2$ avec $n = 4, \ldots,$ dans ce cas on travaillera en base n^2 ;
- générer des grilles aléatoires (qui admettent une solution) ;
- développer une interface avec Qt par exemple, la documentation de la librairie est bien détaillée et regorge d'exemples.