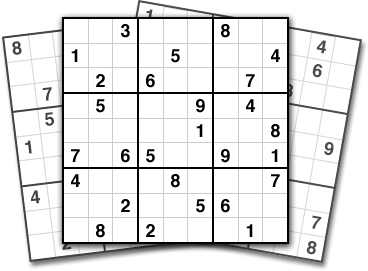
****

Projet d’INF431 - Sudoku

Mehdi Kouhen et Timothée Rebours

*Sujet proposé par Jean-Pierre Tillich*

Ce projet aborde deux points. Tout d’abord, la résolution d’une grille de Sudoku par backtracking simple dans un premier temps, et en utilisant l’algorithme X de Knuth par couverture exacte dans un second temps. Puis la génération de grilles de Sudoku de difficultés variables.



**Table des matières**

1. Introduction 3

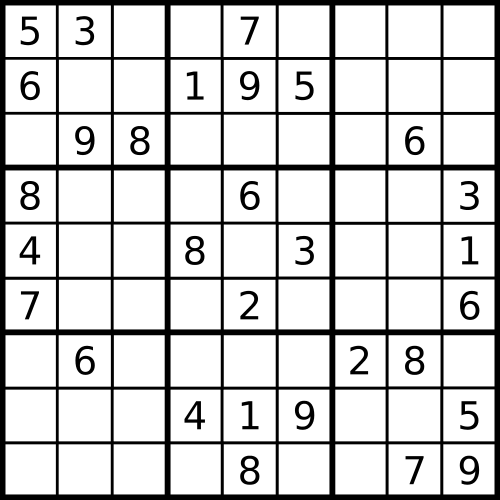
# Introduction

Ce projet a pour objectif de traiter de plusieurs manières la résolution d’une grille de Sudoku générique d’ordre , c’est-à-dire de taille , où est la taille d’une sous-grille et le nombre de sous-grilles par côté. Le but du sudoku est, à partir d’une grille partiellement remplie, de compléter les autres cases en respectant les contraintes suivantes :

* Chaque ligne et colonne contient exactement une fois tous les nombres de à ,
* Chaque bloc carré contient lui aussi exactement une fois tous les nombres de 1 à

Généralement, on débute d’une grille qui n’a qu’une seule solution possible. À titre d’information, on ne peut avoir une grille à solution unique si elle ne contient pas 17 entrées ou plus au départ. On appelle d’autre part grille minimale une grille ne possédant qu’une solution et qui, si on supprime n’importe quelle case, perd cette unicité.

Voici un exemple de grille minimale, et de sa solution pour  :

Pour résoudre le Sudoku, nous étudierons tout d’abord un algorithme de *backtracking* simple qui est de complexité temporelle .

Nous emploierons ensuite l’algorithme X de Knuth appliqué à la résolution de Sudoku à l’aide de listes doublement chaînées et la technique des *dancing links*.

Ensuite, nous développerons un programme permettant de générer des grilles valides de Sudoku.

Pour finir nous nous emploierons à générer des grilles minimales en combinant les deux programmes précédents.

1. **Backtracking simple**

Cet algorithme est le plus naïf possible outre le brute-force sans vérification à chaque fois qu’une case est remplie.

Il s’agit d’un appel récursif d’une fonction solve(int[][] matrix, int x, int y) qui remplit une case donnée, où matrix est un tableau représentant le Sudoku d’ordre en cours de résolution, (x,y) sont les coordonnées de la case à remplir.

Tout d’abord on choisit les coordonnées de la case suivante en allant de gauche à droite et de haut en bas. Si on dépasse la taille cela retourne False. Si on choisit une case déjà remplie, on la résout.

Pour résoudre une case vide, on teste toutes les possibilités entre 1 et dans cette case en modifiant la matrice. Si l’une d’entre elle passe tous les tests, on retourne True, sinon on remet la valeur de la case à et on retourne False.