Université de Rennes 1

Master 1 - CyberSécurité

RAPPORT

LLP - SUDOKU

AUTEUR :

Faical Sid Ahmed RAHMANI

31 décembre 2022

Table des matières

1 Introduction

La programmation informatique a une place très importante au sein du cursus informatique et notamment en CyberSécurité. C'est pourquoi pour mettre en pratique nos connaissances et nous développer en Langage C, l'unité d'enseignement Low level programming nous propose de réaliser un projet qui consiste a développer des outils logiciels autour du jeu de Sudoku.

Le sudoku est une grille carrée divisée en n région de n cases et possède n colonnes, n lignes et n^2 cases. La seule régle à respecter est : dans chaque ligne, chaque colonne, chaque région, les chiffres de 1 à n apparaissent une et une seule fois.

Ce projet à été implementé en Langage C et son exécution peut être effectué sur une ligne de commande permettant de tester les différentes options mises en place à l'aide d'un parseur qu'on a mis au début du programme. La réalisation de ce projet se base principalement sur 2 parties :

- Construire un Solveur : une grande partie du solveur à été réalisée pendant les devoirs, et cela consiste à créer un outil qui permet de résoudre les différentes grilles données en paramètre sous forme d'un fichier.
- Création d'un générateur : le génerateur permet de générer une grille à résoudre à partir d'une taille passée en paramètre, et cette grille peut avoir une solution unique ou bien plusieur solutions.

la difficulté du projet (proportions de solutions au regard des grilles possibles) est basée sur la manière d'implementer le backtracking [1] et les différentes heuristiques [2] qui permettent de résoudre les grilles avec une optimisation maximale, ce qui nous donnera les meilleurs complexités possible en temps de calcul.

1. Algorithme de parcours en profondeur d'un arbre avec des contraintes sur les noeuds.
2. Algorithme d'analyse permet d'aboutir en un temps limité à des solutions acceptables.

2 Organisation du projet

2.1 Architecture du programme

Concernant l'architecture du programme, on peut distinguer 3 modules :

Sudoku : ce module représente le module principale, dans lequel on trouve la fonction main, le parseur, et le backtrack.

Grid : ce module permet de reprénsenter la structure d'une grille, et il contient toutes les fonctions nécéssaire qui permettent de gérer une grille : allocation, désallocation, modification, vérification, application des heuristiques....etc.

Colors : il nous offre une structure de cellule qui permet de gérer efficacement une grille quand on essaie de la résoudre. Ce module contient les différentes fonctions qui servent à résoudre une grille comme les heuristiques.

2.2 Structures de données

Afin de bien gérer les grilles, on a définit 3 structure de données :

colors_t : elle représente un entier sur 64 bits (taille maximale d'une grille possible), où les bits 0 à 63 sont utilisés pour détecter la possibilité ou non d'avoir un choix dans une cellule au niveau de la grille.

grid_t : cette structure permet de présenter une grille, elle contient deux champs : le premier permet de stocker la taille d'une grille, et le deuxieme pointe sur la grille.

choice_t : représente un choix pour une cellule au niveau d'une grille ; elle contient la ligne et la colonne de la cellule, ainsi que le choix pour cette cellule.

2.3 Bibliothèques

Tout au long de ce projet, j'ai utilisé les bibliothèques suivantes :

— *<stdio.h>* : fournit les capacités centrales d'entrée/sortie du langage C.

— *<stdlib.h>* : pour exécuter diverses opérations dont la conversion, l'allocation de mé-

moire.

— *<err.h>* : fournit des fonctions permettent d'afficher des messages d'erreurs sur la sortie d'erreur.

— *<unistd.h>* : composée en grande partie de fonctions d'enveloppe d'appel système et de primitives d'E/S (read, write, close, etc.) qui servent à bien manipuler les fichiers.

— *<stdbool.h>* : pour avoir une sorte de type booléen.

— *<time.h>* : pour convertir entre différents formats de date et d'heure.

— *<string.h>* : pour la gestion de chaînes de caractères.

— *<math.h>* : permet de fournir des fonctions mathématiques utiles pour mes calculs.

— *<getopt.h>* : donne accés à la fonction *getopt_long()* qui sert à parser les arguments de la ligne de commande.

2.4 Parser

Pour analyser la ligne de commande pendant l'exécution de programme, et produire le résultat souhaitée par l'utilisateur, j'ai implémenté un parseur à l'aide de la bibliothèque *<getopt.h>* qui fournit la fonction *getopt_long()*, celle-ci permet d'analyser les arguments données par l'utilisateur au moment de l'exécution, ainsi que les options suivantes :

-h –help : cette option donne des informations concernant la manière d'exécuter le programme.

-V –version : cette option donne des informations concernant la version programme.

-g –generate : sert à générer une grille selon la taille spécifiée

-o –output : permet de rediriger les résultats de l'exécution dans un fichier passé juste aprés cette option

-a –all : utile lorsque on veut résoudre une grille, et elle permet d'avoir toutes les solutions possible pour la grille

-u –unique : utile à la génération d'une grille avec une seule solution possible

-v –verbose : cette option permet de donner des informations concernant le pourcentage des cellule remplies, le nombre de choix éffectués par le backtrack, ainsi que le nombre d'appel si on est en mode solver, par contre, si on est en mode générateur, on donne

aussi la solution pour la grille générée

3 Solver

Le mode solver permet de résoudre des grilles de différente taille. Il dépends de deux aspects : les heuristiques et l'algorithme du backtraking. l'implémentation du solver m'a pris énormément de temps, vu qu'on était sensés faire beaucoup d'éffort en ce qui concerne l'optimisation des différentes fonctions afin de pouvoir passer les challenges notamment ceux qui sont liées à des grilles avec de grande taille 49x49 et 64x64.
Dans un premier temps, j'avais un problème avec les grilles 64x64-03 et 25x25-03 du devoir 4 car ils prenaient beaucoup temps, ce que m'obligeait d'arrêter l'exécution à chaque fois afin d'éssayer d'améliorer les heuristiques. Jusqu'à présent, j'ai réussi à arriver un temps de calcul de 03s à 06s pour ces deux grilles, par contre, j'arrive pas encore à dépasser les challenges level-04 et level-05 du devoir 5.

3.1 Heuristiques

Dans cette partie, je parlerai sur l'implémentation des Heuristiques que j'ai utilisées pour résoudre une grille en donnant la complexité de chacune. Ces fonctions sont appliquées sur les lignes, les colonnes, et les blocs de la grille, elles servent à analyser, éliminer des choix et proposer des solutions possible. les heuristiques implémentées sont les suivantes :

3.1.1 cross hatching

La fonction cherche d'abord tous les choix qui sont déja résolus, et puis elle parcours toutes les cellules pour enlever les choix résolus afin de diminuer les choix possibles.
complexité : $O(n)$

3.1.2 lone number

On fait un premier parcours pour chercher les choix qui ne sont possible que dans une seule cellule, ensuite un deuxième parcours afin de fixer ces choix.
complexité : $O(n)$

3.1.3 naked subset

Cette fonction m'a pris un peu de temps pour son optimisation. Pour chaque cellule de n choix on cherche si ces choix se répètent dans n cellules, alors on les éliminent des autres cellules.
complexité : $O(n^2)$

3.1.4 hidden subset

Pareil que la fonction précédente, cette fonction a demandé quelques éfforts pour l'optimiser un peu. Pour chaque cellule de n choix, on va vérifier si on trouve l'un de ces choix dans n cellules, alors on élimine les autres choix dans les n cellules.
complexité : $O(n^2)$

3.2 Backtraking

L'application des heuristiques n'assure pas forcément la résolution d'une grille. Parfois on est obligé d'utiliser l'algorithme du backtrack qui permet de chercher la solution en faisant un parcours d'un arbre.

3.2.1 Implémentation

La fonction du backtrack fonctionne en deux mode : le mode first qui permet de retourner la première solution trouvée, et le mode all qui cherche toutes les solutions possible. La recherche des solutions consiste à appliquer le principe de la récursivité sur la fonction. Pour chaque appel,

ma fonction du backtrack commence par appliquer les heuristiques sur la grille, et selon la valeur retournée par les heuristiques, on peut distinguer 3 cas possible :

- grille inconsistante : dans ce cas on arrête la recherche et on revient à l'état précédent.
- une solution est trouvée : on retourne cette solution et on revient à l'etat précédent.
- on n'a pas d'inconsistance et on a pas trouvé de soulution : dans ce cas, on doit boucler sur les différents choix pour une cellule donnée, et pour chaque itération on fait un appel récursif de notre backtrack, ce qui permet de refaire toute le traitement pour chaque choix.

3.2.2 Complexité

On peut imaginer le pire des cas comme une gille de taille n complètement vide, et on suppose que chaque choix ne donne ni une grille consistante ni une solution. Ce qui signifie qu'on va parcourir toutes les choix n pour chaque celulle de la grille et cela donne une complexité de $O(n^n)$ et pour chaque parcours on applique les heuristiques qui ont une comlexité de $n^2 * n = O(n^3)$. ce qui donne à la fin la complexité : $O(n^n * n^3)$

4 Générateur de grille

4.1 Implémentation

Afin de permettre au utilisateur de générer une grille à travers l'option -g, j'ai implementé un générateur qui retourne une grille remplie à 60%, celle-ci peut avoir une ou plusieur solution selon le mode choisi avec l'option -u pour unique.
L'implémentation consiste à créer une fonction principale *generator* dans le module *Sudoku.c* qui fait appel aux fonctions suivantes :

- *grid_random* (grid.c) : cette fonction permet de créer une grille et d'initialiser sa première ligne d'une manière aléatoire.
- *grid_remove_cells* (grid.c) : cette fonction permet de vider 40% de la grille passée en paramètre.

— *check_solution* (sudoku.c) : cette fonction permet de vérifier si une grille a une seul solution ou bien plusieurs. Contrairement au backtrack, cette fonction arrête la recherche si elle détecte que la grille a au moins deux solution (à l'aide d'un compteur global).
— *grid_solver* (sudoku.c) : cette fonction représente le backtrack qui sera utilisé en mode_first.

4.1.1 Mode normal

L'idée du générateur en mode normal, c'est de créer une grille aléatoire à l'aide de *grid_random*, ensuite la remplir à l'aide du *grid_solver*, ce qui garantit d'avoir au moins une solution, et en fin, on appelle la fonction *grid_remove_cells* qui permet de vider 40% des cases.

4.1.2 Mode unique

Ce mode est activé à travers l'option -u afin de récupérer une grille qui a une seule solution. Pour ce mode, on suit les mêmes étapes que pour le mode normal avec une boucle qui s'arrête quand on génère une grille ayant une seule solution, et cela est vérifié à l'aide de la focntion *check_solution*. Concernant ce mode, la génération des grilles de taille 49 et 64 prend beaucoup du temps à cause des problèmes d'optimisation liées aux heuristiques.

4.2 Complexité

On peut considérer que la complexité dépend de la complexité du solver qu'on a indiquée précédement : $O(n^n * n^3)$ car les complexité des autres fonctions sont négligées par rapport à cette complexité.

5 Tests réalisés

Durant ce porjet, on était sensé faire plusieur tests afin d'assurer le bon fonctionnement du jeux. Les tests que j'utilisais le plus souvent concerne beaucoup plus le temps de calcul et l'allocation

mémoire :

5.1 time

Cette commande m'a permise de faire des tests concernant le temps de calcul de mon programme, je l'utilisais souvent pour me permettre d'améliorer la complexité de mes fonctions. Si on prend l'exemple des grilles du devoir 4, j'ai pu avoir un temps de calcul qui ne dépasse pas 01s pour toutes les grilles sauf les grilles 64x64-03 et 25x25-03 qui ont un temps de calcul entre 03s et 06s, ce qui nécessite d'améliorer encore les fonctions pour avoir les meilleurs résultasts.

5.2 valgrind

Cette commande est utilsée pour détecter s'il y a des corruption ou bien des fuite mémoire pendant l'exécution du programme. Elle nécessite d'être utilisé sur les différentes taille afin de s'assurer qu'on a pas d'érreur au niveau de la mémoire. Grace à cette commande, j'ai pu repérer mais érreur qui concerne beaucoup plus la lébération de la mémoire et le test des pointeurs, ce qui m'a permis de les corrigér et d'assurer de ne pas avoir de fuite mémoire.

6 Conclusion

6.1 Objectifs remplis ?

La réalisation de ce projet, nécessite une organisation, une bonne maitrise du langage C, une gestion du temps. J'ai eu un peu de mal à mettre en place toutes les recommandations pour mener à bien mon projet à cause d'un manque de maitrise en ce qui concerne quelques notions du langage C vu que c'est mon premier projet avec ce langage, et au fait que j'ai passé beaucoup de temps à essayer d'optimiser les différentes fonctions et heuristiques, et en fin, le fait de ne pas bien comprendre les exigences concernant le code en terme de style et implémentation (redondance, affichage sur les sorties d'erreurs et standard, tester les valeurs de retour .....) à respecter et aussi la

manière de tester les différents cas.
Néanmoins, j'ai presque tout implémenté en ce qui concerne les fonctions de base de notre programme. En commençant par implémenter le solveur, qui permet de résoudre l'ensemble des grilles passées en paramètre en réduisant le plus que possible le temps de calcul à l'aide des heuristiques et du backtracking, ensuite par la création du générateur, qui permet de générer une grille selon la taille souhaitée par l'utilisateur avec une ou plusieurs solutions.

6.2 Améliorations possibles

Concernant les améliorations possibles, j'aurait pu améliorer encore les heuristiques que j'ai mises en place ou bien en ajouter d'autres afin de gagner plus de temps.

6.3 Leçons retenues

Tout au long de ce projet, j'ai retenu plusieurs leçons : la nécessité de bien structurer et organiser le code, l'importance de faire plusieurs tests sur un programme avant d'estimer qu'il est parfait en terme de temps de calcul, et de fuite memoire, ce qui permet d'améliorer le programme.
la réalisation de ce projet a été une très belle expérience pour moi, cela m'a permis de mettre en pratique les notions de programmation en C apprissent en cours.