Projet SudoMips

Yassine Hamoudi

8 Janvier 2014

Table des matières

1 Introduction

Le projet SudoMips consiste à réaliser un programme MIPS de résolution de grilles de Sudoku de taille 9x9. Le programme joint à ce rapport effectue cette tâche. Nous nous attacherons ici à présenter son fonctionnement et à détailler les algorithme mis en place.

2 Utilisation du programme

La procédure d'utilisation du programme est brièvement présentée.

2.1 Exécution

Le programme joint (*"Programme.s"*) peut être chargé à l'intérieur du logiciel QtSpim. Cependant, pour améliorer sa rapidité d'exécution, il est recommandé de l'exécuter en entrant la commande suivante dans le terminal :

```
spim -file ./Programme.s
```

2.2 Procédure d'utilisation

L'utilisateur du programme doit tout d'abord choisir une stratégie de résolution à adopter. Deux choix lui sont proposés : la stratégie MIN ou la stratégie MAX. Le fonctionnement de ces stratégies est détaillé partie 3.2.

Il faut ensuite initialiser la grille de départ. Celle-ci peut être remplie depuis le terminal, ou inscrite directement dans le fichier *"Programme.s"*.

Le programme vérifie que la grille de départ ne contient pas déjà des conflits, puis exécute l'algorithme de résolution. Si une solution existe, elle est affichée, sinon l'utilisateur est informé de l'absence de solutions.

3 Fonctionnement du programme

Les algorithmes mis en place, ainsi que leur fonctionnement, sont détaillés.

3.1 Structures de données

La grille du Sudoku est stockée sur 81 bytes consécutifs. Chaque case occupe une position entre 0 et 80. Connaissant cette position, il est possible d'en déduire la ligne et la colonne auxquelles elle correspond dans la grille de Sudoku.

Les différents registres mis à disposition, ainsi que la structure de pile, sont également utilisés tout au long du programme.

3.2 Backtracking

L'algorithme de Backtracking est le coeur du programme. Il met en place une procédure complète de résolution de grilles de Sudoku.

3.2.1 Présentation générale

Le Backtracking consiste à remplir la grille de Sudoku autant que possible, en partant de la première case vide, et à revenir sur ses pas lorsqu'il n'y a plus aucune solution d'envisageable. Deux stratégies de remplissage sont proposées :

- **la stratégie MIN :** dans chaque case de la grille, on essaye de mettre *le plus petit chiffre* compris entre 1 et 9 qui ne provoque pas de conflit.
- **la stratégie MAX :** dans chaque case de la grille, on essaye de mettre *le plus grand* chiffre compris entre 1 et 9 qui ne provoque pas de conflit.

L'algorithme considère la première case vide de la grille et essaye de lui assigner une valeur selon la stratégie choisie. S'il y parvient, il passe à la case suivante, sinon il revient à la case précédemment modifiée et lui assigne une valeur différente.

La pile est utilisée pour stocker les cases modifiées successives. A chaque fois qu'une case subie une assignation, sa position est ajoutée en haut de pile. Ainsi, lorsque l'on souhaite revenir en arrière, il suffit de récupérer le haut de pile pour connaître la case précédemment modifiée.

Sur l'exemple figure 1, dans la grille à droite, l'algorithme avec stratégie MIN s'apprête à inscrire le nombre 7 dans la case 6 (première case non vide) et à empiler la valeur 6. A l'étape suivante, aucun chiffre ne pourra être inscrit dans la case 7. L'algorithme va alors revenir en arrière et changer le nombre 7, placé dans la case 6, par le nombre 8.

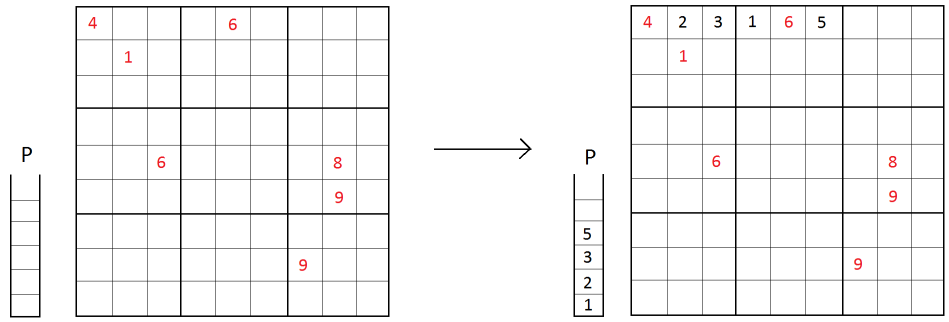

FIGURE 1 – A gauche, l'état initial. A droite, l'état après un certain nombre d'opérations (stratégie MIN). P représente la pile.

3.2.2 Algorithme

Voici l'algorithme de Bactracking avec stratégie MIN appliquée :

Algorithme 1 : Backtracking avec stratégie MIN

input : G : grille initiale (tableau de taille 81). Les cases initialement vides contiennent 0.
output : G remplie, s'il existe une solution, ou un message d'erreur.

```
P ⟵ −1            // P est la pile. On empile -1 pour repérer le fond.
pos ⟵ 0           // position courante de travail dans la grille.
while pos < 81 do
    if pos est une position modifiable then
        if MIN(G[pos]) est défini          // plus petit nombre strictement
        supérieur à G[pos] pouvant être mis à la position pos.
        then
            G[pos] ⟵ MIN(G[pos])
            push(P, pos)                   // pos est ajouté sur la pile
            pos = pos + 1
        else
            pos ⟵ pop(P)                   // on récupère le haut de pile.
            if pos = −1                    // si le fond de pile est atteint
            then
                return Aucune solution n'existe;
    else
        pos = pos + 1
return G;
```

3.2.3 Complexité

Le problème du remplissage d'une grille de Sudoku étant NP-complet, aucun algorithme de résolution s'exécutant en temps polynomial n'est actuellement connu. L'algorithme mis en place ici peut donc s'exécuter en temps exponentiel sur certaines entrées.

On constate cependant à l'usage que l'algorithme termine en moins de 30s sur la majorité des entrées. Il demeure toutefois difficile d'établir une complexité précise. Ainsi, certaines entrées à première vue anodine requièrent de très nombreuses opérations. Si on considère par exemple la grille *"Grille difficile 1"* figurant dans le fichier *"TEST"*, on remarque qu'en appliquant la stratégie MIN la grille n'est que très faiblement remplie au bout de 10^8 combinaisons testées. Cependant, si on applique la stratégie MAX sur cette même grille, une solution est obtenue après moins de 10^5 tests.

Bien qu'il soit difficile d'estimer le temps nécessaire à la résolution d'une grille donnée, il est recommandé de changer de stratégie si le programme prend trop de temps à s'exécuter.

4 Conclusion

Le problème majeur de l'algorithme mis en place ici demeure sa lenteur d'exécution sur certaines entrées. Différentes modifications de cet algorithme existent et semblent, pour certaines, améliorer la rapidité d'exécution. Ainsi, il peut être intéressant par exemple de modifier le parcours de remplissage de la grille. L'algorithme X est considéré aujourd'hui comme un des meilleurs algorithmes de résolution de Sudoku.

Ces différentes améliorations n'ont pas été implémentées en MIPS dans le cadre de ce projet. Les structures de données qu'elles nécessitent (listes, graphes...) peuvent être difficiles à représenter dans un langage de bas niveau. Par ailleurs, j'ai jugé préférable de travailler sur la clarté et la correction du code pour cette première utilisation du langage assembleur, laissant tout usage plus poussé à des projets futurs.