UNIVERSITÉ TOULOUSE III PAUL SABATIER — Université de Toulouse

L2 info - Algorithmique S3 - Devoir Maison

Résolution de sudokus

Date limite de dépôt sur Moodle : 29 Novembre.

1 Principe du jeu de sudoku

Le jeu prend la forme d'une grille de 9×9 cases structurées en 9 sous-grilles de 3×3 cases, appelées régions . Quelques cases contiennent des chiffres, elles sont dites "dévoilées". Le but du jeu est de remplir les cases vides (c'est à dire de les "dévoiler"). La contrainte à satisfaire est la suivante :

Chaque case doit contenir un chiffre compris entre 1 *et* 9 *de telle sorte que dans chaque groupement (ligne, colonne ou région) on trouve une unique occurrence de tout chiffre allant de* 1 *à* 9.

Lorsque la solution est unique, la grille prend le nom de sudoku.

Exemple :

0	8	7	0	0	0	5	2	0
9	1	0	5	0	2	0	4	6
2	0	0	0	0	0	0	0	7
0	9	0	0	2	0	0	1	0
0	0	0	1	0	6	0	0	0
0	4	0	0	9	0	0	8	0
6	0	0	0	0	0	0	0	3
5	7	0	3	0	1	0	6	8
0	3	8	0	0	0	9	5	0

Les cases non dévoilées sont à 0.

Vocabulaire : Les groupements sont les lignes, colonnes et régions. Ils sont numérotés de 0 à 8. La numérotation des régions est :

0	1	2
3	4	5
6	7	8

On peut :

- pour une région donnée, exprimer les coordonnées de la case située en haut et à gauche de cette région:

$$lig = 3 * (region/3) \qquad col = 3 * (region\%3)$$

- pour une case [lig][col], exprimer le numéro de sa région: $3 * (lig/3) + (col/3)$
- pour une case [lig][col], exprimer son numéro (compris entre 0 et 80) : $9 * lig + col$
- pour une case de numéro c (compris entre 0 et 80), exprimer son indice de ligne: $c/9$ et de colonne: $c\%9$

2 Méthode de résolution: implémentation de diverses règles

Pour chaque case non dévoilée les candidats sont les chiffres $1, 2, 3, 4, 5, 6, 7, 8, 9$. En tout, il y a donc : $9^{\text{nombre de cases non dévoilées}}$ grilles candidates, nombre trop important pour pouvoir faire une recherche exhaustive. Afin de diminuer ce nombre on applique les règles détaillées ci-dessous. Les divers principes déductifs utilisés dans la résolution sont décrits sur le site "http://www.mots-croises.ch/Manuels/Sudoku/"

Alors que certaines règles permettent une détermination directe du contenu d'une case non dévoilée (R1 candidats seuls, R3 candidats uniques), la règle R2 permet seulement de diminuer le nombre de candidats de certaines cases.

- **R1 Candidats seuls**: S'il n'y a qu'un candidat à une case, alors il doit être placé dans la case.

- **R2 Une seule occurrence de chaque chiffre dans chaque groupement**: Un candidat à une case ne peut être qu'un chiffre non encore placé dans un groupement (ligne, colonne et région) auquel appartient cette case.

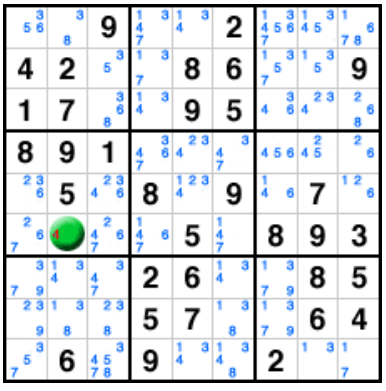
  Ainsi, dans la grille ci-contre, la liste des candidats de la case 0 ne peut contenir:

  · ni 9 ni 2 : déjà placés sur la ligne 0

  · ni 4 ni 1 ni 8 : déjà placés dans la colonne 0

  · ni 2 ni 7 : déjà plaçés dans la région 0

  il ne reste donc plus que 3 candidats : 3 5 et 6 pour cette case aprés l'application de R2.



  L'application de ces deux seules règles est suffisante pour résoudre des sudokus faciles (cf exemple 1) mais, en général, elles ne suffisent pas. Dans l'exemple 2 leur utilisation permet seulement de diminuer à 3.5×10^{26} le nombre de grilles candidates. Il est donc nécessaire de faire intervenir d'autres règles.

- **R3 Candidats uniques**: Un candidat n'apparaissant que pour une seule case d'un groupement peut être placé dans la case correspondante.
  Exemple d'application pour une ligne:

  Avant application de la règle:

  Après application de la règle:

  Dans l'exemple 2 l'application de R3 permet de diminuer à 1.12×10^{15} le nombre de grilles candidates ce qui n'est pas encore assez faible pour envisager une recherche exhaustive.

  Certaines grilles ne peuvent être résolues par l'application des règles précédentes, mais le nombre de grilles candidates devient assez faible pour permettre une recherche exhaustive.

3 Travail à réaliser, notation

On veut écrire un programme de résolution d'un sudoku: lorsqu'on lui fournit une grille initiale de sudoku, le programme cherche la solution et l'affiche à l'écran. Pour cela, il vous est fourni un fichier *sudoku.h* résumant l'interface de programmation qu'il vous est demandé d'implémenter **dans le fichier sudoku.c**. On y trouve la déclaration des types de base (cf. section suivante), ainsi que des fonctions que vous devrez écrire et qui seront testées par le script *compilation_et_notation.py*.

3.1 Contraintes d'implantation:

Une case sera implémentée par le type T_case suivant:

```
typedef struct
{
    int val;              /* Valeur de la case (0 si vide)*/
    int n_candidats; /* Nombre de candidats possibles */
    int candidats[9]; /* Liste de candidats */
} T_case;
```

Lorsque pour une case le champ val vaut 0 (la case est vide), le nombre de candidats peut fluctuer lorsqu'on applique des règles. A chaque suppression d'un candidat, les candidats restants doivent être groupés au début du tableau candidat (entre la case d'indice 0 et la case d'indice n_candidats-1), et toutes les cases situées entre les indices n_candidats et la fin du tableau doivent prendre la valeur 0. Lorsqu'une valeur est affectée à une case (val !=0), le nombre n_candidats de la case doit être mis à 0 et toutes les cases du tableau de

candidats doivent être mises à 0. **Attention, ces contraintes sont testées par le script de notation, leur non-respect correspond à la note 0 sur la règle en cours d'évaluation.**

Pour des considérations d'optimisation de la mémoire, une grille de sudoku sera implémentée par le type T_sudoku suivant:

```
typedef struct
{
    T_case grille[81]; /*vecteur de 81 cases*/
}
T_sudoku;
```

En plus de son indice dans la grille, chaque case peut être caractérisée par ses coordonnées (ligne, colonne) via le type T_coordonnees suivant:

```
typedef struct
{
    int ligne;
    int colonne;
}
T_coordonnees;
```

A partir d'une case d'indice c (entier compris entre 0 et 80) on peut déterminer facilement l'indice de ligne ou de colonne associé. (voir page 1)

Vous devez obligatoirement utiliser les structures de données précédentes lors de votre codage.

Les fonctions que vous devez implémenter sont réparties (et évaluées) en groupements:

- Dans la partie 0, on vous demande d'implémenter les fonctions *obtenirCoords* qui à partir d'un indice retourne les coordonnées correspondantes, *obtenirIndice* qui à partir d'une variable de type coordonnées, renvoie l'indice correspondant dans la grille, *debutRegion* qui renvoie les coordonnées du premier élément (en haut à gauche) d'une région à partir de l'indice de la région, *indiceRegion* qui à partir des coordonnées d'une case renvoie l'indice de la région correspondante, et *lireSudoku* qui à partir d'un chemin (chaine de caractères) vers un fichier lit le sudoku présent dans ce fichier (9 lignes de 9 entiers séparés par une tabulation). Le script de test et de notation de ces fonctions est présent dans le fichier *notation0.c* qu'il ne faut pas modifier,

- Dans la partie 1, on vous demande d'implémenter la règle R1 sous la forme de deux fonctions: la fonction *R1_case* qui applique la règle sur une case donnée et retourne 1 si la règle a été appliquée (une valeur a été affectée à la case) et 0 si la règle n'a pas été appliquée, et la fonction *R1_sudoku* qui appelle *R1_case* successivement sur toutes les cases du sudoku, et à chaque fois en cas d'affectation de valeur à une case retire cette valeur des listes de candidats des cases de la même ligne, même colonne et même région. Le script de test et de notation de ces fonctions est présent dans le fichier *notation1.c* qu'il ne faut pas modifier,

- Dans la partie 2, on vous demande d'implémenter la règle R2 sous la forme de deux fonctions: la fonction *R2_case* qui applique la règle sur une case donnée et retourne 1 si la règle a été appliquée (au moins un candidat a été retiré de la case) et 0 si la règle

n'a pas été appliquée, et la fonction *R2_sudoku* qui appelle *R2_case* successivement sur toutes les cases du sudoku. Le script de test et de notation de ces fonctions est présent dans le fichier *notation1.c* qu'il ne faut pas modifier,

- Dans la partie 3, on vous demande d'implémenter la règle R3 sous la forme de deux fonctions: la fonction *R3_case* qui applique la règle sur une case donnée et retourne 1 si la règle a été appliquée (une valeur a été affectée à la case) et 0 si la règle n'a pas été appliquée, et la fonction *R3_sudoku* qui appelle *R3_case* successivement sur toutes les cases du sudoku, et à chaque fois en cas d'affectation de valeur à une case retire cette valeur des listes de candidats des cases de la même ligne, même colonne et même région. Le script de test et de notation de ces fonctions est présent dans le fichier *notation3.c* qu'il ne faut pas modifier,

- Dans la partie 4 (bonus), on vous demande d'implémenter une solution de force brute qui résout un sudoku à partir de son état courant. Cette fonction construit successivement tous les sudokus possibles (en utilisant les candidats restants des cases n'ayant pas encore de valeur), et renvoie 1 lorsqu'elle a réussi à construire une grille complète. Dans le cas où cette fonction épuise toutes les possibilités sans trouver de solution correcte, elle renvoie 0. Le script de test et de notation de cette fonction est présent dans le fichier *notation4.c* qu'il ne faut pas modifier. Cette évaluation peut prendre quelques minutes à s'exécuter, selon votre machine. Il vous est donc conseillé de tester votre fonction de brute force avant de lancer l'évaluation.

Pour vous aider, vous pouvez implémenter les fonctions (qui ne seront pas évaluées séparément):

- *rechercherValeur* qui étant donné une case et une valeur retourne l'indice correspondant à cette valeur dans le tableau de candidats de la case. Si aucun candidat ne correspond, la fonction renvoie le nombre n_candidats de la case,

- *supprimerValeur* qui étant donné une case et un indice supprime l'élément du tableau de candidats de la case correspondant à l'indice donné (et réduit n_candidats de 1). Le tableau résultant contient toutes les valeurs non nulles à gauche et toutes les valeurs supprimées (remplacées par des 0) à droite,

- *valide* qui étant donné un sudoku renvoie 1 si la grille de ce sudoku est complète (pas de 0) et valide (selon les règles de base du sudoku),

- *verifResultat* qui étant donné le sudoku dans son état de départ et dans son état courant vérifie que l'état courant est complet et valide, et qu'il correspond bien au sudoku de départ (les cases initialement non-nulles ont toujours la même valeur dans le sudoku courant.

Les deux premières fonctions (recherche et suppression d'un candidat dans une case) sont utiles pour l'implémentation de la règle R2, et les deux dernières (validité) peuvent être utiles pour implémenter le brute force.

3.2 Notation

La notation se fera par le biais d'un script qui testera vos diverses fonctions sur des exemples. Ce script vous est fourni, ainsi qu'une partie des exemples qui seront utilisés

lors de l'évaluation, afin que vous puissiez tester et améliorer vos fonctions. La note provisoire fournie par le script reflète votre score uniquement sur les tests fournis et ne garantit donc pas votre note finale. Le script de notation compile et teste séparément **et dans l'ordre** les différentes parties (0 à 4). Si l'une des parties ne compile pas, le script s'arrête et affiche votre note provisoire : vous devez donc implémenter vos fonctions dans l'ordre des parties. Les parties 0 et 1 sont notées 2/10 chacune, les parties 2 et 3 sont notées 3/10 chacune, et la partie 4 est notée 3/10 en points bonus pour la note finale. Pour obtenir la moyenne, il faut donc réussir au minimum l'implémentation des parties 0 à 2 inclue. Le script s'exécute rapidement pour les parties 0 à 3 inclue, mais peut prendre une à plusieurs minutes (selon la puissance de votre machine) pour la partie 4 (si la fonction de force brute est implémentée).

3.3 Rendu

Il vous est demandé de déposer sur moodle le fichier **sudoku.c** complété avant le **29 Novembre 23h55**.

4 Exemples:

4.1 Exemple 1: R1+R2

0	8	7	0	0	0	5	2	0
9	1	0	5	0	2	0	4	6
2	0	0	0	0	0	0	0	7
0	9	0	0	2	0	0	1	0
0	0	0	1	0	6	0	0	0
0	4	0	0	9	0	0	8	0
6	0	0	0	0	0	0	0	3
5	7	0	3	0	1	0	6	8
0	3	8	0	0	0	9	5	0

Après application des règles R1 et R2, le sudoku est résolu.

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

4.2 Exemple 2: R1+R2+R3

0	0	0	0	7	0	0	0	5
2	0	7	1	0	0	4	0	0
0	6	0	4	0	0	1	0	7
5	0	6	7	0	0	2	0	0
0	0	0	0	9	0	0	0	4
0	9	0	5	0	0	3	0	0
0	0	0	0	0	0	0	0	8
3	0	5	0	1	8	0	6	2
9	0	8	0	4	2	0	1	3

Après application des règles R1 et R2:
12 cases dévoilées.

0	0	0	0	7	0	0	0	5
2	0	7	1	0	0	4	0	0
8	6	0	4	0	0	1	0	7
5	0	6	7	0	0	2	0	0
0	0	0	0	9	0	0	0	4
0	9	0	5	0	0	3	0	0
0	0	0	3	5	7	9	4	8
3	4	5	9	1	8	7	6	2
9	7	8	6	4	2	5	1	3

Après application des règles R1 R2 et R3:
sudoku résolu

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