clement.rouvroy@ens.psl.eu

Sudoku

Thèmes Retour sur trace

Consignes Le candidat respectera le langage imposé (C) et ne devra pas utiliser de librairie hors-programme. Il est invité à **faire des schémas clairs et précis** de ses algorithmes lors des appels au correcteur, c'est à la fois un gain de temps pour les deux et une manière de montrer qu'il a compris ce qu'il manipule. Les preuves écrites doivent être formelles, sauf si la consigne précise que ce n'est pas nécessaire, la rigueur sera évaluée. L'examinateur ne débugera pas votre code, en revanche en cas de doute ("ai-je le droit à telle ou telle librairie ?", "je suis sortie de la VM, comment y revenir ?", "ai-je le droit à une indication pour cette question ?") n'hésitez pas à poser votre question. Elle ne vous dévalorisera pas, si la réponse peut vous retirer des points, l'examinateur vous demandera avant de vous donner la réponse si vous l'acceptez (*si vous n'avez pas de chance, le jour de l'oral il vous retirera les points rien que pour avoir posé la question, par exemple si vous demandez "comment trier une liste en $O(n\,log(n))$?" ou un autre résultat classique du programme, ça sera sûrement retenu contre vous*). Enfin, si l'examinateur n'est pas à votre portée quand vous avez besoin d'une aide / d'une question oral à donner, gardez le bras levé et lisez la suite, ne restez jamais passif.

Introduction Dans ce TP vous allez programmer un solveur pour une grille de Sudoku donnée en retour sur trace. Pour rappel:

- Une grille de Sudoku est une grille de taille 9×9, découpée en 9 carrés de taille 3×3
- On cherche à remplir la grille avec des nombres dans $[\![1; 9]\!]$ de sorte que chaque petit carré contienne une fois chaque chiffre de $[\![1; 9]\!]$
- Aucune colonne ou ligne ne doit contenir le même chiffre.

Rappels Voici quelques fonctions C de la bibliothèque `stdio.h` (et des opérateurs génériques) qui peuvent vous êtres utiles, le programme précise que vous devez être capable de les maîtriser après rappel de la syntaxe.

- `fopen(filename: char*, mode: char*)` permet d'ouvrir un fichier. Le mode sera `r` (lecture), `r+` (lecture + écriture) ou `w` (écriture). Cette fonction renvoie un `FILE*`
- `fscanf(stream: FILE*, format: char*, arg1,..., argn)` effectue `scanf` sauf que son entrée n'est pas l'entrée standard mais le fichier passé en argument.
- `fprintf` même logique que `fscanf`
- `fclose(stream: FILE*)` permet de fermer un fichier
- `<<` est un left shift (un décalage à gauche) binaire. On l'utilise avec `x<<k` pour décaler `x` de `k` bits vers la gauche.
- `>>` est un right shift (comme pour `<<` mais vers la droite)

clement.rouvroy@ens.psl.eu

I - Pendant que ça charge

Question 1 à l'écrit

ANNULE: Prouver que le rendu de monnaie est glouton-solvable pour le système européen.

1. Quel tri doit-on faire sur les tâches pour que la réservation de salle soit optimale ?
2. Le système $(1, 3, 4)$ est-il glouton-solvable pour le rendu de monnaie ?
3. Qu'est-ce qu'une table de hachage ? (En 2 phrases ou en 1 dessin)

II - Primitives

II.1 - Type

On propose le type suivant (et il vous est imposé):

- Une `grille` est un tableau de dimension 9×9 de `int`.
- Le `int` est soit 0 (case non remplie), soit $i \in [\![1; 9]\!]$ (case remplie avec i)

Question 2 code

Implémenter le type `grille` en `C`, ainsi que la fonction `grille creer_grille_vide()`

Question 3 en commentaire de code

Pourquoi ne renvoie-t-on pas une `grille*` ? En particulier, pourquoi les modifications faîtes sur le terrain seront pris en compte même si notre fonction de modification a un type de retour `unit` ?

II.2 - Grille correcte

Question 4 code

Ecrire une fonction `bool compatible(grille g, int i, int j, int k)` indiquant si l'ajout de la valeur k en (i, j) est compatible aux règles du sudoku.

Question 5 code

Ecrire une fonction `bool correcte(grille g)` qui dit si la grille passée en entrée est correcte.

II.3 - Charger depuis un fichier

Vous avez 3 fichiers `grille1`, `grille2` et `grille3` dans le répértoire du sujet. Le format est simple:

```
x11 x12 x13 x14 x15 x16 x17 x18 x19
x21 x...
...
```

Question 6 code

Ecrire une fonction `grille grille_depuis_fichier(char* nom)` qui charge une grille depuis un fichier.

III - Force brute

On appelle une grille **complète** si chaque valeur a été complétée.

Question 7 à l'écrit

Soit une grille de sudoku g qui a k valeurs déjà remplies. On souhaite procéder par force brute pour trouver une solution, combien de cas vont être testés dans le pire cas (en fonction de k) ?

Question 8 à l'écrit

Si on procède par bruteforce en testant tous les remplissages **complets** possibles, quelle fonction sera utile: `compatible` ou `correcte` ?

On ne codera pas l'algorithme bruteforce ici, pour se concentrer sur le retour sur trace.

IV - Retour sur trace

Question 9 à l'écrit

Définir le backtracking en une idée clé et un schéma caractéristique.

Question 10 code

Implémenter un algorithme de backtracking pour le sudoku.

Question 11 à l'écrit

Complexité en espace ?

Question 12 à l'écrit

Parmi les 3 grilles d'exemples, lesquelles ont une solution ? Si votre code ne termine pas en un temps acceptable, indiquez "impossible" sur votre copie.

V - Amélioration de la vérification

Cette partie va améliorer l'algorithme de détection de collisions grace à l'usage des nombrs binaires.

On va manipuler des sous-ensembles de $[\![1;9]\!]$, on va représenter un sous-ensemble E par $n_E = \sum_{k=0}^{8} a_k 2^k$ avec $a_k = 1$ ssi $k+1 \in E$ et 0 sinon.

Question 13 code

Ecrire une fonction `int singleton(int k)` qui renvoie $n_{\{k\}}$ en une opération binaire.

On se donne désormais une structure `sudoku` qui contient une `grille` ainsi que trois `tableaux` d'entiers de taille 9 représentant les ensembles de valeurs déjà placées par lignes, colonnes et carrés.

Question 14 code

Ecrire une fonction `sudoku depuis_grille(grille g)` qui convertit un grille en un type `sudoku`.

Question 15 code

Ecrire une fonction `bool collision_bis(sudoku s, int i, int j, int k)` qui renvoie vrai ssi l'ajout de k en (i,j) créé une collision.

Question 16 à l'écrit

Quelle est la complexité de `collision_bis` ?

Question 17 code

Ecrire une fonction `void ajoute(sudoku s, int i, int j, int k)` qui ajoute à une grille de sudoku l'entier k (elle pré-suppose que l'ajout ne créé pas de collision). On pensera à maintenir les trois nouveaux tableaux.

Question 18 à l'écrit

Quelle est la perte en mémoire liée à cette structure ? Quel est le gain en temps lié à cette structure ?

Question 19 code

Ecrire une fonction `bool resoud(sudoku s)` qui résoud une grille `s` avec cette nouvelle structure.