Projet “Nombres croisés”

Module IF2IK3

Nombres croisés

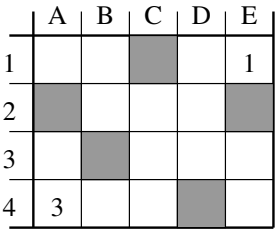
De manière similaire aux mots croisés, le jeu des nombres croisés demande de remplir les cases blanches d'une grille avec des nombres naturels. Ces nombres doivent satisfaire les contraintes posées par les indications données pour chaque suite de (au moins deux) cases blanches horizontalement et verticalement (voir la figure 1). Les indications peuvent être, par exemple : une opération arithmétique ($162 - 43$), une propriété du nombre (palindrome, carré, puissance de 2, nombre premier de 3 chiffres), un encodage (1101 en base 10), une décomposition (facteurs de 168, somme des nombres égale à 38), etc.

Projet minimum

Pour cette partie, on considère que les seules indications possibles sont les facteurs des nombres donnés et que chaque case contient un chiffre (donc des nombres de 1 à 9).

La classe `Grille` : Le projet doit définir une classe `Grille` qui inclut :

- la définition de la *structure de données* représentant la grille du jeu et ses indications,
- un *constructeur* par défaut et un constructeur ayant comme paramètre les dimensions de la grille (c'est-à-dire, le nombre de cases horizontalement et verticalement),
- la *méthode de lecture* d'une grille à partir d'un fichier. Cette méthode sera donnée en TP. Le format du fichier lu par cette méthode est explique ci-dessous.
- une *méthode d'affichage* d'une grille.
- une *méthode* vérifiant que la *grille* du jeu est *correcte* par rapport aux règles fixées, par exemple : absence d'indication 0, indication pour toutes les suites de cases blanches de longueur au moins 2, absence de case pré-remplie à 0, absence de case pré-remplie qui n'est pas un facteur du nombre indiqué.
- une *méthode* de *pré-traitement* de la grille qui remplit les cases “imposées” par les indications et par le pré-remplissage de la grille. Par exemple, dans la figure 1, les avant dernières cases de la première ligne et de la première colonne sont imposées à 8 et 6 respectivement.
- une *méthode* de *résolution* de la grille. Cette méthode commence par vérifier que la grille du jeu est correcte, puis elle lance le pré-traitement de la grille pour remplir les cases



1. facteurs de 20 ; carré.
2. facteurs de 70.
3. facteurs de 10.
4. facteurs de 72.

A. facteurs de 18.
B. facteurs de 35.
C. carré.
D. facteurs de 80.
E. carré.

Fig. 1 – Exemple de grille de nombre croisés.

"imposées" ; si la grille n'est pas remplie complètement par le pré-traitement, elle lance l'algorithme de recherche de solution(s).
- une *méthode de test* de l'algorithme de résolution qui lit une grille à partir d'un fichier dont le nom est donné en paramètre et appelle la méthode de résolution.

La classe `Grille` doit aussi inclure les méthodes utilisées par la méthode de résolution de la grille ; ces méthodes seront discutées en cours et en TP.

Recherche de solutions : Pour cette partie du projet, la recherche de solution **doit être** faite en utilisant un algorithme de *backtracking explicite*. Les principes généraux des algorithmes de *backtracking* ainsi que des exemples de tels algorithmes seront vu en cours.

Le programme fait en utilisant cet algorithme est supposé pouvoir dire si la grille de départ a une et une seule solution (et dans ce cas l'afficher), aucune solution (et dans ce cas le dire) ou plusieurs solutions (et dans ce cas en donner au moins 2, voire les donner toutes).

Format du fichier de grilles : Les fichiers qui seront lus par la méthode fournie en TP sont des fichiers texte ayant le format suivant :
- en première ligne un entier N qui indique le nombre de lignes,
- en deuxième ligne un entier M qui indique le nombre de colonnes,
- les N lignes suivantes contiennent chacune M caractères avec la convention suivante : `-` indique une case blanche, `X` indique une case noire et tout autre nombre positif donne un pré-remplissage de la case.
- les N lignes suivantes sont préfixées par des nombres de 1 à N et donnent les indications pour les lignes. Ces indications sont des listes, chaque élément de la liste ayant le format `codeop=nombre`, où `codeop` est une chaîne de caractère qui indique le type d'indication et `nombre` est le nombre indication. Pour le projet minimum, le seul code d'opération présent est `fact`. D'autres codes d'opérations sont considérés dans la 3ème variation du projet.
- les M lignes suivantes sont préfixées par les premières M lettres de l'alphabet et donnent les indications pour les colonnes, avec le même format que pour les lignes.

```
4
5
- - X - 1
X - - - X
- X - - -
3 - - X -
1 fact=20 carre=0
2 fact=70
3 fact=10
4 fact=72
A fact=18
B fact=35
C carre=0
D fact=80
E carre=0
```

Par exemple, pour la grille donnée dans la figure 1, le fichier d'entrée est donné ci-contre. (L'indication `carre` signifie que le nombre écrit dans la suite des cases est un carré ; c'est pourquoi le nombre indication est 0, c'est-à-dire inconnu.)

Toutes les variations qui suivent doivent être implémentées dans des classes qui étendent (héritent) de la classe `Grille`.

Première variation

Reprendre la recherche de solution en utilisant un algorithme *récursif* plutôt qu'un algorithme explicite de backtracking.

Deuxième variation

Traiter le même problème si les cases peuvent être remplies avec des nombres naturels non-nuls. Pour réduire l'espace de recherche de solution, il est possible de calculer pour chaque case l'ensemble de nombres qui peuvent être affectés à cette case compte tenu des indications

et du pré-remplissage. Cet ensemble est mis àjour au fur et à mesure des choix faits par l'algorithme de backtracking. Pour implémenter cette solution, il faut définir une nouvelle classe `GrilleControle` qui étend la classe `Grille` afin de réutiliser les méthodes écrites dans le projet minimum. La classe `GrilleControle` comporte un nouveau champ qui est une grille de la même dimension que la grille du jeux et dont les cases sont des ensemble de nombres naturels. Ce champ supplémentaire est utilisé par les méthodes permettant de générer une valeur possible pour une case dans l'algorithme de backtracking.

Troisième variation

En supposant mise en place la grille de contrôle, on peut l'utiliser avant de lancer le backtracking pour vérifier que le problème n'est pas trivialement sans solution en regardant que (a) pour chaque case inconnue, l'ensemble de valeurs possibles est non vide, (b) pour chaque suite de cases à remplir, les valeurs possibles dans chaque case permettent de satisfaire la contrainte donnée en indication.

Toujours en disposant de la grille de contrôle, on peut améliorer la méthode de pré-traitement pour calculer d'autres cases "imposées" : il s'agit des cases pour lesquelles l'ensemble de valeurs possibles est formé d'un seul élément. De plus, le pré-traitement peut être répété jusqu'à ce qu'aucune case ne soit ainsi imposée. Si, par chance, la grille est maintenant pleine, le problème est résolu et peut être classé "Facile". Sinon, on reprend les contrôles (a) et (b) ci-dessus ; si ceux-ci ne montrent pas que le problème est impossible, on relance le backtracking sur une grille plus pleine qu'au début.

Quatrième variation

Reprendre la recherche de solution pour accepter d'autres indications comme : carré, palindrome, somme, puissance, codage binaire, etc. La méthode de lecture donne une liste possible de ces indications.

Compléments indépendants

On peut, une fois la recherche de solution(s) terminée,

- si une seule grille a été trouvée, chercher si la grille d'origine est minimale, c'est-à-dire que dès qu'on enlève une case pré-remplie, le problème devient à solutions multiples,
- si plusieurs solutions ont été trouvées, déterminer le nombre minimal de nouvelles cases à fixer pour obtenir une seule grille comme solution,
- s'il n'y a pas de solution, déterminer si on peut obtenir une solution en enlevant certaines valeurs pré-remplies.

Programmation en Java

Plusieurs notions de Java seront (re)vues en cours : héritage, méthode `toString`, interface, API pour les structures de données classiques (pile, vecteur, ensemble), écriture de la documentation des programmes avec Javadoc.

Il est attendu que vous utilisiez ces notions dans votre projet.

Les noms des classes utilisées ci-dessus n'ont rien d'obligatoire. Chacun peut choisir ceux-ci comme il l'entend.

Organisation

Les soutenances seront individuelles, même si le projet a été réalisé en binôme. Elles auront lieu sous Eclipse dans la configuration disponible en TP. Le jury écoutera votre présentation du travail effectué, testera avec vous les différentes parties du projet et vous posera des questions sur le code présenté, le rapport du projet et les notions vues en cours. Chaque soutenance dure 20 minutes.

Les projets pour lesquels la version minimale ne marche pas n'auront pas 10.

Le rapport du projet sera remis au secrétariat du département avant le 18 décembre à 12h. Le rapport doit contenir : environ 10 pages d'explications en français et un listing du programme. La partie explications présente pour chaque variation, ce qui a été réalisé effectivement, les éléments nécessaires pour comprendre les algorithmes implémentés, ainsi que des remarques qualitative et quantitatives sur les tests effectués pour ces algorithmes (performances en temps et mémoire, difficultés techniques, etc.). Le rapport indiquera également les outils Java utilisés. Pour un projet fait en binôme, un seul rapport sera remis ; celui-ci indiquera ce que chacun a fait ou n'a pas fait dans le programme présenté.

Les programmes testés lors de la soutenance doivent être commentés. Ils peuvent varier un peu de la version contenue dans le listing du rapport, par exemple si les variations corrigent des bogues ou implémentent des variations.