Programmation 2 – Projet

Ocan Sankur <ocan.sankur@ens-cachan.fr>

April 1, 2012

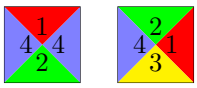
À rendre avant le **16 Avril 2012** (minuit). Date limite stricte.
La page des TPs: `http://www.lsv.ens-cachan.fr/~sankur/prog2`

1 Puzzle

Dans ce projet, vous allez écrire un programme en **OCaml** (obligatoirement) qui résoud le puzzle suivant. On considère un tableau de taille $m \times n$. On a $m \cdot n$ pièces carrées, dont chaque côté est coloré, parmi les couleurs $1 \dots c$.

L'entrée est constituée de m, n, c, et $m \cdot n$ pièces. Le but est de determiner si on peut placer les pièces dans le tableau de manière à ce que les côtés des pièces qui sont en contact aient la même couleur, et d'afficher une solution s'il y en a une.

Un exemple de pièces est donné dans la figure suivante. Les deux pièces peuvent être placées côte à côte puisque les deux côtés ont la même couleur 4.



Votre programme sera constitué de deux parties.

Librairie. Vous écrirez une librairie de résolution de ce puzzle, qui fournit principalement des fonctions pour créer des instances aléatoires et des instances aléatoires qui admettent une solution, une fonction de recherche de solution (par recherche exhaustive). Les données du problème (m, n, c) ne sont bien sûr pas fixées. La librairie doit également permettre à l'utilisateur de fixer une pièce à une cellule du tableau, et chercher les solutions sous cette contrainte. La librairie doit être capables d'enregistrer une instance du jeu (la liste des pièces) dans un fichier, dans le format décrit en annexe, et de la relire.

Interface. Vous écrirez une interface graphique (a priori simple) qui, en utilisant votre librairie, permettra de résoudre ce problème. Elle permettra par exemple de générer une instance et d'en lire une dans un fichier, et affichera les pièces générées, et affichera une solution une fois que celle-là est calculée. Vous pouvez aussi permettre à l'utilisateur de fixer les pièces dans le tableau. Vous êtes libres d'utiliser le module Graphics standard de OCaml, ou bien Tk ou encore lablgtk, au choix. Faites attention par contre à ne pas vous lancer dans une librairie que vous connaissez mal, et perdre du temps.

L'objectif principal de ce projet est de concevoir une librairie et un programme propres, écrits de manière claire et réutilisable, tout en cachant tout ce qui relève de details d'implémentation.

Par exemple, la librairie doit avoir une interface (via un module ou une classe) abstraite et limpide. Je devrais être capable de comprendre votre interface et de l'utiliser rapidement en regardant les fichiers `.mli`, sans me plonger dans votre code (fichiers `.ml`). L'interface graphique est une démonstration de l'utilisation de votre librairie, et elle a aussi pour fonction de me convaincre que vous avez bien conçu votre librairie. Notez qu'il ne s'agit pas d'un projet de *hacking* qui vise à résoudre ce problème d'algorithmique le plus efficacement possible. C'est tout d'abord l'organisation du programme qui comptera dans l'évaluation de votre projet. Un effort sur l'algorithmique sera néanmoins apprécié (voir plus bas).

Vous devez fournir avec votre programme une documentation de la librairie générée par `ocamldoc`[1] (ou un autre si vous êtes adeptes d'autres systèmes de documentation).

Votre programme doit comporter aussi un fichier LISEZMOI qui explique comment compiler et comment exécuter le programme, et d'autres informations que vous jugerez utiles.

Vous vous assurerez de la correction de votre librairie par des tests systématiques. Ainsi, votre programme doit comporter une fonction qui génère des entrées aléatoirement, résout, et vérifie la solution est correcte (que les couleurs correspondent et que chaque pièce apparaît le bon nombre de fois). Votre fichier LISEZMOI doit expliquer comment (simplement) lancer un tel test.

Vous m'enverrez votre projet par mail dans une archive (tgz, zip, ...) avant le **16 Avril 2012 minuit**.

Compétition. J'ai décrit en haut dans ce document les fonctionnalités minimales qui sont demandées. Il y a bien sûr beaucoup d'améliorations possibles en algorithmique et en interface. Ainsi, je vous propose deux prix spéciaux:

Prix spécial performance. J'évaluerai la performance en temps de tous les programmes que j'aurai reçu sur les mêmes instances, générées aléatoirement. Le *hackeur* qui aura programmé la librairie la plus performante aura la gloire de remporter le prix spécial performance! Pour être elligible, vous devez fournir dans votre projet un petit programme `solve.ml`, séparé du reste, qui lit une instance du problème dans l'entrée standard, résoud l'instance, et imprime une solution dans la sortie standard (voir l'annexe pour le format). Chaque programme sera compilé avec le compilateur natif (`ocamlopt`) et sera executé sur la même machine.

Notez que l'implémentation des optimisations multiples et compliquées n'est pas un prétexte pour rendre votre code illisible. Vous soignerez toujours votre programme.

Prix spécial interface graphique La plus belle interface graphique utilisateur remportera ce prix. Je ne sais pas encore comment évaluer. Mais on peut faire une vote.

NB Il doit être clair que la note attribuée à votre projet n'est pas forcément liée par exemple à la performance de votre programme ni à la qualité du graphisme. Vous donnerez la priorité à l'organisation de votre code, comme décrit en haut. Assurez-vous de terminer le projet avant de commencer ces améliorations.

2 Annexe

Le format de fichier pour une instance du problème est comme suit. La première ligne contient 3 entiers m n c, séparés par des espaces simples. Ici m est le nombre de lignes, et n le nombre de colonnes du tableau. Ensuite, le fichier contient mn lignes, qui correspondent aux pièces. Chaque pièce est décrite par les 4 couleurs a b c d appartenant à $\{1, \ldots, c\}$, enumérées dans le sens direct en commençant par le côté de droite. Par exemple, la première pièce est décrite par 4 1 4 2 et la deuxième par 1 2 4 3.

[1] C'est facile, voir la documentation.

Un exemple d'entrée avec ces deux pièces:

```
1 2 4
4 1 4 2
1 2 4 3
```

Le format de sortie d'une solution calculée est la même, sauf que les pièces sont ordonnées ligne par ligne, c-à-d, les premières n pièces sont celles qui se trouvent, dans la solution, dans la première ligne du tableau (ordonnées de gauche à droite), les n pièces suivantes correspondent à la deuxième ligne, et ainsi de suite.