projet INF441

Les liens dansants

Sujet proposé par Jean-Christophe Filliâtre

24 avril 2015

Rien n'est plus excitant en informatique que d'écrire un programme qui calcule en quelques secondes un résultat qu'on ne pourrait obtenir sans l'aide d'un ordinateur. La combinatoire est un domaine particulièrement fertile en la matière. Si certains problèmes se prêtent volontiers à l'analyse mathématique, comme le nombre total de livres dans la bibliothèque de Borgès ou encore le nombre de labyrinthes parfaits rectangulaires, beaucoup d'autres exigent de recourir à la programmation. Dans ce contexte, ce projet a pour but de mettre en œuvre un algorithme particulier, dit des liens dansants, pour résoudre des problèmes de pavage.

1 La couverture exacte de matrice

Le problème de la *couverture exacte de matrice*, désigné par EMC par la suite, est le suivant : étant donnée une matrice contenant uniquement des 0 et des 1, il s'agit de déterminer un sous-ensemble de ses lignes contenant un 1 et un seul par colonne. Étant donné un problème EMC, on peut s'intéresser uniquement à la question de savoir s'il possède une solution, au problème d'en construire une le cas échéant, au problème de dénombrer les solutions, ou encore au problème de les construire toutes. Ainsi le problème EMC suivant

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

possède exactement deux solutions, à savoir les sous-ensembles de lignes $\{0, 4\}$ et $\{1, 3\}$.

Une légère variante d'EMC, qui nous sera utile par la suite, distingue deux sortes de colonnes dans la matrice : des colonnes dites *primaires*, qui doivent nécessairement être couvertes, et des colonnes dites *secondaires*, qui peuvent ne pas être couvertes.

L'intérêt d'EMC est qu'il est très facile d'y encoder de nombreux problèmes. Ainsi, nous montrerons plus loin dans la section 3 comment réduire un problème de pavage au problème EMC.

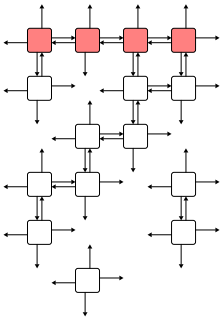
2 Les liens dansants

Donald Knuth a proposé en 2000 un algorithme appelé *dancing links* [2] pour résoudre le problème EMC. Cet algorithme applique la technique du *backtracking* pour découvrir toutes les solutions, une par une. Son intérêt vient de la très grande économie de moyens mis en œuvre pour y parvenir.

L'algorithme des liens dansants, appelé DLX par la suite, consiste à relier ensemble tous les 1 de la matrice EMC dans un réseau de listes circulaires doublement chaînées horizontales et verticales, puis à ôter successivement les éléments de ces listes, au fur et à mesure des choix qui sont faits, et à les réinsérer ensuite. L'efficacité de cet algorithme est due notamment à la possibilité de réinsérer un élément qui vient d'être supprimé d'une liste doublement chaînée sans utiliser d'autre information que celle qui se trouve déjà dans les deux pointeurs vers ses anciens voisins.

Si on reprend le problème EMC donné en introduction, l'algorithme DLX va construire l'ensemble de listes doublement chaînées suivant :

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$



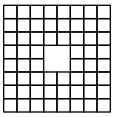
Chaque 1 de la matrice est matérialisé par un nœud de liste chaînée (carré blanc). Par ailleurs, chaque colonne de la matrice contient un nœud supplémentaire, appelé entête (carré rouge). Les entêtes sont également liés entre eux dans une liste chaînée circulaire.

L'algorithme DLX va alors chercher à couvrir les colonnes de la matrice. Pour cela, il choisit une colonne, par exemple la première, et examine toutes les façons de la couvrir, en parcourant la liste chaînée verticale de cette colonne. Pour chaque élément, il supprime alors de la matrice les lignes qui entrent en conflit avec ce choix. Si par exemple on choisit la première ligne pour couvrir la première colonne, alors il faut supprimer les trois lignes suivantes. Par ailleurs, on supprime les colonnes couvertes par le choix de la première ligne. Il ne reste alors plus que la seconde colonne à couvrir et seulement la dernière ligne de la matrice. On applique de nouveau l'algorithme, ce qui donne ici immédiatement une première solution. Puis en revient en arrière pour effectuer les autres choix possibles à chaque étape. Sur cet exemple, on trouvera exactement deux solutions, comme indiqué dans l'introduction.

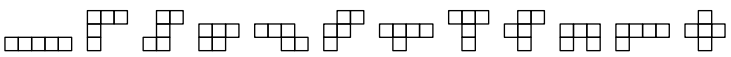
3 Application au problème du pavage

Considérons le problème du pavage d'une surface constituée de plusieurs cellules par un ensemble de pièces. On peut le réduire facilement au problème EMC en construisant une matrice où chaque colonne représente une cellule à paver et chaque ligne une façon de poser une pièce. Par définition, une solution au problème EMC aura couvert chaque cellule une fois et une seule.

Donnons un exemple avec le problème de Scott [3]. Il s'agit de paver un échiquier évidé en son centre



avec les douze pentaminos, c'est-à-dire les douze pièces suivantes :



Chaque pièce doit être utilisée une et une seule fois, mais peut être tournée ou retournée à volonté. On construit donc une matrice avec 72 colonnes : 60 colonnes pour représenter les 60 cases à paver de l'échiquier et 12 colonnes pour représenter les 12 pentaminos. Chaque ligne de la matrice correspond à une façon de poser un pentamino sur l'échiquier : un 1 dans chaque colonne représentant une case couverte et un 1 dans la colonne représentant ce pentamino. Au total, il y a 1400 lignes dans la matrice. Les 12 colonnes représentant les différents pentaminos sont là pour garantir que chaque pièce est utilisée une fois et une seule.

L'algorithme DLX permet alors de déterminer rapidement les 520 solutions. On peut facilement éliminer les symétries du problème en ne considérant qu'une orientation possible pour l'une des pièces n'ayant aucune symétrie (par exemple la pièce ⊞). On trouve alors 65 solutions uniques.

4 Travail demandé

Le projet pourra être réalisé en Java ou en OCaml, au choix. Il est demandé de réaliser au minimum les éléments suivants :

l'algorithme DLX On commencera par lire soigneusement l'article de Knuth [2]. La réalisation de l'algorithme DLX devra permettre de parcourir toutes les solutions avec un *itérateur*. On déduira de cet itérateur des fonctions pour dénombrer toutes les solutions, trouver une solution particulière lorsqu'elle existe et renvoyer toutes les solutions.

pavage Le programme devra permettre de décrire un problème de pavage sur une grille 2D, de dénombrer ses solutions et d'afficher une solution, le cas échéant.

On prendra un soin particulier en ce qui concerne la modularité du code, avec une séparation claire entre l'algorithme DLX et son application au problème du pavage.

Instructions. Vous devez soumettre votre projet sous la forme d'une archive au format `.zip` ou `.tar.gz` telle que, une fois l'archive décompressée, on obtient un fichier `rapport.pdf` et un sous-répertoire `src` contenant **tous** les fichiers source Java ou OCaml. (Donc, ne créez pas de sous-répertoires dans `src`.) Nommez le fichier principal `Main.java` ou `main.ml`. Votre projet sera compilé comme suit :

```
cd src
javac *.java           # pour Java
ocamlbuild main.native # pour OCaml
```

Il sera exécuté comme suit :

```
cd src
java -ea Main <argument> # pour Java
./main.native <argument> # pour OCaml
```

Votre programme recevra exactement un argument sur la ligne de commande, à savoir `emc` ou `pavage`.

Argument `emc`. Si l'argument est `emc`, votre programme devra lire sur l'entrée standard un problème EMC sous la forme suivante :

nombre de colonnes primaires
nombre de colonnes secondaires
nombres de lignes
ligne 1
ligne 2
etc.

Par exemple, le problème donné en exemple dans ce sujet sera passé à votre programme sous la forme suivante :

```
4
0
5
1011
0110
1101
1001
0100
```

Votre programme devra alors afficher le nombre total de solutions, ici 2 sur cet exemple.

Argument `pavage`. Si l'argument est `pavage`, votre programme devra lire sur l'entrée standard un problème de pavage 2D sous la forme suivante :

nombre de colonnes C
nombre de lignes L
ligne 1
...
ligne L
nombre de pièces P
pièce 1
...
pièce P

Chaque pièce est elle-même décrite par plusieurs lignes, sous la forme suivante :

nombre de colonnes
nombre de lignes
ligne 1
etc.

Chaque ligne décrivant la figure à paver ou une pièce à poser contient un caractère `*` pour dénoter une case significative et tout autre caractère sinon. Chaque pièce doit être utilisée exactement une fois et peut être tournée et retournée arbitrairement. Par exemple, le problème de Scott décrit plus haut peut être représenté par le texte donné en figure 1. Votre programme devra alors afficher le nombre total de solutions, ici 520 sur cet exemple.

Des fichiers de tests pour seront fournis sur la page de suivi du projet [1]. Votre programme sera testé sur un plus large jeu de tests.

Pour aller plus loin (bonus). Si le temps et l'envie le permettent, on pourra considérer les extensions suivantes :

- afficher une solution, lorsqu'elle existe, sur la sortie standard ou dans une fenêtre graphique ;
- exploiter les symétries du problème de pavage, le cas échéant ;
- chercher à traduire d'autres problèmes vers EMC, comme par exemple le problème des N-reines (voir [2]).

5 Conseils généraux

Utilisez un système de contrôle de versions pour gérer l'évolution de votre travail et l'interaction entre vous si vous travaillez en binôme. `git`, par exemple, est gratuit et très bien conçu. Si vous n'avez jamais utilisé un tel outil, ce projet est une bonne occasion d'apprendre.

Privilégiez la correction vis-à-vis de la performance. L'efficacité ne peut être que secondaire vis-à-vis de la correction : en effet, l'efficacité d'un programme incorrect n'a aucun sens. En particulier, vous pouvez insérer des assertions `assert (...)` dans votre code, qui seront vérifiées pendant l'exécution.

N'optimisez pas prématurément. Ne cédez donc pas à la tentation de l'optimisation prématurée. Accordez la priorité à la clarté de votre code. Faites tout pour qu'il soit facile à modifier, de façon à pouvoir expérimenter rapidement de nouvelles idées.

Testez. Un jeu de tests sera publié sur la page de suivi du projet [1]. N'hésitez pas pour autant à écrire vos propres tests.

Commencez tôt, car un projet prend toujours plus de temps que ce que l'on croît.

Lors de l'exposé, essayez dans la mesure du possible de ne pas répéter ce qui est déjà contenu dans ce sujet. Expliquez plutôt vos contributions.

Références

[1] Jean-Christophe Filliâtre. Page de suivi. `http://www.enseignement.polytechnique.fr/informatique/INF441/projets/dlx/`

[2] Donald E. Knuth. Dancing links. Millenial Perspectives in Computer Science, 2000. `http://arxiv.org/abs/cs/0011047`.

[3] Dana S. Scott. Programming a combinatorial puzzle. Technical Report 1, Department of Electrical Engineering, Princeton University, 1958.

```
8
8
********
********
********
***..***
***..***
********
********
********
12
5
1
*****
3
3
***
*..
*..
3
3
.**
.*.
**.
3
2
***
**.
4
2
***.
..**
```

```
3
3
.**
**.
*..
4 2
****
.*..
3
3
***
.*.
.*.
3
3
.**
**.
.*.
3
2
***
*.*
4
2
****
*...
3
3
.*.
***
.*.
```

FIGURE 1 – Le problème de Scott.