

★ Modalités

Ce projet est à réaliser en binôme. Vous indiquerez la composition de votre binôme dans un fichier dénommé **AUTHORS** (placé à la racine du répertoire de votre projet) qui doit contenir les logins UNIX des membres du groupe, un par ligne.

Travail à rendre : Vous devez rendre un mini-rapport de projet (5 pages maximum, format pdf) en plus de vos fichiers sources. Vous y détaillerez les difficultés auxquelles vous avez été confronté, et comment vous les avez résolues. Vous indiquerez également le nombre d'heures passées sur les différentes étapes de ce projet (conception, codage, tests, rédaction du rapport) par chaque membre du groupe.

Comment rendre votre projet : Vous déposerez sur ARCHE une archive (.zip ou .tar.gz) contenant le code source de votre projet ainsi que votre rapport.

<http://arche.univ-lorraine.fr/mod/assign/view.php?id=176315>

Avant le 27 Février 2014, à 23h55.

(aucune soumission en retard ne sera acceptée)

Évaluation : Un projet ne compilant pas sera sanctionné par une note adéquate. Aucune soumission par email ne sera acceptée. Des soutenances individuelles de projet seront organisées à une date ultérieure. Vous serez jugé sur la qualité de votre programme, celle de votre rapport et votre capacité à expliquer son fonctionnement.

★ Travail personnel et honnêteté

Ne trichez pas ! Ne copiez pas ! Si vous le faites, vous serez lourdement sanctionnés. Nous ne ferons pas de distinction entre copieur et copié. Vous n'avez pas de (bonne) raison de copier. En cas de problème, nous sommes prêt à vous aider. Encore une fois : en cas de doute, envoyez un courriel à vos enseignants, ça ne les dérange pas.

Par tricher, nous entendons notamment :

- Rendre le travail d'un collègue avec votre nom dessus ;
- Obtenir une réponse par GoogleTM ou autre et mettre votre nom dessus ;
- Récupérer du code et ne changer que les noms de variables et fonctions ou leur ordre avant de mettre votre nom dessus (*"moving chunks of code around is like moving food around on your plate to disguise the fact that you haven't eaten all your brussel sprouts"*) ;
- Permettre à un collègue de *s'inspirer* de votre travail. Assurez vous que votre répertoire de travail n'est lisible que par vous même.

Il est plus que très probable que nous détectons les tricheries. Chacun a son propre style de programmation, et personne ne code la même chose de la même manière. De plus, il existe des programmes très efficaces pour détecter les similarités douteuses entre copies (MOSS, <http://theory.stanford.edu/~aiken/moss/>).

En revanche, il est possible (voire conseillé) de discuter du projet et d'échanger des idées avec vos collègues. Mais vous ne pouvez rendre que du code écrit par vous-même. Vous indiquerez dans votre rapport toutes vos sources d'inspiration (comme les sites internet de vulgarisation de l'informatique que vous auriez consulté).

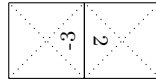
★ Bibliographie

Le problème présenté a été proposé par Stephen Weiss en 2003 en tant que *Nifty Assignment* au groupe d'intérêt *ACM Special Interest Group on Computer Science Education (SIGCSE)*. Le sujet original peut être consulté à l'adresse suivante : <http://nifty.stanford.edu/2003/backtracking/>.

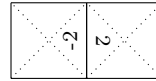
La documentation de l'API Java : <http://docs.oracle.com/javase/7/docs/api/>.

★ Présentation du problème : 9-square puzzle

Imprimer cette page et découper les neuf carrés. Ensuite, essayer d'arranger ces carrés en une grille 3×3 tels que pour deux carrés adjacents la somme des valeurs côte-à-côte soit égale à 0. Bien entendu vous êtes autorisés à faire pivoter les carrés.

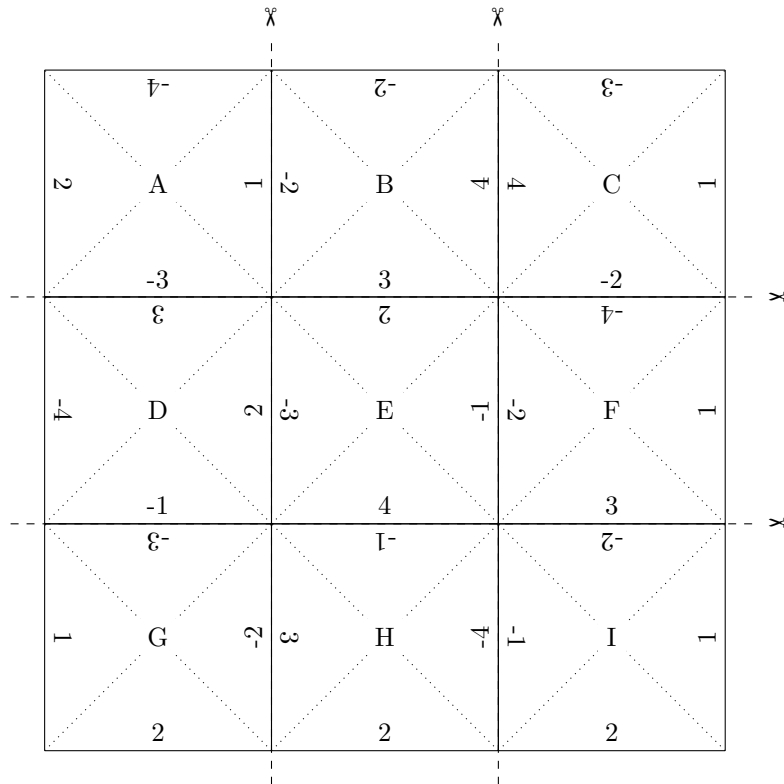


invalid



valide

Sachez qu'il y a $4^9 \times 9! = 95\,126\,814\,720$ arrangements possibles, et que l'instance du problème donnée ci-dessous n'admet que 4 solutions (en réalité 4 rotations de la même solution).



Remarque : Un puzzle est dit *parfait* si chaque pièce contient une et une seule fois les nombres 1, 2, 3 et 4 dont deux sont positifs et deux sont négatifs. Le puzzle donné en exemple n'est donc pas parfait.

★ Travail à réaliser

L'objectif de ce mini-projet de réaliser un programme reposant sur un algorithme récursif avec retour arrière (*backtracking*) qui calcule les solutions à une instance du problème présenté précédemment.

Votre programme devra indiquer le nom du puzzle, si ce puzzle est parfait ou non. Il devra afficher le nombre de solutions trouvées ainsi que le détail de ces solutions. Il devra également afficher le temps de calcul total et d'autres statistiques que vous jugerez intéressantes (nombre d'appels récursifs, nombre de solutions explorées ou rejetées, ...).

Votre programme devra être capable de lire un fichier contenant la description de l'instance du problème à résoudre. Ce fichier respectera le format suivant :

```

data/data1.txt
1 Puzzle 1 (4 solutions)
2 A -4 1 -3 2
3 B -2 4 3 -2
4 C -3 1 -2 4
5 D 3 2 -1 -4
6 E 2 -1 4 -3
7 F -4 1 3 -2
8 G -3 -2 2 1
9 H -1 -4 2 3
10 I -2 1 2 -1

```

- La première ligne contient un titre,
- les neuf lignes suivantes représentent chacune une pièce.
 - Une étiquette (un caractère),
 - les 4 valeurs de la pièce classées dans l'ordre des aiguilles d'une montre en commençant par la valeur du haut de la pièce.

Vous trouverez sur ARCHE (<http://arche.univ-lorraine.fr/course/view.php?id=7973>), différents fichiers contenant les différentes instances du problème que votre programme doit être capable de résoudre.

★ Bonus (tâches facultatives)

- Généraliser votre programme pour résoudre des problèmes dont la taille du plateau est variable et non plus seulement limitée à 3×3 pièces.
- Généraliser votre programme pour résoudre ce problème avec des pièces en forme de cube et non plus des pièces carrées.

★ Indications

Dans ce projet vous devez mettre en applications les connaissances que vous avez acquises dans le module POO ainsi que dans le module TOP.

Vous devrez donc réaliser un programme en respectant les principes de la programmation objet. Il vous est donc fortement conseillé de chercher à modéliser le problème en manipulant des classes et des objets. Par exemple, vous serez amenés à définir une classe pour le programme principal, une pour une pièce, une pour le plateau de jeu (qui peut représenter soit une solution trouvée, soit une solution en cours de construction), une pour représenter le pot des pièces non placées, ...

Afin de vous aider, vous trouverez ci-dessous une proposition de description sous forme de classes du programme. Vous êtes libres d'en proposer/utiliser une autre.

★ Classe/Responsabilités/Collaborations

Classe : <code>puzzle.Piece</code>	
Responsabilités : <ul style="list-style-type: none"> - Conserve les 4 valeurs et l'étiquette d'une pièce carrée du puzzle. - Gère l'affichage d'une pièce. - Gère les rotations d'une pièce. - Détermine si une pièce est <i>parfaite</i> ou non. 	Collaborateurs :

Classe : <code>puzzle.Pool</code>	
Responsabilités : <ul style="list-style-type: none"> - Conserve toutes les pièces d'un puzzle. - Gère la disponibilité des pièces (une pièce ne peut être utilisée qu'une seule fois dans un plateau). - Gère le chargement d'une liste de pièces depuis une sous-partie d'un fichier formaté. - Détermine si l'ensemble des pièces est <i>parfait</i> ou non. 	Collaborateurs : <ul style="list-style-type: none"> - <code>puzzle.Piece</code>

Classe : <code>puzzle.Board</code>	
Responsabilités : <ul style="list-style-type: none"> - Gère le placement d'une pièce sur un plateau tout en respectant les règles du puzzle. - Gère l'affichage d'un plateau. 	Collaborateurs : <ul style="list-style-type: none"> - <code>puzzle.Piece</code>

Classe : <code>puzzle.NineSquarePuzzle</code>	
Responsabilités : <ul style="list-style-type: none"> - Détermine si une instance du puzzle est <i>parfaite</i> ou non. - Résout une instance du puzzle. - Conserve les solutions trouvées. - Gère le chargement d'une instance d'un puzzle à partir d'un fichier formaté. 	Collaborateurs : <ul style="list-style-type: none"> - <code>puzzle.Pool</code> - <code>puzzle.Board</code> - <code>puzzle.Piece</code>

Classe : <code>puzzle.Main</code>	
Responsabilités : <ul style="list-style-type: none"> - Interagit avec l'utilisateur (affichage, demande de saisie du fichier de données à charger, affichage des statistiques, ...). - Crée une instance du problème (<code>NineSquarePuzzle</code>) et demande la résolution de cette instance. 	Collaborateurs : <ul style="list-style-type: none"> - <code>puzzle.NineSquarePuzzle</code> - <code>puzzle.Instrumentations</code>

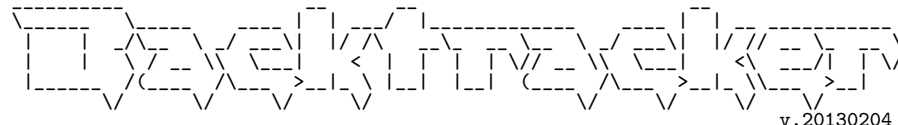
Classe : <code>puzzle.Instrumentations</code>	
Responsabilités : <ul style="list-style-type: none"> - Collecte les différentes statistiques (temps d'exécution, nombre d'appels de méthode, ...). - Gère l'affichage des statistiques collectées. 	Collaborateurs :

Étant donné que le nombre d'arrangements possibles est très élevé (> 95 milliards de possibilités), il est important de bien mettre en œuvre le *backtracking* et surtout de limiter au maximum la recherche aux solutions potentielles (élaguer au plus tôt les branches menant des instances non valides).

★ Cas d'utilisation

Vous trouverez ci-dessous à titre d'exemple, l'affichage donné lors l'utilisation du programme que vous devez réaliser.

```

...

v.20130204

Veuillez saisir le nom du fichier de données (ou q pour quitter) -> ../../data/data1.txt
Le puzzle "Puzzle 1 (4 solutions)" (../../data/data1.txt) n'est pas parfait.
Il contient les 9 pièces suivantes :

+-----+
|      |
| -4    |
| 2 [A0] 1 |
| -3    |
+-----+
+-----+
|      |
| -2    |
| -2 [B0] 4 |
| 3     |
+-----+

[...]

+-----+
|      |
| -2    |
| -1 [I0] 1 |
| 2     |
+-----+

Calcul des solutions en cours...

Solution 1:
+-----+
|      |      |      |
| 1     | -2    | 3     |
| -4 [A3] -3 | 3 [B1] -2 | 2 [H1] -1 |
| 2     | 4     | -4    |
+-----+
+-----+
|      |      |      |
| -2    | -4    | 4     |
| -3 [G3] 2 | -2 [F0] 1 | -1 [E2] -3 |
| 1     | 3     | 2     |
+-----+
+-----+
|      |      |      |
| -1    | -3    | -2    |
| 2 [D2] -4 | 4 [C0] 1 | -1 [I0] 1 |
| 3     | -2    | 2     |
+-----+

[...]

Solution 4:
+-----+
|      |      |      |
| 2     | -2    | 3     |
| 1 [I2] -1 | 1 [C2] 4 | -4 [D0] 2 |
| -2    | -3    | -1    |
+-----+
+-----+
|      |      |      |
| 2     | 3     | 1     |
| -3 [E0] -1 | 1 [F2] -2 | 2 [G1] -3 |
| 4     | -4    | -2    |
+-----+
+-----+
|      |      |      |
| -4    | 4     | 2     |
| -1 [H3] 2 | -2 [B3] 3 | -3 [A1] -4 |
| 3     | -2    | 1     |
+-----+

Temps de calcul estimé 00:00:00.036 pour trouver 4 solution(s)

-- Statistics
Recursive calls = 4 312
Pieces tried = 81 192
Pieces tried and placed = 4 311

```