TP3. Arbres implicites

1 Sudoku

Vous trouverez sur e-campus des fichiers vous permettant de générer et de manipuler des sudokus. Créez une nouvelle classe Solveur qui va permettre d'exécuter les actions suivantes sur le code.

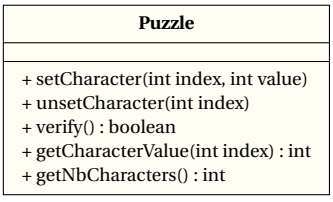
1. Écrivez une méthode [1] qui vérifie qu'un sudoku est valide (c'est à dire qu'aucune valeur n'est présente deux fois dans la même ligne, colonne ou block). On souhaite que votre méthode soit utilisable aussi sur un sudoku partiellement rempli.
   Vous pouvez utiliser les méthodes `getLine`, `getColumn` et `getBlock` de la classe Sudoku pour accéder de la même façon aux cases d'une ligne, colonne ou block.
2. Écrivez une méthode de backtrack qui cherche de façon exhaustive s'il existe une solution au sudoku.
   Pour vous simplifier la tâche, vous pouvez utiliser la méthode `firstEmptySquare` de la classe Sudoku, a condition d'utiliser aussi les méthodes de Sudoku pour faire les `set(Square, int)` et `unset(Square)`.
3. Si votre méthode est trop lente, il est probable qu'il faille réduire un peu la combinatoire en évitant d'explorer des situations invalides (en utilisant la méthode écrite à la question 1).
4. Votre méthode permet-elle de récupérer la solution trouvée ? Si non, adaptez-la pour que cela soit fait. Ne remplit-elle pas de toutes façons le sudoku par *effet de bord* ?
5. Faites une autre méthode qui compte le nombre de solutions à un sudoku.

2 Cryptarythmes

Un cryptarythme est un casse tête où les chiffres d'une opération posée sont représentée par des lettres. Un exemple classique est représenté ci-contre, mais il en existe de nombreux autres exemples. Le principe est que chaque lettre représente un chiffre distinct, et que l'opération doit être valide si on remplace les lettres par les bonnes valeurs.

		S	E	N	D
$+$		M	O	R	E
	M	O	N	E	Y

On souhaite faire un outil de résolution automatique de ce type de problèmes. Tout le modèle est déjà fait, il reste juste à écrire l'algorithme. Pour simplifier, les caractères sont désignés par leurs indices. On a donc une méthode `setCharacter(int index, int value)` qui permet de définir une valeur (entre 0 et 9) au caractère d'indice *index*, une méthode `unsetCharacter(int index)` pour retirer la valeur du caractère d'indice `index`. La fonction `verify()` permet de vérifier le résultat de l'opération quand tous les caractères sont affectés.



Pour simplifier, dans les deux questions suivantes, on ne vérifiera pas que les valeurs sont toutes distinctes, et on autorisera donc que plusieurs lettres modélisent le même chiffre.

6. Complétez la fonction `solve(Puzzle problem)` qui teste l'ensemble des valeurs possibles sur le puzzle en paramètre jusqu'à trouver une solution. La fonction laissera les valeurs affectées quand le problème est réussi et retournera `true`, mais retournera `false` si aucune solution n'est trouvée.
7. Complétez la fonction `nbSolutions(Puzzle problem)` afin qu'elle compte le nombre total de solutions distinctes du problème.

1. pas nécessairement récursive

3 Le cavalier.

Dans cette section, nous nous posons la question des possibilités de déplacement du cavalier (des échecs). Pour rappel, le cavalier se déplace de deux cases dans une direction et une case dans la direction orthogonale. Depuis la case $(0,0)$, il atteint $(1,2)$ ou $(2,1)$.

Nous allons essayer de trouver pour quelles tailles de plateau le cavalier peut atteindre toutes les cases. (Caché derrière cette question, il y a bien sûr un arbre, lequel ?)

8. Créez une classe d'objets Plateau qui s'instancient avec un entier n et contient un plateau d'entiers $n \times n$. Ajoutez une méthode pour afficher le plateau (`toString`).
9. Écrivez une méthode récursive qui prend en entrée une position initiale d'un cavalier et marque d'un 1 toutes les cases du plateau accessibles par le cavalier après un ou plusieurs coups. Bien sûr, le cavalier peut décrire des cycles sur un plateau, comment allez vous vous assurer que la méthode s'arrête ?
10. Pour quelles tailles de plateau atteignez vous toutes les cases ?
11. On peut aussi se demander combien de coups il faut au cavalier pour atteindre une case donnée. Écrivez une méthode récursive qui avec pour entrée une position initiale du cavalier indique pour chaque case le nombre minimal de déplacements nécessaires pour l'atteindre. (On pourra utiliser 0 pour la position initiale et -1 pour les cases inaccessibles).
12. Quel est votre avis sur ce que donnerait la question 4 avec un parcours en largeur ?

4 Minimax (*)

Vous trouverez une implémentation vous permettant de manipuler des plateaux de Morpion partiellement remplis.

13. (*) Écrivez une méthode qui prend en entrée un plateau de Morpion, et un joueur (représenté par `Content.CROSS` ou `Content.CIRCLE`) et renvoie 1 si le joueur indiqué peut gagner à coup sûr quand il commence, -1 si son adversaire peut, 0 si le meilleur résultat que chacun peut atteindre est un match nul.
    Des méthode de test vous sont fournis pour vous assurer que vous n'avez pas d'erreur dans votre code.
14. (***) Améliorez votre code selon le principe du alpha-bêta, pour ne pas explorer des branches inutiles.