Fiche de TP5 Résolution du jeu du Taquin par un algorithme récursif et un algorithme de type séparation et évaluation

Contenus et objectifs

Expérimenter la résolution d'un problème NP-difficile par 2 techniques de diviser-pour-règner et toucher du doigt l'impact de l'explosion combinatoire sur les temps de calcul.
ATTENTION : Pensez à commenter vos programmes Java. Pour le rendu, archivez dans un fichier `Gx_TP5.tgz` ou `Gx_TP5.zip` tous vos fichiers `.java` et téléverser le fichier archive compressé sur Chamilo dans le groupe x dédié.

Pour ce TP, vous disposez d'un ensemble de 10 instances de test dans le fichier `Taquin_tests.zip`

Chaque fichier instance donne le taille N du taquin de taille NxN, ainsi que les configurations initiale et finale. Les configurations sont solubles. Comme vu en cours, nous souhaitons rechercher ici **la séquence de mouvements de tuile minimale** qui permet de transformer la configuration initiale en la configuration finale tout en respectant les contraintes entre tuiles et les limites du plateau de jeu.

Pour toutes classes et méthodes demandées ci-dessous, les noms sont imposés, mais vous êtes libres de définir leurs signatures (types et paramètres d'entrée, types de retour).

A. Implémenter la classe `Plateau`

1.1. Ecrivez la classe JAVA `Plateau` (avec tous les attributs et constructeurs/méthodes nécessaires) qui permet d'initier le plateau de jeu à partir de fichiers d'instance fournis. Vous avez le choix de la structure de données pour stocker et gérer le plateau de jeu.

- Pensez à définir des attributs permettant de gérer la séquence de mouvements de tuile minimale trouvée.
- Pensez à ajouter une méthode qui affiche la configuration courante du plateau `affichePlateau()`.

1.2. Ajouter dans cette classe `Plateau` des méthodes qui permettent de déplacer les tuiles (haut, bas, gauche, droite) : `dHaut(), dBas(), dGauche(), dDroite()`.

1.3. Ajouter dans cette classe `Plateau` une méthode qui permet vérifier si le plateau est dans l'état résolu, à savoir que la configuration courante correspond à la configuration finale donnée dans le fichier de test : `estResolu()`

B. Résolution par récursion

Dans une classe `TaquinSolveur`, implémenter une méthode de résolution `solRec()` fondée sur la récursivité et qui exploite la classe `Plateau` pour résoudre un taquin donné par un fichier de test.

Une fois la résolution faite, vous devez être en mesure d'afficher dans un fichier de sortie : la séquence de mouvements de tuile minimale, le nombre d'appels récursifs effectués pour cette résolution et le temps de calcul.

Question bonus : Implémenter une méthode de résolution `solRecM()` qui exploiterait l'usage de la fonction de distance de Manhattan h() afin de réduire le nombre d'appels récursifs.

C. Résolution par séparation et évaluation

Implémenter une autre méthode de résolution `solSE()` fondée sur la technique de séparation et évaluation et qui exploite la classe `Plateau` pour résoudre un taquin donné par un fichier de test. Vous devez exploiter la fonction de distance de Manhattan h() afin d'effectuer un parcours « meilleurs d'abord » des configurations.

Pour cette méthode de résolution, nous devez impérativement réutiliser un arbre AVL ou un Tas (vus dans les TP 3 et 4) pour gérer la file de priorité. Vous ne pouvez pas utiliser celle proposée par JAVA pour ce TP.

Une fois la résolution faite, vous devez être en mesure d'afficher dans un fichier de sortie : la séquence de mouvements de tuile minimale, le nombre de configurations visitées pour cette résolution et le temps de calcul. Comparez ces résultats à ceux de la résolution par récursion.

Question bonus : changer le parcours « meilleurs d'abord » des configurations en « profondeur d'abord », puis en « largeur d'abord ». Comparer ces résultats avec ceux obtenus avec le parcours « meilleurs d'abord » et avec ceux de la ou des 2 récursion.s.