Licence Informatique 2e année — Université d'Orléans
UE Projet informatique — 2014-2015

Jeu de taquin

I. Préambule

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

10	6	4	12
1	14	3	7
5	15	11	13
8		2	9

Figure 1: Notre position idéale et une position engendrée aléatoirement

Le jeu de taquin (Fifteen Puzzle) est constitué d'un plateau carré, subdivisé en seize cases, sur lequel on pose quinze pièces carrées numérotées de un à quinze. Il reste donc une case libre ou trou. Le seul mouvement ou coup autorisé consiste à faire glisser l'une des pièces adjacentes au trou vers celui-ci, ce qui revient à échanger leurs positions respectives.

Ce jeu, imaginé par Sam Loyd dans les années 1870, suscita immédiatement un grand intérêt à travers l'Occident. L'une des raisons peut-être de ce rapide succès était une récompense de mille dollars promise par Loyd à quiconque parviendrait à transformer une position de départ en une position d'arrivée fixée.

De façon relativement abstraite, on peut voir l'espace des configurations du jeu de taquin comme un graphe où chaque sommet représente une configuration et où les successeurs d'un sommet sont les configurations obtenues grâce à un coup quelconque. Chaque sommet a donc deux, trois ou quatre successeurs. Le jeu consiste alors à trouver un chemin à travers ce graphe entre un sommet de départ et un sommet d'arrivée fixés.

Loyd était taquin: les positions de départ et d'arrivée qu'il avait fixées ne sont en fait reliées par aucun chemin dans le graphe des configurations. Ce fait a été publié dès 1879, et se démontre aujourd'hui facilement. Dans le cadre de ce projet, la position de départ sera variable, et la position d'arrivée sera celle illustrée à gauche de la figure 1.

II. Le sujet

Réaliser un programme en java calculant de différentes manières une solution d'un jeu de taquin. Votre programme permettra:

- de lire une position initiale,
- de tester si le problème est soluble,
- de jouer en mode interactif sur un terminal,
- de calculer une solution et de la présenter sous la forme d'une animation,
- de comparer et d'évaluer les différentes méthodes proposées.

Pour candidater à note supérieur 16/20, votre système devra mettre en œuvre tous les méthodes proposées et tout particulièrement la méthode d'élagage. Un bonus de 3 points, vous est proposé si vos méthodes d'appliquer à des formes non rectangulaires.

Calcul d'une solution

A. Méthode de résolution (Source wikipédia)

Dans l'hypothèse où la case vide se trouve en bas à droite :

1) remettre le jeu dans l'ordre ligne par ligne en commençant par la ligne du haut ;
2) quand il ne reste plus que deux lignes mélangées, les réordonner colonne par colonne en commençant par celle de gauche.

Cette méthode ne garantit pas qu'un nombre minimal de mouvements sera effectué, mais est simple à mémoriser et aboutit dans tous les cas où une solution est possible.

B. Parcours de graphe

L'objet de la méthode est de parcourir le graphe à partir de la position initiale et dès que la position finale est atteinte, nous déduisons la suite d'actions menant à cette position finale. Nous considérons pour ce parcours deux ensembles :

1) Marqué : ensemble des positions parcours
2) Atraité : ensemble des positions nécessitant le parcours des successeurs.

L'algorithme s'écrit informellement de la manière suivante :

```
Marqué.ajout(Initial) ;
ATraité.ajout(Initial) ;
Tant que (ATraité.nonVide()) {
     pos = ATraite.prend() ;
     pour p : pos.succ() {
          si ( ! Marqué.appartient(p)) {
               Marqué.ajout(p) ;
               ATraité.ajout(p) ;
          }
     }
}
```

Cet algorithme doit être adapté pour s'arrêter dès que l'état final est atteint et pour enregistrer les informations nécessaires à l'extraction de la solution.

Nous distinguons plusieurs types de parcours en fonction du choix de la structure de données utilisée pour coder l'ensemble `ATraité` :

1) File: parcours en largeur
2) Pile: parcours en profondeur
3) Tas (i. e. file à priorité) : sélection de la position ayant la plus faible valeur d'une fonction donnée. Nous considérons les fonctions :
    a. Distance de Manhattan
    b. Profondeur + distance de Manhattan
    c. Une fonction de votre choix

Le tableau suivant donne la valeur de l'option `–algo` en fonction du type de parcours :

Pile	`-algo pile`
File	`-algo file`
distance de Manhattan	`-algo manhattan`
Profondeur + Manhattan	`-algo pmanhattan`
Fonction au choix	`-algo choix`

C. Parcours de graphe à information incomplète

Dans cette méthode, il s'agit de remplacer l'ensemble `Marqué` par une table de booléens d'une taille donnée à l'avance. Ainsi, à chaque position, nous associons un indice dans le tableau. Une position sera considérée comme traitée si sa valeur dans le tableau est vraie. Attention, il faudrait adapter la méthode pour extraire la solution quand l'algorithme est concluant.

Le tableau suivant donne la valeur de l'option `–algo` en fonction du type de parcours :

Pile	`-algo bit <taille> pile`
File	`-algo bit <taille> file`
distance de Manhattan	`-algo bit <taille> manhattan`
Profondeur + Manhattan	`-algo bit <taille> pmanhattan`
Fonction au choix	`-algo bit <taille> choix`

Avec `<taille>` désigne la taille de la table.

D. Parcours en profondeur amélioré

Dans le cas d'un parcours en profondeur, l'ensemble `ATraité` peut être remplacé par une suite d'actions et la position obtenue en appliquant cette suite d'actions à partir de la position initiale. Notons que nous n'avons plus besoin d'enrichir l'ensemble Marqué d'informations complémentaires pour extraire la solution.

L'ordre de parcours des successeurs d'une position permet de définir plusieurs parcours de le graphe. Nous distinguerons trois méthodes :

1) appliquer les actions toujours dans le même ordre,
2) choisir en priorité la position ayant la plus faible distance de Manhattan
3) choisir en priorité la position ayant la plus faible valeur d'une fonction de votre choix

Le tableau suivant donne la valeur de l'option `–algo` en fonction du type de parcours

même ordre	`-algo`
distance de Manhattan	`-algo prof manhattan`
fonction au choix	`-algo prof choix`
même ordre + info. incomplète	`-algo bit <taille> prof`
distance de Manhattan + info. incomplète	`-algo bit <taille> prof manhattan`
fonction au choix+ info. incomplète	`-algo bit <taille> prof choix`

E. Parcours en profondeur à approfondissement progressif

La méthode consiste à lancer une série d'exploration du graphe en profondeur sans enregistrer les positions traitées. Afin que les algorithmes terminent, la taille du chemin est bornée à la distance de Manhattan pour la première exploration et incrémentée de 1 après chaque exploration non concluante.

Le tableau suivant donne la valeur de l'option `–algo` en fonction du type de parcours en profondeur utilisé.

même ordre	`-algo progressif`
distance de Manathan	`-algo progressif manhattan`
fonction au choix	`-algo progressif choix`

Indication : Si dans une exploration limitée à `n`, l'algorithme traite une position à une profondeur de `p` et ayant une distance de Manhattan `q` avec `n<p+q`, il est inutile de poursuivre le parcours sur ses successeurs.

F. Elagage des séquences de coups redondantes

La méthode d'élagage est décrite dans :
[1] Larry A. Taylor, Richard E. Korf: Pruning Duplicate Nodes in Depth-First Search. AAAI 1993: 756-761

Elle utilise l'algorithme suivant :
[2] Alfred V. Aho, Margaret J. Corasick: Efficient String Matching: An Aid to Bibliographic Search. Commun. ACM 18(6): 333-340 (1975)

Une description complète de la méthode avec des indications sur l'implémentation sera donnée en cours.

Le tableau suivant donne la valeur de l'option `–algo` en fonction du type de parcours en profondeur utilisé.

même ordre	`-algo redondant n`
distance de Manathan	`-algo redondant n manhattan`
fonction au choix	`-algo redondant n choix`

où `n` désigne la profondeur de l'apprentissage.

G. Votre méthode

Proposer votre méthode. L'option est `–algo moi` avec éventuellement un complément d'option.

III. Que doit faire votre programme ?

Le programme java sera composé d'une unique archive jar **taquin.jar**. Ce programme devra s'exécuter dans un terminal avec les options suivantes :

- `java -jar taquin.jar -name` affiche vos noms et prénoms
- `java -jar taquin.jar -h` rappelle la liste des options du programme
- `java -jar taquin.jar –sol fichier.taq -j` test si le jeu a une solution à partir de la position initiale donnée dans le fichier `fichier.taq`. Le format d'un fichier taq est le suivant : largeur, longueur, liste des valeurs contenues dans le tableau avec 0 représentant la case vide.
- `java -jar taquin.jar –joue fichier.taq` permet à l'utilisateur de jouer sur le terminal. Le fichier `fichier.taq` contient la position initiale du jeu.

- `java -jar taquin.jar -cal delai algo fichier.taq` calcule une solution en utilisant l'algo `algo`. Votre programme renvoie une solution (si possible) sous la forme d'une liste de positions (une par ligne). Si le temps d'exécution de votre programme excède la durée `delai`, il doit s'interrompre brutalement.
- `java -jar taquin.jar -anime delai algo fichier.taq` est identique à la commande précédente sauf que la solution est présentée sous la forme d'une animation.
- `java -jar taquin.jar -stat delai algo fichier.taq` est identique à la commande précédente sauf que le programme renvoie des statistiques sur l'exécution du programme : taille de la solution, nombre de positions parcourues et temps d'exécution.
- `java -jar taquin.jar -stat delai fichier.taq` applique la commande précédente à tous les algorithmes et renvoie sous la forme d'un tableau html toutes les statistiques.
- `java -jar taquin.jar -aleatoire n largeur hauteur delai fichier.taq` applique tous vos algorithmes à `n` positions initiales générées de façon aléatoire de taille `largeur x hauteur`. Vous donnerez pour chaque algorithme : le nombre de problème résolut, et les moyennes des tailles des solutions, du nombre de solutions parcourus et du temps d'exécution. Les résultats seront présentés sous la forme d'un tableau html.

Pour obtenir une note supérieure à 16, la méthode d'élagage des séquences redondantes devra être correctement mise en œuvre.

Nous vous proposons de gagner 3 points de bonus si les commandes précédentes s'appliquent à des formes non nécessairement rectangulaires. Pour les distinguer des commandes précédentes vous ajouterez l'option `-plus`.

IV. Quoi et quand rendre ?

A rendre le 15 mai 2015.
Quoi :

- format papier : rapport à remettre au secrétariat
- par mail :
  - contenant les personnes du groupe,
  - rapport (format pdf) ,
  - le programme (fichier jar exécutable, contenant aussi les sources),
  - dossier compressé contenant les données d'expérimentation.

V. Quelles références disponibles sur la toile et à la bibliothèque des sciences

1. http://fr.wikipedia.org/wiki/Taquin pour connaître les règles du jeu
2. http://www.termsys.demon.co.uk/vtansi.htm contient des commandes de contrôle du terminal
3. http://docs.oracle.com/javase/8/ API java
4. http://java.sun.com/docs/codeconv/index.html donne les conventions d'écriture d'un programme java
5. La programmation en pratique, Brian W. Kernigan et Rob Pike. Editeur Vuibert informatique. Lire le premier chapitre sur la manière de commenter un programme et choisir des noms de va- riables

6. Au coeur de Java 2 (Volume 1), Notions Fondamentales, Cay S. Horstmann et Gary Cornell. Editeur CampusPress. Livre recommandé.
7. Algorithmique - 3ème édition - Cours avec 957 exercices et 158 problèmes, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Dunod.