Lundi 12 Décembre 2022

SAE 2.02
GRAPHE

Yoann GAUTIER
Benjamin CONSEIL
1E2

SOMMAIRE

Introduction

Le problème du tour du cavalier consiste à trouver un chemin qui permet à un cavalier de visiter toutes les cases d'un plateau d'échecs, sans jamais repasser sur une case déjà visitée. Pour résoudre ce problème, nous pouvons utiliser la théorie des graphes, en représentant le plateau comme un graphe, chaque case comme un sommet de ce graphe et les mouvements possibles du cavalier a partir de cette case comme les arêtes de ce graphe.

On peut donc considérer ce problème comme le fait de chercher un chemin hamiltonien, c'est-à-dire, un chemin qui parcourt le graphe en passant une et une seule fois par chaque sommet.

Ce problème présente plusieurs particularités et difficultés. Tout d'abord, la taille du plateau peut changer, ce qui peut poser un problème de rapidité due au nombre très élevé de solutions, avec les plateaux de grande taille. De plus, nous avons eu du mal à savoir comment comptabiliser les cases visitées et effectuer l'algorithme en backtracking puisqu'il faut garder en mémoire le nombre de coups essayé pour chaque case, mais le supprimer si on revient en arrière. Enfin, nous avons essayé de faire un algorithme itératif, par contraste avec le récursif, normalement utilisé pour les algorithmes en backtracking.

Description

Pour réussir à trouver un tour du cavalier, nous avons construit un programme Python en backtracking, cherchant un chemin hamiltonien, via un parcours du graphe en profondeur, en partant de la case donnée et de nous l'afficher sous forme de tableau si il existe.

Le premier programme que nous avons fait est un programme itératif.

Il demande tout d'abord la dimension de l'échiquier que nous allons étudier:

```
Entrez le nombre de colonnes(entre 1 et 15): 6
Entrez le nombre de lignes(entre 1 et 15): 6
```

Ensuite, il nous demande la case de départ souhaitée:

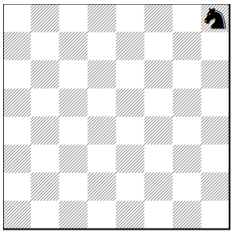
```
Entrez la ligne de la case de départ (entre 1 et  6 ): 1
Entrez la colonne de la case de départ (entre 1 et  6 ): 1
```

Il vérifie si les valeurs entrées sont cohérentes et commence sa recherche.

Il part de la première case, et vérifie quelles cases le cavalier peut atteindre et qui ne sont pas déjà visitées. Il choisit l'une d'entre elles et répète le même principe avec cette nouvelle case. Si il est bloqué, il revient à la case précédente et choisit un autre case qu'il peut atteindre(backtracking).

A la fin du programme, 2 cas sont possibles:

1. Il a exploré toutes les possibilités et est retourné à la case de départ sans trouver de chemin hamiltonien.
2. il a trouvé un chemin hamiltonien et nous le représente sous forme de tableau



Voici un exemples de résultat pour un cavalier placé comme ci-dessus:

```
Entrez le nombre de colonnes(entre 1 et 15): 6
Entrez le nombre de lignes(entre 1 et 15): 6
Entrez la ligne de la case de départ (entre 1 et  6 ): 1
Entrez la colonne de la case de départ (entre 1 et  6 ): 6
6 6
6 1
tour du cavalier:
[ 36 ] [ 31 ] [ 18 ] [ 29 ] [ 34 ] [ 1 ]

[ 17 ] [ 22 ] [ 35 ] [ 32 ] [ 19 ] [ 28 ]

[ 6 ] [ 11 ] [ 30 ] [ 21 ] [ 2 ] [ 33 ]

[ 23 ] [ 16 ] [ 7 ] [ 12 ] [ 27 ] [ 20 ]

[ 10 ] [ 5 ] [ 14 ] [ 25 ] [ 8 ] [ 3 ]

[ 15 ] [ 24 ] [ 9 ] [ 4 ] [ 13 ] [ 26 ]
```

Comparaison argumentée

On a choisi de partir sur deux programmes, qui possèdent un algorithme différent. D'un côté on y retrouve un programme orienté objet, utilisant de l'itératif et d'un autre côté, un programme procédural utilisant de la récursive. En termes de points communs entre les deux programmes, on y retrouve l'utilisation d'algorithmes de recherche en profondeur, ainsi que du backtracking.

Les 2 programmes utilisent une fonction qui recherche une liste de coups possible, faisant elle-même appel à une fonction vérifiant que le coup est valide. Il n'y a donc pas de différence notable sur ce point.

Cependant, pour le parcours du plateau, en backtracking, l'algorithme est totalement différent.
En effet, l'un effectue le parcours en profondeur via une fonction récursive tandis que l'autre le fait via une fonction itérative.
Ces 2 possibilitées offrent chacun leur avantage:

- l'algorithme récursive permet de réduire le nombre de lignes du code, ce qui permet de l'épurer et rendre moins lourde la lecture du code par le développeur. Cependant, il est plus compliqué de comprendre le fonctionnement du code, surtout pour un développeur novice. De plus, l'algorithme récursif est bien souvent plus lent que l'itératif.
- l'algorithme itératif, quant à lui, permet d'expliciter le code, puisque les étapes sont affichées dans l'ordre, pas à pas. Cependant, il exige de nombreux tests et multiplie le nombre de lignes, ce qui peut amener le développeur à se perdre dans son programme. De plus, l'algorithme itératif est bien souvent plus rapide que récursif.

Pour l'affichage, les 2 programmes utilisent chacun une version différente, mais on ne peut pas noter de différence notable, mise à part que l'un est bien plus concis que l'autre, réduisant le nombre de ligne, facilitant la compréhension pour des développeurs expérimentés mais la complique pour des novices.

Conclusion

En conclusion, on peut dire que nos différents algorithmes sont relativement efficaces pour des échiquiers de taille moyenne, mais pourraient être optimisés pour des échiquiers plus grands.

Des améliorations possibles pourraient inclure l'utilisation d'une autre méthode de recherche de chemin, notamment l'utilisation des probabilités qui permettrait de calculer en temps réel la probabilité du meilleur coup à jouer pour pouvoir rendre le programme encore plus efficace et donc plus précis.

Il serait également possible d'ajouter des fonctionnalités supplémentaires, telles que la recherche d'un chemin hamiltonien pour tous les points de départ possibles sur l'échiquier, ou également la recherche d'un cycle hamiltonien fermé. Sur ce point d'ailleurs, nous avons essayé de faire un algorithme de recherche de cycle fermé, mais nous nous sommes rapidement rendu compte du gouffre de vitesse de calcul séparant les deux. En effet, lorsque le programme de recherche de chemin prends quelques secondes le programme de recherche de cycle lui met plusieures heures(4 heures pour un échiquier de 6 * 6).