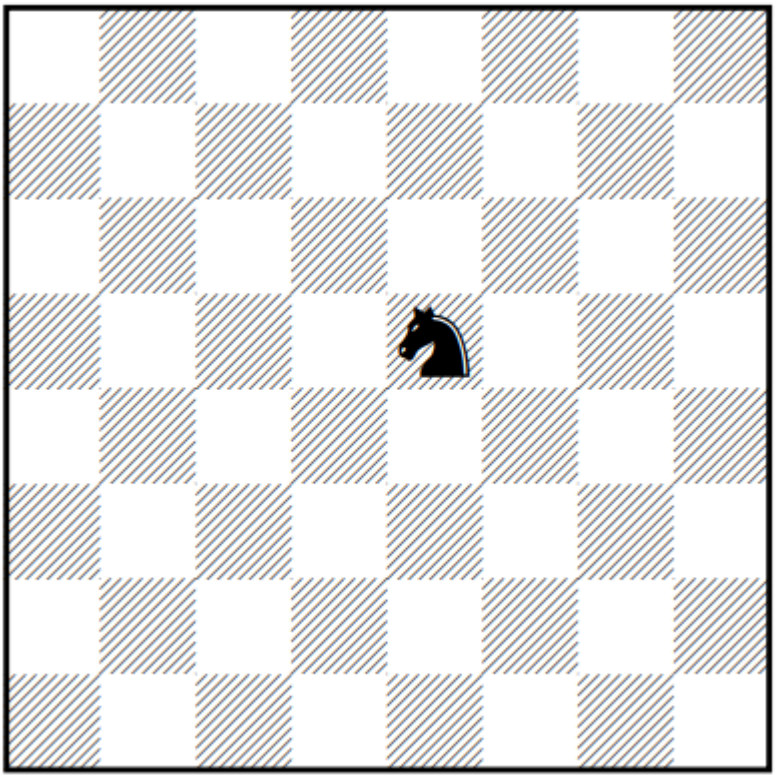
1A1
Mahé Pineau
Mathis Le Boite

Le tour du cavalier

Compte rendu



Sommaire

Sommaire 1

Introduction 2

1.L’algorithme de résolution 3

2.Comparaison avec l’algorithme de L’université de Lyon 1 5

Conclusion 7

Introduction

Ce problème connu depuis la nuit des temps dans le monde arabe est basé sur un seul objet très commun : un jeu d'échecs et plus précisément l'échiquier et une pièce de cavalier. Le but paraît très simple, il faut faire passer le cavalier sur toutes les cases de l'échiquier. Bien que la version avec l'échiquier de 8 lignes et 8 colonnes qui contient le cavalier au centre de la grille reste la plus connue il en existe d'autres que ce soit avec une grille de 130x130, avec un pion qui se situe dans un des coins ou encore avec un chemin qui retourne à case de départ. Pour notre cas, on ne souhaite pas que cette dernière variante soit appliquée mais seulement que notre algorithme fonctionne quelque soit la taille de l'échiquier et la position de la pièce.

1.L’algorithme de résolution

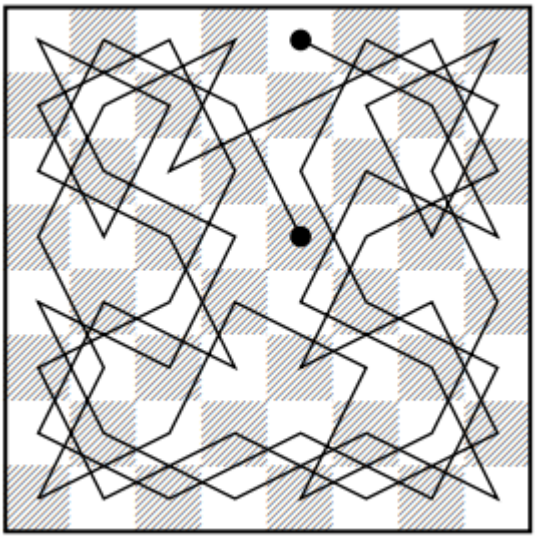
Le programme commence par définir la Taille de l’échiquier selon l’envie de l’utilisateur puis il crée une chaîne de caractères pour servir de "séparateur" lors de l'affichage de l'échiquier si une animation est demandée, et initialise l'échiquier avec le caractère ".". Ensuite, il définit les huit mouvements possibles d'un cavalier

La fonction ”prochain_coup” fait jouer le cavalier avec tous ses coups possibles puis vérifie si ses coups sont valables (dans l’échiquier et position jamais atteinte). Ensuite, elle compte le nombre de coups possibles pour chaque situation et ajoute toutes les positions obtenues précédemment ainsi que leur nombre de coups possible sur cette position sous forme d’un couple dans la liste “coup_suivant”. Pour faciliter l’exécution du programme et pour améliorer son temps d'exécution, nous avons trié la liste “coup_suivant” sur les nombres de coups possibles et de manière croissante afin de prioriser les positions où il y a peu de coups à jouer. Ensuite nous retournons toutes les position (trié) de “coup_suivant” sous la forme d’une liste

La fonction principale “soluce” est récursive et elle prend en entrée une position de départ et un compteur. Elle commence par marquer les coordonnées mises en entrée à l’aide du numéro du coup qui est “compteur” (le paramètre d’entrée). Puis si une animation a été demandée, elle fait attendre le programme pendant 1 seconde, puis affiche le numéro du coup puis ensuite affiche le coup en question. Si cette animation n’est pas demandée, elle ne prend pas en compte ces instructions et passe directement à la suite. Juste après, nous avons défini comme condition d’arrêt de la récursivité le fait que le compteur est égale à la taille de l’échiquier au carré. Nous avons utilisé cette condition car le nombre de coups total possible est justement la taille de l’échiquier au carré. Dans le cas où le compteur n’est pas égal à la taille au carré, la fonction appel “soluce” pour toutes les valeurs que “prochain_coup” renvoie (boucle for ligne 49 puis ligne 50) et incrémente le compteur.

La suite du programme demande à l’utilisateur la position de départ du cavalier en définissant la colonne et la ligne de départ (elle doit être comprise entre 1 et n et si la valeur entrée n’est pas conforme, une nouvelle saisie est demandée). Ensuite, l’utilisateur choisit s' il veut une animation (un affichage progressif des mouvements du cavalier). Une vérification est aussi appliquée à cette saisie. Ensuite, “start_time” reçoit en valeur, “l’heure de lancement de programme” soit le moment où le programme est lancé. Ensuite la fonction “soluce” est appelée en lui attribuant en entrée la position de départ (lig et col modifié afin de

correspondre au demande de l'utilisateur) et le compteur qui est mis à 1. Puis on effectue l'affichage final et on fait la même chose à "end_time" qu'a "start_time" afin d'effectuer une soustraction entre les deux et d'obtenir le temps d'exécution.



(b) Une solution : chemin hamiltonien

2.Comparaison avec l'algorithme de L'université de Lyon 1

Le programme avec lequel nous allons nous comparer est celui de L'université de Lyon 1. Vous pouvez le retrouvez ici:
https://math.univ-lyon1.fr/irem/Formation_ISN/formation_recursivite/grille/cavalier.html

puis en cliquant sur "un code ".

Sachant que leur programme est différent du nôtre (la position de départ est aléatoire, la taille est fixé à 6 et il n'on pas implémenté de système d'animation), nous avons modifié leur programme afin qu'il ressemble un peu plus au nôtre en ajoutant la possibilité de choisir la position de départ et la taille de l'échiquier, mais nous avons aussi ajouté de quoi mesurer le temps d'exécution. Nous avons également adapté notre programme afin que l'animation influe le moins possible sur le temps d'exécution. Cependant, pour être le plus sûr du temps d'exécution obtenu, les mesures ont été prises sans animation.

Voici les mesures prisent concernant le temps d'exécution avec taille qui correspond au nombre de cases de la grille et col et lig aux coordonnées de départ du cavalier :
Nous:

Taille(5), col(3) et lig(3) : Temps d'exécution : 0.0008409023284912109 secondes

Taille(5), col(3) et lig(4) (pas de chemin possible): Temps d'exécution : 0.0008518695831298828 secondes

Taille(6), col(2) et lig(5) : Temps d'exécution : 0.0015223026275634766 secondes

Taille(7), col(1) et lig(7) : Temps d'exécution : 0.0016858577728271484 secondes

Taille(18), col(5) et lig(5) : Temps d'exécution : 0.022462844848632812 secondes

Taille(31), col(15) et lig(15) : Temps d'exécution : 0.0724935531616211 secondes

Ne fonctionne pas au-delà de cette taille.

Université de Lyon 1:

Taille(5), col(3) et lig(3) : Temps d'exécution : 0.017077207565307617 secondes

Taille(5), col(3) et lig(4) (pas de chemin possible): Temps d'exécution : 4.557119846343994 secondes

Taille(6), col(2) et lig(5) : Temps d'exécution : 11.37851071357727 secondes

Taille(7), col(1) et lig(7) : Temps d'exécution : 14.240478277206421 secondes
Ne fonctionne pas/mal au-delà de cette taille.

Notre programme est donc bien plus rapide que celui de l’université de Lyon 1.

De plus, notre programme supporte une complexité bien supérieure au leur. En effet, dans le cadre de nos deux programmes, le seul élément qui influe sur la complexité est la taille de l’échiquier. Sachant que leur programme n'est plus aussi efficace à partir de 7 cases de long, on peut dire que notre programme est aussi meilleur sur ce point car il peut aller jusqu'à une taille de 31 cases de long !

Conclusion

En conclusion, bien que notre programme fonctionne parfaitement, on aurait aimé ajouter une interface graphique pour rendre le tout plus lisible et agréable. De plus les animations ne suppriment pas ce qu'il y a dans le terminal au fur et à mesure car nous n'avons pas réussi à implémenter cette fonctionnalité (c'est pour cette raison que nous traçons des traits du style :"------" entre chaque coup). On trouve que notre algorithme fonctionne très bien, il est rapide au point que l'on soit obligé de rajouter du temps d'attente pour bien voir les coups qu'il fait au fur et à mesure et efficace par le fait qu'il fonctionne dans presque tous les cas. En effet, si je dois chipoter, il ne peut pas traiter tous les cas car il ne marche plus quand la taille de l'échiquier est supérieur à 31, non pas par manque de puissance mais parce que nous dépassons le nombre d'appels récursif maximum. Voici ce que j'obtiens lorsque j'utilise notre programme dans ces conditions :

```
Traceback (most recent call last):
  File "main.py", line 72, in <module>
    if soluce((lig-1,col-1),1):
  File "main.py", line 50, in soluce
    if soluce(pos_suivante, compteur+1): # Incrémentation du compteur afin d'identifier le numéro du coup
  File "main.py", line 50, in soluce
    if soluce(pos_suivante, compteur+1): # Incrémentation du compteur afin d'identifier le numéro du coup
  File "main.py", line 50, in soluce
    if soluce(pos_suivante, compteur+1): # Incrémentation du compteur afin d'identifier le numéro du coup
  [Previous line repeated 993 more times]
  File "main.py", line 49, in soluce
    for pos_suivante in prochain_coup(pos): # Boucle utilisant la récursivité afin de faire jouer le cavalier grâce à prochain_coup
  File "main.py", line 28, in prochain_coup
    coup_suivant.sort(key=lambda x: x[1]) # Triage de coup_suivant par rapport au nombre de coups possible après avoir joué
(compteur). Le trie est fait de manière croissante
RecursionError: maximum recursion depth exceeded
```

Nous aurions aussi pu rajouter une fonctionnalité qui permet de trouver si pour une certaine taille d'échiquier, il existe une solution au problème du tour du cavalier.