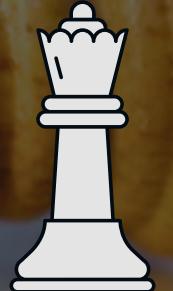


SAE 2.02

Exploration et comparaison algorithmique



Le problème des huit dames



Info - 1A

Yanis PONTHOU | Yohann MEAR | Dorian POLICE

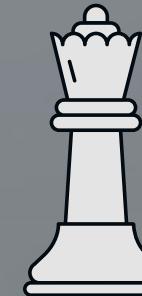
SOMMAIRE

- ▶ I. Introduction
- ▶ II. Description Algorithme n°1
- ▶ III. Description Algorithme n°2
- ▶ IV. Comparaison des Algorithmes
- ▶ V. Conclusion

I. Introduction



Le problème des huit dames



Le but du problème des huit dames est de placer huit dames d'un jeu d'échecs sur un échiquier de 8×8 cases sans que les dames puissent se menacer mutuellement, conformément aux règles du jeu d'échecs (la couleur des pièces étant ignorée).

Par conséquent, deux dames ne doivent jamais partager la même rangée, colonne, ou diagonale.



I. Introduction

Stratégie d'approche

Afin de pouvoir résoudre ce problème, il faut :

- Analyser les cas, pour un échiquier de taille $n*n$ (exemple $8*8$ qui admet 92 solutions) + Représenter : nombre de dames pouvant être placées en même temps sur l'échiquier (avec leurs positions)
- Réponse au problème : arbre de décision
- Analyse de l'arbre de décision : utilisation d'algorithme utilisant le principe du backtracking
- Backtracking : retour en arrière lorsque la solution proposée n'est pas la bonne.
- Comparaison : les 2 algo recherche la solution à partir d'une dame à une position donnée mais de manière différente, le 1er le fait au début sur la 1ere ligne et le 2e le fait à la fin parmi toutes les solutions possibles

II. Description Algorithme n°1

Les différentes fonctions :

- ▶ est_valide(plateau, ligne, col)
- ▶ backtrack(plateau, ligne)
- ▶ resoudre_reines(n)

Fonction est_valide(plateau,ligne,col) :

Fonction : renvoie un booléen (si la dame peut être placé à la position donnée).

La fonction utilise une première boucle for, qui parcourt les lignes.

On compare ensuite **3 cas possibles** dans le if :

- Si une dame est déjà placée dans la **même colonne** (`plateau[i] == col`)
- Si une dame est déjà placée dans la **même diagonale montante**
- Si une dame est déjà placée dans la **même diagonale descendante**

Si jamais un de ces 3 cas est réalisé, la variable “**valide**” qui permet de savoir si l'on peut placer le pion dans la case prend la valeur **False**.

On renvoie ensuite la valeur du résultat.



II. Description Algorithme n°1

Fonction backtrack(plateau,ligne) :

Départ : définition du cas de base, si la ligne est égale à n.

Si jamais elle l'est, on renvoie ainsi une copie du tableau, car cela voudra dire qu'on aura placé des dames à toutes les lignes sans qu'elles se menacent mutuellement (sauf dans le cas où la taille de l'échiquier n'admet aucune solution).

On crée par la suite un tableau, "res", pour stocker les résultats des solutions au problème des huit reines.



II. Description Algorithme n°1

Fonction resoudre_reines(n) :

- Fonction : contient les 2 autres fonctions
- Début : demande à l'utilisateur la position de la 1ere dame
- Initialisation des variables du tableaux à [-1] * n.
- Fonction backtrack pour afficher les solutions possibles en précisant comme 2ème argument la ligne à partir de laquelle les reines seront placées.
- Demande à l'utilisateur si il veut afficher toutes les solutions.
- Affichage des solutions en format tableau et liste
- On parcourt les solutions du tableau avec des boucles "for" imbriquées

La dame est représentée par la lettre R

Les cases vides sont représentées par un .

II. Description Algorithme n°1

Les erreurs et problèmes rencontrés :

- Difficultées pour comprendre ce qu'on attendait de nous, à la fois dans le programme, mais également dans le compte rendu
- Difficultées pour afficher uniquement les solutions "possibles" (exemple : au départ de l'algorithme, des solutions étaient trouvées pour $n = 4$)

Exemple d'affichage d'une solution :

Sous forme graphique

Solution 12:
.....|R|...|...|...|...|
....|...|...|...|...|R|...|
...|...|...|...|...|...|...|R|
.|...|...|...|...|R|...|...|
..|...|...|...|...|...|...|R|
...|...|...|R|...|...|...|...|
...|...|...|...|...|R|...|...|

Solution 13:
.....|R|...|...|...|...|
....|...|...|...|...|R|...|
...|...|...|...|...|...|...|R|
.|...|...|...|...|R|...|...|
..|...|...|...|...|...|...|R|
...|...|...|...|...|...|...|R|
...|...|...|R|...|...|...|...|

Sous forme de liste avec le numéro de colonne

[2, 5, 7, 0, 4, 6, 1, 3], [2, 5, 7, 1, 3, 0, 6, 4]

III. Description Algorithme n°2

Les différentes fonctions :

- ▶ est_valide(plateau,ligne,colonne)
- ▶ n_reines(n, ligne=0, plateau=None,solutions=None)
- ▶ affiche_solutions(sol,n,x,y)
- ▶ TransformerSolutionsEnCoordo(liste)



Fonction est_valide(plateau,ligne,col) :

Même principe que l'algorithme n°1, les diagonales sont vérifiées à l'aide de la propriété mathématiques tiré de la géométrie dans l'espace qui est : si $|xa - xb| == |ya - yb|$ alors les reines sont sur la même diagonale.

Si la reine n'est pas menacée alors la fonction renvoie "True"

III. Description Algorithme n°2

Fonction n_reines(n,ligne=0,plateau=None,solutions=None):

- **n** est la taille de l'échiquier et donc le nombre de reines à placer
- **ligne** est l'indice de la ligne actuelle que nous sommes en train de considérer. Elle est initialisée à 0.
- **plateau** est un tableau qui représente l'échiquier. Il est initialisé à None, mais lors de la première itération, on crée un tableau de longueur **n** contenant des -1.
- **solutions** est une liste de listes contenant toutes les configurations possibles des **n** reines sur l'échiquier. Elle est initialisée à None.



III. Description Algorithme n°2

Fonction n_reines (suite):

- Début : initialisation du tableau plateau + liste de solutions (si nécessaire)
- Vérifie si la dernière ligne a été atteinte, dans ce cas, elle ajoute la configuration actuelle à la liste des solutions et retourne la fonction pour poursuivre la récursivité.
- Sinon, elle parcourt toutes les colonnes de la ligne actuelle, vérifie grâce à la fonction est_valide si une reine peut être placée, puis si elle peut, elle marque la case comme occupée.
- Puis appelle récursivement la fonction n_reines pour passer à la ligne suivante.
- Si la fonction est valide renvoie False alors une possible solution est fermé parmi toute les solutions

III. Description Algorithme n°2

Fonction affiche_solutions(sol,n,x,y):

But de la fonction : Parcourir les solutions contenues dans la liste "sol" puis va trouver et ajouter à la liste l les solutions pour lesquelles le positionnement de la dame passée en paramètre est présent et vérifié.
Cette fonction permet de connaître les solutions pour une dame donnée parmi une liste de solutions.

TransformerSolutionsEnCoordo(liste):

But de la fonction : Transforme une liste qui représente une solution en coordonnées pour permettre une compréhension du résultat plus simple.

On part du principe que nous avons créés nos listes de solutions avec l'indice de la liste = la ligne.



IV. Comparaison des Algorithmes

Comparaison :

- Même Logique pour les 2 algorithmes
- Pas la même manière de réfléchir
- 1er Algo : Demande à l'utilisateur d'une solution de préférence avec la position de la 1ere reine
 - Permet de limiter le nombre de calculs (nombre de solutions)
- 2e Algo : Il calcule toutes les solutions
 - Récupère parmi les solutions celles qui admettent une dame aux coordonnées souhaitées



IV. Comparaison des Algorithmes

Inconvénients :

- 1er Algo : Ne permet pas d'avoir les solutions pour une dame qui est placée au-delà de la première ligne
- 2e Algo : Très lourd
 - Plus la taille de l'échiquier est grande, plus le temps de réponse est long
 - Il va chercher à trouver toutes les solutions pour un échiquier contrairement au premier algo
- Le 1er algo va donc être plus rapide grâce à la récursivité limitée, mais également en limitant le placement de la reine à la première ligne ce qui permet d'éviter d'autres boucles.



V. Conclusion

Axe d'amélioration :

- Incorporer dans le 1er algo une solution pour permettre de placer une reine sur une autre ligne pour avoir un résultat plus fidèle à une demande client.
- Trouver un moyen d'améliorer le temps de réflexion du 2e algo pour qu'il soit plus efficace sur de plus grands échiquiers.



Avantages :

- Le 1er algo est plus rapide que le 2e (3x plus long à exécuter pour un échiquier de taille 12*12)
- Si on prend le cas le plus “logique”, où l’utilisateur saisit la première dame sur la première ligne, le premier algorithme est ainsi plus efficace.
- Malgré tout, si on veut connaître les solutions pour le cas où la dame est placée sur une autre ligne, il faut privilégier le deuxième algo

V. Conclusion



Conclusion :

Chaque algorithme est ainsi très efficace pour ce qu'on lui demande : si l'on veut un algorithme plus rapide, mais où la première dame doit être placée uniquement sur la première ligne, mieux vaut choisir le premier algorithme. Dans l'autre cas, où la vitesse d'exécution de l'algorithme n'est pas primordial, l'algorithme numéro 2 sera préférable. Il est malgré tout important de préciser que, pour un échiquier plus petit, comme un échiquier de 4*4, la différence de vitesse d'exécution est légère..

Merci de votre attention

