IUT de Lannion
BUT Informatique

Année universitaire 2023-2024
Semestre 2

SAÉ 2.02

Rapport

Exploration algorithmique

Le problème des huit reines, backtracking

1C1

Pfranger Mathéo

Roselier Lisa

Poupon Camille

Fauchet Malo

26 mars 2024

Sommaire

1 - Introduction

Le problème des huit reines est l'un des problèmes classiques en informatique et en mathématiques. Il consiste à placer huit reines sur un échiquier de taille 8x8 de manière à ce qu'aucune reine ne puisse menacer une autre. C'est-à-dire qu'aucune reine ne puisse être sur la même ligne, colonne ou diagonale qu'une autre reine (Il peut être étendu au problème des *N* reines avec un échiquier de taille *N*x*N*).

L'algorithme de backtracking est une technique puissante souvent utilisée pour résoudre des problèmes de recherche de solution comme le problème des huit reines. Il fonctionne en explorant toutes les possibilités de manière récursive, en plaçant une reine à la fois sur l'échiquier et en vérifiant si cette configuration est valide. Si une configuration valide est trouvée, l'algorithme continue à explorer d'autres possibilités. Sinon, il revient en arrière (backtrack) pour essayer une autre configuration.

Dans ce rapport, nous présenterons plusieurs solutions au problème des huit reines en utilisant l'algorithme de backtracking. Nous détaillerons l'implémentation de cet algorithme en mettant l'accent sur la façon dont il est appliqué pour résoudre le problème, ainsi que sur les résultats obtenus et leur analyse. Enfin, nous discuterons des implications de ces solutions et des possibilités d'extension vers d'autres problèmes similaires.

2 - Première Solution

2.1 - Initialisation du Graphe

L'une des premières étapes de notre approche pour résoudre le problème des huit reines est d'initialiser un graphe représentant l'état du jeu. Dans notre implémentation, nous utilisons la classe *HuitReines* pour gérer cette initialisation.

La méthode *initGraphe()* de la classe *HuitReines* est responsable de cette initialisation. À l'intérieur de cette méthode, nous utilisons une double boucle pour parcourir toutes les cases de l'échiquier. Pour chaque case, nous ajoutons un sommet au graphe, représentant ainsi chaque position possible pour placer une reine.

Cette initialisation crée un graphe où chaque sommet est une case de l'échiquier, permettant ainsi une représentation efficace de l'état du jeu. Ce graphe sera utilisé par la suite pour calculer les menaces entre les reines placées et pour effectuer le backtracking afin de trouver une solution au problème.

L'initialisation du graphe constitue donc une étape fondamentale de notre approche pour résoudre le problème des huit reines. Elle fournit une structure de données appropriée pour représenter l'état du jeu et effectuer les calculs nécessaires pour trouver une solution.

2.2 - Backtracking

L'algorithme de backtracking est au cœur de notre approche pour résoudre le problème des huit reines. Il fonctionne en explorant toutes les configurations possibles pour placer les huit reines sur l'échiquier, en suivant un processus de recherche récursif.

Dans notre implémentation, l'algorithme de backtracking est réalisé dans la méthode *backtracking()* de la classe *HuitReines*. Cette méthode prend en charge le placement récursif des reines sur l'échiquier et la vérification de la validité de chaque configuration.

Le processus de backtracking commence par placer une reine dans la première colonne de l'échiquier. Ensuite, il explore récursivement toutes les possibilités de placement pour les colonnes suivantes. À chaque placement d'une reine, les menaces sont recalculées. L'algorithme place ensuite une reine dans la colonne suivante à la première ligne non menacée. Si aucune n'est trouvée, le programme revient une colonne en arrière et déplace la reine à la prochaine ligne disponible, puis recalcule les menaces.

Si une reine est placée sur la dernière colonne, cela signifie qu'une solution a été trouvée. Cette solution est alors enregistrée dans la liste des solutions. Sinon, l'algorithme revient en arrière (backtrack) pour explorer d'autres possibilités.

Le processus de backtracking se poursuit jusqu'à ce que toutes les configurations possibles aient été explorées ou qu'une solution soit trouvée. À la fin de l'exécution de l'algorithme, la liste des solutions contient toutes les configurations valides pour résoudre le problème des huit reines.

3 - Deuxième Solution

3.1 - Initialisation et Fonctions Utilisées

L'une des premières étapes de notre approche pour résoudre le problème des huit reines est d'initialiser une structure de données pour représenter l'échiquier ainsi que les fonctions nécessaires pour vérifier si une reine peut être placée à une certaine position.

Nous utilisons une classe Graphe pour représenter un graphe non orienté. La méthode *ajouterArete()* permet d'ajouter une arête entre deux sommets, et la méthode *obtenirVoisins()* retourne une liste des voisins d'un sommet.

De plus, la fonction *checkReine()* vérifie s'il est possible de placer une reine à une position donnée sur un plateau en vérifiant les colonnes et les diagonales. Enfin, la fonction *bfs()* effectue un parcours en largeur pour trouver les configurations valides du problème des huit reines.

3.2 - Méthode de Recherche en Largeur (BFS)

Dans la fonction *bfs()*, nous utilisons une approche de recherche en largeur pour explorer toutes les configurations possibles du placement des reines.

Nous utilisons une file pour stocker les configurations à explorer, en commençant par la configuration initiale avec une seule reine placée. À chaque étape, nous retirons une configuration de la file et explorons toutes les possibilités pour ajouter une nouvelle reine tout en vérifiant si elle ne menace pas les autres reines déjà placées.

Avant d'ajouter une nouvelle reine à une configuration, nous vérifions si elle est sûre en utilisant la fonction *checkReine()*, qui s'assure qu'aucune reine ne menace les autres dans la configuration actuelle. Si une configuration valide est trouvée avec toutes les reines placées, elle est ajoutée à la liste des solutions.

Bien que le backtracking ne soit pas explicitement utilisé, cette méthode permet d'explorer toutes les configurations possibles de manière systématique, garantissant ainsi que toutes les solutions valides pour le problème des huit reines sont trouvées.

4 - Troisième Solution

4.1 - Fonctions Utilisées

La méthode *disponible()* retourne une liste des colonnes disponibles pour placer une nouvelle reine dans la configuration actuelle.

La méthode *creer_fils()* génère tous les fils possibles à partir du nœud en ajoutant une nouvelle reine dans les colonnes disponibles.

La méthode *interdites(i, pos_i)* retourne un ensemble des colonnes interdites pour placer une nouvelle reine dans la ligne suivante, en fonction de la position d'une reine actuelle.

4.2 - Méthode de Recherche en Profondeur (DFS)

La méthode DFS est un algorithme de recherche qui explore le plus loin possible une branche du graphe avant de faire marche arrière (backtrack) et d'explorer une autre branche.

On commence par initialiser une pile pour stocker les nœuds à explorer. On ajoute initialement le nœud racine à cette pile.

Tant que la pile n'est pas vide, on répète les étapes suivantes :

- On retire le premier nœud de la pile.
- Si ce nœud correspond à une solution (toutes les reines sont placées sur l'échiquier), on le rajoute dans la liste des solutions.
- Sinon, on explore ses fils (c'est-à-dire les configurations suivantes possibles). On ajoute ces fils à la pile.

L'algorithme s'arrête lorsque la pile est vide, ce qui signifie que tous les nœuds de l'arbre ont été explorés. À ce stade, toutes les solutions possibles ont été trouvées et placées dans la liste des solutions.

Si une branche explorée ne conduit pas à une solution, l'algorithme fait marche arrière pour explorer une autre branche. Cela signifie qu'il retire les nœuds explorés de la pile et revient à un point de décision antérieur pour explorer une autre branche de l'arbre.

4.3 - Filtrage des Solutions

Cette fonction prend la colonne où se trouve la première reine en entrée.

Elle crée un nœud racine avec la colonne spécifiée et utilise la recherche en profondeur pour trouver toutes les solutions possibles en tenant compte de cette position initiale de la première reine avec la fonction *dfs_trouver_solution*().

Elle filtre ensuite les solutions pour ne conserver que celles où la première reine est placée dans la colonne spécifiée.

Elle retourne une liste de ces solutions filtrées.

5 - Résultats

Après avoir implémenté et testé différentes approches pour résoudre le problème des huit reines, nous avons obtenu des résultats significatifs qui nous permettent de comparer ces méthodes et d'évaluer leurs performances.

Tout d'abord, en ce qui concerne le temps d'exécution, nous avons constaté que la méthode de recherche en largeur (BFS) est plus rapide que la méthode de recherche en profondeur (DFS). Cela est dû au fait que le BFS explore systématiquement toutes les configurations possibles à chaque niveau de l'arbre de recherche, tandis que le DFS explore en profondeur une branche spécifique avant de revenir en arrière. De plus, le DFS explore des chemins inutiles, alors que le BFS sait les éviter. Ainsi, le BFS trouve généralement une solution plus rapidement que le DFS.

Cependant, en termes de consommation de mémoire, le BFS utilise généralement plus de mémoire que le DFS. Cela est dû au fait que le BFS doit stocker toutes les configurations possibles dans une file d'attente, ce qui peut nécessiter beaucoup d'espace mémoire, surtout pour des échiquiers de grande taille. En revanche, le DFS n'a pas besoin de stocker toutes les configurations en mémoire simultanément, car il explore une branche à la fois.

En résumé, bien que le BFS soit plus rapide en termes de temps d'exécution, il peut consommer plus de mémoire que le DFS. Le choix entre ces deux méthodes dépend donc des contraintes spécifiques de chaque problème, telles que la disponibilité de la mémoire et la taille de l'échiquier.

Voici un tableau présentant le temps d'exécution des deux algorithmes (BFS et DFS) pour deux tailles d’échiquier différentes :

	6x6	8x8
BFS	0.00009749 seconde	0.001 seconde
DFS	0.002047 seconde	0.04 seconde

6 - Conclusion

En conclusion, nous avons exploré plusieurs approches pour résoudre le problème classique des huit reines. Nous avons mis en œuvre ces approches et évalué leurs performances en termes de temps d'exécution et de consommation de mémoire.

Nous avons constaté que l'algorithme de backtracking est une méthode efficace pour résoudre le problème des huit reines, capable de trouver toutes les configurations valides pour placer les huit reines sur l'échiquier. Nous avons également constaté que différentes variantes de l'algorithme, telles que la recherche en largeur (BFS) et la recherche en profondeur (DFS), présentent des avantages et des inconvénients en termes de temps d'exécution et de consommation de mémoire.

En termes d'améliorations possibles, plusieurs pistes peuvent être explorées. Par exemple, il pourrait être intéressant d'implémenter des techniques d'optimisation pour réduire le temps d'exécution de l'algorithme, telles que l’utilisation de plusieurs processus afin d’effectuer plusieurs recherches sur plusieurs chemins en parallèle…

En résumé, le problème des huit reines est un sujet passionnant qui invite à explorer de nouvelles idées et méthodes en matière d'algorithmes. En continuant à étudier et à expérimenter avec différentes méthodes, nous pourrions découvrir de nouvelles façons innovantes de résoudre ce problème classique, ainsi que d'autres problèmes similaires en informatique et en mathématiques.