Rapport de SAE : Exploration algorithmique.

FROC Gabriel, MABIT Baptiste, POMMEREUL Noa, COCHET IWAN

Le problème des “8 reines”:

Introduction

Le problème des 8 reines est un problème mathématique classique qui consiste à placer 8 reines sur un échiquier de taille 8x8 de manière à ce qu'aucune reine ne puisse attaquer une autre reine. Une reine peut attaquer une autre reine si elles se trouvent sur la même ligne, colonne ou diagonale. Malgré sa simplicité apparente, le problème est complexe avec 92 solutions possibles et nécessite souvent des approches avancées pour être résolu efficacement.

Approches Algorithmiques

Trois approches différentes ont été implémentées pour résoudre le problème des 8 reines, chacune avec son propre algorithme.

Code avec des classes

Dans cette implémentation, une classe `Échiquier` est utilisée pour représenter l'échiquier et effectuer les opérations nécessaires pour placer les reines de manière appropriée. L'algorithme utilise une méthode de backtracking pour explorer toutes les combinaisons possibles de placement des reines. L'utilisateur peut choisir de placer la première reine ou non, puis afficher toutes les solutions ou seulement une seule. Le code permet une personnalisation du nombre de dames à placer dans l'échiquier.

Code Version Backtracking

Dans cette version, un algorithme de backtracking est utilisé pour résoudre le problème. L'utilisateur peut spécifier le nombre de dames qu'il souhaite placer sur l'échiquier, puis le programme génère toutes les solutions possibles en utilisant le backtracking.

Code avec Graphes (Première version)

Cette implémentation utilise des graphes pour modéliser les menaces potentielles entre les reines. L'algorithme commence par placer la première reine sur une position choisie par l'utilisateur, puis génère toutes les solutions possibles en explorant récursivement les positions des autres reines. Le nombre total de solutions est affiché à la fin.

Code avec Graphes (Deuxième version)

Cette version est similaire à la précédente, mais l'utilisateur ne peut choisir la position de la première reine que sur la première ligne de l'échiquier. Une fois la première reine placée, le programme génère toutes les solutions possibles en explorant récursivement les positions des autres reines. Le nombre total de solutions est également affiché à la fin.

Comparaison des Approches Algorithmiques

Les différentes approches algorithmiques pour résoudre le problème des 8 reines présentent des avantages et des inconvénients distincts, chacun adapté à des situations spécifiques.

1. Approche avec des Classes :

Extrait de Code :

```
"""""
class Echiquier:
        def __init__(self, nbDames):
                ...

        def placer_dame(self, ligne):
        ...

        def est_possible(self, ligne, colonne):
                ...

        def afficher_solutions(self, affiche=False):
        ...

        def placer1Dame(self):
        ...
```

```
NbDames=int(input("Le nombre de dames a placer dans l'echequier de dames*dames : "))
A1 = Echiquier(NbDames)
A1.placer1Dame()
"""
```

Avantages :

- La structure de classe fournit une encapsulation pratique pour gérer l'échiquier et les opérations associées.

- L'utilisation du backtracking est bien intégrée dans la méthode `placer_dame()`, ce qui simplifie la logique de l'algorithme.

- L'utilisateur a la possibilité de choisir la position de la première reine et d'afficher toutes les solutions ou seulement une seule.

Inconvénients :

- La complexité de la structure de classe peut rendre le code un peu plus difficile à comprendre pour les novices.

- Le choix de la position initiale de la première reine est limité à toute position sur l'échiquier, ce qui peut restreindre la flexibilité dans certains cas.

2. Approche Version Backtracking :

Extrait de Code :

```
"""
def print_solution(solution, nbdame):
    ...

def backtracking(g, case, nbdame, solutions):
    ...
def possible(g, lig, col, nbdame):
    ...

nbdame = int(input("Quelle est le nombre de dames que vous souhaitez : "))
solutions = []
grille = [[0] * nbdame for _ in range(nbdame)]
backtracking(grille, 0, nbdame, solutions)
"""
```

Avantages :

- L'implémentation du backtracking est directe et efficace, ce qui en fait une solution rapide pour trouver toutes les solutions possibles.

- L'utilisateur peut spécifier le nombre de dames à placer, ce qui permet une personnalisation facile.

Inconvénients :

- Le code est relativement plus verbeux et moins encapsulé par rapport à l'approche basée sur des classes, ce qui peut rendre la maintenance plus difficile.

- L'utilisateur ne peut pas choisir la position de départ des reines, ce qui limite la flexibilité dans certains cas.

3. Approche avec Graphes (Première version) :

Extrait de Code :

```
"""
class Graphe:
    def __init__(self, n):
    ...

    def creergraphe(self):
    ...

    def is_safe(graphe, noeud, reines):
        ...

    def resoudre(graphe, reines=[], row=0):
        ...

    def Affichagesolutions(solutions, n):
        ...

    def resoudre_problem(n):
        ...

    resoudre_problem(8)
"""
```

Avantages :

- L'utilisation des graphes pour modéliser les menaces entre les reines offre une représentation visuelle claire du problème.

- L'utilisateur peut choisir la position initiale de la première reine sur n'importe quelle case de l'échiquier.

Inconvénients :

- L'algorithme explore toutes les combinaisons possibles, ce qui peut être coûteux en termes de temps et de mémoire pour des échiquiers de grande taille.

- L'utilisateur ne peut pas spécifier le nombre de dames à placer, ce qui peut limiter l'utilité de l'algorithme dans certains cas.

4. Approche avec Graphes (Deuxième version) :

Extrait de Code :

```
”””””
//la même chose mais avec des modifications telles que :
n = int(input("Entrez la taille du plateau (par exemple, pour 8x8, entrez 8): "))
while True:
        ...

resoudre_probleme(n, ligne, colonne)
”””””
```

Avantages :

- Tout comme la première version, l'utilisation de graphes offre une représentation visuelle claire du problème.

- L'utilisateur peut choisir la position de la première reine sur la première ligne de l'échiquier, offrant une certaine flexibilité.

Inconvénients :

- L'utilisateur est limité dans le choix de la position initiale de la première reine, ce qui peut réduire la flexibilité dans certains cas.

- Comme pour la première version, l'algorithme explore toutes les combinaisons possibles, ce qui peut être coûteux pour des échiquiers de grande taille.

Réponses aux questions :

Nombre de solutions pour un placement imposé de la première reine (pour un échiquier de taille 8x8) :

Pour une position imposée de la première reine, le nombre de solutions dépendra des positions restantes des autres reines sur l'échiquier. Ce nombre peut être trouvé en utilisant un algorithme de backtracking pour explorer toutes les configurations possibles à partir de cette position initiale.Nous avons trouvé 92 solutions si l'on ne choisit pas la position de cette première dame .

Existence d'autres solutions en partant d'autres positions (pour un échiquier de taille 8x8) :

Pour chaque position possible de la première reine, nous avons vérifié si les autres reines peuvent être placées sur l'échiquier sans se menacer mutuellement. C'est-à-dire en analysant les lignes, les colonnes et les diagonales.

Toutes les solutions pour des échiquiers de taille 6x6 et 8x8 (et échiquiers de taille quelconque n × n) :

Pour trouver toutes les solutions pour des échiquiers de taille 6x6 et 8x8, ainsi que pour des échiquiers de taille quelconque n × n, vous pouvez utiliser un algorithme de backtracking similaire à celui fourni dans les exemples de code, en ajustant la taille de l'échiquier en conséquence.

Construction d'un graphe représentant les cases de l'échiquier :

Pour construire un graphe où les sommets représentent les cases de l'échiquier et les arêtes représentent les contraintes entre les cases, on peut utiliser une approche de modélisation basée sur les contraintes. Ensuite, on recherche un ensemble maximal de sommets (un ensemble indépendant) où aucune reine ne se menace mutuellement.

Comparaison des performance :

Temps d'exécution avec 8 dames	Backtracking	Classes	Graphes V1	Graphes V2
	0.007sec	0.012sec	0.009	0.001

Conclusion :

Chaque approche présente ses propres avantages et inconvénients, et le choix de l'approche dépendra des exigences spécifiques du problème à résoudre, de la flexibilité nécessaire dans le choix des paramètres initiaux et de la performance souhaitée.