**SAE 2.02 : Problème des huit reines**

**Sommaire :**

1. Descriptif du problème
2. Présentation des algorithmes
3. Comparaison des performances
4. Conclusion

**Descriptif du problème :**

Le problème des huit reines consiste à placer huit reines sur un échiquier de manière qu’aucune ne puisse menacer une autre conformément aux règles des échecs. C’est-à-dire qu’aucune reine ne soit sur la même ligne, colonne ou diagonale.

Nous avons donc réalisé plusieurs algorithmes en python qui permettent de résoudre ce problème, d’afficher le temps d’exécution et d’afficher toutes les solutions si le joueur le désire.

En plus de cela, nous avons adapté les algorithmes afin qu’ils puissent fonctionner peu importe la taille de l'échiquier et donc pour placer n-reines sur ce dernier.

Nous avons codé quatre algorithmes qui abordent des approches différentes.

Le premier utilise la méthode dite “brute force”, le second utilise également cette méthode mais l’utilisateur décide de la position de la première reine. Ensuite, nous avons utilisé la méthode du backtracking. Pour finir, nous avons utilisé des graphes.

**Présentation des algorithmes :**

**Algorithme force brute :**

L'algorithme force brute explore toutes les combinaisons possibles de placements des reines sur l'échiquier, en testant chaque combinaison, il n’est donc pas du tout optimisé et par conséquent, extrêmement lent.

En effet, il commence par initialiser une liste vide pour stocker toutes les solutions.

Ensuite, il génère toutes les combinaisons possibles de positions pour les n-reines. Pour ce faire, il utilise des boucles imbriquées entre elles.

Pour chaque combinaison générée, l'algorithme vérifie alors si elle est correcte. Si c’est le cas, elle est ajoutée à la liste des solutions.

**Algorithme force brute première reine imposée :**

Cet algorithme utilise lui aussi le principe de force brute mais à la différence du premier il impose à l’utilisateur de choisir la position de la première reine. Ce qui permet d’alléger un peu l’algorithme qui reste tout de même lent et peu efficace.

**Algorithme utilisant le backtracking :**

Le deuxième algorithme utilise la logique du backtracking.

À chaque étape, l'algorithme tente de placer une reine dans une case en vérifiant si elle ne menace aucune autre reine déjà placée dans la ligne, la colonne ou la diagonale.

Si une configuration est valide, la reine est placée et le processus continue récursivement pour les reines suivantes. Si à un moment donné une configuration est impossible, l'algorithme revient en arrière pour tester d’autres combinaisons, d’où le terme de backtracking.

**Algorithme utilisant les graphes :**

Ce dernier algorithme utilise des graphes. Pour ce faire, nous avons réutilisé la classe Graphe() vu en cours que nous avons adapté au problème.

L'algorithme commence par générer un graphe représentant toutes les configurations possibles. Les positions possibles des reines sont représentées par les sommets et les mouvements possibles par les arêtes.

Ensuite, l'algorithme explore le graphe en utilisant une méthode de recherche.

À chaque étape, il examine les différentes configurations en plaçant une reine dans une case en s'assurant qu'aucune reine n’en menace une autre en se trouvant dans la même colonne, ligne ou diagonale. Il utilise une file pour gérer les configurations à explorer, en ajoutant de nouvelles configurations quand elles sont découvertes.

L'algorithme continue jusqu'à ce qu'il ait exploré toutes les combinaisons.

**Comparaison des algorithmes :**

**Performances :**

|  | **Force brute** | **Force brute contraint** | **Backtracking** | **Graphes** |
| --- | --- | --- | --- | --- |
| **8 x 8** | 0.073 s | Varie en fonction de la reine de départ | 0.0072 s | 0.0071 s |
| **10 x 10** | 8.452 s | Varie en fonction de la reine de départ | 0.170 s | 0.125 s |
| **12 x 12** | 4612.5 s  = 76.8 m | Varie en fonction de la reine de départ | 6.76 s | 6.71 s |

**Analyse** :

|  | **Force brute** | **Force brute contraint** | **Backtracking** | **Graphes** |
| --- | --- | --- | --- | --- |
| **Avantages** | Facile à coder | Plus efficace que le force brute | Stratégie efficace (moins de possibilités testées) | Le plus efficace des programmes |
| **Inconvénients** | Exécution lente | Ne donne pas toutes les solutions et reste lent | Retours en arrière coûteux | Complexe à comprendre et à coder |

**Conclusion :**

Pour conclure, on a vu que ces quatre programmes fonctionnent de manières différentes, mais aboutissent aux mêmes résultats. Pour des échiquiers de petite taille, l’écart de performances des algorithmes n’est pas très important, ils peuvent donc tous correspondre. Cependant, pour des échiquiers plus grands, la méthode de force brute rencontre ses limites et il est préférable d’opter pour les algorithmes qui utilisent les graphes ou le backtracking qui ont des performances assez similaires et largement supérieures à celles du force brute.