DS 4 d'informatique de l'année 2024/2025 - MPSI - Le problème des n dames

Consignes :

- ⋆ Les calculatrices sont interdites. Numérotez vos feuilles et faites apparaître les questions dans l'ordre du sujet.
- ⋆ Si vous repérez une erreur d'énoncé, signalez-la sur votre copie et poursuivez votre composition.

1 Introduction

Le problème des n dames consiste à placer n dames sur une grille de taille $n \times n$ en faisant en sorte qu'elles ne puissent pas s'attaquer. Dans tout l'énoncé, on considère qu'une dame attaque toutes les cases se trouvant sur la même ligne, sur la même colonne ou bien en diagonale par rapport à elle.

Les trois grilles de la figure 1 correspondent à $n = 8$. Dans la grille de gauche, les cases attaquées par la dame sont repérées par une croix. La grille du milieu est une solution au problème des 8 dames. La grille de droite n'est pas solution au problème puisque certaines dames peuvent s'attaquer en diagonale.

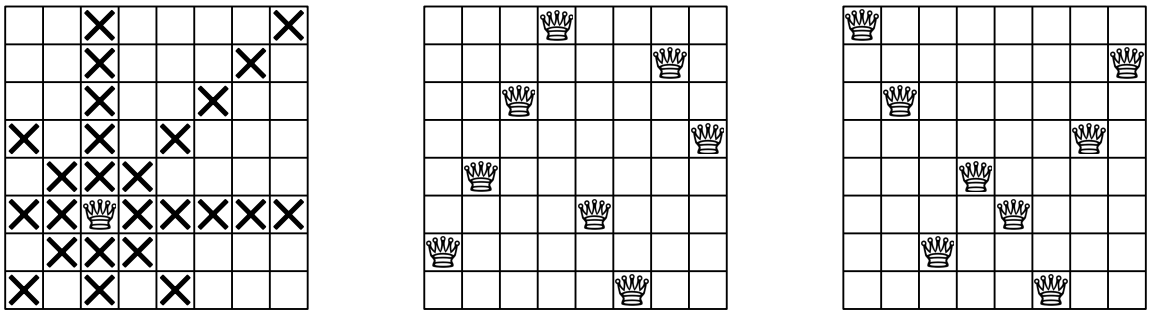


FIGURE 1

Dans le cas où $n = 1$, il y a une seule solution au problème des n dames. En effet, il suffit de placer la dame sur l'unique case de la grille pour obtenir la solution. Dans le cas où $n = 2$, il n'y a pas de solution au problème des n dames. En effet, quelle que soit la case où se trouve la première dame, elle attaque les trois autres cases. Il n'est donc pas possible de placer la seconde dame.

1. (a) Déterminer le nombre de solutions au problème des n dames pour $n = 3$. Justifier.
   (b) Donner sans justification **toutes** les solutions au problème des n dames pour $n = 4$.
   (c) Donner sans justification **une** solution au problème des n dames pour $n = 5$.

Dans la suite, on numérote les lignes de la grille par un entier $i \in [\![0; n-1]\!]$ et les colonnes par un entier $j \in [\![0; n-1]\!]$. L'indice $i = 0$ correspond à la ligne du haut, $i = n-1$ à la ligne du bas, $j = 0$ à la colonne de gauche et $j = n-1$ à la colonne de droite. On appellera case de coordonnées (i, j) la case qui se trouve sur la ligne d'indice i et la colonne d'indice j. Par exemple, la dame de la grille de gauche de la figure 1 est sur la case de coordonnées $(5, 2)$.

Dans une solution au problème des n dames, il y a nécessairement une et une seule dame sur chaque ligne. On peut donc représenter une solution potentielle par une liste `Q` de taille n telle que la dame se trouvant sur la ligne d'indice `i` est sur la case de coordonnées `(i, Q[i])`. Si on reprend les exemples de la figure 1, la grille du milieu correspond à la liste `[3, 6, 2, 7, 1, 4, 0, 5]` et celle de droite à `[0, 7, 1, 6, 3, 4, 2, 5]`.

2 Vérification d'une solution

Étant donnée une liste « `Q: list[int]` » de taille n, il y a deux conditions à vérifier pour s'assurer que `Q` représente une solution au problème des n dames : chaque colonne contient une et une seule dame et les dames ne peuvent pas s'attaquer en diagonale.

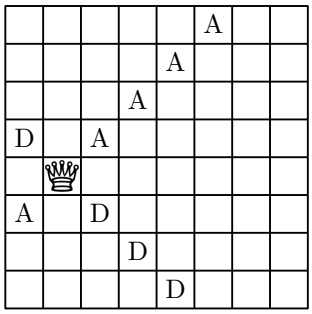
2. Écrire une fonction `testE(Q: list[int]) -> bool` qui renvoie `True` si chaque élément e de `Q` vérifie $0 \leqslant e \leqslant n-1$. Dans le cas contraire, votre fonction renverra `False`.

Dans toute la suite, on peut supposer sans le vérifier que la liste `Q` est telle que `testE(Q)` vaut `True`.

3. Le but de cette question est de tester si plusieurs dames se trouvent sur la même colonne.
   (a) Écrire une fonction `makeC(Q: list[int]) -> list[int]` qui renvoie une liste `C` de taille n telle que pour tout `j`, `C[j]` est le nombre de dames se trouvant sur la colonne d'indice `j`.
   (b) En déduire une fonction `testC(Q: list[int]) -> bool` qui renvoie `True` si sur chaque colonne de la grille, il y a exactement une dame. Dans le cas contraire, votre fonction renverra `False`.
   (c) Quelle est la complexité de la fonction `testC` ? Justifier.

Sur une grille, il y a deux types de diagonales, qu'on appellera les ***diagonales*** et les ***anti-diagonales***. Dans la grille à gauche de la figure 2, les cases qui se trouvent sur la même diagonale (resp. anti-diagonale) que la dame sont marquées par un D (resp. A).

On s'intéresse dans un premier temps aux anti-diagonales. Pour chaque case de la grille, calculons la somme $i+j$ où (i, j) sont les coordonnées de la case. Par exemple, pour $n = 8$, on obtient les valeurs indiquées à droite de la figure 2. On remarque alors que la case de coordonnées (i_1, j_1) et la case de coordonnées (i_2, j_2) sont sur la même anti-diagonale si et seulement si $i_1 + j_1 = i_2 + j_2$.



0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14

FIGURE 2

4. Écrire une fonction `testAD(Q: list[int]) -> bool` qui renvoie `True` si sur chaque anti-diagonale de la grille, il y a au plus une dame. Dans le cas contraire, votre fonction renverra `False`. Cette fonction devra être de complexité linéaire en n.
5. Passons maintenant aux diagonales de la grille.
   (a) Considérons deux cases de coordonnées (i_1, j_1) et (i_2, j_2). Trouver une condition nécessaire et suffisante sur i_1, j_1, i_2, j_2 pour que les deux cases soient sur la même diagonale. On attend une justification similaire à la justification donnée par l'énoncé dans le cas des anti-diagonales.
   (b) Écrire une fonction **récursive** `testD(Q: list[int]) -> bool` qui renvoie `True` si sur chaque diagonale de la grille, il y a au plus une dame. Dans le cas contraire, votre fonction renverra `False`. Votre fonction devra être une fonction **récursive** et de complexité linéaire en n.
6. (a) Écrire une fonction `estSol(Q: list[int]) -> bool` qui indique si la configuration donnée en entrée correspond à une solution du problème des n dames.
   (b) Quelle est la complexité de votre fonction ? Justifier.

3 Calcul d'une solution

Cette partie présente deux techniques permettant de trouver une solution au problème des n dames.

3.1 Génération aléatoire

Une première stratégie pour résoudre le problème des n dames consiste à générer aléatoirement une configuration `Q` puis à vérifier si c'est une solution à l'aide de la fonction `estSol`. Tant que `Q` n'est pas solution, on recommence en générant une nouvelle configuration aléatoire.

Dans un souci d'optimisation, on fera en sorte qu'il y ait une et une seule dame sur chaque colonne de la grille. En d'autres termes, on souhaite générer une liste de taille n dans laquelle chaque élément de $[\![0; n-1]\!]$ apparaît exactement une fois. Pour cela, on va utiliser le ***mélange de Fisher-Yates*** dont le principe est le suivant :

→ On crée la liste `Q = [0, 1, ..., n-2, n-1]`.

→ Pour `i` variant de `n-1` à `1` :

- Soit `j` un entier aléatoire de $[\![0\,;\,i]\!]$
- On échange les éléments `Q[i]` et `Q[j]`.

Pour tirer aléatoirement l'entier `j`, on utilisera la commande `random.randint(a,b)` qui renvoie un entier aléatoire `c` tel que $a \leqslant c \leqslant b$. On pourra supposer sans le réécrire que le module `random` a été importé :

```
import random
```

7. Écrire une fonction `FY(n: int) -> list[int]` qui renvoie une configuration aléatoire générée à l'aide du mélange de Fisher-Yates.
8. En déduire une fonction `solAlea(n: int) -> list[int]` qui renvoie une solution au problème des `n` dames. Si `n < 0` ou bien si $n \in \{2, 3\}$, votre fonction déclenchera une erreur.

3.2 Algorithme de réparation itérative

La fonction `solAlea` obtenue dans la question 8 n'est pas satisfaisante car elle est très lente. Par exemple, il lui faut plusieurs secondes pour calculer une solution lorsque $n = 15$. À la place, on peut utiliser un ***algorithme de réparation itérative***. L'idée est de partir d'une configuration quelconque (par exemple d'une configuration aléatoire) et de modifier la position d'une dame dans le but de diminuer le nombre de dames attaquées. On recommence ensuite l'opération tant que la configuration n'est pas une solution.

Pour mettre en place cet algorithme, on calcule pour chaque case de la grille le nombre de dames qui attaquent la case en question. La figure 3 présente deux exemples de grilles. Les dames y sont repérées par des cercles et chaque case contient le nombre de dames qui l'attaquent. Par convention, une dame attaque la case où elle se trouve. La grille de gauche correspond à la configuration de droite de la figure 1.

①	2	2	4	2	2	3	3
5	3	3	2	3	2	3	②
3	③	3	2	2	4	4	4
3	3	5	3	3	4	②	2
3	2	3	③	5	4	2	3
2	3	3	5	④	3	2	3
2	3	②	4	3	5	4	2
3	4	4	3	2	①	5	3

①	2	2	3	3	2	2	4
5	3	3	2	2	3	2	③
3	③	3	2	2	3	4	4
3	3	5	3	3	4	2	②
3	2	3	③	5	3	2	3
2	3	3	5	③	4	1	4
2	3	②	3	4	5	3	3
3	4	3	4	2	①	4	4

FIGURE 3

Dans la suite, on utilisera une liste de listes d'entiers notée `A` telle que pour tout $i \in [\![0\,;\,n-1]\!]$ et tout $j \in [\![0\,;\,n-1]\!]$, l'entier `A[i][j]` est le nombre de dames qui attaquent la case de coordonnées `(i,j)`. Voici le principe de l'algorithme de réparation itérative :

(Étape 1) On génère une liste `Q` à l'aide de la fonction `FY` de la question 7.

(Étape 2) (a) Si `Q` est une solution au problème des n dames, on renvoie cette solution.

(b) Sinon, on choisit une dame au hasard parmi celles attaquées et on la déplace sur une nouvelle case de la même ligne. On choisira la case la moins attaquée différente de la case actuelle de la dame.

(Étape 3) On recommence l'étape 2 au maximum 1000 fois. Si après ces 1000 essais, on n'a toujours pas trouvé de solution, on recommence l'algorithme depuis le début, à l'étape 1.

Commençons par donner deux exemples pour l'étape 2b. Avec l'exemple de gauche de la figure 3, il faut choisir une dame au hasard parmi celles qui sont attaquées. Les dames qui ne sont pas attaquées sont celles se trouvant sur les lignes d'indices $i = 0$ et $i = 7$. Il s'agit donc dans un premier temps de choisir aléatoirement un indice $i \in \{1, 2, 3, 4, 5, 6\}$. Par exemple, si on choisit $i = 3$, alors la dame en coordonnées $(3, 6)$ est déplacée vers la case $(3, 7)$ puisque c'est la case la moins attaquée de la ligne. On obtient alors

la configuration à droite de la figure 3. Appliquons maintenant l'étape 2b à cette nouvelle configuration. Sous l'hypothèse que la ligne sélectionnée est celle d'indice $i = 1$, alors la dame est déplacée vers la case de coordonnées $(1, 3)$ ou $(1, 4)$ ou $(1, 6)$. Le choix entre ces trois cases se fait de manière arbitraire.

9. Le but de cette question est d'implémenter l'étape 2b. On suppose que les deux listes `Q` et `A` ont été créées précédemment.
   (a) Écrire une fonction `dameABouger(Q: list[int], A: list[list[int]]) -> int` qui choisit au hasard une dame parmi les dames attaquées. La fonction renvoie l'indice de la ligne où se trouve la dame. On utilisera la fonction `random.randint` décrite dans la partie 3.1.
   (b) Écrire une fonction `nvPos(i: int, A: list[list[int]]) -> (int, int)` qui renvoie deux indices `j1` et `j2` correspondant aux deux cases les moins attaquées de la ligne d'indice `i`. Ainsi, la case `(i,j1)` devra être moins attaquée que la case `(i,j2)` qui devra être moins attaquée que toutes les autres cases de la ligne `i`. Si plusieurs choix sont valides, on fera en sorte que `j1` et `j2` soient les plus petits possibles. Votre fonction devra être de complexité linéaire en n. On pourra supposer que $n \geqslant 2$.

   Remarque : dans la fonction `nvPos`, il est nécessaire de renvoyer deux indices car si la dame se trouve sur la case la moins attaquée de la ligne, elle doit quand même être déplacée.
   (c) Écrire une fonction `etape2b(Q: list[int], A : list[list[int]]) -> list[int]` La liste en sortie représente la configuration obtenue après avoir appliqué l'étape 2b à la configuration `Q`.
10. Écrire une fonction `makeA(Q: list[int]) -> list[list[int]]` qui renvoie la liste `A`. Votre fonction devra être de complexité quadratique en n.
11. Écrire une fonction `solRI(n: int) -> list[int]` qui résout le problème des n dames à l'aide de l'algorithme de réparation itérative. Cette fonction doit donc appliquer les étapes 1, 2 et 3 décrites ci-dessus.

4 Nombre de solutions au problème des n dames

Afin de compter le nombre de solutions au problème des n dames, on adopte une stratégie récursive. L'idée est de placer les dames une à une. On parlera de "la dame numéro `i`" pour faire référence à la dame que l'on souhaite placer sur la ligne d'indice `i`.

Supposons que les dames des `i` premières lignes aient déjà été placées et qu'elles ne se menacent pas mutuellement. Alors :

→ On définit `L` une liste contenant les indices des colonnes que la dame numéro `i` peut occuper sans être menacée.

→ Pour chaque élément `j` dans `L` :

- On place la dame numéro `i` sur la colonne d'indice `j`.
- On calcule récursivement le nombre de solutions possibles au problème des n dames en plaçant les dames restantes.

→ Le nombre de solutions est alors la somme du nombre de solutions obtenues dans les `len(L)` appels récursifs effectués au point précédent.

12. Écrire une fonction `nbSol(n: int) -> int` qui calcule le nombre de solutions au problème des n dames en utilisant la stratégie décrite ci-dessus.