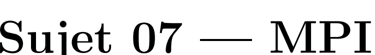

Sujet 07 — MPI

Le problème des 5 reines

Durée : 3h

Centrale-Supélec

Préambule

Ce sujet d'oral d'informatique est à traiter, sauf mention contraire, en respectant l'ordre du document. Votre examinatrice ou votre examinateur peut vous proposer en cours d'épreuve de traiter une autre partie, afin d'évaluer au mieux vos compétences.

Le sujet comporte plusieurs types de questions. Les questions sont différenciées par une icône au début de leur intitulé :

- les questions marquées avec ⌨ nécessitent d'écrire un programme dans le langage demandé. Le jury sera attentif à la clarté du style de programmation, à la qualité du code produit et au fait qu'il compile et s'exécute correctement ;
- les questions marquées avec 🕴 sont des questions à préparer pour présenter la réponse à l'oral lors d'un passage de l'examinatrice ou l'examinateur. Sauf indication contraire, elles ne nécessitent pas d'appeler immédiatement l'examinatrice ou l'examinateur. Une fois la réponse préparée, vous pouvez aborder les questions suivantes ;
- les questions marquées avec ✍ sont à rédiger sur une feuille, qui sera remise au jury en fin d'épreuve.

Votre examinatrice ou votre examinateur effectuera au cours de l'épreuve des passages fréquents pour suivre votre avancement. En cas de besoin, vous pouvez signaler que vous sollicitez explicitement son passage. Cette demande sera satisfaite en tenant également compte des contraintes d'évaluation des autres candidates et candidats.

APPENDICE.

Un de nos anciens amis, Mr de R*** a bien voulu enrichir notre ouvrage d'un mémoire sur le *problème des cinq dames*, dont nous joignons ici la traduction*). On a vu, dans notre premier volume, pag. 122—135, qu'il y a, en tout, 92 manières différentes de placer huit dames sur l'échiquier, de façon qu'elles en attaquent toutes les cases, hormis celles qu'elles remplissent elles-mêmes. C'est là, évidemment, le *maximum* de dames satisfaisant à la condition décrite. Or divers amateurs d'échecs ont signalé, depuis, l'existence d'un *minimum* correspondant. Car ils ont remarqué que *cinq* dames suffisaient pour tenir en échec les 59 cases restantes de l'échiquier, tout en l'occupant de manière à ne point s'entre-attaquer. Mais cette remarque n'ayant été appuyée que de deux ou trois exemples particuliers, il restait à découvrir et à discuter le total des solutions du problème ainsi modifié. C'est ce qu'a fait notre savant ami, dont le mémoire indique, en outre, l'application des mêmes idées aux échiquiers carrés moindres que celui de 64 cases. Le problème en question, quoique mathématique de sa nature, est, d'ailleurs, à tel point rebelle au calcul, que les méthodes connues ne fournissent même pas le moyen de *l'énoncer analytiquement*. Cette circonstance rehaussera, sans doute, aux yeux des géomètres, l'intérêt des résultats auxquels Mr de R*** est parvenu par des essais systématiques très-pénibles.

*) Du consentement de l'auteur, nous y avons donné plus de développement aux considérations qui se rattachent à la géométrie de situation. Nous n'avons eu, en cela, que le but de relier cet appendice aux passages des deux premiers volumes où se trouvent exposés les principes généraux des considérations dont il s'agit.

Figure 1 – Extrait du traité *Applications de l'analyse mathématique au jeu des échecs*, von Jaenisch, 1862.

1 Introduction

On s'intéresse dans ce sujet à une variante du problème des 8 reines, dans lequel on cherche à placer le plus petit nombre possible de reines sans prises sur un échiquier $n \times n$ pour en surveiller toutes les cases.

Ce sujet, à réaliser entièrement dans le langage OCaml, comporte cinq parties qui ne sont pas indépendantes et qui doivent être abordées dans l'ordre.

1.1 Contexte

Le problème des n reines a été proposé par Max Bezzel, un joueur d'échecs allemand, en 1848 pour l'échiquier standard 8×8 puis généralisé par François-Joseph Eustache Lionnet en 1869 pour un échiquier $n \times n$. La question est de savoir s'il est possible de placer n reines sur un échiquier de taille $n \times n$ sans qu'aucune reine ne soit *en prise*, c'est-à-dire ne soit attaquée par une autre. Carl Friedrich Gauss (1777-1855) étudia le problème et le résolut partiellement en proposant 72 solutions. La première solution complète fut proposée en 1850 par Franz Nauck : il existe 92 dispositions convenables pour l'échiquier 8×8. Une telle solution est proposée à la figure 2. Remarquons que dans toute solution, toutes les cases sont *surveillées* par au moins une reine.

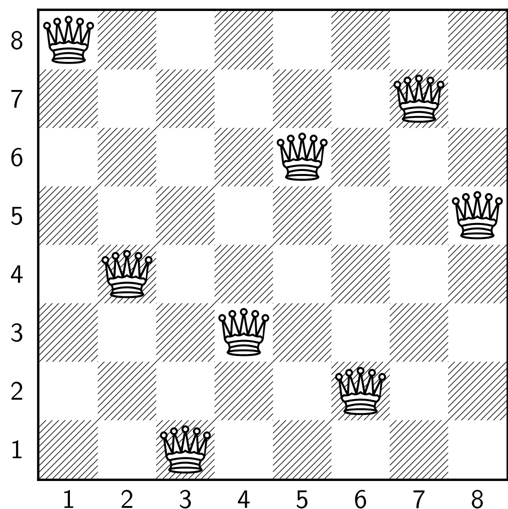

FIGURE 2 – Une solution au problème des 8 reines.

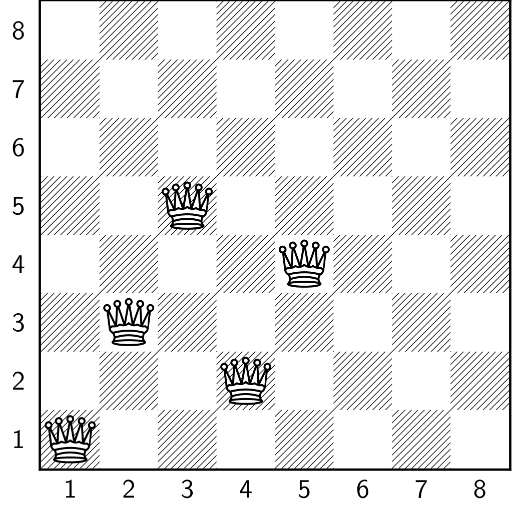

FIGURE 3 – Une solution au problème des 5 reines.

Dans son traité sur les applications de l'analyse mathématique au jeu des échecs (figure 1), Karl Ferdinand VON JAENISCH, un théoricien d'échecs russe-finois, étudie une variante où l'on s'intéresse non plus au nombre *maximal* de reines sans prises que l'on peut placer, mais au nombre *minimal* de reines sans prises que l'on peut placer tout en surveillant toutes les cases de l'échiquier.

1.2 Formalisation

On considère un échiquier de taille $n \times n$ avec $n \geq 1$ et on cherche le nombre minimal de reines que l'on peut placer sur l'échiquier de telle manière à ce qu'aucune reine ne soit en prise avec une autre et telle que toute case soit *couverte* par au moins une reine. Une case est *couverte* par une reine si elle se trouve sur la même ligne, la même colonne ou

une même diagonale. Une reine est en prise avec une autre si elle se trouve sur une case couverte par cette autre reine. La figure 3 propose une solution pour un échiquier de taille 8×8 avec 5 reines, ce qui est une solution optimale.

▷ **Question 1.** ✍ Donner une solution avec une reine pour $n = 3$ et avec trois reines pour $n = 4$.

▷ **Question 2.** ✍ On admet que pour $n \geq 4$, il est toujours possible de placer n reines sans prises sur un échiquier de taille $n \times n$ (c'est le problème classique des n-reines). Montrer que le problème abordé dans ce sujet admet toujours une solution optimale et majorer simplement la valeur de cette solution.

▷ **Question 3.** ✍ Proposer un minorant pour la valeur d'une solution optimale.

2 Recherche des solutions optimales

Le fichier OCaml `reines.ml` propose un certain nombre de fonctions déjà implémentées et est à compléter. Attention, la fonction `couverture_par_reines` comporte des erreurs de syntaxe et de type, qu'il sera demandé de corriger, elle est donc commentée.

On modélise une couverture (partielle ou complète) de l'échiquier par des reines sans prises par un enregistrement en OCaml. La matrice `m` indique, pour chaque case de coordonnées (i, j), le nombre de reines qui couvrent cette case. Le champ `z` sera introduit et utilisé plus tard. Remarquons qu'une case contenant une reine a nécessairement pour valeur 1.

```
type couverture = {m : int array array; mutable z : int}
```

La fonction `Array.make_matrix` dont on pourra consulter la documentation permet d'initialiser une matrice en OCaml. On peut définir un enregistrement avec la même syntaxe que ci-dessus avec des « = » pour spécifier la valeur des champs. Si `c : couverture` est un enregistrement, on peut accéder à la valeur d'un champ à l'aide d'un point « `.` » et modifier un champ mutable avec une flèche « `<-` ». Par exemple :

```
# let c = {m = Array.make_matrix 8 8 0; z = 0};;
val c : couverture = {m = [|...|]; z = 0}
# c.z <- c.z + 1;;
- : unit = ()
# c.z;;
- : int = 1
```

La couverture de la figure 3 correspond à la matrice :

```
[|
  [|2; 1; 1; 1; 1; 1; 1; 1|];
  [|2; 2; 1; 1; 2; 1; 1; 1|];
  [|1; 2; 2; 2; 2; 1; 1; 1|];
  [|3; 2; 1; 4; 3; 2; 2; 1|];
  [|3; 4; 3; 4; 1; 2; 1; 1|];
  [|3; 1; 4; 3; 4; 2; 1; 1|];
  [|3; 3; 4; 1; 2; 2; 2; 1|];
  [|1; 3; 3; 3; 3; 1; 2; 2|]
|]
```

Une couverture sans aucune reine placée correspond donc à une matrice `m` dont toutes les cases sont à 0.

▷ **Question 4.** 🖮 Implémenter la fonction `finale : couverture -> bool` qui s'évalue à `true` si une couverture est complète, c'est-à-dire si toutes les cases de l'échiquier sont couvertes par au moins une reine, et à `false` sinon.

▷ **Question 5.** 🖮 Implémenter la fonction

```
mise_a_jour : couverture -> int -> int -> int -> unit
```

L'appel `mise_a_jour c i j v` doit mettre à jour la couverture `c` suite au placement (si `v = 1`) où à la suppression (si `v = -1`) d'une reine sur une case de coordonnées `(i, j)`. *Il est fortement recommandé de bien tester cette fonction sur des exemples en ajoutant/supprimant successivement des reines sur un échiquier initialement vide. On pourra adapter la taille de l'échiquier lors des tests.*

2.1 Recherche d'une solution optimale

On propose une approche de type *séparation et évaluation* (*branch and bound* en anglais) pour résoudre le problème introduit dans ce sujet.

▷ **Question 6.** 🕴 Rappeler le principe de la méthode par séparation et évaluation.

Une fonction `couverture_par_reines : int -> int` permettant de résoudre ce problème vous est proposée. Cette fonction comporte cependant un certain nombre d'erreurs de syntaxe et de type.

▷ **Question 7.** 🖮 Décommenter cette fonction et en corriger les erreurs de syntaxe et de type.

▷ **Question 8.** ✍ Donner pour $n \in \{5, 6, 7, 8, 9\}$ le nombre minimal de reines sans prises nécessaires pour couvrir l'échiquier.

▷ **Question 9.** 🖮🕴 En remarquant qu'il n'est pas utile de poursuivre l'exploration lorsque l'on est sûr que l'on ne peut pas obtenir de meilleure solution, proposer un élagage simple et le mettre en œuvre.

On représente une solution par la liste de type `(int * int) list` des coordonnées des placements des reines.

On pourra utiliser la fonction `affiche_reines : int -> (int * int) list -> unit` qui prend en entrée un entier n et une liste de coordonées des positions des reines pour en réaliser un affichage simple.

▷ **Question 10.** 🖮 Modifier la fonction `couverture_par_reines` pour obtenir également en sortie les positions des reines d'une solution optimale.

▷ **Question 11.** 🕴 Afficher une telle solution pour $n = 9$ et sauvegarder ce résultat.

En l'état, la fonction `finale` doit reparcourir entièrement la matrice pour vérifier si une couverture est complète, pour une complexité en $\mathcal{O}(n^2)$. On se propose désormais de maintenir l'invariant suivant pour une couverture : « le champ `z` correspond au nombre de cases non couvertes de l'échiquier ».

▷ **Question 12.** 🖮 Modifier la fonction `mise_a_jour` pour maintenir cet invariant. Proposer une nouvelle fonction `finale` dont la complexité doit être en $\mathcal{O}(1)$. Vérifier le bon fonctionnement de votre programme.

2.2 Nombre de solutions optimales

Il est conseillé de bien sauvegarder votre travail avant d'apporter des modifications à la fonction `couverture_par_reines` pour pouvoir présenter votre travail à votre examinatrice ou examinateur.

▷ **Question 13.** ⌨ Modifier la fonction `couverture_par_reines` pour obtenir le nombre de solutions optimales.

▷ **Question 14.** 🕴 Afficher pour chaque $n \in [\![1, 10]\!]$, le nombre de reines d'une solution optimale et le nombre de solutions optimales différentes. On compilera le programme avec le compilateur `ocamlopt`.

3 Une approche probabiliste

L'approche précédente est bien trop longue pour espérer un résultat pour de grandes valeurs de n. Dans cette section on propose une approche probabiliste pour trouver une solution approchée en temps raisonnable.

▷ **Question 15.** 🕴 Rappeler la différence entre un algorithme probabiliste de type Monte Carlo et un algorithme de type Las Vegas.

On considère le procédé suivant pour choisir de manière uniforme une valeur parmi une suite $x_0, x_1, \ldots x_{m-1}$ en une seule passe (m n'étant pas connu *a priori*). On commence par retenir la valeur $x := x_0$, puis, pour chaque $i \in [\![1, m-1]\!]$, dans l'ordre, on met à jour cette valeur ($x := x_i$) avec probabilité p_i ou on conserve la valeur de x avec une probabilité $1 - p_i$.

▷ **Question 16.** ✍ Déterminer la probabilité p_i pour que la valeur x à l'issu de ce procédé soit choisie de manière uniforme parmi les valeurs de la suite. *On pourra commencer par étudier les cas* $m \in \{1, 2, 3\}$.

▷ **Question 17.** 🕴 Expliquer comment adapter ce procédé, toujours en une seule passe, dans le cas où l'on dispose d'un prédicat $\mathcal{P}$ et que l'on souhaite tirer uniformément une valeur parmi celles qui vérifient le prédicat $\mathcal{P}$, c'est-à-dire choisir une valeur uniformément parmi les valeurs $\{x_i \,|\, i \in [\![0, m-1]\!], \mathcal{P}(x_i)\}$ ou la valeur particulière $\perp$ si aucun élément ne convient.

On rappelle que l'appel `Random.int r` permet d'obtenir une valeur tirée uniformément dans $[\![0, r-1]\!]$ et que l'on peut supposer que les tirages sont indépendants. On pourra consulter la documentation du module `Random`.

▷ **Question 18.** ⌨ Implémenter la fonction `choix_position : int array -> int` telle que `choix_position` choisit uniformément un indice du tableau parmi les indices des valeurs nulles ou -1 si tous les éléments sont non nuls. On effectuera un unique parcours du tableau.

On propose l'approche probabiliste suivante pour trouver une solution, non nécessairement optimale pour $n \geq 2$. On effectue un certain nombre k d'essais et on conserve la meilleure solution trouvée jusqu'à présent. Pour chaque essai, on part d'une couverture initialement vide et pour chaque ligne, dans l'ordre, avec probabilité p on ajoute une

reine sur une case non couverte choisie uniformément parmi les cases non couvertes, s'il y en a une, avant de passer à la ligne suivante. Si on obtient ainsi une solution, on met à jour la meilleure solution trouvée si nécessaire. Notons que comme précédemment, on peut parfois interrompre prématurément la recherche si on sait que l'on ne pourra pas améliorer la meilleure solution trouvée jusqu'à présent.

Dans un premier temps on prendra $p = \frac{n-1}{n}$.

▷ **Question 19.** ⌨ Implémenter la fonction `couverture_par_reines_aleatoire` de type `int -> int -> int` qui implémente cette stratégie. *Il est vivement conseillé d'afficher des informations intermédiaires pour faciliter le débogage.*

▷ **Question 20.** ✍ En effectuant $k = 10^6$ essais, donner un majorant du nombre de reines nécessaires pour couvrir un échiquier $n \times n$ avec $n \in \{20, 30, 100\}$.

▷ **Question 21.** �✋ Quelle est la complexité de cette approche ?

▷ **Question 22.** 🖐 Proposer un autre choix pertinent pour p. Cette probabilité pourra dépendre de la meilleure solution trouvée jusqu'à présent et du nombre de reines déjà placées.

▷ **Question 23.** 🖐 Déterminer la proportion d'essais qui se soldent par un échec, par un élagage précoce ou par une mise à jour de la meilleure valeur trouvée. Commenter.

4 Parallélisation

Dans cette partie on se propose de paralléliser le programme précédent pour accélérer la recherche. Pour cela on va utiliser les modules `Thread` et `Mutex`. On pourra consulter la documentation de ces modules.

Les fils d'exécution sont de type `Thread.t`. Les verrous sont de type `Mutex.t`. On dispose en particulier des fonctions suivantes :

- `Thread.create : ('a -> 'b) -> 'a -> Thread.t` qui prend en entrée une fonction `f : 'a -> 'b` et un paramètre `x`, et qui créé un nouveau fil qui commence immédiatement à exécuter l'appel `f x`. Cette fonction renvoie un identifiant de type `Thread.t` qui permet ensuite de faire référence au nouveau fil. La valeur de retour de `f` n'est pas utilisée ;
- `Tread.join : Thread.t -> unit` qui prend l'identifiant d'un fil en entrée et attend que ce fil termine son exécution ;
- `Mutex.create : unit -> Mutex.t` qui permet de créer un nouveau mutex ;
- `Mutex.lock : Mutex.t -> unit` qui permet d'attendre qu'un mutex soit libre puis de le verrouiller ;
- `Mutex.unlock : Mutex.t -> unit` qui permet de relâcher un mutex.

Il sera désormais impérativement nécessaire de travailler avec un exécutable compilé et non en mode interactif. Pour compiler un programme utilisant ces modules on utilisera :

```
ocamlopt -I +threads unix.cmxa threads.cmxa reines.ml -o reines
```

En OCaml, le module `Thread` ne permet pas réellement d'exécuter plusieurs fils en *parallèle*. Il sera donc normal de ne pas observer d'accélération des performances en pratique, voire même une dégradation.

▷ **Question 24.** ⌨ Proposer une version parallélisée du programme de la partie précédente.

▷ **Question 25.** 🕴 Expliquer quelles sont les ressources partagées entre les différents fils et ce que vous avez mis en place pour garantir un accès concurrent à ces ressources.

▷ **Question 26.** 🕴 Illustrer le déroulement de votre programme avec quelques affichages bien choisis.

5 Une variante

Cette dernière partie ne doit être abordée que si vous avez terminé toutes les parties précédentes.

On considère maintenant une autre variante du problème de la couverture où l'on n'impose plus que le placement des reines soit sans prises. On cherche donc simplement le plus petit nombre de reines qui permettent de surveiller l'échiquier, sans contraintes supplémentaires.

▷ **Question 27.** ✍ Déterminer le nombre minimal de reines nécessaires pour $n = 4$.

▷ **Question 28.** ⌨ Déterminer le nombre minimal de reines nécessaires pour des valeurs de n aussi grandes que possible.