# Cours LIFI - 2004

TD n°3

12 octobre 2004

## 1 Le problème des 8 reines

On veut résoudre le problème d'échecs suivant: étant donné huits reines, comment les placer sur un échiquier $8 \times 8$ de manière à ce qu'aucune des reines ne soit en prise avec une autre. Deux reines sont en prise si elles sont sur la même ligne, ou la même colonne, ou la même diagonale. Pour information, il y a 92 solutions possibles dont 12 seulement si on regroupe les solutions qui sont des rotations ou des symmétries les unes des autres.

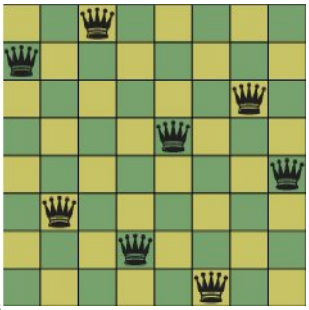

Figure 1: Un exemple de solution aux problèmes des 8 reines.

On commence par généraliser le problème: on cherche à placer $N$ reines sur un échiquier de $N \times N$. On crée dábord une classe pour tester si deux reines sont en prise. Voici son squelette :

```
class Queen {
    private int i,j;
    public Queen(int i,int j) {
        this.i = i;
        this.j = j;
    }
    public boolean estEnPriseAvec(Queen q) {
        // ... compléter ...
    }
}
```

10

**Question 1 (Une classe `Queen`).** *Compléter la méthode* `Queen::estEnPriseAvec()`.

**Question 2 (Tester une solution).** *Écrire une méthode de classe* `neSontPasEnPrise()` *qui prend en paramètre un vecteur de* `Queen` *et renvoie vrai si et seulement si aucune des reines n'est en prise avec un autre.*

Une approche pour résoudre le problème est de construire tous les positionnements possibles de 8 reines sur le plateau et de les tester avec `neSontPasEnPrise()` jusqu'à trouver une solution. Cette approche est *brute force*. On peut-être plus malin en remarquant qu'il n'est pas possible que deux reines soient sur la même colonne. Il doit donc y avoir exactement une reine dans chaque colonne. Il suffit donc de ne tester que les positionnements avec une reine par colonne.

**Question 3 (Analyse en complexité).** *Trouver la formule qui donne le nombre de positionnements à tester dans les 2 approches en fonction de $N$.*

On va donc représenter un positionnement de $N$ reines par un vecteur de $N$ entiers chacun compris entre 0 et $N-1$. Par exemple, le positionnement de la figure 1 est codé par $[6, 2, 7, 1, 4, 0, 5, 3]$. Pour cela on crée la classe suivante :

```
class Layout {
    private int[ ] rows;
    public Layout(int n) {
        rows = new int[n];
        for (int i=0;i<rows.length;++i) rows[i] = -1;
    }
    public int size() { return rows.length; }
    public String toString() {
        String x="";
        for (int i=0;i<rows.length;++i) x += rows[i]+" ";                    10
        return x;
    }
    public int getRow(int i) { return rows[i]; }
    public void setRow(int i,int j) { rows[i] = j; }
    public void print()          { // ... α compléter ... }
    public boolean isValid() { // ... α compléter ... }
}
```

**Question 4 (Tester un positionnement).** *Compléter la méthode* `Layout::isValid()` *qui teste si un positionnement est une solution au problème. On pensera à utiliser la classe* `Queen`*.*

**Question 5 (Afficher l'équichier).** *Compléter la méthode* `Layout::print()` *qui affiche un échiquier en mode texte avec un "." par case vide et un Q par case contenant une reine. Par exemple pour $N = 4$ :*

```
. Q . .
. . . Q
Q . . .
. . Q .
```

Pour énumérer tous les positionnements, on va utiliser une méthode dite de *backtracking*. Les positionnements possibles peuvent se représenter comme un arbre de hauteur $N$ avec une seule racine. À chaque niveau correspond la décision de placer la reine suivante. Chaque branche se divise donc en $N$ sous-branches. On a $N^N$ feuilles qui représentent tous les positionnements possibles. Le *backtracking* consiste à explorer l'arbre en profondeur d'abord. Quand on arrive sur une feuille, on teste si le positionnement correspondant est valide. Si oui, on a trouvé une solution et on s'arrête. Si non, on revient (*backtrace* en anglais) dans l'arbre jusqu'au dernier embranchement qui n'a pas été complètement visité.

**Question 6 (Backtracking).** *Écrire un méthode de classe* `backtrace(Layout b,int i)` *qui prend un positionnement (on suppose que les $i$ premières reines sont déjà fixées) et explore toutes les variations pour les $N-i$ reines restantes. La fonction renverra vrai si parmi tous ces positionnements il y en a un valide et faux sinon. Écrire un programme principal qui cherche une solution et l'affiche.*