Devoir à la maison pour le stage « Algorithmique » : *backtracking*

1 Les reines sur l'échiquier

Le problème est, classiquement, de placer n reines sur un échiquier carré $n \times n$ sans qu'elles se menacent les unes les autres. Rappelons qu'une reine menace toutes les cases situées sur la même ligne, la même colonne ou une même diagonale qu'elle. Voici une démarche possible.

Codage d'une position
On numérote les lignes et les colonnes par un entier entre 0 et $n-1$. Comme les listes de Xcas et Python sont naturellement numérotées à partir de zéro, c'est plus naturel que de coder les rangées entre 1 et n.
On représentera un échiquier par une liste `L` de n entiers compris entre -1 et $n-1$. Si le k^{e} élément de la liste est un entier ℓ compris entre 0 et $n-1$, cela signifie qu'il y a une reine sur la ligne ℓ dans la colonne k ; s'il vaut -1, cela signifie que la reine de la colonne k n'a pas encore été placée.
Par exemple, les listes `[1,-1,0,-1]` et `[1,3,0,2]` désignent les échiquiers suivants, où D désigne l'emplacement des reines :

		D		ligne 0
D				ligne 1
				ligne 2
				ligne 3

		D		ligne 0
D				ligne 1
			D	ligne 2
	D			ligne 3

Principe
On remplit l'échiquier colonne par colonne, de gauche à droite. Au départ, il n'y a aucune reine sur l'échiquier, on part de la liste `[-1,...,-1]`. Supposons avoir rempli les $k-1$ premières colonnes. On parcourt toutes les cases de la colonnes k et on regarde s'il est possible d'y placer une reine compte tenues des colonnes précédentes.

Des détails
– Que fait la fonction ci-dessous ?

Python

```
def valide(L,ligne, col) :
    ok=True
    for d in range(1,ligne+1) :
        if L[ligne-d]==col or L[ligne-d]==col-d or L[ligne-d]==col+d :
            ok=False
            break
    return ok
```

– La fonction suivante détermine tous les échiquiers de taille 4×4. Proposer une variante `dame(n)`, par exemple récursive, prenant en paramètre la taille `n` de l'échiquier.

Python

```
def dame() :
    L=[-1]*4
    for a in range(4) :
        L[0]=a
        for b in range(4) :
            if valide(L,1,b) : L[1]=b
            else : continue
            for c in range(4) :
                if valide(L,2,c) : L[2]=c
                else : continue
                for d in range(4) :
                    if valide(L,3,d) :
                        L[3]=d
                        print L
                    else :continue
dame()
```

Une variante

En procédant comme précédemment, on teste n positions pour chaque colonne, ce qui fait un nombre de tests quadratique en n. On peut diminuer le nombre de tests de la façon suivante. On représente un échiquier par une liste de n listes : la k^{e} liste désigne l'ensemble des positions possibles pour la reine de la colonne k, compte tenu des $k-1$ colonnes précédentes. Au départ, toutes les listes de l'échiquier sont donc `[1,2,...,n]`. À la fin, une solution est constituée de n listes à 1 élément, par exemple `[[1],[3],[0],[2]]` (pour $n=4$). Le prix à payer est que l'on manipule des données plus grosses.

- Écrire une fonction `purge(L,k,l)`, où `L` est une liste de listes, `k` un indice de colonne, `l` un indice de ligne. La fonction doit remplacer `L[k]` par `[k]` et supprimer, dans les listes `L[k+1]`,..., `L[n-1]`, les indices des lignes qu'interdit une reine en position (`k`,`l`).
  Par exemple, `purge([[2],[0],[1,3],[0,1,3]],0)` doit renvoyer `[[2],[0],[3],[1,3]]`.
- Écrire une fonction `resoudre(L,k)`, où `k` un indice de colonne et `L` est une liste de listes désignant un échiquier partiellement rempli jusqu'à la colonne `k-1`. Si `k`=`n`, la fonction renvoie `L`. Sinon, pour chaque valeur `l` de `L[k]`, on `purge` l'échiquier, ce qui produit un nouvel échiquier `M` et on collecte les échiquiers `resoudre(M,k+1)`.
- Comparer, au moins expérimentalement, le nombre de tests à effectuer par chacun des algorithmes pour trouver toutes les solutions du problème.

Une autre approche

Introduisons un graphe : les sommets sont les cases de l'échiquier ; il y a une arête entre deux cases si deux reines placées sur ces cases se menacent mutuellement. Comment le problème se traduit-il en termes de graphe ?

NB : une version en ligne : `http://math.univ-lyon1.fr/~germoni/reines/`.
NB : un problème dans le même esprit : trouver une façon pour un cavalier d'un jeu d'échec pour parcourir toutes les cases de l'échiquier sans jamais repasser deux fois au même endroit.

2 Résolveur de sudoku

Il s'agit de programmer un résolveur de sudoku.

Approche naïve
Dans cette approche, on code une grille partiellement remplie par une matrice 9×9, ou plus généralement [1] $n \times n$, dont les éléments sont les entiers de 1 à 9 (resp. n) si la case correspondante est remplie, 0 ou `X` sinon. On peut également coder une case non remplie par la liste de toutes les valeurs possibles en principe.
Le remplissage est récursif : on choisit une case dont le nombre de remplissages possibles est minimal, on attribue toutes les valeurs possible à cette case, ce qui constitue une grille partiellement remplie avec une case sûre de plus ; on purge la grille des déductions évidentes (on élimine la valeur choisie des cases non remplies de la même rangée ou zone, ce qui peut faire apparaître de nouvelles cases sûres, etc.). Cette étape est nécessaire car, apparemment, le nombre de niveaux de récursivité autorisé par xcas est de l'ordre de 50. Aussi, si on a une grille remplie avec une vingtaine de cases au départ, on est certain de ne pas pouvoir remplir la soixantaine de cases restantes par récursivité.

Approche par graphe
On construit un graphe dont les sommets sont les couples $(k, \ell) \in \{1, \dots, n\}^2$; il y a une arête entre deux sommets s'ils représentent des cases de la même ligne, la même colonne ou la même zone. Remplir un sudoku consiste à attribuer une « couleur » à chaque sommet de sorte que deux sommets reliés par une arête sont de couleurs différentes.

Application : fabrication de grilles
Comment exploiter ce résolveur pour fabriquer des grilles à commercialiser ?

1. Ne pensez pas que ce goût de la généralisation est gratuit : rendre le nombre de cases générique permet de tester le programme sur des grilles... plus petites pour débuguer !