TD 8 : Backtracking

Février 2013

"Essayer. Rater. Essayer encore. Rater encore. Rater mieux."
-Samuel Beckett

Le backtracking

Le backtracking consiste à trouver toutes les solutions d'un problème de satisfaction de contraintes. On fait des hypothèses au fur et à mesure, et on s'arrête dès qu'une hypothèse rend les choses contradictoires. Cette technique a l'avantage d'être beaucoup plus efficace qu'une recherche purement exhaustive.

Les fonctions de backtracking seront très très très récursives, avec le schéma suivant :

Procédure `Résolution` :

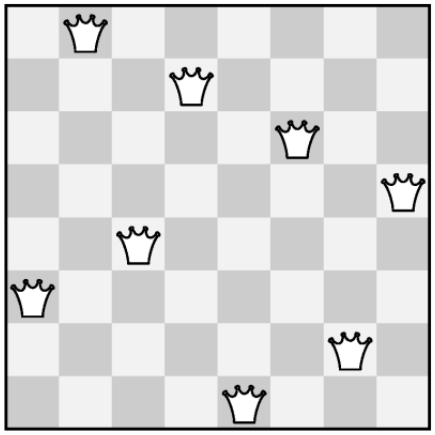
1. Si la solution partielle actuelle est complète et correcte, on l'affiche ou on l'enregistre.
2. Si la solution partielle actuelle ne satisfait pas les contraintes, on revient en arrière.
3. Sinon, pour chaque nouvelle hypothèse possible à faire :
   (a) On ajoute l'hypothèse à la solution partielle actuelle pour créer une nouvelle solution partielle plus précise.
   (b) On rappelle récursivement la procédure `Résolution` sur cette nouvelle solution.
   (c) On continue avec la première solution partielle.

Génération de permutations

On cherche à afficher, pour un n donné, toutes les permutations de $\{1, \dots, n\}$ qui ne contiennent pas de sous-suite croissante de taille 3 ou plus. On s'emploiera tout particulièrement à ne pas calculer les $n!$ différentes permutations.

Adapter le code de `permutations.mw` (sur la page web du TD) pour résoudre le problème posé par backtracking.

Le problème des 8 dames



Sur l'échiquier précédent, les 8 dames ne se menacent pas mutuellement. On cherche à savoir de combien de façons on peut réaliser cette configuration, avec du backtracking.

Question 1 Combien y-a-t'il de façons de placer 8 dames sur un échiquier sans aucune contrainte (si ce n'est de n'en mettre qu'une par case) ?

Pour modéliser l'échiquier, on utilisera une liste de 8 listes de taille 8 :

```
echiquier:=[seq([seq(0,i=1..8)],j=1..8)];
```

On passera la variable `echiquier` à la plupart de nos fonctions. On représentera par 0 les cases vides, par 1 les dames, et par -1 les cases menacées par des dames.

Question 2 Ecrire `disponibles:=proc(echiquier)` qui renvoie la liste des cases disponibles sur l'échiquier (sous forme de couple (i,j)) et `nDames:=proc(echiquier)` qui calcule le nombre de dames posées actuellement sur l'échiquier.

En déduire `mauvaise:=proc(echiquier)` qui détermine si une configuration partielle est insoluble ou non.

Question 3 Ecrire `poserDame:=proc(echiquier,i,j)`. La case (i,j) de l'échiquier doit être libre et la fonction renvoie un nouvel échiquier complèté avec une dame en (i,j) et les nouvelles cases menacées.

Question 4 Utiliser intelligemment les fonctions précédentes pour résoudre le problème des 8 dames, avec le même squelette que pour les permutations.

Sudoku

Une grille de Sudoku est une grille carrée de coté 9, où chaque case doit être remplie par un entier entre 1 et 9. Les contraintes suivantes doivent être respectées :

▷ Chaque ligne contient tous les chiffres de 1 à 9

▷ Chaque colonne contient tous les chiffres de 1 à 9

▷ Chacun des 9 petits sous-carrés 3*3 de la grille contient tous les chiffres de 1 à 9

Adapter le problème précédent *mutatis mutandis* pour écrire `resoutSudoku:=proc(sudoku)`, qui résout la grille contenue dans la variable `sudoku` ; puis résoudre (pas à la main) la grille suivante :

8			4		6			7
						4		
	1					6	5	
5		9		3		7	8	
				7				
	4	8		2		1		3
	5	2					9	
		1						
3			9		2			5