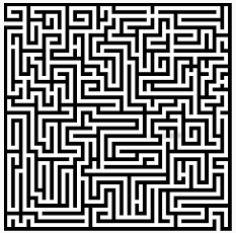
[CI2] Cours 8: Backtracking ou retour sur trace

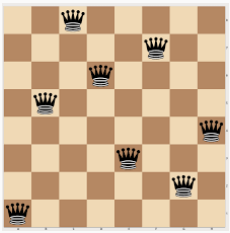
Daniela Petrişan
Université Paris Cité, IRIF

Backtracking ou retour sur trace

Le backtracking ou le retour sur trace est une technique de programmation qui permet d'optimiser la recherche exhaustive de solutions d'un problème.

Nous économisons des calculs inutiles, non pas parce qu'ils ont déjà été effectués, mais plutôt parce que nous déterminons à l'avance qu'ils sont inutiles.





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

Schéma général de backtracking

```
static void backtrack(){          // on est sur un noeud
  if(solutionComplete()){         // si ce noeud est une feuille (solution)
    nbSol++;
    System.out.println(nbSol+":"+sol);
    //  on affiche une ou toutes les solutions (au choix, selon le contexte)
  } else {                                // on est sur un noeud interne
    for(Integer i:possibles()){           // on parcourt les successeurs possibles
      sol.add(i);                         // on descend vers le noeud successeur
      backtrack();                        // on lance la récursion
      sol.remove(sol.size()-1);           // on remonte
    }
}
```

Les variables globales, leurs types, les méthodes `solutionComplete()` et `possibles()` sont à personnaliser.

Le problème des N dames

Le but du problème des N dames est de placer N dames d'un jeu d'échecs sur un échiquier de N × N cases sans que les dames ne puissent se menacer mutuellement, conformément aux règles du jeu d'échecs. Par conséquent, deux dames ne devraient jamais partager la même rangée, colonne, ou diagonale.

Comment modéliser une solution ? Quel type faut-il utiliser ?

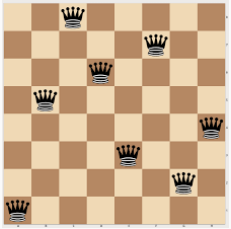
Le problème des N dames

Le but du problème des N dames est de placer N dames d'un jeu d'échecs sur un échiquier de N × N cases sans que les dames ne puissent se menacer mutuellement, conformément aux règles du jeu d'échecs. Par conséquent, deux dames ne devraient jamais partager la même rangée, colonne, ou diagonale.

Comment modéliser une solution ? Quel type faut-il utiliser ? Une matrice ?

Le problème des N dames

Le but du problème des N dames est de placer N dames d'un jeu d'échecs sur un échiquier de N × N cases sans que les dames ne puissent se menacer mutuellement, conformément aux règles du jeu d'échecs. Par conséquent, deux dames ne devraient jamais partager la même rangée, colonne, ou diagonale.

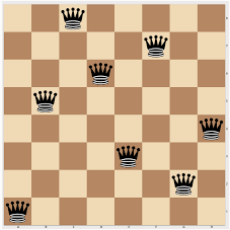


Comment modéliser une solution ? Quel type faut-il utiliser ? Une matrice ?
Ce serait un gaspillage d'espace… un tableau unidimensionnel `sol` suffit. Puisque sur chaque ligne il peut y avoir au plus une dame, nous conserverons dans `sol[i]` le numéro de la colonne sur laquelle se trouve la dame de la ligne `i`.

Le problème des N dames

Le but du problème des N dames est de placer N dames d'un jeu d'échecs sur un échiquier de N × N cases sans que les dames ne puissent se menacer mutuellement, conformément aux règles du jeu d'échecs. Par conséquent, deux dames ne devraient jamais partager la même rangée, colonne, ou diagonale.

```
static void backtrack(){
  if(solutionComplete()){
    nbSol++;
    System.out.println(nbSol+":"+sol);
  } else {
    for(Integer i:possibles()){
      sol.add(i);
      backtrack();
      sol.remove(sol.size()-1);
    }
}
```



Le problème des N dames avec backtracking

```
class NDames{
  static ArrayList<Integer> sol;  // sol est le vecteur solution
                                  //(en cours de construction)
                                  // ArrayList est redimensionnable
                                  ↪  dynamiquement.
                                  // ici: le i-ème élément est le numéro de col
                                  ↪  de la dame de la ligne i
```

.

Le problème des N dames avec backtracking

```
class NDames{
  static ArrayList<Integer> sol;  // sol est le vecteur solution
                                  //(en cours de construction)
                                  // ArrayList est redimensionnable
                                  ↪  dynamiquement.
                                  // ici: le i-ème élément est le numéro de col
                                  ↪  de la dame de la ligne i
  static int dim;                 // dim est une des dimensions du problème
                                  // ici: le nombre de lignes (= de colonnes)
                                  ↪  de l'échiquier

  static int nbSol;               // nbSol est le nombre de solution(s)
```

```
class NDames{
  static ArrayList<Integer> sol;  // vecteur solution (en cours de construction)
  static int dim;                 //  le nombre de lignes (= de colonnes) de l'échiquier
  static int nbSol;               // nbSol est le nombre de solution(s)
  static boolean solutionComplete() {
    return sol.size() == dim;
  }
  static ArrayList<Integer> possibles() {
    ArrayList<Integer> pos = new ArrayList<Integer>();
    // les dames déjà en place sont aux coordonnées:
    // (0,sol.get(0)), (1,sol.get(1)), ..., (sol.size()-1,sol.get(sol.size()-1))
    boolean menace=false;
    for(int col = 0; col < dim; col++) {    // numéros de colonne candidats
 }
```

```
class NDames{
  static ArrayList<Integer> sol;  // vecteur solution (en cours de construction)
  static int dim;                 //  le nombre de lignes (= de colonnes) de l'échiquier
  static int nbSol;               // nbSol est le nombre de solution(s)
  static boolean solutionComplete() {
    return sol.size() == dim;
  }
  static ArrayList<Integer> possibles() {
    ArrayList<Integer> pos = new ArrayList<Integer>();
    // les dames déjà en place sont aux coordonnées:
    // (0,sol.get(0)), (1,sol.get(1)), ..., (sol.size()-1,sol.get(sol.size()-1))
    boolean menace=false;
    for(int col = 0; col < dim; col++) {    // numéros de colonne candidats
      menace = false;









  }
```

```
class NDames{
  static ArrayList<Integer> sol;  // vecteur solution (en cours de construction)
  static int dim;                 //  le nombre de lignes (= de colonnes) de l'échiquier
  static int nbSol;               // nbSol est le nombre de solution(s)
  static boolean solutionComplete() {
    return sol.size() == dim;
  }
  static ArrayList<Integer> possibles() {
    ArrayList<Integer> pos = new ArrayList<Integer>();
    // les dames déjà en place sont aux coordonnées:
    // (0,sol.get(0)), (1,sol.get(1)), ..., (sol.size()-1,sol.get(sol.size()-1))
    boolean menace=false;
    for(int col = 0; col < dim; col++) {    // numéros de colonne candidats
      menace = false;
      for(int i=0; i < sol.size(); i++) {   // numéros de ligne des dames placées









  }
```

```
class NDames{
  static ArrayList<Integer> sol;  // vecteur solution (en cours de construction)
  static int dim;                 //  le nombre de lignes (= de colonnes) de l'échiquier
  static int nbSol;               // nbSol est le nombre de solution(s)
  static boolean solutionComplete() {
    return sol.size() == dim;
  }
  static ArrayList<Integer> possibles() {
    ArrayList<Integer> pos = new ArrayList<Integer>();
    // les dames déjà en place sont aux coordonnées:
    // (0,sol.get(0)), (1,sol.get(1)), ..., (sol.size()-1,sol.get(sol.size()-1))
    boolean menace=false;
    for(int col = 0; col < dim; col++) {    // numéros de colonne candidats
      menace = false;
      for(int i=0; i < sol.size(); i++) {   // numéros de ligne des dames placées
        // S'il y a déjà une dame sur la même colonne  ou même diagonale
          menace=true;
      }}
      if(!menace) pos.add(col);
    }
    return pos;
  }
```

```java
class NDames{
  static ArrayList<Integer> sol;  // vecteur solution (en cours de construction)
  static int dim;                 //  le nombre de lignes (= de colonnes) de l'échiquier
  static int nbSol;               // nbSol est le nombre de solution(s)
  static boolean solutionComplete() {
    return sol.size() == dim;
  }
  static ArrayList<Integer> possibles() {
    ArrayList<Integer> pos = new ArrayList<Integer>();
    // les dames déjà en place sont aux coordonnées:
    // (0,sol.get(0)), (1,sol.get(1)), ..., (sol.size()-1,sol.get(sol.size()-1))
    boolean menace=false;
    for(int col = 0; col < dim; col++) {    // numéros de colonne candidats
      menace = false;
      for(int i=0; i < sol.size(); i++) {   // numéros de ligne des dames placées
        if (col == sol.get(i) || Math.abs(col-sol.get(i)) == Math.abs(sol.size()-i)) {
          menace=true;       // Il y a déjà une dame sur la même colonne  ou même diagonale
      }}
      if(!menace) pos.add(col);
    }
    return pos;
  }
```