Backtracking - n queens

OPTION INFORMATIQUE - TP n° 3.1 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ Implémenter un algorithme de recherche par la force brute
- ☞ Implémenter un algorithme de recherche par retour sur trace
- ☞ garantir qu'une fonction retourne toujours le même type et éventuellement `unit ()`.
- ☞ coder une fonction sur une liste de manière récursive en utilisant le filtrage de motif et les fonctions auxiliaires
- ☞ utiliser module List pour coder des équivalents aux codes récursifs (iter, map, filter, fold)

Ce TP a pour but d'appréhender la thématique de l'exploration par la force brute puis par la technique du retour sur trace.

A Le problème des n reines

On cherche à placer sur un échiquier de $n \times n$ cases n reines sans que celles-ci s'attaquent les unes les autres. On rappelle que la reine peut attaquer une pièce sur la ligne, la colonne et les diagonales qui partent de sa position.

B Modélisation de l'échiquier pour les n reines

Il est possible de représenter un échiquier à l'aide de différentes structures de données. La première qui vient à l'esprit, certainement à cause de la visualisation de l'échiquier, est le tableau à deux dimensions. Néanmoins, lorsqu'on observe de plus près la répartition des reines pour des configurations solutions, il apparaît clairement qu'une même ligne ne peut accueillir qu'une seule reine. C'est pourquoi il est possible d'implémenter l'échiquier par une liste : l'indice de la liste représente le numéro de la ligne sur laquelle se situe la reine. Le ième élément de la liste représente la colonne sur laquelle se trouve la reine. S'il n'y a pas de reine sur la ligne i, le ième élément de la liste vaut -1.

Par exemple, l'échiquier représenté sur la figure 1 est encodé par la liste `board=[7;3;0;2;5;1;6;4]`. On note que le premier élément de `board` vaut 7, ce qui signifie qu'il y a une reine sur la première ligne et la huitième colonne. Un échiquier `[3;-1;2;-1]` est un échiquier de 4x4 avec une reine en $(0,3)$ et une autre en $2,2$.

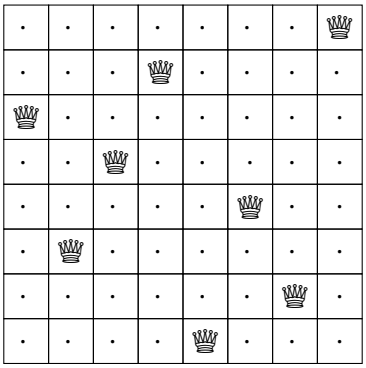
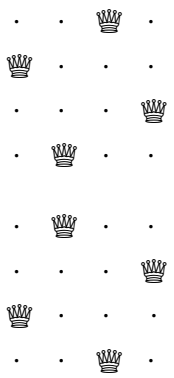


FIGURE 1 – Exemple d'échiquier 8x8 solution du problème des huit reines.

C Résolution par force brute

C1. Combien y-a-t-il de solutions au problème des quatre reines [1] ?

Solution : Il y a deux solutions :



C2. On souhaite afficher sur la console l'échiquier avec les reines comme sur la figure 2. La commande `print_string "\u{2655}"` permet d'imprimer le symbole UTF-8 de la reine d'un jeu d'échec.

(a) Écrire une fonction dont le signature est `show : int list -> unit` qui affiche sur la console l'échiquier comme sur les figures 2 et 3. **On utilisera la fonction `iter` du module List.**

(b) Écrire une fonction **récursive** dont le signature est `rec_show : int list -> unit` qui affiche sur la console l'échiquier comme sur les figures 2 et 3.

C3. On souhaite placer une reine sur la ligne r et la colonne c. D'autres reines sont déjà présentes sur l'échiquier.

(a) Écrire une fonction de signature `same_col : int -> int list -> bool` qui teste si une reine située sur la colonne c est attaquée par une autre reine présente sur la même colonne. **On écrira une version récursive et une autre version qui utilise la fonction `mem` module List.**

1. On peut vérifier le résultat en comparant avec la séquence A000170

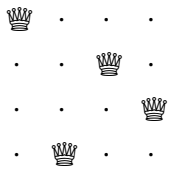


FIGURE 2 – Résultat sur la console de la fonction `show board` avec `board=[0;2;1;3]`

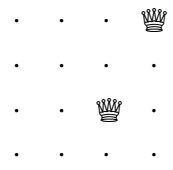


FIGURE 3 – Résultat de la fonction `show` sur la liste `[3;-1;2;-1]`

(b) Écrire une fonction de signature `same_row : int -> int list -> bool` qui teste si une reine située sur la ligne r est attaquée par une autre reine présente sur la même ligne. Si elle est attaquée, cela signifie que l'élément r de la liste est un nombre différent de -1. **On écrira une version récursive et une autre version qui utilise la fonction `nth` du module List.**

(c) Écrire une fonction de signature `down_diag : int -> int -> int list -> bool` qui teste si une reine située en (r,c) (donnés en paramètres) est attaquée par une autre reine présente sur la diagonale allant du haut vers le bas de l'échiquier. **On écrira une version récursive et une autre version qui utilise la fonction `mapi` du module List.**

(d) Écrire une fonction de signature `up_diag : int -> int -> int list -> bool` qui teste si une reine située en (r,c) (donnés en paramètres) est attaquée par une autre reine présente sur la diagonale allant du bas vers le haut. **On écrira une version récursive et une autre version qui utilise la fonction `mapi` du module List.**

C4. Écrire une fonction de signature `under_attack : int -> int -> int list -> bool` qui vérifie si la reine en (r,c) est attaquée par une autre reine présente sur l'échiquier. Il est nécessaire de masquer la présence éventuelle de cette reine avant de tester l'échiquier [2].

C5. Écrire une fonction de signature `valid_solution : int list -> bool` teste si une configuration (liste) est une configuration valide (c'est à dire aucune reine n'est attaquée).

C6. Résoudre le problème des quatre reines en écrivant un code qui trouve toutes les solutions par la force brute.

Pour améliorer la compacité du code, on peut remarquer qu'avec la représentation sous forme de liste de l'échiquier, une configuration valide est forcément une permutation de la liste `[0;1;2;3;4;5;6;7]` . On peut donc se contenter de tester toutes les permutations de cette liste. On se propose donc d'écrire un code qui génère toutes les permutation d'une liste.

C7. (a) Écrire une fonction récursive de signature `rm : 'a -> 'a list -> 'a list` qui supprime toutes les occurrences d'un élément spécifié d'une liste et renvoie la liste ainsi modifiée.

2. une sous fonction serait adaptée à la création de cette liste masquée.

(b) Écrire une fonction de signature `create_board : int -> int list` qui renvoie la liste des entiers des 0 à $n-1$.

(c) Écrire une fonction récursive de signature `permutations : 'a list -> 'a list list` qui génère la liste de toutes les permutations d'une liste. On pourra raisonner comme suit :

1. si `my_list` est vide, renvoyer une liste vide,
2. si `my_list` possède un seul élément, renvoyer la liste qui contient cet élément,
3. sinon pour chaque élément de `my_list`, créer toutes les permutations de `my_list` sans cet élément et insérer l'élément en tête de ces listes. Concaténer toutes les listes obtenues ainsi.

C8. Écrire une fonction de signature `brute_force_permutation : int -> unit` qui teste la validité sur l'échiquier de toutes les permutations d'une liste à n éléments. Cette fonction affiche les échiquiers solutions et le nombre de solutions sur la console.

C9. Cette fonction est-elle efficace pour huit reines? Peut-on résoudre le problème pour des n plus grands que 8?

Solution : Pour $n = 9$, on dépasse déjà la capacité de la pile d'exécution. (Stack overflow)

Solution :

Code 1 – Résolution par la force brute du problème des n reines

```
let n = 4;;

let test_board = [2;0;3;2];;

let show board =
  let print_row v =
    for i = 0 to (List.length board) - 1 do
      print_string (if i=v then "\u{2655} " else ". ");
    done;
    print_newline()
  in List.iter print_row board; print_newline();;
show test_board;;

let rec_show board =
  let print_row col =
        for c = 0 to (List.length board) - 1 do
            if col = c then print_string "\u{2655} " else print_string ". " done
                ; print_newline()
  in let rec aux b = match b with
               | [] -> print_newline()
               | col::t -> print_row col; aux t;
  in aux board;;
rec_show test_board;;

show [3;-1;2;-1];;

let same_col c board = List.mem c board;;
same_col 1 test_board;;
```

```
same_col 3 test_board;;

let rec rec_same_col c board = match board with
    | [] -> false
    | head::tail -> if c = head then true else rec_same_col c tail;;
rec_same_col 1 test_board;;
rec_same_col 3 test_board;;


let same_row r board = (List.nth board r) != -1;;
same_row 1 test_board;;
same_row 2 test_board;;
same_row 3 test_board;;
same_row 0 test_board;;
same_row 1 [3;-1;2;-1];;
same_row 2 [3;-1;2;-1];;
same_row 3 [3;-1;2;-1];;

let rec_same_row r board =
    let rec aux b i = match b with
        | [] -> false
        | head::tail -> if i = r && head != -1 then true else aux tail (i + 1)
        in aux board 0;;
show test_board;;
rec_same_row 1 test_board;;
rec_same_row 2 test_board;;
rec_same_row 3 test_board;;
rec_same_row 0 test_board;;
rec_same_row 1 [3;-1;2;-1];;
rec_same_row 2 [3;-1;2;-1];;
rec_same_row 3 [3;-1;2;-1];;


let up_diag r c board =
  let rc = r + c
    in let diag board = List.mapi (fun index elem -> if elem != -1 then Some (
        index + elem) else None) board
      in List.mem (Some rc) (diag board);;
up_diag 0 0 test_board;;
up_diag 3 0 test_board;;
up_diag 0 3 test_board;;
up_diag 3 3 test_board;;
up_diag 2 2 test_board;;
up_diag 0 1 test_board;;
up_diag 1 1 test_board;;

let rec_up_diag r c board =
  let rc = r + c
    in let rec diag b i = match b with
        | [] ->  []
        | head::tail -> if head != -1 then (i + head)::(diag tail (i + 1)) else
            diag tail (i + 1)
    in List.mem rc (diag board 0);;
rec_up_diag 0 0 test_board;;
rec_up_diag 3 0 test_board;;
```

```
rec_up_diag 0 3 test_board;;
rec_up_diag 3 3 test_board;;
rec_up_diag 2 2 test_board;;
rec_up_diag 0 1 test_board;;
rec_up_diag 1 1 test_board;;


let down_diag r c board =
  let rc = r - c
    in let diag board = List.mapi (fun index elem -> if elem != -1 then Some (
        index - elem) else None) board
      in List.mem (Some rc) (diag board);;
down_diag 0 0 test_board;;
down_diag 3 0 test_board;;
down_diag 0 3 test_board;;
down_diag 3 3 test_board;;
down_diag 2 2 test_board;;
down_diag 0 1 test_board;;
down_diag 1 1 test_board;;
down_diag 2 1 test_board;;

let rec_down_diag r c board =
  let rc = r - c
    in let rec diag b row = match b with
        | [] ->  []
        | col::tail -> if col != -1 then (row - col)::(diag tail  (row + 1))
            else diag tail (row + 1)
    in List.mem rc (diag board 0);;
rec_down_diag 0 0 test_board;;
rec_down_diag 3 0 test_board;;
rec_down_diag 0 3 test_board;;
rec_down_diag 3 3 test_board;;
rec_down_diag 2 2 test_board;;
rec_down_diag 0 1 test_board;;
rec_down_diag 1 1 test_board;;
rec_down_diag 2 1 test_board;;


let attack_test_board = [2;-1;3;-1];;
rec_show attack_test_board;;
show attack_test_board;;
let under_attack r c board =
  let masked = List.mapi (fun index elem -> if r = index && c = elem then -1
      else elem) board
      in same_row r masked || same_col c masked || up_diag r c masked  ||
          down_diag r c  masked;;
under_attack 0 0 attack_test_board;;
under_attack 1 0 attack_test_board;;
under_attack 1 1 attack_test_board;;
under_attack 1 2 attack_test_board;;
under_attack 1 3 attack_test_board;;
under_attack 0 2 attack_test_board;;
under_attack 3 1 attack_test_board;;
same_row 0 attack_test_board;;
```

```
same_row 1 attack_test_board;;
same_row 2 attack_test_board;;
same_row 3 attack_test_board;;
same_col 0 attack_test_board;;
same_col 1 attack_test_board;;
same_col 2 attack_test_board;;
same_col 3 attack_test_board;;
down_diag 3 0 attack_test_board;;
up_diag 3 0 attack_test_board;;
down_diag 3 1 attack_test_board;;
up_diag 3 1 attack_test_board;;


let rec_under_attack r c board =
  let masked =
   let rec aux b row = match b with
        | [] -> []
        | col::tail -> if (row,col) = (r,c) then -1::(aux tail (row + 1)) else
            col::(aux tail (row + 1))
        in aux board 0
  in same_row r masked || same_col c masked || up_diag r c masked  || down_diag
      r c  masked;;
rec_under_attack 0 0 attack_test_board;;
rec_under_attack 1 0 attack_test_board;;
rec_under_attack 1 1 attack_test_board;;
rec_under_attack 1 2 attack_test_board;;
rec_under_attack 1 3 attack_test_board;;
rec_under_attack 0 2 attack_test_board;;
rec_under_attack 3 1 attack_test_board;;


let valid_board = [2;0;3;1];;

let valid_solution board =
  let res = List.mapi (fun r c -> under_attack r c board) board
    in not (List.mem true res);;
valid_solution attack_test_board;;
valid_solution test_board;;
valid_solution valid_board;;

let rec_valid_solution board =
    let rec check row b = match b with
        | [] -> true
        | col::tail -> if under_attack row col board then false else check (row
            + 1) tail
    in  check 0 board;;
rec_valid_solution attack_test_board;;
rec_valid_solution test_board;;
rec_valid_solution valid_board;;


let raw_force_4_queens () =
  let board = []
  in for i=0 to n - 1 do
        let bi = i::board in
```

```
184         for j=0 to n - 1 do
185           let bij = j::bi in
186           for k=0 to n - 1 do
187             let bijk = k::bij in
188             for l=0 to n - 1 do
189               let bijkl = l::bijk in
190                   if valid_solution bijkl then (Printf.printf "[%i,%i,%i,%i]\n"
                          i j k l; rec_show bijkl;)
191             done;
192           done;
193         done;
194       done;;
195 raw_force_4_queens();;
196
197
198
199 let rec rm x my_list = match my_list with
200   | [] -> []
201   | h::t -> if h=x then rm x t else  h::rm x t;;
202
203 let rm x my_list = List.filter ((<>) x) my_list;;
204
205 let create_board s =
206     let rec aux l i = if s=i then l else i::(aux l (i + 1))
207     in aux [] 0;;
208 create_board 5;;
209
210 let create_board s = List.init s (fun x -> x);;
211 create_board 5;;
212
213 (* Fixed-head solution*)
214 let rec permutations my_list =  match my_list with
215   | [] -> []
216   | x::[] -> [[x]]
217   | l -> List.fold_left (fun acc x -> acc @ List.map (fun p -> x::p) (
        permutations (rm x l))) [] l;;
218
219 permutations (create_board 4);;
220 permutations (create_board 5);;
221 permutations (create_board 6);;
222
223 (* Fixed-head solution*)
224 let rec permutations my_list =  match my_list with
225   | [] -> []
226   | x::[] -> [[x]]
227   | l -> List.fold_left (fun acc x -> acc @ List.map (fun p -> x::p) (
        permutations (rm x l))) [] l;;
228
229 permutations (create_board 4);;
230 permutations (create_board 5);;
231 permutations (create_board 6);;
232
233
234
235 let brute_force_permutation s =
```

```
236   let sol_nb = ref 0
237     in let print_if_valid b = if valid_solution b then (incr sol_nb;
          print_string "Solution #"; print_int !sol_nb; print_newline(); rec_show
          b)
238         in let perms = permutations (create_board s)
239           in List.iter print_if_valid perms;;
240
241 brute_force_permutation 4;;
242 brute_force_permutation 5;;
243 brute_force_permutation 6;;
244 brute_force_permutation 7;;
245 brute_force_permutation 8;;
246 brute_force_permutation 9;;
247 brute_force_permutation 12;;
```

D Résolution par retour sur trace

L'algorithme de retour sur trace 1 construit au fur et à mesure les solutions partielles du problème et les rejette dès qu'il découvre une impossibilité.

Algorithme 1 Algorithme de retour sur trace

```
1 : Fonction RETOUR_SUR_TRACE(v)                    ▷ v est un nœud de l'arbre de recherche
2 :     si v est une feuille alors
3 :         renvoyer Vrai
4 :     sinon
5 :         pour chaque fils u de v répéter
6 :             si u peut compléter une solution partielle au problème 𝒫 alors
7 :                 RETOUR_SUR_TRACE(u)
8 :     renvoyer Faux
```

On utilise la modélisation de l'échiquier précédente, c'est à dire qu'on construit la liste des colonnes occupées. On procède par ligne en positionnant d'abord une reine puis une autre sur la deuxième ligne... Ainsi de suite, la liste augmente de taille jusqu'à atteindre la taille n.

On n'a plus besoin de traiter le cas où la colonne est vide[3] car à chaque fois qu'on envisage une solution partielle (une liste partielle), les lignes qui précèdent possèdent nécessairement une reine sur une des colonnes de l'échiquier.

D1. Comment coder «v est une feuille» en OCaml?

> **Solution :** Lorsque la taille de la liste qui représente l'échiquier dépasse n, alors on a rempli l'échiquier. Si l'indice de la ligne considéré est n alors on est arrivé sur une feuille.

D2. Comment coder «u peut compléter une solution partielle au problème $\mathcal{P}$»?

3. que l'on avait dû traiter dans le cas de la force brute avec -1

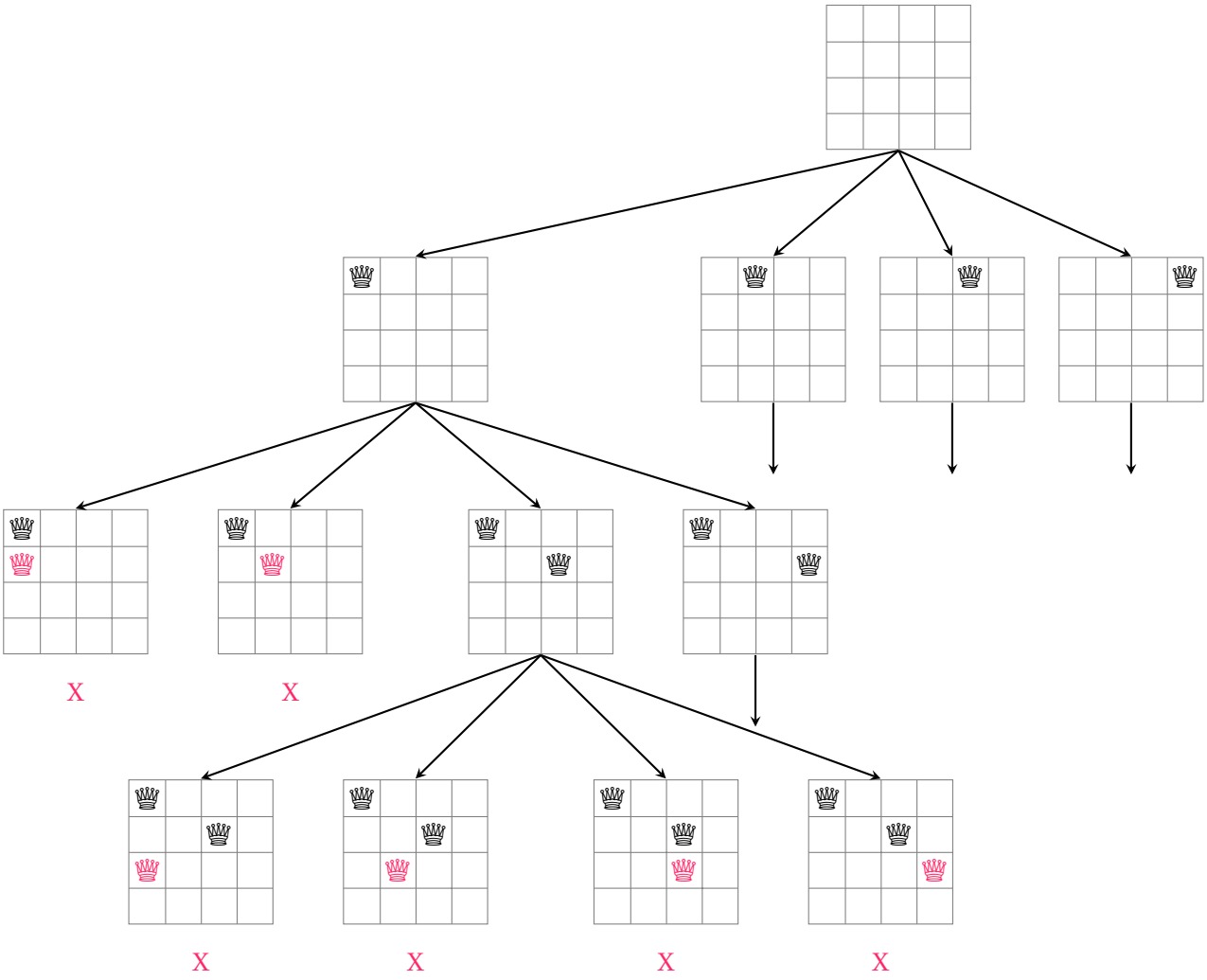


FIGURE 4 – Exemple d'arbre de recherche structurant l'algorithme de retour sur trace. Application au problème de quatre reines.

Solution : On considère la nouvelle position u `(row,col)`. Alors, il faut considérer la solution si cette reine n'est pas attaquée.

```
1             if not (under_attack row col board) then
```

D3. Implémenter un algorithme de retour sur trace pour le problème des quatre reines.

D4. Tester ce programme sur le problème des n reines et comparer les résultats avec l'algorithme de force brute. Vérifier que vous retrouvez les mêmes résultats.

D5. Compiler le programme. Quelle valeur de n engendre un temps d'exécution supérieur à la minute ?

D6. Implémenter cet algorithme en Python et comparer les performances.

Solution : Sur ma machine, trois fois plus vite en compilé. OCaml plus rapide que Python non compilé.

Code 2 – N queens backtracking

```
1
2  let show_sol board nb =
3      let print_row col = for c = 0 to (List.length board) - 1 do if col = c then
           print_string "\u{2655} " else print_string ". " done; print_newline() in
4          let rec aux b = match b with
5              | [] -> print_newline()
6              | col::[] -> print_row col; print_newline()
7              | col::t -> print_row col; aux t;
8          in print_string "Solution #"; print_int nb; print_newline(); aux board
9    ;;
10
11
12   (* rec version *)
13 let under_attack row col board =
14   let up_diag =
15     let rec aux i b = match b with
16     | [] -> []
17     | j::t -> (i + j)::(aux (i + 1) t)
18     in aux 0 board
19   in let down_diag =
20       let rec aux i b = match b with
21         | [] -> []
22         | j::t -> (i - j)::(aux (i + 1) t)
23     in aux 0 board
24   in List.mem col board || List.mem (row + col) up_diag  || List.mem (row - col)
         down_diag
25 ;;
26
27
28 (* List version *)
29 let under_attack row col board =
30   let up_diag = List.mapi (fun index elem ->  index + elem) board
31   in let down_diag = List.mapi (fun index elem ->  index - elem) board
32   in List.mem col board || List.mem (row + col) up_diag  || List.mem (row - col)
         down_diag
33 ;;
34
35
36 let n_queens n =
37   let sol_nb = ref 0
38   in let rec build_solutions row board =
39       if row = n  then
40         (incr sol_nb; show_sol board !sol_nb)
41       else
42         for c = 0 to n - 1  do
43             if not (under_attack row c board) then
44               build_solutions (row+1) (board@[c])
45         done
46       in build_solutions 0 [];;
47
48 n_queens 4;;
```

```
49
50 n_queens 5;;
51
52 n_queens 6;;
53
54 n_queens 7;;
55
56 n_queens 8;;
57
58 (*n_queens 12;;
59 n_queens 15;;*)
```

Code 3 – N queens backtracking

```
1  # coding: utf8
2  # nb solution --> https://oeis.org/A000170/list
3  n = 12
4  on_board = [None for _ in range(n)]
5  lower_diag = []
6  upper_diag = []
7  solutions_number = 0
8
9  def print_board():
10     for row in on_board:
11         for i in range(len(on_board)):
12             if i == row:
13                 print('\u2655', end=" ")  # Queen symbol utf-8
14             else:
15                 print('.', end=" ")
16         print()
17     print("________")
18
19
20 def under_attack(r, c):
21     return c in on_board or r + c in lower_diag or r - c in upper_diag
22
23 def bt_n_queens(r=0):
24     global solutions_number   # be explicit since is modified
25     if r == n:
26         return True
27     else:
28         for c in range(n):
29             #print(r,c)
30             if not under_attack(r, c):
31                 on_board[r] = c           # Set Queen
32                 lower_diag.append(r + c)  # danger aware !
33                 upper_diag.append(r - c)  # danger aware !
34                 if bt_n_queens(r + 1):
35                     solutions_number += 1
36                     print("New solution !", solutions_number)
37                     print_board()
38                 on_board[r] = None     # Unset Queen
39                 lower_diag.pop()       # remove danger
40                 upper_diag.pop()       # remove danger
41     return False
```

```
42
43
44 if __name__ == "__main__":
45     print_board()
46     bt_n_queens()
```