Backtracking et piles

Algorithmique et structures de données, 2022-2023

P. Albuquerque (B410), P. Künzli et O. Malaspinas (A401), ISC, HEPIA

2022-11-23

En partie inspirés des supports de cours de P. Albuquerque

Tri à bulle (1/4)

Algorithme

- Parcours du tableau et comparaison des éléments consécutifs:
  - Si deux éléments consécutifs ne sont pas dans l'ordre, ils sont échangés.
- On recommence depuis le début du tableau jusqu'à avoir plus d'échanges à faire.

Que peut-on dire sur le dernier élément du tableau après un parcours?

Tri à bulle (1/4)

Algorithme

- Parcours du tableau et comparaison des éléments consécutifs:
  - Si deux éléments consécutifs ne sont pas dans l'ordre, ils sont échangés.
- On recommence depuis le début du tableau jusqu'à avoir plus d'échanges à faire.

Que peut-on dire sur le dernier élément du tableau après un parcours?

- Le plus grand élément est **à la fin** du tableau.
  - Plus besoin de le traiter.
- A chaque parcours on s'arrête un élément plus tôt.

Tri à bulle (2/4)

Exemple

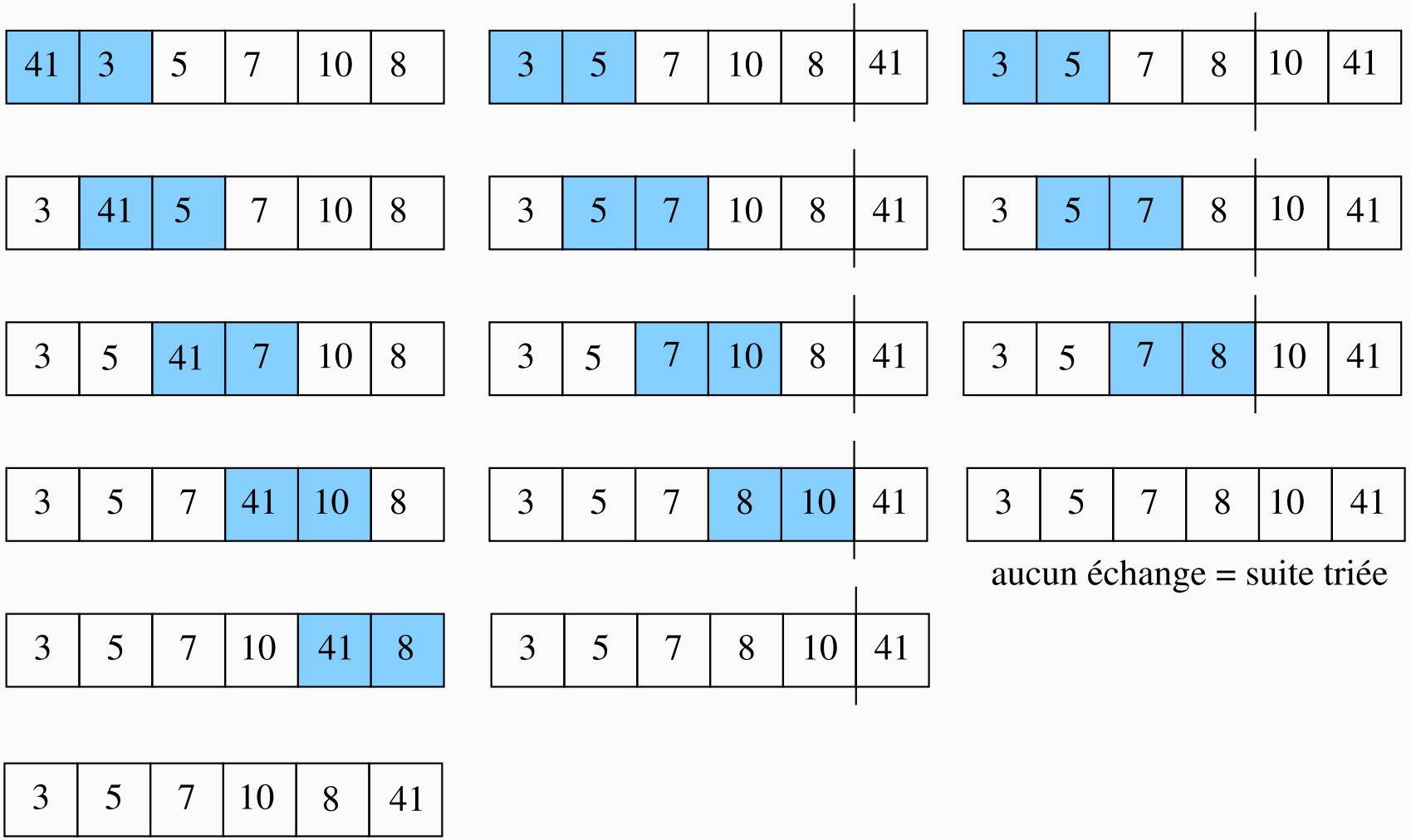


Figure 1: Tri à bulles d'un tableau d'entiers

Tri à bulle (3/4)

Exercice: écrire l'algorithme (poster le résultat sur matrix)

Tri à bulle (3/4)

Exercice: écrire l'algorithme (poster le résultat sur matrix)

```
rien tri_a_bulles(entier tableau[])
    pour i de longueur(tableau)-1 à 1:
        trié = vrai
        pour j de 0 à i-1:
            si (tableau[j] > tableau[j+1])
                échanger(array[j], array[j+1])
                trié = faux

        si trié
            retourner
```

Tri à bulle (4/4)

Quelle est la complexité du tri à bulles?

Tri à bulle (4/4)

Quelle est la complexité du tri à bulles?

- Dans le meilleurs des cas:
  - Le tableau est déjà trié: $\mathcal{O}(N)$ comparaisons.
- Dans le pire des cas, $N \cdot (N-1)/2 \sim \mathcal{O}(N^2)$:

$$\sum_{i=1}^{N-1} i \text{ comparaison et } 3 \sum_{i=1}^{N-1} i \text{ affectations (swap)} \Rightarrow \mathcal{O}(N^2).$$

- En moyenne, $\mathcal{O}(N^2)$ ($N^2/2$ comparaisons).

L'algorithme à la main

Exercice *sur papier*

- Trier par tri à bulles le tableau `[5, -2, 1, 3, 10, 15, 7, 4]`

Problème des 8-reines

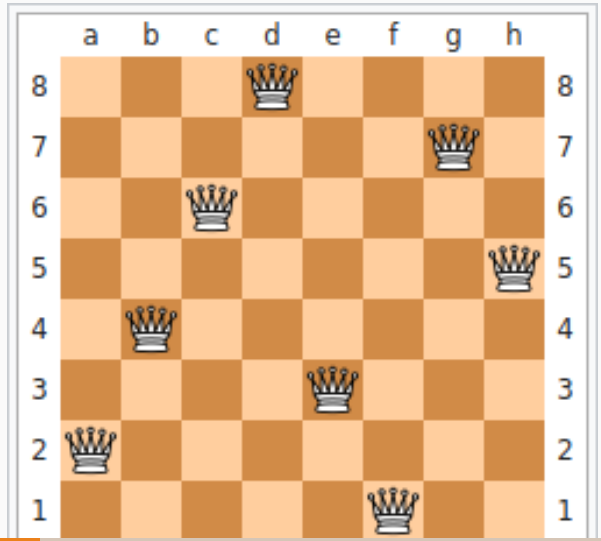
- Placer 8 reines sur un échiquier de 8×8.
- Sans que les reines ne puissent se menacer mutuellement (92 solutions).

Conséquence

- Deux reines ne partagent pas la même rangée, colonne, ou diagonale.
- Donc chaque solution a **une** reine **par colonne** ou **ligne**.

Généralisation

- Placer N reines sur un échiquier de $N \times N$.
- Exemple de **backtracking** (retour en arrière) $\Rightarrow$ récursivité.



Problème des 2-reines

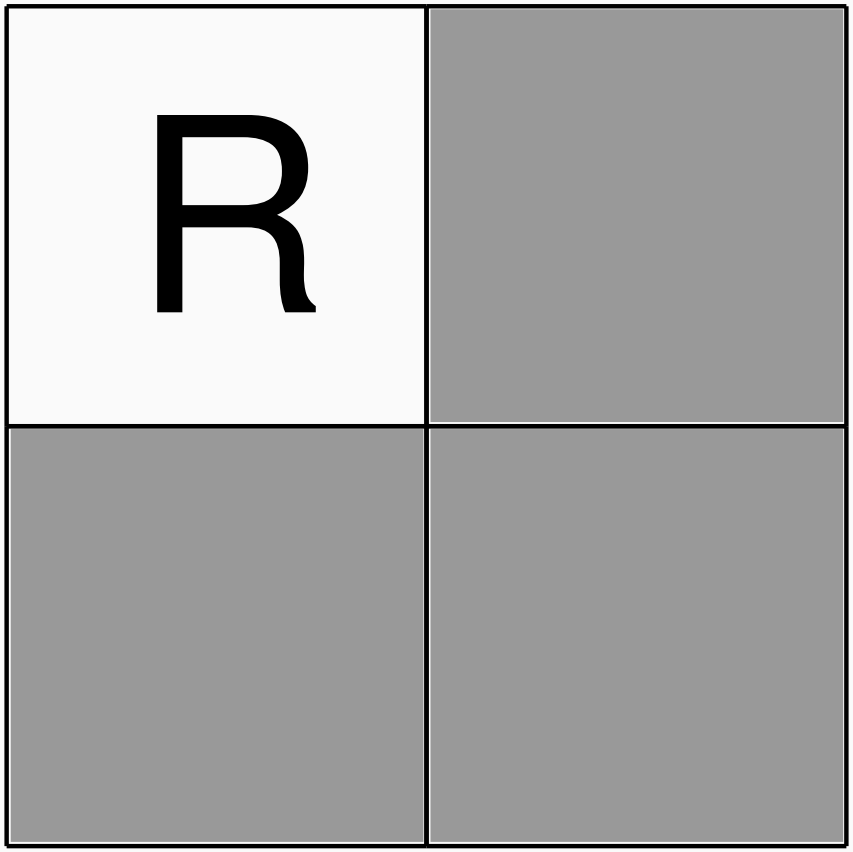


Figure 3: Le problème des 2 reines n'a pas de solution.

Comment trouver les solutions?

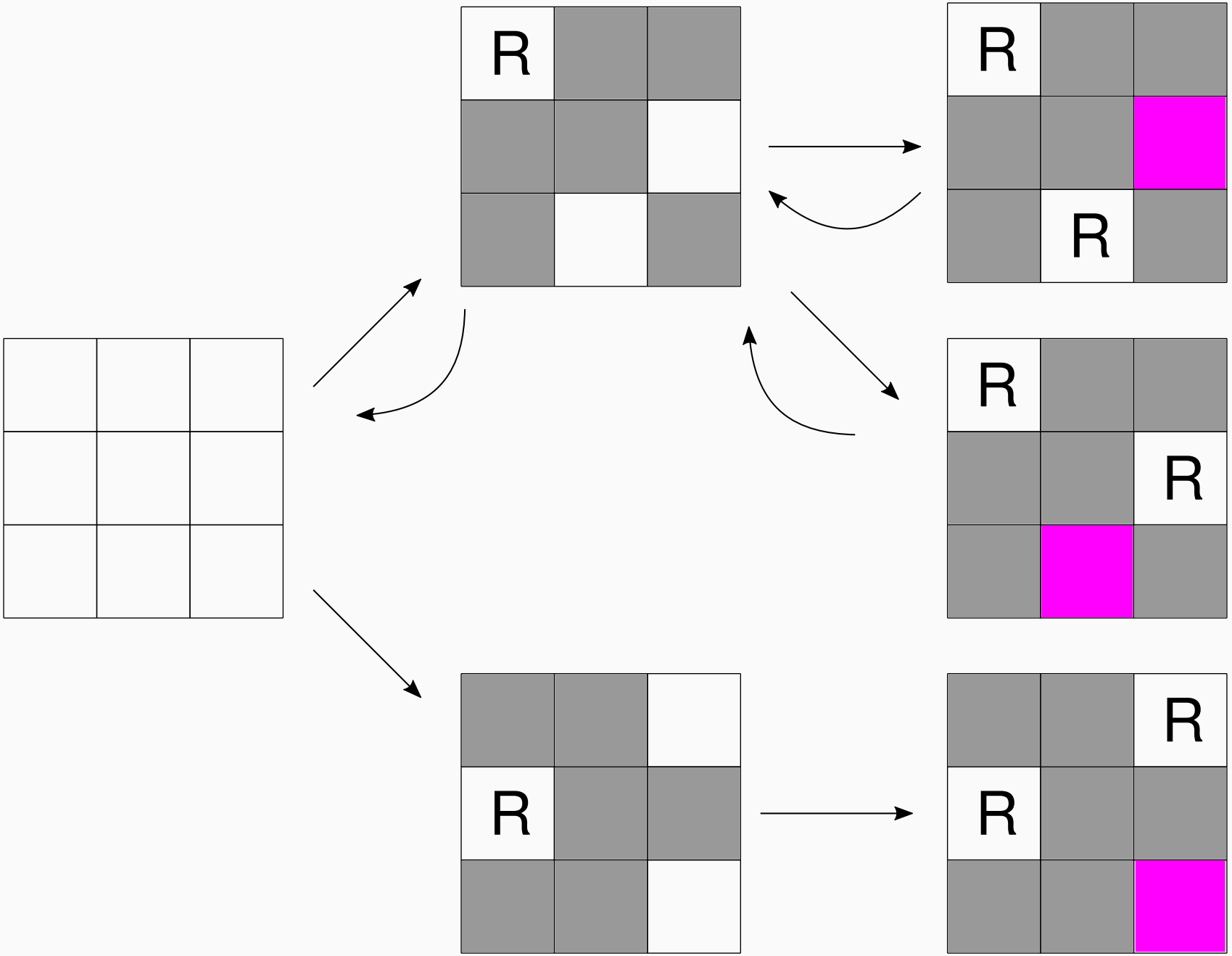
- On pose la première reine sur la première case disponible.
- On rend inaccessibles toutes les cases menacées.
- On pose la reine suivante sur la prochaine case non-menacée.
- Jusqu'à ce qu'on puisse plus poser de reine.
- On revient alors en arrière jusqu'au dernier coup où il y avait plus qu'une possibilité de poser une reine.
- On recommence depuis là.

Comment trouver les solutions?

- On pose la première reine sur la première case disponible.
- On rend inaccessibles toutes les cases menacées.
- On pose la reine suivante sur la prochaine case non-menacée.
- Jusqu'à ce qu'on puisse plus poser de reine.
- On revient alors en arrière jusqu'au dernier coup où il y avait plus qu'une possibilité de poser une reine.
- On recommence depuis là.

- Le jeu prend fin quand on a énuméré *toutes* les possibilités de poser les reines.

Problème des 3-reines



Problème des 4-reines

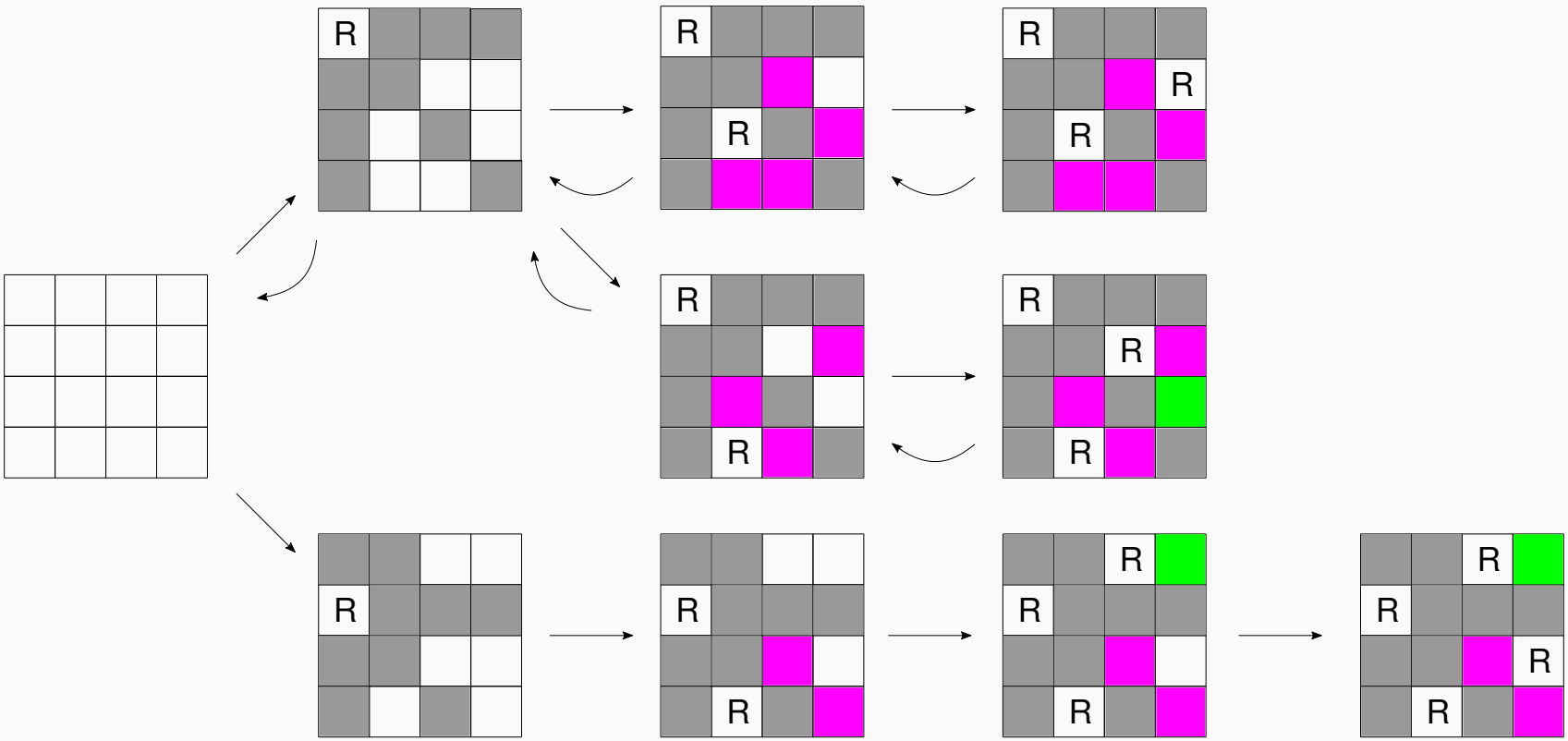


Figure 5: Le problème des 4 reines a une solution.

Problème des 4-reines, symétrie

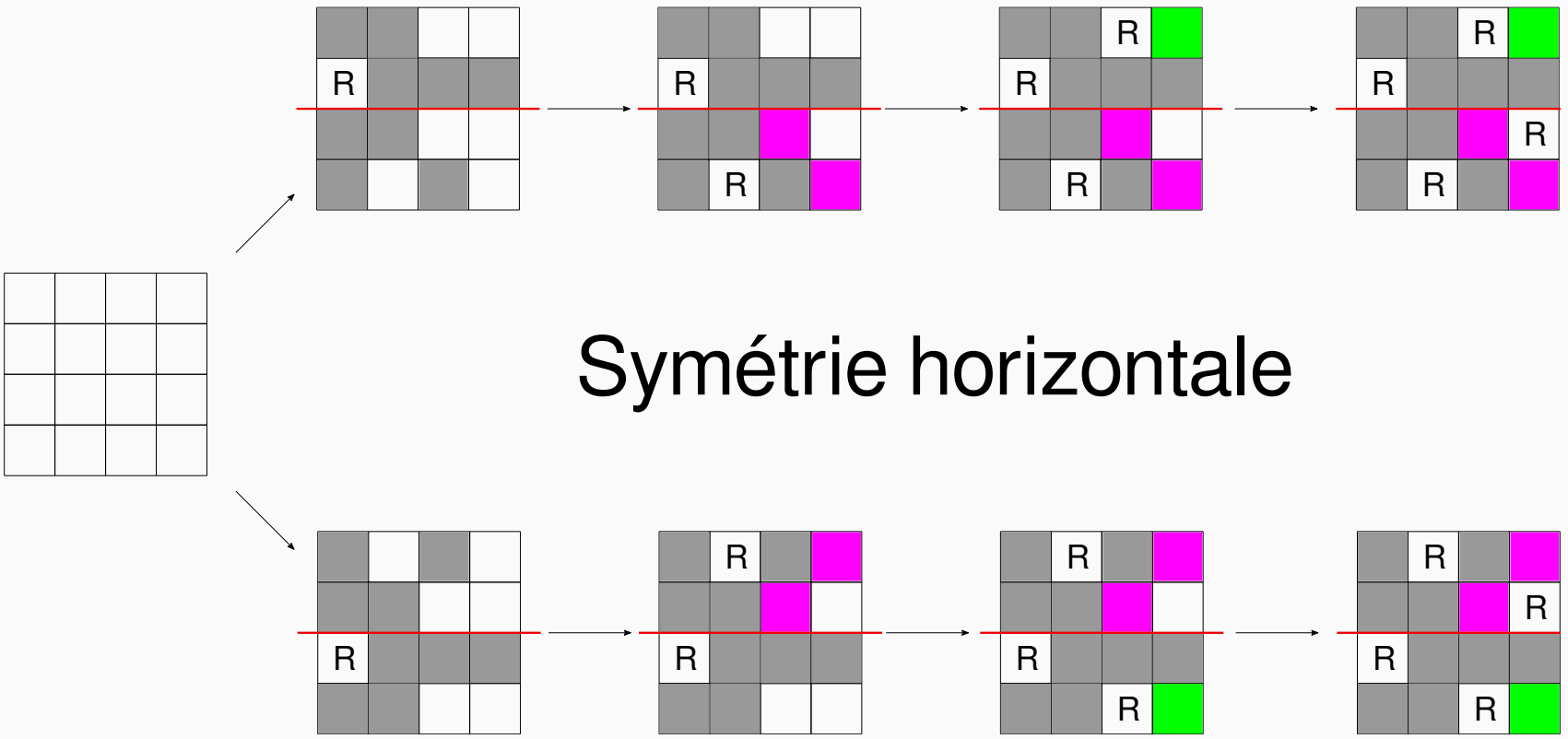


Figure 6: Le problème des 4 reines a une autre solution (symétrie horizontale).

Problème des 5 reines

Exercice: Trouver une solution au problème des 5 reines

- Faire une capture d'écran / une photo de votre solution et la poster sur matrix.

Quelques observations sur le problème

- Une reine par colonne au plus.
- On place les reines sur des colonnes successives.
- On a pas besoin de “regarder en arrière” (on place “devant” uniquement).
- Trois étapes:
  - On place une reine dans une case libre.
  - On met à jour le tableau.
  - Quand on a plus de cases libres on “revient dans le temps” ou c’est qu’on a réussi.

Le code du problème des 8 reines (1/N)

Quelle structure de données?

Le code du problème des 8 reines (1/N)

Quelle structure de données?

Une matrice de booléens fera l'affaire:

```
bool board[n][n];
```

Quelles fonctionnalités?

Le code du problème des 8 reines (1/N)

Quelle structure de données?

Une matrice de booléens fera l'affaire:

```
bool board[n][n];
```

Quelles fonctionnalités?

```
// Pour chaque ligne placer la reine sur toutes les colonnes
//    et compter les solutions
void nbr_solutions(board, column, counter);
// Copier un tableau dans un autre
void copy(board_in, board_out);
// Placer la reine à li, co et rendre inaccessible devant
void placer_devant(board, li, co);
```

Le code du problème des 8 reines (2/N)

Le calcul du nombre de solutions

```
// Calcule le nombre de solutions au problème des <n> reines
nbr_solutions(board, column, count)
   // pour chaque ligne
       // si la case libre
          // si column < n - 1
              // copier board dans un "new" board,
              //   y poser une reine
              //   et mettre à jour ce "new" board
              // nbr_solutions(new_board, column+1, count)
          // sinon
              // on a posé la n-ème et on a gagné
              // count += 1
```

Le code du problème des 8 reines (3/N)

Le calcul du nombre de solutions

```
// Placer une reine et mettre à jour
placer_devant(board, ligne, colonne)
    // board est occupé à ligne/colonne
        // toutes les cases des colonnes
        //    suivantes sont mises à jour
```

Le code du problème des 8 reines (4/N)

Compris? Alors écrivez le code et postez le!

Le code du problème des 8 reines (4/N)

Compris? Alors écrivez le code et postez le!

Le nombre de solutions

```
// Calcule le nombre de solutions au problème des <n> reines
void nb_sol(int n, bool board[n][n], int co, int *ptr_cpt) {
    for (int li = 0; li < n; li++) {
        if (board[li][co]) {
            if (co < n-1) {
                bool new_board[n][n]; // alloué à chaque nouvelle tentative
                copy(n, board, new_board);
                prises_devant(n, new_board, li, co);
                nb_sol(n, new_board, co+1, ptr_cpt);
            } else {
                *ptr_cpt = (*ptr_cpt)+1;
            }
        }
    }
}
```

Le code du problème des 8 reines (5/N)

Placer devant

```
// Retourne une copie du tableau <board> complété avec les positions
// prises sur la droite droite par une reine placée en <board(li,co)>
void prises_devant(int n, bool board[n][n], int li, int co) {
    board[li][co] = false; // position de la reine
    for (int j = 1; j < n-co; j++) {
        // horizontale et diagonales à droite de la reine
        if (j <= li) {
            board[li-j][co+j] = false;
        }
        board[li][co+j] = false;
        if (li+j < n) {
            board[li+j][co+j] = false;
        }
    }
}
```