REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Larbi Ben M'Hidi Oum El Bouaghi

Cours Méthodes algorithmiques
1ere année master AD

Chapitre 4
Backtracking

DR.BELHOUCHETTE

Introduction au **backtracking**

- Le **backtracking** est une méthode algorithmique essentielle utilisée pour résoudre des problèmes combinatoires et de recherche dans des espaces de solutions complexes.

- Son principe repose sur l'exploration systématique d'un **arbre de décision**, où chaque nœud représente un état partiel de la solution.

Définition et concept du Backtracking

À chaque étape de la construction de l'algorithme backtracking :

- Il prend une **décision** qui étend la solution partielle actuelle.
- Il **vérifie** si cette solution respecte les contraintes du problème.
- Si une contrainte est violée ou si la solution devient non prometteuse, l'algorithme **revient en arrière** (backtrack), annule la dernière décision, et essaie une autre possibilité.

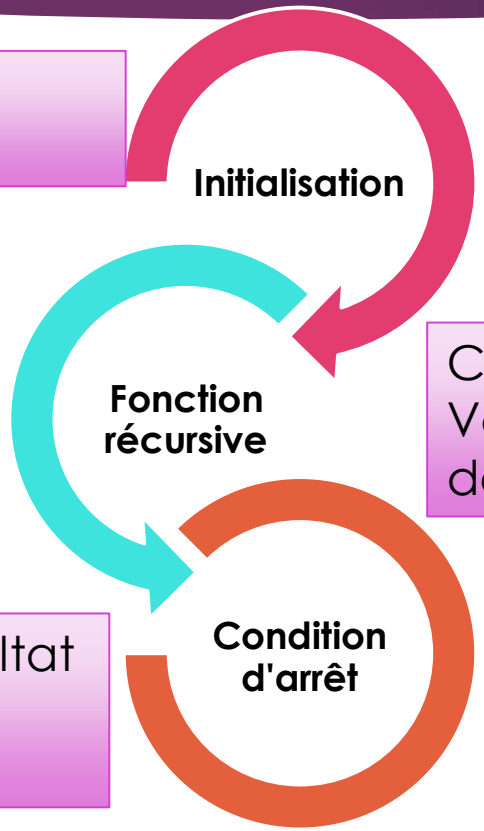
Principe de Fonctionnement du Backtracking

1. **Exploration:** Construire une solution partielle en prenant une décision à chaque étape.
2. **Vérification des contraintes:** Vérifier si la solution partielle respecte les contraintes du problème.
3. **Retour en arrière:** Si une contrainte est violée, revenir en arrière pour essayer une autre option.
4. **Solution complète:** Si une solution respecte toutes les contraintes et est complète, elle est validée.

Étapes Principales du Backtracking

Structure Générale d'un Algorithme de Backtracking

Définir les données du problème et les contraintes.



Construire la solution étape par étape.
Vérifier les contraintes après chaque décision.

Si la solution est complète, retourner le résultat
Sinon, continuer l'exploration ou revenir en arrière.

Schéma Algorithmique de Backtracking

- **Entrée :** Les données et contraintes du problème.
- **Sortie :** Une solution valide ou un message indiquant qu'aucune solution n'existe.

- **Processus :**

1. Si la solution est complète, retourner le résultat.
2. Sinon, pour chaque choix possible :
   - Vérifier si le choix respecte les contraintes.
   - Si oui, appeler récursivement la fonction pour explorer davantage.
   - Si non, passer au choix suivant.
3. Revenir en arrière si aucun choix ne mène à une solution.

Backtracking VS la Programmation Dynamique

Caractéristiques	Backtracking	la Programmation Dynamique
Complexité et Efficacité	Complexité souvent **exponentielle**, car il explore toutes les solutions possibles (dans le pire des cas).Peut être optimisé avec des techniques d'élagage (pruning), mais reste inefficace pour les grands espaces de solutions.	Complexité généralement **polynomiale**, car elle évite les calculs redondants.Elle est particulièrement efficace lorsque le problème présente une **structure de recouvrement des sous-problèmes** et une **optimalité des sous-structures**.

Backtracking VS la Programmation Dynamique

Caractéristiques	Backtracking	la Programmation Dynamique
Utilisation de la Mémoire	▪ Utilise généralement une pile d'appels récursive pour stocker les états partiels de la solution. ▪ Faible consommation mémoire dans sa version de base.	▪ Nécessite une table de mémorisation (tableau ou dictionnaire) pour stocker les résultats intermédiaires. ▪ Consommation mémoire plus élevée, mais avec une optimisation possible (ex. réduction en espace pour les algorithmes linéaires).

Backtracking VS la Programmation Dynamique

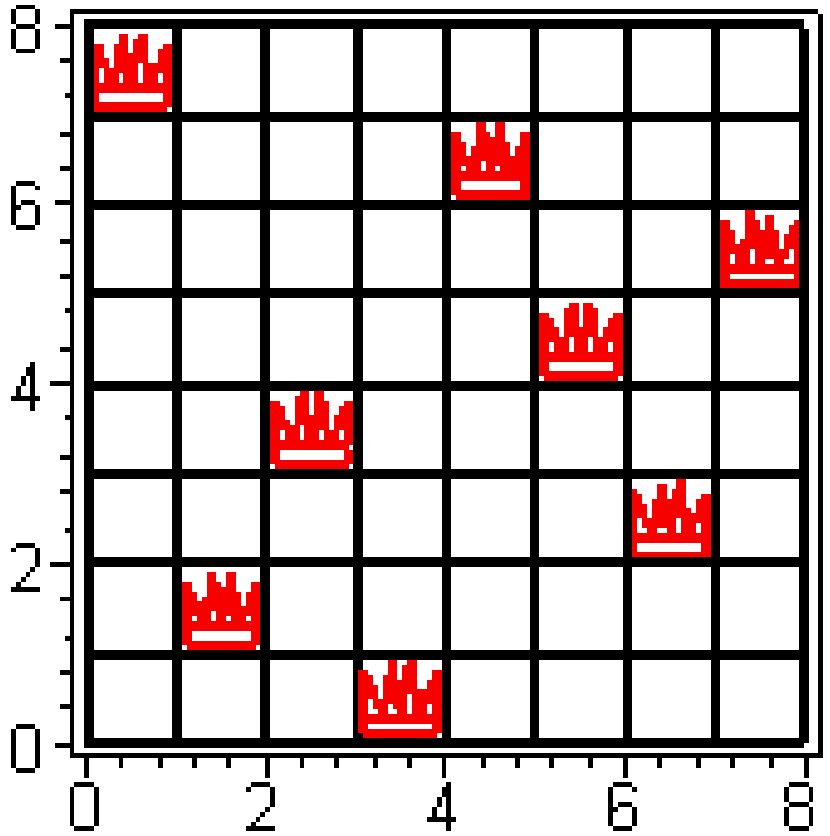
Caractéristiques	Backtracking	la Programmation Dynamique
Applications Typiques	▪ Résolution de puzzles (ex. Sudoku, problème des N-reines). ▪ Problèmes combinatoires (ex. génération de permutations, arbre de décision). ▪ Recherche exhaustive avec des contraintes complexes.	▪ Problèmes d'optimisation (ex. problème du sac à dos, plus longue sous-séquence commune). ▪ Problèmes sur les graphes (ex. plus court chemin avec Bellman-Ford). ▪ Problèmes récurrents (ex. découpage de chaînes ou matrices, alignement de séquences).

Backtracking VS la Programmation Dynamique

Caractéristiques	Backtracking	la Programmation Dynamique
Avantage principal	Simple à implémenter pour explorer toutes les solutions.	Très efficace pour les problèmes d'optimisation.
Inconvénient majeur	Très lent pour les grands problèmes (complexité exponentielle).	Nécessite de reconnaître les sous-problèmes pour l'implémentation.
Adaptabilité	Convient mieux aux problèmes avec peu de contraintes.	Plus adapté aux problèmes ayant une structure répétitive.

Problèmes Classiques Résolus par Backtracking

1. Le Problème des N-Reines



Problèmes Classiques Résolus par Backtracking

1. Le Problème des N-Reines

Qu'est-ce que le Problème des N-Reines ?

Objectif : Placer **N reines** sur un échiquier de taille N×N de manière à ce qu'aucune reine ne puisse attaquer une autre.

Contexte :

- Chaque reine peut attaquer en ligne, en colonne et en diagonale.
- Il s'agit d'un problème combinatoire classique souvent utilisé pour illustrer le backtracking.

Problèmes Classiques Résolus par Backtracking

1. Le Problème des N-Reines

Définition Formelle

Données :

N : Taille de l'échiquier (N×N).

Contraintes :

Une seule reine par ligne, colonne et diagonale.

Objectif : Trouver toutes les configurations possibles ou une seule configuration valide.

Problèmes Classiques Résolus par Backtracking

1. Le Problème des N-Reines

Approche Backtracking

Pourquoi utiliser le Backtracking ?

- Exploration systématique des positions possibles.
- Retour en arrière lorsqu'une solution partielle ne respecte pas les contraintes.
- Optimisation : L'algorithme n'explore que les configurations valides.

Problèmes Classiques Résolus par Backtracking

1. Le Problème des N-Reines

1.Placer une reine sur une ligne.

2.Vérifier si la position est valide.

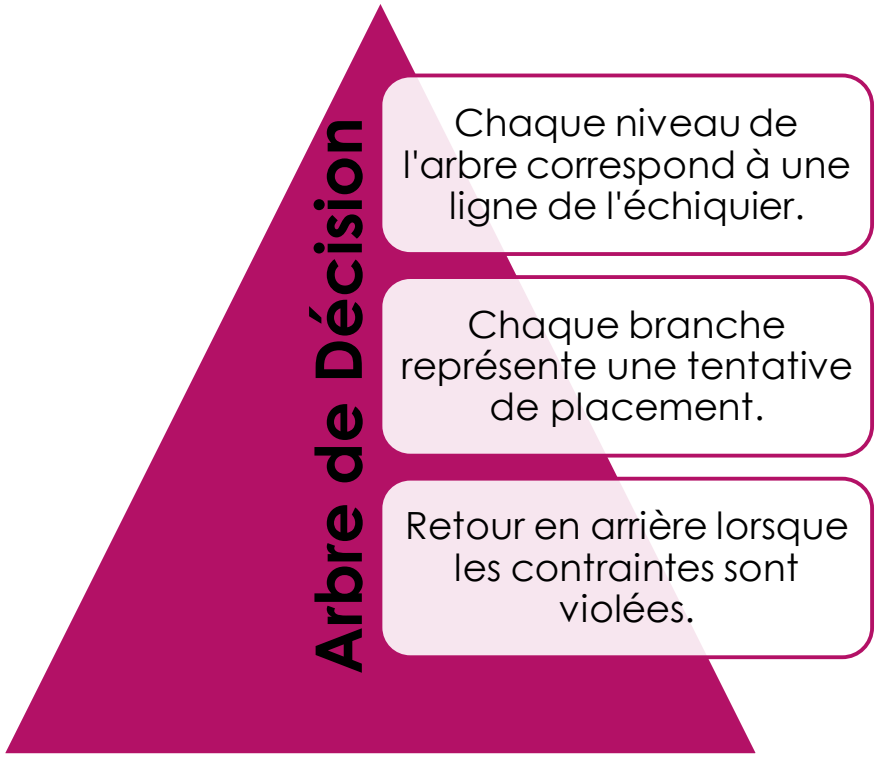
3.Si valide, passer à la ligne suivante.

4.Sinon, revenir en arrière et essayer une autre position.

Schéma général

Problèmes Classiques Résolus par Backtracking

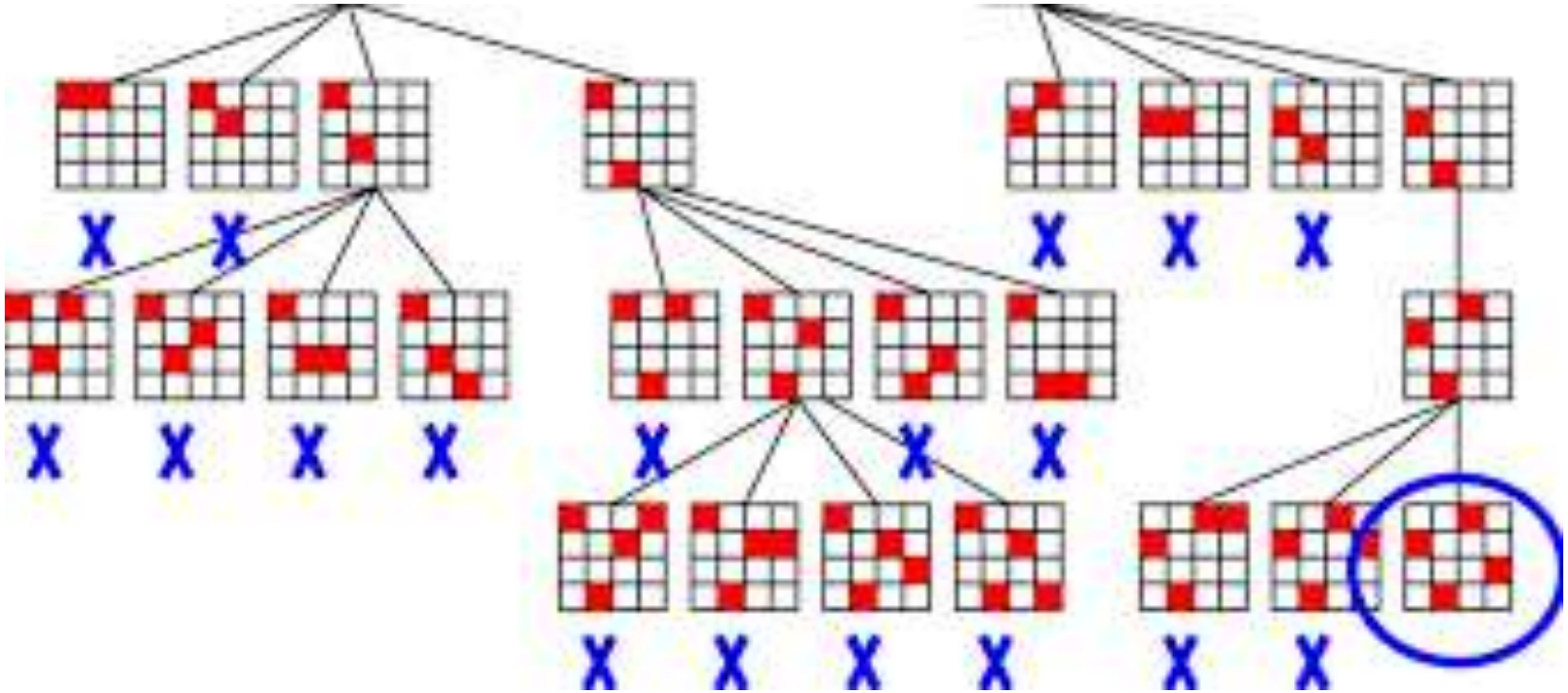
1. Le Problème des N-Reines



Visualisation du Processus

Problèmes Classiques Résolus par Back tracking

1. Le Problème des N-Reines



Problèmes Classiques Résolus par Backtracking

1. Le Problème des N-Reines

Pseudocode de l'Algorithme

```
fonction résoudreNReines(ligne, échiquier) :
    si ligne == N :
        afficher(echiquier)  → Solution trouvée
    sinon :
        pour chaque colonne de 0 à N-1 :
            si positionValide(ligne, colonne, échiquier) :
                placerReine(ligne, colonne, échiquier)
                résoudreNReines(ligne + 1, échiquier)
                retirerReine(ligne, colonne, échiquier)  → Backtrack
```

Problèmes Classiques Résolus par Backtracking

Code python

```python
def est_sûr(echiquier, ligne, col, N):
    for i in range(ligne):
        if echiquier[i] == col:
            return False
    for i, j in zip(range(ligne - 1, -1, -1), range(col - 1, -1, -1)):
        if echiquier[i] == j:
            return False
    for i, j in zip(range(ligne - 1, -1, -1), range(col + 1, N)):
        if echiquier[i] == j:
            return False
    return True

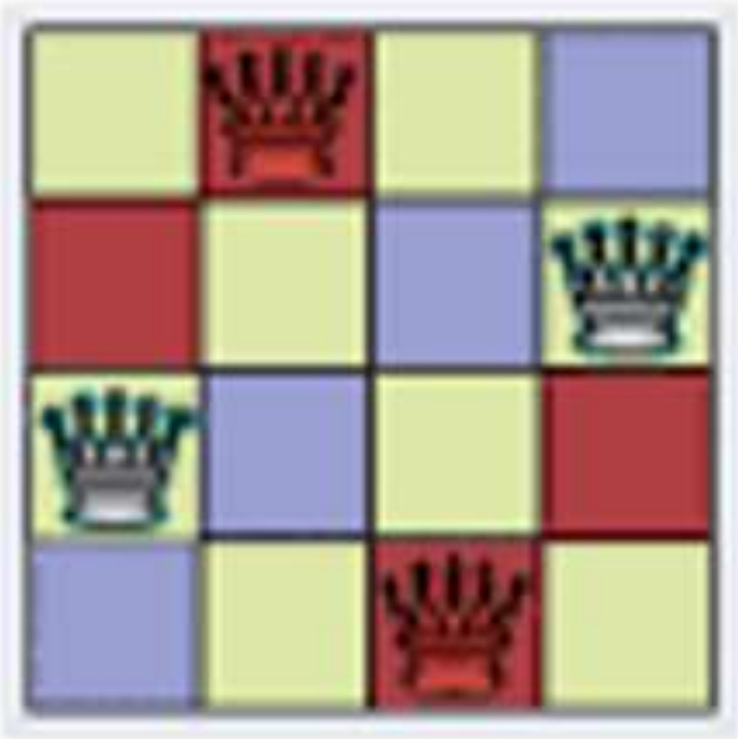
def resoudre_N_reines(echiquier, ligne, N):
    if ligne == N:
        print(echiquier)
        return
    for col in range(N):
        if est_sûr(echiquier, ligne, col, N):
            echiquier[ligne] = col
            resoudre_N_reines(echiquier, ligne + 1, N)
            echiquier[ligne] = -1

N = int(input("Entrer le nombre de reines N : "))
echiquier = [-1] * N
resoudre_N_reines(echiquier, 0, N)
```

Problèmes Classiques Résolus par Backtracking

1. Le Problème des N-Reines

Exemple avec N = 4



Optimisation et Réduction de l'Espace de Recherche

- Dans le cadre du backtracking, l'optimisation de l'espace de recherche est cruciale pour limiter la complexité de calcul, qui, en l'absence de contraintes, croît de manière exponentielle.

- Pour un problème combinatoire avec **n** décisions possibles à chaque étape sur une profondeur de **m**, l'arbre de recherche pourrait théoriquement contenir jusqu'à $O(n^m)$ solutions partielles

Optimisation et Réduction de l'Espace de Recherche

- **Objectif** : Rendre l'algorithme de backtracking plus efficace en évitant d'explorer inutilement des solutions non valides ou non prometteuses.



Optimisation et Réduction de l'Espace de Recherche

1. **Élagage des Branches (Pruning):**

Supprimer les branches de l'arbre de décision qui ne peuvent pas conduire à une solution valide.

Exemples :

- Arrêter si une contrainte est violée.
- Rejeter une configuration partielle dépassant une limite donnée (ex. poids d'un sac à dos).

Optimisation et Réduction de l'Espace de Recherche

2. Contraintes et Heuristiques

Contraintes : Définir des règles pour limiter les choix possibles à chaque étape.

Exemple : Placer une reine uniquement dans une colonne sans conflit.

Heuristiques : Prioriser les choix les plus prometteurs pour explorer d'abord les branches les plus probables.

Exemple : Dans un labyrinthe, commencer par les directions les plus proches de la sortie.

Optimisation et Réduction de l'Espace de Recherche

3. Techniques de Mémorisation

- Stocker les solutions partielles déjà explorées pour éviter de les recalculer.
- Réduire le temps d’exploration pour des sous-problèmes répétitifs.

Optimisation et Réduction de l'Espace de Recherche

4. Limitation de la Profondeur

- Fixer une profondeur maximale dans l'arbre de décision pour réduire la complexité: utilisé dans les problèmes où une solution complète n'est pas nécessaire.