BACKTRACKING (retour sur trace)

Note: Les questions des exercices ne constituent que la trame principale de ce que vous devez faire pour vous guider, rien n'interdit bien sûr que vous alliez plus loin, que vous testiez d'autres choses, bref que vous expérimentiez…

INTRODUCTION

Source Wikipédia :
« Le **retour sur trace** (appelé aussi **backtracking** en anglais) est un algorithme qui consiste à revenir légèrement en arrière sur des décisions prises afin de sortir d'un blocage. La méthode des essais et erreurs constitue un exemple simple de backtracking. Le terme est surtout utilisé en programmation, où il désigne une stratégie pour trouver des solutions à des problèmes de satisfaction de contraintes. »

La résolution d'un problème par la méthode de backtracking repose sur la construction d'une solution partielle que l'on va améliorer afin de s'approcher de plus en plus de la solution finale. Si une solution partielle ne peut pas être améliorée, elle est abandonnée et l'on revient en arrière pour examiner d'autres solutions possibles.

PRINCIPE

Dans l'utilisation du backtracking pour résoudre un problème particulier, nous avons besoin de deux choses :

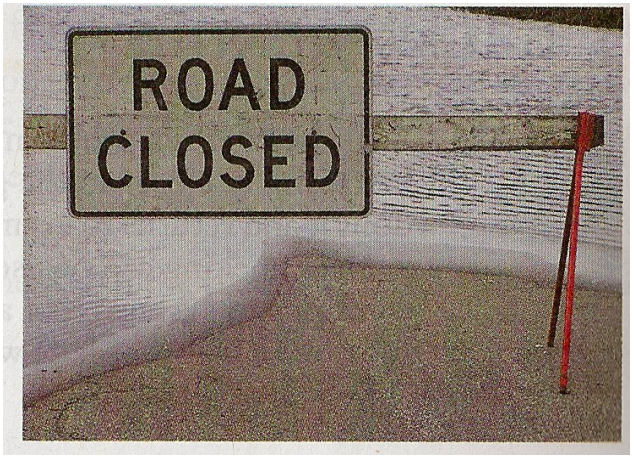
✓ Une procédure pour examiner une solution partielle (notée SP par la suite) afin de déterminer si :

• Il s'agit d'une solution actuelle ACCEPTABLE.

• Il s'agit d'une solution actuelle à ABANDONNER (elle ne respecte pas la règle du jeu).

• Il faut poursuivre L'EXTENSION de la solution actuelle.

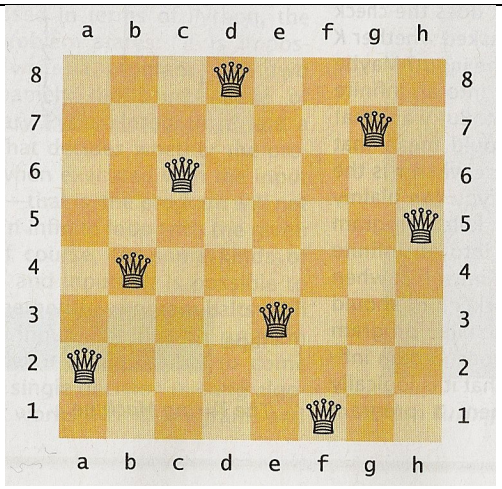
✓ Une procédure pour ETENDRE la SP, générant une ou plusieurs solutions qui se rapprochent de la solution finale.



Dans un algorithme de backtraking, on explore toutes les solutions possibles, quand une solution possible se termine en impasse, on revient en arrière et on teste d'autres solutions.

MISE EN ŒUVRE SUR LE PROBLEME DES HUIT REINES

Dans ce TP, vous devez développer un programme qui détermine toutes les solutions **du problème des huit reines** : Il s'agit de trouver la position de huit reines sur un jeu d'échec de telle façon qu'aucune des reines ne puisse en attaquer une autre. En d'autres termes, il ne peut pas y avoir deux reines sur la même ligne, sur la même colonne et sur la même diagonale.
La figure ci-contre représente une solution possible.



Dans ce problème, il est assez facile d'examiner une SP : Si deux reines s'attaquent, il faut la rejeter. Si les huit reines ne s'attaquent pas, on l'accepte. Autrement, on continue la procédure.
Une SP sera représentée par **une liste dont les éléments sont des chaînes de caractères** (strings) où chaque élément donne la position d'une reine par la notation classique du jeu d'échecs, par exemple : solutionpartielle $= \left["a1","e2","h3","f4"\right]$.

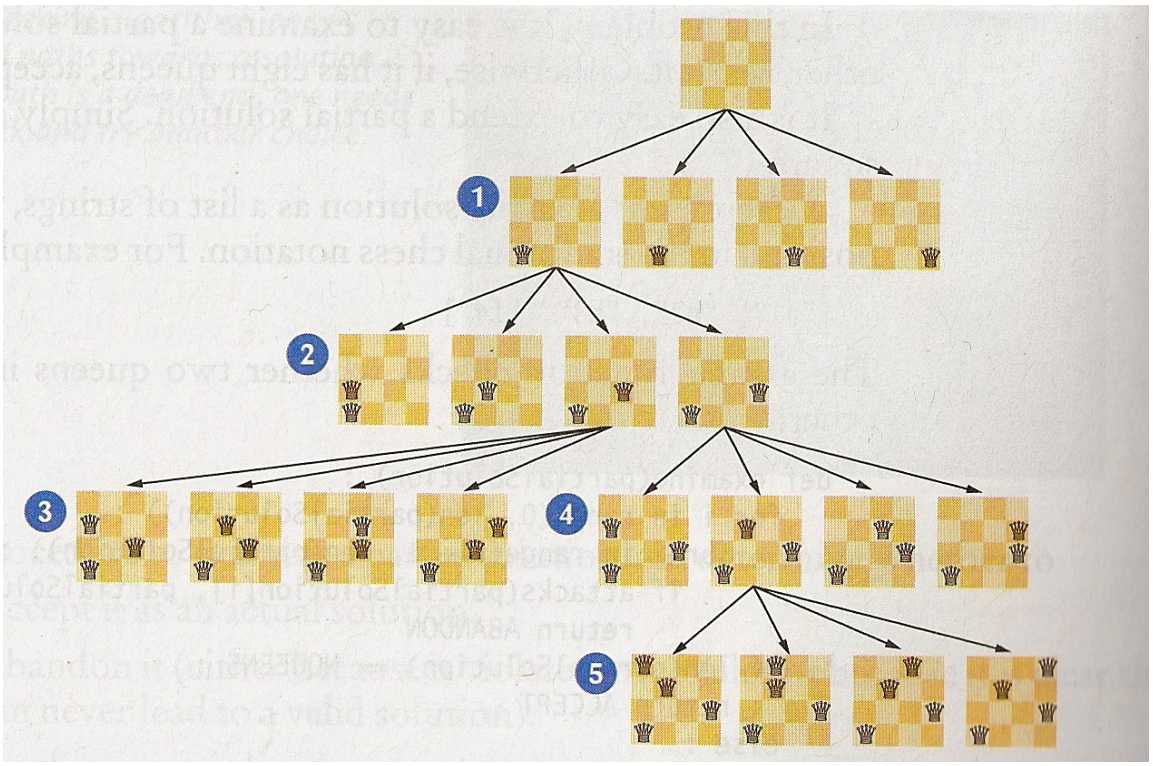
Votre programme pourra en particulier comprendre les fonctions suivantes :

➔ examine (solutionpartielle) qui contrôle si deux reines de la SP s'attaquent.
- Paramètre d'entrée : La liste solutionpartielle.
- Paramètre de sortie : ACCEPTE si la solution est complète, ABANDON si la solution est invalide ou CONTINUE si la solution est partielle.

➔ extend (solutionpartielle) qui, à partir d'une SP, en fait huit copies. Chaque copie prend une nouvelle reine dans une colonne différente.
- Paramètre d'entrée : La liste solutionpartielle.
- Paramètre de sortie : Une liste results de toutes les solutions partielles qui ont une reine ajoutée dans la colonne qui suit.

➔ attack (p1, p2) qui contient l'algorithme qui permet de déterminer si deux reines s'attaquent.
- Paramètre d'entrée : Les positions p1, p2 des deux reines.
- Paramètre de sortie : Une valeur booléenne vraie si deux reines sont sur une même colonne ou une même ligne ou une même diagonale .

➔ solve (solutionpartielle) qui imprime à l'écran toutes les solutions finales à partir de SP qui ont été étendues.
- Paramètre d'entrée : solutionpartielle
- Paramètre de sortie : Aucun



La figure ci-dessus représente une procédure de backtracking pour le problème à 4 reines.