**Thuật toán Backtracking:**

Dùng để giải bài toán liệt kê các cấu hình. Mỗi cấu hình được xây dựng bằng từng phần tử. Mỗi phần tử lại được chọn bằng cách thử tất cả các khả năng.

Các bước trong việc liệt kê cấu hình dạng X[1...n]:

* Xét tất cả các giá trị X[1] có thể nhận, thử X[1] nhận các giá trị đó. Với mỗi giá trị của X[1] ta sẽ:
* Xét tất cả giá trị X[2] có thể nhận, lại thử X[2] cho các giá trị đó. Với mỗi giá trị X[2] lại xét khả năng giá trị của X[3]...tiếp tục như vậy cho tới bước:
* ...
* ....
* Xét tất cả giá trị X[n] có thể nhận, thử cho X[n] nhận lần lượt giá trị đó.
* Thông báo cấu hình tìm được.

Bản chất của quay lui là một quá trình tìm kiếm theo chiều sâu(Depth-First Search).

### \*) Mã giả cho thuật toán quay lui:

Backtracking(k) {

for([Mỗi phương án chọn i(thuộc tập D)]) {

if ([Chấp nhận i]) {

[Chọn i cho X[k]];

if ([Thành công]) {

[Đưa ra kết quả];

} else {

Backtracking(k+1);

[Bỏ chọn i cho X[k]];

}

}

}

}

**Ví dụ: Bài toán tám hâ**

import java.util.ArrayList;

import java.util.List;

public class NQueens {

public static void main(String[] args) {

int n = 8; // Số quân hậu (và kích thước bàn cờ)

List<List<String>> solutions = solveNQueens(n);

System.out.println("Số lời giải cho bài toán " + n + "-Queens: " + solutions.size());

for (List<String> solution : solutions) {

printBoard(solution);

System.out.println();

}

}

public static List<List<String>> solveNQueens(int n) {

List<List<String>> solutions = new ArrayList<>();

char[][] board = new char[n][n];

// Khởi tạo bàn cờ toàn dấu '.'

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

board[i][j] = '.';

}

}

backtrack(0, board, solutions, n);

return solutions;

}

private static void backtrack(int row, char[][] board, List<List<String>> solutions, int n) {

if (row == n) {

// Thêm lời giải vào danh sách

solutions.add(constructSolution(board));

return;

}

for (int col = 0; col < n; col++) {

if (isSafe(row, col, board, n)) {

board[row][col] = 'Q'; // Đặt quân hậu

backtrack(row + 1, board, solutions, n);

board[row][col] = '.'; // Quay lui

}

}

}

private static boolean isSafe(int row, int col, char[][] board, int n) {

// Kiểm tra cột dọc

for (int i = 0; i < row; i++) {

if (board[i][col] == 'Q') {

return false;

}

}

// Kiểm tra đường chéo chính

for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--) {

if (board[i][j] == 'Q') {

return false;

}

}

// Kiểm tra đường chéo phụ

for (int i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++) {

if (board[i][j] == 'Q') {

return false;

}

}

return true;

}

private static List<String> constructSolution(char[][] board) {

List<String> solution = new ArrayList<>();

for (char[] row : board) {

solution.add(new String(row));

}

return solution;

}

private static void printBoard(List<String> board) {

for (String row : board) {

System.out.println(row);

}

}

}