1.A Knights Tour Problem

public class KnightsTourSolver {

private static final int N = 8; // Chessboard size

public static boolean solveKnightsTour(int[][] board, int moveX, int moveY, int moveCount, int[] xMove, int[] yMove) {

if (moveCount == N \* N) {

// All squares visited

return true;

}

for (int i = 0; i < 8; i++) {

int nextX = moveX + xMove[i];

int nextY = moveY + yMove[i];

if (isValidMove(nextX, nextY, board)) {

board[nextX][nextY] = moveCount;

if (solveKnightsTour(board, nextX, nextY, moveCount + 1, xMove, yMove)) {

return true;

}

board[nextX][nextY] = -1; // Backtrack

}

}

return false;

}

private static boolean isValidMove(int x, int y, int[][] board) {

return x >= 0 && x < N && y >= 0 && y < N && board[x][y] == -1;

}

public static void main(String[] args) {

int[][] board = new int[N][N];

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

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

board[i][j] = -1;

}

}

int[] xMove = {2, 1, -1, -2, -2, -1, 1, 2};

int[] yMove = {1, 2, 2, 1, -1, -2, -2, -1};

board[0][0] = 0; // Start from cell (0, 0)

if (solveKnightsTour(board, 0, 0, 1, xMove, yMove)) {

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

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

System.out.print(board[i][j] + "\t");

}

System.out.println();

}

} else {

System.out.println("No solution exists.");

}

}

}

2.A Rat In Maze

public class RatMaze {

final int N = 4;

boolean isSafe(int row, int col, int[][] maze) {

return row >= 0 && row < N && col >= 0 && col < N && maze[row][col] == 1;

}

boolean solveMazeUtil(int[][] maze, int row, int col, int[][] sol) {

if (row == N - 1 && col == N - 1) {

sol[row][col] = 1;

return true;

}

if (isSafe(row, col, maze)) {

sol[row][col] = 1;

if (solveMazeUtil(maze, row, col + 1, sol))

return true;

if (solveMazeUtil(maze, row + 1, col, sol))

return true;

// Backtrack

sol[row][col] = 0;

}

return false;

}

void solveMaze(int[][] maze) {

int[][] sol = new int[N][N];

if (!solveMazeUtil(maze, 0, 0, sol)) {

System.out.println("No valid paths found.");

return;

}

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

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

System.out.print(sol[i][j] + " ");

}

System.out.println();

}

}

public static void main(String[] args) {

RatMaze rat = new RatMaze();

int[][] maze = {

{1, 0, 0, 0},

{1, 1, 0, 1},

{0, 1, 0, 0},

{1, 1, 1, 1}

};

rat.solveMaze(maze);

}

}

3.A N Queen Problem

public class NQueensSolver {

final int N = 8;

boolean isSafe(int[][] board, int row, int col) {

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

if (board[row][i] == 1 || board[i][col] == 1)

return false; // Check row and column

}

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

if (board[i][j] == 1)

return false; // Upper left diagonal

}

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

if (board[i][j] == 1)

return false; // Lower left diagonal

}

return true;

}

boolean solveNQueensUtil(int[][] board, int col) {

if (col >= N)

return true; // All queens placed

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

if (isSafe(board, i, col)) {

board[i][col] = 1;

if (solveNQueensUtil(board, col + 1))

return true;

board[i][col] = 0;

}

}

return false;

}

boolean solveNQueens() {

int[][] board = new int[N][N];

if (!solveNQueensUtil(board, 0)) {

System.out.println("No valid solution exists.");

return false;

}

// Print the solution

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

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

System.out.print(board[i][j] == 1 ? "Q " : ". ");

}

System.out.println();

}

return true;

}

public static void main(String[] args) {

NQueensSolver solver = new NQueensSolver();

solver.solveNQueens();

}

}