Manjula Nannuri

Day 16 and 17_Assignment

Task 1: The Knight's Tour Problem

Create a function bool SolveKnightsTour(int[,] board, int moveX, int moveY, int moveCount, int[] xMove, int[] yMove) that attempts to solve the Knight's Tour problem using backtracking. The function should return true if a solution exists and false otherwise. The board represents the chessboard, moveX and moveY are the current coordinates of the knight, moveCount is the current move count, and xMove[], yMove[] are the possible next moves for the knight. Fill the chessboard such that the knight visits every square exactly once. Keep the chessboard size to 8x8.

```
package com.example;

import java.util.Arrays;

public class KnightTourProblem {

private static final int N = 8;

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

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

public static boolean SolveKnightsTour(int[][] board, int moveX, int moveY, int moveCount)
{

if (moveCount == N * N) {

return true;

}

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

int nextX = moveX + xMove[i];

int nextY = moveY + yMove[i];

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

board[nextX][nextY] = moveCount;

if (SolveKnightsTour(board, nextX, nextY, moveCount + 1)) {

return true;
```

```
} else {
board[nextX][nextY] = -1;
}
}
}
return false;
}
private static boolean isSafe(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[] row : board) {
Arrays.fill(row, -1);
}
board[0][0] = 0;
if (SolveKnightsTour(board, 0, 0, 1)) {
printSolution(board);
} else {
System.out.println("Solution does not exist");
}
}
private static void printSolution(int[][] board) {
for (int[] row : board) {
```

```
for (int cell : row) {

System.out.printf("%2d ", cell);

}

System.out.println();

}

}

}
```

OUTPUT:

```
<terminated> KnightTourProblem [Java Applica
 0 59 38 33 30 17  8 63
37 34 31 60  9 62 29 16
58  1 36 39 32 27 18  7
35 48 41 26 61 10 15 28
42 57  2 49 40 23  6 19
47 50 45 54 25 20 11 14
56 43 52  3 22 13 24  5
51 46 55 44 53  4 21 12
```

Task 2: Rat in a Maze

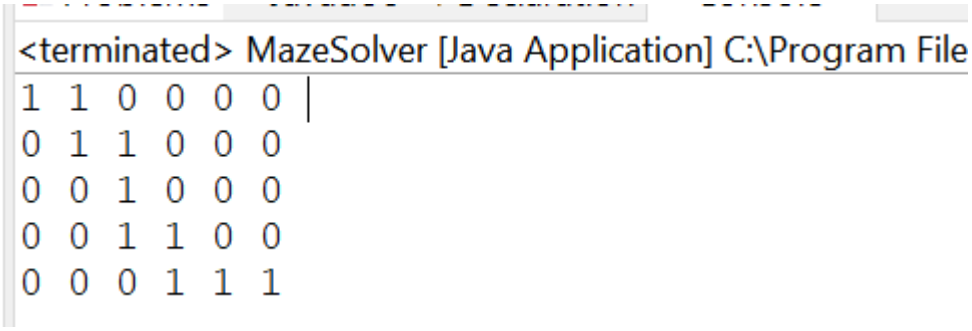
mplement a function bool SolveMaze(int[,] maze) that uses backtracking to find a path from the top left corner to the bottom right corner of a maze. The maze is represented by a 2D array where 1s are paths and 0s are walls. Find a rat's path through the maze. The maze size is 6x6.

```
package com.example;

public class MazeSolver {

private static final int N = 6;

private static void printSolution(int[][] solution) {

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

```
for (int j = 0; j < N; j++) {
System.out.print(solution[i][j] + " ");
}
System.out.println();
}
}
public static boolean SolveMaze(int[][] maze) {
int[][] solution = new int[N][N];
if (!solveMazeUtil(maze, 0, 0, solution)) {
System.out.println("No solution exists");
return false;
}
printSolution(solution);
return true;
}
private static boolean isSafe(int[][] maze, int x, int y) {
return (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1);
}
private static boolean solveMazeUtil(int[][] maze, int x, int y, int[][] solution) {
if (x == N - 1 && y == N - 1) {
solution[x][y] = 1;
return true;
}
if (isSafe(maze, x, y)) {
```

```
solution[x][y] = 1;
if (solveMazeUtil(maze, x + 1, y, solution)) {
return true;
}
if (solveMazeUtil(maze, x, y + 1, solution)) {
return true;
}
solution[x][y] = 0;
return false;
}
return false;
}
public static void main(String[] args) {
int[][] maze = {
{1, 0, 0, 0, 0, 0},
{1, 1, 0, 1, 1, 0},
{0, 1, 1, 0, 1, 0},
{0, 0, 1, 0, 1, 1},
{1, 1, 1, 1, 0, 0},
{1, 0, 0, 1, 1, 1}
};
SolveMaze(maze);
}
}
```

OUTPUT:



Task 3: N Queen Problem

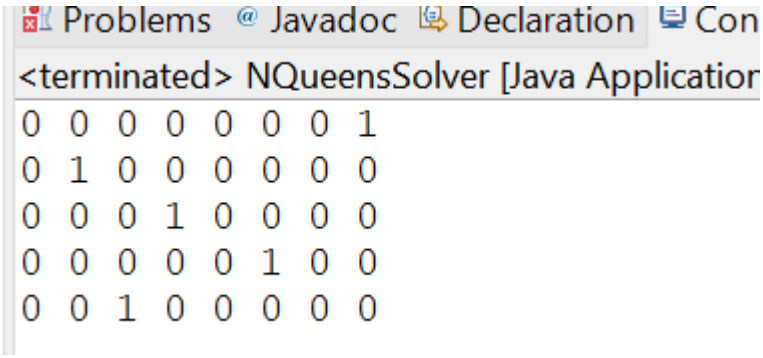
Write a function bool SolveNQueen(int[,] board, int col) in C# that places N queens on an N x N chessboard so that no two queens attack each other using backtracking. Place N queens on the board such that no two queens can attack each other. Use a standard 8x8 chessboard.

```
package com.example;

public class NQueensSolver {

private static final int N = 8;

private static void printSolution(int[][] board) {

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

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

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

}

System.out.println();

}

}

public static boolean SolveNQueens(int[][] board) {

if (!solveNQueensUtil(board, 0)) {

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

return false;
```

```
}
printSolution(board);
return true;
}
private static boolean solveNQueensUtil(int[][] board, int col) {
if (col >= N) {
return true;
}
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;
}
private static boolean isSafe(int[][] board, int row, int col) {
int i, j;
for (i = 0; i < col; i++) {
if (board[row][i] == 1) {
return false;
```

```
}

}

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

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

return false;

}

}

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

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

return false;

}

}

return true;

}

public static void main(String[] args) {

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

if (!SolveNQueens(board)) {

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

}

}

}
```

OUTPUT:


Problems
Javadoc
Declaration
Con
<terminated> NQueensSolver [Java Applicatior
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0