ASSIGNMENT: Day 16 and 17

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.

ANS:

```
public class KnightsTour {


   private static int N = 8;


   public static boolean solveKnightsTour(int[][] board, int moveX, int moveY, int moveCount, int[] xMove, int[] yMove) {
      int nextX, nextY;
      if (moveCount == N * N) {
         return true;
      }


      for (int k = 0; k < 8; k++) {
         nextX = moveX + xMove[k];
         nextY = moveY + yMove[k];
         if (isSafe(nextX, nextY, board)) {
            board[nextX][nextY] = moveCount;
            if (solveKnightsTour(board, nextX, nextY, moveCount + 1, xMove, yMove)) {
               return true;
            } else {
```

```
            board[nextX][nextY] = -1; // Backtracking
        }
      }
    }
    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 x = 0; x < N; x++) {
        for (int y = 0; y < N; y++) {
            board[x][y] = -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;

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

```
            System.out.println("Solution does not exist");
        } else {
            printSolution(board);
        }
    }

    private static void printSolution(int[][] board) {
        for (int x = 0; x < N; x++) {
            for (int y = 0; y < N; y++) {
                System.out.print(board[x][y] + " ");
            }
            System.out.println();
        }
    }
}
//code_by_RUBY
```

Task 2: Rat in a Maze

Implement 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.

ANS:

```
public class RatInMaze {

    private static int N = 6;
```

```
public static boolean solveMaze(int[][] maze) {
    int[][] solution = new int[N][N];
    if (!solveMazeUtil(maze, 0, 0, solution)) {
        System.out.println("Solution doesn't exist");
        return false;
    }

    printSolution(solution);
    return true;
}

private static boolean solveMazeUtil(int[][] maze, int x, int y, int[][] solution) {
    if (x == N - 1 && y == N - 1 && maze[x][y] == 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; // Backtracking
        return false;
    }

    return false;
}

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 void printSolution(int[][] solution) {
    for (int x = 0; x < N; x++) {
        for (int y = 0; y < N; y++) {
            System.out.print(solution[x][y] + " ");
        }
        System.out.println();
    }
}

public static void main(String[] args) {
    int[][] maze = {
```

```
        { 1, 0, 0, 0, 0, 0 },
        { 1, 1, 0, 1, 1, 1 },
        { 0, 1, 0, 1, 0, 0 },
        { 1, 1, 1, 1, 1, 1 },
        { 0, 0, 0, 0, 0, 1 },
        { 1, 1, 1, 1, 1, 1 }
    };

    solveMaze(maze);
  }
}
//code_by_RUBY
```

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.

ANS:

```
public class NQueen {

  private static int N = 8;

  public static boolean solveNQueen(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 (solveNQueen(board, col + 1)) {
                return true;
            }

            board[i][col] = 0; // Backtracking
        }
    }

    return false;
}

private static boolean isSafe(int[][] board, int row, int col) {
    for (int i = 0; i < col; i++) {
        if (board[row][i] == 1) {
            return false;
        }
    }

    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j] == 1) {
```

```
            return false;
        }
    }

    for (int i = row, j = col; j >= 0 && i < N; i++, j--) {
        if (board[i][j] == 1) {
            return false;
        }
    }

    return true;
}

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]==1?"N":"X") + " ");
        }
        System.out.println();
    }
}

public static void main(String[] args) {
    int[][] board = new int[N][N];
```

```
        if (!solveNQueen(board, 0)) {
            System.out.println("Solution does not exist");
        } else {
            printSolution(board);
        }
    }
}


//code_by_RUBY
```