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.

```
public class KnightsTour {


    static final int N = 8;


    public static boolean SolveKnightsTour(int[][] board, int moveX, int moveY, int moveCount,
int[] xMove, int[] yMove) {
        if (moveCount == N * N) {
            return true;
        }
        board[moveX][moveY] = moveCount + 1;
        for (int i = 0; i < 8; i++) {
            int nextX = moveX + xMove[i];
            int nextY = moveY + yMove[i];
            if (isSafe(board, nextX, nextY)) {
                if (SolveKnightsTour(board, nextX, nextY, moveCount + 1, xMove, yMove)) {
                    return true;
                }
            }
        }
        board[moveX][moveY] = 0; // Unmark the square
        return false;
```

```
    }

    public static boolean isSafe(int[][] board, int x, int y) {
        return (x >= 0 && x < N && y >= 0 && y < N && board[x][y] == 0);
    }

    public static void main(String[] args) {
        int[][] board = new int[N][N];
        int[] xMove = {2, 1, -1, -2, -2, -1, 1, 2};
        int[] yMove = {1, 2, 2, 1, -1, -2, -2, -1};
        if (SolveKnightsTour(board, 0, 0, 0, xMove, yMove)) {
            System.out.println("Knight's Tour solved!");
        } else {
            System.out.println("No solution exists for the Knight's Tour problem.");
        }
    }
}
```

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.

```
public class MazeSolver {

  static final int N = 6;

  public static boolean SolveMaze(int[][] maze) {
    int[][] visited = new int[N][N];
    return solveMazeUtil(maze, 0, 0, visited);
  }

  private static boolean solveMazeUtil(int[][] maze, int row, int col, int[][] visited) {
    if (row == N - 1 && col == N - 1 && maze[row][col] == 1) {
      return true;
    }
    if (row < 0 || col < 0 || row >= N || col >= N || maze[row][col] == 0 || visited[row][col] == 1)
{
      return false;
    }
    visited[row][col] = 1;
    if (solveMazeUtil(maze, row + 1, col, visited)) {
      return true;
    }
    if (solveMazeUtil(maze, row, col + 1, visited)) {
      return true;
    }
    if (solveMazeUtil(maze, row - 1, col, visited)) {
```

```
    return true;
   }
   if (solveMazeUtil(maze, row, col - 1, visited)) {
    return true;
   }
   visited[row][col] = 0;
   return false;
  }

  public static void main(String[] args) {
   int[][] maze = {
    {1, 0, 0, 0},
    {1, 1, 0, 1},
    {0, 1, 0, 0},
    {1, 1, 1, 1}
   };

   if (SolveMaze(maze)) {
    System.out.println("Maze solved! A path exists from top left to bottom right.");
   } else {
    System.out.println("No solution exists for the given maze.");
   }
  }
}
```

Task 3: N Queen Problem

Write a function bool Solve NQueen(int[,] board, int col) in JAVA 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.

```
public class NQueens {

  static final int N = 8;

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

    return true;
  }

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

    if (SolveNQueen(board, 0)) {
      System.out.println("N-Queens problem solved!");
    } else {
      System.out.println("No solution exists for the N-Queens problem on an 8x8 chessboard.");
    }
  }
}
```