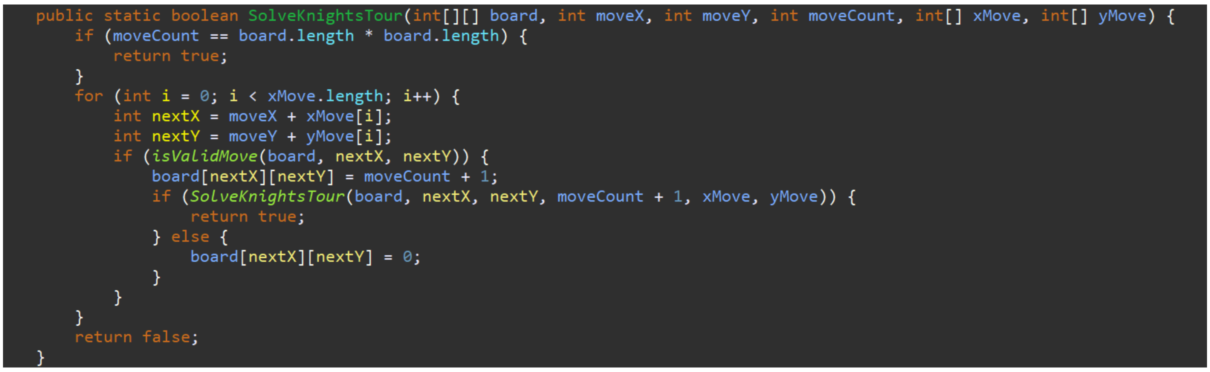
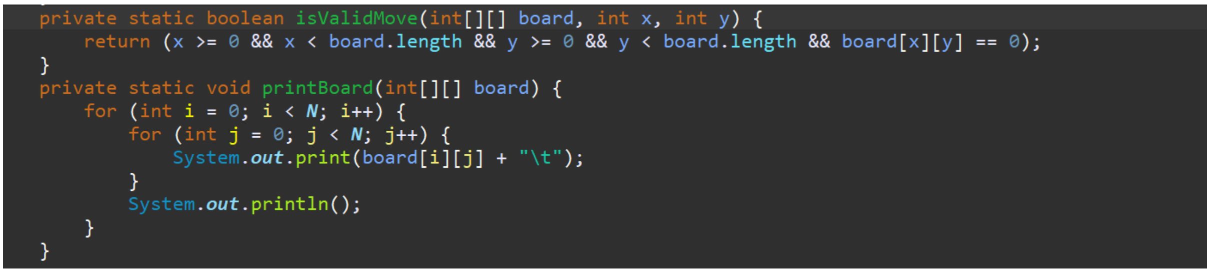
Task 1: The Knight’s Tour Problem

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.

Function: -

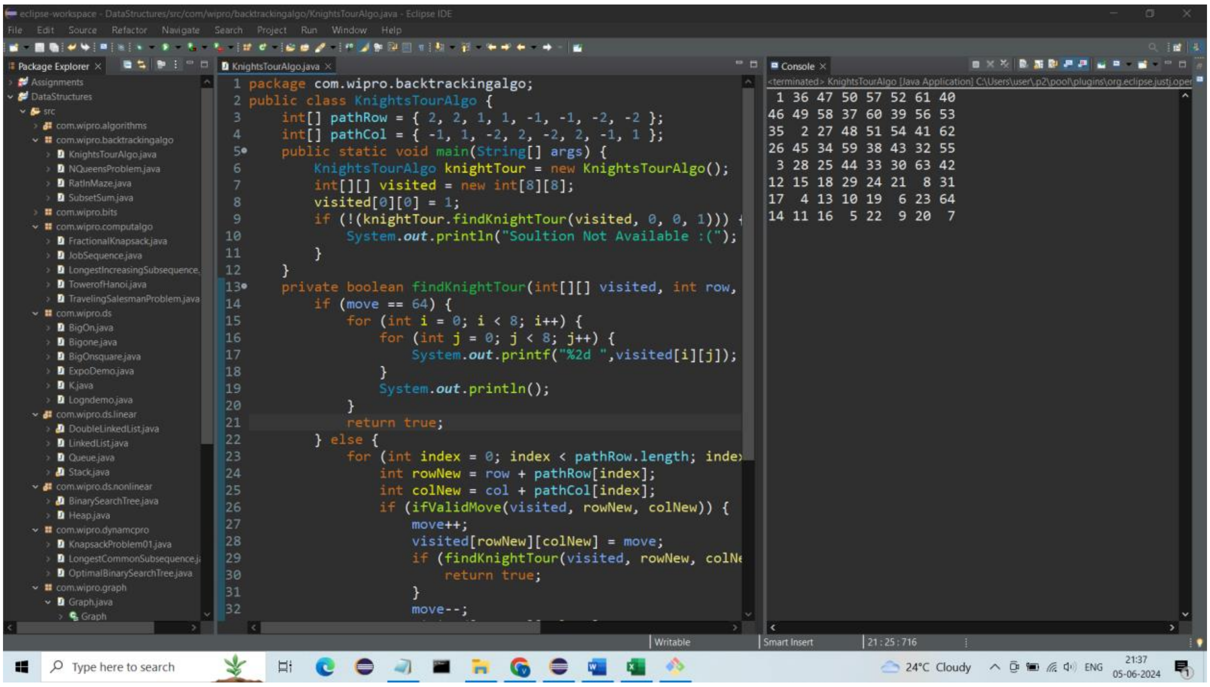


```
public static boolean SolveKnightsTour(int[][] board, int moveX, int moveY, int moveCount, int[] xMove, int[] yMove) {
    if (moveCount == board.length * board.length) {
        return true;
    }
    for (int i = 0; i < xMove.length; i++) {
        int nextX = moveX + xMove[i];
        int nextY = moveY + yMove[i];
        if (isValidMove(board, nextX, nextY)) {
            board[nextX][nextY] = moveCount + 1;
            if (SolveKnightsTour(board, nextX, nextY, moveCount + 1, xMove, yMove)) {
                return true;
            } else {
                board[nextX][nextY] = 0;
            }
        }
    }
    return false;
}
```



```
private static boolean isValidMove(int[][] board, int x, int y) {
    return (x >= 0 && x < board.length && y >= 0 && y < board.length && board[x][y] == 0);
}
private static void printBoard(int[][] board) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            System.out.print(board[i][j] + "\t");
        }
        System.out.println();
    }
}
```

Output: -


eclipse-workspace - DataStructures/src/com/wipro/backtrackingalgo/KnightsTourAlgo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
KnightsTourAlgo.java
Console
Writable
Smart Insert
21 : 25 : 716
Type here to search
24°C Cloudy
ENG
21:37
05-06-2024

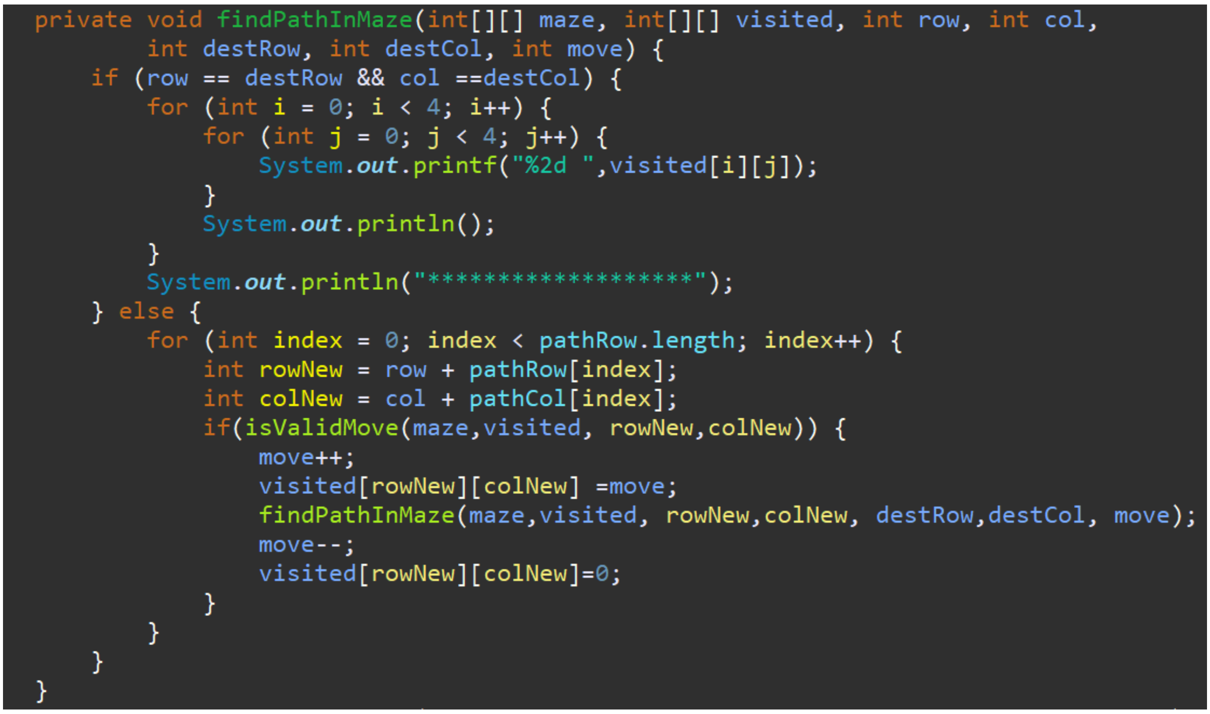
```
package com.wipro.backtrackingalgo;
public class KnightsTourAlgo {
    int[] pathRow = { 2, 2, 1, 1, -1, -1, -2, -2 };
    int[] pathCol = { -1, 1, -2, 2, -2, 2, -1, 1 };
    public static void main(String[] args) {
        KnightsTourAlgo knightTour = new KnightsTourAlgo();
        int[][] visited = new int[8][8];
        visited[0][0] = 1;
        if (!(knightTour.findKnightTour(visited, 0, 0, 1))) {
            System.out.println("Soultion Not Available :(");
        }
    }
    private boolean findKnightTour(int[][] visited, int row,
        if (move == 64) {
            for (int i = 0; i < 8; i++) {
                for (int j = 0; j < 8; j++) {
                    System.out.printf("%2d ",visited[i][j]);
                }
                System.out.println();
            }
            return true;
        } else {
            for (int index = 0; index < pathRow.length; inde
                int rowNew = row + pathRow[index];
                int colNew = col + pathCol[index];
                if (ifValidMove(visited, rowNew, colNew)) {
                    move++;
                    visited[rowNew][colNew] = move;
                    if (findKnightTour(visited, rowNew, colNe
                        return true;
                    }
                    move--;
```

```
<terminated> KnightsTourAlgo [Java Application] C:\Users\user\.p2\pool\plugins\org.eclipse.justj.oper
 1 36 47 50 57 52 61 40
46 49 58 37 60 39 56 53
35  2 27 48 51 54 41 62
26 45 34 59 38 43 32 55
 3 28 25 44 33 30 63 42
12 15 18 29 24 21  8 31
17  4 13 10 19  6 23 64
14 11 16  5 22  9 20  7
```

Task 2: Rat in a Maze

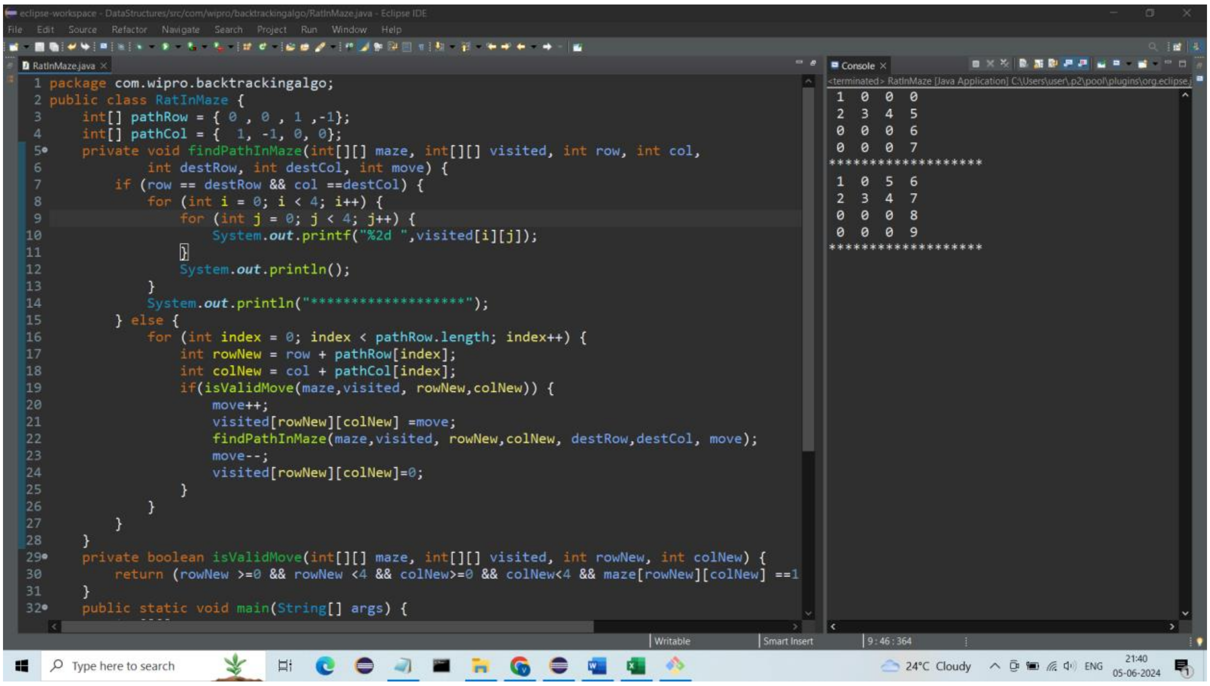
Implement a function bool SolveMaze(int[,] maze) that uses backtracking to find a path from the top left corner to the bottom right corner of a maze. The maze is represented by a 2D array where 1s are paths and 0s are walls. Find a rat's path through the maze. The maze size is 6x6.

Function: -



```java
private void findPathInMaze(int[][] maze, int[][] visited, int row, int col,
        int destRow, int destCol, int move) {
    if (row == destRow && col ==destCol) {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                System.out.printf("%2d ",visited[i][j]);
            }
            System.out.println();
        }
        System.out.println("*******************");
    } else {
        for (int index = 0; index < pathRow.length; index++) {
            int rowNew = row + pathRow[index];
            int colNew = col + pathCol[index];
            if(isValidMove(maze,visited, rowNew,colNew)) {
                move++;
                visited[rowNew][colNew] =move;
                findPathInMaze(maze,visited, rowNew,colNew, destRow,destCol, move);
                move--;
                visited[rowNew][colNew]=0;
            }
        }
    }
}
```

Output: -



eclipse-workspace - DataStructures/src/com/wipro/backtrackingalgo/RatInMaze.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
RatInMaze.java
Console
Writable
Smart Insert
9 : 46 : 364
Type here to search
24°C Cloudy
ENG
21:40
05-06-2024

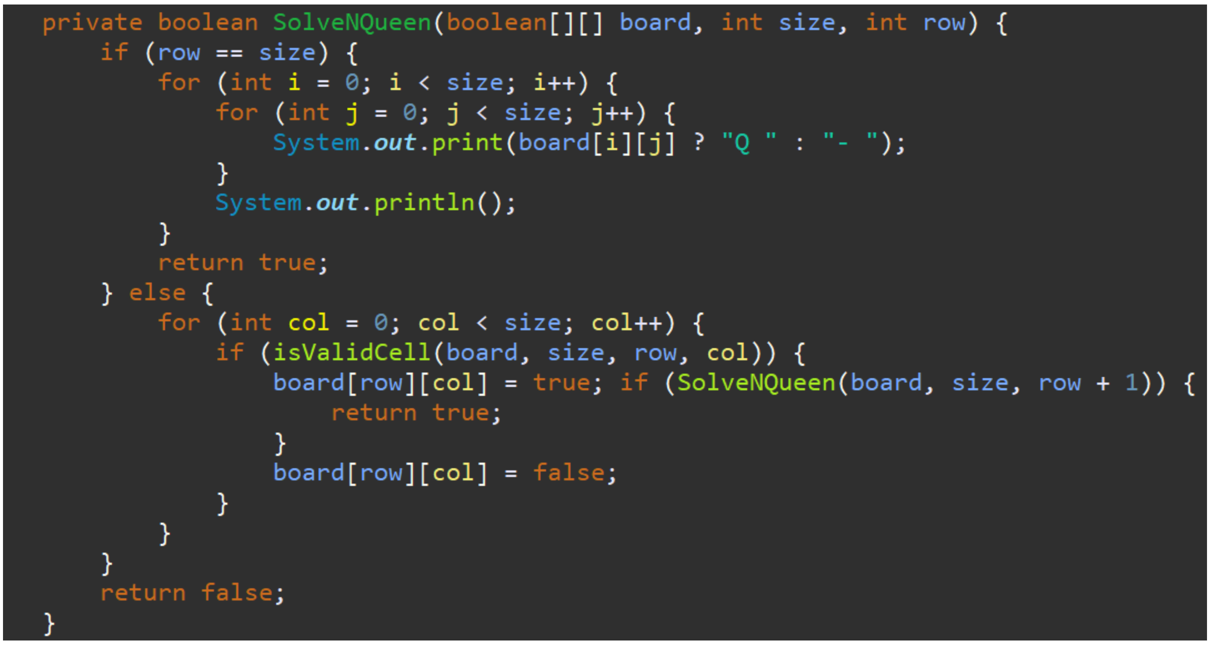
```java
package com.wipro.backtrackingalgo;
public class RatInMaze {
    int[] pathRow = { 0 , 0 , 1 ,-1};
    int[] pathCol = {  1, -1, 0, 0};
    private void findPathInMaze(int[][] maze, int[][] visited, int row, int col,
            int destRow, int destCol, int move) {
        if (row == destRow && col ==destCol) {
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                    System.out.printf("%2d ",visited[i][j]);
                }
                System.out.println();
            }
            System.out.println("*******************");
        } else {
            for (int index = 0; index < pathRow.length; index++) {
                int rowNew = row + pathRow[index];
                int colNew = col + pathCol[index];
                if(isValidMove(maze,visited, rowNew,colNew)) {
                    move++;
                    visited[rowNew][colNew] =move;
                    findPathInMaze(maze,visited, rowNew,colNew, destRow,destCol, move);
                    move--;
                    visited[rowNew][colNew]=0;
                }
            }
        }
    }
    private boolean isValidMove(int[][] maze, int[][] visited, int rowNew, int colNew) {
        return (rowNew >=0 && rowNew <4 && colNew>=0 && colNew<4 && maze[rowNew][colNew] ==1
    }
    public static void main(String[] args) {
```

```
<terminated> RatInMaze [Java Application] C:\Users\user\.p2\pool\plugins\org.eclipse.j
 1  0  0  0
 2  3  4  5
 0  0  0  6
 0  0  0  7
*******************
 1  0  5  6
 2  3  4  7
 0  0  0  8
 0  0  0  9
*******************
```

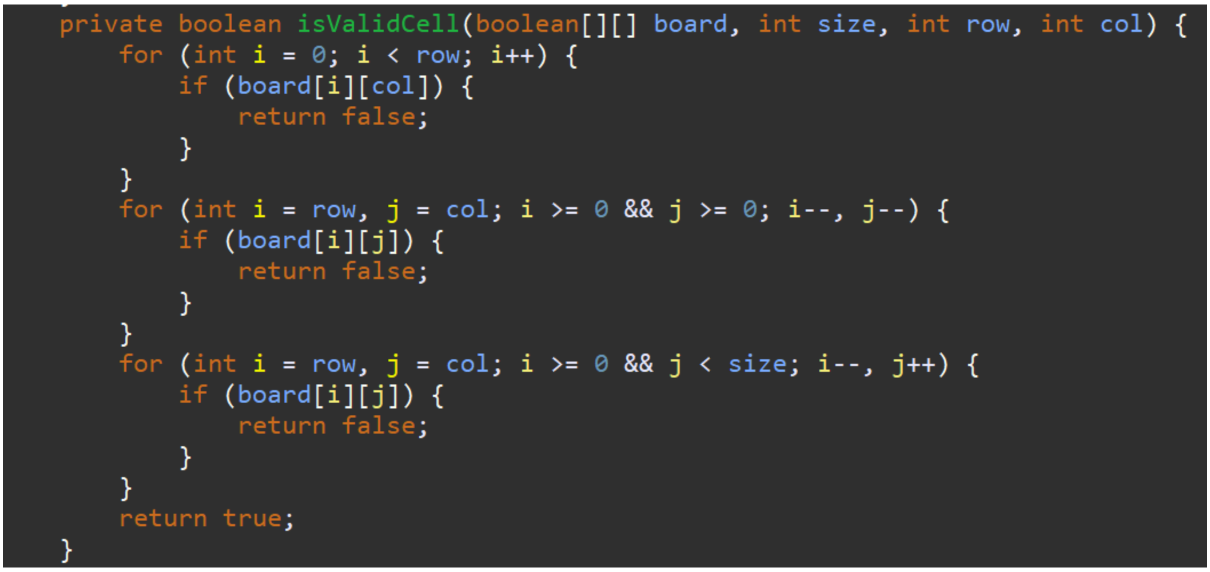
Task 3: N Queen Problem

Write a function bool SolveNQueen(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.

Function: -

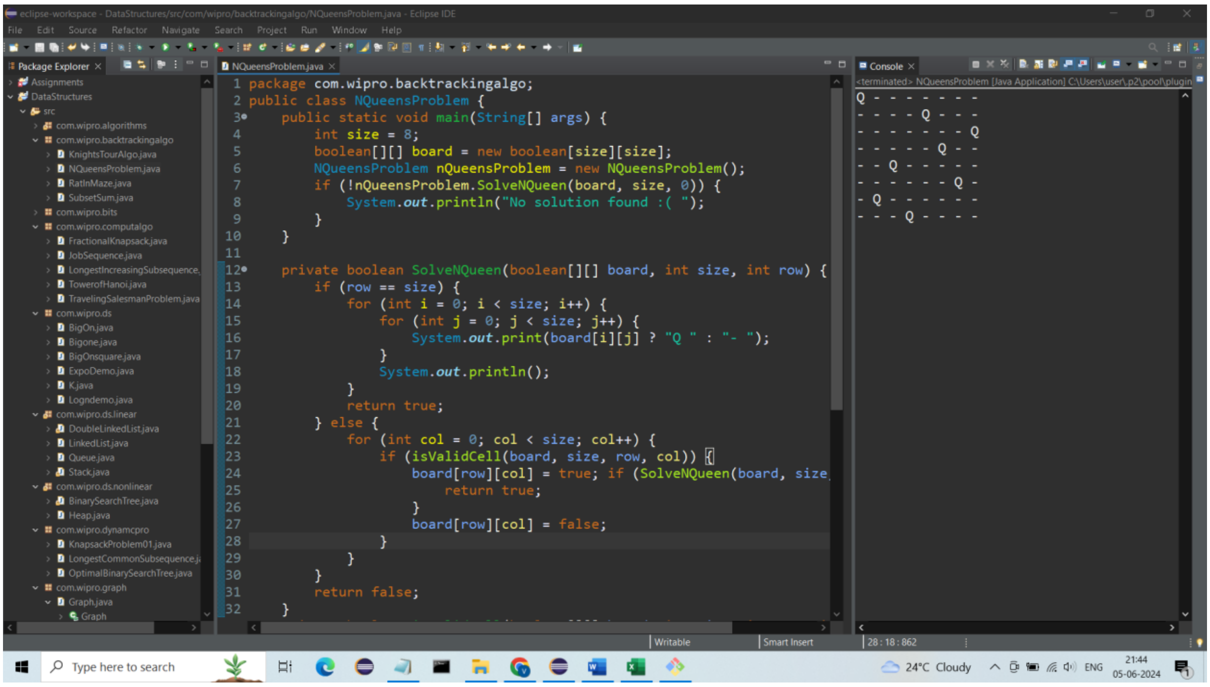


```java
private boolean SolveNQueen(boolean[][] board, int size, int row) {
    if (row == size) {
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                System.out.print(board[i][j] ? "Q " : "- ");
            }
            System.out.println();
        }
        return true;
    } else {
        for (int col = 0; col < size; col++) {
            if (isValidCell(board, size, row, col)) {
                board[row][col] = true; if (SolveNQueen(board, size, row + 1)) {
                    return true;
                }
                board[row][col] = false;
            }
        }
    }
    return false;
}
```



```java
private boolean isValidCell(boolean[][] board, int size, int row, int col) {
    for (int i = 0; i < row; i++) {
        if (board[i][col]) {
            return false;
        }
    }
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j]) {
            return false;
        }
    }
    for (int i = row, j = col; i >= 0 && j < size; i--, j++) {
        if (board[i][j]) {
            return false;
        }
    }
    return true;
}
```

Output: -



```java
package com.wipro.backtrackingalgo;
public class NQueensProblem {
    public static void main(String[] args) {
        int size = 8;
        boolean[][] board = new boolean[size][size];
        NQueensProblem nQueensProblem = new NQueensProblem();
        if (!nQueensProblem.SolveNQueen(board, size, 0)) {
            System.out.println("No solution found :( ");
        }
    }

    private boolean SolveNQueen(boolean[][] board, int size, int row) {
        if (row == size) {
            for (int i = 0; i < size; i++) {
                for (int j = 0; j < size; j++) {
                    System.out.print(board[i][j] ? "Q " : "- ");
                }
                System.out.println();
            }
            return true;
        } else {
            for (int col = 0; col < size; col++) {
                if (isValidCell(board, size, row, col)) {
                    board[row][col] = true; if (SolveNQueen(board, size
                        return true;
                    }
                    board[row][col] = false;
                }
            }
        }
        return false;
    }
```

```
<terminated> NQueensProblem [Java Application] C:\Users\user\.p2\pool\plugin
Q - - - - - - -
- - - - Q - - -
- - - - - - - Q
- - - - - Q - -
- - Q - - - - -
- - - - - - Q -
- Q - - - - - -
- - - Q - - - -
```