

RPS DAY 16-17 Assignments

Assignment 8

Name: Akshada Baad

Batch - CPPE

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.

```
import java.util.Arrays;

public class KnightsTour {

    static int N = 8;

    static boolean isSafe(int x, int y, int[][] board) {
        return (x >= 0 && x < N && y >= 0 && y < N && board[x][y] == -1);
    }

    static boolean solveKT() {
        int[][] board = new int[N][N];
        for (int[] row : board)
            Arrays.fill(row, -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 (!solveKTUtil(0, 0, 1, board, xMove, yMove)) {
        System.out.println("Solution does not exist");
        return false;
    } else {
        printSolution(board);
    }

    return true;
}

static boolean solveKTUtil(int x, int y, int movei, int[][] board, int[] xMove, int[] yMove) {
    int k, next_x, next_y;
    if (movei == N * N)
        return true;

    for (k = 0; k < 8; k++) {
        next_x = x + xMove[k];
        next_y = y + yMove[k];
        if (isSafe(next_x, next_y, board)) {
            board[next_x][next_y] = movei;
            if (solveKTUtil(next_x, next_y, movei + 1, board, xMove, yMove))
                return true;
            else
                board[next_x][next_y] = -1;
        }
    }

    return false;
}

```

```

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();
    }
}

```

```

public static void main(String[] args) {
    solveKT();
}

```

OUTPUT:

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Lists the project structure, including the 'com.wipro.java' package and various Java files like 'BalancedBinary.java', 'BFS.java', 'BitManipulation.java', etc.
- Main Editor:** Displays the 'KnightsTour.java' file. The code includes:


```

package com.wipro.java;
import java.util.Arrays;
public class KnightsTour {
    static int N = 8;

    static boolean isSafe(int x, int y, int[][] board) {
        return (x >= 0 && x < N && y >= 0 && y < N && board[x][y] == -1);
    }

    static boolean solveKT() {
        int[][] board = new int[N][N];
        for (int row : board)
            Arrays.fill(row, -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 (!solveKTUtil(0, 0, 1, board, xMove, yMove)) {
            System.out.println("Solution does not exist");
            return false;
        } else {
            printSolution(board);
        }
        return true;
    }

    static boolean solveKTUtil(int x, int y, int moveI, int[][] board, int[] xMove, int[] yMove) {
        // ... (rest of the recursive logic)
    }
}

```
- Console:** Shows the output of the program, which is a 6x6 grid of numbers representing the knight's path:


```

0 59 30 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

```
- Problems View:** Shows 40 warnings and 0 errors.

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 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 && visited[rowNew][colNew] == 0);

}

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

    int[][] visited = new int[4][4];
    visited[0][0] = 1;

    RatInMaze ratInMaze = new RatInMaze();
    ratInMaze.findPathInMaze(maze, visited, 0 ,0 ,3,3, 1);

}

```

}

OUTPUT:

```
1 package com.wipro.java;
2
3 public class RatInMaze {
4     int[] pathRow = { 0, 0, 1, -1};
5     int[] pathCol = { 1, -1, 0, 0};
6
7     private void findRatInMaze(int[][] maze, int[][] visited, int row, int col, int destRow, int destCol) {
8         if (row == destRow && col == destCol) {
9             for (int i = 0; i < 4; i++) {
10                 for (int j = 0; j < 4; j++) {
11                     System.out.print("%2d ", visited[i][j]);
12                 }
13                 System.out.println();
14             }
15             System.out.println("*****");
16         } else {
17             for (int index = 0; index < pathRow.length; index++) {
18                 int rowNew = row + pathRow[index];
19                 int colNew = col + pathCol[index];
20                 if (isValidMove(maze, visited, rowNew, colNew)) {
21                     move++;
22                     visited[rowNew][colNew] = move;
23                     findRatInMaze(maze, visited, rowNew, colNew, destRow, destCol, move);
24                     move--;
25                     visited[rowNew][colNew] = 0;
26                 }
27             }
28         }
29     }
30 }
31
```

```
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 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.

```
public class NQueensProblem {
```

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

```
        int size = 8;
```

```
        boolean[][] board = new boolean[size][size];
```

```
        NQueensProblem nQueensProblem = new NQueensProblem();
```

```
        if (!nQueensProblem.nQueen(board, size, 0)) {
```

```
            System.out.println("No solution found :( ");
```

```
        }
```

```
    }
```

```

private boolean nQueen(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 (nQueen(board, size, row + 1)) {
                    return true;
                }
                board[row][col] = false; // backtrack
            }
        }
    }
    return false;
}

```

```

private boolean isValidCell(boolean[][] board, int size, int row, int col) {
    // check column
    for (int i = 0; i < row; i++) {
        if (board[i][col]) {
            return false;
        }
    }
}

```

```
}
```

```
// check upper left diagonal
```

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

```
    if (board[i][j]) {
```

```
        return false;
```

```
    }
```

```
}
```

```
// check upper right diagonal
```

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

```
    if (board[i][j]) {
```

```
        return false;
```

```
    }
```

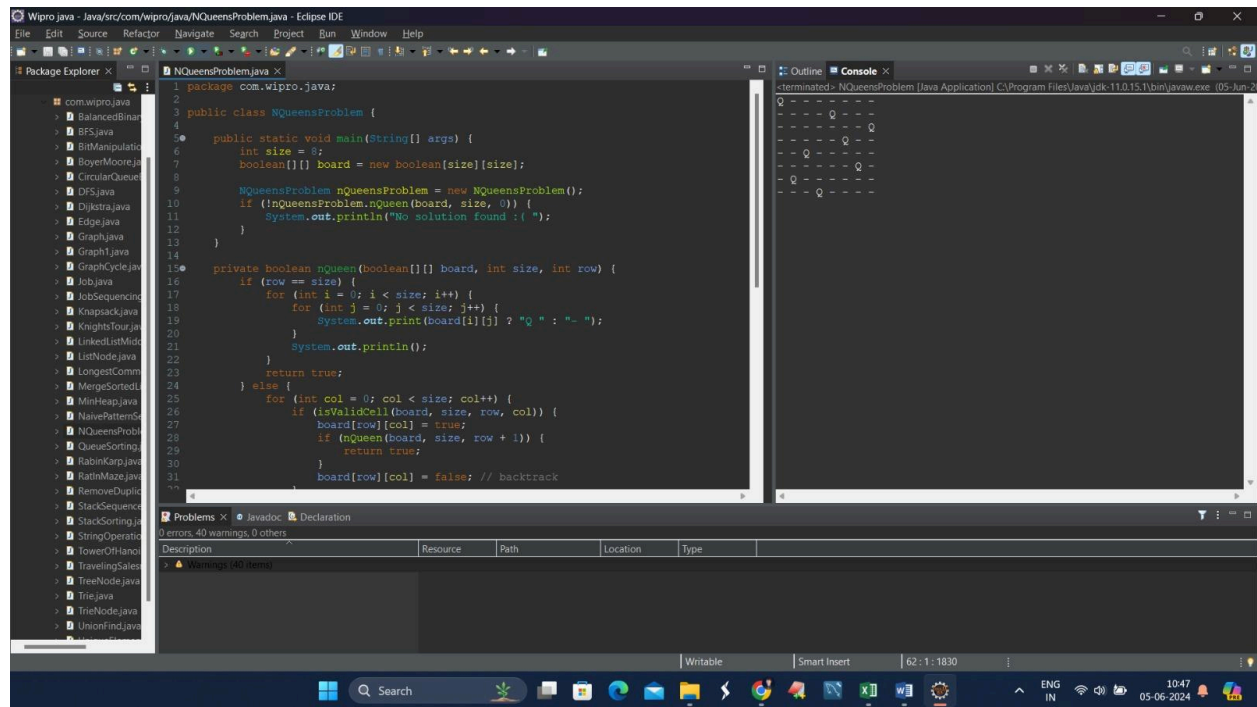
```
}
```

```
return true;
```

```
}
```

```
}
```


OUTPUT:



The screenshot displays the Eclipse IDE interface. The main editor shows the `NQueensProblem.java` file with the following code:

```
1 package com.wipro.java;
2
3 public class NQueensProblem {
4
5     public static void main(String[] args) {
6         int size = 8;
7         boolean[][] board = new boolean[size][size];
8
9         NQueensProblem nQueensProblem = new NQueensProblem();
10        if (!nQueensProblem.nQueen(board, size, 0)) {
11            System.out.println("No solution found :( ");
12        }
13    }
14
15    private boolean nQueen(boolean[][] board, int size, int row) {
16        if (row == size) {
17            for (int i = 0; i < size; i++) {
18                for (int j = 0; j < size; j++) {
19                    System.out.print(board[i][j] ? "Q " : "- ");
20                }
21                System.out.println();
22            }
23            return true;
24        } else {
25            for (int col = 0; col < size; col++) {
26                if (isValidCell(board, size, row, col)) {
27                    board[row][col] = true;
28                    if (nQueen(board, size, row + 1)) {
29                        return true;
30                    }
31                    board[row][col] = false; // backtrack
32                }
33            }
34        }
35    }
36
37    private boolean isValidCell(boolean[][] board, int size, int row, int col) {
38        for (int i = 0; i < row; i++) {
39            if (board[i][col]) {
40                return false;
41            }
42        }
43        for (int i = 0; i < size; i++) {
44            if (board[row][i]) {
45                return false;
46            }
47        }
48        int diff = 1;
49        while (col - diff > -1) {
50            if (board[row - diff][col - diff]) {
51                return false;
52            }
53            diff++;
54        }
55        diff = 1;
56        while (col + diff < size) {
57            if (board[row - diff][col + diff]) {
58                return false;
59            }
60            diff++;
61        }
62        return true;
63    }
64}
```

The console output shows the following pattern:

```
<terminated> NQueensProblem [Java Application] C:\Program Files\Java\jdk-11.0.15\bin\javaw.exe (05-Jun-2024)
Q - - - - -
- - - Q - - -
- - - - - Q - -
- - - - - - - Q -
- - - - - - - - Q -
- - - - - - - - - Q
- - - - - - - - - - Q
- - - - - - - - - - - Q
```

The Problems view at the bottom shows 0 errors, 40 warnings, and 0 others.