

# Assignment Day-16&17

## Core Java with DS and Algorithms

Name: Joshnitha Rangolu

### 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 day16and17;

public class KnightTour {

    static int N = 8;

    static boolean isSafe(int x, int y, int sol[][])

    {

        return (x >= 0 && x < N && y >= 0 && y < N

        && sol[x][y] == -1);

    }

    static void printSolution(int sol[][])

    {

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

            for (int y = 0; y < N; y++)

                System.out.print(sol[x][y] + " ");

            System.out.println();

        }

    }

    static boolean solveKT()

    {
```

```

int sol[][] = new int[8][8];

for (int x = 0; x < N; x++)

for (int y = 0; y < N; y++)

sol[x][y] = -1;

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

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

sol[0][0] = 0;

if (!solveKTUtil(0, 0, 1, sol, xMove, yMove)) {

System.out.println("Solution does not exist");

return false;

}

else

printSolution(sol);

return true;

}

static boolean solveKTUtil(int x, int y, int movei,

int sol[][], 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, sol)) {

sol[next_x][next_y] = movei;

if (solveKTUtil(next_x, next_y, movei + 1,

```

```

    sol, xMove, yMove))

    return true;

else

    sol[next_x][next_y]
    = -1; // backtracking

}

}

return false;

}

public static void main(String args[])

{

    solveKT();

}

}

```

The screenshot shows the Eclipse IDE with the KnightTour.java file open. The code implements a Knight's Tour solution using backtracking. The console window displays the output of the program, which is an 8x8 grid of numbers representing the sequence of moves.

```

1 package day16and17;
2
3 public class KnightTour {
4     static int N = 8;
5     static boolean isSafe(int x, int y, int sol[][])
6     {
7         return (x >= 0 && x < N && y >= 0 && y < N
8             && sol[x][y] == -1);
9     }
10    static void printSolution(int sol[][])
11    {
12        for (int x = 0; x < N; x++) {
13            for (int y = 0; y < N; y++)
14                System.out.print(sol[x][y] + " ");
15            System.out.println();
16        }
17    }
18    static boolean solveKT()

```

Console Output:

```

<terminated> KnightTour [Java Application] C:\Users\DELL\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v2023020
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

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.

```
package day16and17;

import java.util.ArrayList;

public class RatInMaze {

    static String direction = "DLRU";

    static int[] dr = { 1, 0, 0, -1 };

    static int[] dc = { 0, -1, 1, 0 };

    static boolean isValid(int row, int col, int n, int[][] maze)

    {

        return row >= 0 && col >= 0 && row < n && col < n

        && maze[row][col] == 1;

    }

    static void findPath(int row, int col, int[][] maze,

        int n, ArrayList<String> ans,

        StringBuilder currentPath)

    {

        if (row == n - 1 && col == n - 1) {

            ans.add(currentPath.toString());

            return;

        }

        maze[row][col] = 0;

        for (int i = 0; i < 4; i++) {

            int nextrow = row + dr[i];

            int nextcol = col + dc[i];

            if (isValid(nextrow, nextcol, n, maze)) {
```

```

currentPath.append(direction.charAt(i));

findPath(nextrow, nextcol, maze, n, ans, currentPath);

currentPath.deleteCharAt(currentPath.length() - 1);

}

}

maze[row][col] = 1;

}

public static void main(String[] args)

{

    int[][] maze = { { 1, 0, 0, 0 },

    { 1, 1, 0, 1 },

    { 1, 1, 0, 0 },

    { 0, 1, 1, 1 } };

    int n = maze.length;

    ArrayList<String> result = new ArrayList<>();

    StringBuilder currentPath = new StringBuilder();

    if (maze[0][0] != 0 && maze[n - 1][n - 1] != 0) {

        findPath(0, 0, maze, n, result, currentPath);

    }

    if (result.size() == 0)

        System.out.println(-1);

    else

        for (String path : result)


            System.out.print(path + " ");

        System.out.println();

    }

}

```



```
44     int n = maze.length;
45     ArrayList<String> result = new ArrayList<>();
46     StringBuilder currentPath = new StringBuilder();
47
48     if (maze[0][0] != 0 && maze[n - 1][n - 1] != 0) {
49         findPath(0, 0, maze, n, result, currentPath);
50     }
51
52     if (result.size() == 0)
53         System.out.println(-1);
54     else
55         for (String path : result)
56             System.out.print(path + " ");
57     System.out.println();
58 }
59 }
60
```

Console x

<terminated> RatInMaze [Java Application] C:\Users\DELL\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_17.0.6.v20230204-1729\j...  
DDRRR DRDDR

### 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.

```
package day16and17;

public class NQueenProblem {

    final int N = 8;

    void printSolution(int board[][])

    {

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

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

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

                    System.out.print("Q ");

                else

                    System.out.print(". ");

            }

            System.out.println();

        }

    }

}
```

```

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;

}

boolean solveNQUtil(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 (solveNQUtil(board, col + 1) == true)

return true;

board[i][col] = 0;

}

}

return false;

}

```

```

boolean solveNQ()

{

int board[][] = { { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

if (solveNQUtil(board, 0) == false) {

System.out.print("Solution does not exist");

return false;

}

printSolution(board);

return true;

}

public static void main(String args[])

{

NQueenProblem Queen = new NQueenProblem();

Queen.solveNQ();

}

}

```



```
TravelingSa... Job.java Knapsack.java LongestCommo... KnightTour.java RatInMaze.java *NQueenProb... ×
1 package day16and17;
2
3 public class NQueenProblem {
4     final int N = 8;
5     void printSolution(int board[][])
6     {
7         for (int i = 0; i < N; i++) {
8             for (int j = 0; j < N; j++) {
9                 if (board[i][j] == 1)
10                    System.out.print("Q ");
11                else
12                    System.out.print(". ");
13            }
14            System.out.println();
15        }
16    }
17    boolean isSafe(int board[][], int row, int col)
18    {
19        // Check if there is a queen in the same row
20        for (int j = 0; j < N; j++) {
21            if (j != col && board[row][j] == 1)
22                return false;
23        }
24        // Check upper diagonal
25        int r = row, c = col;
26        while (r >= 0 && c >= 0) {
27            if (board[r][c] == 1 && (r, c) != (row, col))
28                return false;
29            r--; c--;
30        }
31        // Check lower diagonal
32        r = row, c = col;
33        while (r < N && c >= 0) {
34            if (board[r][c] == 1 && (r, c) != (row, col))
35                return false;
36            r++; c--;
37        }
38        return true;
39    }
40    // Driver code
41    public static void main(String[] args) {
42        NQueenProblem nqp = new NQueenProblem();
43        nqp.printSolution(nqp.isSafe(nqp.getBoard(), 0, 0));
44    }
45    private int[][] getBoard() {
46        int board[][] = new int[N][N];
47        // Pre-filled board with a solution
48        board[0][7] = 1;
49        board[1][2] = 1;
50        board[2][3] = 1;
51        board[3][1] = 1;
52        board[4][5] = 1;
53        board[5][4] = 1;
54        board[6][6] = 1;
55        board[7][0] = 1;
56        return board;
57    }
58 }
```

Console ×

<terminated> NQueenProblem [Java Application] C:\Users\DELL\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_17.0.6.v20230204-

```
Q . . . . .
. . . . Q .
. . . . . Q
. Q . . . .
. . . Q . .
. . . . Q .
. . Q . . .
```