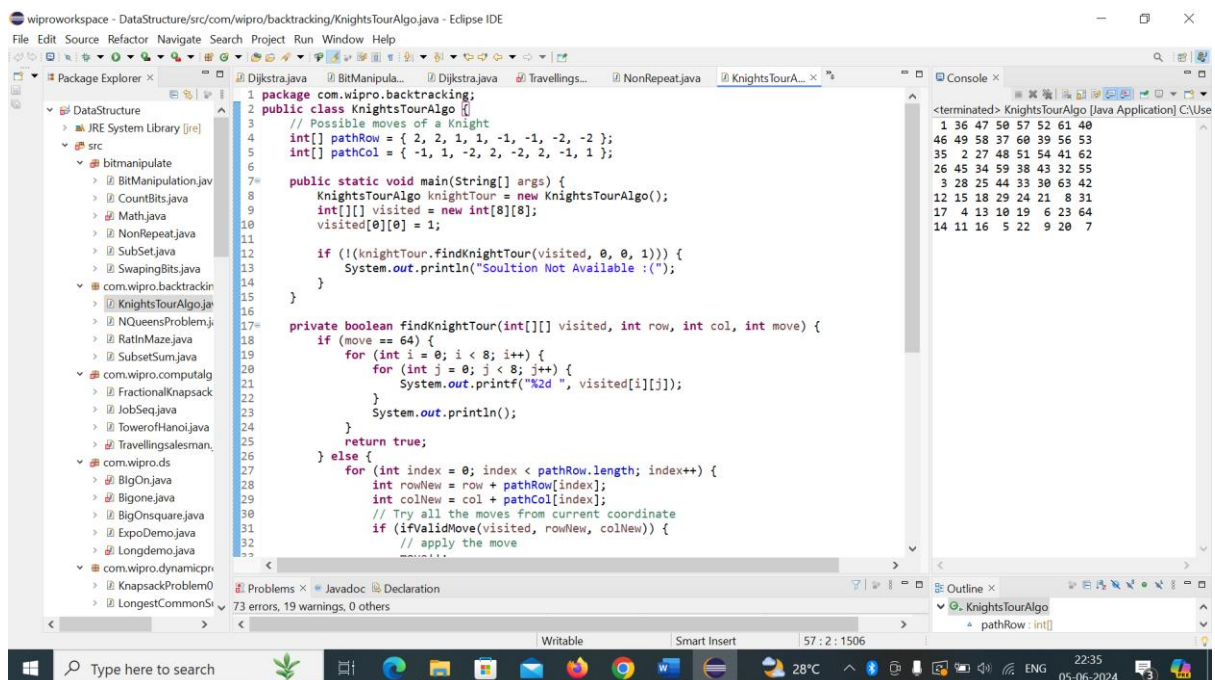


# Day 16 and 17

## 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.

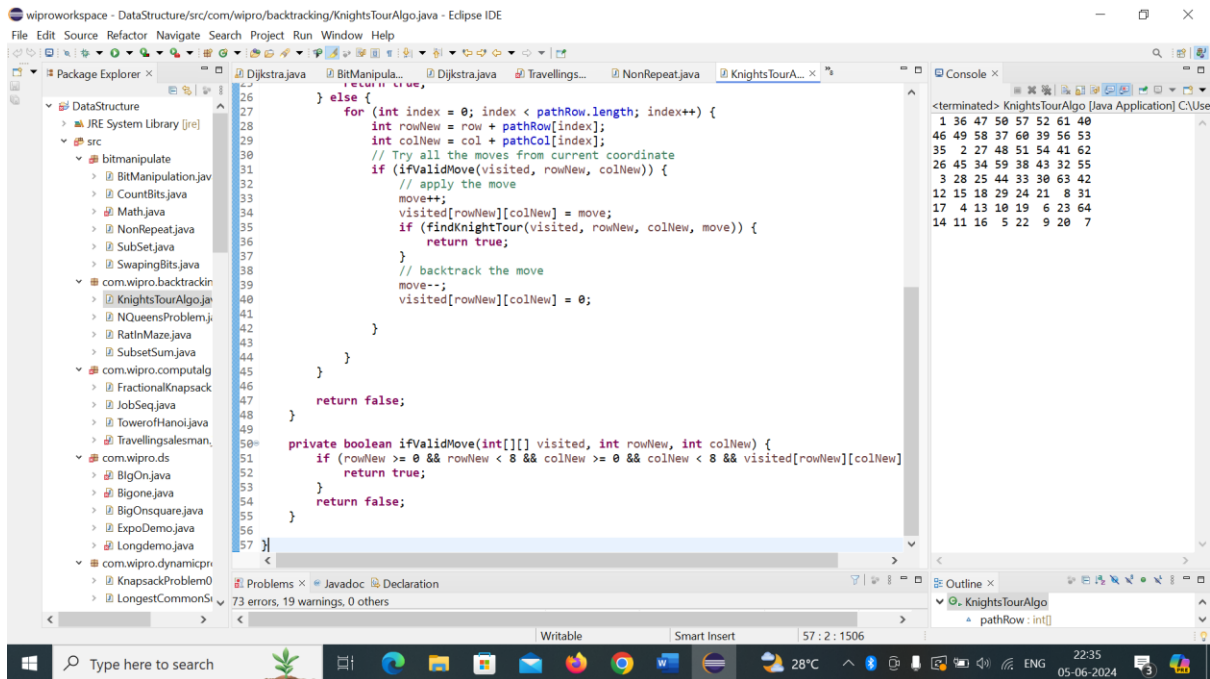


The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays a project named 'wiproworkspace' with a package 'com.wipro.backtracking' containing several Java files, including 'KnightsTourAlgo.java'. The main editor window shows the source code for 'KnightsTourAlgo.java'. The code defines a class 'KnightsTourAlgo' with a 'main' method and a 'findKnightTour' method. The 'main' method initializes an 8x8 'visited' array and calls 'findKnightTour'. The 'findKnightTour' method uses backtracking to find a path for the knight. It checks if the current move count is 64 (indicating a full tour) and returns true. Otherwise, it iterates through possible moves (defined in 'pathRow' and 'pathCol' arrays) and recursively calls 'findKnightTour' for each move. The console on the right shows the output of the program, which is a terminated Java application. The status bar at the bottom indicates '73 errors, 19 warnings, 0 others'.

```
1 package com.wipro.backtracking;
2 public class KnightsTourAlgo {
3     // Possible moves of a Knight
4     int[] pathRow = { 2, 2, 1, 1, -1, -1, -2, -2 };
5     int[] pathCol = { -1, 1, -2, 2, -2, 2, -1, 1 };
6
7     public static void main(String[] args) {
8         KnightsTourAlgo knightTour = new KnightsTourAlgo();
9         int[][] visited = new int[8][8];
10        visited[0][0] = 1;
11
12        if (!knightTour.findKnightTour(visited, 0, 0, 1)) {
13            System.out.println("Soulution Not Available :(");
14        }
15    }
16
17    private boolean findKnightTour(int[][] visited, int row, int col, int move) {
18        if (move == 64) {
19            for (int i = 0; i < 8; i++) {
20                for (int j = 0; j < 8; j++) {
21                    System.out.printf("%2d ", visited[i][j]);
22                }
23                System.out.println();
24            }
25            return true;
26        } else {
27            for (int index = 0; index < pathRow.length; index++) {
28                int rowNew = row + pathRow[index];
29                int colNew = col + pathCol[index];
30                // Try all the moves from current coordinate
31                if (isValidMove(visited, rowNew, colNew)) {
32                    // apply the move
33                    visited[rowNew][colNew] = move;
34                    if (findKnightTour(visited, rowNew, colNew, move + 1)) {
35                        return true;
36                    }
37                    visited[rowNew][colNew] = 0;
38                }
39            }
40            return false;
41        }
42    }
43
44    private boolean isValidMove(int[][] visited, int row, int col) {
45        return row >= 0 & row < 8 & col >= 0 & col < 8 & visited[row][col] == 0;
46    }
47 }
```

Console Output:

```
<terminated> KnightsTourAlgo [Java Application] C:\Use
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
```



```
package com.wipro.backtracking;

import java.util.*;

public class KnightsTourAlgo {
    private int row, col;
    private int pathRow, pathCol;
    private boolean[][] visited;

    public KnightsTourAlgo(int row, int col) {
        this.row = row;
        this.col = col;
        this.pathRow = 0;
        this.pathCol = 0;
        this.visited = new boolean[row][col];
    }

    public boolean findKnightTour(int row, int col) {
        return findKnightTour(visited, row, col, 0);
    }

    private boolean findKnightTour(boolean[][] visited, int row, int col, int move) {
        if (row >= 0 && row < 8 && col >= 0 && col < 8 && !visited[row][col]) {
            visited[row][col] = true;
            pathRow = row;
            pathCol = col;
            move++;
            if (move == 64) {
                return true;
            }
            if (findKnightTour(visited, row + 1, col, move)) {
                return true;
            }
            if (findKnightTour(visited, row - 1, col, move)) {
                return true;
            }
            if (findKnightTour(visited, row, col + 1, move)) {
                return true;
            }
            if (findKnightTour(visited, row, col - 1, move)) {
                return true;
            }
            // backtrack the move
            visited[row][col] = false;
            return false;
        }
        return false;
    }

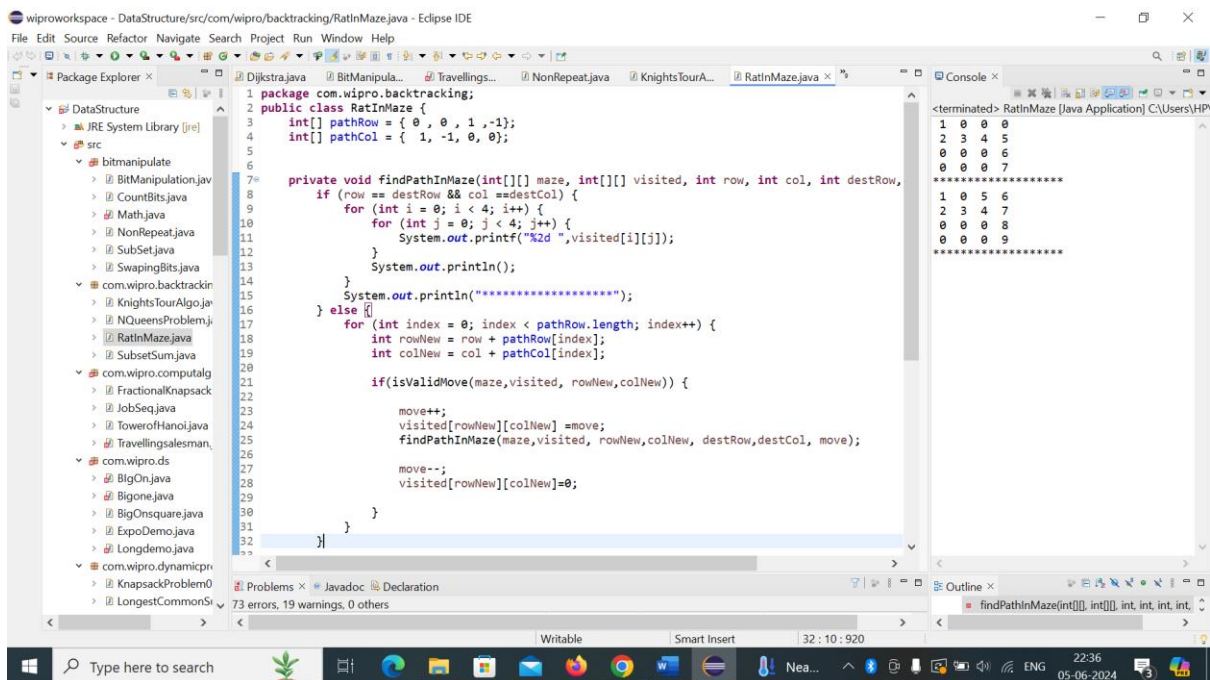
    private boolean isValidMove(int[][] visited, int rowNew, int colNew) {
        if (rowNew >= 0 && rowNew < 8 && colNew >= 0 && colNew < 8 && !visited[rowNew][colNew]) {
            return true;
        }
        return false;
    }
}
```

Console Output:

```
<terminated> KnightsTourAlgo [Java Application] C:\Users\HP\
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.



```
package com.wipro.backtracking;

import java.util.*;

public class RatInMaze {
    private int row, col;
    private int pathRow, pathCol;
    private boolean[][] visited;

    public RatInMaze(int row, int col) {
        this.row = row;
        this.col = col;
        this.pathRow = 0;
        this.pathCol = 0;
        this.visited = new boolean[row][col];
    }

    public boolean findPathInMaze(int row, int col) {
        return findPathInMaze(visited, row, col, 0);
    }

    private boolean findPathInMaze(boolean[][] visited, int row, int col, int move) {
        if (row >= 0 && row < 6 && col >= 0 && col < 6 && !visited[row][col]) {
            visited[row][col] = true;
            pathRow = row;
            pathCol = col;
            move++;
            if (move == 25) {
                return true;
            }
            if (findPathInMaze(visited, row + 1, col, move)) {
                return true;
            }
            if (findPathInMaze(visited, row - 1, col, move)) {
                return true;
            }
            if (findPathInMaze(visited, row, col + 1, move)) {
                return true;
            }
            if (findPathInMaze(visited, row, col - 1, move)) {
                return true;
            }
            // backtrack the move
            visited[row][col] = false;
            return false;
        }
        return false;
    }

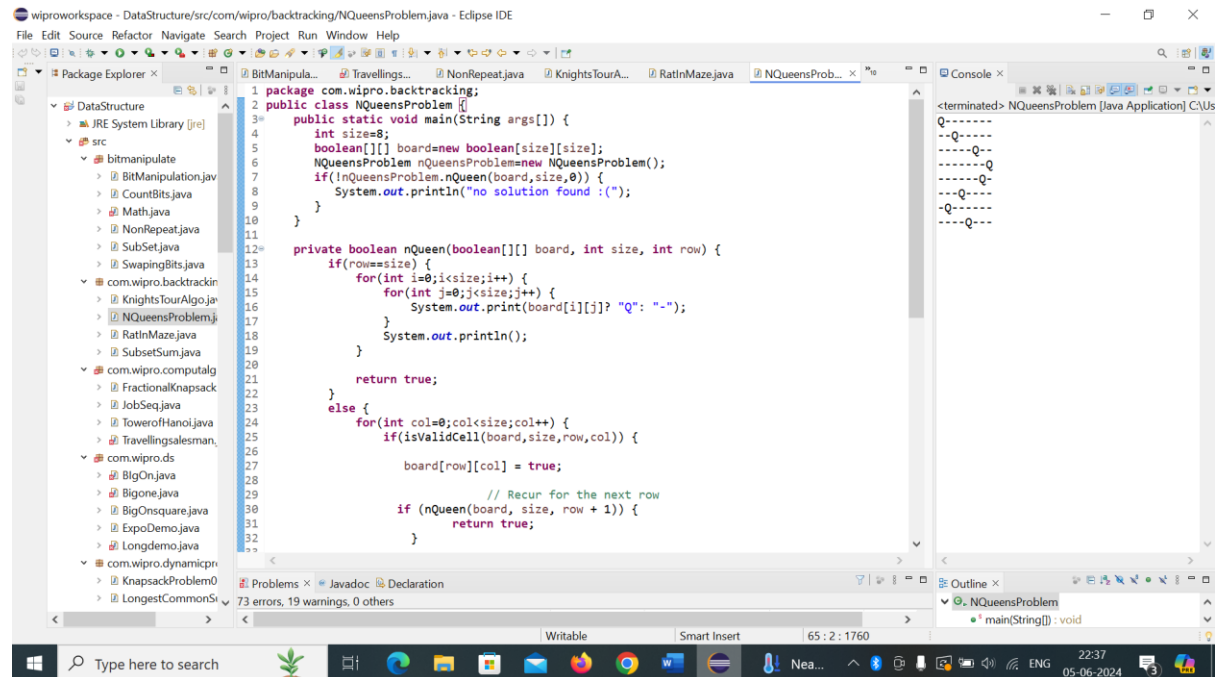
    private boolean isValidMove(int[][] maze, int rowNew, int colNew) {
        if (rowNew >= 0 && rowNew < 6 && colNew >= 0 && colNew < 6 && maze[rowNew][colNew] == 1 && !visited[rowNew][colNew]) {
            return true;
        }
        return false;
    }
}
```

Console Output:

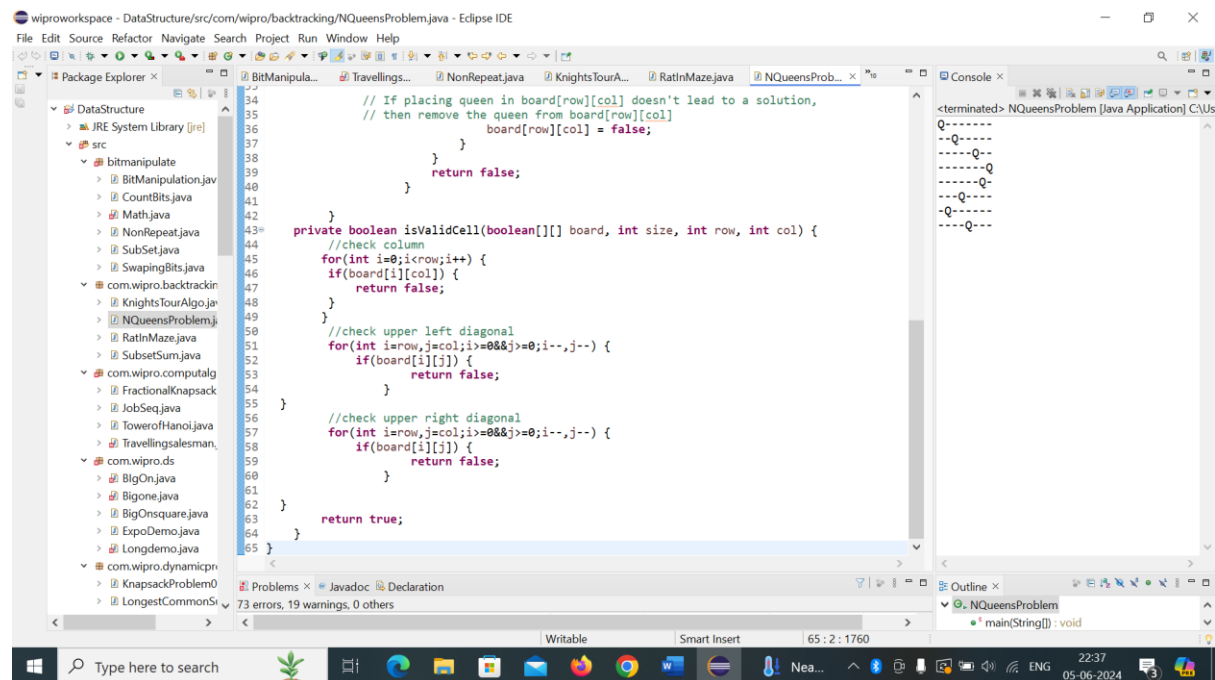
```
<terminated> RatInMaze [Java Application] C:\Users\HP\
1 0 0 0
2 3 4 5
0 0 0 6
0 0 0 7
*****
1 0 5 6
2 3 4 7
0 0 0 8
0 0 0 9
*****
```

### 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.



```
1 package com.wipro.backtracking;
2 public class NQueensProblem {
3     public static void main(String args[]) {
4         int size=8;
5         boolean[][] board=new boolean[size][size];
6         NQueensProblem nQueensProblem=new NQueensProblem();
7         if(!nQueensProblem.nQueen(board,size,0)) {
8             System.out.println("no solution found :(");
9         }
10    }
11
12    private boolean nQueen(boolean[][] board, int size, int row) {
13        if(row==size) {
14            for(int i=0;i<size;i++) {
15                for(int j=0;j<size;j++) {
16                    System.out.print(board[i][j]? "Q": "-");
17                }
18                System.out.println();
19            }
20            return true;
21        }
22        else {
23            for(int col=0;col<size;col++) {
24                if(isValidCell(board,size,row,col)) {
25                    board[row][col] = true;
26
27                    // Recur for the next row
28                    if (nQueen(board, size, row + 1)) {
29                        return true;
30                    }
31                }
32            }
33        }
34    }
35
36    // If placing queen in board[row][col] doesn't lead to a solution,
37    // then remove the queen from board[row][col]
38    board[row][col] = false;
39    }
40    }
41    }
42    }
43    }
44    }
45    }
46    }
47    }
48    }
49    }
50    }
51    }
52    }
53    }
54    }
55    }
56    }
57    }
58    }
59    }
60    }
61    }
62    }
63    }
64    }
65    }
```



```
34 // If placing queen in board[row][col] doesn't lead to a solution,
35 // then remove the queen from board[row][col]
36 board[row][col] = false;
37 }
38 }
39 }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
```