Assignment - 18

Name - Md. Firoze Baba firozebaba68@gmail.com

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.

Program:

```
public class KnightsTour {
     static int n=8;
     public static void main(String[] args) {
           solveKnightTour();
     private static boolean solveKnightTour() {
           int[][] board=new int[n][n];
           for (int i=0; i < n; i++)
                 for (int j=0; j < n; j++)
                      board[i][j]=-1;
           int[] imove= \{2,1,-1,-2,-2,-1,1,2\};
           int[] jmove= \{1, 2, 2, 1, -1, -2, -2, -1\};
           int count=1;
           board[0][0]=0;
           if(!solveKnightTour(board, 0, 0, count, imove, jmove)) {
                 System.out.println("solution does not exist");
                 return false;
           }
           else {
                 System.out.println("Solution found");
                 printSolution(board);
                 return true;
           }
     private static void printSolution(int[][] board) {
           for(int i=0;i<n;i++) {
                 for (int j=0; j < n; j++)
                       System.out.print(board[i][j]+" ");
           System.out.println();
     private static boolean solveKnightTour(int[][] board, int i,
                        int j, int count, int[] imove, int[] jmove)
           int nexti,nextj;
           if (count==n*n)
                 return true;
           for (int k=0; k<8; k++) {
```

```
nexti=i+imove[k];
                nextj=j+jmove[k];
                if(isValid(nexti,nextj,board)) {
                      board[nexti][nextj]=count;
                      if (solveKnightTour(board, nexti, nextj,
                            count+1, imove, jmove))
                           return true;
                      else
                           board[nexti] [nextj] = -1;
                }
           return false;
     private static boolean isValid(int i, int j, int[][] board) {
           return (i>=0 && j>=0 && i<n && j<n && board[i][j]==-1);
     }
}
Output:
Solution found:
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.

Program:

```
{ 1, 1, 1, 0, 0, 1 } };
           solveMaze(maze);
     private static boolean solveMaze(int[][] maze) {
           int[][] solve = new int[n][n];
           if (!solveMazeUtil(maze, 0, 0, solve)) {
                System.out.println("Solution does not exist");
                return false;
           } else {
                System.out.println("Solution found: ");
                System.out.println();
                printSolution(solve);
                return true;
           }
     }
     private static void printSolution(int[][] solve) {
           for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++)
                      System.out.print(solve[i][j] + " ");
                System.out.println();
           }
     }
     private static boolean solveMazeUtil(int[][] maze, int i,
                                            int j, int[][] solve) {
           if (i == n - 1 \&\& j == n - 1) {
                solve[i][j] = 1;
                return true;
           if (isvalid(i, j, maze)) {
                solve[i][j] = 1;
                if (solveMazeUtil(maze, i + 1, j, solve))
                      return true;
                if (solveMazeUtil(maze, i, j + 1, solve))
                      return true;
                solve[i][j] = 0;
                return false;
           return false;
     }
     private static boolean isvalid(int i, int j, int[][] maze) {
           return (i >= 0 && j >= 0 && i < n && j < n &&
                     maze[i][j] == 1);
Output:
Solution found:
1 0 0 0 0 0
1 1 1 1 0 0
0 0 0 1 0 0
```

{ 1, 1, 0, 1, 1, 1 },

```
0 0 0 1 0 0
0 0 0 1 1 1
0 0 0 0 0 1
```

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.

Program:

```
public class NQueen {
     static int N=8;
     public static void main(String[] args) {
           solveNQueen();
     private static boolean solveNQueen() {
           int[][] board = new int[N][N];
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                board[i][j] = 0;
        if (!solveNQueensUtil(board, 0)) {
            System.out.println("Solution does not exist");
            return false;
        System.out.println("Solution found: ");
        System.out.println();
        printSolution(board);
        return true;
     }
     private static void printSolution(int[][] board) {
           for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++)
                System.out.print(board[i][j] + " ");
            System.out.println();
        }
     }
     private static boolean solveNQueensUtil(int[][] board, int k)
{
           if (k >= N)
            return true;
           for (int i = 0; i < N; i++) {
                if (isSafe(board, i, k)) {
                      board[i][k] = 1;
                      if (solveNQueensUtil(board, k + 1))
                    return true;
                     board[i][k] = 0;
                }
```

```
return false;
     }
     private static boolean isSafe(int[][] board, int r, int k) {
           int i, j;
        for (i = 0; i < k; i++)
               if (board[r][i] == 1)
                return false;
        for (i = r, j = k; i \ge 0 \&\& j \ge 0; i--, j--)
            if (board[i][j] == 1)
                return false;
        for (i = r, j = k; j >= 0 && i < N; i++, j--)
            if (board[i][j] == 1)
                return false;
          return true;
     }
Output:
Solution found:
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
```