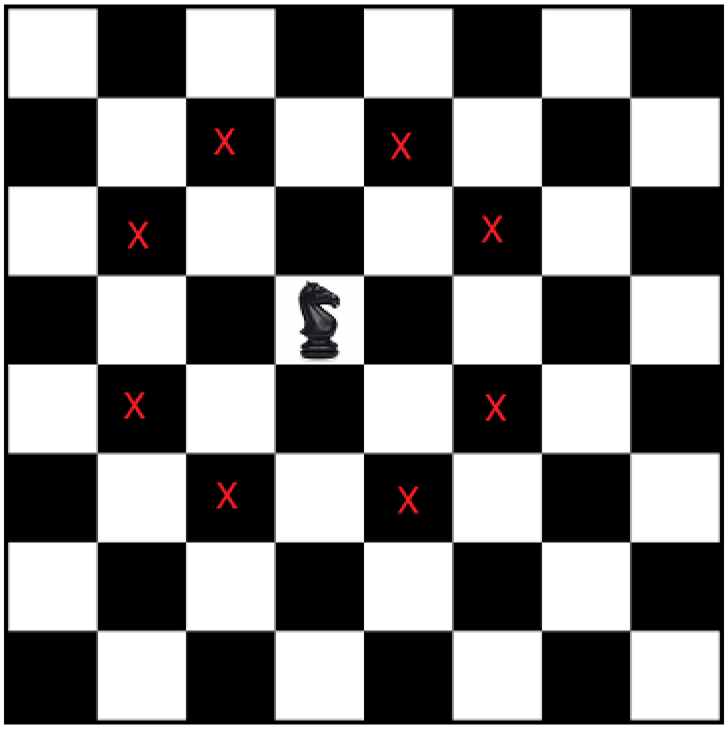
Knight’s Tour

A knight's tour is a sequence of moves by a knight such that it visits, each square of the chessboard exactly once.

Note : The knight can start the tour from any square.



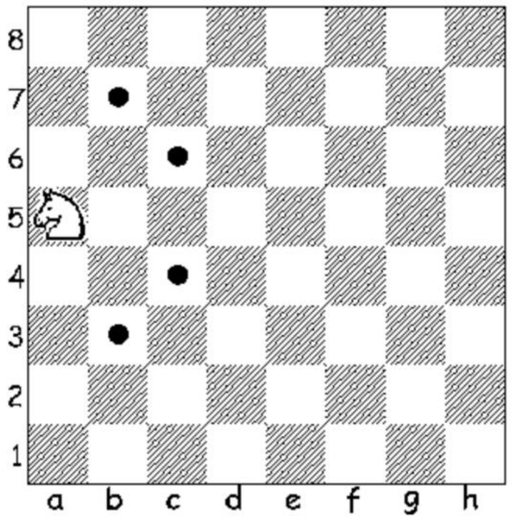


Figure : Knight can only move to one of the 4 possible squares (denoted with black dots)

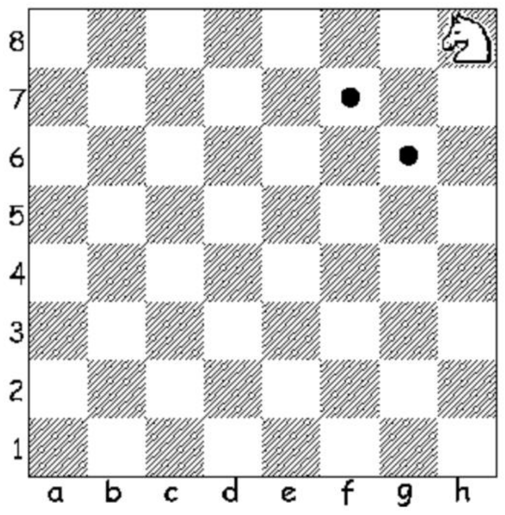
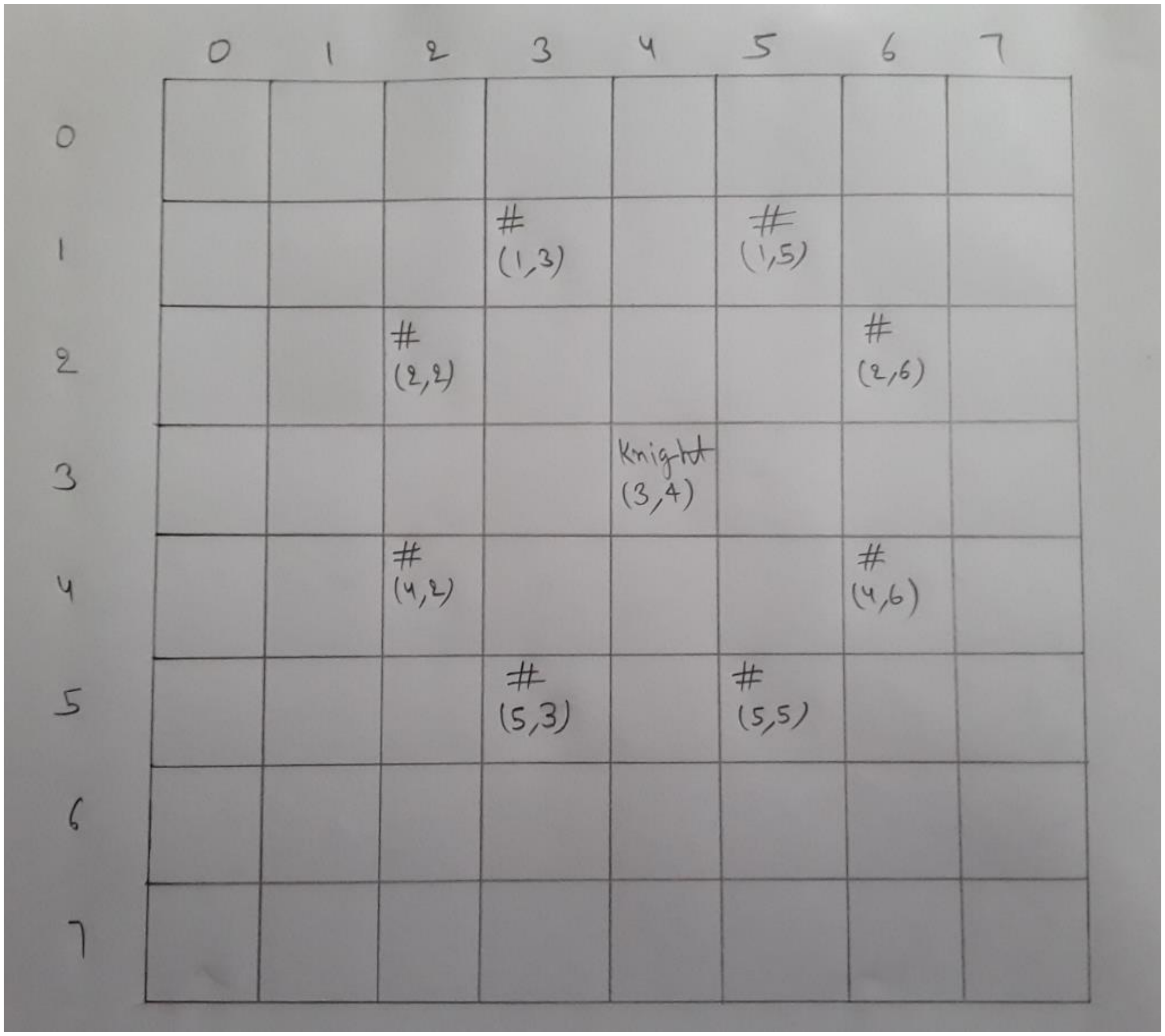
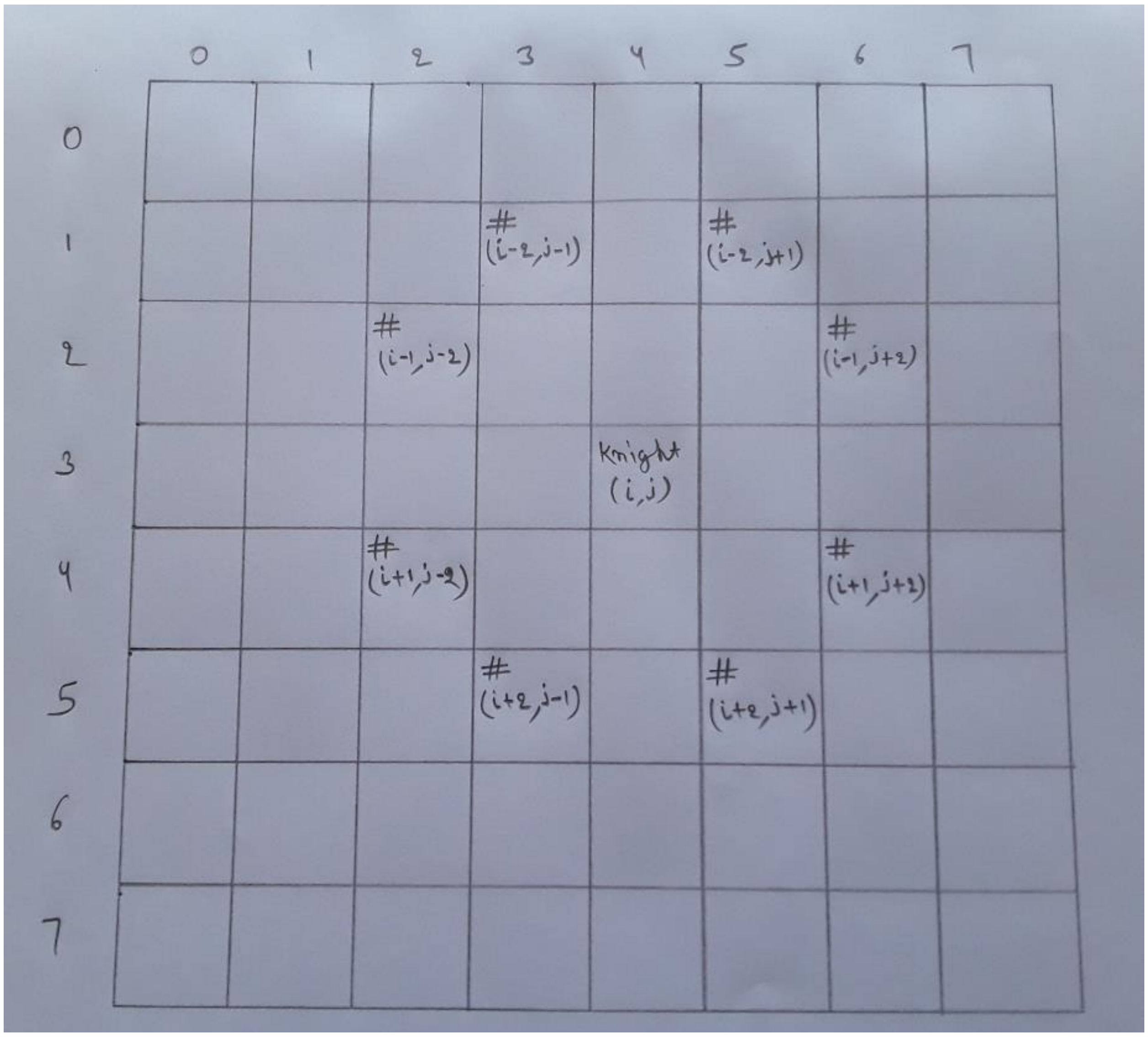


Figure : Knight can only move to one of the 2 possible squares (denoted with black dots)



	0	1	2	3	4	5	6	7
0								
1				# (1,3)		# (1,5)		
2			# (2,2)				# (2,6)	
3					Knight (3,4)			
4			# (4,2)				# (4,6)	
5				# (5,3)		# (5,5)		
6								
7								



Approach :

1. Start the knight tour from the top left corner, cell (0,0) , try knight's 8 possible moves one by one, if a move is valid and the knight haven't already visited that square, then make the next move to that square.
2. Keep visiting the unvisited squares until knight visits each square of the chessboard.

Implementation

```
public class App {

	public static void main(String[] args) {
		final int  chess_board_size = 8;
		KnightTour knightTour = new KnightTour(chess_board_size);
		knightTour.solveKnightTourProblem();
	}
}


class KnightTour {
	private int BOARD_SIZE;
	private int[][] visited;
	private int[] xMoves = { 2, 1, -1, -2, -2, -1, 1, 2 };
	private int[] yMoves = { 1, 2, 2, 1, -1, -2, -2, -1 };

	public KnightTour(int chessBoardSize) {
		this.BOARD_SIZE = chessBoardSize;
		this.visited = new int[BOARD_SIZE][BOARD_SIZE];
		this.initializeBoard();
	}

	private void initializeBoard() {
		for (int i = 0; i < BOARD_SIZE; i++)
			for (int j = 0; j < BOARD_SIZE; j++)
				this.visited[i][j] = Integer.MIN_VALUE;
	}


	public void printSolution() {
		for (int i = 0; i < BOARD_SIZE; i++) {
			for (int j = 0; j < BOARD_SIZE; j++) {
				System.out.print(visited[i][j] + " ");
			}
			System.out.println();
		}
	}

	…
}
```

```
public void solveKnightTourProblem() {
            visited[0][0] = 0;
            // start knight tour from top left corner square (0, 0)
            if( solveProblem(1, 0, 0)) {
                  printSolution();
            } else {
                  System.out.println("No feasible solution found...");
            }
}


public boolean solveProblem(int moveCount, int x, int y) {
            // Base Case : We were able to move to each square exactly once
            if (moveCount == BOARD_SIZE * BOARD_SIZE) {
                  return true;
            }

            for (int i = 0; i < xMoves.length; ++i) {
                  int nextX = x + xMoves[i];
                  int nextY = y + yMoves[i];

                  // check if new position is a valid and not visited yet
                  if ( isValidMove(nextX, nextY) &&
                       visited[nextX][nextY] == Integer.MIN_VALUE) {

                        visited[nextX][nextY] = moveCount;
                        if ( solveProblem(moveCount + 1, nextX, nextY) ) {
                              return true;
                        }

                  // BACKTRACK !!!
                        visited[nextX][nextY] = Integer.MIN_VALUE;
                  }
            }
      return false;
}


public boolean isValidMove(int x, int y) {
            if (x < 0 || x >= BOARD_SIZE || y < 0 || y >= BOARD_SIZE) {
                  return false;
            } else {
                  return true;
            }
}
```