Name: Vinayak Madan Shete

Roll No.: TYCOC303

Div: C Batch: C4

Course Name: Design and Analysis of Algorithms Laboratory

Course Code: BCE5412

==========

Assignment 05:Implementing Knights Tour Problem.

==========

Input:

```
# Python3 program to solve Knight Tour problem using Backtracking
# Chessboard Size
n = 8
def isSafe(x, y, board):
      '''
            A utility function to check if i,j are valid indexes
            for N*N chessboard
      '''
      if(x >= 0 and y >= 0 and x < n and y < n and board[x][y] == -1):
            return True
      return False
def printSolution(n, board):
      '''
            A utility function to print Chessboard matrix
      '''
      for i in range(n):
            for j in range(n):
                  print(board[i][j], end=' ')
            print()
```

```
def solveKT(n):
	'''
		This function solves the Knight Tour problem using
		Backtracking. This function mainly uses solveKTUtil()
		to solve the problem. It returns false if no complete
		tour is possible, otherwise return true and prints the
		tour.
		Please note that there may be more than one solutions,
		this function prints one of the feasible solutions.
	'''
	# Initialization of Board matrix
	board = [[-1 for i in range(n)]for i in range(n)]
	# move_x and move_y define next move of Knight.
	# move_x is for next value of x coordinate
	# move_y is for next value of y coordinate
	move_x = [2, 1, -1, -2, -2, -1, 1, 2]
	move_y = [1, 2, 2, 1, -1, -2, -2, -1]

	# Since the Knight is initially at the first block
	board[0][0] = 0

	# Step counter for knight's position
	pos = 1

	# Checking if solution exists or not
	if(not solveKTUtil(n, board, 0, 0, move_x, move_y, pos)):
		print("Solution does not exist")
	else:
		printSolution(n, board)


def solveKTUtil(n, board, curr_x, curr_y, move_x, move_y, pos):
	'''
		A recursive utility function to solve Knight Tour
```

```
            problem
      '''

      if(pos == n**2):
            return True

      # Try all next moves from the current coordinate x, y
      for i in range(8):
            new_x = curr_x + move_x[i]
            new_y = curr_y + move_y[i]
            if(isSafe(new_x, new_y, board)):
                  board[new_x][new_y] = pos
                  if(solveKTUtil(n, board, new_x, new_y, move_x, move_y,
pos+1)):
                        return True

                  # Backtracking
                  board[new_x][new_y] = -1
      return False
# Driver Code
if __name__ == "__main__":

      # Function Call
      solveKT(n)
```

Output:

```
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
```

==========