K. J. Somaiya Institute of Engineering and Information Technology, Sion, Mumbai

Accredited 'A' Grade by NAAC with 3.21 CGPA
3 Programs Accredited by National Board of Accreditation
Permanently Affiliated to University of Mumbai,
Best College Award by University of Mumbai (Urban Region), ISTE (MH), and CSI (Mumbai)
UGC Recognized Institute under Section 2(f) and 12(B) of the UGC Act, 1956

DEPARTMENT OF INFORMATION TECHNOLOGY

Academic Year (2020-2021) Odd Semester VII

Course: Artificial Intelligence

Experiment No. 05

Aim: Implementation of Constraint Satisfaction Problem(CSP).

5.1 Implement the Knight Tour Problem.

Objectives:

To make students understand various AI methods like searching and game playing and how to apply them to solve real applications

Outcomes:

Develop intelligent algorithms for constraint satisfaction problems and also design intelligent systems for Game Playing

Theory:

5.1 Implement the Knight Tour Problem.

a) Introduction to Knight Tour problem.
A knight's tour is a sequence of moves of a knight on a chessboard such that the knight visits every square exactly once. If the knight ends on a square that is one knight's move from the beginning square, the tour is closed; otherwise, it is open.
Problem Statement:
Given a N*N board with the Knight placed on the first block of an empty board. Moving according to the rules of chess, knights must visit each square exactly once. Print the order of each cell in which they are visited.

b) Backtracking Algorithm/Pseudocode.

Backtracking is an algorithmic paradigm that tries different solutions until it finds a solution that works. Problems which are typically solved using backtracking technique have the following property in common. These problems can only be solved by trying every possible configuration and each configuration is tried only once. A Naive solution for these problems is to try all configurations and output a configuration that follows given problem constraints. Backtracking works in an incremental way and is an optimization over the Naive solution where all possible configurations are generated and tried.

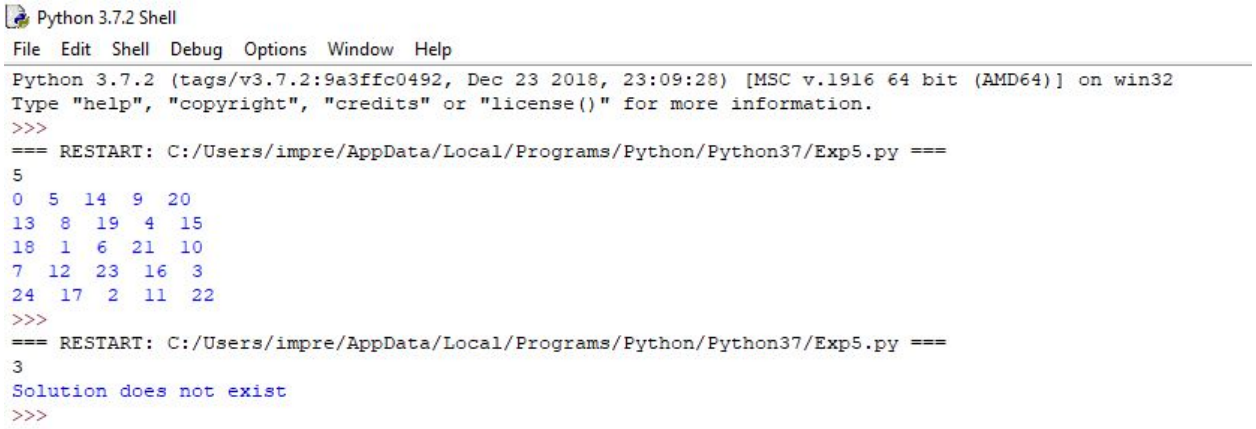
CODE:

```
n = int(input())
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 print 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:



Conclusion:

From this experiment, we understood the concept of Constraint satisfaction problem (CSP) and implemented the Knight Tour problem successfully.