Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Mini Project Report

COMP 314

(For partial fulfillment of 3rd Year/ 2nd Semester in Computer Engineering)

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submitted by:

Neha Malla

C.E.

Roll No. 27

Task:

Write a program to solve the Knight's tour problem using backtracking. Start with a chessboard of size 3x3, then increase the size and find the size of the largest chessboard your computer can solve. Record the time taken to solve the problem for each size of the board.

Problem Statement:

A Knight's tour problem requires finding a sequence of moves of a knight on a chessboard such that the knight visits every square exactly once.

Algorithm Used:

For solving the Knight's tour problem, a backtracking algorithm is used.

Firstly, a square board(2D array) of required size is initialized with all its elements to "-1". The value "-1" represents an unvisited square in the board.

A Knight is placed on the [0, 0] index of the array. Then its next moves are checked to be safe or not, i.e. safe if the square is unvisited, or otherwise.

If it is safe then that square is marked visited by the 'count' which keeps track of the number of visited squares. Then the function is recursively called for the next move of the currently marked move.

> If each move and its next move is safe, the recursive call continues and thus the Knight travels till the count reaches the total number of squares in the board.
>
> Else if the next move is unsafe i.e. all the 8 possible squares are visited already then, it backtracks the last move, meaning, the last visited square is unmarked(made unvisited) and instead of that choice of square, another safe one from 8 possible moves is considered and carried on.

After this process of selection from 8 possible squares, recursion and backtracking, if there isn't any solution obtained then a message is displayed for it.

Else the solution board is printed out in the terminal.

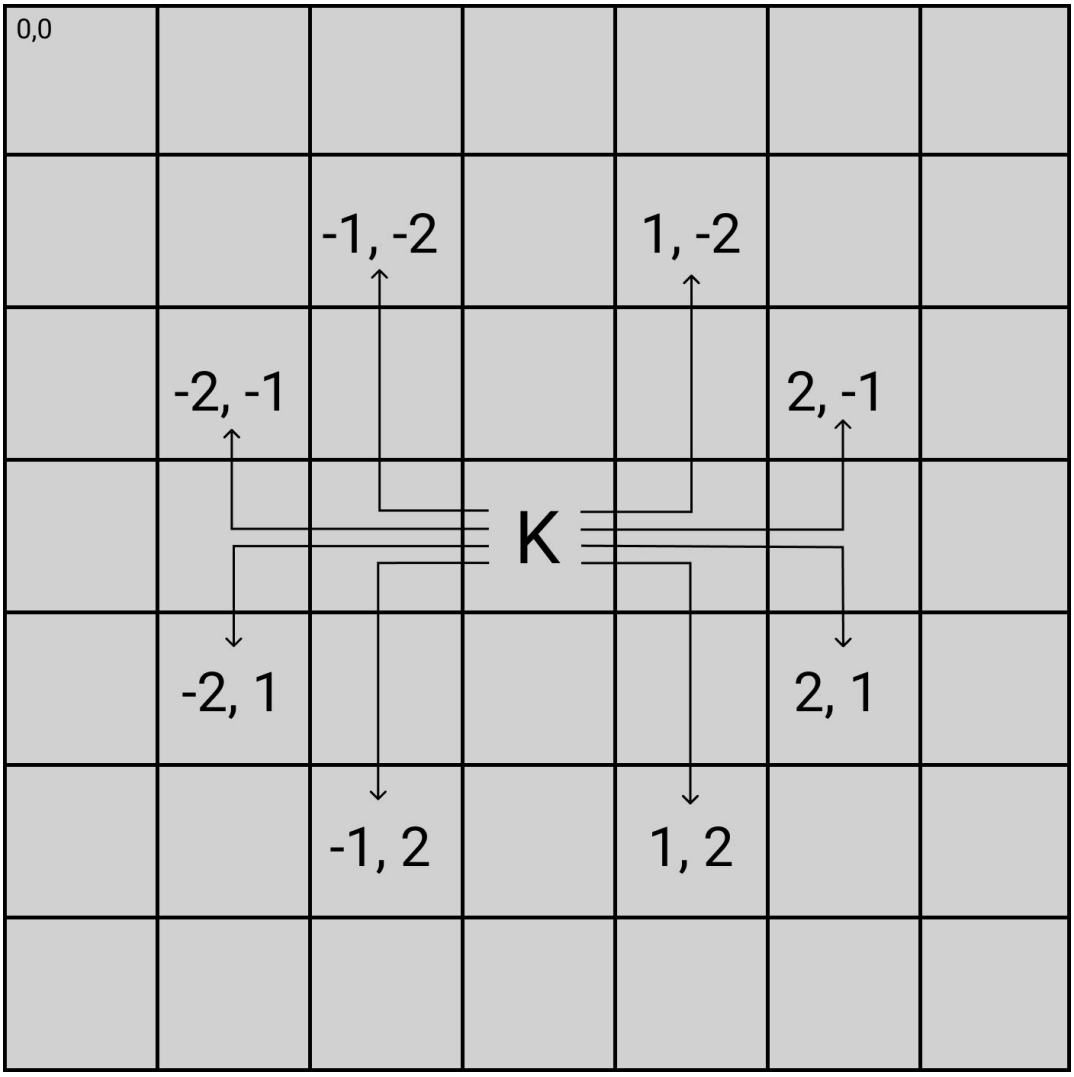

Figure: 8 possible moves relative to the current position of the Knight

Pseudocode:

Three functions are made, starting from Initialise(n), where n*n is the size of the board.

Initialise(n):

```
board = [[-1]*n]*n // 2D array of n*n filled with -1

// 8 possible moves
moves = [[2, 1], [2, -1], [-2, 1], [-2, -1], [-1, 2], [1, 2], [-1, -2], [1, -2]]

board[0][0] = 0 // first placement
count = 1 // variable for next position
```

```
    if KnightTour(n, board, moves, [0,0], count): //solvable
        for row in board:
            print(row)
    else:
        print("No solution exists for n = ", n)


KnightTour(n, board, moves, currentMove, count):
    if count == n*n: // each square is visited
        return True

    /*loop for trying every possible move for safe placement or backtrack to previous safe
    placement*/
    for i in range(8):
        // location for next move
        nextMove = [currentMove[0]+moves[i][0], currentMove[1]+moves[i][1]]

        if(SafeMove(nextMove, n, board)): // next move is safe
            board[nextMove[0]][nextMove[1]] = count // placement

            // recursive call for next move
            if (KnightTour(n, board, moves, nextMove, count + 1)):
                return True // successful at safe placement

            // no safe move, so backtrack
            board[nextMove[0]][nextMove[1]] = -1

    return False //unable to find safe moves


SafeMove(nextMove, n, board):
    // x and y displacements
    x = nextMove[0]
    y = nextMove[1]

    if x >= 0 and x < n and y >= 0 and y < n: // move is within bound
        if board[x][y] == -1: // square is unvisited
            return True // safe to place
    return False
```

Input:

For Board size (n*n), input is given from n = 3 to 7.

Output:

```
C:\Users\neha\Algorhythm>python mini.py
No solution exists for n =  3
done with  3
No solution exists for n =  4
done with  4
[0, 5, 14, 9, 20]
[13, 8, 19, 4, 15]
[18, 1, 6, 21, 10]
[7, 12, 23, 16, 3]
[24, 17, 2, 11, 22]
done with  5
[0, 11, 20, 27, 6, 9]
[21, 28, 7, 10, 19, 26]
[12, 1, 22, 3, 8, 5]
[29, 34, 31, 16, 25, 18]
[32, 13, 2, 23, 4, 15]
[35, 30, 33, 14, 17, 24]
done with  6
[0, 27, 38, 35, 24, 29, 8]
[37, 40, 25, 28, 9, 34, 23]
[26, 1, 36, 39, 22, 7, 30]
[41, 18, 43, 16, 31, 10, 33]
[44, 47, 2, 21, 4, 13, 6]
[19, 42, 17, 46, 15, 32, 11]
[48, 45, 20, 3, 12, 5, 14]
done with  7
[3, 4, 5, 6, 7]
[0.0003993511199951172, 0.010975122451782227, 0.3351023197174072, 11.222206354141235, 0.37793803
```

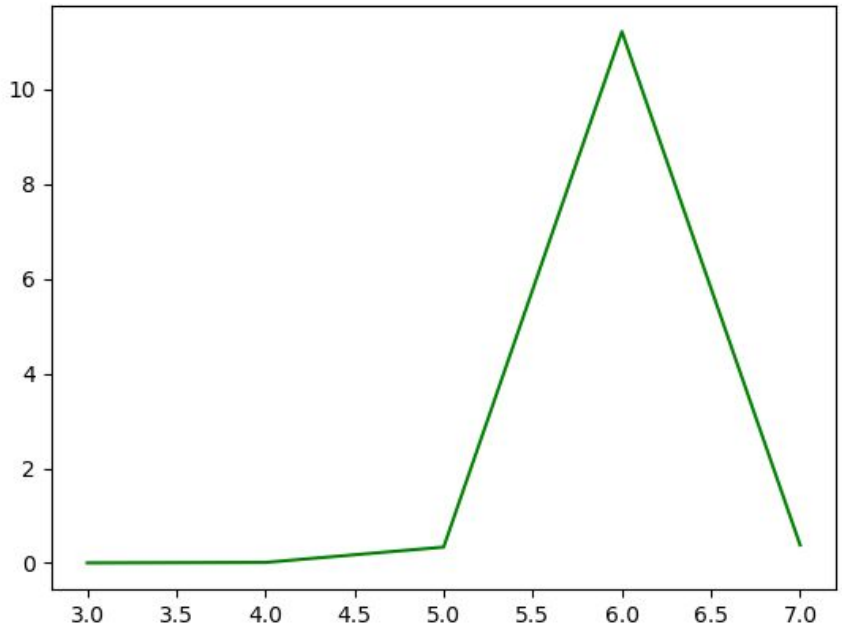

Figure: Solution and graph for time taken

Summarizing the output here in the table below:

Board Size	Time Taken	Output
3*3	0.39935 ms	No solution exists for n = 3
4*4	10.97512 ms	No solution exists for n = 4
5*5	0.33510 s	[0, 5, 14, 9, 20] [13, 8, 19, 4, 15] [18, 1, 6, 21, 10] [7, 12, 23, 16, 3] [24, 17, 2, 11, 22]
6*6	11.22221 s	[0, 11, 20, 27, 6, 9] [21, 28, 7, 10, 19, 26] [12, 1, 22, 3, 8, 5] [29, 34, 31, 16, 25, 18] [32, 13, 2, 23, 4, 15] [35, 30, 33, 14, 17, 24]
7*7	0.37794 s	[0, 27, 38, 35, 24, 29, 8] [37, 40, 25, 28, 9, 34, 23] [26, 1, 36, 39, 22, 7, 30] [41, 18, 43, 16, 31, 10, 33] [44, 47, 2, 21, 4, 13, 6] [19, 42, 17, 46, 15, 32, 11] [48, 45, 20, 3, 12, 5, 14]

Conclusion:

Knight's Tour problem was solved by using backtracking. The maximum size of the board that could be solved was 7*7 on my computer. For some reason the computer took forever on 8*8 and higher order boards. The source code is submitted separately, along with this report.