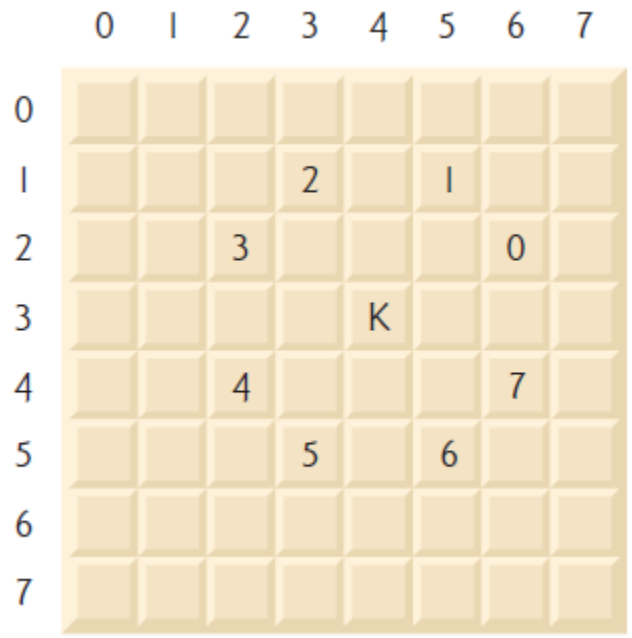
Knights Tour

November 24, 2021

1 Knight's Tour Problem

An interesting puzzler for chess buffs is the Knight's Tour problem, originally proposed by the mathematician Euler. Can the knight piece move around an empty chessboard and touch each of the 64 squares once and only once?

The knight makes only L-shaped moves (two spaces in one direction and one space in a perpendicular direction). Thus, as shown in the figure below, from a square near the middle of an empty chessboard, the knight (labeled K) can make eight different moves (numbered 0 through 7).



Now let's develop a script that will move the knight around a chessboard represented by an eight-by-eight two-dimensional array named board. Initialize each square to zero. We describe each of the eight possible moves in terms of its horizontal and vertical components. For example, a move of type 0, as shown in the preceding figure, consists of moving two squares horizontally to the right and one square vertically upward. A move of type 2 consists of moving one square horizontally to the left and two squares vertically upward. Horizontal moves to the left and vertical moves upward are indicated with negative numbers. The eight moves may be described by two one-dimensional arrays, horizontal and vertical, as follows:

[75]:
```
import numpy as np
import random

board = np.zeros((8,8), dtype = np.int8)
current_row = np.random.randint(0,7)
current_col = np.random.randint(0,7)
board[current_row, current_col] = 1


horizontal = [2, 1, -1, -2, -2, -1, 1, 2]
vertical = [-1, -2, -2, -1, 1, 2, 2, 1]
```

Develop a functhon to print the board

[76]:
```
def print_board(board):
    for row in board:
        #print("-"*24)
        for col in row:
            print(f" {col:02d} ",end="")
        print("\n")
```

[77]:
```
#print("initial board state")
print_board(board)
```

```
00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  01  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00
```

The following function moves the Knight around the board.

[78]:
```
def move(move_number,current_row,current_col):
    new_row = current_row + vertical[move_number]
    new_col = current_col + horizontal[move_number]

    return new_row, new_col
```

The following function checks whether or not a move is valid:

[79]:
```
def validMoves(board,current_row,current_col):
    valid_moves = []
    for move_number in range(8):
        new_row, new_col = move(move_number,current_row,current_col)
        #print(f"new position state:{board[new_row,new_col]}")
        #print(type(board[new_row,new_col]))
        if new_row >= 0 and new_row < 8 and new_col >= 0 and new_col < 8 and␣
 ↪board[new_row,new_col] == 0:
            valid_moves.append(move_number)
    return valid_moves
```

Lets test these functions

[80]:
```
print_board(board)
print("Current row and column:", current_row,current_col)
print("Valid moves from this position:",␣
 ↪validMoves(board,current_row,current_col))
```

```
00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  01  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

Current row and column: 4 4
Valid moves from this position: [0, 1, 2, 3, 4, 5, 6, 7]
```

[81]:
```
for i in range(2,65):
    # find all valid moves from the current position
    valid_moves = validMoves(board, current_row,current_col)
    if valid_moves == []:
        # exit if no valid moves are possible
        print(f"Tour not Complete: No moves possible after {i-1} moves")
        break
    else:
        selected_move = random.choice(valid_moves) # randomly select a move index
```

```
        # apply the selected move
        current_row,current_col = move(selected_move, current_row,current_col)
        # write the write the count on the board
        board[current_row,current_col]=i
    if i == 64:
        print(f"Tour Completed")
print("Random Choice:\n")
print_board(board)
```

```
Tour not Complete: No moves possible after 26 moves
Random Choice:

 00  26  00  00  13  00  21  00

 06  00  12  25  00  00  00  00

 11  24  07  14  00  22  00  20

 00  05  10  23  00  03  00  00

 00  08  15  04  01  00  19  00

 00  00  00  09  16  00  02  00

 00  00  00  00  00  00  00  18

 00  00  00  00  00  17  00  00
```

1.1 Accessiblity Heuristic

You may have observed that the outer squares are more troublesome than the squares nearer the center of the board. In fact, the most troublesome or inaccessible squares are the four corners.

Intuition may suggest that you should attempt to move the knight to the most troublesome squares first and leave open those that are easiest to get to so that when the board gets congested near the end of the tour, there will be a greater chance of success.

We could develop an "accessibility heuristic" by classifying each of the squares according to how accessible it is and always moving the knight (using the knight's L-shaped moves) to the most inaccessible square. We fill two dimensional array accessibility with numbers indicating from how many squares each particular square is accessible. On a blank chessboard, each of the 16 squares nearest the center is rated as 8, each corner square is rated as 2, and the other squares have accessibility numbers of 3, 4 or 6 as follows:

2	3	4	4	4	4	3	2
3	4	6	6	6	6	4	3
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
3	4	6	6	6	6	4	3
2	3	4	4	4	4	3	2

[82]:
```
def accessibility(board):
    #calculate accessibility of each cell
    heuristic = np.zeros((8,8), dtype = np.int8)
    for row in range(0,8):
        for col in range(0,8):
            #print(col)
            heuristic[row,col] = len(validMoves(board, row, col))
    return heuristic

def␣
 ↪valid_move_accessibity(accessiblity_boad,valid_moves,current_row,current_col):
    #calculate accessibility after execution of valid moves
    accessibility = []
    for m in valid_moves:
        new_row, new_col = move(m,current_row,current_col)
        accessibility.append(accessiblity_boad[new_row,new_col])
    return accessibility
```

Lets test these functions

[83]:
```
print("Current Board:")
print_board(board)
print("\n Accessiblity:")
print_board(accessibility(board))
```

```
Current Board:
 00  26  00  00  13  00  21  00

 06  00  12  25  00  00  00  00

 11  24  07  14  00  22  00  20

 00  05  10  23  00  03  00  00

 00  08  15  04  01  00  19  00
```

```
00  00  00  09  16  00  02  00

00  00  00  00  00  00  00  18

00  00  00  00  00  17  00  00


Accessiblity:
00  00  01  03  01  03  01  02

01  02  02  02  02  04  02  02

00  03  04  03  05  02  06  01

02  02  04  03  03  03  05  02

02  02  05  05  04  06  02  03

02  04  05  06  03  06  03  04

02  02  03  03  04  03  03  01

02  03  03  03  03  01  03  01
```

[84]:
```
test_board = np.zeros((8,8), dtype = np.int8)

print("Test Board:")
print_board(test_board)
print("Accessibility of Test Board:" )
print_board(accessibility(test_board))
```

```
Test Board:
00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00

00  00  00  00  00  00  00  00
```

```
 00  00  00  00  00  00  00  00

Accessibility of Test Board:
 02  03  04  04  04  04  03  02

 03  04  06  06  06  06  04  03

 04  06  08  08  08  08  06  04

 04  06  08  08  08  08  06  04

 04  06  08  08  08  08  06  04

 04  06  08  08  08  08  06  04

 03  04  06  06  06  06  04  03

 02  03  04  04  04  04  03  02
```

Lets solve the problem by using the accessiblity heuristic

[85]:
```
del board # delete the existing bard and initialize a new one

board = np.zeros((8,8), dtype = np.int8)
current_row = np.random.randint(0,7)
current_col = np.random.randint(0,7)
board[current_row, current_col] = 1

print_board(board)
```

```
 00  00  00  00  00  00  00  00

 00  00  00  00  00  00  00  00

 00  00  00  00  00  00  00  00

 00  00  00  00  00  00  00  00

 00  00  00  00  00  00  00  00

 00  00  00  00  00  00  00  00

 00  00  00  01  00  00  00  00

 00  00  00  00  00  00  00  00
```

[86]:
```
for i in range(2,65):
    valid_moves = validMoves(board, current_row,current_col)
    if valid_moves == []:
        print(f"Tour Incomplete: No moves possible after {i-1} moves")
        break
    else:
        h = accessibility(board)
        valid_h = valid_move_accessibity(h,valid_moves,current_row,current_col)
        selected_move = valid_moves[valid_h.index(min(valid_h))]
        current_row, current_col = move(selected_move, current_row,current_col)
        board[current_row,current_col]=i
    if i == 64:
        print(f"Tour Completed")

print("Min Acess:\n")
print_board(board)
```

```
Tour Completed
Min Acess:

 08  37  06  47  10  35  30  63

 05  46  09  36  31  64  11  34

 38  07  48  45  58  33  62  29

 43  04  57  32  61  50  55  12

 22  39  44  49  56  59  28  51

 03  42  21  60  25  54  13  16

 20  23  40  01  18  15  52  27

 41  02  19  24  53  26  17  14
```

[ ]:

[ ]: