# PBL Task 2

SIT215

Daniel, Amulya, Bella, Ramamdeep and Devesh

## Contents

## Task Problem

In this task you are to create a piece of software that can solve the Knight's Tour problem on an $NN \times NN$ board. You may select any $NN$, and ideally demonstrate a solution for different values of $NN$. A good starting point would be the standard $8 \times 8$ board, although a $6 \times 6$ board may be easier, depending on the solution method you choose.

The solution method is your choice. This problem can be solved using State Space Search methods. It has also shown to be solvable using Artificial Neural Networks. You will need to conduct some research to identify the various solution methods, and some learning to support your understanding of them.

While undertaking your research, you will likely come across software solutions for this problem. You could submit one of those solutions, ensuring that you reference the source correctly. Of course, you won't receive marks for this code, and you can only then receive marks based on your presentation of the problem and the solution algorithm. Ideally students should attempt to implement their own software and can be guided by the solutions of others (but please don't simply copy the code of other people). Ensure that you reference all your sources appropriately.

[Danny – 219065161]

## Knights Tour Problem

The knights tour problem is one example of the Hamiltonian Path, a graph theory. Challenges that come with this problem, is deciding on how to solve it. It can be solved in linear time thus we can backtrack (A algorithmic paradigm that computes many different solutions until the solution is found) (Geeks, 2020). This allows us to use any size board up to 8x8.

[Danny – 219065161]

On a chessboard, a Knight's tour is a series of moves in which the Knight crosses each square precisely once according to the rules of chess. The tour is closed if the Knight finishes on a square one Knight's move from the beginning square, otherwise, if it finishes anywhere else then it is open.[1]

Only cycle tours are accepted and there is not any difference between a tour and its reverse.

Knights' legal moves in 8*8 chessboard.

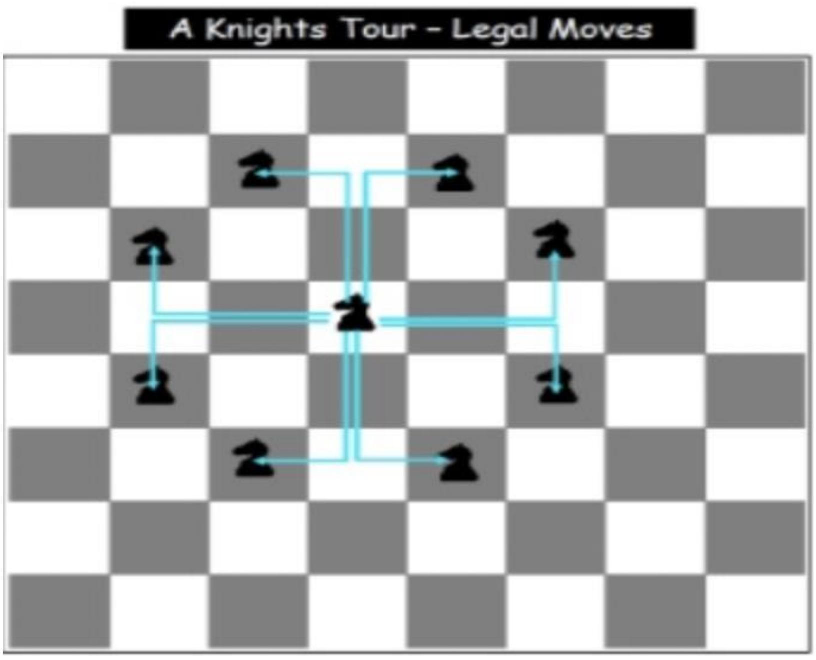


*Figure 1 (Senanayake, 2011)*

[Ramandeep- 218676476]

Graphical representation

Knight’s tour can be represented as a graph. The vertices represent the square of the board and the edges represent a knight’s legal moves between squares.

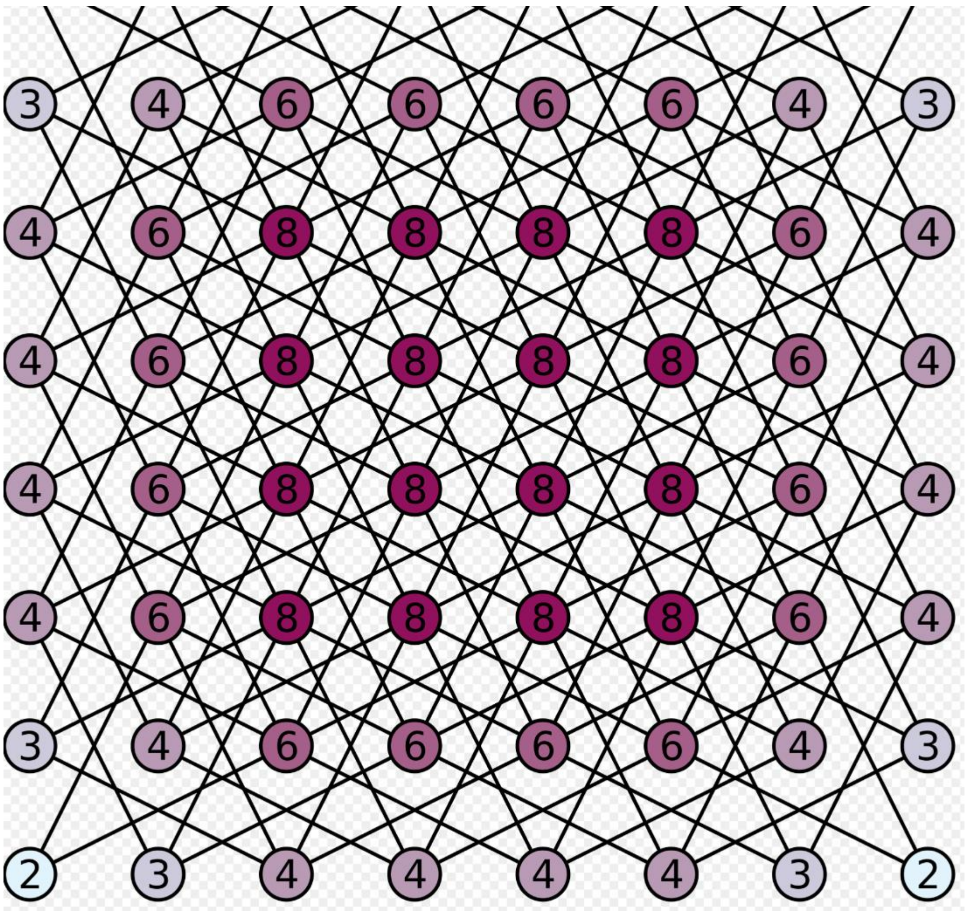


*Figure 2 (Wikipedia, 2021)*

[Ramandeep- 218676476]

## Solution/Algorithm

Backtracking solves problems with a brute force approach, removing any solutions that fail to solve the problem. Using the DFS (Depth-First Search) method, it branches from move to move checking if this meets the constraints, continuing if it does and removing the branch if it does not, returning to the branch before.
[Danny – 219065161]

Pseudocode

```
If all squares visited
    Show solution
Else
    Move knight to new square from AvailableMoves, check if next possible move leads to solution, recursively.
    If move don't lead to solution
        Remove move from AvailableMoves
        Try new Move from AvailableMoves
    If no move returns solution
        Return false, no possible solution
```

(Geeks, 2020)

[Danny – 219065161]

Approach to Knight's Tour Problem

- Brute Force Search is general problem-solving technique and for the selected 8*8 chess board, there are 4 x 10^51 possible move sequences.
- Start moving Knight.
- if knight goes to the cell that there is not further cell available (i.e., it is not reached to the solution yet)
- Then backtrack and change the path.
- Now, check all the available moves from the cell that has the knight and recursively check if it reaches to the solution or not yet.
- If it still hasn't reached to the solution, then again change the path.
- Keep track of each move in the matrix.
- The matrix itself tells us whether we reached to the solution or not (whether we have covered each cell)
- Once all the cells are covered, the value of the last cell must be N*N (if the value is marked from 1 as the steps go up)
- For the above step, the value of the last cell in our case should be (8*8 = 64 ideally) but it is 63 because we marked our first step as zero.

Hence, this is the approach that is used in the Knight's tour problem. [2]

[Ramandeep- 218676476]

## Code

Our solution is coded in python using Visual Studio Code as the IDE (Integrated Development Environment).

It has five functions, and a few variables.

board() – This function passes the solution over to the pygame file that handles the visuals

isSafe(x,y,board) – This function checks whether the move to be taken is within the board dimensions. It takes x, y and board as inputs, which tells it the length and height of the board.

printSol(board) – This function prints the solution once it is found. This is only used for debugging purposes.

Solve() – This function provides a move to another function SolveCheck(board, x, y, moveX, moveY, pos) which tells us whether it is true or false.

SolveCheck(board, x, y, moveX, moveY, pos) – This function checks whether the move given is true or false. Meaning, it checks if the move is safe. This is the backtracking part of our solution.

boardSize – This variable is the board size as an integer.

board – This variable is the board represented as a matrix.  It contains all numbers from 0 to 63, which represents a 8x8 chess board.

moveX – This variable is an array which contains all possible moves the piece can go along the x axis ( across) The negative values mean left and positive mean right.

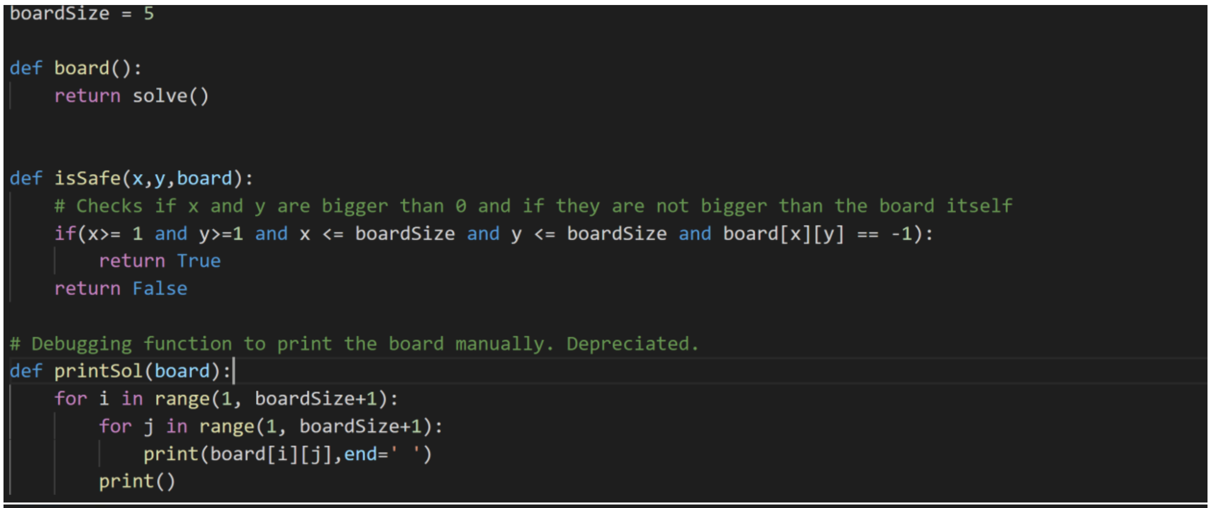
moveY - This variable is an array which contains all possible moves the piece can go along the y axis ( up) The negative values mean down and positive mean up.

Pos – Position will always be 1 at the beginning.

currX – Current x move

curry – Current y move

[Danny – 219065161]

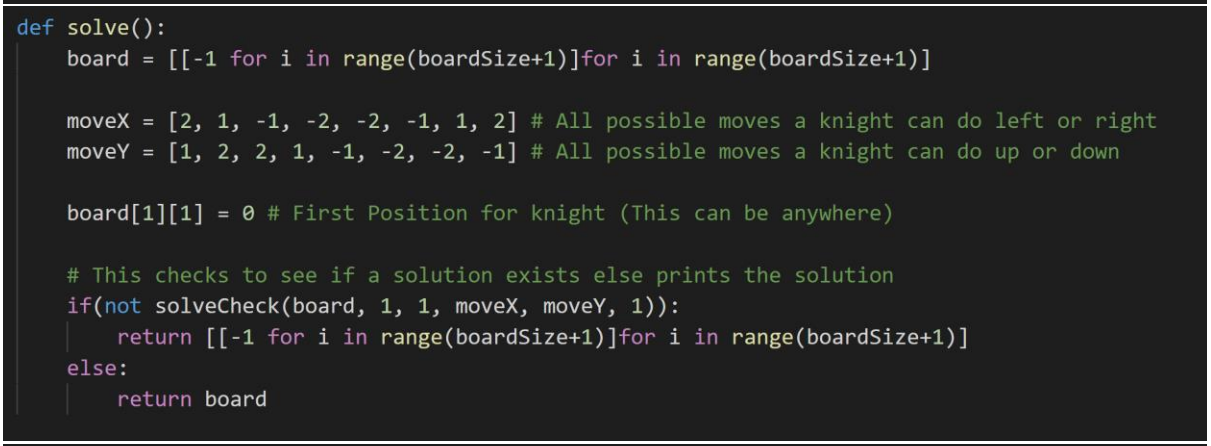


```
boardSize = 5

def board():
    return solve()


def isSafe(x,y,board):
    # Checks if x and y are bigger than 0 and if they are not bigger than the board itself
    if(x>= 1 and y>=1 and x <= boardSize and y <= boardSize and board[x][y] == -1):
        return True
    return False

# Debugging function to print the board manually. Depreciated.
def printSol(board):
    for i in range(1, boardSize+1):
        for j in range(1, boardSize+1):
            print(board[i][j],end=' ')
        print()
```

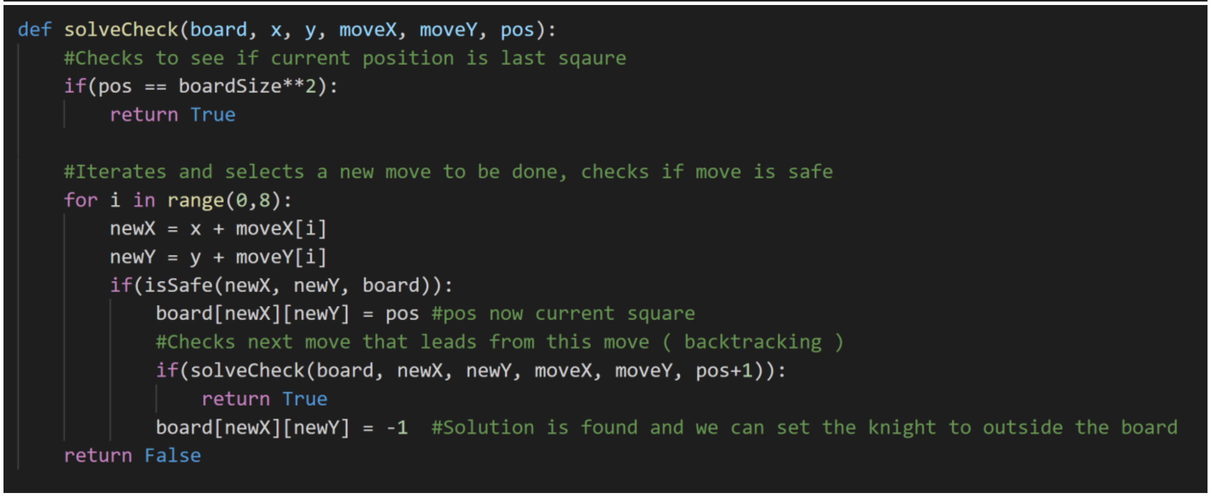


```
def solve():
    board = [[-1 for i in range(boardSize+1)]for i in range(boardSize+1)]

    moveX = [2, 1, -1, -2, -2, -1, 1, 2] # All possible moves a knight can do left or right
    moveY = [1, 2, 2, 1, -1, -2, -2, -1] # All possible moves a knight can do up or down

    board[1][1] = 0 # First Position for knight (This can be anywhere)

    # This checks to see if a solution exists else prints the solution
    if(not solveCheck(board, 1, 1, moveX, moveY, 1)):
        return [[-1 for i in range(boardSize+1)]for i in range(boardSize+1)]
    else:
        return board
```



```
def solveCheck(board, x, y, moveX, moveY, pos):
    #Checks to see if current position is last sqaure
    if(pos == boardSize**2):
        return True

    #Iterates and selects a new move to be done, checks if move is safe
    for i in range(0,8):
        newX = x + moveX[i]
        newY = y + moveY[i]
        if(isSafe(newX, newY, board)):
            board[newX][newY] = pos #pos now current square
            #Checks next move that leads from this move ( backtracking )
            if(solveCheck(board, newX, newY, moveX, moveY, pos+1)):
                return True
            board[newX][newY] = -1  #Solution is found and we can set the knight to outside the board
    return False
```

Discussion of Solution

Currently this solution outputs the squares as numbers, which represents in what order did they travel.
Though this can be visually represented better, using animations and a faster search method we can watch what moves are taken instead of only seeing the output.
Examples of this are shown in our presentation, going through a few test cases.
We can further improve this by including a proper chessboard layout, chess pieces and a one by one square movement.
[Danny – 219065161]

The implementation for the Knight’s tour problem is shown and one of the possible solutions is printed in the form of 2D matrix. The output is 2D 8*8 matrix and it has numbers from 0 to 63 and these are the steps that are made by the Knight. [Ramandeep- 218676476]

## References

Brown, A. J., 2017. *San Jose State University.* [Online]
Available at: https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=8383&context=etd_theses
[Accessed 19 04 2021].

Datta, S., 2020. *Baeldung.* [Online]
Available at: https://www.baeldung.com/cs/backtracking-algorithms
[Accessed 14 April 2021].

De, S., n.d. *Code Speedy.* [Online]
Available at: https://www.codespeedy.com/the-knights-tour-problem-in-python/
[Accessed 14 April 2021].

Field, B., n.d. [Online]
Available at: https://bradfieldcs.com/algos/graphs/knights-tour/
[Accessed 14 April 2021].

Geeks, G. F., 2020. *Geeks For Geeks.* [Online]
Available at: https://www.geeksforgeeks.org/the-knights-tour-problem-backtracking-1/
[Accessed 14 April 2021].

Sam, S., 2018. *Tutorials Point.* [Online]
Available at: https://www.tutorialspoint.com/The-Knight-s-tour-problem
[Accessed 14 April 2021].