Print all possible Knight's tours on a chessboard

Given a chessboard, print all sequences of moves of a knight on a chessboard such that the knight visits every square only once.

For example, for the standard `8 × 8` chessboards, below is one such tour. We have started the tour from the top-leftmost of the board (marked as 1), and the next number represents the knight's consecutive moves.

```
 1  50  45  62  31  18   9  64
46  61  32  49  10  63  30  17
51   2  47  44  33  28  19   8
60  35  42  27  48  11  16  29
41  52   3  34  43  24   7  20
36  59  38  55  26  21  12  15
53  40  57   4  23  14  25   6
58  37  54  39  56   5  22  13
```

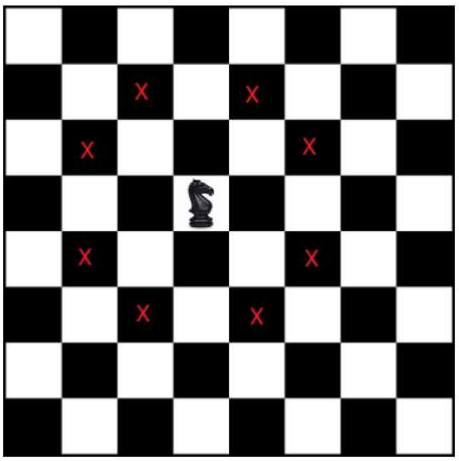
Practice this problem

Suggested Read:

Chess Knight Problem | Find the shortest path from source to destination

The knight should search for a path from the starting position until it visits every square or exhausts all possibilities. We can easily achieve this with the help of backtracking. We start from the given source square in the chessboard and recursively explore all eight paths possible to check if they lead to the solution or not. If the current path doesn't reach the destination or explored all possible routes from the current square, backtrack. To make sure that the path is simple and doesn't contain any cycles, keep track of squares involved in the current path in a chessboard, and before exploring any square, ignore the square if it is already covered in the current path.

We know that a knight can move in 8 possible directions from a given square, as illustrated in the following figure:



We can find all the possible locations the knight can move to from the given location by using the array that stores the knight movement's relative position from any location. For example,

```
If the current location is (x, y), we can move to position (x + row[k], y + col[k]) for 0
<= k <=7 using the following arrays:

row[] = [ 2, 1, -1, -2, -2, -1, 1, 2, 2 ]
col[] = [ 1, 2, 2, 1, -1, -2, -2, -1, 1 ]

So, from a position (x, y) in the chessboard, the valid moves are:

(x + 2, y + 1)
(x + 1, y + 2)
(x - 1, y + 2)
(x - 2, y + 1)
(x - 2, y - 1)
(x - 1, y - 2)
(x + 1, y - 2)
(x + 2, y - 1)
```

Important Note: Please avoid changing sequence of above arrays. The order in which the knight will move is circular and will be optimum. Using the above order, we will get to a vacant position in a few moves. Also, it is always better to start backtracking from any corner of the chessboard. If we start from somewhere middle, the knight can go in 8 different directions. If we start from the corner, the knight can go to only two points from there. Since the algorithm is exponential, optimized input to it can make a huge difference.

Following is the C++, Java, and Python program that demonstrates it:

C++ **Java** Python

```java
import java.util.Arrays;

class Main
{
    // `N x N` chessboard
    public static final int N = 5;

    // Below arrays detail all eight possible movements for a knight.
    // Don't change the sequence of the below arrays
    public static final int[] row = { 2, 1, -1, -2, -2, -1, 1, 2, 2 };
    public static final int[] col = { 1, 2, 2, 1, -1, -2, -2, -1, 1 };

    // Check if `(x, y)` is valid chessboard coordinates.
    // Note that a knight cannot go out of the chessboard
    private static boolean isValid(int x, int y)
    {
        if (x < 0 || y < 0 || x >= N || y >= N) {
            return false;
        }

        return true;
    }

    private static void print(int[][] visited)
    {
        for (var r: visited) {
            System.out.println(Arrays.toString(r));
        }
        System.out.println();
    }
    public static void knightTour(int[][] visited, int x, int y, int pos)
    {
        // mark the current square as visited
        visited[x][y] = pos;

        // if all squares are visited, print the solution
        if (pos >= N*N)
        {
            print(visited);
            // backtrack before returning
            visited[x][y] = 0;
            return;
        }

        // check for all eight possible movements for a knight
        // and recur for each valid movement
        for (int k = 0; k < 8; k++)
        {
            // get the new position of the knight from the current
            // position on the chessboard
            int newX = x + row[k];
            int newY = y + col[k];

            // if the new position is valid and not visited yet
            if (isValid(newX, newY) && visited[newX][newY] == 0) {
                knightTour(visited, newX, newY, pos + 1);
            }
        }

        // backtrack from the current square and remove it from the current path
        visited[x][y] = 0;
    }

    public static void main(String[] args)
    {
        // `visited[][]` serves two purposes:
```

```
    }

    public static void main(String[] args)
    {
        // `visited[][]` serves two purposes:
        // 1. It keeps track of squares involved in the knight's tour.
        // 2. It stores the order in which the squares are visited.
        int[][] visited = new int[N][N];
        int pos = 1;

        // start knight tour from corner square `(0, 0)`
        knightTour(visited, 0, 0, pos);
    }
}
```

Output:

```
[1, 6, 15, 10, 21]
[14, 9, 20, 5, 16]
[19, 2, 7, 22, 11]
[8, 13, 24, 17, 4]
[25, 18, 3, 12, 23]

[1, 6, 11, 18, 21]
[12, 17, 20, 5, 10]
[7, 2, 15, 22, 19]
[16, 13, 24, 9, 4]
[25, 8, 3, 14, 23]

[1, 6, 11, 16, 21]
[12, 15, 20, 5, 10]
[7, 2, 13, 22, 17]
[14, 19, 24, 9, 4]
[25, 8, 3, 18, 23]

[1, 6, 17, 12, 21]
[16, 11, 20, 5, 18]
[7, 2, 9, 22, 13]
[10, 15, 24, 19, 4]
[25, 8, 3, 14, 23]

… and 300 other knight’s tours
```