Knight’s Tour Problem

Introduction

The Knight's Tour Problem is one of the famous problem in which we have the knight on a chessboard.

The knight is supposed to visit every square exactly once.

Following it we have two cases:

- **Closed Tour** : The knight ends on a square that is one move away from the beginning square.

  This means that if it continues to move, with the same path that it has followed till now, it will be able to traverse all squares again.
- **Open Tour** : The knight ends on any other square.

  This problem is solved on a n x n chessboard.

  "n" could have any value.

  This problem is derived from the **Hamiltonian Path Problem** in **Graph Theory**.

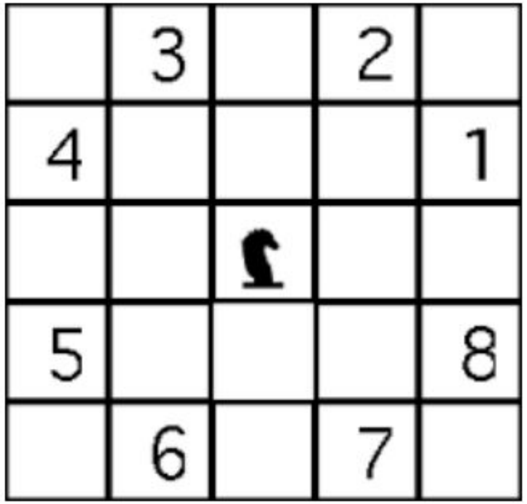
Knight Tour Problem

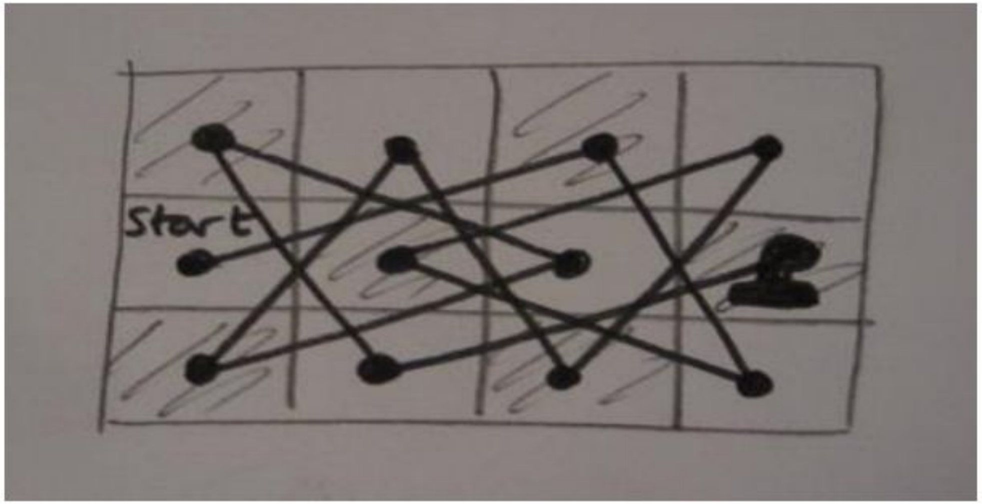
• The knight is placed on any block of an empty board and is move according to the rules of chess, must visit each square exactly once.

• 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 tour. It is also called as Hamiltonian path.

• A cycle that uses each graph vertex of a graph exactly once is called a Hamiltonian cycle

• Knight's tour can be defined on any grid pattern.


3
2
4
1
5
8
6
7


Start

Building Knight Graph

- Each square on the chessboard can be represented as a node in the graph.
- Each legal move by the knight can be represented as an edge in the graph.
- Legal move is shift one square along one axis and two square along the other axis

Legal moves for a knight

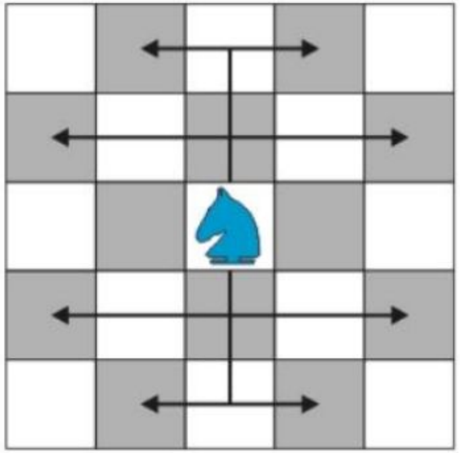
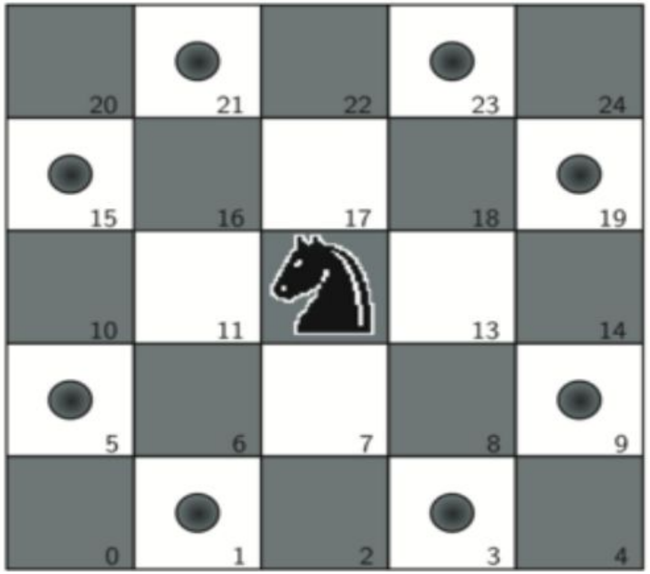
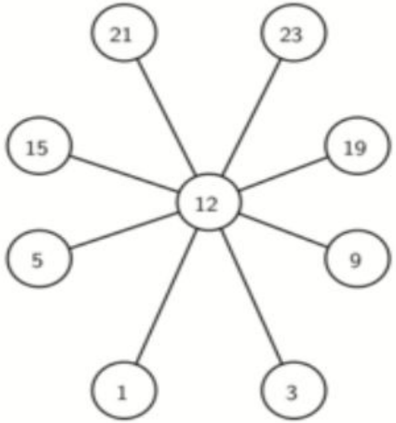


Figure 1: Legal moves for a knight





Bipartite Graph

A bipartite is a graph whose vertices can be divided into two disjoint sets U and V (that is, U and V are each independent sets) such that every edge connects a vertex in U to one in V.

- The two sets U and V may be thought of as a coloring of the graph with two colors: if one colors all nodes in U blue, and all nodes in V green, each edge has endpoints of differing colors.
- All acyclic graph is bipartite, and a cyclic graph is bipartite if all its cycles are of even length.
- A graph is bipartite if and only if it does not contain an odd cycle. if it is 2-colorable

Hamiltonian Path Problem

Hamiltonian Path Problem is focused on finding out if there is a path or route in undirected graph from the beginning to ending node. However, it can gain a lot of complexity even on little increase in its size.

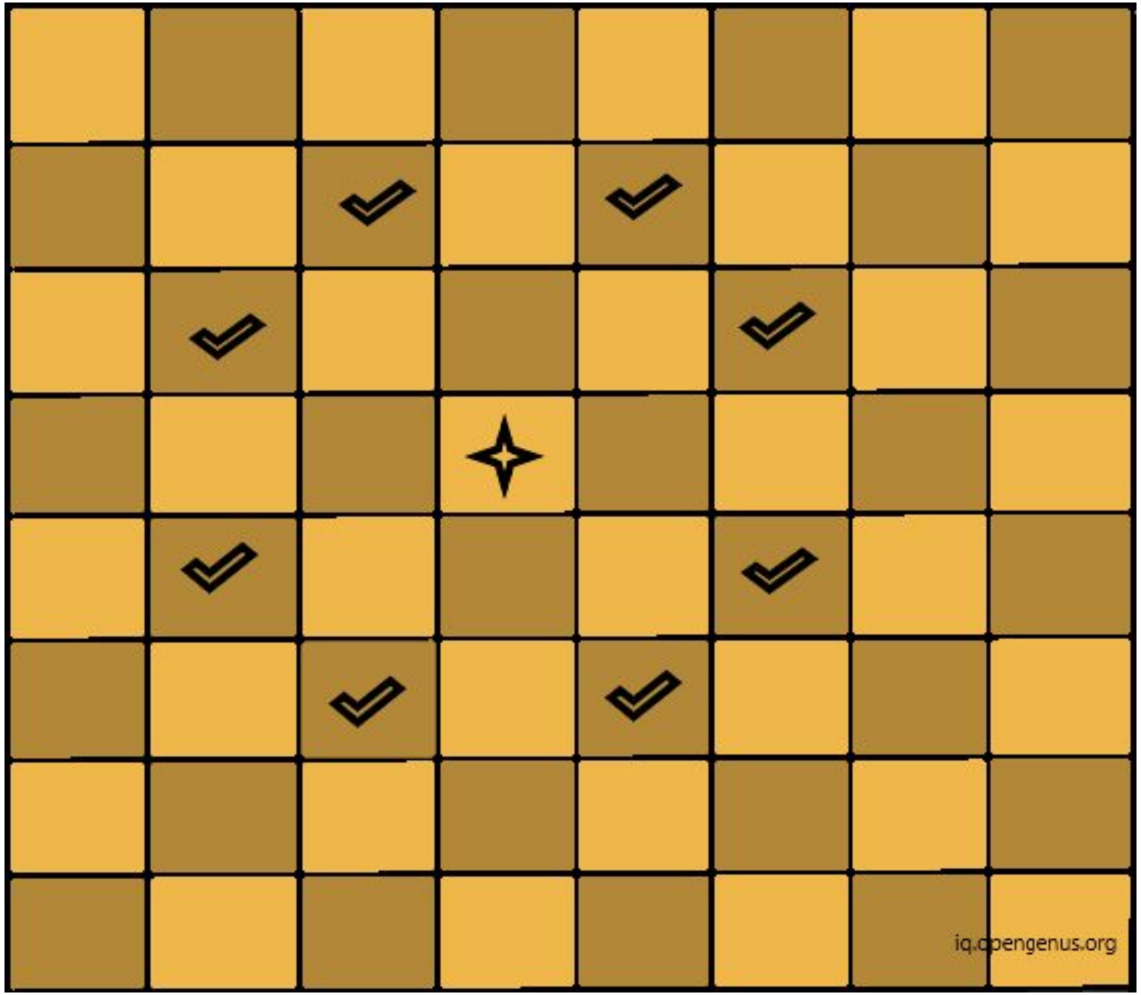
Hamiltonian Path is the one that visits each vertex of the graph only once. A graph with Hamiltonian Path in it is called traceable graph.

However, there are ways to solve this problem in linear time.

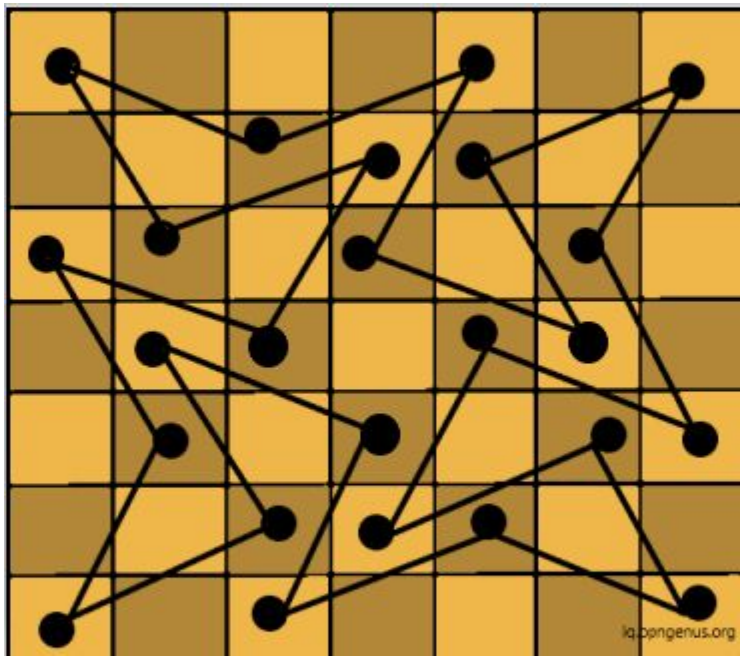
Consider the below image, let the diamond represent the "knight" and the tick marks represent its possible moves at given position. A "knight" can move two squares in cardinal direction then one square in orthogonal direction.

Special Properties of Knight Graph

- Knight Graph are bipartite graph.
- In Knight graph, no two graph vertices within the same set are adjacent.
- A knight move always alternates between white and black squares.
- Knight graph are equivalent to two-colorable graph. Its chromatic number is 2.


iq.opengenus.org

Example of Knight's Tour Problem for 7x7 board:



Algorithm

There are many variants possible for this problem like having two knights and so on. In this article we will explore the basic problem with one knight and how it can cover all squares without revisiting them.One way to solve this problem is Naive Approach, to find all possible configurations and choose the correct one. Since, this would take much longer if the value of n in n x n board is large.

Algorithm:

```
while all path are not visited

{

    traverse next path

    if all squares are covered {

        print the path

    }

    }
```

Backtracking

Other technique that can be used for this problem is **Backtracking**. It is optimized version of our naive technique.

Backtracking is the technique in which we try all possible solutions or combinations till we find the one that is correct. In this each combination is tried only once.

It incrementally makes the solution using recursion. We consider on step at a time. If they do not satisfy the constraints then we remove those particular steps.

Algorithm:

```
If all squares have been visited then return the solution
Else:
    Consider one move and check each move if it leads to solution recursively.
    If it does not lead us to solution then remove it and check for another solution
    If none of them works then:
        If its not the first call, then return previous item to be added to solution
        Else return "no solution exists"
```

Time Complexity: O(n * n) for traversing the n x n squares.

Python implementation

```
import numpy as np

def knight_tour(n):

   board=[[-1 for i in range(n)]for j in range(n)]

   knight_tour_helper(n=n,board=board,x=0,y=0,counter=0)

   print(np.array(board))

def knight_tour_helper(n,board,x,y,counter):

   if counter==n*n:

      return True

   if (x<0) or (x>=n) or (y<0) or (y>=n)or board[y][x]!=-1:

      return False

   board[y][x]=counter

   for x_move,y_move in zip([-2,-2,-1,-1,1,1,2,2],[-1,1,-2,2,-2,2,-1,1]):

      if knight_tour_helper(n,board,x+x_move,y+y_move,counter+1):

         return True

   board[y][x]=-1

   return False

knight_tour(8)
```

Output:

```
[[ 0  9 30 63 32 25 52 61]
 [11  6 27 24 29 62 33 50]
 [ 8  1 10 31 26 51 60 53]
 [ 5 12  7 28 23 34 49 40]
 [ 2 17  4 35 48 39 54 59]
 [13 20 15 22 45 56 41 38]
 [16  3 18 47 36 43 58 55]
 [19 14 21 44 57 46 37 42]]
```

Warnsdorff’s Rule:

1. We can start from any initial position of the knight on the board.
2. We always move to an adjacent, unvisited square with minimal degree (minimum number of unvisited adjacent).

This algorithm may also more generally be applied to any graph.

Some definitions:

- A position Q is accessible from a position P if P can move to Q by a single Knight’s move, and Q has not yet been visited.
- The accessibility of a position P is the number of positions accessible from P.

Algorithm:

1. Set P to be a random initial position on the board
2. Mark the board at P with the move number “1”
3. Do following for each move number from 2 to the number of squares on the board:
    - let S be the set of positions accessible from P.
    - Set P to be the position in S with minimum accessibility
    - Mark the board at P with the current move number
4. Return the marked board — each square will be marked with the move number on which it is visited.

Solution Approach

- Brute force
- Neural networks
- Depth first search with backtracking:- Knight moves to a square that has the lowest number of next moves available. The idea is that at the end of the tour it will visit squares that have more move choices available.
- Ant colony optimization:- ACO has a good capability of finding better tour path, which not only uses the feedback principle to quicken evolution process of colonies but also is an essential parallel algorithm.
- The Knight’s tour problem can be solved in linear time.