**Artificial Intelligence and Soft Computing**

**CS3101**

**Chess Knight Problem**

**Shruti Gupta**

**Reg. No – 209301001**

**Reva Shandil**

**Reg. No - 209301389**

**Chess Knight Problem**

Given a chessboard, find the shortest distance (minimum number of steps) taken by a knight to reach a given destination from a given source.

1. **PEAS**

Performance Measure – Goal, Minimum distance to reach destination

Environment – Chess board

Agent – Player, Knight’s Move

Sensors – Eyes of the player

State – a 8 x 8 matrix, a knight

Initial State – (x, y)

Action – Movement of knight - L shape - up, down, right, left

2 squares in any direction vertically followed by 1 square horizontally or vice versa.

Final State – (x, y)

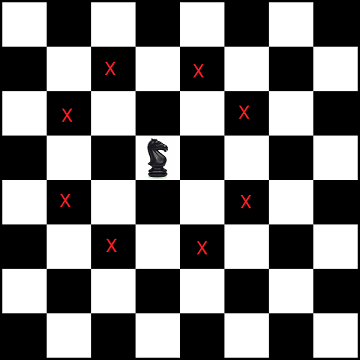
Transition States

For example, if the current location is (x, y), we can move to (x + row[k], y + col[k]) for 0 <= k <= 7 using the following array:

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

So, from position (x, y) knight’s can move to:

(x + 2, y – 1)  
(x + 2, y + 1)  
(x – 2, y + 1)  
(x – 2, y – 1)  
(x + 1, y + 2)  
(x + 1, y – 2)  
(x – 1, y + 2)  
(x – 1, y – 2)



1. **Search Technique** – Uniformed Search – Breadth First Search (BFS)

[**Breadth–first search (BFS)**](https://www.techiedelight.com/breadth-first-search/)  it is the shortest path problem. Following is the complete algorithm:

1. Create an empty queue and enqueue the source cell having a distance of 0 from the source (itself).
2. Loop till queue is empty:
   1. Dequeue next unvisited node.
   2. If the popped node is the destination node, return its distance.
   3. Otherwise, we mark the current node as visited. For each of eight possible movements for a knight, enqueue each valid movement with +1 distance (minimum distance of a given node from the source is one more than the minimum distance of parent from source).
3. **Optimization Technique** – Global Heuristic

Initial State – (7,0)

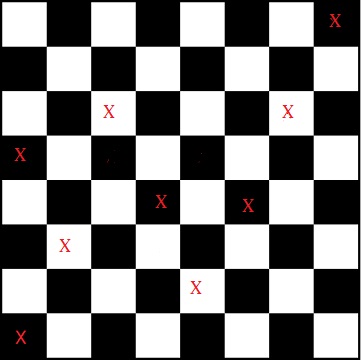
Action – Movement of knight - L shape - up, down, right, left

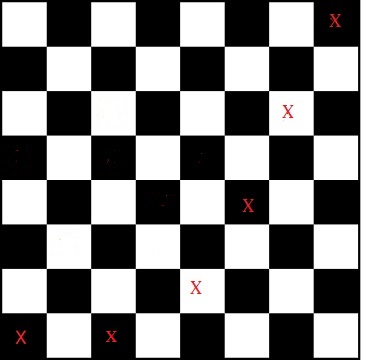
2 squares in any direction vertically followed by 1 sqaure horizontally or vice versa.

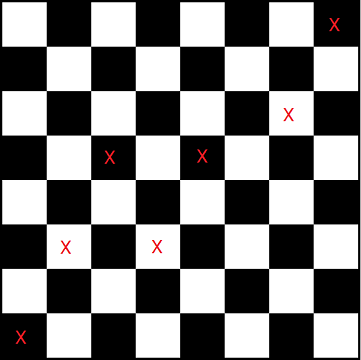
Final State – (0,7)

Path Cost – Minimum distance from current square to destination square (1 cost per square)

Transition States –

**** (7,0) 🡪 (6,2) 🡪 (5,4) 🡪 (4,6) 🡪 (3,4) 🡪 (2,2) 🡪(0,3) 🡪 (1,5) 🡪 (0,7) (8 moves)

****(7,0)🡪(6,2)🡪(5,4)🡪(4,6)🡪(2,7)🡪 further move not possible (Sol not possible)

 (7,0) 🡪(6,2) 🡪 (4,3) 🡪(3,5) 🡪(2,3) 🡪(1,5) 🡪 (0,7) (6 moves)

**As, the minimum distance taken by knight to reach the destination from the given source is within 6 moves, so this is the optimal solution. (After exploring all the states)**

**Python Code**

import sys

from collections import deque

# A queue node used in BFS

class Node:

# (x, y) represents chessboard coordinates

# `dist` represents its minimum distance from the source

def \_\_init\_\_(self, x, y, dist=0):

self.x = x

self.y = y

self.dist = dist

# As we are using `Node` as a key in a dictionary,

# we need to override the `\_\_hash\_\_()` and `\_\_eq\_\_()` function

def \_\_hash\_\_(self):

return hash((self.x, self.y, self.dist))

def \_\_eq\_\_(self, other):

return (self.x, self.y, self.dist) == (other.x, other.y, other.dist)

# Below lists detail all eight possible movements for a knight

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

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

# Check if (x, y) is valid chessboard coordinates.

# Note that a knight cannot go out of the chessboard

def isValid(x, y, N):

return not (x < 0 or y < 0 or x >= N or y >= N)

# Find the minimum number of steps taken by the knight

# from the source to reach the destination using BFS

def findShortestDistance(src, dest, N):

# set to check if the matrix cell is visited before or not

visited = set()

# create a queue and enqueue the first node

q = deque()

q.append(src)

# loop till queue is empty

while q:

# dequeue front node and process it

node = q.popleft()

x = node.x

y = node.y

dist = node.dist

# if the destination is reached, return distance

if x == dest.x and y == dest.y:

return dist

# skip if the location is visited before

if node not in visited:

# mark the current node as visited

visited.add(node)

# check for all eight possible movements for a knight

# and enqueue each valid movement

for i in range(len(row)):

# get the knight's valid position from the current position on

# the chessboard and enqueue it with +1 distance

x1 = x + row[i]

y1 = y + col[i]

if isValid(x1, y1, N):

q.append(Node(x1, y1, dist + 1))

# return infinity if the path is not possible

return sys.maxsize

if \_\_name\_\_ == '\_\_main\_\_':

N = 8 # N x N matrix

src = Node(0, 7) # source coordinates

dest = Node(7, 0) # destination coordinates

print("The minimum number of steps required is",

findShortestDistance(src, dest, N))

**Output**

The minimum number of steps required is 6