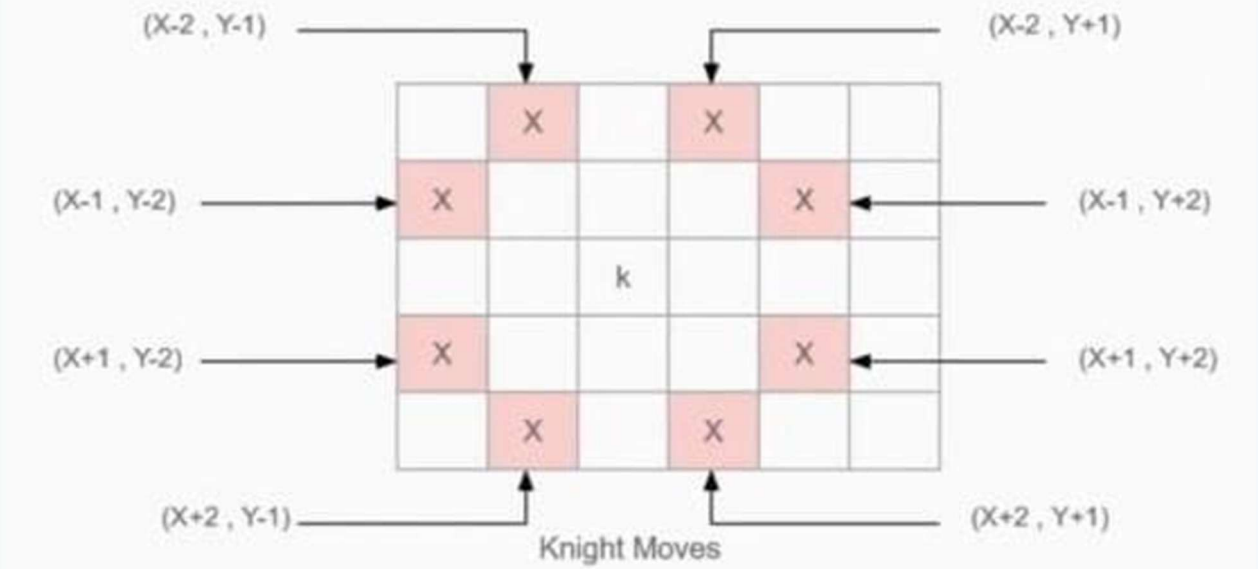
DATA STRUCTURES AND ALGORITHM ASSIGNMENT PRESENTATION

BATCH MEMBERS:
BY:
20L108-DHEVAA DHARSHINI B
20L116-JEYASHRI R
20L141-SHEENA S

MINIMUM NUMBER OF MOVES REQUIRED BY THE KNIGHT TO REACH THE DESTINATION

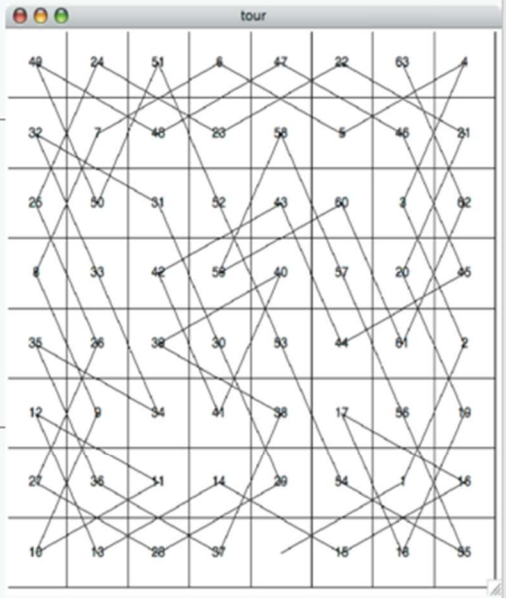
- We have implemented the code in python language using python idle text editor.
- We are required to reach the target cell without visiting a node twice.This concept of finding the shortest path is implemented using Breadth First Search(BFS).Each cell is considered to be a node in a graph.
- Graph traversal is a technique used for searching a vertex in a graph. The graph traversal is also used to decide the order of vertices is visited in the search process. A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into looping path.



BFS(BREADTH FIRST SEARCH)

Breadth–first search (bfs) is an algorithm for traversing or searching tree or graph data structures. it starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key') and explores the neighbor nodes first before moving to the next-level neighbors.

- **Step 1 -** Define a Queue of size total number of vertices in the graph.
- **Step 2 -** Select any vertex as **starting point** for traversal. Visit that vertex and insert it into the Queue.
- **Step 3 -** Visit all the non-visited **adjacent** vertices of the vertex which is at front of the Queue and insert them into the Queue.
- **Step 4 -** When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.
- **Step 5 -** Repeat steps 3 and 4 until queue becomes empty.
- **Step 6 -** When queue becomes empty, then produce final spanning tree by removing unused edges from the graph



ALGORITHM:

The idea is to use Breadth–first search (BFS) as it is the shortest path problem. Following is the complete algorithm:

- Create an empty queue and enqueue the source cell having a distance of 0 from the source (itself).
- Loop till queue is empty:
- Dequeue next unvisited node.
- If the popped node is the destination node, return its distance.
- 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).

SOURCE CODE:

```
import sys
from collections import deque
class Node:
    def __init__(self, x, y, dist=0):
        self.x = x
        self.y = y
        self.dist = dist
row = [2, 2, -2, -2, 1, 1, -1, -1]
col = [-1, 1, 1, -1, 2, -2, 2, -2]
def isValid(x, y, N):
    return not (x < 0 or y < 0 or x >= N or y >= N)
def findShortestDistance(src, dest, N):
    visited = set()
    q = deque()
    q.append(src)
    while q:
        node = q.popleft()
        x = node.x
        y = node.y
        dist = node.dist
        if x == a and y == b:
            return dist
        if node not in visited:
            visited.add(node)
```

```
            for i in range(len(row)):
                x1 = x + row[i]
                y1 = y + col[i]
                if isValid(x1, y1, N):
                    q.append(Node(x1, y1, dist + 1))

    return sys.maxsize


if __name__== '__main__':

    N = 8
    print("Enter the coordinates of souce and destination")
    x=int(input())
    y=int(input())
    a=int(input())
    b=int(input())
    src=Node(x,y)
    dest=Node(a,b)
    print("The minimum number of steps required is",findShortestDistance(src, dest, N))

```

OUTPUT SCREEN:

```
========== RESTART: C:\Users\Admin\.android\Desktop\dsa final code.py ==========
Enter the coordinates of souce and destination
0
7
7
0
The minimum number of steps required is 6
>>>
========== RESTART: C:\Users\Admin\.android\Desktop\dsa final code.py ==========
Enter the coordinates of souce and destination
0
0
1
0
The minimum number of steps required is 3
>>>
```

Thank you