```
def knight_tour_bfs(board_size):
    #Starting position
    #Top left position
    start_row, start_column = 0, 0

    #board is going to be a 2D comprehension, the outer list create a board_size numrow, and the inner list create board_size element of value -1 with each row.
    board = [[-1 for i in range(board_size)]for i in range(board_size)]
    #set 0 as a starting position as a first unvisited square
    board[start_row][start_column] = 0

    #initialised as a list containing tuple representing the starting position
    queue = [(start_row, start_column)]
    #move count initialise to 0 to keep track to the knight move
    move_count = 0

    #list of represent the eight possible knight move from any given square. Each tuple represent the change of row(dx) and the change of column(dy) for knight move.
    possible_moves  = [
        (1, -2), (-2, 1), (-1, 2), (2, -1),
        (-1, -2), (2, 1), (-2, -1), (1, 2)
    ]

    #while loop is the core of BFS Algorithm, it contain as long as the queue is not empty, while if there have any empty square, will keep explore
    while queue:
        #the line remove the first position from the queue, assign the its as row and column variables, explore the position at the front of the queue
        row, column = queue.pop(0)
        #just to keep track the knight's move number of the board
        move_count += 1

        #loop iterative through each possible move dx, dy in the list of possible_move
        for dx, dy in possible_moves:
            #calculate the potential new row and new column base on the current possible(row and column) and the possible move of (dx, dy)
            new_row, new_column = row + dx, column + dy

            #check the new row and new column are they within the boundaries, and if the square of the position are unvisited.
            if (0 <= new_row < board_size and 0 <= new_column < board_size and board[new_row][new_column] == -1):
                #if the new position is valid and unvisited
                board[new_row][new_column] = move_count
                queue.append((new_row, new_column))

    #check solution is found, using list comprehension with the all function to check all the element in the board are not =-1.
    #if all tin the board have visited, means BFS found the solution
    if all(element != -1 for row in board for element in row):
        return board
    else:
        #if no solution found, return None
        return None

#select the board size
board_size = 8
#call the function
result = knight_tour_bfs(board_size)
```

```
#
if result:
    for row in result:
        print(row)
else:
    print("No solution found")
```