```
#######################################BFS
ALGORITHM##################################################################

from collections import deque
import sys
import time

board = []


def ValidMove(board, x, y):
    """
    :param board: NxN board
    :param x: x position
    :param y: y position
    :return: return 2d board
    """
    return 0 <= x < len(board) and 0 <= y < len(board[0])


def ShowBoard(board):
    """
    :param board: NxN board
    :return: Print board on console
    """
    for row in board:
        print("".join(row))
    print()


def Neighbours(board, pos):
    """
    :param board: NxN board
    :param pos: New position of robot
    :return: List of its neighbor
    """
    neighbors = []
    x, y = pos
    for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
        new_x, new_y = x + dx, y + dy
        if ValidMove(board, new_x, new_y) and board[new_x][new_y] != 'O':
            neighbors.append((new_x, new_y))
    return neighbors


def Initstate():
    """
    :return: Initial state of the board
    """
    print('Initial Board\n')
    file_obj = open(sys.argv[1])
    rows = file_obj.readlines()
    for line in rows:
        if line != "":
            board.append(line.strip())
        else:
            break
    return board


def Successor(board, robot, boxes, storage):
    """
    :param board: NxN board
    :param robot: Location of robot
    :param boxes: Location of boxes
    :param storage: Location of the storage
    :return:Show next board state
    """
    start_state = (robot, boxes)
    visited = set()
    queue = deque([(start_state, [])])

    while queue:
        (robot, boxes), actions = queue.popleft()

        if sorted(boxes) == sorted(storage):
            return actions

        if (robot, boxes) in visited:
            continue

        visited.add((robot, boxes))

        solution_board = [list(row) for row in board]
        for box in boxes:
            x, y = box
            solution_board[x][y] = 'B'
        solution_board[robot[0]][robot[1]] = 'R'
        ShowBoard(solution_board)

        for neighbor in Neighbours(board, robot):
            new_robot = neighbor
            new_boxes = list(boxes)
            for i, box in enumerate(new_boxes):
                if box == new_robot:
                    new_box = (box[0] + (box[0] - robot[0]), box[1] + (box[1] - robot[1]))
                    if ValidMove(board, *new_box) and board[new_box[0]][new_box[1]] != 'O':
                        new_boxes[i] = new_box
                        break

            new_state = (new_robot, tuple(new_boxes))
            new_actions = actions + [neighbor]
            queue.append((new_state, new_actions))

    return None


def GetStoragePos(board):
    """
    :param board: NxN board
    :return: Positions of the Storages
    """
    return {(x, y) for x in range(len(board)) for y in range(len(board[0])) if board[x][y] ==
'S'}


def GetRobotPos(board):
    """
    :param board: NxN board
    :return: Positions of the Robot
    """
    return [(x, y) for x in range(len(board)) for y in range(len(board[0])) if board[x][y] ==
'R'][0]


def GetBoxPos(board):
    """
    :param board:NxN board
    :return: Positions of the boxes
    """
    return tuple((x, y) for x in range(len(board)) for y in range(len(board[0])) if board[x]
[y] == 'B')
```

```
# Driver Code
board = Initstate()

storage = GetStoragePos(board)
robot = GetRobotPos(board)
boxes = GetBoxPos(board)

start_time = time.time()
actions = Successor(board, robot, boxes, storage)

if actions:
    print("End  solution  can  be  found")
    solution_board = [list(row) for row in board]
    for action in actions:
        x, y = action
        solution_board[x][y] = 'R'
    ShowBoard(solution_board)
else:
    print("End State cannot be found.")
print(f'Total execution time is {time.time() - start_time}')
```