```
#####################################GBS
ALGORITHM#########################################################################
from heapq import heappop, heappush
import time
import sys
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 ManhattanDistance(a, b):
    """
    :param a: Any position
    :param b: Any position
    :return: Absolute distance between 2 co-ordinates
    """
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def Heuristic(board, robot, boxes, targets):
    """
    :param board: NxN Board
    :param robot: Robot position
    :param boxes: Boxes position
    :param targets: Target locations
    :return: total distance
    """
    total_dist = 0
    for box in boxes:
        min_dist = min(ManhattanDistance(box, target) for target in targets)
        total_dist += min_dist
    return total_dist

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

def Successor(board, robot, boxes, targets):
    """
    :param board: NxN board
    :param robot: Location of robot
    :param boxes: Location of boxes
    :param targets: Location of the targets
    :return:Show next board state
    """
    directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]

    def is_goal(boxes):
        return sorted(boxes) == sorted(targets)

    def apply_move(entity, direction):
        return (entity[0] + direction[0], entity[1] + direction[1])

    start_state = (robot, tuple(sorted(boxes)))
    visited = set()
    priority_queue = [(Heuristic(board, robot, boxes, targets), start_state)]  # (heuristic,
(robot, boxes))

    while priority_queue:
```

```
        _, (robot, boxes) = heappop(priority_queue)

        if is_goal(boxes):
            return boxes

        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 direction in directions:
            new_robot = apply_move(robot, direction)
            if not ValidMove(board, *new_robot) or board[new_robot[0]][new_robot[1]] == 'O':
                continue

            if (new_robot, boxes) not in visited:
                heappush(priority_queue, (Heuristic(board, new_robot, boxes, targets),
(new_robot, boxes)))

    return None

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 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 = []
board = Initstate()

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

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

if result:
    print("End  state can be found:")
    solution_board = [list(row) for row in board]
    for box in result:
        x, y = box
        solution_board[x][y] = 'B'
    print("\n".join(["".join(row) for row in solution_board]))
    print()

else:
    print("End state  cannot  be  found.")

print(f'Total execution time is {time.time() - start_time}')
```