ASSIGNMENT-6

①
```
def astar(maze, start, end):
    def heuristic(node, goal):
        return abs(node[0]-goal[0]) + abs(node[1]-goal[1])
    open_list = [(start, 0)]
    came_from = {}
    g_score = {start: 0}
    while open_list:
        current, _ = min(open_list, key = lambda x: g_score[x[0]]
            + heuristic(x[0], end))
        open_list.remove((current, g_score[current]))
        if current == end:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]
        for neighbour in [(0,-1), (0,1),(-1,0),(1,0),(-1,-1),(-1,1),(1,-1),
                                                        (1,1)]:
            neighbor_pos = (current[0]+ neighbor[0], current[1]+
                                                    neighbor[1])
            if (neighbor_pos[0] <0 or neighbor_pos[0] >= len(maze)
                or neighbor_pos[1] <0 or neighbor_pos[1] >= len(maze)
                or maze[neighbor_pos[0]][neighbor_pos[1]] == 1):
                continue
            tentative_g_score = g_score[current] +1
            if neighbor_pos not in g_score or tentative_g_score
            < g_score[neighbor_pos]:
                came_from[neighbor_pos] = current
```

```
            g_score[neighbor_pos] = tentative_g_score
            open_list.append((neighbor_pos, g_score[neighbor_pos]))

    return []

def main():
    board = [[0,0,0,0,0,0,0],
             [0,0,0,0,0,0,0],
             [0,0,1,0,0,0,0],
             [0,0,1,0,0,0,0],
             [0,0,1,0,0,0,0],
             [0,0,1,0,0,0,0],
             [0,0,1,0,0,0,0],
             [0,0,0,0,0,0,0],]

    start = (3,0)
    end = (5,4)
    path = astar(board, start, end)
    if not path:
        print("Path not found").
    else:
        print("Path found:", path)

if __name__ == '__main__':
    main()
```

OUTPUT

Path found = [(3,0), (2,1), (1,2), (2,3), (3,4), (4,4), (5,4)]

② def is_valid(x, y, maze):
    if 0 <= x < len(maze) and 0 <= y < len(maze[0]) and
      maze[x][y] == 0:
        return True
    return False

def dfs(maze, start, end, depth):
    if start == end:
        return [start]
    if depth <= 0:

```
    return None
x, y = start
directions = [(0,1),(0,-1),(1,0),(-1,0)]
for dx, dy in directions:
    new_x, new_y = x+dx, y+dy
    if is_valid(new_x, new_y, maze):
        path = dfs(maze, (new_x, new_y), end, depth-1)
        if path is not None:
            return [start] + path
return None

def ids(maze, start, end):
    max_depth = 11
    for depth in range(max_depth):
        path = dfs(maze, start, end, depth)
        if path is not None:
            return path
    return None

def main():
    board = [[0,0,0,0,0,0,0],
             [0,0,0,0,0,0,0],
             [0,0,1,0,0,0,0],
             [0,0,1,0,0,0,0],
             [0,0,1,0,0,0,0],
             [0,0,1,0,0,0,0],
             [0,0,1,0,0,0,0],
             [0,0,0,0,0,0,0],]
    start = (3,0)   end=(5,4)
    path = ids(board, start, end)
    if not path:
        print("No path found")
    else:
        print("Path found = ", path)
if __name__ == '__main__':
    main()
```

OUTPUT

Path found = [(3,0), (3,1), (4,1), (5,1), (6,1), (7,1), (7,2), (7,3),
(7,4), (6,4), (5,4)]