```
import heapq
import math

class Cell:
    def __init__(self, parent_i, parent_j, f, g, h):
        self.parent_i = parent_i
        self.parent_j = parent_j
        self.f = f
        self.g = g
        self.h = h
def is_valid(row, col, ROW, COL):
    return 0 <= row < ROW and 0 <= col < COL
def is_unblocked(grid, row, col):
    return grid[row][col] == 1
def is_destination(row, col, dest):
    return row == dest[0] and col == dest[1]
def calculate_h_value(row, col, dest):
    return math.sqrt((row - dest[0]) ** 2 + (col - dest[1]) ** 2)
def trace_path(cell_details, dest):
    print("\nThe Path is ", end="")
    row, col = dest
    path = []
    while not (cell_details[row][col].parent_i == row and
cell_details[row][col].parent_j == col):
        path.append((row, col))
        temp_row, temp_col = cell_details[row][col].parent_i,
cell_details[row][col].parent_j
        row, col = temp_row, temp_col
    path.append((row, col))
    for p in reversed(path):
        print("->", p, end=" ")
    print()
def a_star_search(grid, src, dest):
    ROW, COL = len(grid), len(grid[0])
    if not (0 <= src[0] < ROW and 0 <= src[1] < COL):
        print("Source is invalid")
        return
    if not (0 <= dest[0] < ROW and 0 <= dest[1] < COL):
        print("Destination is invalid")
        return
    if not is_unblocked(grid, src[0], src[1]) or not is_unblocked(grid,
dest[0], dest[1]):
        print("Source or destination is blocked")
        return
    if is_destination(src[0], src[1], dest):
        print("We are already at the destination")
        return
    closed_list = [[False] * COL for _ in range(ROW)]
    cell_details = [[Cell(-1, -1, math.inf, math.inf, math.inf) for _ in
range(COL)] for _ in range(ROW)]
    i, j = src
    cell_details[i][j] = Cell(i, j, 0.0, 0.0, 0.0)
    open_list = [(0.0, i, j)]
    found_dest = False
    while open_list:
        f, i, j = heapq.heappop(open_list)
        if closed_list[i][j]:
            continue
```

```
        closed_list[i][j] = True
        successors = [
            (i - 1, j), (i + 1, j), (i, j + 1), (i, j - 1),
            (i - 1, j + 1), (i - 1, j - 1), (i + 1, j + 1), (i + 1, j -
1)
        ]
        for successor in successors:
            row, col = successor
            if is_valid(row, col, ROW, COL) and is_unblocked(grid, row,
col):
                if is_destination(row, col, dest):
                    cell_details[row][col].parent_i = i
                    cell_details[row][col].parent_j = j
                    print("The destination cell is found")
                    trace_path(cell_details, dest)
                    found_dest = True
                    return
                if not closed_list[row][col]:
                    g_new = cell_details[i][j].g + 1.0
                    h_new = calculate_h_value(row, col, dest)
                    f_new = g_new + h_new
                    if cell_details[row][col].f == math.inf or
cell_details[row][col].f > f_new:
                        heapq.heappush(open_list, (f_new, row, col))
                        cell_details[row][col].f = f_new
                        cell_details[row][col].g = g_new
                        cell_details[row][col].h = h_new
                        cell_details[row][col].parent_i = i
                        cell_details[row][col].parent_j = j
    if not found_dest:
        print("Failed to find the Destination Cell")
# Driver code
grid = [
    [1, 0, 1, 1, 1, 1, 0, 1, 1, 1],
    [1, 1, 1, 0, 1, 1, 1, 0, 1, 1],
    [1, 1, 1, 0, 1, 1, 0, 1, 0, 1],
    [0, 0, 1, 0, 1, 0, 0, 0, 0, 1],
    [1, 1, 1, 0, 1, 1, 1, 0, 1, 0],
    [1, 0, 1, 1, 1, 1, 0, 1, 0, 0],
    [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
    [1, 0, 1, 1, 1, 1, 0, 1, 1, 1],
    [1, 1, 1, 0, 0, 0, 1, 0, 0, 1]
]
src = (8, 0)
dest = (0, 0)
a_star_search(grid, src, dest)
```