```
import heapq

def dijkstra(graph, start, end):
    queue = [(0, start, [])]
    visited = set()
    while queue:
        (cost, node, path) = heapq.heappop(queue)
        if node in visited:
            continue
        path = path + [node]
        visited.add(node)
        if node == end:
            return (cost, path)
        for (next_node, weight) in graph.get(node, []):
            if next_node not in visited:
                heapq.heappush(queue, (cost + weight, next_node, path))
    return float("inf"), []



import math
import heapq

def heuristic(a, b):
    # Menghitung jarak Euclidean sebagai heuristik
    return math.sqrt((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2)

def a_star(graph, start, end):
    queue = [(0, start)]
    costs = {start: 0}
    parents = {start: None}
    while queue:
        (current_cost, current) = heapq.heappop(queue)
        if current == end:
            path = []
            while current:
                path.append(current)
                current = parents[current]
            return path[::-1]
        for neighbor, weight in graph[current]:
            new_cost = costs[current] + weight
            if neighbor not in costs or new_cost < costs[neighbor]:
                costs[neighbor] = new_cost
```

```
            priority = new_cost + heuristic(neighbor, end)
            heapq.heappush(queue, (priority, neighbor))
            parents[neighbor] = current
    return []



import numpy as np

def cell_decomposition(grid, start, goal):
    rows, cols = len(grid), len(grid[0])
    queue = [(start, [start])]
    visited = set()
    while queue:
        (current, path) = queue.pop(0)
        if current == goal:
            return path
        for direction in [(0, 1), (1, 0), (0, -1), (-1, 0)]: # kanan, bawah, kiri, atas
            next_cell = (current[0] + direction[0], current[1] + direction[1])
            if 0 <= next_cell[0] < rows and 0 <= next_cell[1] < cols and
grid[next_cell[0]][next_cell[1]] == 0 and next_cell not in visited:
                visited.add(next_cell)
                queue.append((next_cell, path + [next_cell]))
    return []
```