# Name : Haider Ali

# SAP ID : 70069779

import heapq

import random

import math

def heuristic\_cost\_estimate(*current*, *goal*):

    """

    Calculate the Manhattan distance heuristic between two points.

    Parameters:

    - current: Current node (x, y).

    - goal: Goal node (x, y).

    Returns:

    - Heuristic distance estimate.

    """

    return abs(*current*[0] - *goal*[0]) + abs(*current*[1] - *goal*[1])

def simulated\_annealing(*current*, *neighbor*, *temperature*):

    """

    Simulated Annealing acceptance probability function.

    Parameters:

    - current: Current node.

    - neighbor: Neighbor node.

    - temperature: Current temperature.

    Returns:

    - True if the move is accepted, False otherwise.

    """

    # Calculate the change in cost (heuristic distance)

    delta = heuristic\_cost\_estimate(*current*, *neighbor*)

    # If the new solution is better, accept it

    if delta < 0:

        return True

    # If the new solution is worse, accept it with a probability

    probability = math.exp(-delta / *temperature*)

    return random.uniform(0, 1) < probability

def hill\_climbing(*grid*, *current*, *goal*):

    """

    Hill Climbing algorithm to explore neighboring nodes.

    Parameters:

    - grid: 2D grid representing the environment.

    - current: Current node (x, y).

    - goal: Goal node (x, y).

    Returns:

    - List of neighboring nodes.

    """

    movements = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    neighbors = []

    for movement in movements:

        neighbor = (*current*[0] + movement[0], *current*[1] + movement[1])

        # Check if the neighbor is within the grid and is passable

        if (

            0 <= neighbor[0] < len(*grid*) and

            0 <= neighbor[1] < len(*grid*[0]) and

*grid*[neighbor[0]][neighbor[1]] == 0

        ):

            neighbors.append(neighbor)

    return neighbors

def a\_star\_with\_simulated\_annealing(*grid*, *start*, *goal*, *initial\_temperature*=1000, *cooling\_rate*=0.99):

    """

    A\* algorithm with Simulated Annealing.

    Parameters:

    - grid: 2D grid representing the environment.

    - start: Starting node (x, y).

    - goal: Goal node (x, y).

    - initial\_temperature: Initial temperature for Simulated Annealing.

    - cooling\_rate: Rate at which the temperature decreases.

    Returns:

    - Path from start to goal if a path is found, None otherwise.

    """

    open\_set = [(0, *start*)]  # Priority queue to store nodes based on total cost

    closed\_set = set()  # Set to keep track of visited nodes

    g\_score = {*start*: 0}  # Dictionary to store the cost to reach each node

    f\_score = {*start*: heuristic\_cost\_estimate(*start*, *goal*)}  # Dictionary to store total estimated cost for each node

    temperature = *initial\_temperature*

    while open\_set:

        current\_f\_score, current\_node = heapq.heappop(open\_set)

        if current\_node == *goal*:

            # Reconstruct the path if the goal is reached

            path = []

            while current\_node in g\_score:

                path.append(current\_node)

                current\_node = g\_score[current\_node]

            return path[::-1]

        closed\_set.add(current\_node)

        neighbors = hill\_climbing(*grid*, current\_node, *goal*)

        for neighbor in neighbors:

            if neighbor not in closed\_set:

                tentative\_g\_score = g\_score[current\_node] + 1

                # Use Simulated Annealing to decide whether to accept the move

                if simulated\_annealing(current\_node, neighbor, temperature):

                    if neighbor not in open\_set or tentative\_g\_score < g\_score.get(neighbor, 0):

                        g\_score[neighbor] = tentative\_g\_score

                        f\_score[neighbor] = tentative\_g\_score + heuristic\_cost\_estimate(neighbor, *goal*)

                        heapq.heappush(open\_set, (f\_score[neighbor], neighbor))

        temperature \*= *cooling\_rate*  # Decrease the temperature

    return None  # No path found

# Example usage:

grid\_example = [

    [0, 0, 0, 0, 0],

    [0, 1, 1, 0, 0],

    [0, 0, 0, 0, 0],

    [0, 0, 1, 1, 1],

    [0, 0, 0, 0, 0],

]

start\_point = (0, 0)

goal\_point = (4, 4)

path\_result = a\_star\_with\_simulated\_annealing(grid\_example, start\_point, goal\_point)

print("Path:", path\_result)