**A\* Search Algorithm**

**Aim**

implement the A\* search algorithm using Python To to find the shortest path from a start node to a goal node in a weighted graph using heuristics.

**Procedure**

1. Define the graph with nodes, edges (costs), and heuristic values.
2. Initialize the open list with the start node and an empty closed list.
3. Loop until the goal is found or the open list is empty:
4. Select the node with the lowest f = g + h from the open list.
5. Move it to the closed list.
6. For each neighbor:
   1. If it's not in closed list and has a better path, update g, h, and f.
   2. Add or update it in the open list.
7. Reconstruct the path from start to goal using parent pointers.

**Program**

import heapq

def a\_star(grid, heuristics, start, goal):

rows, cols = len(grid), len(grid[0])

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

open\_list = [(heuristics[start[0]][start[1]], 0, start, [start])]

visited = set()

while open\_list:

f, g, curr, path = heapq.heappop(open\_list)

if curr == goal:

return path, g

if curr in visited:

continue

visited.add(curr)

for dx, dy in dirs:

x, y = curr[0] + dx, curr[1] + dy

if 0 <= x < rows and 0 <= y < cols and grid[x][y] == 0:

if (x, y) not in visited:

new\_g = g + 1

heapq.heappush(open\_list, (new\_g + heuristics[x][y], new\_g, (x, y), path + [(x, y)]))

return None, float('inf')

r, c = map(int, input("Enter rows and columns: ").split())

print("Enter grid (0=free, 1=blocked):")

grid = [list(map(int, input().split())) for \_ in range(r)]

print("Enter heuristic values:")

heuristics = [list(map(int, input().split())) for \_ in range(r)]

start = tuple(map(int, input("Enter start (row col): ").split()))

goal = tuple(map(int, input("Enter goal (row col): ").split()))

path, cost = a\_star(grid, heuristics, start, goal)

print("Path:", path if path else "No path")

print("Cost:", cost if path else "∞")

**Input**

Enter rows and columns: 5 5

Enter grid (0=free, 1=blocked):

0 0 0 0 0

0 1 1 1 0

0 1 0 0 0

0 1 0 1 0

0 0 0 0 0

Enter heuristic values:

4 3 2 1 0

3 2 1 1 1

2 1 0 2 3

3 2 1 3 4

4 3 2 3 4

Enter start (row col): 0 0

Enter goal (row col): 4 4

**Output**

Path: [(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (3, 3), (4, 3), (4, 4)]

Cost: 8

**Result**

The A\* algorithm successfully calculates the shortest path from the start to the goal, considering both the actual cost and heuristic estimates. If no path exists, it outputs "No path" with an infinite cost.