PROGRAM TITLE 10

A* ALGORITHM

AIM:

To Write the python program to implement A* algorithm

PROCEDURE:

1. Initialize Open and Closed Sets:
    - Create two sets: Open and Closed.
    - The Open set contains nodes to be evaluated, initially containing the start node.
    - The Closed set is empty.
2. Initialize Costs:
    - Set the cost-so-far (g) of the start node to 0.
    - Calculate the estimated remaining cost-to-goal (h) for the start node.
3. Main Loop:
    - While the Open set is not empty:
        - Select the node with the lowest total cost (f = g + h) from the Open set.
        - Move the selected node from the Open set to the Closed set.
        - If the selected node is the goal, the path is found.
4. Explore Neighbors:
    - For each neighbor of the selected node:
        - If the neighbor is in the Closed set, skip it.
        - Calculate the tentative cost-so-far for the neighbor.
        - If the neighbor is not in the Open set or the new cost is lower:

        - Update the neighbor's cost-so-far and estimated remaining cost-to-goal.
        - Add the neighbor to the Open set.

5. Termination:
    - If the Open set is empty and the goal has not been reached, there is no path.
6. Path Reconstruction:
    - If the goal is reached, reconstruct the path from the start to the goal using the parent pointers.

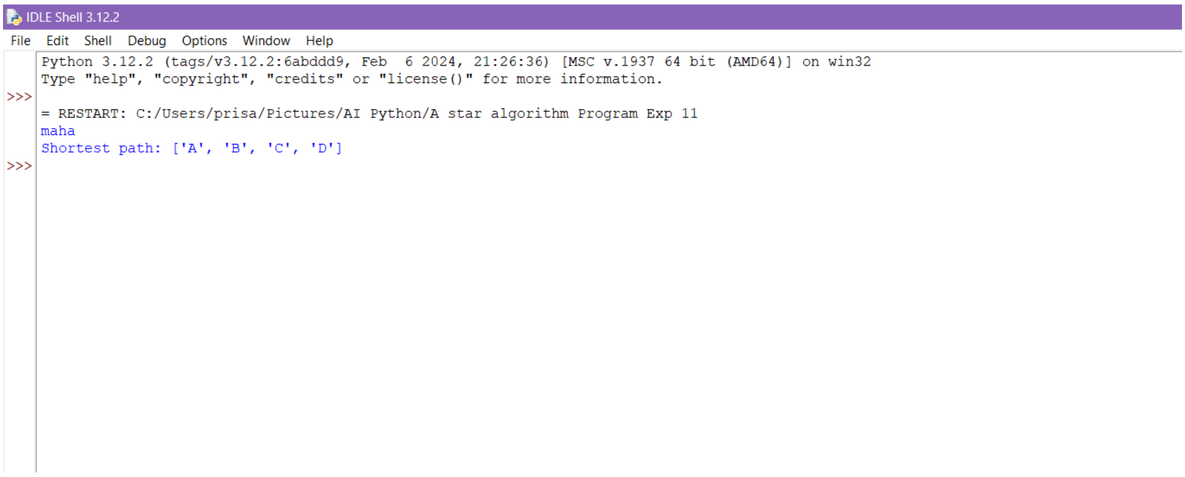
CODING:

```
import heapq


class Node:     def __init__(self, x, y,

obstacle=False):

        self.x = x          self.y =

y          self.obstacle =

obstacle          self.g =

float('inf')          self.h = 0

        self.f = 0

self.parent = None


    def __lt__(self, other):

return self.f < other.f


def calculate_heuristic(current, goal):


    return abs(current.x - goal.x) + abs(current.y - goal.y)
```

```
def get_neighbors(grid, node):
    neighbors = []     rows, cols = len(grid),
len(grid[0])     directions = [(1, 0), (-1, 0),
(0, 1), (0, -1)]

    for dx, dy in directions:
        x, y = node.x + dx, node.y + dy         if 0 <= x < rows and 0
<= y < cols and not grid[x][y].obstacle:
            neighbors.append(grid[x][y])

    return neighbors

def astar(grid, start, goal):     open_set =
[]     heapq.heappush(open_set, start)
start.g = 0     start.h =
calculate_heuristic(start, goal)     start.f =
start.g + start.h

    while open_set:
        current = heapq.heappop(open_set)

        if current == goal:
            path = []
while current:
                path.append((current.x, current.y))
current = current.parent             return
path[::-1]
```

```
        for neighbor in get_neighbors(grid, current):
            tentative_g = current.g + 1              if tentative_g
< neighbor.g:                neighbor.parent = current
neighbor.g = tentative_g                neighbor.h =
calculate_heuristic(neighbor, goal)                neighbor.f =
neighbor.g + neighbor.h                if neighbor not in
open_set:
                    heapq.heappush(open_set, neighbor)

    return None

if __name__ == "__main__":

    grid = [[Node(x, y, obstacle=False) for y in range(5)] for x in range(5)]
grid[1][2].obstacle = True     grid[2][2].obstacle = True
grid[3][2].obstacle = True


    start_node = grid[0][0]
goal_node = grid[4][4]     path =
astar(grid, start_node, goal_node)


    if path:
        print("Path found:")
for x, y in path:
            print(f"({x}, {y})", end=" ")
else:
    print("No path found.")
```

OUTPUT:



```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/prisa/Pictures/AI Python/A star algorithm Program Exp 11
maha
Shortest path: ['A', 'B', 'C', 'D']
>>>
```

RESULT:

Hence the program been successfully executed and verified.