IMPLEMENTATION OF A* ALGORITHM

SOURCE CODE:

```
import heapq


class Node:
    def __init__(self, state, parent, g, h):
        self.state = state
        self.parent = parent
        self.g = g
        self.h = h
        self.f = g + h

    def __lt__(self, other):
        return self.f < other.f


def a_star_search(start, goal, heuristic, neighbors):
    open_list = []
    closed_list = set()

    start_node = Node(start, None, 0, heuristic(start, goal))
    heapq.heappush(open_list, start_node)

    while open_list:
        current_node = heapq.heappop(open_list)

        if current_node.state == goal:
            # Reconstruct the path from start to goal
            path = []
            while current_node:
```

```
            path.append(current_node.state)
            current_node = current_node.parent
        return path[::-1]

    closed_list.add(current_node.state)

    for neighbor, cost in neighbors(current_node.state):
        if neighbor in closed_list:
            continue

        g = current_node.g + cost
        h = heuristic(neighbor, goal)
        neighbor_node = Node(neighbor, current_node, g, h)

        if not any(open_node.state == neighbor and open_node.f <= neighbor_node.f for open_node
in open_list):
            heapq.heappush(open_list, neighbor_node)

  return None  # If no path is found

# Example of usage

def heuristic(state, goal):
  return abs(state[0] - goal[0]) + abs(state[1] - goal[1])

def neighbors(state):
  x, y = state
  return [
    ((x + 1, y), 1),  # Right
    ((x - 1, y), 1),  # Left
    ((x, y + 1), 1),  # Up
```

```
        ((x, y - 1), 1)   # Down
    ]

start = (0, 0)
goal = (3, 3)

path = a_star_search(start, goal, heuristic, neighbors)

print("Path from start to goal:", path)
```

OUTPUT:

For start = (0, 0) and goal = (3, 3):

Path from start to goal: [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), (3, 3)]