AO\*SEARCH ALGORITHM

# PROGRAM:

import heapq  
  
class Node:  
    def \_\_init\_\_(self, state, g\_value, h\_value, parent=None):  
        self.state = state  
        self.g\_value = g\_value  
        self.h\_value = h\_value  
        self.parent = parent  
  
    def f\_value(self):  
        return self.g\_value + self.h\_value  
  
def ao\_star\_search(initial\_state, is\_goal, successors, heuristic):  
    open\_list = [Node(initial\_state, 0, heuristic(initial\_state), None)]  
    while open\_list:  
        open\_list.sort(key=lambda node: node.f\_value())  
        current\_node = open\_list.pop(0)  
  
        if is\_goal(current\_node.state):  
            path = []  
            while current\_node:  
                path.append(current\_node.state)  
                current\_node = current\_node.parent  
            return list(reversed(path))  
  
        best\_g = float('inf')  
        for child\_state in successors(current\_node.state):  
            g\_value = current\_node.g\_value + 1  
            h\_value = heuristic(child\_state)  
            child\_node = Node(child\_state, g\_value, h\_value, current\_node)  
  
            if child\_node.f\_value() < best\_g:  
                best\_g = child\_node.f\_value()  
                open\_list.append(child\_node)  
  
    return None  
  
def is\_goal(state):  
    return state == (4, 4)  
  
def successors(state):  
    x, y = state  
    return [(x + 1, y), (x, y + 1)]  
  
def heuristic(state):  
    x, y = state  
    return abs(4 - x) + abs(4 - y)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    initial\_state = (0, 0)  
    path = ao\_star\_search(initial\_state, is\_goal, successors, heuristic)  
    if path:  
        print("Path found:", path)  
    else:  
        print("No path found")

OUTPUT:

Path found: [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)]  
  
=== Code Execution Successful ===

|  |  |
| --- | --- |
|  |  |