|  |  |
| --- | --- |
| **EX.N0 : 05** | **AO\* SEARCH ALGORITHM** |
| **DATE : 27.03.2024** |

**AIM:**

To implement the AO\* Search Algorithm using python

**ALGORITHM:**

Step 1: Start

Step 2: Initialize an open list with the start node and an empty closed list. Step 3: While the open list is not empty:

1. Select the node with the lowest estimated total cost (f-value).
2. If the selected node is the goal, return the solution.
3. Generate successor nodes, calculate their costs, and add them to the open list if they are better or not in the closed list.
4. Add the selected node to the closed list.

Step 4: If the open list becomes empty, and no solution is found, the problem has no solution.

Step 5: Stop

**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)]

    closed\_set = set()

    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))

        closed\_set.add(current\_node.state)

        for child\_state in successors(current\_node.state):

            if child\_state in closed\_set:

                continue

            g\_value = current\_node.g\_value + 1

            h\_value = heuristic(child\_state)

            child\_node = Node(child\_state, g\_value, h\_value, current\_node)

            for i, node in enumerate(open\_list):

                if node.state == child\_state:

                    if node.g\_value > g\_value:

                        open\_list.pop(i)

                        break

                elif node.g\_value > g\_value:

                    open\_list.insert(i, child\_node)

                    break

            else:

                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)]

**RESULT:**

Thus implemention of the AO\* Search Algorithm using python is executed successfully