Fr. Conceicao Rodrigues College of Engineering Fr. Agnel Ashram, Bandstand, Bandra (W), Mumbai - 400050

**Department of Computer Engineering**

**Academic Term II: 23-24**

**Class: T.E (Computer), Sem – VI Subject Name: Artificial Intelligence Student Name: Roll No: 9538**

| **Practical No:** | **5** |
| --- | --- |
| **Title:** | Eight puzzle game solution by A\* algorithm |
| **Date of Performance:** |  |
| **Date of Submission:** |  |

**Rubrics for Evaluation:**

| **Sr.**  **No** | **Performance Indicator** | **Excellent** | **Good** | **Below**  **Average** | **Marks** |
| --- | --- | --- | --- | --- | --- |
| 1 | On time Completion &  Submission (01) | 01 (On  Time) | NA | 00 (Not on  Time) |  |
| 2 | Logic/Algorithm Complexity analysis (03) | 03(Correct ) | 02(Partial) | 01 (Tried) |  |
| 3 | Coding Standards (03):  Comments/indention/Naming conventions  Test Cases /Output | 03(All  used) | 02 (Partial) | 01 (rarely  followed) |  |
| 4 | Post Lab Assignment (03) | 03(done  well) | 2 (Partially  Correct) | 1(submitte  d) |  |
| **Total** | | | | |  |

**Signature of the Teacher:**

import heapq

import numpy as np

class Node:

def \_\_init\_\_(self, state, parent=None, g=0, h=0):

self.state = state

self.parent = parent

self.g = g # Cost from the start node to the current node

self.h = h # Heuristic cost from the current node to the goal node

def \_\_lt\_\_(self, other):

# Comparison method for priority queue

return (self.g + self.h) < (other.g + other.h)

def manhattan\_distance(state, goal):

# Calculate Manhattan Distance heuristic

distance = 0

for i in range(3):

for j in range(3):

if state[i, j] != 0:

goal\_position = np.where(goal == state[i, j])

distance += abs(i - goal\_position[0]) + abs(j - goal\_position[1])

return distance

def get\_successors(node):

# Generate successor nodes by moving tiles

successors = []

zero\_position = np.where(node.state == 0)

for move in [(0, 1), (1, 0), (0, -1), (-1, 0)]:

new\_position = (zero\_position[0] + move[0], zero\_position[1] + move[1])

if 0 <= new\_position[0] < 3 and 0 <= new\_position[1] < 3:

new\_state = np.copy(node.state)

new\_state[zero\_position], new\_state[new\_position] = new\_state[new\_position], new\_state[zero\_position]

successors.append(Node(new\_state, node))

return successors

def a\_star(initial\_state, goal\_state):

open\_list = [Node(initial\_state, None, 0, manhattan\_distance(initial\_state, goal\_state))]

closed\_set = set()

while open\_list:

current\_node = heapq.heappop(open\_list)

if np.array\_equal(current\_node.state, goal\_state):

# Reconstruct the solution path

solution\_path = []

while current\_node:

solution\_path.append(current\_node.state)

current\_node = current\_node.parent

return solution\_path[::-1]

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

for successor in get\_successors(current\_node):

if tuple(successor.state.flatten()) not in closed\_set:

successor.g = current\_node.g + 1

successor.h = manhattan\_distance(successor.state, goal\_state)

heapq.heappush(open\_list, successor)

return None

def print\_solution\_path(solution\_path):

for i, state in enumerate(solution\_path):

print(f"Step {i + 1}:")

print(state)

print()

if \_\_name\_\_ == "\_\_main\_\_":

initial\_state = np.array([[1, 2, 3], [4, 0, 5], [6, 7, 8]])

goal\_state = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])

solution\_path = a\_star(initial\_state, goal\_state)

if solution\_path:

print("Solution Path:")

print\_solution\_path(solution\_path)

else:

print("No solution found.")

Output:

Solution Path:

Step 1:

[[1 2 3]

[4 0 5]

[6 7 8]]

Step 2:

[[1 2 3]

[0 4 5]

[6 7 8]]

Step 3:

[[0 2 3]

[1 4 5]

[6 7 8]]

Step 4:

[[2 0 3]

[1 4 5]

[6 7 8]]

Step 5:

[[2 3 0]

[1 4 5]

[6 7 8]]

Step 6:

[[2 3 5]

[1 4 0]

[6 7 8]]

Step 7:

[[2 3 5]

[1 0 4]

[6 7 8]]

Step 8:

[[2 0 5]

[1 3 4]

[6 7 8]]

Step 9:

[[0 2 5]

[1 3 4]

[6 7 8]]

Step 10:

[[1 2 5]

[0 3 4]

[6 7 8]]

Step 11:

[[1 2 5]

[3 0 4]

[6 7 8]]

Step 12:

[[1 2 5]

[3 4 0]

[6 7 8]]

Step 13:

[[1 2 0]

[3 4 5]

[6 7 8]]

Step 14:

[[1 0 2]

[3 4 5]

[6 7 8]]

Step 15:

[[0 1 2]

[3 4 5]

[6 7 8]]

>