## Experiment:3

**1.Programs on Problem Sloving**

a).Implement of A\* algorithm

Aim: To Implement program to solve A\* algorithm

#### Program:

import heapq

def a\_star(graph, start, goal, heuristic):

open\_set = [(0 + heuristic(start, goal), 0, start, [])]

closed\_set = set()

while open\_set:

f\_score, g\_score, current, path = heapq.heappop(open\_set)

if current == goal:

return path + [current], g\_score

if current in closed\_set:

continue

closed\_set.add(current)

for neighbor, cost in graph.get(current, []):

new\_g\_score = g\_score + cost

new\_f\_score = new\_g\_score + heuristic(neighbor, goal)

add\_to\_open = True

for i, (f, g, n, p) in enumerate(open\_set):

if n == neighbor and g <= new\_g\_score:

add\_to\_open = False

break

if add\_to\_open:

heapq.heappush(open\_set, (new\_f\_score, new\_g\_score, neighbor, path + [current]))

return None

graph = {

'A': [('B', 2), ('C', 4)],

'B': [('C', 1), ('D', 7)],

'C': [('D', 3), ('E', 5)],

'D': [('E', 1),('Z', 8)],

'E': [('Z', 7)],

'Z': []

}

def heuristic(node, goal):

heuristic\_values = {

'A': 10,

'B': 8,

'C': 5,

'D': 3,

'E': 2,

'Z': 0

}

return heuristic\_values.get(node, float('inf'))

start\_node = 'A'

goal\_node = 'Z'

result = a\_star(graph, start\_node, goal\_node, heuristic)

if result:

path, cost = result

print(f"Path: {path}")

print(f"Cost: {cost}")

else:

print("No path found.")

#### Output:

Path: ['A', 'C', 'D', 'Z']

Cost: 15

Result: Thus the program has been implemented to find a\*algorithm