EXP NO : 3

DATE ·

WATERJUG .

AIM:

To find the shortest path between a start node & goal node in a graph or grid, exploring only the most promising paths.

CODE

```
from Queue import priority Queue.
  def a - start- search (graph, start, goal):
    open- list = priority Queue()
   open - list . put ((0, start))
     g- score = { start : 0 }.
     f - scroe = { start : heuristic (start, goal)
        came - from [start] = none.
  while not open- list . empty():
     current = open - list . get () [1].
     if current = = goal:
        return
       reconstruct -path (come-from, current)
  fror neighboar , cost in graph [current]:
  tentative -g- score = g score [current]+ cos
  if neighborar not in g- score or tentative
                                      -g- score
        < g- scrore [neighbour]:
```

```
come -from [neighbour] = current.
g- score [neighbour] = tentative -g -score.
f- score[. neighbour] = tentative -g- score
                        + heuristic (neighbor, goal)
open _list . put ((f- score [neighbour],
                   neighbour))
return home.

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

def reconstruct -path (come-from , current):
    total path = [current]
    while current in come- from and come-from
               [current] is not none :
        current = come- from [current]
        total- path. append (current).
    total -path . reverse ()
    return total- path.
```

```
graph = { (0, 0): [((0, 1), 1) ((1, 0), 1)]
    (0, 1): [((0, 0), 1), ((1, 1), 1)]
   (1, 0): [((0, 0) 1), ((1, 1), 1)]
  (1, 1): [((1, 0), 1), ((0, 1), 1), ((2, 2), 1]
      (2, 2). [ ]. }
  start = (0, 0)
   goal = (2, 2)
  path = a_star_search (graph, start, goal)
 print(f "path found : { path }")
```

Result

Solution [1, 0]

Output: Thus the water jug program is successfully executed and output is verified.