Ex. No : 3.

Date :

WATER JUG USING DFS.

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 - score = { start : heuristic (start,
                                        goal) }

        came - from [start] = none.
   while not open - list .empty () :
        current = open - list . get () [1].
        if current == goal :
        return
        reconstruct = path (came - from,
                                current)

        for neighbour, cost in graph [current]:
         tentative - g - score = g score [current]
                                  + cost.
   if neighbour not in g - score or tentative
        - g - score < g - score [neighbour]:
```

```
come - from [neighbour ] - current · g - score
[neighbour] = tentative - g - score.
f - score [neighbour] = tentative - g - score +
                         heuristic (neighbor, goal)
 open - list - push ((f - score [neighbour]. neighbour);

return none.
def heuristic (node, goal) :
    return abs (node [0] - goal [0] - 1 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)
 printf(" path found : {path} ").
```

OUTPUT ::

```
Path found : (4,2).
```

Result :

The program is successfully executed & the output is verified.