PROGRAM

```
import heapq
def astar(start, goal, grid):
    def h(a, b): return abs(a[0]-b[0])+abs(a[1]-b[1])
    def neighbours(p): return [(p[0]+dx, p[1]+dy) for dx,dy
                               in [(0,1),(1,0),(0,-1),(-1,0)] if
open_set, came_from, cost_so_far = [(h(start,goal),0,start)], {}, {start:0}
                                   0<=p[0]+dx<len(grid) and
                                   0<=p[1]+dy<len(grid[0]) and
                                   grid[p[0]+dx][p[1]+dy]==0)
while open_set:
    _, curr_cost, curr = heapq.heappop(open_set)
    if curr==goal:
        path, p = [], curr
        while p in came_from:
            path.append(p)
            p = came_from[p]
        return path[::-1]+[goal]
    for next in neighbours(curr):
        new_cost = curr_cost+1
        if next not in cost_so_far or new_cost<cost_so_far[next]:
            cost_so_far[next] = new_cost
            heapq.heappush(open_set, (new_cost+h(goal,next),
                                      new_cost,next))
return None
```

GOOGLE COLLAB

```
[(0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (3, 3), (3, 3)]
```

OUTPUT

[(0,1), (0,2), (0,3), (1,3), (2,3) (3,3), (3,3)]

10/09/2024

EXP NO: 4

EXP NAME: A* ALGORITHM

AIM: To implement A* in python

PROGRAM:

1) Initialise starting node with priority que containing start node cost 0. Also two dictionaries came_from and cost_so_far

2) Pick Lowest cost Node: Pop the node with lowest cost from queue.

3) If current node is goal reconstruct pathusing back tracking.

4) Calculate new cost for neighbouring nodes. If has not been visited update cost.

5) Continue the process.

RESULT

Thus A* has been implement in Python