# Implementation of A* search Algorithm.

Aim: To implement A* search Algorithm.

Case scenario:
a delivery robot in a warehouse needs to be find the shortest path from the entrance to a specific package location. The warehouse is represented by a grid with some obstacles. Implement the A* search algorithm to help the robot navigate efficiently.

Procedure:
Define the Node class with attributes, position, parent, cost(g), heuristic(h) and total cost (f = g + h)

Implement the heuristic function using the Manhattan distance formula.

Initialize A search* with:

(i) open_list
(ii) closed_list

While list not empty,
- Extract the node with lowest f-value
- If the goal is reached, trace back the path and return it.

Program

import heapq

define grid and movements.

class node:

def __init__(self, position, parent = None, g=0, h=0):

self.position = position # (row, col)

self.parent = parent # Parent node

self.g = g

self.h = h

self.f = g + h.

def __lt__(self, other):

return abs(a[0]-b[0]) + abs(a[1]-b[1])

def heuristic(a,b)

return abs(a[0]-b[0]) + abs(a[1]-b[1])

def a_star(grid, start, goal):

rows, cols = len(grid), len(grid[0])

open_list = []

heapq.heappush(open_list, Node(start, None, 0,

heuristic(start, goal)))

closed_set = set()

while open_list:

current_node = heapq.heappop(open_list)

if current_node.position == goal:

path = []

```
while current node
    path.append(current node.position)
    current_node = current_node.parent
    return path[::-1]


closed_set.add(current_node.position)
for de, de in [(-1,0),(1,0),(0,-1),(0,1)]
    new_pos = (current_node.position[0] +
        dr, current_node.position[1] + de)
    If (0<=new_pos[0]<rows and 0<=
        new_pos[1] not in closed_set):
    new_node = Node(new_pos, current_node,
        current_node.g + 1, heuristic(new_pos,
    goal))
```

Example grid 0: free space 1: obstacle

```
warehouse_grid = [
  [0, 0, 0, 0, 1]
  [1, 1, 0, 1, 0]
  [0 0 0 0 0]
  [0 1 1 1 0]
  [0 0 0 0 0]
]
```

Output

Optimal path [(0,0), (0,1), (0,2), (1,2), (2,2)
(2,3) (2,4) (3,4) (4,4)]

- add the current node to closed_set.
- generate new valid moves (up, down, left, right) ensuring that they are within bounds
- calculate new cost g, heuristic h, total f.
- add new nodes to open list for further exploration.

return the optimal path. if found else return None if no path exists.