Ex.no: 4 A* SEARCH ALGORITHM.

AIM

To implement A* search algorithm technique to find paths and traverse graphs.

ALGORITHM:

Step 1: Initialize the open list.

Step 2: Initialize closed list, put starting node on open list.

Step 3: while open list not empty

a). find node with the least f on the open list, call it 'q'.

b) pop q off the open list

c). generate q's 8 successors and set their parents to q.

d). for each successor.

i). if successor is goal, stop search

ii) else compute both g & h for successor.

iii). if a node with same position as successor is in the OPEN list skip successor.

iv) if a node with same position as successor is in CLOSED list which has a lower f than successor, skip successor else add node to open list.

end (for loop).

step 4:- Push q on the closed list end (while loop).

PROGRAM:-

```
import math.
import heapq.
class cell:
    def __init__(self):
        self.parent_i = 0
        self.parent_j = 0
        self.f = float('inf')
        self.g = float('inf')
        self.h = 0.

row = 9.
col = 10.
def is_valid (row, col):
    return (row >= 0) and (row < row) and.
    (col >= 0) and (col < col).
def is_unblocked (grid, row, col):
    return grid [row] [col] == 1
def is_destination (row, col, dest):
    return row == dest [0] and col ==
                                    dest [1]
```

```
def calculate_h_value(row, col, dest):
    return ((row - dest[0]) ** 2 + (col - dest[1]
        ) ** 2) ** 0.5

def trace_path(cell_details, dest):
    print("The path is ")
    path = []
    row = dest[0]
    col = dest[1]
    while not (cell_details[row][col].
    parent_i == row and cell_details[row]
    [col].parent_j == col):
        path.append((row, col))
        temp_row = cell_details[row][col].
                                parent_i
        temp_col = cell_details[row][col].
                                parent_j

        row = temp_row
        col = temp_col
        path.append((row, col))
        path.reverse()
        for i in path:
            print("->", i, end=" ")
        print()
```

```
def a_star_search(grid, src, dest):
    if not is_valid(src[0], src[1]) or not
is_valid(dest[0], dest[1]):
        print("source/destination invalid")
        return

    i = src[0]
    j = src[1]
    cell_details[i][j].f = 0
    cell_details[i][j].g = 0
    cell_details[i][j].h = 0
    open_list = []
    heapq.heappush(open_list, (0.0, i, j))
    found_dest = False
    while len(open_list) > 0:
        p = heapq.heappop(open_list)
        i = p[1]
        j = p[2]
        closed_list[i][j] = True
        directions = [(0,1), (0,-1), (1,0),
                      (-1,0), (1,1), (1,-1), (-1,1), (-1,-1)]

def main():
    grid = [
        [1, 0, 1, 1, 1, 0, 1, 1, 1],
        [1, 0, 1, 0, 1, 1, 1, 0, 1],
        [1, 1, 1, 0, 1, 1, 1, 0, 1],
        [1, 1, 1, 0, 1, 1, 0, 1, 1],
        [0, 0, 0, 1, 1, 0, 1, 0, 1],
```

```
    [1, 1, 1, 0, 0, 0, 1, 0],
    [1, 1, 1, 0, 0, 0, 1, 0, 0, 1]]
src = [8, 0]
dest = [0, 0]
a_star_search(grid, src, dest)
if __name__ == "__main__":
    main()
```

OUTPUT::

The destination cell is found

The Path is

$\rightarrow(8,0) \rightarrow (7,0) \rightarrow (6,0) \rightarrow (5,0) \rightarrow (4,1) \rightarrow$
$(3,2) \rightarrow (2,1) \rightarrow (1,0) \rightarrow (0,0)$.

Result:

The program is successfully executed & the output is verified.