

**EX.NO:5**

**DATE:4/9/2024**

**Reg.no:220701037**

### **A\* SEARCH ALGORITHM**

A heuristic algorithm sacrifices optimality, with precision and accuracy for speed, to solve

problems faster and more efficiently. All graphs have different nodes or points which the algorithm has to take, to reach the final node. The paths between these nodes all have a numerical value, which is considered as the weight of the path. The total of all paths transverse gives you the cost of that route.

Initially, the Algorithm calculates the cost to all its immediate neighboring nodes,  $n$ , and chooses

the one incurring the least cost. This process repeats until no new nodes can be chosen and all

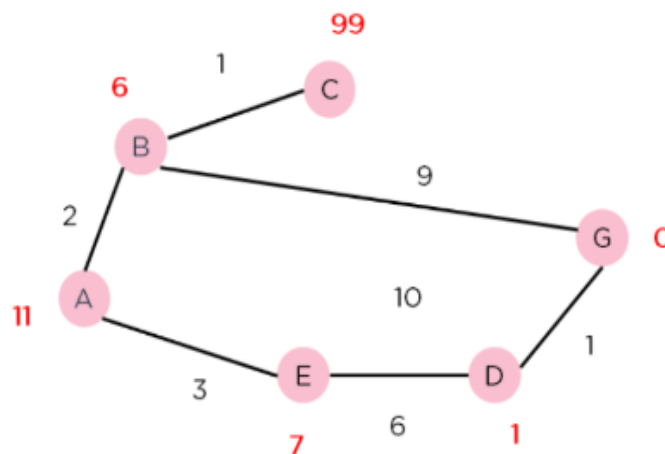
paths have been traversed. Then, you should consider the best path among them. If  $f(n)$  represents

the final cost, then it can be denoted as :

$f(n) = g(n) + h(n)$ , where :

$g(n)$  = cost of traversing from one node to another. This will vary from node to node

$h(n)$  = heuristic approximation of the node's value. This is not a real value but an approximation cost.



## CODE:

```
import heapq

def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def astar(grid, start, end):
    open_list = []
    heapq.heappush(open_list, (0 + heuristic(start, end), 0, start))
    g_cost = {start: 0}
    came_from = {}
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    while open_list:
        _, current_g, current = heapq.heappop(open_list)

        if current == end:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]

        for dx, dy in directions:
            neighbor = (current[0] + dx, current[1] + dy)
            if (0 <= neighbor[0] < len(grid) and 0 <= neighbor[1] <
len(grid[0]) and grid[neighbor[0]][neighbor[1]] == 0):
                tentative_g = current_g + 1
                if neighbor not in g_cost or tentative_g <
g_cost[neighbor]:
                    g_cost[neighbor] = tentative_g
                    f_cost = tentative_g + heuristic(neighbor, end)
                    heapq.heappush(open_list, (f_cost, tentative_g,
neighbor))

                    came_from[neighbor] = current

    return None

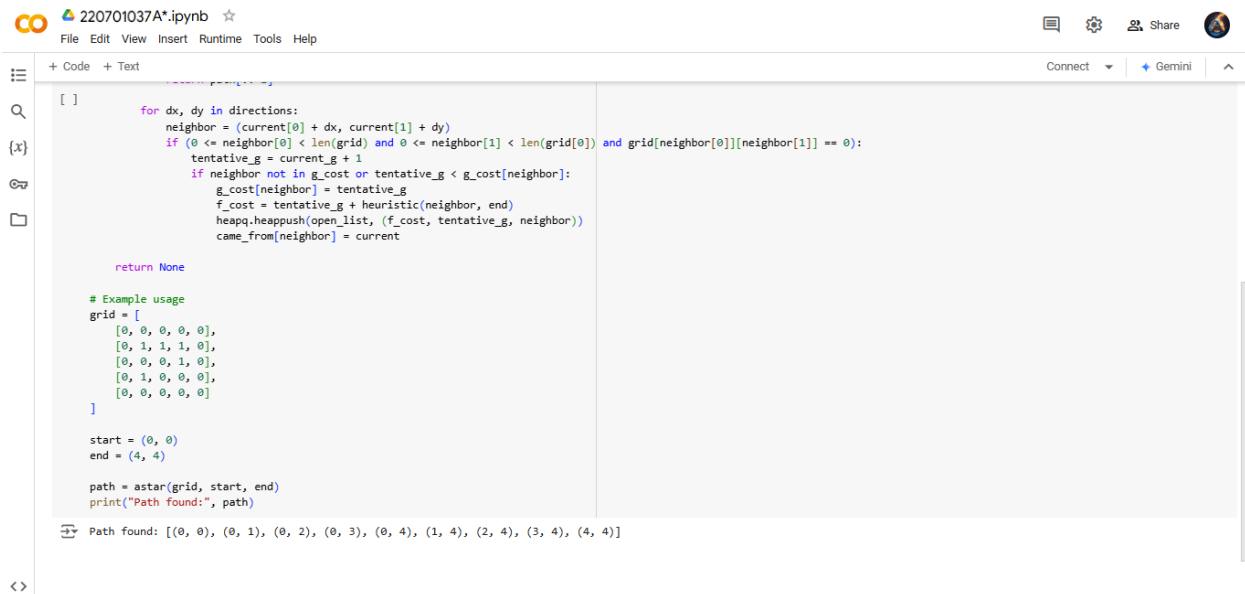
# Example usage
grid = [
    [0, 0, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 0, 0, 0],
```

```
[0, 0, 0, 0, 0]
]

start = (0, 0)
end = (4, 4)

path = astar(grid, start, end)
print("Path found:", path)
```

## OUTPUT:



220701037A\*.ipynb ☆

File Edit View Insert Runtime Tools Help

+ Code + Text

Connect Gemini

```
[ ]
    for dx, dy in directions:
        neighbor = (current[0] + dx, current[1] + dy)
        if (0 <= neighbor[0] < len(grid) and 0 <= neighbor[1] < len(grid[0]) and grid[neighbor[0]][neighbor[1]] == 0):
            tentative_g = current_g + 1
            if neighbor not in g_cost or tentative_g < g_cost[neighbor]:
                g_cost[neighbor] = tentative_g
                f_cost = tentative_g + heuristic(neighbor, end)
                heapq.heappush(open_list, (f_cost, tentative_g, neighbor))
                came_from[neighbor] = current

    return None

# Example usage
grid = [
    [0, 0, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0]
]

start = (0, 0)
end = (4, 4)

path = astar(grid, start, end)
print("Path found:", path)
```

Path found: [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4), (3, 4), (4, 4)]