EX.NO:5 **DATE:4/9/2024**

Reg.no:220701025

A* SEARCH ALGORITHM

AIM: To implement A* Algorithm in python

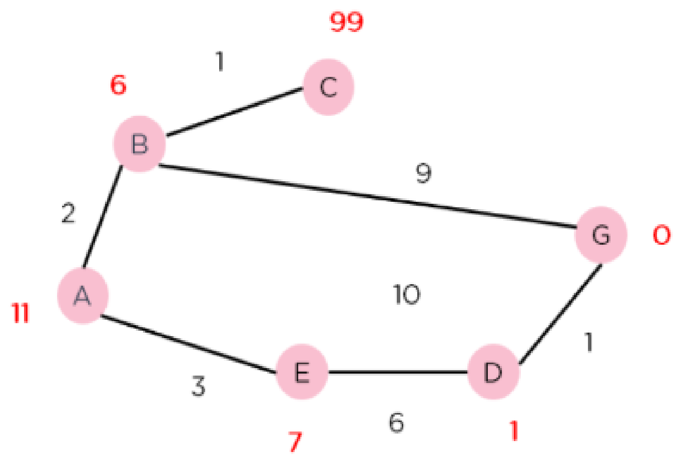
A heuristic algorithm sacrifices optimality, with precision and accuracy for speed, to solve problems faster and more efficiently. All graphs have different nodes or points which the algorithm has to take, to reach the final node. The paths between these nodes all have a numerical value, which is considered 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.



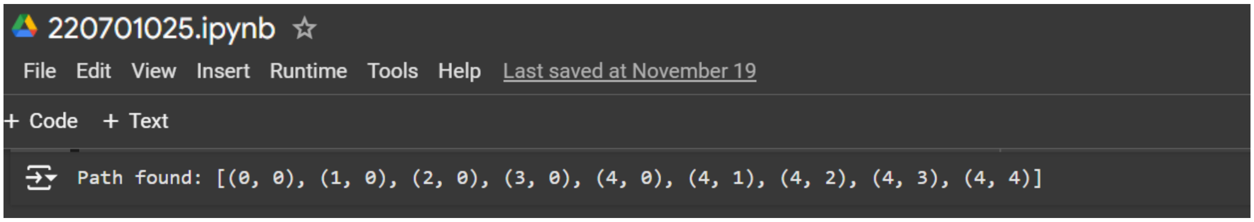
CODE:

```
import heappop, heappush


class Node:

    def _init_(self, position, parent=None):

        self.position = position

        self.parent = parent

        self.g = 0

        self.h = 0

        self.f = 0

    def _eq_(self, other):

        return self.position == other.position

    def _lt_(self, other):

        return self.f < other.f

def a_star(start, goal, grid):

        start_node = Node(start)

    goal_node = Node(goal)
```

```
        open_list = []

    closed_list = set()


       heappush(open_list, start_node)


        while open_list:

              current_node = heappop(open_list)

        closed_list.add(current_node.position)


             if current_node == goal_node:

            path = []

            while current_node:

                path.append(current_node.position)

                current_node = current_node.parent

            return path[::-1]  # Return reversed path


              neighbors = [(0, -1), (0, 1), (-1, 0), (1, 0)]

        for n in neighbors:

        neighbor_position = (current_node.position[0] + n[0],
current_node.position[1] + n[1])


      if 0 <= neighbor_position[0] < len(grid) and 0 <=
neighbor_position[1] < len(grid[0]) and
grid[neighbor_position[0]][neighbor_position[1]] == 0:

  neighbor_node = Node(neighbor_position, current_node)

 if neighbor_node.position in closed_list:

                    continue
```

```
                  neighbor_node.g = current_node.g + 1

                neighbor_node.h = abs(neighbor_node.position[0] -
goal_node.position[0]) + abs(neighbor_node.position[1] -
goal_node.position[1])

                neighbor_node.f = neighbor_node.g + neighbor_node.h




                if all(neighbor_node != open_node for open_node in
open_list):

                    heappush(open_list, neighbor_node)



    return None

grid = [

    [0, 1, 0, 0, 0],

    [0, 1, 0, 1, 0],

    [0, 0, 0, 1, 0],

    [0, 1, 1, 1, 0],

    [0, 0, 0, 0, 0]

]

start = (0, 0)

goal = (4, 4)

path = a_star(start, goal, grid)

print("Path found:", path)
```

OUTPUT:


220701025.ipynb
File Edit View Insert Runtime Tools Help Last saved at November 19
+ Code + Text
Path found: [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)]