

10. Write the python program to implement A* algorithm.

Program:

```
class Node():
    """A node class for A* Pathfinding"""

    def __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position

        self.g = 0
        self.h = 0
        self.f = 0

    def __eq__(self, other):
        return self.position == other.position


def astar(maze, start, end):
    """Returns a list of tuples as a path from the given start to the given end in the given maze"""

    start_node = Node(None, start)
    start_node.g = start_node.h = start_node.f = 0
    end_node = Node(None, end)
    end_node.g = end_node.h = end_node.f = 0

    open_list = []
    closed_list = []

    open_list.append(start_node)

    while len(open_list) > 0:

        current_node = open_list[0]
        current_index = 0
        for index, item in enumerate(open_list):
            if item.f < current_node.f:
                current_node = item
                current_index = index

        open_list.pop(current_index)
        closed_list.append(current_node)

        if current_node == end_node:
            path = []
```

```

current = current_node
while current is not None:
    path.append(current.position)
    current = current.parent
return path[::-1]

```

```

children = []
for new_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]: # Adjacent
squares

    node_position = (current_node.position[0] + new_position[0], current_node.position[1] +
new_position[1])

    if node_position[0] > (len(maze) - 1) or node_position[0] < 0 or node_position[1] >
(len(maze[len(maze)-1]) - 1) or node_position[1] < 0:
        continue

    if maze[node_position[0]][node_position[1]] != 0:
        continue

    new_node = Node(current_node, node_position)

    children.append(new_node)

for child in children:

    for closed_child in closed_list:
        if child == closed_child:
            continue

    child.g = current_node.g + 1
    child.h = ((child.position[0] - end_node.position[0]) ** 2) + ((child.position[1] -
end_node.position[1]) ** 2)
    child.f = child.g + child.h

    for open_node in open_list:
        if child == open_node and child.g > open_node.g:
            continue

    open_list.append(child)

```

```

def main():

```

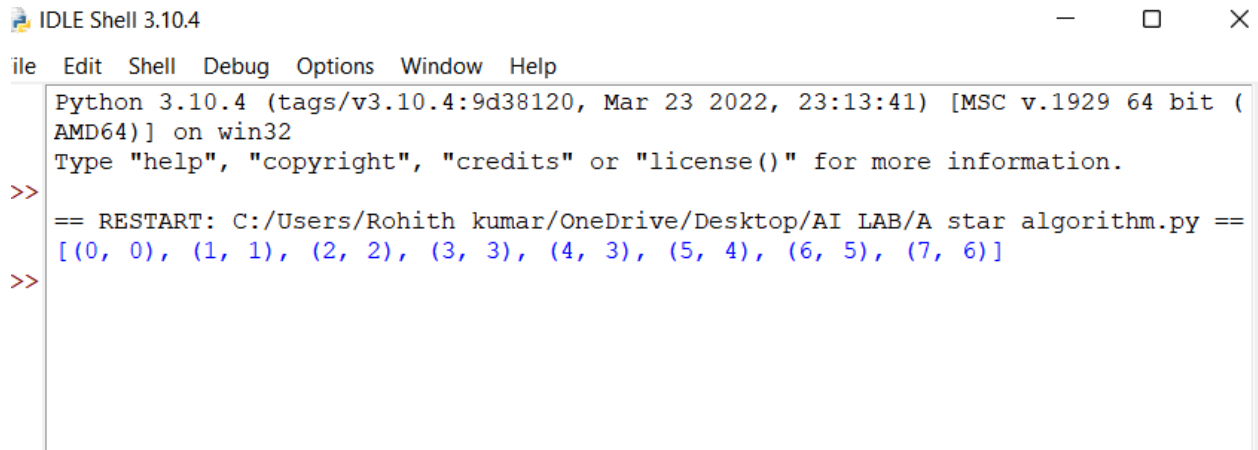
```
maze = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

```
start = (0, 0)
end = (7, 6)
```

```
path = astar(maze, start, end)
print(path)
```

```
if __name__ == '__main__':
    main()
```

OUTPUT:



```
IDLE Shell 3.10.4
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>> == RESTART: C:/Users/Rohith kumar/OneDrive/Desktop/AI LAB/A star algorithm.py ==
>> [(0, 0), (1, 1), (2, 2), (3, 3), (4, 3), (5, 4), (6, 5), (7, 6)]
```