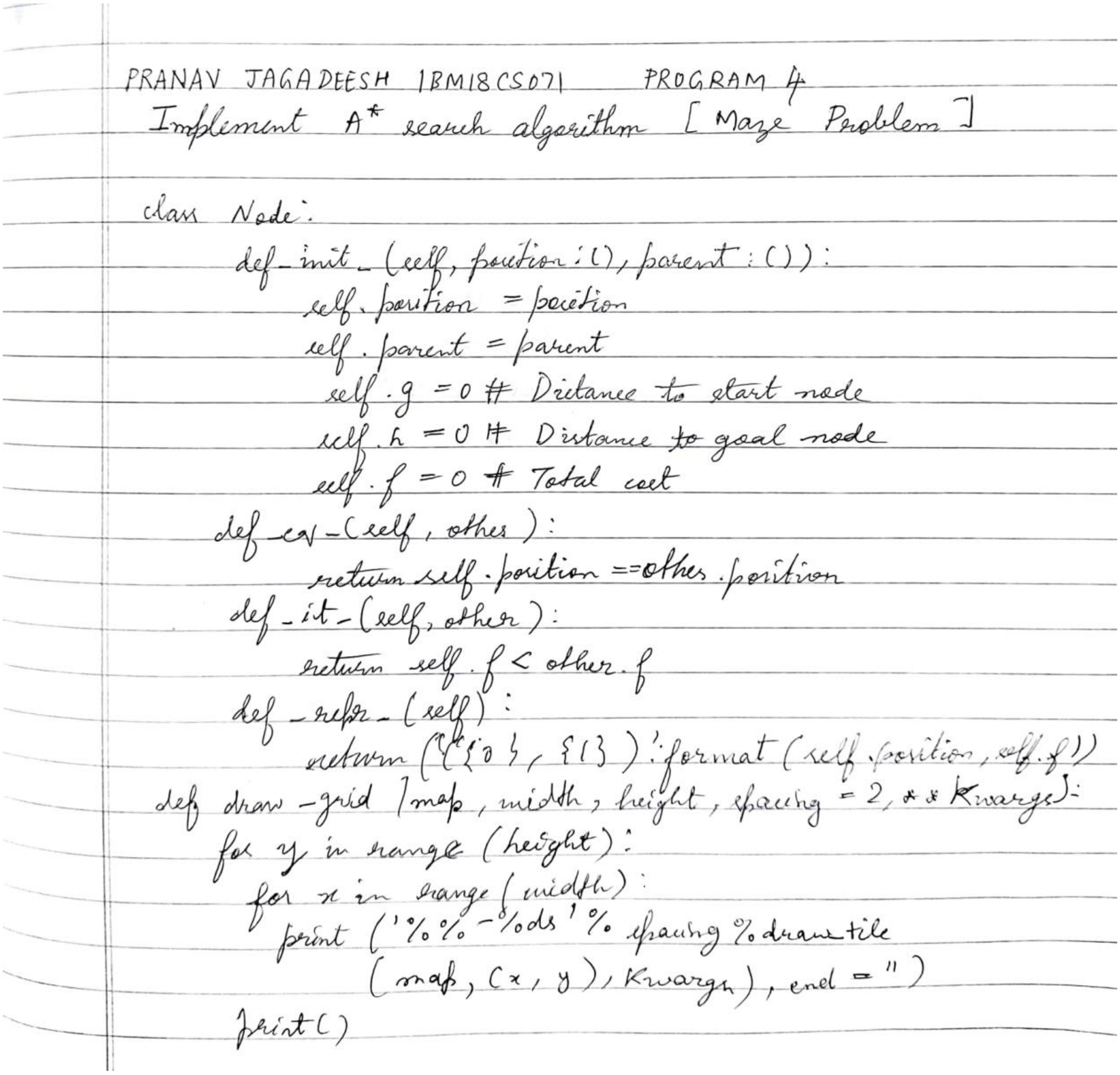
Program 4: Implement A* search algorithm.

Write_up:



PRANAV JAGADEESH 1BM18CS071 PROGRAM 4

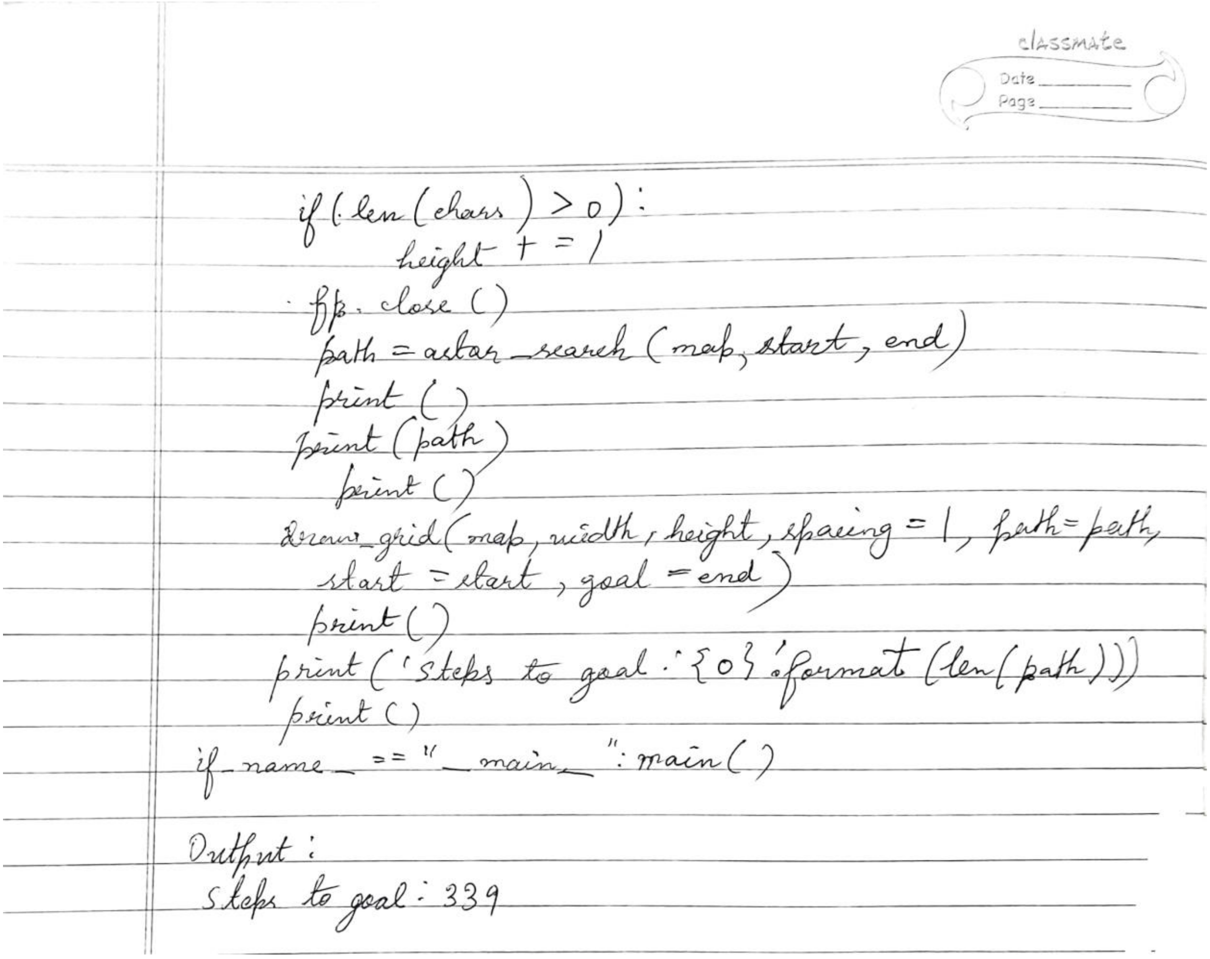
Implement A* search algorithm [Maze Problem]

```
class Node:
        def-init-(self, position:(), parent:()):
            self.position = position
            self.parent = parent
            self.g = 0 # Distance to start node
            self.h = 0 # Distance to goal node
            self.f = 0 # Total cost
        def-eq-(self, othes):
            return self.position ==othes.position
        def-it-(self, other):
            return self.f < other.f
        def-refr-(self):
            return ('({0}, {1})'.format(self.position, self.f))
def draw-grid(map, width, height, spacing = 2, **Kwargs):
    for y in range(height):
        for x in range(width):
            print('%%-%ds' % spacing % draw tile
                  (map, (x, y), Kwargs), end = '')
        print()
```

```
def draw_tile(map, position, Kwargs):
    value = map.get(position)
    if 'path' in Kwargs and position in Kwargs['path': value='+'
    if 'start' in Kwargs and position == Kwargs['start']: value='@'
    if 'goal' in Kwargs and position == Kwargs['goal']: value='$'
    return value

def astar_search(map, start, end):
    open=[]
    closed=[]
    start_node = Node(start, None)
    start_node = Node(End, None)
    open.append(start_node)
    while len(open) > 0:
      open.sort()
      current_node = open pop(0)
      closed.append(current_node)
      if current_node == goal_node:
         path = []
      while current_node != start_node:
       path.append(current_node.position)
       current_node=current_node.parent
      return path[::-1]
      (x, y) = current_node:position
      neighbors = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]
      for next in neighbors:
        map_value = map.get(next)
         if (map_value == '#'):
           continue
        neighbor = Node(next, current_node)
         if (neighbor in closed):
            continue
```

```
neighbor.g = abs( neighbor.position[0]-start_node.position[0])
            + abs (neighbor.position[1]-start_node.position[1])
neighbor.h= abs ( neighbor.positon [0]-goal_node.position[0])
           + abs (neighbor.position [1]-goal_node.position[1])
neighbor.f = neighbor.g+ neighbor.h
 if (add_to_open (open, neighbor) == True):
     open.append (neighbor)
 # Return None, no path is found
 return None
def add_to_open(open, neighbor):
     for node in open:
         if (neighbor == node and neighbor.f >= node.f):
             return False
         return True
    def main():
        map = {}
        chars = ['c']
        start = None
        end = None
        width = 0
        height = 0
         fp = open('maze.txt', 'r')
   while len(chars) > 0:
       chars = [str(i) for i in fp.readline().strip()]
       width = len(chars) if width == 0 else width
       for x in range(len(chars)):
           map[(x, height)] = chars[x]
       if (chars[x] == '@'):
           start = (x, height)
         elif (chars[x] == '$'):
             end = (x, height)
```



```
if (len (chars) > 0):
        height += 1
fp.close()
path = astar_search (map, start, end)
print ( )
print (path)
  print ( )
draw_grid(map, width, height, spacing = 1, path=path,
  start = start, goal = end)
print()
print('steps to goal: {0}'.format(len(path)))
print ( )
if_name_ == "_main_": main()
```

Output:

```
Steps to goal: 339
```

Program:

```
class Node:
    def __init__(self, position: (), parent: ()):
        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 __repr__(self):
        return ('({0},{1})'.format(self.position, self.f))


def draw_grid(map, width, height, spacing=2, **kwargs):
    for y in range(height):
        for x in range(width):
            print('%%-%ds' % spacing % draw_tile(map, (x, y), kwargs), end='')
        print()


def draw_tile(map, position, kwargs):

    value = map.get(position)

    if 'path' in kwargs and position in kwargs['path']: value = '+'

    if 'start' in kwargs and position == kwargs['start']: value = '@'

    if 'goal' in kwargs and position == kwargs['goal']: value = '$'

    return value
```

```
def astar_search(map, start, end):

    open = []
    closed = []

    start_node = Node(start, None)
    goal_node = Node(end, None)

    open.append(start_node)


    while len(open) > 0:

        open.sort()

        current_node = open.pop(0)

        closed.append(current_node)


        if current_node == goal_node:
            path = []
            while current_node != start_node:
                path.append(current_node.position)
                current_node = current_node.parent


            return path[::-1]
        (x, y) = current_node.position

        neighbors = [(x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)]

        for next in neighbors:

            map_value = map.get(next)

            if (map_value == '#'):
                continue

            neighbor = Node(next, current_node)

            if (neighbor in closed):
                continue

            neighbor.g = abs(neighbor.position[0] - start_node.position[0]) + abs(
                neighbor.position[1] - start_node.position[1])
            neighbor.h = abs(neighbor.position[0] - goal_node.position[0]) + abs(
                neighbor.position[1] - goal_node.position[1])
            neighbor.f = neighbor.g + neighbor.h

            if (add_to_open(open, neighbor) == True):

                open.append(neighbor)

    return None
```

```
def add_to_open(open, neighbor):
    for node in open:
        if (neighbor == node and neighbor.f >= node.f):
            return False
    return True




def main():

    map = {}
    chars = ['c']
    start = None
    end = None
    width = 0
    height = 0

    fp = open('maze.txt','r')



    while len(chars) > 0:

        chars = [str(i) for i in fp.readline().strip()]

        width = len(chars) if width == 0 else width

        for x in range(len(chars)):
            map[(x, height)] = chars[x]
            if (chars[x] == '@'):
                start = (x, height)
            elif (chars[x] == '$'):
                end = (x, height)



        if (len(chars) > 0):
            height += 1

    fp.close()

    path = astar_search(map, start, end)
    print()
    print(path)
    print()
    draw_grid(map, width, height, spacing=1, path=path, start=start, goal=end)
    print()
    print('Steps to goal: {0}'.format(len(path)))
    print()



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

Output:

```
[(39, 39), (38, 39), (37, 39), (36, 39), (35, 39), (34, 39), (33, 39), (33, 38), (33, 37), (32, 37),
(31, 37), (31, 38), (31, 39), (30, 39), (29, 39), (29, 38), (29, 37), (29, 36), (29, 35), (29, 34),
(29, 33), (29, 32), (29, 31), (29, 30), (29, 29), (28, 29), (27, 29), (27, 28), (27, 27), (27, 26),
(27, 25), (26, 25), (25, 25), (24, 25), (23, 25), (22, 25), (21, 25), (21, 26), (21, 27), (21, 28),
(21, 29), (22, 29), (23, 29), (23, 30), (23, 31), (24, 31), (25, 31), (25, 32), (25, 33), (25, 34),
(25, 35), (25, 36), (25, 37), (25, 38), (25, 39), (24, 39), (23, 39), (22, 39), (21, 39), (21, 38),
(21, 37), (21, 36), (21, 35), (22, 35), (23, 35), (23, 34), (23, 33), (22, 33), (21, 33), (21, 32),
(21, 31), (20, 31), (19, 31), (19, 32), (19, 33), (19, 34), (19, 35), (18, 35), (17, 35), (17, 34),
(17, 33), (17, 32), (17, 31), (17, 30), (17, 29), (16, 29), (15, 29), (15, 30), (15, 31), (15, 32),
(15, 33), (14, 33), (13, 33), (13, 34), (13, 35), (14, 35), (15, 35), (15, 36), (15, 37), (16, 37),
(17, 37), (18, 37), (19, 37), (19, 38), (19, 39), (18, 39), (17, 39), (16, 39), (15, 39), (14, 39),
(13, 39), (13, 38), (13, 37), (12, 37), (11, 37), (11, 38), (11, 39), (10, 39), (9, 39), (8, 39), (7,
39), (6, 39), (5, 39), (5, 38), (5, 37), (5, 36), (5, 35), (4, 35), (3, 35), (3, 34), (3, 33), (2, 33),
(1, 33), (1, 32), (1, 31), (1, 30), (1, 29), (1, 28), (1, 27), (2, 27), (3, 27), (4, 27), (5, 27), (6,
27), (7, 27), (7, 26), (7, 25), (7, 24), (7, 23), (8, 23), (9, 23), (9, 24), (9, 25), (9, 26), (9, 27),
(9, 28), (9, 29), (8, 29), (7, 29), (6, 29), (5, 29), (5, 30), (5, 31), (5, 32), (5, 33), (6, 33), (7,
33), (7, 34), (7, 35), (7, 36), (7, 37), (8, 37), (9, 37), (9, 36), (9, 35), (9, 34), (9, 33), (9, 32),
(9, 31), (10, 31), (11, 31), (11, 30), (11, 29), (11, 28), (11, 27), (12, 27), (13, 27), (13, 26),
(13, 25), (12, 25), (11, 25), (11, 24), (11, 23), (12, 23), (13, 23), (14, 23), (15, 23), (15, 22),
(15, 21), (14, 21), (13, 21), (12, 21), (11, 21), (10, 21), (9, 21), (8, 21), (7, 21), (6, 21), (5, 21),
(5, 22), (5, 23), (4, 23), (3, 23), (2, 23), (1, 23), (1, 22), (1, 21), (1, 20), (1, 19), (1, 18), (1,
17), (2, 17), (3, 17), (3, 16), (3, 15), (2, 15), (1, 15), (1, 14), (1, 13), (2, 13), (3, 13), (4, 13),
(5, 13), (5, 12), (5, 11), (6, 11), (7, 11), (7, 12), (7, 13), (8, 13), (9, 13), (9, 12), (9, 11), (9,
10), (9, 9), (8, 9), (7, 9), (6, 9), (5, 9), (4, 9), (3, 9), (3, 10), (3, 11), (2, 11), (1, 11), (1, 10), (1,
9), (1, 8), (1, 7), (1, 6), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5), (6, 5), (7, 5), (8, 5), (9, 5), (10, 5),
(11, 5), (11, 6), (11, 7), (11, 8), (11, 9), (11, 10), (11, 11), (11, 12), (11, 13), (11, 14), (11, 15),
(10, 15), (9, 15), (8, 15), (7, 15), (7, 16), (7, 17), (6, 17), (5, 17), (5, 18), (5, 19), (6, 19), (7,
19), (8, 19), (9, 19), (10, 19), (11, 19), (12, 19), (13, 19), (14, 19), (15, 19), (15, 18), (15, 17),
(15, 16), (15, 15), (14, 15), (13, 15), (13, 14), (13, 13), (14, 13), (15, 13), (16, 13), (17, 13),
(18, 13), (19, 13), (20, 13), (21, 13), (21, 12), (21, 11), (20, 11), (19, 11), (18, 11), (17, 11),
(17, 10), (17, 9), (16, 9), (15, 9), (15, 8), (15, 7), (14, 7), (13, 7), (13, 6), (13, 5), (13, 4), (13,
3), (12, 3), (11, 3), (11, 2), (11, 1)]
```

```
###############################################################################
#######.#...#....$....#..................#...#.........#.......#............#.......##.#.#.#.###+###.#########.
#########.#.######.#####.#####.#.#.########.###.#.######.##...#.....#+++#.#.........#.#.....#.#.
..#...#...#.......#.#.#.......#.#.#.#...#.################+#.#.##########.#.###.#.###.#.###.#.#.###
####.###.#######.#.#.#.#.#.##+++++++++++#+#...#.#.....#...#...#...#.#.#.#.#...#...#.......#.......
#.#.#.#.#.##+##########+#+######.#.#.#.#.###.######.#.#.#.#.######.#.##########.###.###.##
#.#.#.##+#........+#+++#...#.#.#.#...#.....#.#.#.#...#.#...#.......#.....#.#...#...#...##+##########+#.#
+###.#.#.######.###.#.#.#.#.#.###.#.##########.######.#.#.###.######.##+#++++++++#+#.#++
+#...#.#.....#.#.#.#...#.#.....#.#.....#.#...#...#.......#...#.##+#+######+#+#.###+######.#.######.#.#
.###.#.########.###.#.#.###.#.#############.#.#.##+++#++++#+#+#...#++++++#.#.......#.#.#...#.
....#...#...#.....#.#.#...#...#...#...#######+#+#+#+##########+#.########.#.###.########.#.###.#
#########.###.#.#.#.#.#########+++++#++++#+#++++++++++#.......#.#...#.#.#.....#.#.....#.......#
...#.#.#.#.#.....##+##########+#+##########.###.###.###.#.#.#.###.#.#.###.#.########.###.#.
###.#.###.##+++#.#++++++#++++#.....#.#.#...#.#.#.....#...#.#.#...#.#...#...#...#.#.#...#...#.####+#.
#+######.#+#.#.###.#.###.#.#.######.###.###.######.###.#.#.#.###.#.#.######.#.##+++#++++#.
....#+#.#.#...#...#.....#...#.#...#..........#.#.#...#...#.......#.#.##+###+##########+#.#.#.###.#.####
#.#.#.###.###.############.#.######.##########.###.##+#..++++++++++++#.#.......#.#...#.#.#.
..#.#...#.#.......#.......#.#...#.....#...##+#.##############.##########.#.#.###.###.#.#.###.#.####
#.#.########.#.#.#.######.#.##+#.#++++++++++++#.#.#.#.....#.#.....#...#.#.....#...#.#.#.#.#...#.#.
#.#.#.....#.##+###+##########+#.#.#.########.########.###.######.###.#.#.#.#.###.#.#.#.#.##
###.#.##+++++#++++#++++++#...#.........#.....#...#.....#...#...#.#.....#.#...#.#.#.....#.##.######+#+#
+########.############.########.#.########.###.#.###.###.######.#.#.######.##.....#+#+#++
+#...#.#++++++++#.........#.#...#.......#.#.#...#...#.....#.#.#...#.#########+#+###+#.###.#+######+
#.######.###.#.#.#.########.#.######.###.######.#.###.#.##++++++++#+#++++#.....#+#...#+#...#
.#.....#.#.#.#.#.....#...#...#...#.....#...#.#.##+########+#+#.######.#+###.#+###.#.########.#.#.#.
#.########.#.###.#.###.######.#.#.##+#.#++++++#+#.#++++#.#++++#.#++++#...#.#...#.#...#.#.....#
.#...#...#...#.......#...##+#.#+######+#.#+#+######+#.###+###.#.#.#.#.######.######.#.######.##
###.##########.####+#..+#..+++#.#+#+#++++#++++#.#+#...#...#.#.#...#.....#...#.#...#...#.....#...#
.#.##+###+###+#.###+#+#+#+####+#.#+#.########.#.#.#.######.###.#.#.###.#.######.###.#.#
.##+++#++++#+#.#++++#+#+#++++#+#.#+#.#.......#...#.........#.#...#...#.#...#...#.#...##.#+###+#+
#.#+###+#+###+#+#.#+#.###.###.############.###.#.###.###.###.###.#.###.##.#+++#+#+
#.#+++#+++#++++#+#.#+#.....#...#...#.....#.#...#.....#.....#.#...#...#.##.###+#+#+######+######+
#.#+#.#+########.###.#.######.#.#.##############.#.#.###.#.#.##...#+#+++#+++#++++++#+
#.#+#.#+#+++#...#.#.#.......#.#.#...#...#...#...#.#.#.#...####.#+######+#+######+#+###+#.#+#+
#+#.###.#.##########.#.#.#.#.#.#.#.######.#.#.#######...#++++++++#++++++++#++++++..#+++
#++++++++@..........#...#...#...#.......#.......#########################################
###############################################################################
```

Steps to goal: 339

Process finished with exit code 0