# COMSATS UNVERISTY ISLAMABAD

# Artificial Intelligence

# Lab 7

**Submitted by:**

Hasaan Ahmad    SP22-BSE-017

**Submitted to:**

**Sir Waqas Ali**

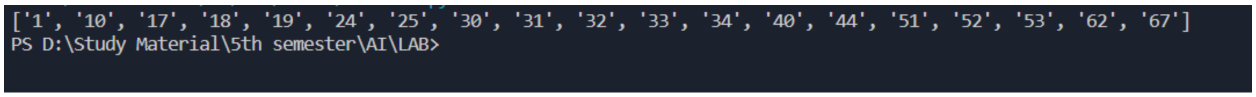
# Graded Lab 1:

## Manhattan Distance:

```python
import math


class Node:
    def __init__(self, state, parent, actions, totalCost, heuristic):
        self.state = state
        self.parent = parent
        self.actions = actions
        self.totalCost = totalCost
        self.heuristic = heuristic


def findMin(frontier):
    minV = math.inf
    node = None
    for i in frontier:
        if minV > frontier[i][1]:
            minV = frontier[i][1]
            node = i
    return node


def actionSequence(graph, initialState, goalState):
    solution = [goalState]
    currentParent = graph[goalState].parent
    while currentParent is not None:
        solution.append(currentParent)
        currentParent = graph[currentParent].parent
    solution.reverse()
    return solution


def Astar():
    initialState = '1'
    goalState = '67'

    graph = {
        '1': Node('1', None, [('2', 1), ('10', 1)], 0, (0, 0)),
        '2': Node('2', None, [('3', 1), ('11', 1)], 0, (0, 1)),
    '3': Node('3', None, [('4', 1), ('12', 1)], 0, (0, 2)),
    '4': Node('4', None, [('5', 1), ('11', 1)], 0, (0, 3)),
    '5': Node('5', None, [('6', 1), ('12', 1)], 0, (0, 5)),
    '6': Node('6', None, [('7', 1), ('11', 1)], 0, (0, 6)),
    '7': Node('7', None, [('8', 1), ('12', 1)], 0, (1, 7)),
    '8': Node('8', None, [('9', 1), ('11', 1)], 0, (1, 8)),
    '9': Node('9', None, [('10', 1), ('12', 1)], 0, (2, 9)),
    '10': Node('10', None, [('11', 1), ('17', 1)], 0, (1, 0)),
    '11': Node('11', None, [('12', 1)], 0, (1, 1)),
    '12': Node('12', None, [('13', 1)], 0, (1, 3)),
    '13': Node('13', None, [('14', 1)], 0, (1, 4)),
    '14': Node('14', None, [('15', 1)], 0, (1, 5)),
    '15': Node('15', None, [('16', 1)], 0, (1, 8)),
    '16': Node('16', None, [('17', 1)], 0, (1, 9)),
    '17': Node('17', None, [('18', 1)], 0, (2, 0)),
    '18': Node('18', None, [('19', 1)], 0, (2, 1)),
    '19': Node('19', None, [('24', 1),("18",1)], 0, (2, 2)),
    '20': Node('20', None, [('21', 1)], 0, (2, 5)),
    '21': Node('21', None, [('22', 1)], 0, (2, 6)),
    '22': Node('22', None, [('23', 1)], 0, (3, 9)),
    '23': Node('23', None, [('24', 1)], 0, (4, 0)),
    '24': Node('24', None, [('25', 1)], 0, (4, 2)),
    '25': Node('25', None, [('24', 1),('30',1)], 0, (4, 3)),
    '26': Node('26', None, [('27', 1)], 0, (4, 4)),
    '27': Node('27', None, [('28', 1)], 0, (4, 5)),
    '28': Node('28', None, [('29', 1)], 0, (5, 0)),
    '29': Node('29', None, [('30', 1)], 0, (5, 1)),
    '30': Node('30', None, [('31', 1)], 0, (5, 3)),
    '31': Node('31', None, [('32', 1)], 0, (5, 4)),
    '32': Node('32', None, [('33', 1)], 0, (5, 5)),
    '33': Node('33', None, [('34', 1)], 0, (5, 6)),
    '34': Node('34', None, [('40', 1)], 0, (5, 7)),
    '35': Node('35', None, [('36', 1)], 0, (5, 8)),
    '36': Node('36', None, [('37', 1)], 0, (5, 9)),
    '37': Node('37', None, [('38', 1)], 0, (6, 0)),
    '38': Node('38', None, [('39', 1)], 0, (6, 1)),
    '39': Node('39', None, [('40', 1)], 0, (6, 2)),
    '40': Node('40', None, [('44', 1)], 0, (6, 3)),
    '41': Node('41', None, [('42', 1)], 0, (6, 9)),
    '42': Node('42', None, [('43', 1)], 0, (7, 0)),
    '43': Node('43', None, [('44', 1)], 0, (7, 1)),
    '44': Node('44', None, [('45', 1),('43', 1),('51', 1)], 0, (7, 2)),
    '45': Node('45', None, [('46', 1)], 0, (7, 3)),
    '46': Node('46', None, [('47', 1)], 0, (7, 4)),
    '47': Node('47', None, [('48', 1)], 0, (7, 5)),
        '48': Node('48', None, [('49', 1), ('52', 1)], 0, (8, 0)),
        '49': Node('49', None, [('50', 1)], 0, (8, 1)),
        '50': Node('50', None, [('51', 1)], 0, (8, 2)),
        '51': Node('51', None, [('52', 1)], 0, (8, 3)),
        '52': Node('52', None, [('53', 1)], 0, (8, 4)),
        '53': Node('53', None, [('62', 1)], 0, (9, 0)),
        '54': Node('54', None, [('55', 1)], 0, (9, 1)),
        '55': Node('55', None, [('56', 1)], 0, (9, 2)),
        '56': Node('56', None, [('57', 1)], 0, (9, 3)),
        '57': Node('57', None, [('58', 1)], 0, (9, 4)),
        '58': Node('58', None, [('59', 1)], 0, (9, 5)),
        '59': Node('59', None, [('60', 1)], 0, (9, 6)),
        '60': Node('60', None, [('61', 1)], 0, (9, 7)),
        '61': Node('61', None, [('62', 1)], 0, (9, 8)),
        '62': Node('62', None, [('67', 1)], 0, (9, 9)),
        '63': Node('63', None, [('64', 1)], 0, (10, 0)),
        '64': Node('64', None, [('65', 1)], 0, (10, 1)),
        '65': Node('65', None, [('66', 1),("59",1)], 0, (9, 6)),
        '66': Node('66', None, [('60', 1),("65" , 1)], 0, (9, 7)),
        '67': Node('67', None, [('62', 1)], 0, (9, 9)),
    }

    frontier = {}
    heuristicCost = abs(graph[initialState].heuristic[0] -
graph[goalState].heuristic[0]) + abs(graph[initialState].heuristic[1] -
graph[goalState].heuristic[1])

    frontier[initialState] = (None, heuristicCost)
    explored = {}

    while len(frontier) != 0:
        currentNode = findMin(frontier)
        del frontier[currentNode]

        if graph[currentNode].state == goalState:
            return actionSequence(graph, initialState, goalState)


        heuristicCost = abs(graph[currentNode].heuristic[0] -
graph[goalState].heuristic[0]) + abs(graph[currentNode].heuristic[1] -
graph[goalState].heuristic[1])

        currentCost = graph[currentNode].totalCost
        explored[currentNode] = (graph[currentNode].parent, heuristicCost +
currentCost)
```

```
        for child, cost in graph[currentNode].actions:
            currentCost = cost + graph[currentNode].totalCost

            # Manhattan Distance
            heuristicCost = abs(graph[child].heuristic[0] -
graph[goalState].heuristic[0]) + abs(graph[child].heuristic[1] -
graph[goalState].heuristic[1])




            if child in explored:
                if graph[child].parent == currentNode or child == initialState or
\
                        explored[child][1] <= currentCost + heuristicCost:
                    continue

            if child not in frontier:
                graph[child].parent = currentNode
                graph[child].totalCost = currentCost
                frontier[child] = (graph[child].parent, currentCost +
heuristicCost)
            else:
                if frontier[child][1] < currentCost + heuristicCost:
                    graph[child].parent = frontier[child][0]
                    graph[child].totalCost = frontier[child][1] - heuristicCost
                else:
                    frontier[child] = (currentNode, currentCost + heuristicCost)
                    graph[child].parent = frontier[child][0]
                    graph[child].totalCost = currentCost


solution = Astar()
print(solution)
```

## Output:

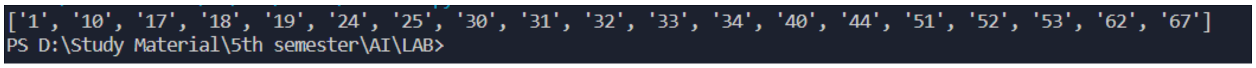


```
['1', '10', '17', '18', '19', '24', '25', '30', '31', '32', '33', '34', '40', '44', '51', '52', '53', '62', '67']
PS D:\Study Material\5th semester\AI\LAB>
```

## Diagonal Distance:

```
import math


class Node:
    def __init__(self, state, parent, actions, totalCost, heuristic):
        self.state = state
        self.parent = parent
        self.actions = actions
        self.totalCost = totalCost
        self.heuristic = heuristic


def findMin(frontier):
    minV = math.inf
    node = None
    for i in frontier:
        if minV > frontier[i][1]:
            minV = frontier[i][1]
            node = i
    return node


def actionSequence(graph, initialState, goalState):
    solution = [goalState]
    currentParent = graph[goalState].parent
    while currentParent is not None:
        solution.append(currentParent)
        currentParent = graph[currentParent].parent
    solution.reverse()
    return solution


def Astar():


    initialState = '1'
    goalState = '67'

    graph = {
        '1': Node('1', None, [('2', 1), ('10', 1)], 0, (0, 0)),
        '2': Node('2', None, [('3', 1), ('11', 1)], 0, (0, 1)),
        '3': Node('3', None, [('4', 1), ('12', 1)], 0, (0, 2)),
        '4': Node('4', None, [('5', 1), ('11', 1)], 0, (0, 3)),
        '5': Node('5', None, [('6', 1), ('12', 1)], 0, (0, 5)),
        '6': Node('6', None, [('7', 1), ('11', 1)], 0, (0, 6)),
        '7': Node('7', None, [('8', 1), ('12', 1)], 0, (1, 7)),
        '8': Node('8', None, [('9', 1), ('11', 1)], 0, (1, 8)),
        '9': Node('9', None, [('10', 1), ('12', 1)], 0, (2, 9)),
        '10': Node('10', None, [('11', 1), ('17', 1)], 0, (1, 0)),
        '11': Node('11', None, [('12', 1)], 0, (1, 1)),
        '12': Node('12', None, [('13', 1)], 0, (1, 3)),
        '13': Node('13', None, [('14', 1)], 0, (1, 4)),
        '14': Node('14', None, [('15', 1)], 0, (1, 5)),
        '15': Node('15', None, [('16', 1)], 0, (1, 8)),
        '16': Node('16', None, [('17', 1)], 0, (1, 9)),
        '17': Node('17', None, [('18', 1)], 0, (2, 0)),
        '18': Node('18', None, [('19', 1)], 0, (2, 1)),
        '19': Node('19', None, [('24', 1),("18",1)], 0, (2, 2)),
        '20': Node('20', None, [('21', 1)], 0, (2, 5)),
        '21': Node('21', None, [('22', 1)], 0, (2, 6)),
        '22': Node('22', None, [('23', 1)], 0, (3, 9)),
        '23': Node('23', None, [('24', 1)], 0, (4, 0)),
        '24': Node('24', None, [('25', 1)], 0, (4, 2)),
        '25': Node('25', None, [('24', 1),('30',1)], 0, (4, 3)),
        '26': Node('26', None, [('27', 1)], 0, (4, 4)),
        '27': Node('27', None, [('28', 1)], 0, (4, 5)),
        '28': Node('28', None, [('29', 1)], 0, (5, 0)),
        '29': Node('29', None, [('30', 1)], 0, (5, 1)),
        '30': Node('30', None, [('31', 1)], 0, (5, 3)),
        '31': Node('31', None, [('32', 1)], 0, (5, 4)),
        '32': Node('32', None, [('33', 1)], 0, (5, 5)),
        '33': Node('33', None, [('34', 1)], 0, (5, 6)),
        '34': Node('34', None, [('40', 1)], 0, (5, 7)),
        '35': Node('35', None, [('36', 1)], 0, (5, 8)),
        '36': Node('36', None, [('37', 1)], 0, (5, 9)),
        '37': Node('37', None, [('38', 1)], 0, (6, 0)),
        '38': Node('38', None, [('39', 1)], 0, (6, 1)),
        '39': Node('39', None, [('40', 1)], 0, (6, 2)),
        '40': Node('40', None, [('44', 1)], 0, (6, 3)),
        '41': Node('41', None, [('42', 1)], 0, (6, 9)),
        '42': Node('42', None, [('43', 1)], 0, (7, 0)),
        '43': Node('43', None, [('44', 1)], 0, (7, 1)),
        '44': Node('44', None, [('45', 1),('43', 1),('51', 1)], 0, (7, 2)),
        '45': Node('45', None, [('46', 1)], 0, (7, 3)),
        '46': Node('46', None, [('47', 1)], 0, (7, 4)),
        '47': Node('47', None, [('48', 1)], 0, (7, 5)),
        '48': Node('48', None, [('49', 1), ('52', 1)], 0, (8, 0)),
        '49': Node('49', None, [('50', 1)], 0, (8, 1)),
        '50': Node('50', None, [('51', 1)], 0, (8, 2)),
        '51': Node('51', None, [('52', 1)], 0, (8, 3)),
        '52': Node('52', None, [('53', 1)], 0, (8, 4)),
        '53': Node('53', None, [('62', 1)], 0, (9, 0)),
        '54': Node('54', None, [('55', 1)], 0, (9, 1)),
        '55': Node('55', None, [('56', 1)], 0, (9, 2)),
        '56': Node('56', None, [('57', 1)], 0, (9, 3)),
        '57': Node('57', None, [('58', 1)], 0, (9, 4)),
        '58': Node('58', None, [('59', 1)], 0, (9, 5)),
        '59': Node('59', None, [('60', 1)], 0, (9, 6)),
        '60': Node('60', None, [('61', 1)], 0, (9, 7)),
        '61': Node('61', None, [('62', 1)], 0, (9, 8)),
        '62': Node('62', None, [('67', 1)], 0, (9, 9)),
        '63': Node('63', None, [('64', 1)], 0, (10, 0)),
        '64': Node('64', None, [('65', 1)], 0, (10, 1)),
        '65': Node('65', None, [('66', 1),("59",1)], 0, (9, 6)),
        '66': Node('66', None, [('60', 1),("65" , 1)], 0, (9, 7)),
        '67': Node('67', None, [('62', 1)], 0, (9, 9)),
    }


    frontier = {}
    dx = abs(graph[initialState].heuristic[0] - graph[goalState].heuristic[0])
    dy = abs(graph[initialState].heuristic[1] - graph[goalState].heuristic[1])
    heuristicCost = (dx + dy) + (math.sqrt(2) - 2) * min(dx, dy)
    frontier[initialState] = (None, heuristicCost)
    explored = {}

    while len(frontier) != 0:
        currentNode = findMin(frontier)
        del frontier[currentNode]

        if graph[currentNode].state == goalState:
            return actionSequence(graph, initialState, goalState)

        dx = abs(graph[currentNode].heuristic[0] - graph[goalState].heuristic[0])
        dy = abs(graph[currentNode].heuristic[1] - graph[goalState].heuristic[1])
        heuristicCost = (dx + dy) + (math.sqrt(2) - 2) * min(dx, dy)
        currentCost = graph[currentNode].totalCost
        explored[currentNode] = (graph[currentNode].parent, heuristicCost +
currentCost)

        for child, cost in graph[currentNode].actions:
            currentCost = cost + graph[currentNode].totalCost
            # Euclidean Distance
            # heuristicCost = math.sqrt((graph[goalState].heuristic[0] -
graph[child].heuristic[0]) ** 2 +
            #                            (graph[goalState].heuristic[1] -
graph[child].heuristic[1]) ** 2)


            # Manhattan Distance
            # heuristicCost = abs(graph[child].heuristic[0] -
graph[goalState].heuristic[0]) + abs(graph[child].heuristic[1] -
graph[goalState].heuristic[1])


            # Diagonal Distance
            dx = abs(graph[child].heuristic[0]  -  graph[goalState].heuristic[0])
            dy = abs(graph[child].heuristic[1]  -  graph[goalState].heuristic[1])
            heuristicCost = ( dx+ dy) + (math.sqrt(2)- 2)* min(dx, dy)

            if child in explored:
                if graph[child].parent == currentNode or child == initialState or
\
                        explored[child][1] <= currentCost + heuristicCost:
                    continue

            if child not in frontier:
                graph[child].parent = currentNode
                graph[child].totalCost = currentCost
                frontier[child] = (graph[child].parent, currentCost +
heuristicCost)
            else:
                if frontier[child][1] < currentCost + heuristicCost:
                    graph[child].parent = frontier[child][0]
                    graph[child].totalCost = frontier[child][1] - heuristicCost
                else:
                    frontier[child] = (currentNode, currentCost + heuristicCost)
                    graph[child].parent = frontier[child][0]
                    graph[child].totalCost = currentCost


solution = Astar()
print(solution)
```

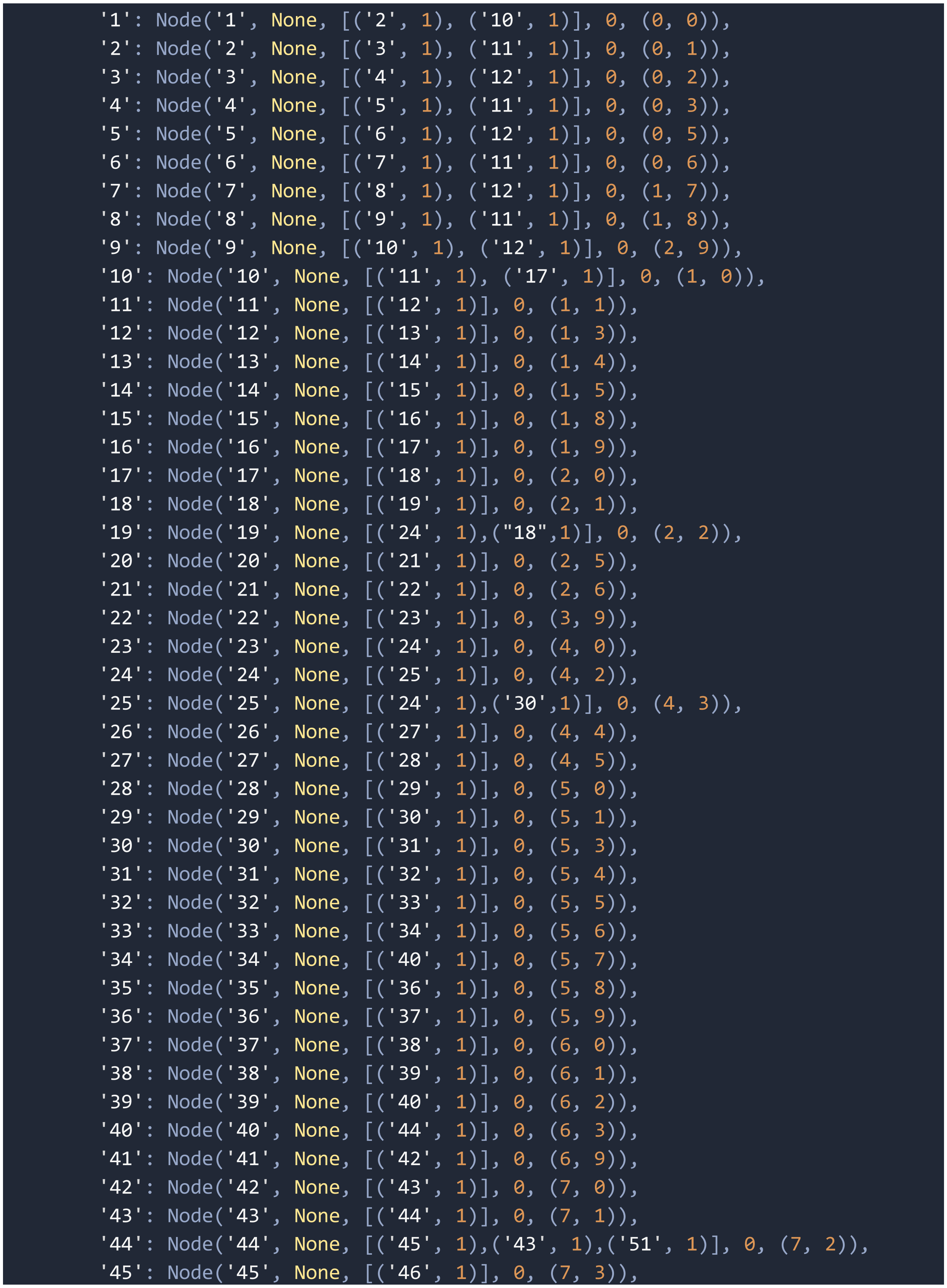
## Output:



```
['1', '10', '17', '18', '19', '24', '25', '30', '31', '32', '33', '34', '40', '44', '51', '52', '53', '62', '67']
PS D:\Study Material\5th semester\AI\LAB>
```

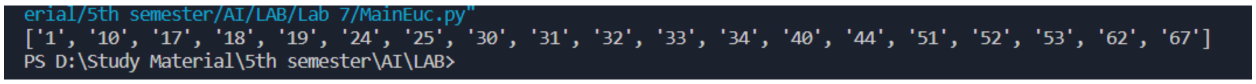
## Euclidean Distance:

```python
import math


class Node:
    def __init__(self, state, parent, actions, totalCost, heuristic):
        self.state = state
        self.parent = parent
        self.actions = actions
        self.totalCost = totalCost
        self.heuristic = heuristic


def findMin(frontier):
    minV = math.inf
    node = None
    for i in frontier:
        if minV > frontier[i][1]:
            minV = frontier[i][1]
            node = i
    return node


def actionSequence(graph, initialState, goalState):
    solution = [goalState]
    currentParent = graph[goalState].parent
    while currentParent is not None:
        solution.append(currentParent)
        currentParent = graph[currentParent].parent
    solution.reverse()
    return solution


def Astar():
    initialState = '1'
    goalState = '67'

    graph = {
```



```
        '1': Node('1', None, [('2', 1), ('10', 1)], 0, (0, 0)),
        '2': Node('2', None, [('3', 1), ('11', 1)], 0, (0, 1)),
        '3': Node('3', None, [('4', 1), ('12', 1)], 0, (0, 2)),
        '4': Node('4', None, [('5', 1), ('11', 1)], 0, (0, 3)),
        '5': Node('5', None, [('6', 1), ('12', 1)], 0, (0, 5)),
        '6': Node('6', None, [('7', 1), ('11', 1)], 0, (0, 6)),
        '7': Node('7', None, [('8', 1), ('12', 1)], 0, (1, 7)),
        '8': Node('8', None, [('9', 1), ('11', 1)], 0, (1, 8)),
        '9': Node('9', None, [('10', 1), ('12', 1)], 0, (2, 9)),
        '10': Node('10', None, [('11', 1), ('17', 1)], 0, (1, 0)),
        '11': Node('11', None, [('12', 1)], 0, (1, 1)),
        '12': Node('12', None, [('13', 1)], 0, (1, 3)),
        '13': Node('13', None, [('14', 1)], 0, (1, 4)),
        '14': Node('14', None, [('15', 1)], 0, (1, 5)),
        '15': Node('15', None, [('16', 1)], 0, (1, 8)),
        '16': Node('16', None, [('17', 1)], 0, (1, 9)),
        '17': Node('17', None, [('18', 1)], 0, (2, 0)),
        '18': Node('18', None, [('19', 1)], 0, (2, 1)),
        '19': Node('19', None, [('24', 1),("18",1)], 0, (2, 2)),
        '20': Node('20', None, [('21', 1)], 0, (2, 5)),
        '21': Node('21', None, [('22', 1)], 0, (2, 6)),
        '22': Node('22', None, [('23', 1)], 0, (3, 9)),
        '23': Node('23', None, [('24', 1)], 0, (4, 0)),
        '24': Node('24', None, [('25', 1)], 0, (4, 2)),
        '25': Node('25', None, [('24', 1),('30',1)], 0, (4, 3)),
        '26': Node('26', None, [('27', 1)], 0, (4, 4)),
        '27': Node('27', None, [('28', 1)], 0, (4, 5)),
        '28': Node('28', None, [('29', 1)], 0, (5, 0)),
        '29': Node('29', None, [('30', 1)], 0, (5, 1)),
        '30': Node('30', None, [('31', 1)], 0, (5, 3)),
        '31': Node('31', None, [('32', 1)], 0, (5, 4)),
        '32': Node('32', None, [('33', 1)], 0, (5, 5)),
        '33': Node('33', None, [('34', 1)], 0, (5, 6)),
        '34': Node('34', None, [('40', 1)], 0, (5, 7)),
        '35': Node('35', None, [('36', 1)], 0, (5, 8)),
        '36': Node('36', None, [('37', 1)], 0, (5, 9)),
        '37': Node('37', None, [('38', 1)], 0, (6, 0)),
        '38': Node('38', None, [('39', 1)], 0, (6, 1)),
        '39': Node('39', None, [('40', 1)], 0, (6, 2)),
        '40': Node('40', None, [('44', 1)], 0, (6, 3)),
        '41': Node('41', None, [('42', 1)], 0, (6, 9)),
        '42': Node('42', None, [('43', 1)], 0, (7, 0)),
        '43': Node('43', None, [('44', 1)], 0, (7, 1)),
        '44': Node('44', None, [('45', 1),('43', 1),('51', 1)], 0, (7, 2)),
        '45': Node('45', None, [('46', 1)], 0, (7, 3)),
        '46': Node('46', None, [('47', 1)], 0, (7, 4)),
        '47': Node('47', None, [('48', 1)], 0, (7, 5)),
        '48': Node('48', None, [('49', 1), ('52', 1)], 0, (8, 0)),
        '49': Node('49', None, [('50', 1)], 0, (8, 1)),
        '50': Node('50', None, [('51', 1)], 0, (8, 2)),
        '51': Node('51', None, [('52', 1)], 0, (8, 3)),
        '52': Node('52', None, [('53', 1)], 0, (8, 4)),
        '53': Node('53', None, [('62', 1)], 0, (9, 0)),
        '54': Node('54', None, [('55', 1)], 0, (9, 1)),
        '55': Node('55', None, [('56', 1)], 0, (9, 2)),
        '56': Node('56', None, [('57', 1)], 0, (9, 3)),
        '57': Node('57', None, [('58', 1)], 0, (9, 4)),
        '58': Node('58', None, [('59', 1)], 0, (9, 5)),
        '59': Node('59', None, [('60', 1)], 0, (9, 6)),
        '60': Node('60', None, [('61', 1)], 0, (9, 7)),
        '61': Node('61', None, [('62', 1)], 0, (9, 8)),
        '62': Node('62', None, [('67', 1)], 0, (9, 9)),
        '63': Node('63', None, [('64', 1)], 0, (10, 0)),
        '64': Node('64', None, [('65', 1)], 0, (10, 1)),
        '65': Node('65', None, [('66', 1),("59",1)], 0, (9, 6)),
        '66': Node('66', None, [('60', 1),("65" , 1)], 0, (9, 7)),
        '67': Node('67', None, [('62', 1)], 0, (9, 9)),
    }

    frontier = {}
    heuristicCost = math.sqrt((graph[goalState].heuristic[0] -
graph[initialState].heuristic[0]) ** 2 +
                                      (graph[goalState].heuristic[1] -
graph[initialState].heuristic[1]) ** 2)
    frontier[initialState] = (None, heuristicCost)
    explored = {}

    while len(frontier) != 0:
        currentNode = findMin(frontier)
        del frontier[currentNode]

        if graph[currentNode].state == goalState:
            return actionSequence(graph, initialState, goalState)

        heuristicCost = math.sqrt((graph[goalState].heuristic[0] -
graph[goalState].heuristic[0]) ** 2 +
                                      (graph[goalState].heuristic[1] -
graph[goalState].heuristic[1]) ** 2)
        currentCost = graph[currentNode].totalCost
        explored[currentNode] = (graph[currentNode].parent, heuristicCost +
currentCost)

        for child, cost in graph[currentNode].actions:
            currentCost = cost + graph[currentNode].totalCost

            # Euclidean Distance
            heuristicCost = math.sqrt((graph[goalState].heuristic[0] -
graph[child].heuristic[0]) ** 2 +
                                      (graph[goalState].heuristic[1] -
graph[child].heuristic[1]) ** 2)



            if child in explored:
                if graph[child].parent == currentNode or child == initialState or
\
                        explored[child][1] <= currentCost + heuristicCost:
                    continue

            if child not in frontier:
                graph[child].parent = currentNode
                graph[child].totalCost = currentCost
                frontier[child] = (graph[child].parent, currentCost +
heuristicCost)
            else:
                if frontier[child][1] < currentCost + heuristicCost:
                    graph[child].parent = frontier[child][0]
                    graph[child].totalCost = frontier[child][1] - heuristicCost
                else:
                    frontier[child] = (currentNode, currentCost + heuristicCost)
                    graph[child].parent = frontier[child][0]
                    graph[child].totalCost = currentCost


solution = Astar()
print(solution)
```

## Output:



```
erial/5th semester/AI/LAB/Lab 7/MainEuc.py"
['1', '10', '17', '18', '19', '24', '25', '30', '31', '32', '33', '34', '40', '44', '51', '52', '53', '62', '67']
PS D:\Study Material\5th semester\AI\LAB>
```