******COMSATS University Islamabad (Lahore** **Campus)**

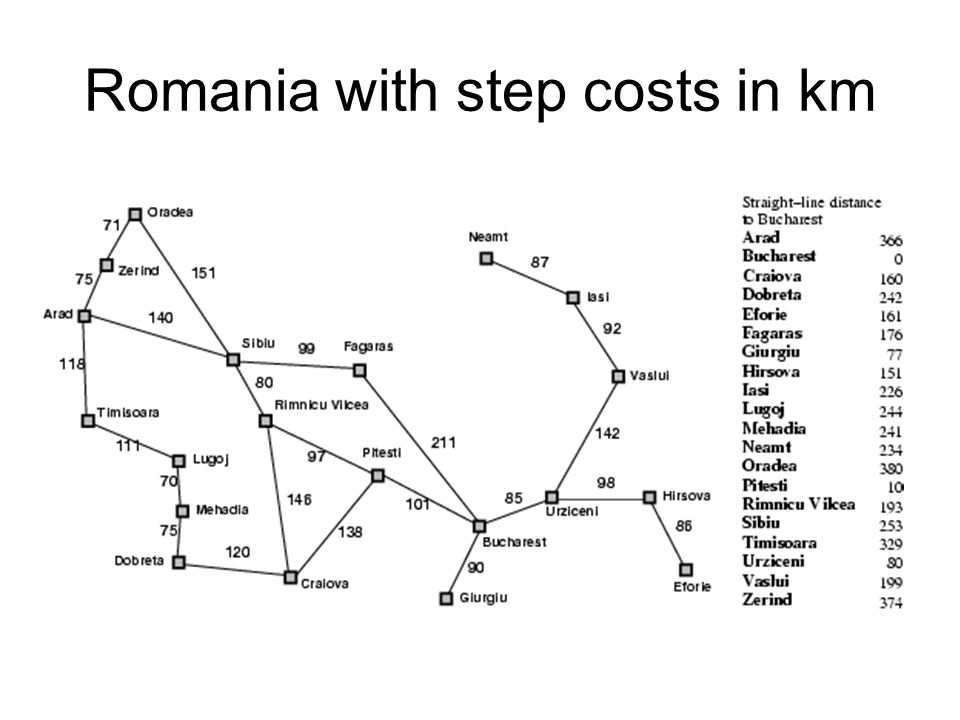
**Lab Assignment <2> FALL 2023**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Course Title: | Artificial Intelligence Lab | | | Course Code: | | CSC462 | Credit Hours: | 3(2,1) |
| Course Instructor: | Dr. Atifa Athar | | | Programme Name: | | BS Computer Science | | |
| Semester: | 6th | Batch: | SP20 | Section: | A | Date: | 09-10-2023 | |
| **Due Date:** | **16-04-2023** | | | **Maximum Marks:** | | | **10** | |
| **Student Name:** | **Hassan Mahmood** | | | **Registration No.** | | | **SP20-BCS-114** | |
| **Important Instructions / Guidelines:**   * **No late submissions will be accepted.** * **All assignments are required to be submitted using the attached template only.** | | | | | | | | |

**Question No 1. Marks: =10**

***CLO: <6>; Bloom Taxonomy Level: <Applying>***

Consider the following map of Romania. Apply the A\* search to find out the path and cost to reach Bucharest from Arad.



**Node Class:**

#  creating the every Node of the Graph acording to Constructor.

class Node:

    def \_\_init\_\_(self, state, parent, actions, heuristic, totalCost):

        self.state = state

        self.parent = parent

        self.actions = actions

        self.heuristic = heuristic  # Eculidean Distance.

        self.totalCost = totalCost

**Romanian Graph Implementation:**

# Romanian Map Graph for Search Traversal using A\* Algo.

graph = {

    'Arad':      Node('Arad', None, [('Sibiu', 140), ('Zerind', 75), ('Timisoara', 118)], (0,0), 366),

    'Zerind':    Node("Zerind", None, [('Arad', 75), ('Oradea', 71)], (0,0), 374),

    'Oradea':    Node('Oradea', None, [('Zerind', 71), ('Sibiu', 151)], (0,0), 380),

    'Sibiu':     Node('Sibiu', None, [('Arad', 140), ('Oradea', 151), ('Fagaras', 99), ('Rimnicu', 80)], (0,0), 253),

    'Timisoara': Node('Timisoara', None, [('Arad', 118), ('Lugoj', 111)], (0,0), 329),

    'Lugoj':     Node('Lugoj', None, [('Timisoara', 111), ('Mehadia', 70)], (0,0), 244),

    'Mehadia':   Node('Mehadia', None, [('Lugoj', 70), ('Drobeta', 75)], (0,0), 241),

    'Drobeta':   Node('Drobeta', None, [('Mehadia', 75), ('Craiova', 120)], (0,0), 242),

    'Craiova':   Node('Craiova', None, [('Drobeta', 120), ('Rimnicu', 146), ('Pitesti', 138)], (0,0), 160),

    'Rimnicu':   Node('Rimnicu', None, [('Sibiu', 80), ('Craiova', 146), ('Pitesti', 97)], (0,0), 193),

    'Fagaras':   Node('Fagaras', None, [('Sibiu', 99), ('Bucharest', 211)], (0,0), 176),

    'Pitesti':   Node('Pitesti', None, [('Rimnicu', 97), ('Craiova', 138), ('Bucharest', 101)], (0,0), 10),

    'Bucharest': Node('Bucharest', None, [('Fagaras', 211), ('Pitesti', 101), ('Giurgiu', 90), ('Urziceni', 85)], (0,0), 0),

    'Giurgiu':   Node('Giurgiu', None, [('Bucharest', 90)], (0,0), 77),

    'Urziceni':  Node('Urziceni', None, [('Bucharest', 85), ('Vaslui', 142), ('Hirsova', 98)], (0,0), 80),

    'Hirsova':   Node('Hirsova', None, [('Urziceni', 98), ('Eforie', 86)], (0,0), 151),

    'Eforie':    Node('Eforie', None, [('Hirsova', 86)], (0,0), 161),

    'Vaslui':    Node('Vaslui', None, [('Iasi', 92), ('Urziceni', 142)], (0,0), 199),

    'Iasi':      Node('Iasi', None, [('Vaslui', 92), ('Neamt', 87)], (0,0), 226),

    'Neamt':     Node('Neamt', None, [('Iasi', 87)], (0,0), 234)

}

**Find Min Function Implementation:**

# Iterate over frontier nodes and find the min out of them and return it. Frontier are the neighbours node of current node.

def findMin(frontier):

    minNode = None

    minCost = float("inf")

    for node, (parent, cost) in frontier.items():

        if cost < minCost:

            minNode = node

            minCost = cost

    return minNode

**Action Sequence:**

# It generates the sequence from initial state to goal state using A\* Algo

def actionSequence(graph, initialState, goalState):

    sequence = []

    currentNode = goalState

    while currentNode != initialState:

        parent = graph[currentNode].parent

        for action, \_ in graph[parent].actions:

            if action == currentNode:

                sequence.insert(0, action)

                break

        currentNode = parent

    sequence.insert(0, initialState)

    return sequence

**A\* Algorithm Code:**

def Astar\_Algo(graph, initialState, goalState):

    frontier = {}

    explored = {}

    heuristicCost = math.sqrt(

        (graph[goalState].heuristic[0] - graph[initialState].heuristic[0]) \*\* 2

        + (graph[goalState].heuristic[1] - graph[initialState].heuristic[1]) \*\* 2

    )

    # print(round(heuristicCost,2))

    frontier[initialState] = (None, heuristicCost)

    while frontier:

        currentNode = findMin(frontier)

        currentCost = frontier[currentNode][1]

        del frontier[currentNode]

        if currentNode == goalState:

            return actionSequence(graph, initialState, goalState)

        explored[currentNode] = (graph[currentNode].parent, currentCost)

        for child, actionCost in graph[currentNode].actions:

            childCost = currentCost + actionCost

            heuristicCost = math.sqrt(

                (graph[goalState].heuristic[0] - graph[child].heuristic[0]) \*\* 2

                + (graph[goalState].heuristic[1] - graph[child].heuristic[1]) \*\* 2

            )

            # print(round(heuristicCost,2))

            if child in explored:

                if (

                    graph[child].parent == currentNode

                    or child == initialState

                    or explored[child][1] <= childCost + heuristicCost

                ):

                    continue

            if child not in frontier or childCost < frontier[child][1]:

                graph[child].parent = currentNode

                frontier[child] = (graph[child].parent, childCost + heuristicCost)

                # print(round(heuristicCost,2))

    return None

initialState = "Arad"

goalState = "Bucharest"

result = Astar\_Algo(graph, initialState, goalState)

print(result)

**Results:**

[('Arad', 0), ('Sibiu', 140), ('Rimnicu', 80), ('Pitesti', 97), ('Bucharest', 101)]