sp21-bcs-017-A-assignment_2

October 12, 2023

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

[ ]:
```
import math

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

romania_map = {
    'Arad': [('Zerind', 75), ('Timisoara', 118), ('Sibiu', 140)],
    'Zerind': [('Arad', 75), ('Oradea', 71)],
    'Oradea': [('Zerind', 71), ('Sibiu', 151)],
    'Timisoara': [('Arad', 118), ('Lugoj', 111)],
    'Lugoj': [('Timisoara', 111), ('Mehadia', 70)],
    'Mehadia': [('Lugoj', 70), ('Drobeta', 75)],
    'Drobeta': [('Mehadia', 75), ('Craiova', 120)],
    'Sibiu': [('Arad', 140), ('Oradea', 151), ('Fagaras', 99), ('Rimnicu␣
 ↪Vilcea', 80)],
    'Fagaras': [('Sibiu', 99), ('Bucharest', 211)],
    'Rimnicu Vilcea': [('Sibiu', 80), ('Pitesti', 97), ('Craiova', 146)],
    'Craiova': [('Drobeta', 120), ('Rimnicu Vilcea', 146), ('Pitesti', 138)],
    'Pitesti': [('Craiova', 138), ('Rimnicu Vilcea', 97), ('Bucharest', 101)],
    'Bucharest': [('Fagaras', 211), ('Pitesti', 101), ('Giurgiu', 90),␣
 ↪('Urziceni', 85)],
    'Giurgiu': [('Bucharest', 90)],
    'Urziceni': [('Bucharest', 85), ('Vaslui', 142), ('Hirsova', 98)],
    'Vaslui': [('Urziceni', 142), ('Iasi', 92)],
    'Iasi': [('Vaslui', 92), ('Neamt', 87)],
    'Neamt': [('Iasi', 87)],
    'Hirsova': [('Urziceni', 98), ('Eforie', 86)],
    'Eforie': [('Hirsova', 86)]
}
```

```
def euclidean_distance(p1, p2):
    return math.sqrt(((p2[0] - p1[0]) ** 2 + (p2[1] - p1[1]) ** 2))


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

def findMin(frontier):
    minV = math.inf
    node = None
    for key, value in frontier.items():
        if value[1] < minV:
            minV = value[1]
            node = key
    return node

def a_star(graph, initialState, goalState):
    frontier = dict()
    explored = dict()

    heuristicCost = euclidean_distance(graph[goalState].heuristic,␣
 ↪graph[initialState].heuristic)

    frontier[initialState] = (None, heuristicCost)

    while frontier:
        currentNode = findMin(frontier)
        del frontier[currentNode]

        if currentNode == goalState:
            return actionSequence(graph, initialState, goalState)

        heuristicCost = euclidean_distance(graph[goalState].heuristic,␣
 ↪graph[currentNode].heuristic)
        currentCost = graph[currentNode].totalCost
        explored[currentNode] = (graph[currentNode].parent, heuristicCost +␣
 ↪currentCost)

        for child, step_cost in romania_map[currentNode]:
            child_node = graph[child]
            child_state = child_node.state

            currentCost = step_cost + graph[currentNode].totalCost
            heuristicCost = euclidean_distance(graph[goalState].heuristic,␣
 ↪child_node.heuristic)

            if child_state in explored:
                if child_node.parent == currentNode or child_state ==␣
 ↪initialState or explored[child_state][1] <= currentCost + heuristicCost:
                    continue

            if child_state not in frontier:
                child_node.parent = currentNode
                child_node.totalCost = currentCost
                frontier[child_state] = (child_node.parent, currentCost +␣
 ↪heuristicCost)
            else:
                if frontier[child_state][1] < currentCost + heuristicCost:
                    child_node.parent = frontier[child_state][0]
                    child_node.totalCost = frontier[child_state][1] -␣
 ↪heuristicCost
                else:
                    frontier[child_state] = (currentNode, currentCost +␣
 ↪heuristicCost)
                    child_node.parent = frontier[child_state][0]
                    child_node.totalCost = currentCost

    return None

initialState = 'Arad'
goalState = 'Bucharest'

# Create initial nodes with heuristics
graph = {}
for city in romania_map:
    graph[city] = Node(city, None, None, [0, 0], math.inf)

# Set the initial node's total cost to 0
graph[initialState].totalCost = 0

# Call A* search
result = a_star(graph, initialState, goalState)

if result:
    path, cost = result
    print("Optimal Path:", path)
    print("Total Cost:", cost)
else:
    print("No path found.")
```

```
Optimal Path: ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
Total Cost: 418
```