**Bahria University,**

Karachi Campus



## LAB EXPERIMENT NO.

## 4

## LIST OF TASKS

|  |  |
| --- | --- |
| **TASK NO** | **OBJECTIVE** |
| **1** | Implement A\* Search and find the optimal path for finding the goal  Start = S  Goal = E |
| **2** | Arad to neamt using A\* ALGORITHM |

**Submitted On:**

18 feb 2024

(Date: DD/MM/YY)

**TASK # 1:**

import heapq

graph = {

    'S': [('A', 1), ('B', 2)],

    'A': [('S', 1), ('Y', 7), ('X', 4)],

    'B': [('S', 2), ('C', 7), ('D', 1)],

    'C': [('B', 7), ('E', 5)],

    'D': [('B', 1), ('E', 12)],

    'E': [('C', 5), ('D', 12), ('X', 2), ('Y', 3)],

    'X': [('E', 2)],

    'Y': [('A', 7), ('E', 3)]

}

heuristics = {'A': 5, 'B': 6, 'C': 4, 'D': 15, 'E': 0, 'S': 6, 'X': 5, 'Y': 8}

def a\_star\_search(graph, start, goal, heuristics):

    pq = []

    heapq.heappush(pq, (heuristics[start], 0, start, []))  # (f, g, current\_node, path)

    visited = {}

    while pq:

        fx, gx, current, path = heapq.heappop(pq)

        if current in visited and visited[current] <= gx:

            continue

        visited[current] = gx

        new\_path = path + [current]

        if current == goal:

            return new\_path, gx

        for neighbor, cost in graph[current]:

            g\_temp = gx + cost

            f\_temp = g\_temp + heuristics[neighbor]

            if neighbor not in visited or visited[neighbor] > g\_temp:

                heapq.heappush(pq, (f\_temp, g\_temp, neighbor, new\_path))

    return None, None

path, cost = a\_star\_search(graph, 'S', 'E', heuristics)

print("Path:", path)

print("Cost:", cost)



**TASK # 2:**

import heapq

class Node:

    def \_\_init\_\_(self, name, h\_value):

        self.name = name

        self.h\_value = h\_value

        self.g\_value = float('inf')

        self.f\_value = float('inf')

        self.parent = None

        self.neighbors = {}

    def add\_neighbor(self, neighbor, distance):

        self.neighbors[neighbor] = distance

    def \_\_lt\_\_(self, other):

        return self.f\_value < other.f\_value

def a\_star(start, goal):

    open\_list = []

    closed\_set = set()

    start.g\_value = 0

    start.f\_value = start.h\_value

    heapq.heappush(open\_list, start)

    while open\_list:

        current = heapq.heappop(open\_list)

        if current.name == goal.name:

            return reconstruct\_path(current)

        closed\_set.add(current.name)

        for neighbor, distance in current.neighbors.items():

            if neighbor.name in closed\_set:

                continue

            tentative\_g\_value = current.g\_value + distance

            if tentative\_g\_value < neighbor.g\_value:

                neighbor.parent = current

                neighbor.g\_value = tentative\_g\_value

                neighbor.f\_value = tentative\_g\_value + neighbor.h\_value

                if neighbor not in open\_list:

                    heapq.heappush(open\_list, neighbor)

    return None

def reconstruct\_path(node):

    path = []

    while node:

        path.append(node.name)

        node = node.parent

    return path[::-1]

heuristic\_values = {

    'Arad': 366,

    'Bucharest': 0,

    'Neamt': 234,

    'Craiova': 160,

    'Dobreta': 242,

    'Eforie': 161,

    'Fagaras': 178,

    'Giurgiu': 77,

    'Hirsova': 151,

    'Iasi': 226,

    'Lugoj': 244,

    'Mehadia': 241,

    'Neamt': 234,

    'Oradea': 380,

    'Pitesti': 98,

    'Rimnicu Vilcea': 193,

    'Sibiu': 253,

    'Timisoara': 329,

    'Urziceni': 80,

    'Vaslui': 199,

    'Zerind': 374

}

nodes = {name: Node(name, h\_value) for name, h\_value in heuristic\_values.items()}

nodes['Arad'].add\_neighbor(nodes['Zerind'], 75)

nodes['Arad'].add\_neighbor(nodes['Sibiu'], 140)

nodes['Arad'].add\_neighbor(nodes['Timisoara'], 118)

nodes['Zerind'].add\_neighbor(nodes['Oradea'], 71)

nodes['Zerind'].add\_neighbor(nodes['Arad'], 75)

nodes['Oradea'].add\_neighbor(nodes['Zerind'], 71)

nodes['Oradea'].add\_neighbor(nodes['Sibiu'], 151)

nodes['Sibiu'].add\_neighbor(nodes['Oradea'], 151)

nodes['Sibiu'].add\_neighbor(nodes['Arad'], 140)

nodes['Sibiu'].add\_neighbor(nodes['Fagaras'], 99)

nodes['Sibiu'].add\_neighbor(nodes['Rimnicu Vilcea'], 80)

nodes['Timisoara'].add\_neighbor(nodes['Arad'], 118)

nodes['Timisoara'].add\_neighbor(nodes['Lugoj'], 111)

nodes['Lugoj'].add\_neighbor(nodes['Timisoara'], 111)

nodes['Lugoj'].add\_neighbor(nodes['Mehadia'], 70)

nodes['Mehadia'].add\_neighbor(nodes['Lugoj'], 70)

nodes['Mehadia'].add\_neighbor(nodes['Dobreta'], 75)

nodes['Dobreta'].add\_neighbor(nodes['Mehadia'], 75)

nodes['Dobreta'].add\_neighbor(nodes['Craiova'], 120)

nodes['Craiova'].add\_neighbor(nodes['Dobreta'], 120)

nodes['Craiova'].add\_neighbor(nodes['Rimnicu Vilcea'], 146)

nodes['Craiova'].add\_neighbor(nodes['Pitesti'], 138)

nodes['Rimnicu Vilcea'].add\_neighbor(nodes['Sibiu'], 80)

nodes['Rimnicu Vilcea'].add\_neighbor(nodes['Pitesti'], 97)

nodes['Rimnicu Vilcea'].add\_neighbor(nodes['Craiova'], 146)

nodes['Fagaras'].add\_neighbor(nodes['Sibiu'], 99)

nodes['Fagaras'].add\_neighbor(nodes['Bucharest'], 211)

nodes['Pitesti'].add\_neighbor(nodes['Rimnicu Vilcea'], 97)

nodes['Pitesti'].add\_neighbor(nodes['Craiova'], 138)

nodes['Pitesti'].add\_neighbor(nodes['Bucharest'], 101)

nodes['Bucharest'].add\_neighbor(nodes['Fagaras'], 211)

nodes['Bucharest'].add\_neighbor(nodes['Pitesti'], 101)

nodes['Bucharest'].add\_neighbor(nodes['Giurgiu'], 90)

nodes['Bucharest'].add\_neighbor(nodes['Urziceni'], 85)

nodes['Giurgiu'].add\_neighbor(nodes['Bucharest'], 90)

nodes['Urziceni'].add\_neighbor(nodes['Bucharest'], 85)

nodes['Urziceni'].add\_neighbor(nodes['Hirsova'], 98)

nodes['Urziceni'].add\_neighbor(nodes['Vaslui'], 142)

nodes['Hirsova'].add\_neighbor(nodes['Urziceni'], 98)

nodes['Hirsova'].add\_neighbor(nodes['Eforie'], 86)

nodes['Eforie'].add\_neighbor(nodes['Hirsova'], 86)

nodes['Vaslui'].add\_neighbor(nodes['Urziceni'], 142)

nodes['Vaslui'].add\_neighbor(nodes['Iasi'], 92)

nodes['Iasi'].add\_neighbor(nodes['Vaslui'], 92)

nodes['Iasi'].add\_neighbor(nodes['Neamt'], 87)

nodes['Neamt'].add\_neighbor(nodes['Iasi'], 87)

path = a\_star(nodes['Arad'], nodes['Neamt'])

print("Path found by A\*:", path)

