Worksheet 2.2

Student Name: Ayush Yadav **UID: 25IMC13004**
Branch: MCA (AI & ML) **Section/Group: MAM-1 (A)**
Semester: I **Date of Performance: 26/09/25**
Subject Name: Artificial Intelligence Tools - I **Subject Code: 25CAP-614**

1. Aim/Overview of the practical:

Experiment-2:
Implement A* Search Algorithm

2. Coding:

EXAMPLE-1

```
import networkx as nx
import matplotlib.pyplot as plt

def visualize_graph(graph, heuristic, path=None):
    G = nx.Graph()
    for node, neighbors in graph.items():
        for neighbor, _ in neighbors:
            G.add_edge(node, neighbor)

    pos = nx.spring_layout(G, seed=42)

    nx.draw(G, pos, with_labels=False, node_size=700, node_color="lightblue", font_weight="bold")

    labels = {node: f"{node}({heuristic[node]})" for node in G.nodes()}
    nx.draw_networkx_labels(G, pos, labels, font_size=10, font_weight="bold")
```

```
    if path:
        path_edges = list(zip(path, path[1:]))
        nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color="red", width=2.5)

    plt.title("A* Search Path with Heuristics")
    plt.show()

graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('A', 1), ('D', 1), ('E', 4)],
    'C': [('A', 3), ('F', 2)],
    'D': [('B', 1)],
    'E': [('B', 4), ('F', 1)],
    'F': [('C', 2), ('E', 1)]
}

heuristic = {
    'A': 6,
    'B': 5,
    'C': 2,
    'D': 6,
    'E': 1,
    'F': 0  # Goal node
}

visualize_graph(graph, heuristic, path=['A', 'C', 'F'])
```

```
# Function to visualize the graph
def visualize_graph(graph, heuristic, path=None):
    G = nx.Graph()
    for node, neighbors in graph.items():
        for neighbor, cost in neighbors:
            G.add_edge(node, neighbor, weight=cost)

    pos = nx.spring_layout(G, seed=42)

    nx.draw(G, pos, with_labels=False, node_size=700, node_color="lightblue", font_weight="bold")

    labels = {node: f"{node}({heuristic[node]})" for node in G.nodes()}
    nx.draw_networkx_labels(G, pos, labels, font_size=10, font_weight="bold")

    edge_labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

    if path:
        path_edges = list(zip(path, path[1:]))
        nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color="red", width=2.5)

    plt.title("A* Search Path with Heuristics")
    plt.show()

def a_star_search(graph, start, goal, heuristic):
    open_set = []
    heapq.heappush(open_set, (heuristic[start], start))
    came_from = {}

    g_score = {node: float('inf') for node in graph}
    g_score[start] = 0

    f_score = {node: float('inf') for node in graph}
    f_score[start] = heuristic[start]
```

```
    closed_set = set()

    while open_set:
        _, current = heapq.heappop(open_set)

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]

        closed_set.add(current)

        for neighbor, cost in graph[current]:
            if neighbor in closed_set:
                continue

            tentative_g_score = g_score[current] + cost
            if tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = tentative_g_score + heuristic[neighbor]

                if not any(neighbor == item[1] for item in open_set):
                    heapq.heappush(open_set, (f_score[neighbor], neighbor))

    return None

graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('A', 1), ('D', 1), ('E', 4)],
```

```
    'C': [('A', 3), ('F', 2)],
    'D': [('B', 1)],
    'E': [('B', 4), ('F', 1)],
    'F': [('C', 2), ('E', 1)]
}

heuristic = {
    'A': 6,
    'B': 5,
    'C': 2,
    'D': 6,
    'E': 1,
    'F': 0
}

visualize_graph(graph, heuristic)

path = a_star_search(graph, 'B', 'F', heuristic)
print("A* search path:", path)
visualize_graph(graph, heuristic, path)
```

OUTPUT:-

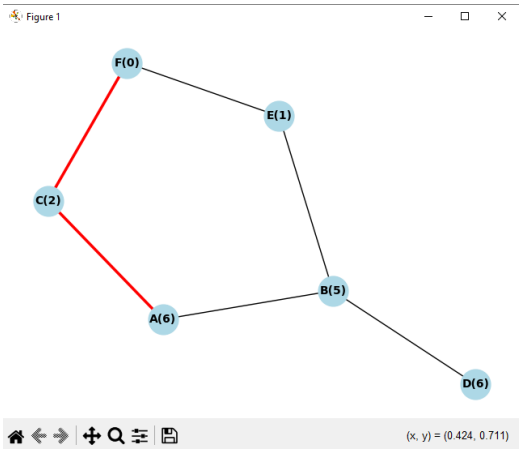
Figure 1
F(0)
E(1)
C(2)
B(5)
A(6)
D(6)
(x, y) = (0.424, 0.711)

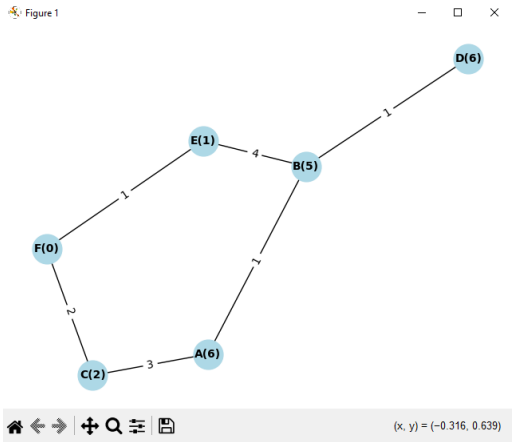
Figure 1
D(6)
E(1)
B(5)
F(0)
A(6)
C(2)
1
4
1
1
2
3
(x, y) = (−0.316, 0.639)

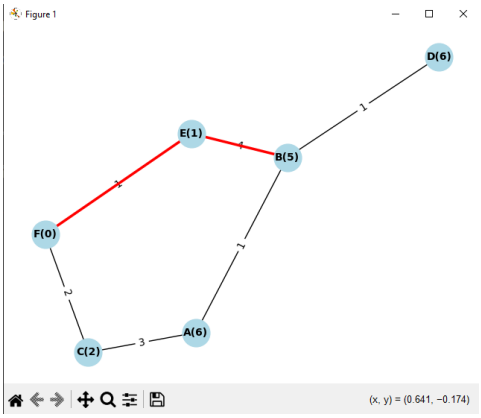
Figure 1
D(6)
E(1)
B(5)
F(0)
A(6)
C(2)
1
1
1
2
3
(x, y) = (0.641, −0.174)

EXAMPLE-2

```
import heapq
import networkx as nx
import matplotlib.pyplot as plt

def visualize_graph(graph, heuristic, path=None):
    G = nx.Graph()
    for node, neighbors in graph.items():
        for neighbor, cost in neighbors:
            G.add_edge(node, neighbor, weight=cost)

    pos = nx.spring_layout(G, seed=42)

    nx.draw(G, pos, with_labels=False, node_size=700,
            node_color="lightblue", font_weight="bold")

    labels = {node: f"{node}({heuristic[node]})" for node in G.nodes()}
    nx.draw_networkx_labels(G, pos, labels, font_size=10, font_weight="bold")

    if path and len(path) > 1:
        path_edges = list(zip(path, path[1:]))
        nx.draw_networkx_edges(G, pos, edgelist=path_edges,
                               edge_color="red", width=2.5)

    title = "Graph with Heuristics" if not path else "Graph with A* Path"
    plt.title(title)
    plt.show()

graph = {
    'S': [('X', 3), ('Z', 4)],
    'X': [('A2', 2), ('C', 3), ('Y', 9)],
    'Y': [('A', 5), ('B', 2), ('X', 9), ('Z', 8)],
    'Z': [('E', 1), ('Y', 8)],
    'A': [('Y', 5), ('D', 2)],
    'B': [('Y', 2), ('D', 4), ('E', 3)],
    'C': [('X', 3), ('Y2', 9), ('F', 3), ('G', 4)],
```

```
    'D': [('C', 9), ('A', 2), ('B', 4)],
    'E': [('B', 3), ('Z', 1), ('H', 7), ('I1', 8)],
    'F': [('C', 3), ('S2', 1), ('J', 2)],
    'G': [('C', 4), ('J', 2), ('K1', 3)],
    'H': [('E', 7), ('K1', 4)],
    'I1': [('E', 8), ('K1', 9)],
    'J': [('F', 2), ('G', 2), ('L', 7), ('H2', 5)],
    'K1': [('I1', 9), ('H', 4), ('G', 3), ('L', 2)],
    'L': [('J', 7), ('K1', 2)],
    'A2': [('Y2', 3), ('X', 2)],
    'Y2': [('A2', 3), ('U', 4), ('C', 9)],
    'U': [('Y2', 4), ('S2', 3)],
    'S2': [('U', 3), ('F', 1), ('H2', 2)],
    'H2': [('S2', 2), ('J', 5)]
}

heuristic = {
    'S': 1, 'X': 2, 'Y': 4, 'Z': 6,
    'A': 4, 'B': 3, 'C': 7, 'D': 2,
    'E': 5, 'F': 4, 'G': 5, 'H': 7,
    'I1': 9, 'J': 3, 'K1': 7, 'L': 1,
    'A2': 1, 'Y2': 2, 'U': 3, 'S2': 2, 'H2': 0  # H2 is goal
}

visualize_graph(graph, heuristic)


def a_star_search(graph, start, goal, heuristic):
    open_set = []
    heapq.heappush(open_set, (heuristic[start], start))

    came_from = {}
    g_score = {node: float('inf') for node in graph}
    g_score[start] = 0

    f_score = {node: float('inf') for node in graph}
    f_score[start] = heuristic[start]
```

```
    closed_set = set()

    while open_set:
        _, current = heapq.heappop(open_set)

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]

        closed_set.add(current)

        for neighbor, cost in graph[current]:
            if neighbor in closed_set:
                continue

            tentative_g_score = g_score[current] + cost
            if tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = tentative_g_score + heuristic[neighbor]

                if not any(neighbor == item[1] for item in open_set):
                    heapq.heappush(open_set, (f_score[neighbor], neighbor))

    return None

visualize_graph(graph, heuristic)

path = a_star_search(graph, 'S', 'H2', heuristic)
print("A* search path:", path)

visualize_graph(graph, heuristic, path)
```

(B). OUTPUT:

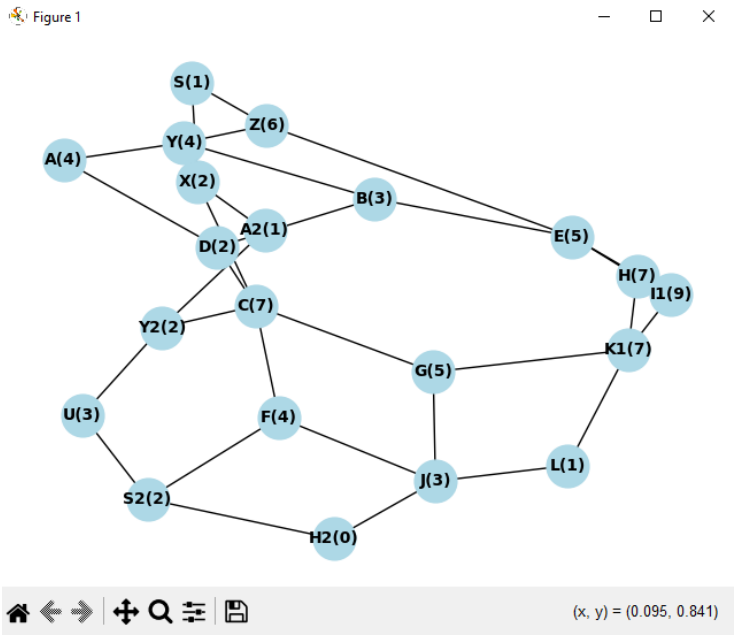
Figure 1
S(1)
Z(6)
Y(4)
A(4)
X(2)
B(3)
A2(1)
D(2)
E(5)
H(7)
I1(9)
C(7)
Y2(2)
K1(7)
G(5)
U(3)
F(4)
L(1)
J(3)
S2(2)
H2(0)
(x, y) = (0.095, 0.841)

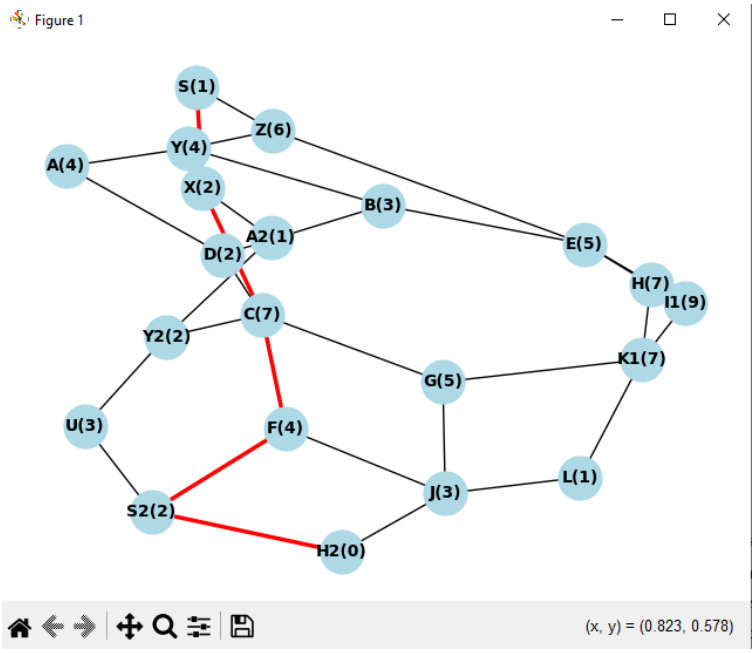
Figure 1
S(1)
Z(6)
Y(4)
A(4)
X(2)
B(3)
A2(1)
D(2)
E(5)
H(7)
I1(9)
C(7)
Y2(2)
K1(7)
G(5)
U(3)
F(4)
L(1)
J(3)
S2(2)
H2(0)
(x, y) = (0.823, 0.578)

3. Learning outcomes (What I have learnt):

- Understand the concept of heuristic-based informed search and how it differs from uninformed search.
- Learn how A* combines actual path cost and estimated cost to efficiently find the shortest path.
- Gain experience in using priority queues and handling graphs in pathfinding problems.
- Be able to implement and visualize A* algorithm in practical applications like maps, robotics, and AI.
- Develop skills to analyze algorithm performance and verify correctness of paths found