LAB ASSIGNMENT- 3

```
import heapq

class Node:
    def__init_(self, name, h):
        self.name = name
        self.h = h
        self.g = float('inf')
        self.parent = None
        self.neighbors = {}

    def__lt__(self, other):
        return (self.g + self.h) < (other.g + other.h)

def a_star(start, goal):
    open_list = []
    closed_set = set()

    start.g = 0
    heapq.heappush(open_list, start)

    while open_list:
        current = heapq.heappop(open_list)

        if current == goal:
            path = []
            while current:
                path.append(current.name)
                current = current.parent
            return path[::-1]

        closed_set.add(current)

        for neighbor, cost in current.neighbors.items():
            if neighbor in closed_set:
                continue

            tentative_g = current.g + cost

            if tentative_g < neighbor.g:
                neighbor.parent = current
                neighbor.g = tentative_g
                f = tentative_g + neighbor.h

                if neighbor not in open_list:
                    heapq.heappush(open_list, neighbor)
                else:
```

```
                # Update position in the heap
                open_list.remove(neighbor)
                heapq.heappush(open_list, neighbor)

    return None

# Create nodes
nodes = {
    'A': Node('A', 10), 'B': Node('B', 8), 'C': Node('C', 5),
    'D': Node('D', 7), 'E': Node('E', 3), 'F': Node('F', 6),
    'G': Node('G', 5), 'H': Node('H', 3), 'I': Node('I', 1),
    'J': Node('J', 0)
}

# Define edges
edges = [
    ('A', 'B', 6), ('A', 'F', 3), ('B', 'D', 2), ('B', 'C', 3),
    ('C', 'D', 1), ('C', 'E', 5), ('D', 'E', 8), ('E', 'I', 5),
    ('E', 'J', 5), ('F', 'G', 1), ('F', 'H', 7), ('G', 'I', 3),
    ('H', 'I', 2), ('I', 'J', 3)
]

# Add neighbors
for start, end, cost in edges:
    nodes[start].neighbors[nodes[end]] = cost
    nodes[end].neighbors[nodes[start]] = cost

# Run A* algorithm
path = a_star(nodes['A'], nodes['J'])

print("Path found by A* algorithm:", ' -> '.join(path))
```

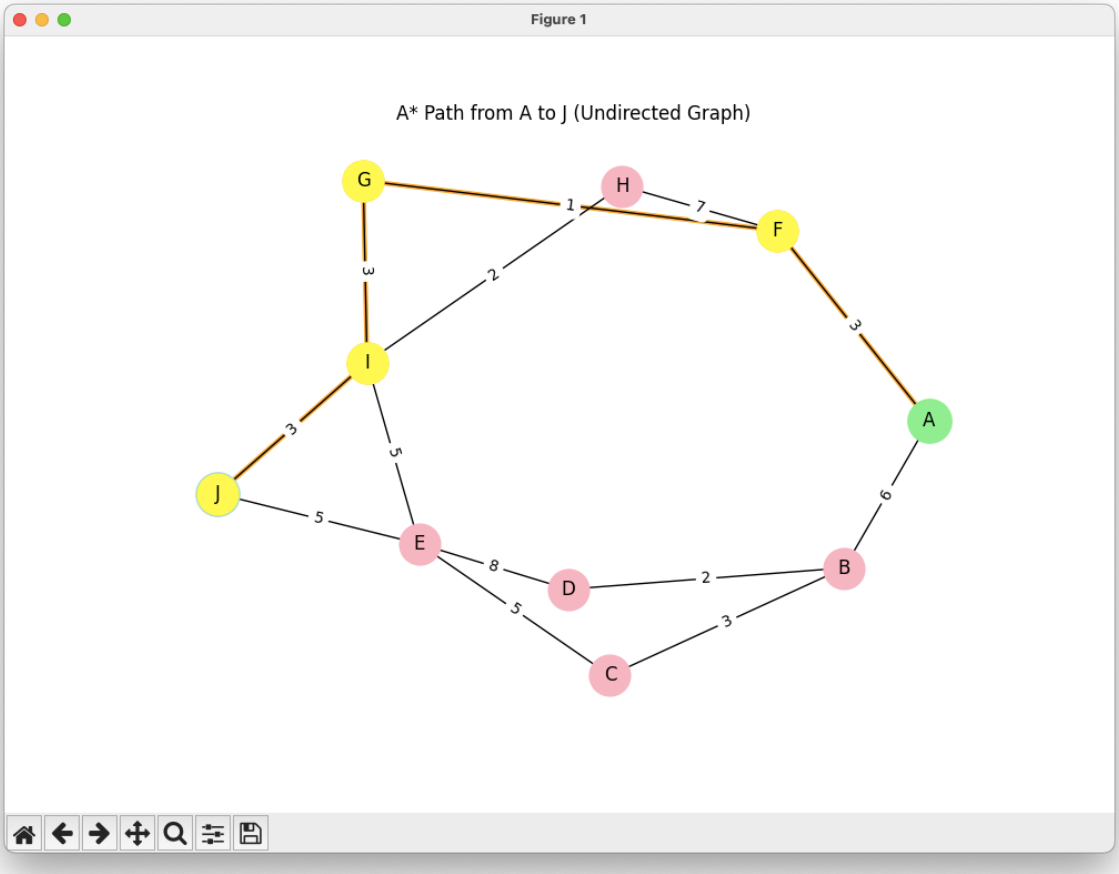
Figure 1
A* Path from A to J (Undirected Graph)
G
H
F
I
A
J
E
D
B
C
1
7
3
2
3
3
5
6
5
8
2
5
3