❖ Artificial Intelligence

❖ Lab Assignment :

- Submitted By: Noman Shakir (FA22-BSE-115)
- Submitted To: Ma'am Maleeha Khalid Khan
- Class = BSE-6A

- Video LINK:
- https://github.com/CodeWithZavi/AI_TASK_NOMAN

Question:

Implement Greedy and A* using simple if while or heap function for the following example:

Graph:

```
graph = {
    'S': {'A': 1, 'B': 1, 'C': 2},
    'A': {'E': 2, 'D': 1},
    'B': {'D': 1},
    'C': {'D': 1, 'F': 2},
    'D': {'A': 1, 'E': 2, 'H': 1, 'C': 2},
    'E': {'A': 2, 'D': 2, 'G': 3},
    'F': {'C': 2, 'H': 2},
    'G': {'E': 3, 'H': 1},
    'H': {'F': 2, 'D': 1, 'G': 1}
}
heuristic = {
    'S': 17, 'A': 2, 'B': 1, 'C': 2,
    'D': 1, 'E': 2, 'F': 2, 'G': 0, 'H': 1
}
```

G

Greedy:

```
[2] def greedy_best_first(graph, start, goal, heuristic):
        open_list = [(start, [start])]  # List of tuples (node, path)
        visited = set()
        while open_list:
            # Find node with minimum heuristic
            min_h = float('inf')
            min_index = -1
            for i in range(len(open_list)):
                node, _ = open_list[i]
                if heuristic[node] < min_h:
                    min_h = heuristic[node]
                    min_index = i
            current_node, path = open_list.pop(min_index)
            print("Visiting:", current_node)
            if current_node == goal:
                print("Goal reached!")
                return path
            visited.add(current_node)
            for neighbor in graph.get(current_node, {}):
                if neighbor not in visited:
                    open_list.append((neighbor, path + [neighbor]))
        return None
```

Greedy Best-First Search (GBFS) uses a heuristic to explore the most promising node first. It maintains an open list sorted by heuristic value and a visited set to avoid revisiting. The algorithm continues until the goal is found or all options are exhausted.

A*:

```
def a_star(graph, start, goal, heuristic):
    open_list = [(start, 0, [start])]  # List of tuples (node, cost_so_far, path)
    visited = set()
    while open_list:
        # Find node with minimum cost + heuristic
        min_f = float('inf')
        min_index = -1
        for i in range(len(open_list)):
            node, cost, _ = open_list[i]
            f = cost + heuristic[node]
            if f < min_f:
                min_f = f
                min_index = i
        current_node, cost_so_far, path = open_list.pop(min_index)
        print("Visiting:", current_node, "with cost", cost_so_far)
        if current_node == goal:
            print("Goal reached!")
            return path
        visited.add(current_node)
        for neighbor, weight in graph.get(current_node, {}).items():
            if neighbor not in visited:
                total_cost = cost_so_far + weight
                open_list.append((neighbor, total_cost, path + [neighbor]))
    return None
```

```
# Example usage
print("\nA* Search Path:")
path = a_star(graph, 'S', 'G', heuristic)
print(" -> ".join(path))

# Example usage
print("Greedy Best-First Search Path:")
path = greedy_best_first(graph, 'S', 'G', heuristic)
print(" -> ".join(path))
```

A Algorithm in 3 Lines (Simple Explanation):*

A* explores nodes by picking the one with the lowest total cost f(n) = g(n) + h(n) (actual cost + heuristic).

It keeps an open list of paths to explore and a visited set to avoid loops.

The search continues until the goal is reached or no nodes are left to check.

Output:

```
A* Search Path:
Visiting: S with cost 0
Visiting: B with cost 1
Visiting: A with cost 1
Visiting: D with cost 2
Visiting: D with cost 2
Visiting: C with cost 2
Visiting: H with cost 3
Visiting: H with cost 3
Visiting: G with cost 4
Goal reached!
S -> B -> D -> H -> G
Greedy Best-First Search Path:
Visiting: S
Visiting: B
Visiting: D
Visiting: H
Visiting: G
Goal reached!
S -> B -> D -> H -> G
```

Question:

Using only matplotlib plot a simple graph with student name on x-axis and marks obtained on quiz, assignment and Midterm on y-axis?

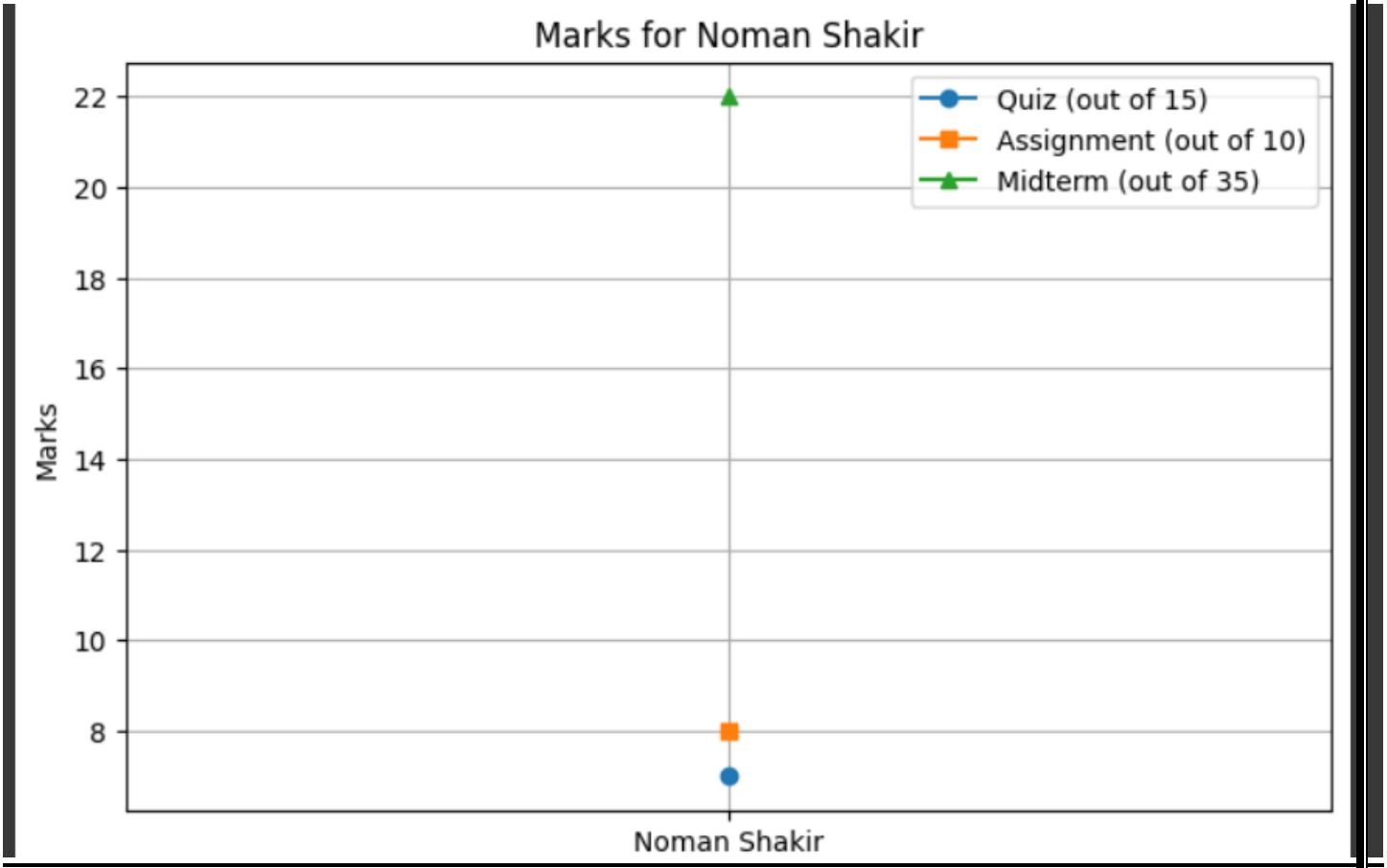
Code:

```python
import matplotlib.pyplot as plt
name = ['Noman Shakir']
quiz_marks = [7]
assignment_marks = [8]
midterm_marks = [22]

plt.figure(figsize=(8, 5))
plt.plot(name, quiz_marks, 'o-', label='Quiz (out of 15)')
plt.plot(name, assignment_marks, 's-', label='Assignment (out of 10)')
plt.plot(name, midterm_marks, '^-', label='Midterm (out of 35)')

plt.xlabel('Name')
plt.ylabel('Marks')
plt.title('Marks for Noman Shakir')
plt.legend()
plt.grid(True)
plt.show()
```

Output:



Matplotlib is used to plot a student's quiz, assignment, and midterm marks using different markers.
Each data point is labeled and shown with a legend for clarity.
The graph includes titles and axis labels and is displayed using plt.show().