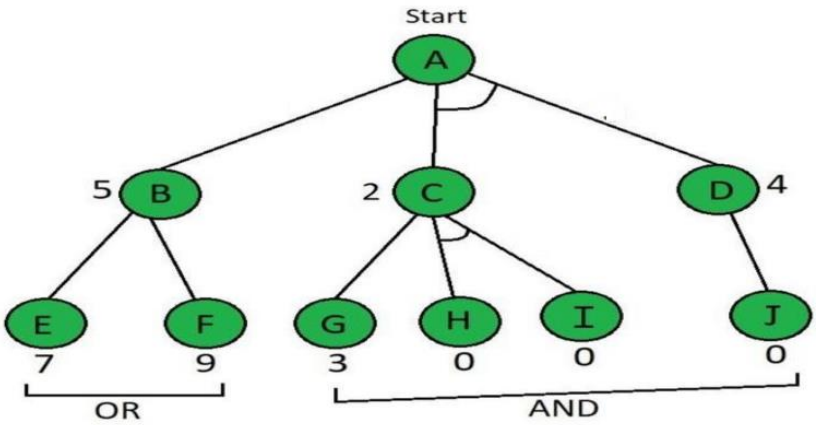
Assignment

Find the shortest path to node 'A' for the following AND-OR graph using the AO* algorithm. Below the Node given is the heuristic value, i.e., $h(n)$. Edge length is considered as 1.



Code:

```
# Define the OR and AND graphs
graph_OR = [[1], [4, 5], [6], [9], [], [], [], [], [], []]
graph_AND = [[2, 3], [], [7, 8], [], [], [], [], [], [], []]

# Initialize heuristic values, solvability flags, and parent array
heuristic_values = [1e6, 5, 2, 4, 7, 9, 3, 0, 0, 0]
solvability_flags = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1]
parent_array = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1]

# Define a depth-first search function to calculate heuristic values
def depth_first_search(node, parent):
    OR_hit = 0

    # Explore nodes in the OR graph
    for neighbor in graph_OR[node]:
        if parent != neighbor:
            depth_first_search(neighbor, node)
        if solvability_flags[neighbor] == 1:
            OR_hit = 1
            heuristic_values[node] = min(heuristic_values[node],
heuristic_values[neighbor] + 1)

    x = 0
    AND_hit = 0

    # Explore nodes in the AND graph
    for neighbor in graph_AND[node]:
        if parent != neighbor:
            depth_first_search(neighbor, node)
        x = x + heuristic_values[neighbor] + 1
```

```
        if solvability_flags[neighbor] == 1:
            AND_hit = 1

    if AND_hit == 1:
        if heuristic_values[node] > x:
            heuristic_values[node] = x

    if OR_hit == 1 or AND_hit:
        solvability_flags[node] = 1

# Define a function to print the shortest path
def print_shortest_path(node):
    if not graph_OR[node] and not graph_AND[node]:
        print(chr(ord('A') + node), end=" ")
        return
    else:
        print(chr(ord('A') + node), "->", end=" ")
        for neighbor in graph_OR[node]:
            if solvability_flags[neighbor] == 1:
                print("(", end="")
                print_shortest_path(neighbor)
                print(")", end=" ")
        count = 0
        for neighbor in graph_AND[node]:
            if solvability_flags[neighbor] == 1:
                count += 1
        if count == 0:
            return
        print("(", end=" ")
        for neighbor in graph_AND[node]:
            if solvability_flags[neighbor] == 1:
                print_shortest_path(neighbor)
                if count > 1:
                    print("AND", end=" ")
                count -= 1
        print(")", end=" ")

# Perform depth-first search starting from node 0
depth_first_search(0, -1)

# Print the shortest path
print("SHORTEST PATH: ", end="")
print_shortest_path(0)
print("")
print('_' * 75)

# Check solvability and print the result
if solvability_flags[0] == 1:
```

```
    print("It is solvable, with cost:", heuristic_values[0])
else:
    print("It is not solvable")

# Print heuristic values and solvability flags
print(heuristic_values)
print(solvability_flags)
```

Output:

```
SHORTEST PATH: A -> ( C -> ( H AND I ) AND D -> (J ) )
_____________________________________________________________________________
It is solvable, with cost: 5
[5, 5, 2, 1, 7, 9, 3, 0, 0, 0]
[1, 0, 1, 1, 0, 0, 0, 1, 1, 1]
```