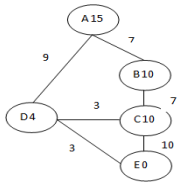
1. Write a python program for A* search for the given tree, in which ‘A’ is the initial state and ‘E’ goal state. For each node in each stage, obtain cost estimates where g(n) is the numeral by the side of an arc and h(n) is the numeral at the node.



Program :

```
from queue import PriorityQueue


class Node:
    def __init__(self, value, cost, heuristic):
        self.value = value
        self.cost = cost
        self.heuristic = heuristic

    def __lt__(self, other):
        return (self.cost + self.heuristic) < (other.cost + other.heuristic)

def a_star_search(graph, start, goal):
    priority_queue = PriorityQueue()
    visited = set()

    start_node = Node(start, 0, heuristic[start])
    priority_queue.put(start_node)
```

```
    while not priority_queue.empty():
        current_node = priority_queue.get()

        if current_node.value == goal:
            print("Path found:", current_node.value)
            return

        if current_node.value not in visited:
            print("Current Node:", current_node.value)
            visited.add(current_node.value)

            for neighbor, cost in graph[current_node.value]:
                neighbor_node = Node(neighbor, current_node.cost + cost, heuristic[neighbor])
                priority_queue.put(neighbor_node)

    print("Path not found.")

# Given tree represented as an adjacency list
graph = {
    'A': [('B', 4), ('C', 2)],
    'B': [('A', 4), ('D', 5), ('E', 12)],
    'C': [('A', 2), ('F', 3)],
    'D': [('B', 5)],
```

```
    'E': [('B', 12)],
    'F': [('C', 3)]
}

# Heuristic values
heuristic = {
    'A': 5,
    'B': 4,
    'C': 2,
    'D': 3,
    'E': 0,
    'F': 1
}

# Start and goal nodes
start_node = 'A'
goal_node = 'E'

a_star_search(graph, start_node, goal_node)
```

Output :

Current Node: A
Current Node: C
Current Node: F
Current Node: B
Current Node: D
Path found: E