Code Implementation:

Answer:

```
tree = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F', 'G'],
    'D': [],
    'E': [],
    'F': [],
    'G': [],
}

import heapq

def a_star(graph, start, goal, heuristic):
    open_set = [(0, start)]
    came_from = {}
    g_score = {start: 0}
    f_score = {start: heuristic(start, goal)}

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

        if current == goal:
            return reconstruct_path(came_from, start, goal)

        for neighbor in graph[current]:
            tentative_g_score = g_score[current] + cost(current, neighbor)
            if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = g_score[neighbor] + heuristic(neighbor, goal)
                if neighbor not in open_set:
                    heapq.heappush(open_set, (f_score[neighbor], neighbor))

    return None

def reconstruct_path(came_from, start, goal):
    path = []
    current = goal
    while current != start:
        path.append(current)
        current = came_from[current]
```

Lab task:


```
    path.append(start)
    return path[::-1]

def neighbors(node):
    return tree[node]

def cost(node1, node2):
    return 1

def heuristic(node, goal):

    return abs(ord(node) - ord(goal))

start = 'A'
goal = 'G'
path = a_star(tree, start, goal, heuristic)
if path:
    print("Path found:", path)
else:
    print("No path found.")
```