```
import heapq

# define the graph with edge costs
graph = {
    'A': {'B': 1, 'C': 3},
    'B': {'D': 1},
    'C': {'D': 1},
    'D': {},
    'E': {}
}

# heuristic values for each node (h(n))
h = {
    'A': 3,
    'B': 2,
    'C': 1,
    'D': 1,
    'E': 0  # heuristic is goal is zero
}

# AF variables
start = 'A'
goal = 'E'
g_scores = {start: 0}
parents = {start: None}
visited = set()
openlist = []


# initial node: f(n) = g(n) + h(n)
heapq.heappush(openlist, (h[start], start, 0)) # (f(n), node, g(n))

while openlist:
    _, node, g = heapq.heappop(openlist)
    print(f"Visiting: {node}, h={h[node]}, g={g}")

    if node in visited:
        continue
    visited.add(node)

    if node == goal:
        print("Goal reached!")
        break

    for neighbor, cost in graph.get(node, {}).items():
        g_neighbor = g + cost
        fscore = g_neighbor + h[neighbor]

        if neighbor not in visited or g_neighbor < g_scores.get(neighbor, float('inf')):
            g_scores[neighbor] = g_neighbor
            parents[neighbor] = node
            heapq.heappush(openlist, (fscore, neighbor, g_neighbor))

# path reconstruction
path = []
current = goal
while current is not None:
    path.append(current)
    current = parents.get(current)
path.reverse()

print("\nShortest path from", start, "to", goal, "is:")
print(" -> ".join(path))
print("Total cost (g):", g_scores[goal])
```