In [22]:
```
import heapq

def a(graph,h,start,goal):
    open_list=[]
    heapq.heappush(open_list,(0+h[start],start))
    g_nodes={start:0}
    parents={start:None}
    while open_list:
        current_f,current_node=heapq.heappop(open_list)
        if current_node==goal:
            path=[]
            while current_node:
                path.append(current_node)
                current_node=parents[current_node]
            path.reverse()
            return path,g_nodes[goal]

        for neighbor,cost in graph[current_node]:
            g_cost=g_nodes[current_node]+cost

            if neighbor not in g_nodes or g_cost<g_nodes[neighbor]:
                g_nodes[neighbor]=g_cost
                f_score=g_cost+h[neighbor]
                heapq.heappush(open_list,(f_score,neighbor))
                parents[neighbor]=current_node
    return None,float('inf')
```

In [23]:
```
graph = {
    'a': [('c', 4), ('d', 1)],
    'b': [('e', 3), ('f', 1)],
    'c': [],
    'd': [],
    'e': [('h', 5)],
    'f': [('i', 2), ('g', 3)],
    'g': [],
    'h': [],
    'i': [],
    's': [('a', 3), ('b', 2)]
}

heuristic = {
    'a': 12,
    'b': 4,
    'c': 7,
    'd': 3,
    'e': 8,
    'f': 2,
    'h': 4,
    'i': 9,
    's': 13,
    'g': 0
}
```

In [24]:
```
start_node = 's'
goal_node = 'g'

path, cost = a(graph, heuristic, start_node, goal_node)

if path:
    print(f'Path: {" -> ".join(path)}')
    print(f'Cost: {cost}')
else:
    print("No path found")
```

```
Path: s -> b -> f -> g
Cost: 6
```

In [ ]:
```

```