```python
import heapq

# Define the graph with nodes, edges, costs, and heuristics
graph = {
    'A': {'B': 6, 'F': 3},
    'B': {'A': 6, 'D': 2,'C':3},
    'C': {'D': 1, 'B':3,'E':5},
    'D': {'C': 1,'E':8,'B': 2 },
    'E': {'C': 5, 'D':8,'J':5},
    'F': {'A': 3,'G':1,'H':3},
    'G': {'F': 1,'I':1},
    'H': {'F': 7,'I':2},
    'I': {'E': 5,'J':3,'G':3,'H':2},
}

# Heuristic values for each node
heuristic = {
    'A': 10,
    'B': 8,
    'C': 5,
    'D': 7,
    'E': 3,
    'F': 6,
    'G': 5,
    'H': 3,
    'I':1,
    'J': 0
}

def astar(graph, heuristic, start, goal):
    frontier = []
    heapq.heappush(frontier, (0, start))  # Priority queue sorted by cost
    came_from = {}
    cost_so_far = {start: 0}

    while frontier:
        current_cost, current_node = heapq.heappop(frontier)

        if current_node == goal:
            path = []
            while current_node in came_from:
                path.append(current_node)
                current_node = came_from[current_node]
            path.append(start)
            path.reverse()
            return path

        for next_node, cost in graph[current_node].items():
            new_cost = cost_so_far[current_node] + cost
            if next_node not in cost_so_far or new_cost < cost_so_far[next_node]:
                cost_so_far[next_node] = new_cost
                priority = new_cost + heuristic[next_node]
                heapq.heappush(frontier, (priority, next_node))
                came_from[next_node] = current_node

    return None

# Find the shortest path from node A to node J
start_node = 'A'
goal_node = 'J'
shortest_path = astar(graph, heuristic, start_node, goal_node)

if shortest_path:
    print("Shortest Path from", start_node, "to", goal_node, ":", shortest_path)
else:
    print("No path found from", start_node, "to", goal_node)
```

```
Shortest Path from A to J : ['A', 'F', 'G', 'I', 'J']
```