Assignment 1

```
from heapq import heappush, heappop

def dijkstras_shortest_path(graph, source, destination):
  """
  This function implements Dijkstra's algorithm to find the shortest path between
  two nodes in a graph represented as a dictionary.

  Args:
     graph: A dictionary where keys are nodes and values are dictionaries of
          neighboring nodes and their weights.
     source: The starting node.
     destination: The destination node.

  Returns:
     A list containing the nodes in the shortest path from source to destination,
     or None if no path exists.
  """
  # Initialize distances for all nodes as infinite.
  distances = {node: float('inf') for node in graph}
  distances[source] = 0

  # Priority queue for keeping track of unvisited nodes with their current distances.
  pq = [(0, source)]

  while pq:
    current_distance, current_node = heappop(pq)

    # If the destination is reached, reconstruct the path and return it.
    if current_node == destination:
      path = []
      while current_node != source:
        path.append(current_node)
        current_node = distances[current_node]
      path.append(source)
      return path[::-1]

    # Iterate through neighbors of the current node.
    for neighbor, weight in graph[current_node].items():
      new_distance = current_distance + weight
      if new_distance < distances[neighbor]:
        distances[neighbor] = new_distance
        heappush(pq, (new_distance, neighbor))

  # No path found.
  return None

# Example usage
graph = {
  "A": {"B": 1},
  "B": {"C": 3, "E": 3.5},
  "C": {"E": 4, "D": 2.5},
  "D": {"G": 2.5},
  "G": {"F": 3.5},
  "E": {"F": 2},
  "F": {"H": 2.5},
  "H": {"I": 1},
}

source = "C"
destination = "F"

shortest_path = dijkstras_shortest_path(graph, source, destination)

if shortest_path:
  print("Shortest path from", source, "to", destination, ":", shortest_path)
else:
  print("No path found between", source, "and", destination)
```