```
import heapq

class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, u, v, weight):
        if u not in self.graph:
            self.graph[u] = []
        self.graph[u].append((v, weight))

    def dijkstra(self, start):
        distances = {vertex: float('inf') for vertex in self.graph}
        distances[start] = 0
        pq = [(0, start)]
        visited = set()

        while pq:
            current_distance, current_vertex = heapq.heappop(pq)

            if current_vertex not in visited:
                visited.add(current_vertex)

                if current_vertex in self.graph:
                    for neighbor, weight in self.graph[current_vertex]:
                        distance = current_distance + weight
                        if distance < distances[neighbor]:
                            distances[neighbor] = distance
                            heapq.heappush(pq, (distance, neighbor))

        return distances

# Example usage
g = Graph()
g.add_edge('A', 'B', 4)
g.add_edge('A', 'C', 2)
g.add_edge('B', 'C', 1)
g.add_edge('B', 'D', 5)
g.add_edge('C', 'D', 8)

start_vertex = 'A'
distances = g.dijkstra(start_vertex)
```

```
print("Shortest distances from vertex", start_vertex)
for vertex, distance in distances.items():
    print(vertex, ":", distance)
```