D.5 Schreiben Sie den Algorithmus `dijkstra` so zu `dijkstra_pq` um, dass die Menge Q als Min Priority Queue repräsentiert wird (vgl. Skript). Statt einer aufwändigen Aktualisierung der Schlüsselwerte in Q fügen Sie einfach den gleichen Knoten mit kleinerem Schlüsselwert der Queue erneut hinzu.

Argumentieren Sie, weshalb Ihr Algorithmus terminiert und führen Sie eine Laufzeitanalyse durch.

```
4   # Verbesserung von dijkstra mittels MinPrioQueue
5   from heapq import heappush, heappop
6
7   def dijkstra_pq(G,s):
8       m = len(G)                # O(1)
9       d = [None]*m              # O(m)  (geschaetzte) Pfadkosten
10      d[s] = 0                  # O(1)   sind 0 fuer den Startknoten
11      Q = []                    # O(1)   Min-Priority-Queue mit Schätzwerten als Schlüssel
12      visited = set()           # O(1)   Menge aller bereits besuchten Knoten
13
14      heappush(Q,(0,s))         # O(1)   Initialisierung mit s
15
16      while Q:                                    # O(m) Iterationen
17          # Auswahl von v als Knoten aus Q mit kleinstem Schlüsselwert
18          (dist,v) = heappop(Q)                   # O(log m)
19
20          # Überprüfe ob Pfadkosten aktualisiert werden müssen
21          if d[v] == None or d[v] > dist:         # O(1)
22              d[v] = dist
23
24          # Nachfolger der nicht-besuchten Knoten in Heap pushen
25          if not v in visited:                    # O(1)
26              for u in G[v]:                      # O(deg(v))
27                  heappush(Q, (dist + G[v][u], u))    # O(log m)
28
29          # v als besucht eintragen
30          visited.add(v)                          # O(1)
31
32      return d
```

Der Algorithmus terminiert, da in Zeile 25 sichergestellt wird, dass alle Knoten nur einmal betrachtet werden. Da in jedem Schleifendurchlauf ein Element aus der Queue entfernt wird, terminiert die while-Schleife nach $O(m)$ Iterationen und somit terminiert auch der Algorithmus als solches.

Die Laufzeit-Analyse ergibt:

$$O(1) + O(m) + O(m) \cdot \big(O(\log m) + O(\deg(v) \cdot O(\log m)\big)$$

Das Hinzufügen von Elementen in die Queue, wir pro Kante (v, u) über alle Iterationen der While-Schleife hinweg einmal durchgeführt, nämlich wenn v aus Q entfernt wird – also $O(k)$ oft. Somit ergibt sich als obere Schranke für die Laufzeit:

$$O(m \cdot \log m + k \cdot \log m)$$