D.6 Schreiben Sie den Algorithmus `prim` so zu `prim_pq` um, dass die Menge Q als Min Priority Queue repräsentiert wird. Statt einer aufwändigen Aktualisierung der Schlüsselwerte in Q fügen Sie einfach den gleichen Knoten mit kleinerem Schlüsselwert der Queue erneut hinzu.

```python
# Verbesserung von prim mittels MinPrioQueue
from heapq import heappush, heappop

def prim_pq(G):
    m = len(G)
    # (geschaetzte) Kosten für die Anbindung von Knoten an den Spannbaum
    # mittels einer weiteren Kante
    T = set()           # Ergebnis: Spannbaum als Kantenmenge
    c = 0               # und dessen Kosten
    Q = []              # Min-Priority-Queue mit Schätzwerten als Schlüssel
    visited = set()     # Menge aller bereits besuchten Knoten

    # Proirity Queue enthält Tupel aus Schätzwert, aktueller Knotennummer und Vorgänger
    heappush(Q,(0,0,None))      # Initialisierung mit Knoten 0

    while Q:
        # Auswahl von v als Knoten aus Q mit kleinstem Schlüsselwert
        (cost, v, p_v) = heappop(Q)

        # Überprüfe ob Knoten noch betrachtet werden muss
        if not v in visited:
            # Kante in Spannbaum hinzufügen
            T.add(frozenset((p_v,v)))
            c += cost
            # Nachfolger in Queue pushen
            for u in G[v]:
                heappush(Q, (G[v][u], u, v))

        # v als besucht eintragen
        visited.add(v)


    return T-{frozenset((None,0))},c
```