B.9:

Ändern Sie den Algorithmus `bfs` aus der Vorlesung so in `bfs_queue` um, dass statt der Liste L mit den Mengen L_i nur eine einzige Queue Q verwendet wird, um die besuchten Knoten zu verwalten. Geben Sie die Menge der erreichten Knoten und die Liste `parent` zurück.

```python
from collections import deque

# BFS als Queue
def bfs_queue(G,s):
    m = len(G)
    Q = deque({s})          # Queue der Schichten, mit Startknoten initialisiert
    Nodes = set({s})        # Menge der erreichten Knoten, mit Startknoten initialisiert
    reached = [False]*m     # bereits besuchte Knoten
    reached[s] = True       #
    parent = [None]*m       # Vorgaenger im BFS-Baum

    while len(Q) != 0:          # solange Queue nicht leer ist
        current_node = Q.pop()  # zu betrachtender Knoten
        for child_node in G[current_node]:  # alle Nachbarn von current_node aus erreichbaren Knoten
            if not reached[child_node]:     # falls neuer Knoten
                Nodes.add(child_node)
                Q.appendleft(child_node)    # neuer Knoten an Anfang der Queue -> wird erst nach allen Knoten
                                            # der aktuellen Schicht betrachtet
                reached[child_node] = True
                parent[child_node] = current_node

    return Nodes, parent
```

```python
G = define_G()
bfs_queue(G,0)


# Vergleich mit bfs aus VL
from vl.lek03.bfs import bfs

for s in range(len(G)):
    print('s = ', s)
    L = bfs(G,s)[0]  # erste Komponente ist Liste der L_i
    # alle L_i vereinigen
    R = { u for i in range(len(L)) for u in L[i]}
    # Vergleich
    print(R == bfs_queue(G,s)[0])
    print(bfs(G,s)[1] == bfs_queue(G,s)[1])
```