Breadth-First-Durchlauf eines Graphen

Schreib eine Funktion, die die Knoten eines gerichteten Graphen *in Breitensuch-Reihenfolge zu einer Liste verkettet*. Weiters soll in jedem Knoten seine *Tiefe* (kleinste Anzahl der Kanten vom Startknoten zum Knoten) gespeichert werden.
Der Graph ist intern mit *Adjazenzlisten der ausgehenden Kanten* gespeichert:

- Jeder *Knoten* ist eine *Struktur* **`node`** mit 3 Membern:
  Dem Zeiger auf seine *Adjazenzliste* (**`edges`**),
  dem Verkettungs-Zeiger (**`next`**) für die zu erstellende BFS-Liste
  (= Zeiger auf den *nächsten Knoten* in Breadth-First-Reihenfolge),
  und einem **`int`**-Member **`level`** für die zu berechnende *Tiefe*.
- Die *Kanten-Strukturen* **`edge`** in den Adjazenzlisten bestehen aus
  einem Zeiger **`neighbor`** auf den *Zielknoten* der Kante
  und der *Listenverkettung* **`next`** der Adjazenzliste.

Die Funktion **`bfs`** wird mit einem *Pointer auf den Startknoten* des BFS-Durchlaufes aufgerufen. Sie soll alle vom Startknoten ausgehend erreichbaren Knoten des Graphen nach dem im Skriptum angegebenen Verfahren *in BFS-Reihenfolge besuchen*
und dabei drei Dinge erledigen:

- *Mitzählen* und als *Returnwert* zurückgeben, *wie viele Knoten* sie besucht hat.
- Das *Tiefen-Feld* **`level`** der besuchten Knoten befüllen:
  Wenn man einen noch nie gesehenen Knoten *erstmals* über eine Kante erreicht,
  dann ist seine Tiefe *eins höher als die eigene Tiefe*.
  Dieses Tiefen-Feld übernimmt auch die Rolle der *"visited"-Markierung*
  im Algorithmus: Die Funktion darf sich darauf verlassen,
  dass die Tiefe aller Knoten vor dem Aufruf auf **`-1`** *initialisiert* wurde.
- Die Knoten über ihr **`next`**-Member in BFS-Reihenfolge
  beginnend mit dem Startknoten zu einer Liste *verketten*.
  Diese Liste übernimmt die *Funktion der internen Queue* des BFS-Algorithmus,
  es ist daher keine separate Queue erforderlich: Jeder Knoten wird
  hinten an diese Liste gehängt, wenn er das erste Mal erreicht wird.
  Die Schleife in der **`bfs`**-Funktion *durchläuft* diese Liste, ohne sie abzubauen.
- Wird **`NULL`** als Startknoten übergeben, soll die Funktion sofort **`0`** returnieren.

Du kannst entweder mein **Hauptprogramm** verwenden oder es selbst versuchen:

- Das Programm wird mit *zwei Zahlen* auf der Befehlszeile aufgerufen:
  Der *Anzahl der Knoten* im Graph und der *Nummer des Startknotens*
  (die Knoten sind von **`0`** bis **`n-1`** durchnummeriert).
- Die *Knoten* werden intern in einem *Array* gespeichert,
  das Array wird *dynamisch* in der richtigen Größe angelegt.
  Die *Tiefe* aller Knoten wird auf **`-1`** initialisiert, die *BFS-Verkettung* auf **`NULL`**.
- Die *Kanten* des Graphen werden *vom Terminal eingelesen* und in *Adjazenzlisten*
  (mit einzeln dynamisch angelegten Listenelementen) gespeichert:
  Die Eingabe der Kanten erfolgt in der Reihenfolge der Ausgangsknoten
  (zuerst die Kanten ausgehend von Knoten 0, dann jene von Knoten 1, usw.).

Eine Kante wird durch die *Knotennummer des Zielknotens* eingegeben,
die Liste der Kanten jedes Knotens wird durch die Eingabe von **−1** abgeschlossen.

Die *Reihenfolge* der Nachbarn eines Knotens in der Eingabe muss erhalten bleiben
und beim Breadth-First-Durchlauf berücksichtigt werden
(der zuerst eingegebene Nachbar wird zuerst besucht).

- Dann wird die *BFS-Funktion* mit dem Startknoten aufgerufen
  und ihr Ergebnis ausgegeben.
- Am *Ende* des Programmes werden die Knoten *in Verkettungs-Reihenfolge*
  (d.h. Breadth-First-Reihenfolge, beginnend mit dem Startknoten) ausgegeben,
  für jeden Knoten wird seine *Nummer* und seine *Tiefe* angezeigt.

  Knoten, die vom Startknoten aus *nicht erreichbar* sind, bleiben unberücksichtigt.