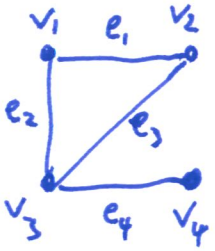
Wie wir im nächsten Abschnitt sehen werden, lohnt sich dafür eine eigene Notation:

Die Kantenliste benötigt $\Theta(m \log n)$ Speicherplatz.

(4) Adjazenzliste



V_1 : V_2, V_3 ;
V_2 : V_1, V_3 ;
V_3 : V_1, V_2, V_4 ;
V_4 : V_3 ;

Das ist etwas praktischer als die Kantenliste, wenn man für Graphenalgorithmen direkten Zugriff auf die Nachbarn eines Knotens benötigt! Man muss nicht die Nachbarn erst mühsam aus einer Liste herraussuchen.

Länge:

Jede Kante taucht doppelt auf, einmal für jeden Knoten.

So also:

$$2n + 4m + n(\lfloor \log_{10} n \rfloor + 1) + 2m(\lfloor \log_{10} n \rfloor + 1)$$

- d.h. $\Theta(n \log n + m \log n)$.

Im allgemeinen sind Graphen mit vielen isolierten Knoten (ohne Kanten!) uninteressant, d.h. z.B. $m \geq \frac{n}{2}$

$m \geq n$ o.ä.

Also wieder $\Theta(m \log n)$.

Jetzt wollen wir noch etwas mehr: den direkten Zugriff auf die Nachbarn der Knoten, wenn wir sie brauchen!

Also:

Liste (!) $\quad \underset{\downarrow}{V_2}, V_3 ; \underset{\downarrow}{V_1}, V_3 ; \underset{\downarrow}{V_1}, V_2, V_4 ; \underset{\downarrow}{V_3}$

Zugriff auf Nachbarn jeweils ab Semikolon.

Algorithmisch: Gehe Liste durch, zähle Semikolons → das dauert

Datenstruktur: Speichere die Stelle ab, an der die Nachbarn von V_3 zu finden sind.

→ Zeiger (engl. "Pointer")

Bekannt bei Webseiten:

Speicherinhalt: z.B. YouTube-Video (etliche MB)

Zeiger: URL (einige Byte)

„Man muss nicht alles wissen, man muss nur wissen wo's steht!"

Für den direkten Zugriff brauchen wir n Zeiger; jeder codiert eine Speicherzelle, d.h. die Nummer eines Bits in der Liste.

So ein Zeiger braucht also selber

$$\log_2 \left\lfloor \left(2n + 4m + n\left(\lfloor \log_2 n \rfloor + 1\right) + 2m\left(\lfloor \log_2 n \rfloor + 1\right)\right)\right\rfloor + 1$$

für $n \geq 10$, $m \geq n$

$$\leq \log_2 \left(9m\left(\log_2 n + 1\right) + 1\right)$$

$$\leq \log_2 9 + \log_2 m + \log_2\left(\log_2 n + 1\right) + 1 \leq 2\log_2 m$$

Bits

Insgesamt benötigen wir also

$$\Theta(n \log_2 m) \quad \text{Bits.}$$

Jetzt ist $m \leq n^2$,

also $\log_2 m \leq \log_2 n^2 = 2 \log_2 n$,

d.h. $n \log m \leq 2n \log n$,

und der insgesamt verbrauchte Speicherplatz ist

$$\Theta(n \log m + m \log n) = \Theta(m \log n).$$

3.7 Wachstum von Funktionen

Im letzten Abschnitt haben wir Funktionen abgeschätzt und auf die „wesentlichen" Bestandteile reduziert, um Größenordnungen und Wachstumsverhalten zu beschreiben. Ein bisschen formaler:

Definition 3.9 (Θ-Notation)

Seien $f, g : \mathbb{N} \to \mathbb{R}$ Funktionen.

Dann gilt $f \in \Theta(g) :\Leftrightarrow$ Es gibt positive Konstanten c_1, c_2, n_0 mit

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ für alle } n \geq n_0.$$

Man sagt: f wächst asymptotisch in derselben Größenordnung wie g.

Beispiele: $2n^2 - 1 \in \Theta(n^2)$

$$\frac{n^3}{1000} + n^2 + n \log n \in \Theta(n^3)$$

3.8 Laufzeit von BFS und DFS

Wenn man einen Graphen durchsucht, sollte man schon alle Knoten und alle Kanten ansehen; also lässt sich eine untere Schranke von $\Omega(n+m)$ nicht unterbieten.
Tatsächlich wird diese Schranke auch erreicht:

SATZ 3.13

Der Graphen-Scan-Algorithmus 3.7 lässt sich so implementieren, dass die Laufzeit $O(n+m)$ ist.

Beweis:

Wir nehmen an, dass G durch eine Adjazenzliste gegeben ist.

Für jeden Knoten verwenden wir einen Zeiger, der auf die „aktuelle" Kante für diesen Knoten in der Liste zeigt (d.h. auf den „aktuellen" Nachbarn).
Anfangs zeigt akt(x) auf das erste Element in der Liste.
In (2.3.1) wird der aktuelle Nachbar ausgewählt und der Zeiger weiterbewegt; wird das Listenende erreicht, wird x aus R entfernt und nicht mehr eingeführt.
Also ergibt sich eine Gesamtlaufzeit von $O(n+m)$. □

KOROLLAR 3.14

Mit Algorithmus 3.7 kann man alle Zusammenhangskomponenten eines Graphen berechnen.

Beweis:

Wende 3.7 an und überprüfe, ob $Y=V$ ist. Falls ja, ist der Graph zshgd.
Falls nein, haben wir eine Zusammenhangskomponente berechnet;
wir lassen den Algorithmus erneut für einen Knoten $s \in V \setminus Y$ laufen, usw.
Wieder wird keine Kante von einem Knoten aus doppelt angefasst,
also bleibt die Laufzeit linear, d.h. $O(n+m)$. □