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

Also:

Liste: $v_2, v_3; v_1, v_3; \underset{\downarrow}{v_1, v_2, v_3}; v_4$

07.12.11

Zugriff auf Nachbarn von v_3 ab zweitem Semikolon!

Algorithmisch: Gehe Liste durch, zähle Semikolons → das kann dauern!

Datenstruktur: Speichere die Stelle ab, an der die Nachbarn von v_3 zu finden sind

↳ Zeiger (oder auch „Pointer")

Unterschied bei Webseiten:

Speicherinhalt: z.B. Video auf YouTube (etliche Megabyte)

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 der Liste

So ein Zeiger braucht also selber

$$\log_2(2n + 4m + n\log_{10} n + 2n\log_{10} n)$$

für $n \geq 10$, $m \geq n$

$$\leq \log_2(9m\log_{10} n) \leq \log_2 9m + \log_2\log_{10} n$$

$$\leq \log_2 9m + \log_2 \log_{10} n$$

$$\leq \log_2 9 + \log_2 m + \log_2 \log_{10} n$$

$$\leq 2 \log_2 m \quad \text{Bits}$$

- also insgesamt

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

Jetzt ist

$$m \leq n^2,$$

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

d.h. $n \log_2 m \leq 2n \log_2 n$,

und der insgesamt benötigte Speicherplatz ist

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

3.5 Wachstum von Funktionen

Im letzten Abschnitt haben wir Funktionen abgeschätzt und auf ihre „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

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

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

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

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

Manchmal hat man nur obere oder untere Abschätzungen:

Definition 3.10 (O-Notation)

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

Dann gilt

$f \in O(g) :\Leftrightarrow$ Es gibt positive Konstanten c, n_0 mit

$$0 \le f(n) \le c g(n) \text{ für alle } n \ge n_0.$$

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

Beispiele:

$2n^2 - 1 \in O(n^2)$
$2n^2 - 1 \in O(n^3)$
$n \log n \in O(n^2)$

Definition 3.11 (Ω-Notation)

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

Dann gilt $f \in \Omega(g) :\Leftrightarrow$ Es gibt positive Konstanten c, n_0 mit

$0 \leq c\, g(n) \leq f(n)$ für alle $n \geq n_0$.

Einige einfache Eigenschaften:

Satz 3.12

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

Dann gilt

(i) $f \in \Theta(g) \Leftrightarrow g \in \Theta(f)$

(ii) $f \in \Theta(g) \Leftrightarrow f \in O(g)$ und $f \in \Omega(g)$

(iii) $f \in O(g) \Leftrightarrow g \in \Omega(f)$

Beweis:

Übung!

3.6 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 x 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 (4) wird der aktuelle Nachbar ausgewählt und der Zeiger weiterbewegt; wird das Listenende erreicht, wird x aus Q entfernt und nicht mehr eingeführt.

Also ergibt sich eine Gesamtlaufzeit von $O(n+m)$. □