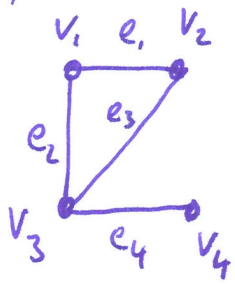
27.11.07

(3) Kantenliste:



$$\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_4\}$$

Benötigt wird Kantennummerierung!

↳ Jeder Index ist eine Binärzahl mit $\log_2 n$ Bits, oder eine Dezimalzahl mit $\log_{10} n$ Stellen.

Unterschied: Ein Faktor 3,3219...,

denn $\log_2 n = \log_2 10 \cdot \log_{10} n$

und $\log_2 10 = 3{,}3219...$!

Für einen Graphen mit m Kanten und n Knoten

ergibt sich in obiger Schreibweise dezimal ein Platzbedarf von

$$(6m-1) + 2m \log_{10} n.$$

Dabei kann man an ein paar Stellen sparen (z.B. "{" weglassen, "{" und "}" weglassen, "," durch Leerzeichen ersetzen),

andererseits kann man es auch teurer machen (z.B. "v" in ASCII codieren) oder anders codieren, etwa binär. So kann man auch

$$2m-1 + 2m \log_2 n$$ erhalten.

Was ist wirklich wichtig dabei?!

(i) Die Kantenliste ist sparsamer als die Inzidenzmatrix, denn wenn n nicht zu klein ist, ist

$$n \geq 2(1+\log_2 n)$$

(Was heißt „nicht zu klein"? $n \geq 8$ reicht!)

Also ist auch

$$mn > (2m+1) + 2m \log_2 n .$$

(ii) Unabhängig von der Codierung ist für die Größe des Speicherplatzes der zweite Ausdruck wichtig, denn

$$\begin{aligned} 2m \log_2 n &\leq (2m-1) + 2m \log_2 n \\ &\leq 2m(\log_2 n + 1) \leq 4m \log_2 n \end{aligned}$$

für $n \geq 2$.

(iii) Letztlich kommt es also gar nicht so sehr auf die Vorfaktoren an (die sind Codierungsabhängig!), sondern auf den Ausdruck

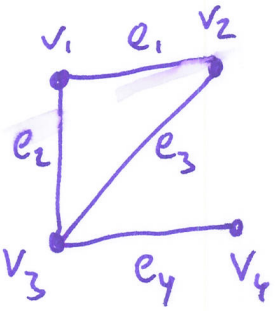
$$m \log n$$

(iv) In $m \log n$ steckt das Wesen der Kantenliste: Zähle für alle m Kanten die Nummern der beteiligten Knoten auf!

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

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

(4) Adjazenzliste:



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

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 heraussuchen.

Länge: Jede Kante taucht doppelt auf, einmal für jeden Knoten

So also: $2n + 4m + n \log_{10} n + 2m \log_{10} n$

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

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

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

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

Also:

Liste: $v_2, v_3; v_1, v_3; \underbrace{v_1, v_2, v_3}; v_4$ (↓ über dem zweiten Semikolon)

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

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

für $n \geq 10$, $m \geq n$

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

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

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

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

- 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

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