28.11.07

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

$$f \in \Theta(g) :\Leftrightarrow \text{Es gibt positive Konstanten } c_1, c_2, n_0 \text{ mit } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ für alle } n \ge n_0.$$

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

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

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

Manchmal hat man nur obere oder untere Abschätzungen:

Definition 3.10 (O-Notation)

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

Dann gilt

$$f \in O(g) :\Leftrightarrow \text{Es gibt positive Konstanten } c, n_0 \text{ mit } 0 \le f(n) \le c g(n) \text{ für alle } n \ge n_0.$$

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 $O(n+m)$. □

Korollar 3.14

Mit Algorithmus 3.7 kann man alle Zusammenhangskomponenten eines Graphen bestimmen.

Beweis:

Wende 3.7 einmal an und überprüfe, ob $R = V$ ist.
Falls ja, ist der Graph zusammenhängend.
Falls nein, haben wir eine Zusammenhangskomponente identifiziert;
wir lassen den Algorithmus erneut für einen beliebigen
Knoten $s' \in V \setminus R$ laufen usw.

Wieder wird keine Kante doppelt angefasst,
also bleibt die Gesamtzeit linear, d.h. $O(n+m)$

□

Korollar 3.15

BFS und DFS haben Laufzeit $O(n+m)$.

Beweis:

Einfügen in Q lässt sich jeweils in Konstanter
Zeit vornehmen, der Rest überträgt sich von Satz 3.13

□

3.7 Besondere Eigenschaften von DFS und BFS

Einfach gesagt:

- DFS ist eine bestmögliche, individuelle Suchstrategie mit lokaler Information.
- BFS ist eine bestmögliche, kooperative Suchstrategie mit globaler Information.

Konkret:

- DFS ist gut geeignet für die Suche nach einem Ausweg aus einem Labyrinth.
- BFS ist gut geeignet für die Suche nach kürzesten Wegen in einem Graphen

Satz 3.16 (Lokale Suche mit DFS)

DFS ist eine optimale lokale Suchstrategie in folgendem Sinne:

(1) DFS findet in jedem Graphen mit n Knoten einen Weg der Länge höchstens $2n-1$, der alle Knoten besucht.

(2) Für jede lokale Suchstrategie gibt es einen Graphen mit n Knoten, so dass der letzte Knoten erst nach einer Weglänge von $2n-1$ besucht wird.

Beweis: Übung!

Für BFS zeigen wir, dass der zugehörige Baum tatsächlich kürzeste Wege im Graphen liefert. Für leichtere Argumentation gebrauchen wir dabei folgende leichte Modifikation von Algorithmus 3.7:

Algorithmus 3.17

Input: Graph $G=(V,E)$, Knoten s

Output:
- Knotenmenge $R \subseteq V$, die von s aus erreichbar ist;
- Für jeden Knoten $v \in R$ die Länge $d(s,v)$ eines kürzesten Weges von s nach v.
- eine Kantenmenge $T \subseteq E$, die die kürzestmöglichen Wege zu den Knoten von R sicherstellt, d.h. einen die Zusammenhangskomponente von s aufspannenden Baum (R,T), der kürzeste Wege von s zu allen Knoten in R liefert.

(1) Sei $R := \{s\}$, $Q := \{s\}$, $T := \emptyset$, $\ell(s) := 0$,

(2) WHILE $(Q \neq \emptyset)$ DO {

wähle erstes Element $v \in Q$

(3) IF (es gibt kein $w \in V \setminus R$ mit $e = \{v,w\} \in E$ THEN

$Q := Q \setminus \{v\}$

(4) ELSE {

wähle ein $w \in V \setminus R$ mit $e = \{v,w\} \in E$;
setze $R := R \cup \{w\}$, $T := T \cup \{e\}$, hänge w an Q an;
setze $\ell(w) := \ell(v) + 1$.

}

}

(5) STOP

Satz 3.18

(i) Verfahren 3.17 ist ein Algorithmus.

(ii) Die Laufzeit ist $O(n+m)$.

(iii) Am Ende ist für jeden erreichbaren Knoten $v \in R$ die Länge eines kürzesten Weges von s nach v im Baum (R,T) durch $\ell(v)$ gegeben.

(iv) Am Ende ist für jeden erreichbaren Knoten $v \in R$ die Länge eines kürzesten Weges von s nach v im Graphen (V,E) durch $\ell(v)$ gegeben.

Beweis:

(i) Wie für Algorithmus 3.7 gelten alle Eigenschaften; zusätzlich ist für jeden Knoten $w \in Q$ per Induktion der Wert $\ell(v)$ tatsächlich definiert.

(ii) Die Laufzeit bleibt von Algorithmus 3.7 erhalten.

(iii) Sei $d_{(R,T)}(s,v)$ die Länge eines kürzesten Weges von s nach v in (R,T). Dann zeigt man durch Induktion über $d_{(R,T)}(s,v)$, dass für alle Knoten $d_{(R,T)}(s,v) = \ell(v)$ gilt: