II.3.1 Rekursive Algorithmen

Dienstag, 1. Dezember 2020 16:02

Bisher:
iterative Algorithmen: mehrfaches Durchlaufen von Programmabschnitten mit Hilfe von Schleifen.
Verwenden oft Akkumulatorvariablen zur schrittweisen Berechnung des Ergebnisses (z.B. res)

Jetzt:
rekursive Algorithmen
Idee: Führe Problem für x zurück auf das Problem für kleineren Wert als x.

Bsp: mathematische Definition der Fakultät

$$fak(x) = \begin{cases} x \cdot fak(x-1), & \text{falls } x > 1 \\ 1, & \text{sonst} \end{cases}$$

Rekursive Definition, denn "fak" tritt in ihrer eigenen Def. auf.

$$fak(3) = 3 \cdot fak(2)$$
$$= 3 \cdot 2 \cdot fak(1)$$
$$= 3 \cdot 2 \cdot 1 = 6$$

- Lösung eines Problems durch rekursiven Algorithmus ist oft völlig analog zur Problemstellung
  (⇒ deklaratives Programmieren
  ⇒ fkt. Programmierung)
- Rekursive Methoden: rekursiver Aufruf mit Argumenten, die "kleiner" als die ursprüngliche Eingabe sind

Klassifikation verschiedener Arten von Rekursion

- lineare / nicht-lineare Rekursion
  linear: in jeder Ausführung des Methodenrumpfs höchstens ein rekursiver Aufruf (Bsp: fak)

linear: in jeder Ausführung des Methodenrumpfs höchstens ein rekursiver Aufruf (Bsp: fak)

nicht-linear: Bsp: fib

Bsp: Fibonacci-Zahlen

Ziel: Vorhersage von Kaninchen-Population

Vereinfachende Annahmen:

- Kaninchen werden nach 1 Monat geschlechtsreif.
- Tragezeit 1 Monat
- Bei jeder Geburt wird ein Kaninchenpaar geboren.
- Kaninchen sind unsterblich.

fib(x) = Anzahl der Kaninchenpaare in Monat x.

Alg. hat nicht-lineare Rekursion, ist aber sehr ineffizient:

fib(20) = fib(19) + fib(18)
= fib(18) + fib(17) + fib(18)
= fib(17) + fib(16) + fib(17) + fib(17) + fib(16)
= ....

fib(19) wird 1 mal ausg.
fib(18) wird 2 mal ausg.
fib(17) wird 3 mal ausg.

fib(16)	5
fib(15)	8
fib(14)	13

↑ Fibonacci-Zahlen

Zur Berechnung von fib(n) müssen "in etwa" 2^n Berechnungsschritte ausgeführt werden.

(Exponentieller Aufwand)

Es ginge auch mit linearem Aufwand.

- direkte/verschränkte Rekursion

direkte Rekursion: Algorithmus ruft sich selbst wieder rekursiv auf (z.B. fak, fib).

Verschränkte Rekursion: z.B. f ruft g auf,

Verschränkte Rekursion: z.B. f ruft g auf,
g ruft f auf

- Endrekursion (tail recursion)

Spezialfall der direkten Rekursion, bei dem rek. Aufrufe nur am Ende des Alg. auftreten.
D.h. weder in Teilausdrücken noch vor weiteren Anweisungen des Algorithmus.
Bsp. fak ist nicht endrekursiv, denn nach Beendigung des rek. Aufrufs muss man noch den ursprünglichen Wert von x kennen und damit multiplizieren.

Bsp: für endrekursive Methode: sqrt

Aufruf mit sqrt(0, x, x).
Führt Intervallschachtelung zur Berechnung von $\sqrt{x}$ durch

uG m oG

1. Fall $\sqrt{x}$ Neues Intervall: [uG, m]

2. Fall $\sqrt{x}$ Neues Intervall [m, oG]

In Schleifen: Werte der lokalen Variablen können in jedem Schleifendurchlauf überschrieben werden.

Bei Rekursion: Alte Werte der Variablen müssen erhalten bleiben, da sie nach dem rek. Aufruf evtl. noch gebraucht werden.
Ausnahme: Endrekursion. Hier dürfen lokale Variablen überschrieben werden.
=> lassen sich leicht in Schleifen überführen

Speicherorganisation bei Rekursion

- Für jeden Methodenaufruf wird ein neuer Speicherrahmen auf

- Für jeden Methodenaufruf wird ein neuer Speicherrahmen auf dem Kellerspeicher angelegt. Hier sind die Werte aller lokalen Variablen, inklusive formaler Parameter und Ergebnis (bei nicht-void Methoden) gespeichert.
- Alte Werte der Variablen werden evtl. nach Beendigung des rek. Aufrufs noch benötigt (Ausnahme: Endrekursion).
- Bei zu vielen rek. Aufrufen: Gefahr des Stack Overflows

Nachteil von Rekursion: benötigt mehr Speicherplatz

Vorteil von Rekursion: manche Algorithmen lassen sich rekursiv viel einfacher + kürzer formulieren

Bsp: Türme von Hanoi

Entwurfstechnik Divide and Conquer (Teile und herrsche)

1. Behandle einfache Fälle
   (Bsp: Verschiebe einen Turm der Höhe h=0.)
2. Divide: Teile das Problem in 2 oder mehr Teilprobleme auf.
   (Bsp: Teile Problem in Verschiebung von Türmen der Höhe h-1 und der Höhe 1.)
3. Conquer: Löse die Teilprobleme (typischerweise rekursiv)
4. Kombiniere die Teillösungen zur Gesamtlösung

Divide + Conquer in unserem Bsp: