Rekursionsformeln

- Falls Algorithmen sich selbst rekursiv aufrufen, so kann ihr Laufzeitverhalten bzw. ihr Speicherplatzbedarf in der Regel durch eine Rekursionsformel (recurrence, RF) beschrieben werden.
- Beispiel: MergeSort.
  - Sei T(n) die (asymptotische) Laufzeit des MergeSort-Algorithmus für die Eingabegröße n:

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & n > 1 \end{cases}$$

Rekursionsformeln

- Bei der Lösung solcher Rekursionsformeln vernachlässigen wir oft gewisse technische Details:
- Z.B. das Auf- und Abrunden der Größen der rekursiven Teilprobleme:
- Beispiel: MergeSort.

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 2T(n/2) + \Theta(n) & n > 1 \end{cases}$$

- Wenn die Laufzeit für die Eingabegröße 1 in $\Theta(1)$ ist (also eine Konstante), schreiben wir in der Regel:

$$T(n) = \begin{cases} c_1 & n = 1 \\ 2T(n/2) + \Theta(n) & n > 1 \end{cases}$$

- Oder noch einfacher:

$$T(n) = 2T(n/2) + \Theta(n)$$

Rekursionsformeln

- Wir werden im folgenden drei Verfahren zum Lösen von RF kennen lernen:
  - Substitutions-Methode (Ersetzungsmethode)
  - Rekursionsbäume
  - Master-Methode (Master-Theorem)

Substitutions-Methode

- Die Substitutions-Methode funktioniert wie folgt:
  - Man „errate“ die Form der Lösung.
  - Man verwende (vollständige) Induktion, um die Konstanten zu finden und um zu zeigen, dass es die Lösung ist.
- Beispiel:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

- Wir „glauben“, dass

  T(n) = O(n log(n)) die Lösung der RF ist.

- Daher versuchen wir zu zeigen, dass

  $T(n) \leq c\, n \log(n)$ für eine geeignet gewählte Konstante c > 0 ist.

Substitutions-Methode

5

- Wir nehmen an, dass die Aussage für $\lfloor n/2 \rfloor$ gilt:

$$T(\lfloor n/2 \rfloor) \le c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)$$

- Induktionsschluss:

$$\begin{aligned} T(n) &\le 2(c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)) + n \\ &\le cn\log(n/2) + n \\ &\le cn\log(n) - cn\log(2) + n \\ &\le cn\log(n) - cn + n \\ &\le cn\log(n) \qquad falls\_c > 1 \end{aligned}$$

Es fehlt der Induktionsanfang n = 1 (siehe nächste Seite)!

Substitutions-Methode

6

- Nehmen wir zunächst an, dass T(1) = 1 ist.
- Dann ist die Grenzbedingung für jede Konstante c > 0 verletzt, da

$$T(1) = 1 > c\log(1) = c * 0 = 0$$

- Der Induktionsanfang ist also nicht erfüllt.
- Wir können dieses Problem natürlich leicht umgehen, in dem wir ausnutzen, dass das asymptotische Laufzeit-verhalten nur für alle n größer gleich einer vorgegebenen ganzen Zahl n_0 gelten muss, d.h.,
- wir müssen nur zeigen, dass

  $T(n) \le cn\log(n)$ für alle $n \ge n_0$ gilt.

- Wir können also den Fall T(2)= 4 oder T(3)=5 als Basis-fall (Induktionsstart) für die vollständige Induktion wählen.
- Dann haben wir kein Problem eine geeignete Konstante c anzugeben.

Substitutions-Methode

- Wie kann man die Lösung der RF „erraten“?
  - Erfahrung: Ähnlichkeit mit bereits bekannten Rekursionsformeln.
  - Man verwende Rekursionbäume, um die asymptotische Größenordnung abzuschätzen.
  - Man verwende untere und obere Schranken, um die asymptotische Größenordnung immer weiter einzuschränken.

Substitutions-Methode

- Manchmal funktioniert die Abschätzung mittels der vollständigen Induktion nicht, obwohl man sich bezüglich der Schätzung sehr sicher ist.
- Beispiel:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

- Wir raten, dass T(n)= O(n), und wir wollen zeigen, dass T(n) $\leq$ cn ist:

$$T(n) = c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 = cn + 1$$

- Wir können das Problem lösen, in dem wir T(n) $\leq$ cn – b zeigen, wobei b > 0 eine Konstante ist.

Substitutions-Methode

$$T(n) \le c\lfloor n/2 \rfloor + c\lceil n/2 \rceil - 2b + 1$$
$$\le cn - 2b + 1$$
$$\le cn - b \qquad falls\_b \ge 1$$

In einigen Beispielen kann man die RF erst dann lösen, wenn man eine geeignete Variablensubstitution durchführt!

$$T(n) = 2T\left(\lfloor \sqrt{n} \rfloor\right) + \log(n)$$

Setze m = log(n):

$$T(2^m) = 2T\left(2^{m/2}\right) + m$$

Setze S(m) = $T(2^m)$:

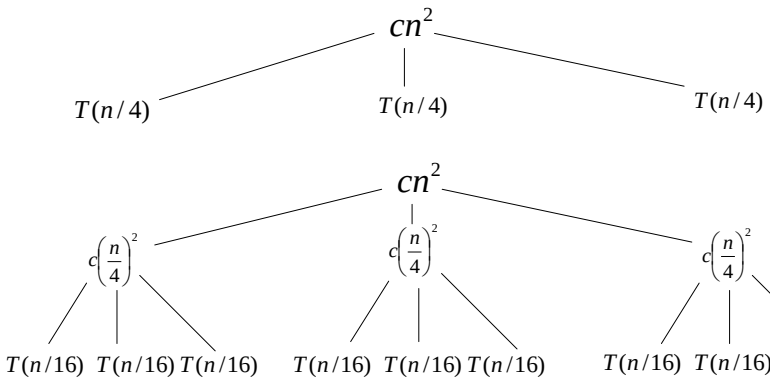
$$S(m) = 2S(m/2) + m \Rightarrow S(m) = O(m\log(m))$$
$$\Rightarrow T(n) = O(\log(n)\log\log(n))$$

Rekursionsbäume

- Rekursive Algorithmen zerlegen das zu lösende Problem in Teilprobleme.
- Die Gesamtlösung wird aus den Lösungen der Teilprobleme zusammengesetzt.
- Die Zerlegung in Teilprobleme kann mittels eines Rekursionsbaumes dargestellt werden.
- In einem solchen Baum repräsentiert jeder Knoten ein Teilproblem bzw. die Kosten eines Teilproblems.
- Rekursionsbäume sind hervorragend geeignet, um die asymptotischen Laufzeiten eines rekursiven Verfahrens abzuschätzen.
- Die asymptotische Laufzeit kann dann mittels der Substitutionsmethode bewiesen werden.

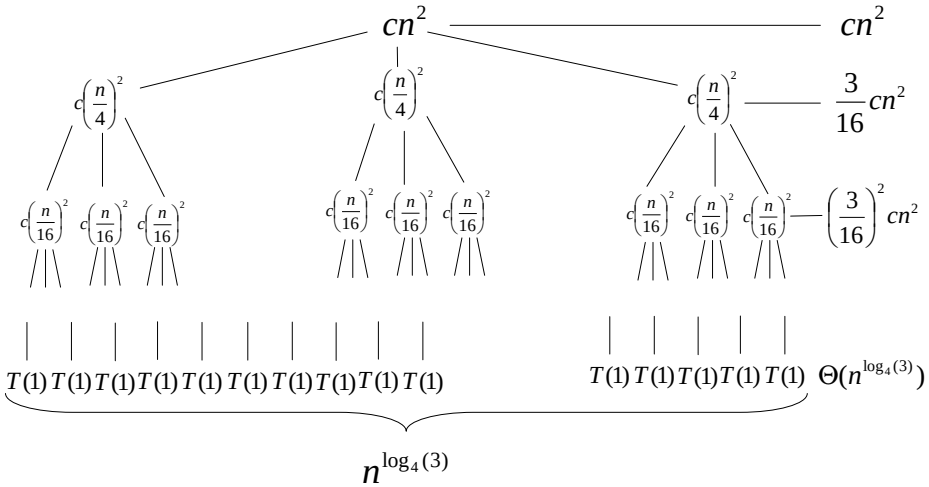
- Als Beispiel betrachten wir die Rekursion:

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2) \Rightarrow T(n) = 3T(n/4) + cn^2$$



- Als Beispiel betrachten wir die Rekursion:

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2) \Rightarrow T(n) = 3T(n/4) + cn^2$$



Rekursionsbäume

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4(n)-1} cn^2 + \Theta(n^{\log_4(3)})$$

$$T(n) = cn^2 \sum_{i=0}^{\log_4(n)-1} \left(\frac{3}{16}\right)^i + \Theta(n^{\log_4(3)})$$

$$T(n) = \frac{(3/16)^{\log_4(n)} - 1}{(3/16) - 1} cn^2 + \Theta(n^{\log_4(3)})$$

$$T(n) \leq 2cn^2 + \Theta(n^{\log_4(3)}) \qquad \text{für } n \geq 4$$

$$T(n) \in O(n^2)$$

Wir vermuten daher, dass T(n) = O(n²) ist, und zeigen nun mittels der Substitutionsmethode, dass T(n) ≤ dn² für ein geeignetes d > 0.

Rekursionsbäume

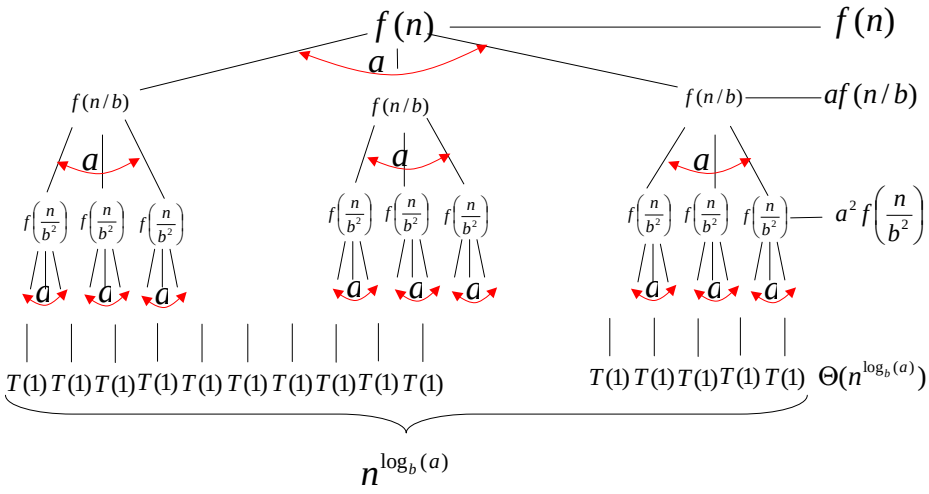
$$\begin{aligned} T(n) &\leq 3T(\lfloor n/4 \rfloor) + cn^2 \\ &\leq 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\leq 3d(n/4)^2 + cn^2 \\ &= \frac{3}{16}dn^2 + cn^2 \\ &\leq dn^2 \end{aligned}$$

Dieser letzte Schritt gilt für alle d mit $d \geq (16/13)c$!

Master-Methode

- Wir betrachten im folgenden Rekursionen der Form

$$T(n) = aT(n/b) + f(n) \qquad a \geq 1 \qquad b > 1 \qquad \forall n \geq n_0 : f(n) \geq 0$$



Master-Theorem

Summieren wir nun die Kosten aller Schichten des Baumes auf, so erhalten wir die folgenden Gesamtkosten.

$$T(n) = f(n) + af\left(\frac{n}{b}\right) + a^2 f\left(\frac{n}{b^2}\right) + \cdots + a^{\log_b(n)-1} f\left(\frac{n}{b^{\log_b(n)-1}}\right) + \Theta(n^{\log_b(a)})$$

$$T(n) = \sum_{i=0}^{\log_b(n)-1} a^i f\left(\frac{n}{b^i}\right) + \Theta(n^{\log_b(a)})$$

$$T(n) = g(n) + \Theta(n^{\log_b(a)}) \qquad g(n) = \sum_{i=0}^{\log_b(n)-1} a^i f\left(\frac{n}{b^i}\right)$$

Wir haben implizit in den obigen Berechnungen immer vorausgesetzt, dass n durch die Potenzen von b teilbar ist, d.h., dass n eine Potenz von b ist. Diese Voraussetzung behalten wir für die folgenden Rechnungen und Beweise bei. Die Berechnungen und Beweise für beliebige ganze Zahlen $n \in N$ können Sie im Lehrbuch „Introduction to Algorithms“ von Cormen et al. nachlesen.

Master-Theorem

Wir diskutieren im folgenden die potentiellen Lösungen von Rekursions-Formeln der allgemeinen Form T(n) = a T(n/b) + f(n) in Abhängigkeit vom asymptotischen Verhalten der Funktion f(n):

Hierbei betrachten wir zuerst den Fall, dass

$$f(n) = O(n^{\log_b(a)-\varepsilon}) \quad \text{für eine Konstante} \quad \varepsilon > 0$$

$$\Rightarrow f\left(\frac{n}{b^i}\right) = O\left(\left(\frac{n}{b^i}\right)^{\log_b(a)-\varepsilon}\right)$$

D.h., es existiert eine Konstante c > 0 und eine natürliche Zahl n_0, so dass für alle $n \geq n_0$ gilt:

$$f\left(\frac{n}{b^i}\right) \leq c\left(\frac{n}{b^i}\right)^{\log_b(a)-\varepsilon}$$

Master-Theorem

Für alle $n \geq n_0$ gilt folglich:

$$\sum_{i=0}^{\log_b(n)-1} a^i f\left(\frac{n}{b^i}\right) \leq c \sum_{i=0}^{\log_b(n)-1} a^i \left(\frac{n}{b^i}\right)^{\log_b(a)-\varepsilon}$$

$$= cn^{\log_b(a)-\varepsilon} \sum_{i=0}^{\log_b(n)-1} \left(\frac{ab^\varepsilon}{b^{\log_b(a)}}\right)^i$$

$$= cn^{\log_b(a)-\varepsilon} \sum_{i=0}^{\log_b(n)-1} b^{\varepsilon i} = cn^{\log_b(a)-\varepsilon} \frac{b^{\varepsilon \log_b(n)} - 1}{b^\varepsilon - 1}$$

$$= cn^{\log_b(a)-\varepsilon} \frac{n^\varepsilon - 1}{b^\varepsilon - 1}$$

Da b und ε Konstanten sind, folgt aus der obigen Abschätzung:

$$\sum_{i=0}^{\log_b(n)-1} a^i f\left(\frac{n}{b^i}\right) = O\left(n^{\log_b(a)}\right) \Rightarrow \quad T(n) = \Theta\left(n^{\log_b(a)}\right)$$

Master-Theorem

Wir betrachten nun den zweiten Fall und nehmen an, dass

$$f(n) = \Theta(n^{\log_b(a)})$$

$$\Rightarrow f\left(\frac{n}{b^i}\right) = \Theta\left(\left(\frac{n}{b^i}\right)^{\log_b(a)}\right)$$

Für alle hinreichend großen n und eine geeignete Konstante c > 0 gilt:

$$\sum_{i=0}^{\log_b(n)-1} a^i f\left(\frac{n}{b^i}\right) \geq c \sum_{i=0}^{\log_b(n)-1} a^i \left(\frac{n}{b^i}\right)^{\log_b(a)} = cn^{\log_b(a)} \sum_{i=0}^{\log_b(n)-1} \left(\frac{a}{b^{\log_b(a)}}\right)^i$$

$$= cn^{\log_b(a)} \sum_{i=0}^{\log_b(n)-1} 1 \qquad = cn^{\log_b(a)} \log_b(n)$$

Analog kann man zeigen, dass es eine Konstante d > 0 gibt, so dass für alle hinreichend großen n gilt:

$$g(n) \leq dn^{\log_b(a)} \log_b(n) \Rightarrow T(n) = \Theta(n^{\log_b(a)} \log_b(n))$$

Master-Theorem

Wir diskutieren im folgenden den dritten Fall und nehmen an, dass

$$af(n/b) \leq cf(n)$$ für eine Konstante c < 1 und für alle n ≥ b

$$\Rightarrow f(n/b) \leq \frac{c}{a} f(n)$$ Wenden wir diese Abschätzung i mal iterativ an, so erhalten wir

$$\Rightarrow f(n/b^i) \leq \left(\frac{c}{a}\right)^i f(n)$$

$$\sum_{i=0}^{\log_b(n)-1} a^i f\left(\frac{n}{b^i}\right) \leq \sum_{i=0}^{\log_b(n)-1} c^i f(n) = f(n) \sum_{i=0}^{\log_b(n)-1} c^i = f(n) \frac{c^{\log_b(n)} - 1}{c-1}$$

Nehmen wir jetzt ferner an, dass

$$f(n) = \Omega(n^{\log_b(a)+\varepsilon})$$ für ein ε > 0, so gilt: $$T(n) = \Theta(f(n))$$

Master-Theorem

Satz [1]:

Seien $a \geq 1$ und $b > 1$ Konstanten, sei f(n) eine (positive) Funktion und sei T(n) die Rekursion der Form

$$T(n) = aT(n/b) + f(n)$$

wobei n/b entweder als $\lfloor n/b \rfloor$ oder als $\lceil n/b \rceil$ interpretiert werden kann.

(1) Ist $f(n) = O(n^{\log_b(a)-\varepsilon})$ für ein $\varepsilon > 0$, so gilt: $T(n) = \Theta\left(n^{\log_b(a)}\right)$

(2) Ist $f(n) = \Theta(n^{\log_b(a)})$ dann ist: $T(n) = \Theta\left(n^{\log_b(a)} \log(n)\right)$

(3) Ist $f(n) = \Omega(n^{\log_b(a)+\varepsilon})$ für ein $\varepsilon > 0$, und ist $a\, f(n/b) \leq c\, f(n)$ für eine Konstante $c < 1$ und genügend großes n, so gilt:

$$T(n) = \Theta(f(n))$$

Anwendung des Master-Theorems auf MergeSort

Die Laufzeit des MergeSort-Algorithmus wird durch die folgende RF beschrieben:

$$T(n) = 2T(n/2) + \Theta(n)$$

Die Konstanten a und b sind gleich 2 und die Funktion f(n), die den Zeitaufwand für das Mischen (Merge) der sortierten Teilfolgen angibt, ist in $\Theta(n)$.

Da $\log_b(a) = \log_2(2) = 1$ ist,

$$f(n) = \Theta(n^{\log_b(a)}) = \Theta(n^1) = \Theta(n)$$

folgt aus Fall (2) des Master-Theorems

Satz [2]:

Die Laufzeit T(n) des Algorithmus MergeSort ist in $\Theta(n \log n)$:

$$T(n) = \Theta(n \log(n))$$

Beispielanwendung des Master-Theorems

Als zweites Beispiel betrachten wir die Rekursion

$$T(n) = 9T(n/3) + n$$

Es gilt: a = 9, b = 3, f(n) = n, $\log_b(a) = \log_3(9) = 2$.

$$n^{\log_b(a)} = n^2 = \Theta(n^2)$$

$$f(n) = O(n^{\log_b(a)-\varepsilon}) \qquad \text{wobei } \varepsilon = 1 \text{ ist.}$$

Dann folgt aus Fall (1) des Master-Theorems

$$T(n) = \Theta\left(n^2\right)$$

Beispielanwendung des Master-Theorems

Als drittes Beispiel betrachten wir die Rekursion

$$T(n) = 3T(n/4) + n\log(n)$$

Es gilt: a = 3, b = 4, f(n) = n log(n), $\log_b(a) = \log_4(3) = 0.793$.

$$f(n) = \Omega(n^{\log_4(3)+\varepsilon}) = \Omega(n) \qquad \text{wobei } \varepsilon \approx 0.2 \text{ ist.}$$

Da für große n gilt: 3 f(n/4) = 3 (n/4) log(n/4) ≤ (3/4) n log(n) = (3/4) f(n), folgt aus Fall (3) des Master-Theorems

$$T(n) = \Theta\left(f(n)\right) = \Theta\left(n\log(n)\right)$$