Wie man sehen kann ist für dieses Resultat vor allem a wesentlich, a_0 und a_1 spielen eine untergeordnete Rolle. Deshalb werden solche Rekursionsgleichungen oft auch geschrieben als $T(n) = aT(\frac{n}{b}) + f(n)$ wobei jedes der a Vorkommen von $\frac{n}{b}$ für $\lceil \frac{n}{b} \rceil$ oder $\lfloor \frac{n}{b} \rfloor$ steht.

Beweis. Sei $g(n) = \sum_{w \in X_I(n)} a_0^{n_0(w)} a_1^{n_1(w)} f(n_w)$. In Fall 1 haben wir

$$g(n) = O\left(\sum_{w \in X_I(n)} a_0^{n_0(w)} a_1^{n_1(w)} n_w^{\log_b a - \varepsilon} \right) = O\left(\sum_{w \in X_I(n)} a_0^{n_0(w)} a_1^{n_1(w)} \left(\frac{n}{b^{|w|}} \right)^{\log_b a - \varepsilon} \right)$$

weil $n_w = \Theta(\frac{n}{b^{|w|}})$ und weiters

$$\begin{aligned}
&= O\left(\sum_{j=0}^{d(n)-1} a^j \left(\frac{n}{b^j} \right)^{\log_b a - \varepsilon} \right) = O\left(n^{\log_b a - \varepsilon} \sum_{j=0}^{d(n)-1} \left(\frac{a}{b^{\log_b a - \varepsilon}} \right)^j \right) \\
&= O\left(n^{\log_b a - \varepsilon} \sum_{j=0}^{d(n)-1} (b^\varepsilon)^j \right) = O\left(n^{\log_b a - \varepsilon} \frac{b^{\varepsilon d(n)} - 1}{b^\varepsilon - 1} \right) \\
&= O(n^{\log_b a - \varepsilon} n^\varepsilon) = O(n^{\log_b a})
\end{aligned}$$

weil $b^{d(n)} = \Theta(n)$. Insgesamt erhalten wir also $T(n) = \Theta(n^{\log_b a}) + O(n^{\log_b a}) = \Theta(n^{\log_b a})$.
In Fall 2 haben wir

$$g(n) = \Theta\left(\sum_{w \in X_I(n)} a_0^{n_0(w)} a_1^{n_1(w)} n_w^{\log_b a} \right) = \Theta\left(\sum_{w \in X_I(n)} a_0^{n_0(w)} a_1^{n_1(w)} \left(\frac{n}{b^{|w|}} \right)^{\log_b a} \right)$$

weil $n_w = \Theta(\frac{n}{b^{|w|}})$ und weiters

$$\begin{aligned}
&= \Theta\left(\sum_{j=0}^{d(n)-1} a^j \left(\frac{n}{b^j} \right)^{\log_b a} \right) = \Theta\left(n^{\log_b a} \sum_{j=0}^{d(n)-1} \left(\frac{a}{b^{\log_b a}} \right)^j \right) \\
&= \Theta\left(n^{\log_b a} d(n) \right) = \Theta(n^{\log_b a} \log_b n) = \Theta(n^{\log_b a} \log n).
\end{aligned}$$

Insgesamt erhalten wir also $T(n) = \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \log n) = \Theta(n^{\log_b a} \log n)$.
In Fall 3 ist $cf(n) \geq a_0 f(\lfloor \frac{n}{b} \rfloor) + a_1 f(\lceil \frac{n}{b} \rceil)$ für alle $n \geq n_0$. Sei nun $d_0(n) = d(n) - p$ wobei $p \in \mathbb{N}$ so gewählt ist, dass $\frac{n}{b^{d_0(n)}} \geq n_0$ für alle $n \geq n_0$. Außerdem ist $\frac{n}{b^{d_0(n)}} = \Theta(1)$. Sei $n \geq n_0$. Wir zeigen zunächst

$$c^j f(n) \geq \sum_{w \in \{0,1\}^j} a_0^{n_0(w)} a_1^{n_1(w)} f(n_w) \text{ für } j = 0, \ldots, d_0(n) \qquad (*)$$

mit Induktion nach j. Falls $j = 0$, dann ist $f(n) = f(n_\varepsilon)$. Für den Induktionsschritt haben wir

$$\begin{aligned}
c^{j+1} f(n) &\geq c \sum_{w \in \{0,1\}^j} a_0^{n_0(w)} a_1^{n_1(w)} f(n_w) \geq \sum_{w \in \{0,1\}^j} a_0^{n_0(w)} a_1^{n_1(w)} (a_0 f(n_{0w}) + a_1 f(n_{1w})) \\
&= \sum_{v \in \{0,1\}^{j+1}} a_0^{n_0(v)} a_1^{n_1(v)} f(n_v).
\end{aligned}$$

Dann ist $c^{d_0(n)} f(n) \geq \sum_{w \in \{0,1\}^{d_0(n)}} a_0^{n_0(w)} a_1^{n_1(w)} f(n_w)$, da aber $\frac{n}{b^{d_0(n)}} = \Theta(1)$ ist wegen Lemma 4.2 auch $\min\{f(n_w) \mid w \in \{0,1\}^{d_0(n)}\}$ eine Konstante q und somit $c^{d_0(n)} f(n) \geq q a^{d_0(n)}$. Damit erhalten wir

$$f(n) \geq q \left(\frac{a}{c}\right)^{\lfloor \log_b n \rfloor - p} = \Theta\left(\left(\frac{a}{c}\right)^{\log_b n}\right) = \Theta(n^{\log_b \frac{a}{c}}).$$

Nun ist $n^{\log_b \frac{a}{c}} = n^{\log_b a - \log_b c}$ und da $c < 1$ war ist $\log_b c < 0$, also ist $f(n) = \Omega(n^{\log_b a + \varepsilon})$ für $\varepsilon = -\log_b a > 0$.

Es bleibt zu zeigen dass $g(n) = \Theta(f(n))$. Zunächst einmal ist klar dass $g(n) = \Omega(f(n))$ da $f(n) = f(n_\varepsilon)$ ja ein Summand von $g(n)$ ist. Für die andere Richtung betrachten wir den Rekursionsbaum bis zur Tiefe $d_0(n)$. Die Anzahl von Blättern ist $a^{d_0(n)} = a^{\lfloor \log_b n \rfloor - p} = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$. Für $j \in \{d_0(n), \ldots, d(n) - 1\}$ haben wir: $\forall w \in \{0,1\}^j$: $\frac{n}{b^{|w|}} = \Theta(1)$, also $\forall w \in \{0,1\}^j$: $n_w = \Theta(1)$, also $\forall w \in \{0,1\}^j$: $f(n_w) = \Theta(1)$. Weiters ist die Anzahl der $f(n_w)$-Summanden an einem Blatt der Tiefe $d_0(n)$ konstant. Also sind die Kosten an einem Blatt der Tiefe $d_0(n)$ konstant und wir erhalten:

$$g(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{d_0(n)-1} \sum_{w \in \{0,1\}^j} a_0^{n_0(w)} a_1^{n_1(w)} f(n_w) \leq \Theta(n^{\log_b a}) + \sum_{j=0}^{d_0(n)-1} c^j f(n)$$

$$< \Theta(n^{\log_b a}) + f(n) \sum_{j=0}^{\infty} c^j = \Theta(n^{\log_b a}) + \frac{1}{1-c} f(n) = \Theta(f(n))$$

da $f(n) = \Omega(n^{\log_b a + \varepsilon})$ für ein $\varepsilon > 0$. Also ist $g(n) = O(f(n))$. Insgesamt erhalten wir also $T(n) = \Theta(n^{\log_b a}) + \Theta(f(n)) = \Theta(f(n))$. □

Korollar 4.1. *Die Rekursionsgleichung*

$$T(n) = T(\left\lfloor \frac{n}{2} \right\rfloor) + T(\left\lceil \frac{n}{2} \right\rceil) + \Theta(n)$$

die sowohl die Laufzeit von Sortieren durch Verschmelzen als auch jene von unserem Teile-und-herrsche Algorithmus für das dichteste Punktepaar beschreibt hat die Lösung $T(n) = \Theta(n \log n)$.

Beweis. In der Notation des master-Theorems gilt $a = b = 2$ und es trifft der zweite Fall zu. Die Lösung ist also $\Theta(n^{\log_b a} \log n) = \Theta(n \log n)$. □

Korollar 4.2. *Die Rekursionsgleichung*

$$T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$$

die die Laufzeit des einfachen rekursiven Algorithmus zur Matrixmultiplikation beschreibt hat die Lösung $T(n) = \Theta(n^3)$.

Beweis. In der Notation des master-Theorems ist $a = 8, b = 2$ und es trifft der erste Fall zu da $n^2 = O(n^{3-\varepsilon})$ für ein $\varepsilon > 0$. Die Lösung ist also $\Theta(n^{\log_b a}) = \Theta(n^3)$. □

Korollar 4.3. *Die Rekursionsgleichung*

$$T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$$

die die Laufzeit des Strassen-Algorithmus zur Matrixmultiplikation beschreibt hat die Lösung $T(n) = \Theta(n^{2{,}807\ldots})$.

Beweis. In der Notation des master-Theorems ist $a = 7, b = 2$. Es ist $\log_2 7 = 2{,}807\ldots$. Da $n^2 = O(n^{2{,}807\ldots-\varepsilon})$ für ein $\varepsilon > 0$ trifft der erste Fall zu. Die Lösung ist also $\Theta(n^{\log_b a}) = \Theta(n^{2{,}807})$. □

Beispiel 4.8. Auf die Rekursionsgleichung

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

ist das master-Theorem in dieser Form nicht anwendbar. Es ist nämlich $a = b = 2$, so dass $n \log n$ mit n verglichen werden muss. Nun ist aber $n \log n \neq \Theta(n)$, $n \log n \neq \Omega(n^{1+\varepsilon})$ sowie $n \log n \neq O(n^{1-\varepsilon})$. Damit trifft keiner der Fälle des master-Theorems zu. Diese Rekursionsgleichung hat die asymptotische Lösung $T(n) = \Theta(n \log^2 n)$.

Kapitel 5

Datenstrukturen

In diesem Kapitel werden wir uns mit Datenstrukturen beschäftigen. Eine Datenstruktur ist eine Struktur zur Speicherung und Organisation von Daten, die typischerweise gewisse Operationen (auf effiziente Weise) zur Verfügung stellt. Eine einfache Datenstruktur, das Datenfeld, haben wir bereits kennengelernt. Es erlaubt Lese- und Schreibzugriff auf ein beliebiges Element dessen Index bekannt ist in konstanter Zeit. Algorithmen und Datenstrukturen gehen Hand in Hand: jede Datenstruktur benötigt Algorithmen, welche die gewünschten Operationen zur Verfügung stellen und umgekehrt: oft sind gewisse Datenstrukturen notwendig, damit ein Algorithmus eine bestimmte Laufzeit erreicht. So haben wir zum Beispiel im teile-und-herrsche Algorithmus zur Lösung des dichtesten Punktepaarproblems (Algorithmus 10) eine Menge auf eine Weise repräsentiert, die das Durchlaufen anhand zweier unterschiedlicher Ordnungen erlaubt hat. Auch diese Repräsentation kann als Datenstruktur betrachtet werden.

Datenstrukturen sind darüber hinaus auch deswegen von großer Bedeutung in der Informatik, weil die Betrachtung von Berechnungsproblemen und Algorithmen als Lösungen dieser zwar für viele aber bei weitem nicht für alle Anwendungen adäquat ist. Oft befindet man sich in der Praxis in einer Situation die dynamisch ist, also nicht durch eine Eingabe-Ausgabe-Relation vollständig beschrieben werden kann.

5.1 Das Wörterbuchproblem

In diesem Abschnitt wollen wir das *Wörterbuchproblem* betrachten. Das *Wörterbuchproblem* besteht darin eine möglichst effiziente Organisation einer endlichen Menge von Datensätzen zu finden die abgefragt und verändert(!) werden kann. Wir nehmen dabei an, dass jeder Datensatz D durch einen eindeutigen Schlüssel $D.x$ identifiziert wird. Wir wollen, dass unser Wörterbuch M zumindest die folgenden Operationen unterstützt:

1. $M.Suche(x)$ gibt jenes D aus M zurück dessen Schlüssel x ist falls es existiert.
2. $M.Einfügen(D)$ fügt den neuen Datensatz D zu M hinzu.
3. $M.Löschen(x)$ entfernt den Datensatz mit Schlüssel x aus M.

Zum Beispiel könnte man an einer Repräsentation aller Studenten dieser Universität interessiert sein. Diese Menge verändert sich im Laufe der Zeit. Die Matrikelnummer kann als eindeutiger Schlüssel dienen. Als Lösung für das Wörterbuchproblem erwarten wir eine geeignete Datenstruktur gemeinsam mit zumindest drei Algorithmen, die die oben beschriebenen Operationen (möglichst effizient) durchführen.

Die einfachste Art ein solches Wörterbuch zu realisieren besteht darin, M als Datenfeld aufzufassen. Die Laufzeitkomplexität der Operationen (wobei $n = |M|$) ist dann wie folgt:

1. $M.\mathit{Suche}(x)$ benötigt Zeit $O(n)$, d.h. genauer: $\Theta(n)$ im schlechtesten Fall und $\Theta(1)$ im besten Fall.
2. $M.\mathit{Einfügen}(D)$ benötigt Zeit $\Theta(n)$ da sichergestellt werden muss, dass kein Duplikat erzeugt wird und dafür im schlechtesten Fall ($D.x$ existiert noch nicht) das gesamte Datenfeld durchlaufen werden muss.
3. $M.\mathit{Löschen}(x)$ benötigt Zeit $\Theta(n)$ da D mit $D.x = x$ zunächst gefunden werden muss und dann das Datenfeld verkürzt werden muss.

Eine bessere Darstellung von M besteht darin, ein nach Schlüssel (aufsteigend) sortiertes Datenfeld zu verwenden. Dann haben wir die folgenden Operationen:

1. $M.\mathit{Suche}(x)$ in Zeit $O(\log n)$ mit binärer Suche (engl. *binary search*), siehe Algorithmus 11.
2. $M.\mathit{Einfügen}(D)$ in Zeit $\Theta(n)$ da das Datenfeld verlängert werden muss.
3. $M.\mathit{Löschen}(x)$ in Zeit $\Theta(n)$ da das Datenfeld verkürzt werden muss.

Algorithmus 11 Binäre Suche

```
Prozedur BSuche(A, x)
    Antworte BSucheRek(A, x, 1, A.Länge)
Ende Prozedur

Prozedur BSucheRek(A, x, l, r)
    Falls l > r dann
        Antworte "nicht gefunden"
    sonst
        m := ⌈(l+r)/2⌉
        Falls A[m] = x dann
            Antworte m
        sonst falls A[m] < x dann
            Antworte BSucheRek(A, x, m + 1, r)
        sonst                                          ▷ A[m] > x
            Antworte BSucheRek(A, x, l, m − 1)
        Ende Falls
    Ende Falls
Ende Prozedur
```

Mit einem sortierten Datenfeld ist die Suche also effizient, Änderung hingegen nicht.

Eine Datenstruktur in der lokale Änderungen auf effiziente Weise gemacht werden können sind verkettete Listen. Hier unterscheidet man zwischen einfach verketteten und doppelt verketteten Listen. Die Grundidee ist, dass die Elemente einer Liste jeweils einzeln an einer beliebigen Stelle im Speicher abgelegt werden und jedes Element auf das nächste (einfache Verkettung) bzw. auf das nächste und das vorherige verweist (doppelte Verkettung). Diese Verweise auf Speicherpositionen werden als *Zeiger* (engl. *pointer*) bezeichnet. Ein Zeiger verweist entweder auf einen bestimmten Ort im Speicher oder er hat den Wert Nil, den Nullzeiger. Realisiert wird ein

einzelner Knoten durch ein Tupel v (für Vertex), dessen Elemente im Fall einer einfach verketteten Liste $v.D$ und $v.nächster$ sind, im Fall einer doppelt verketteten Liste $v.D$, $v.nächster$ und $v.vorheriger$ sind. Der Vorteil einer doppelt verketteten Liste gegenüber einer einfach verketteten besteht darin, dass sie in beide Richtungen durchlaufen werden kann, ihr Nachteil darin, dass sie (konstant) mehr Speicherplatz benötigt. Doppelt verkettete Listen erlauben Einfügen und löschen in konstanter Zeit durch Umsetzen der pointer Eine doppelt verkette Liste als Darstellung von M erlaubt die folgenden Operationen:

1. $M.Suche(x)$ benötigt Zeit $O(n)$ genauer: $\Theta(n)$ im schlechtesten Fall, $\Theta(1)$ im besten Fall und $\Theta(n)$ im Durchschnittsfall.
2. $M.Einfügen(D)$ in Zeit $\Theta(n)$ da sichergestellt werden muss, dass kein Duplikat erzeugt wird
3. $M.Löschen(x)$ wie die Suche in Zeit $O(n)$ da D mit $D.x = x$ gefunden werden muss.

Doppelt verkettete Listen erlauben zusätzlich noch die folgenden effizienten Operationen:

1'. $M.Einfügen(D)$ in Zeit $\Theta(1)$ unter der Annahme dass D noch nicht in M vorkommt.

2'. $M.Löschen(v)$ in Zeit $\Theta(1)$ durch Umsetzen der Zeiger unter der Voraussetzung dass v ein Element von M ist.

Bei verketteten Listen ist zwar im Vergleich zum unsortierten Datenfeld nichts gewonnen was die Suche angeht, allerdings erhalten wir bei den dynamischen Operationen Einfügen und Löschen neue Möglichkeiten.