Diplomarbeit

FU-Berlin
Fachbereich Mathematik

Variationen des Huffman-Verfahrens zur Bestimmung optimaler Bäume und Codes

Nicole Morawe

Januar 1997

Ich danke meinen Eltern dafür, daß sie mich in all den Jahren meines Studiums der Mathematik unterstützt haben und dabei nur ein ganz klein wenig mit der Wimper zuckten.

Danken möchte ich auch Ilona Grandt und Irina Kaldrack. Beide hatten, wie auch andere Freunde, stets ein offenes Ohr für die privaten Probleme.

Desweiteren gilt mein Dank Dr.Stefan Felsner und seiner sehr guten Betreuung .

Zusammenfassung

Beim Versenden von Nachrichten in modernen Datennetzen müssen Texte in binäre Dateien umgewandelt werden. Dabei werden den Buchstaben oft mittels bekannter Tabellen (zum Beispiel dem ASCII-Code) gleichlange Bitfolgen zugewiesen. Andere Codes dagegen weisen verschiedenen Buchstaben unterschiedlich lange Bitfolgen zu. Diese Methoden ermöglichen eine Optimierung der Länge des codierten Textes.

Ein optimaler Code codiert einen festen Text mit minimaler Gesamtlänge. Geht man von einer statistischen Buchstabenverteilung aus, dann minimiert ein optimaler Code den Erwartungswert der Länge der codierten Nachricht, minimiert also den Ausdruck $\sum_v p(v)l(v)$. Dabei ist $p(v)$ die Wahrscheinlichkeit des Buchstaben v und $l(v)$ die Länge der ihm zugewiesenen Bitfolge.

Huffman gab 1952 einen Algorithmus zum Finden von optimalen Codes mit einer Komplexität von $O(n \log_2 n)$ an. In diesem Zusammenhang wird auf das äquivalente Finden eines optimalen Baumes verwiesen.

Die Diplomarbeit betrachtet Arbeiten, in denen optimale Codes mit folgenden Nebenbedingungen konstruiert werden:

Alphabetische Codes
Ein Code heißt alphabetisch, wenn die lexikographischen Ordnungen der zu verschlüsselnden Buchstaben und der Codewörter übereinstimmen. Gesucht wird ein optimaler Code zu einer Buchstabenordnung.

L-beschränkte Codes
Ein Code heißt L-beschränkt, wenn keine Bitfolge mehr als L Zeichen enthält. Gesucht wird ein optimaler L-beschränkter Code. Dies ist von Interesse, wenn die Tabellen, in der die Bitfolgen der Verschlüsselung abgelegt werden, nur Spalten der Wortbreite L besitzt.

On-line Codierung mit Huffman-Codes
Ein Nachteil des Huffman-Algorithmus besteht darin, daß man den zu sendenden Text zweimal „lesen“ muß. Beim ersten Mal wird die Buchstabenverteilung untersucht, beim zweiten Mal wird die Nachricht Zeichen für Zeichen verschlüsselt. Bei der on-line Methode wird die Verteilung während der Verschlüsselung erlernt, diese ändert sich kontinuierlich. Daher muß der Code ständig an die neue Verteilung angepaßt werden.

In der Diplomarbeit wird die historische Entwicklung der Lösungsansätze nachvollzogen. Dabei werden die Algorithmen und ihre Laufzeitanalysen analysierend gegenübergestellt.

Inhaltsverzeichnis

Kapitel 1

Grundlagen

1.1 Codierung und Kompression

Beim Datentransfer schickt der Sender (auch *Quelle* genannt) eine Folge von Zeichen aus einem Zeichenvorrat (dem *Quellalphabet*) an den Empfänger. Diese Zeichen werden zum Zweck der Übertragung oft in Zeichen eines anderen Alphabets umgewandelt. Gründe dafür sind die Notwendigkeit der technischen Anpassung und der Wunsch nach Geheimhaltung, Datenreduzierung oder Kennzeichnung von Übertragungsfehlern.

Im folgenden wird eine störungsfreie Übertragung vorausgesetzt. Der Empfänger erhält also genau die Nachricht, die der Sender an ihn schickt.

Definition 1.1 *Ein* Alphabet *ist eine nichtleere, endliche Menge $\mathcal{A}$ von Zeichen. Diese Zeichen werden* Buchstaben *oder* Symbole *genannt. Folgen $(a_1, \ldots, a_n)$ von Zeichen aus diesem Alphabet heißen* Wörter über $\mathcal{A}$. $\mathcal{A}^*$ *ist die Menge der Wörter von $\mathcal{A}$.*
Seien $\mathcal{A}$,$\mathcal{B}$ Alphabete und f eine injektive Funktion, die jedem Element aus $\mathcal{A}$ ein Wort über $\mathcal{B}$ zuordnet. $\mathcal{A}$ heißt dann Quellalphabet, *$\mathcal{B}$ bezeichnet das* Codealphabet. *Dann kann man eine Funktion f^* definieren mit*

$$\begin{aligned} f^* : \mathcal{A}^* &\longrightarrow \mathcal{B}^* \\ (a_1, \ldots, a_n) &\rightarrow (f(a_1), \ldots, f(a_n)). \end{aligned}$$

f und f^* heißen Codierung eines Zeichens *bzw.* Codierung eines Wortes, *die Elemente von $f(\mathcal{A})$ heißen* Codewörter. *Ein* Code *C ist eindeutig bestimmt durch ein Tripel $(f, \mathcal{A}, \mathcal{B})$.*

Bemerkung 1.2 Aus der Injektivität von f folgt nicht notwendig die Injektivität von f^*.

Beispiel 1.3 Das Quellalphabet bestehe aus den Zeichen a, b, c; das Codealphabet aus den Zeichen $0, 1$. f sei folgendermaßen definiert: $f(a) = 1, f(b) = 0, f(c) = 01$. Dann ist $f(\mathcal{A}) = \{1, 0, 01\}$. f^* ist nicht injektiv, da $01 = f^*(ba) = f^*(c)$ ist.

Die hier betrachteten Codes sind *Einzelzeichen-Codes*, da jedes Wort anhand der Codierung seiner einzelnen Zeichen verschlüsselt wird.

Bemerkung 1.4 Eine Codierung von Daten D ist nur angebracht, wenn $f^*(D)$ sinnvoll decodierbar ist. Was *sinnvoll* ist, hängt von dem Zweck ab, den die Zeichenfolge erfüllen soll.

Im folgenden werden nur *verlustfreie Codierungen* mit $f^{*-1}(f^*(D)) = D$ betrachtet – die entschlüsselte Zeichenfolge entspricht genau der ursprünglichen Zeichenfolge. Dies ist beispielsweise bei Textcodierung Voraussetzung, da kein einziges Zeichen durch die Codierung verloren gehen soll. Notwendig für eine verlustfreie Codierung ist die Injektivität der Funktion f^*.

Andere sinnvolle Codierungen liefern nach der Decodierung nur noch eine Approximation der ursprünglichen Zeichenfolge. Diese Techniken finden Anwendung bei Übertragung von Sprache, Musik und Bildern.

Kompression bezeichnet die Reduzierung einer Datenmenge durch Codierung. Durch Datenreduzierung wird bei der Speicherung von Daten Platz auf dem Speichermedium gespart und bei der Kommunikation wird durch Kompression die Datenübertragungszeit verkürzt.

Definition 1.5 *Das Quellalphabet $\mathcal{A}$ und das Codealphabet $\mathcal{B}$ seien identisch. Dann ist* Datenkompression *die Codierung einer Zeichenfolge D, so daß die neue Zeichenfolge $f^*(D)$ aus weniger Buchstaben besteht als D.*

Beispiel 1.6 Die folgenden Codierungen liefern eine verlustfreie Kompression.

Null-Unterdrückung: Falls in einer Zeichenfolge viele Nullen vorkommen, werden diese nicht einzeln verschickt, sondern nach einem Sondersymbol wird lediglich ihre Anzahl übermittelt.
Beispiel: ABC0000DE00FF wird codiert durch ABC*4DE*2FF

Lauflängen-Codierung: Falls in einer Zeichenfolge gleiche Symbole hintereinander vorkommen, wird nach einem Sonderzeichen das Symbol und seine Häufigkeit übertragen.
Beispiel: ABBBCDDDDEEEF wird codiert durch A*B3C*D4*E3F

Mustererkennung: Zeichenfolgen, die häufig auftreten, werden durch neue Zeichen beziehungsweise durch kürzere Zeichenfolgen ersetzt.
Beispiel: „genau dann wenn“ wird codiert durch „g.d.w.“.

1.2 Präfix-Codes

In Bemerkung 1.2 wurde erwähnt, daß f^* nicht notwendig injektiv ist, die Decodierung von Nachrichten also nicht unbedingt ein eindeutiges Ergebnis liefert. Da nur verlustfreie Codierungen betrachtet werden sollen, werden hinreichende Bedingungen für eine eindeutige Decodierbarkeit angegeben.

Definition 1.7 *Sei C ein Code und y ein Buchstabe aus dem Quellalphabet. $\phi_n(y)$ soll die ersten n Buchstaben des Wortes $f(y)$ angeben, $n \in \mathbf{N_0}$. $\phi_n(y)$ heißt* Präfix *von $f(y)$.*
Ein Code ist ein Präfix-Code*, wenn kein Codewort Präfix eines anderen Codewortes ist: Für je zwei verschiedene Zeichen a_i und a_j des Quellalphabets existiert kein $n \in \mathbf{N_0}$ mit $\phi_n(a_i) = f(a_j)$ oder $\phi_n(a_j) = f(a_i)$.*

Bemerkung 1.8 Das leere Wort $\emptyset$ und das Wort $f(y)$ selbst sind in der Menge der Präfixe von $f(y)$ enthalten. Alle anderen Präfixe nennt man *echte Präfixe*.

Beispiel 1.9 Der Code aus Beispiel 1.3 ist kein Präfix-Code, weil $\phi_1(c) = f(b)$ ist.

Satz 1.10 *Jeder Präfix-Code ist eindeutig decodierbar.*

Beweis 1.10 Das ergibt sich daraus, daß die Funktion f^* eines Präfix-Codes stets injektiv ist. $\diamond$

Bemerkung 1.11 Die Umkehrung gilt nicht, wie man am folgenden Beispiel sieht: Sei $f(a) = 1, f(b) = 10, f(c) = 00$. Dann ist C eindeutig decodierbar, wenn die gesamte Nachricht vor Beginn der Entschlüsselung bekannt ist, jedoch ist C kein Präfix-Code.

Soll zu jedem Zeitpunkt der Datenübertragung eine eindeutige Entschlüsselung möglich sein, dann gilt auch die Umkehrung von Satz 1.10. Denn falls C kein Präfix-Code ist, muß ein spezielles Symbol reserviert werden, welches das Ende der Codierung eines Zeichens kenntlich macht. Mit diesem zusätzlichen Symbol wird aber jeder Code zu einem Präfix-Code. Für die oben formulierte Annahme der ständigen eindeutigen Decodierbarkeit gilt also: Die Menge der eindeutig decodierbaren Codes stimmt mit der Menge der Präfix-Codes überein. Dies wird im nächsten Satz festgehalten.

Satz 1.12 *Soll zu jedem Zeitpunkt der Übertragung eine eindeutige Decodierung des Codes C möglich sein, dann muß C ein Präfix-Code sein.*

Beispiel 1.13 Sei $f(a) = 10, f(b) = 100$ und die gesendete Nachricht sei 10010. Dann wird bei Kenntnis aller fünf Zeichen die Nachricht eindeutig entschlüsselt zu ba. Sind nur die ersten zwei Zeichen bekannt, wird man die Nachricht zu a entschlüsseln. Dann bekommt man bei Fortsetzung der Entschlüsselung einen Fehler und muß die bereits entschlüsselten Zeichen noch einmal betrachten, was nicht erwünscht ist.

Der folgende Satz erläutert den Zusammenhang zwischen der Existenz eines Präfix-Codes über einem Codealphabet der Größe r und der Länge seiner Codewörter.

Satz 1.14

1. *Sei C ein Präfix-Code über einem Codealphabet der Größe r mit n Codewörtern der Längen $m_1, \ldots m_n$.*
   Dann gilt
   $$\sum_{i=1}^{n} r^{-m_i} \leq 1$$
2. *Seien $m_1, .., m_n$ aus $\mathbf{N}$ gewählt mit $\sum_{i=1}^{n} r^{-m_i} \leq 1$.*
   Dann existiert ein Präfix-Code über einem beliebigen Codealphabet der Größe r mit n Codewörtern, die die Längen $m_1, .., m_n$ haben.

Beweis 1.14

1. Sei C ein Präfix-Code mit den Codewörtern $b_1, \ldots, b_n$.
   O.B.d.A. sei $m_n = \max_{i=1,\ldots,n} m_i$. Wir unterscheiden zwei Fälle:

   (a) Es existieren $r-1$ Codewörter der Länge m_n, die in den ersten $m_n - 1$ Zeichen mit b_n übereinstimmen. Diese Wörter seien o.B.d.A. die Wörter $b_{n-r+1}, \ldots, b_{n-1}$. Alle diese Wörter unterscheiden sich also genau im letzten Zeichen.
   Der Beweis erfolgt mit vollständiger Induktion über n.
   Für $n = 1$ ist der Satz richtig, denn für alle k aus $\mathbf{N}$ gilt :$\frac{1}{r^k} \leq 1$.
   Sei der Satz für alle $j < n$ schon bewiesen. Sei $\bar{C}$ ein Code mit den Codewörtern $b_1, \ldots, b_{n-r}, \bar{b}$, wobei $\bar{b}$ aus den ersten $m_n - 1$ identischen Zeichen der Wörter $b_{n-r+1}, \ldots, b_n$ besteht. Die zugehörigen Codewortlängen sind $m_1, \ldots, m_{n-r}, m_n - 1$. Man erkennt, daß $\bar{C}$ ein Präfix-Code ist, und auf diesen kann man die Induktionsvoraussetzung anwenden, so daß gilt:
   $$\sum_{i=1}^{n-r} r^{-m_i} + r^{-m_n+1} \leq 1.$$

Damit folgt dann:

$$\sum_{i=1}^{n} r^{-m_i} = \sum_{i=1}^{n-r} r^{-m_i} + r * r^{-m_n} = \sum_{i=1}^{n-r} r^{-m_i} + r^{-m_n+1} \leq 1.$$

(b) Es existieren weniger als $r-1$ Codewörter der Länge m_n, die in den ersten $m_n - 1$ Zeichen mit b_n übereinstimmen.
Folglich existiert ein Präfix-Code $\bar{C}$, der aus den Codewörtern des Codes C besteht und p weiteren Wörtern aus $\mathcal{B}^*$, so daß die Eigenschaft aus 1.(a) erfüllt wird. Dann folgt mit 1.(a):

$$\sum_{i=1}^{n} r^{-m_i} + p * r^{-m_n} \leq 1.$$

Damit erhält man folgende Abschätzung:

$$\sum_{i=1}^{n} r^{-m_i} < \sum_{i=1}^{n} r^{-m_i} + p * r^{-m_n} \leq 1.$$

2. Sei s_i definiert als die durch $m_1, \ldots, m_n$ bestimmte Anzahl der Codewörter der Länge i und sei $K := \max_{i=1,\ldots,n} m_i$. Dann gilt

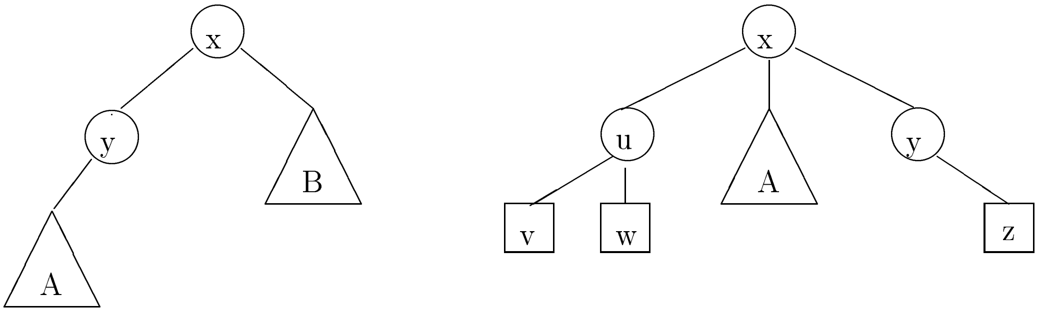
$$\sum_{i=1}^{K} s_i r^{-i} = \sum_{i=1}^{n} r^{-m_i} \leq 1. \tag{1.1}$$

Der gesuchte Code wird nun induktiv konstruiert. Dabei werden im j-ten Schritt alle s_j Codewörter mit j Buchstaben erstellt. Es ist zu zeigen, daß dies möglich ist.
Für $j = 1$ ist die Konstruktion möglich, da aufgrund der Ungleichung (1.1) gilt: $s_1 \leq r$. Zu diesen Werten ist problemlos eine Präfix-Codierung konstruierbar.
Sei der Code für alle $i \leq j$ schon konstruiert. Es gibt r^j mögliche Zeichenkombinationen der Länge j. Da C ein Präfix-Code ist, sind nicht alle Kombinationen zulässig. Zu entfernen sind alle Wörter, von denen ein Präfix als Codewort auftritt, das sind $\sum_{i=1}^{j-1} s_i r^{j-i}$ viele. Man hat also nach dem j-ten Schritt s_j Codewörter mit j Zeichen und $r^j - \sum_{i=1}^{j} s_i r^{j-i}$ mögliche Präfixe zur Weiterverwendung. Im $j+1$.ten Schritt können genau dann s_{j+1} neue Codewörter konstruiert werden, wenn folgende Ungleichung erfüllt ist

$$s_{j+1} \leq r(r^j - \sum_{i=1}^{j} s_i r^{j-i}).$$

Diese Ungleichung folgt aber aus Formel (1.1) durch Multiplikation beider Seiten mit r^{j+1}.

$\diamond$



(x,(y,A,0),B) (x,(u,(v,0,0,0),(w,0,0,0),0),A,(y,0,0,(z,0,0,0)))

Abbildung 1.1: graphische Darstellung von Bäumen

1.3 Bäume

Es wird der Begriff des Baumes und seine graphische Darstellung erläutert. Daran schließen sich einige Sätze über Bäume an.

Definition 1.15 *Sei $\mathcal{X}$ eine Menge von Zeichen.*
Ein t-närer Baum über *$\mathcal{X}$ ist ein geordnetes Tupel $\mathbf{B} = (x, \mathbf{B_1}, \dots, \mathbf{B_t})$, wobei $x \in \mathcal{X}$ und $\mathbf{B_i}$ ein Baum über $\mathcal{X}$ oder die leere Menge $\mathbf{B_i} = \emptyset$ ist.*
x heißt Wurzel von $\mathbf{B}$.
Ein in $\mathbf{B}$ enthaltener Baum heißt Teilbaum von $\mathbf{B}$.
x heißt Knoten von $\mathbf{B}$*, wenn x Wurzel eines Teilbaums von $\mathbf{B}$ ist.*

Bemerkung 1.16 $\mathbf{B}$ selbst ist ebenfalls Teilbaum von $\mathbf{B}$.

Definition 1.17 *Sei $\mathbf{A} = (y, \mathbf{A_1}, \dots, \mathbf{A_t})$ ein Teilbaum von $\mathbf{B}$, die Wurzel von $\mathbf{A_i}$ sei y_i.*
y heißt Vater *von y_i, die y_i heißen* Kinder *von y. Knoten mit gleichem Vater heißen* Geschwister.
Knoten aus den Bäumen $\mathbf{A_i}$ heißen Nachfahren *von y. Knoten aus $\mathbf{B}$, die y als Nachfahren haben, heißen* Vorfahren *von y.*
Knoten ohne Nachfahren werden Blätter *genannt, alle anderen Knoten heißen* innere Knoten*. Die Menge der Blätter wird mit $\beta(\mathbf{B})$ bezeichnet, die Menge der inneren Knoten wird $\gamma(\mathbf{B})$ genannt.*

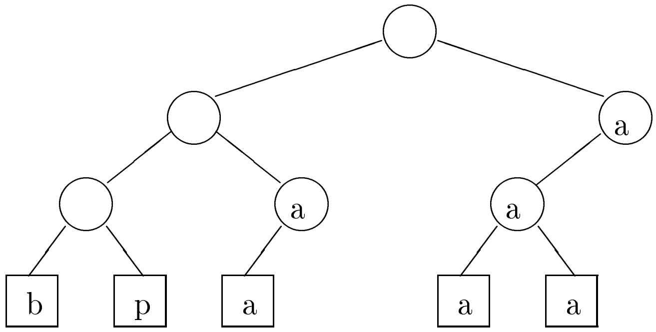


Abbildung 1.2: zum Begriff links/rechts

Bemerkung 1.18 Um den Aufbau eines Baumes leichter zu veranschaulichen, greift man auf die graphische Darstellung zurück. Wie diese aussieht wird in Abbildung 1.1 erläutert. In der graphischen Darstellung werden innere Knoten durch Kreise angegeben, Blätter durch Quadrate und Teilbäume durch Dreiecke.

Jedem Knoten kann in eindeutiger Weise eine natürliche Zahl zugeordnet werden, die angibt, wieviele Vorfahren dieser Knoten besitzt. Diese Größe wird *Weglänge* genannt, da sie ein Maß für die Entfernung eines Knoten von der Wurzel ist. Man schreibt $l(\text{Knoten}) = k$, wenn der Knoten k Vorfahren besitzt.

Ein Knoten befindet sich in *Ebene* k, wenn seine Weglänge k ist. Dieser Begriff ist durch die graphische Darstellung geprägt.

Durch die graphische Darstellung erhält man auch die Idee für die Definition eines weiteren Objekts: Seien u, v Knoten und u Vater von v. Das geordnete Tupel (u, v) wird *Kante* genannt. u heißt der Start- oder Anfangsknoten, v der Endknoten der Kante. Die Kanten eines Baumes sind eindeutig bestimmt durch den Baum.

Um in der abstrakten Baumdarstellung wie in der graphischen Darstellung die Begriffe links/rechts zu benutzen, führt man diese korrekt ein: Ein Knoten u ist genau dann links (rechts) von einem Knoten v, wenn es einen Teilbaum $\mathbf{A} = (y, \mathbf{A_1}, \ldots, \mathbf{A_t})$ gibt, so daß u in $\mathbf{A_i}$ und v in $\mathbf{A_j}$ liegt und $i < j$ $(i > j)$ ist. In Abbildung 1.2 befinden sich Knoten, die mit einem a gekennzeichnet sind, rechts von p, der mit b gekennzeichnete Knoten befindet sich links von p.

Definition 1.19 *Sei* $\mathbf{B}$ *ein* t-närer Baum.
$\mathbf{B}$ *heißt* regulär, *wenn für jeden Teilbaum* $\mathbf{A} = (y, \mathbf{A_1}, \ldots, \mathbf{A_t})$ *gilt: Entweder* $\mathbf{A_1} = \mathbf{A_2} = \ldots = \mathbf{A_t} = \emptyset$ *oder* $\mathbf{A_1}, \ldots, \mathbf{A_t} \neq \emptyset$. *Er heißt* vollständig, *wenn*

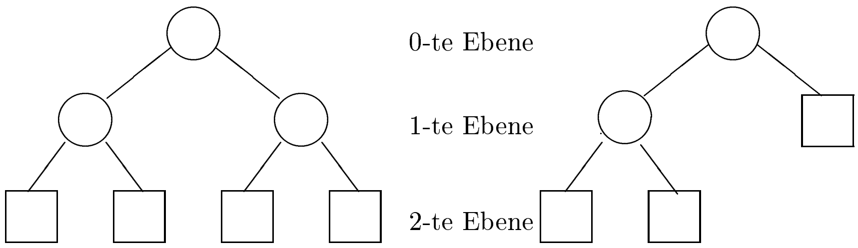
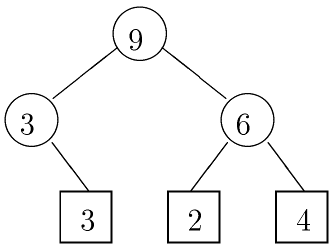


Abbildung 1.3: vollständiger und regulärer Baum

außerdem alle Blätter den gleichen Abstand von der Wurzel haben.
Ein Baum heißt gewichtet*, wenn seinen Knoten oder Kanten Werte, die* Gewichte*, zugeordnet werden.*

Bemerkung 1.20

- Bei $t = 2$ spricht man vom *binären Baum.*
- Die inneren Knoten eines t-nären Baumes haben höchstens t Kinder. Die inneren Knoten eines t-nären regulären Baumes haben genau t Kinder.
- Die Anzahl der inneren Knoten eines binären regulären Baumes ist genau eins geringer als die Anzahl seiner Blätter: $|\gamma(\mathbf{B})| = |\beta(\mathbf{B})| - \mathbf{1}$.



Bemerkung 1.21 Es wird meist eine spezielle Knotengewichtung betrachtet. Bei dieser soll das Gewicht eines Knotens der Summe der Gewichte seiner Kinder entsprechen. Dann ergibt sich schon durch Zuordnung von Gewichten an die Blätter eine eindeutige Gewichtsverteilung an allen Knoten des Baumes. Deshalb wird oft nur den Blättern ein Gewicht ω zugewiesen.

Ist nichts gegenteiliges erwähnt, so ist mit dem Begriff Knotengewichtung diese spezielle Art der Gewichtung gemeint, ein Beispiel dafür ist nebenstehend angegeben.

Ein besonderes Interesse liegt in den folgenden Kapiteln bei regulären knotengewichteten Bäumen. Bei diesen Bäumen wird die gewichtete Pfadlänge $\sum_{u\in\beta(\mathbf{B})}\omega(u)l(u)$ betrachtet. Diesen Wert nennt man die *Kosten des Baumes.*

Satz 1.22 *Die Kosten eines Baumes entsprechen dem Wert, der sich ergibt, wenn die Gewichte aller inneren Knoten addiert werden:*

$$\sum_{u \in \beta(\mathbf{B})} \omega(u) l(u) = \sum_{u \in \gamma(\mathbf{B})} \omega(u)$$

.

1.4 Code-Bäume

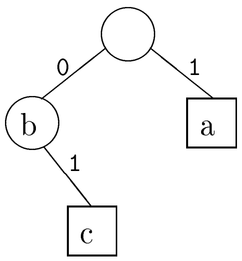
Ein Code läßt sich mit Hilfe eines Baumes darstellen. Dabei weisen die Code-Bäume von Präfix-Codes besondere Eigenschaften auf.

Konstruktion von Code-Bäumen

Sei $\mathcal{B}$ ein Alphabet mit r Buchstaben. Dann kann man einen r-nären kantengewichteten Baum konstruieren, bei dem jeder Kante ein Symbol aus $\mathcal{B}$ zugeordnet ist, so daß Kanten mit gleichem Startknoten unterschiedliche Symbole tragen. Ist der Baum regulär, dann kommt jeder Buchstabe genau an einer wegführenden Kante jedes Knotens vor.

Jedem Knoten in der i-ten Ebene kann nun in eindeutiger Weise ein Wort der Länge i zugeordnet werden. Dieses besteht aus den Buchstaben, die nacheinander an den Kanten des Weges von der Wurzel durch den Baum zu diesem Knoten vorkommen.

Einen Codebaum zum Quellalphabet $\mathcal{A}$ und Codealphabet $\mathcal{B}$ erhält man, indem man den Kanten in der oben beschriebenen Weise Buchstaben aus $\mathcal{B}$ zuordnet. Dann repräsentiert jeder Knoten ein Wort $(b_1, \ldots, b_n)$ in $\mathcal{B}^*$. Existiert ein Buchstabe a aus $\mathcal{A}$, der durch dieses Wort codiert wird (also $f(a) = (b_1, \ldots, b_n)$), wird dieser Knoten mit dem Wert a identifiziert. Betrachtet werden nur die Blätter (und deren Vorfahren), die ein Symbol aus $\mathcal{A}$ repräsentieren.



Der nebenstehende Baum ist der Codebaum zum Beispiel 1.3.

Satz 1.23 *Die Codewörter eines Codes stehen im zugehörigen Code-Baum genau dann an den Blättern, wenn dieser Code ein Präfix-Code ist.*

Beweis 1.23 Ist C ein Präfix-Code, dann können nur Blätter mit Symbolen aus $\mathcal{A}$ beschriftet sein.

Repräsentieren andererseits nur die Blätter Codewörter, dann definiert dies einen Präfix-Code. $\diamond$

Satz 1.24 (Kraftsche Ungleichung) *Sei* $m_1, \ldots, m_n$ *aus* $\mathbf{N}$.
Es existiert genau dann ein r*-närer Baum, dessen* n *Blätter die Abstände* $m_1, .., m_n$ *von der Wurzel haben, wenn*

$$\sum_{i=1}^{n} r^{-m_i} \leq 1$$

ist.
Die Ungleichung ist scharf, wenn der Baum regulär ist.

Beweis 1.24 Der Behauptung folgt aus Satz 1.14 und Satz 1.23. $\diamond$

1.5 Entropie

Dieses Kapitel erläutert den Begriff „Informationsgehalt einer Quelle". Dabei interessiert nicht mehr der Inhalt der Nachricht, sondern wie schwer es ist, eine Vermutung über das nächste eintreffende Zeichen anzustellen. Der Informationsgehalt soll also davon abhängen, wie wahrscheinlich es ist, daß ein bestimmtes Symbol gesendet wird. Er soll umso höher sein, je weniger man sicher ist, welches Symbol als nächstes eintrifft.

Definition 1.25 *Eine* Informationsquelle Q *ist eine Instanz, die nacheinander Zeichen eines endlichen Alphabets, des* Quellalphabets $\mathcal{A}$, *produziert.*
Diese Quelle ist von 1-ter Ordnung/ohne Gedächtnis, *wenn jedem Buchstaben* a *eine Sende-Wahrscheinlichkeit* $p(a)$ *zugeordnet wird mit* $\sum_{a \in \mathcal{A}} p(a) = 1$.
Die durch Senden des Buchstabens a *gelieferte* Information *sei definiert durch* $I(a) = -\log_2 p(a)$. *Die Einheit sei* bit.

Liegt für die Quelle eine Codierung vor, kann man folgende Begriffe festlegen:

Definition 1.26 *Sei* C *ein Code mit Quellalphabet* $\mathcal{A}$.
Die mittlere Codewortlänge $L_C(Q)$ *der Quelle* Q *sei der Erwartungswert der Länge der Codierung des nächsten gesendeten Symbols durch den Code* C, *also* $L_C(Q) = \sum_{a \in \mathcal{A}} p(a) l(f(a))$. *Die Einheit sei* bit/Symbol.
$L^r_{opt}(Q)$ *sei der* minimale mittlere Codieraufwand bei Einzelzeichen-Codierung *über einem Alphabet der Größe* r, *also* $L^r_{opt}(Q) = \min_{C \in \mathcal{C}} L_C(Q)$, *wobei* $\mathcal{C}$ *die Menge aller möglichen Codes* $(f, \mathcal{A}, \mathcal{B})$ *mit* $|\mathcal{B}| = r$ *ist.*

Bemerkung 1.27 Ein Code über einem Codealphabet der Größe r, der die Gleichung $L_C(Q) = L^r_{\text{opt}}(Q)$ erfüllt, wird *optimal* genannt.

Im folgenden wird davon ausgegangen, daß die Entschlüsselung einer Nachricht zu jedem Zeitpunkt der Datenübertragung geschehen kann, die Nachricht also bei Beginn der Entschlüsselung nicht komplett vorliegen muß.

Satz 1.28 *Falls die eindeutige Entschlüsselung einer Nachricht zu jedem Zeitpunkt der Datenübertragung möglich ist, dann gilt* $L^r_{opt}(Q) = \min_{C \in \hat{\mathcal{C}}} L_C(Q)$*, wobei* $\hat{\mathcal{C}}$ *die Menge aller möglichen Präfix-Codes* $(f, \mathcal{A}, \mathcal{B})$ *mit* $|\mathcal{B}| = r$ *ist.*

Beweis 1.28 Aus Satz 1.12 folgt, daß $\mathcal{C}$ wegen der Voraussetzung nur Präfix-Codes enthalten kann. $\diamond$

Da selten der Informationsgehalt einzelner Zeichen von Interesse ist, wird ein Maß für den Informationsgehalt einer Quelle benötigt. Dabei betrachtet man eine Größe, die proportional zum Erwartungswert von I ist, diese Größe heißt *Entropie*.

Definition 1.29 *Sei* Q *eine Quelle erster Ordnung. Die Entropie ist definiert durch* $E^r(Q) = -\sum_{a \in \mathcal{A}} p(a) \log_r p(a)$*. Die Einheit ist* bit/Symbol.

Bemerkung 1.30 Einige Autoren definieren die Entropie folgendermaßen: $\bar{E}(Q) = \sum_{a \in \mathcal{A}} p(a) I(a)$. Dieses nennt man dann auch den *mittleren Informationsgehalt von* Q. Beide Definitionen unterscheiden sich nur um den Faktor $\log_2 r$, da

$$E^r(Q) = -\sum_{a \in \mathcal{A}} p(a) \log_r p(a) = -\frac{1}{\log_2 r} \sum_{a \in \mathcal{A}} p(a) \log_2 p(a) = \frac{1}{\log_2 r} \bar{E}(Q).$$

Bemerkung 1.31 Seien Q_1, Q_2 verschiedene Quellen mit gleichen Wahrscheinlichkeitsverteilungen. Dann gilt $E^r(Q_1) = E^r(Q_2)$.

Satz 1.32 *Sei* Q *Quelle erster Ordnung. Dann gilt:*

$$E^r(Q) \leq L^r_{opt}(Q) < E^r(Q) + 1.$$

Beweis 1.32

- Gezeigt wird $E^r(Q) \leq L_C(Q)$ für jeden Präfix-Code C über einem Codealphabet der Größe r.
  Dazu benutzt man folgende Abschätzung des Logarithmus:

  $$\ln x \leq x - 1 \text{ für alle } x \in \mathbf{R}^+ \backslash \{0\}.$$

Durch Umformen erhält man daraus

$$\log_r x \leq \frac{x-1}{\ln r} \text{ für alle } x \in \mathbf{R}^+ \setminus \{0\}. \tag{1.2}$$

Sei C ein Präfix-Code über einem Codealphabet der Größe r. Die n Codewörter haben die Codewortlängen $l_1, \ldots, l_n$. Dann gilt aufgrund der Kraftschen Ungleichung:

$$\sum_{i=1}^{n} r^{-l_i} \leq 1 = \sum_{i=1}^{n} p_i. \tag{1.3}$$

Mit (1.2) und (1.3) läßt sich folgendermaßen abschätzen:

$$\begin{aligned} E^r(Q) - L_C(Q) &= \sum_{i=1}^{n} p_i \log_r(\frac{1}{p_i}) - \sum_{i=1}^{n} p_i l_i = \sum_{i=1}^{n} p_i \log_r(\frac{1}{p_i} r^{-l_i}) \\ &\overset{(1.2)}{\leq} \frac{1}{\ln r} \sum_{i=1}^{n} p_i(\frac{1}{p_i} r^{-l_i} - 1) = \frac{1}{\ln r}(\sum_{i=1}^{n} r^{-l_i} - \sum_{i=1}^{n} p_i) \\ &\overset{(1.3)}{\leq} 0 \end{aligned}$$

Also gilt $E^r(Q) \leq L_C(Q)$ für jeden Präfix-Code C, auch für solche mit minimaler mittlerer Codewortlänge.

- Sei $l_i \in \mathbf{N}$ mit $-\log_r p_i \leq l_i < -\log_r p_i + 1$, $i = 1, \ldots, n$. Dann gilt $r^{-l_i} \leq p_i$ und daraus folgt die Gültigkeit der Kraftschen Ungleichung. Es existiert also mindestens ein Code C über einem Codealphabet der Größe r mit diesen Codewortlängen. Dann gilt aber:

$$L^r_{\text{opt}}(Q) \leq L_C(Q) = \sum_{i=1}^{n} p_i l_i < \sum_{i=1}^{n} -p_i \log_r p_i + \sum_{i=1}^{n} p_i = E^r(Q) + 1$$

◇

Bemerkung 1.33 Mit steigendem Informationsgehalt vergrößert sich auch die Entropie einer Quelle, die mittlere Codewortlänge wird nach unten durch die Entropie begrenzt. Deshalb werden bei steigendem Informationsgehalt durchschnittlich mehr Zeichen für die Codierung einer Nachricht benötigt.

Definition 1.34 *Sei C ein Code über einem Codealphabet der Größe r. Die* absolute Redundanz $R^r_{abs}(Q) = L_C(Q) - E^r(Q)$ *gibt an, wie komprimiert die Information der Quelle Q bei Codierung mit dem Code C übertragen wird. Dem gegenüber steht die* relative Redundanz $R^r_{rel}(Q) = 1 - E^r(Q)/L_C(Q)$; *$E^r(Q)/L_C(Q)$ gibt die im Mittel pro Symbol übertragene Information an.*

Satz 1.35 *Die absolute Redundanz liegt bei einem optimalen Code zwischen 0 und 1.*

Beweis 1.35 Für einen optimalen Code gilt nach Definition $L_C(Q) = L^r_{\text{opt}}(Q)$. Mit Satz 1.32 folgt: $0 \leq L_C(Q) - E^r(Q) < 1$. Daraus ergibt sich die Behauptung. $\diamond$

Kapitel 2

Huffman-Codes und Huffman-Bäume

Die Länge einer zu sendenden Nachricht ist eine wichtige Größe, da die Datenübertragungszeit und die Länge einer Nachricht direkt zusammenhängen. Die Dauer einer Datenübertragung stellt aber wiederum einen wichtigen Zeit- und Kostenfaktor dar. Darum ist man bemüht, die erwartete Länge der Codierung einer Nachricht möglichst klein zu halten, versucht also die mittlere Codewortlänge L_C zu minimieren. Ein Einzelzeichen-Code C, der diese Minimalitäts-Forderung erfüllt, wird *optimal* genannt. Dabei soll die Entschlüsselung einer Nachricht zu jedem Zeitpunkt der Übermittlung eindeutig möglich sein. Zu beachten ist, daß dies keinesfalls die bestmögliche Codierung liefert, denn mit Wortcodierung oder einem speziellen Code, der an den Inhalt einer Nachricht angepaßt ist, kann die durchschnittliche Codewortlänge weiter vermindert werden.

Huffman [12] löste als Erster die Aufgabe, einen optimalen Code zu gegebener statistischer Buchstabenverteilung algorithmisch zu konstruieren. Sein Verfahren ist heute allgemein bekannt und gehört zum Grundwissen eines jeden Informatikers und Mathematikers. In diesem Kapitel soll nicht nur der Algorithmus von Huffman angegeben werden, sondern auch die Vorgehensweise erklärt werden, die Huffman auf sein Lösungsverfahren brachte. Desweiteren werden Abschätzungen für die Redundanz und die Entropie optimaler Codes angegeben (Aigner [1], Gallager [5]).

2.1 Optimale Codes

Definition 2.1 *$a_1, \ldots, a_n$ seien Buchstaben des Quellalphabets $\mathcal{A}$. Die Größe des Codealphabets $\mathcal{B}$ sei r und o.B.d.A. sei $\mathcal{B} = \{ 0, \ldots, r-1 \}$. Die (statisti-*

sche) Auftrittshäufigkeit des i-ten Buchstabens a_i sei $p_i := p(a_i)$, die Länge der Codierung von a_i sei $l_i := l(f(a_i))$.

Aus Satz 1.12 folgt, daß nur Präfix-Codes für die Codierung in Frage kommen, ein optimaler Code ist hier folglich immer ein Präfix-Code. Ein optimaler Code weist einige besondere Eigenschaften auf, die im nächsten Satz dargestellt werden.

Satz 2.2 *Sei C ein optimaler Code. Dann ist C Präfix-Code und erfüllt folgende Bedingungen:*

1. *Es gibt eine Anordnung $a_1, \ldots, a_n$ der Buchstaben aus $\mathcal{A}$, so daß $p_1 \geq p_2 \geq \ldots \geq p_n$ und $l_1 \leq l_2 \leq \ldots \leq l_{n-1} = l_n$ ist.*
2. *Ist $\phi_{l_n-1}(a)$ der Präfix eines Codewortes der Länge l_n. Dann gibt es mindestens ein und höchstens $r-1$ Codewörter der Länge l_n mit identischem Präfix der Länge $l_n - 1$.*
3. *Jede Sequenz von $l_n - 1$ Zeichen über $\mathcal{B}$*
   - *ist Codewort,*
   - *hat einen Präfix, der Codewort ist, oder*
   - *ist Präfix eines Codewortes.*

Beweis 2.2 Sei ein optimaler Code C gegeben.

1. (a) Angenommen, es existiert ein Buchstabenpaar a_i, a_j mit $p_i > p_j$ und $l_i > l_j$. Dann kann man einen Code $\bar{C} = (\bar{f}, \mathcal{A}, \mathcal{B})$ definieren mit

$$\begin{aligned} \bar{f}(a_k) &= f(a_k) \quad \text{für } k \neq i, j \\ \bar{f}(a_j) &= f(a_i) \\ \bar{f}(a_i) &= f(a_j) \end{aligned}$$

   Bei diesen Code gilt dann $\bar{l}_i := l_j; \bar{l}_j := l_i$ und $\bar{l}_k := l_k$ für alle $k \neq i, j$. Daraus folgt $L_{\bar{C}} < L_C$ und dies ist ein Widerspruch zur Minimalität von C.

   (b) Angenommen, die Länge der Codierung von a_n ist größer als die Codierung von a_{n-1}, also $l_n > l_{n-1}$. Betrachtet man $\phi_{l_{n-1}}(a_n)$, so darf, weil C Präfix-Code ist, diese Zeichenfolge kein Codewort von C darstellen. Setzt man nun $f(a_n) = \phi_{l_{n-1}}(a_n)$, so ist eine Möglichkeit gefunden worden, a_n kürzer zu codieren, und damit kann L_C vorher nicht minimal gewesen sein.

2. Sei a ein Codewort der Länge l_n. Angenommen, alle Codewörter $f(a_i)$, $i \in I$,der Länge l_n haben einen Präfixe der Länge $l_n - 1$ mit $\phi_{l_n-1}(a) \neq \phi_{l_n-1}(a_i), i \in I$. Dann kann bei dem Wort $f(a)$ ohne Erzeugung identischer Codewörter das letzte Codierungszeichen weglassen werden, d.h. man setzt $f(a) = \phi_{l_n-1}(a)$. Dadurch verringert sich die mittlere Codewortlänge. Dies stellt einen Widerspruch zur Minimalität von C dar.
   Wenn mehr als r Codewörter der Länge l_n gleiche Präfixe der Länge $l_n - 1$ besitzen, können die Codewörter nicht mehr unterschiedlich sein, da der letzte Buchstabe der $f(a_i), i \in I$, nur aus einem Zeichenvorrat der Größe r gewählt werden kann.

3. Angenommen, es existiert eine Zeichenfolge der Länge $l_n - 1$, die keine der angegebenen Eigenschaften hat. Dann kann man $f(a_n)$ durch diese Zeichenfolge ersetzen und man erhält einen Präfix-Code mit kleinerer mittlerer Wortlänge.

$\diamond$

2.2 Huffmans Verfahren für ein binäres Codealphabet

Die Buchstaben aus $\mathcal{A} = \{ a_1, \ldots, a_n \}$ seien nach kleiner werdenden Wahrscheinlichkeiten sortiert, also geordnet wie in Satz 2.2.1; das Codealphabet sei $\mathcal{B} = \{0, 1\}$. Zu diesen Alphabeten soll eine minimale Präfix-Codierung gefunden werden.

Aufgrund von Satz 2.2.2 gibt es genau zwei Codewörter mit identischen Präfixen der Länge $l_n - 1$, nämlich a_n und a_{n-1}. Das bedeutet aber für die gesuchte Funktion f, daß o.B.d.A.

$$\begin{aligned} f(a_n) &= \phi_{l_n-1}(a_n)0 \\ f(a_{n-1}) &= \phi_{l_n-1}(a_n)1. \end{aligned}$$

Da $f(a_n)$ und $f(a_{n-1})$ induktiv definiert sind und dafür eine neue Sequenz (nämlich $\phi_{l_n-1}(a_n)$) nach einer Zuweisung verlangt, betrachtet man nun die Menge $\mathcal{A}^1 := \{a_1^1, \ldots, a_{n-1}^1\}$ mit:

$$\begin{aligned} a_i^1 := a_i \quad &\text{mit} \quad p(a_i^1) := p(a_i) \text{ für } i < n-1 \\ a_{n-1}^1 := \{a_{n-1}, a_n\} \quad &\text{mit} \quad p(a_{n-1}^1) := p(a_n) + p(a_{n-1}). \end{aligned}$$

Das ursprüngliche Problem, einen Code für die Menge $\mathcal{A}$ zu finden, hat sich nun auf das Problem reduziert, eine Menge zu codieren, die ein Symbol weniger enthält. Dazu verfährt man wie vorher: Die Menge wird sortiert

a_i	p_i	$f(a_i)$	l_i
a	2/32	1010	4
b	3/32	1011	4
c	5/32	100	3
d	5/32	110	3
e	6/32	111	3
f	11/32	0	1

$L_C = 79/32 \simeq 2,47$

Tabelle 2.1: Verteilung und Codierung der Zeichen des Quellalphabets zum Beispiel 2.3

gemäß Satz 2.2.1. Die beiden Symbole mit kleinster Wahrscheinlichkeit bekommen eine Codierung zugewiesen, die nur im letzten Zeichen verschieden ist. Diese Symbole werden aus der zu betrachtenden Menge entfernt und ihr gemeinsamer Präfix wird dafür hinzugenommen. Die zu betrachtende Menge verringert sich also in jedem Schritt um 1, daß heißt, die Menge $\mathcal{A}^{n-1}$ enthält genau ein Element a_1^{n-1} mit $p(a_1^{n-1}) = 1$. Wenn man diesen Zustand erreicht hat, ist jedem Zeichen a_i ein Code zugeordnet. Man muß nun lediglich diesen Code rückwärts über die Präfixe und die letzten Zeichen mittels der Gleichung

$$\phi_{l_n-1}(a_n) = f(\{a_{n-1}, a_n\}) = f(a_{n-1}^1)$$

erstellen.

Beispiel 2.3 Sei $\mathcal{A} = \{$a,b,c,d,e,f$\}$ und die Wahrscheinlichkeiten verteilt wie in Tabelle 2.1. Führt man die oben beschriebene Manipulation der $\mathcal{A}^i$ aus, erhält man Tabelle 2.2.

Ordnet man beim Zusammenfassen dem unteren Buchstaben der Tabelle die Null als letztes Zeichen der Codierung zu, dem oberen Buchstaben die Eins, so erhält man die Codierung von Tabelle 2.1.

$\mathcal{A}$	$p(\mathcal{A})$	$\mathcal{A}^1$	$p(\mathcal{A}^1)$	$\mathcal{A}^2$	$p(\mathcal{A}^2)$	$\mathcal{A}^3$	$p(\mathcal{A}^3)$	$\mathcal{A}^4$	$p(\mathcal{A}^4)$	$\mathcal{A}^5$	$p(\mathcal{A}^5)$
f	$\frac{11}{32}$	f	$\frac{11}{32}$	f	$\frac{11}{32}$	f	$\frac{11}{32}$	{a,b,c,d,e}	$\frac{21}{32}$	{a,b,c,d,e,f}	1
e	$\frac{6}{32}$	e	$\frac{6}{32}$	{a,b,c}	$\frac{10}{32}$	{d,e}	$\frac{11}{32}$	f	$\frac{11}{32}$		
d	$\frac{5}{32}$	d	$\frac{5}{32}$	e	$\frac{6}{32}$	{a,b,c}	$\frac{10}{32}$				
c	$\frac{5}{32}$	{a,b}	$\frac{5}{32}$	d	$\frac{5}{32}$						
b	$\frac{3}{32}$	c	$\frac{5}{32}$								
a	$\frac{2}{32}$										

Tabelle 2.2: Mengen $\mathcal{A}^i$ zum Beispiel 2.3

Bemerkung 2.4 Es ist erkennbar, daß das Huffman-Verfahren viele verschiedene Codes liefern kann. Diese verschiedenen Codes erhält man,

1. weil die Schlußzeichen $0, 1$ immer wieder anders zugewiesen werden können. So erhält man im obigen Beispiel mit demselben Verfahren 2^5 verschiedene Codes, indem man die Nullen und Einsen beim Zusammenfassen stets anders zuordnet. Die so erhaltenen Codes haben aber alle identische Wortlängenverteilungen, die Spalte l_i sieht also stets gleich aus.

2. weil Freiheiten in der Wahl der zusammenzufassenden Symbole vorliegen können. Dies ist möglich, wenn mehrere Symbole mit gleicher kleinster Wahrscheinlichkeit in einem $\mathcal{A}^i$ enthalten sind. Selbst wenn man festlegt, welches Zeichen die Null und welches die Eins am Ende erhält (im Beispiel erhielt immer das Zeichen mit geringerer Wahrscheinlichkeit die Null), ist der Code also nicht eindeutig festgelegt. In $\mathcal{A}^1$ zum Beispiel hätte man, statt {a,b} und c zusammenzufassen, auch c und d oder {a,b} und d zusammenfassen können. Eine Änderung an dieser Stelle beeinflußt unter Umständen auch die Werte der Spalte l_i, konstant bleibt aber die durchschnittliche Wortlänge L_C.

Huffmans Konstruktion liefert also keinesfalls *den* optimalen Code, sondern eine von der Instanz, die dieses Verfahren ausführt, abhängige Verschlüsselung.

2.3 Huffmans Verfahren für höher-dimensionale Codealphabete

Für höher-dimensionale Alphabete ist das Verfahren ähnlich wie das Verfahren für $r = 2$. Im Gegensatz zum binären Fall ist hier nicht selbstverständlich, daß Satz 2.2.3 erfüllt ist. Notwendig zur Erfüllung dieser Bedingung ist, daß bei jedem Übergang von einem $\mathcal{A}^i$ zu $\mathcal{A}^{i+1}$ genau r Zeichen zusammengefaßt werden. Da dies nicht für alle möglichen Größen $|\mathcal{A}|$ (die Anzahl der Symbole in $\mathcal{A}$) möglich ist, dürfen beim Übergang von $\mathcal{A}$ zu $\mathcal{A}^1$ weniger Symbole zusammengefaßt werden. Die Anzahl dieser Zeichen ist eindeutig bestimmt durch die Größen $|\mathcal{A}|$ und r und wird im folgenden Satz charakterisiert.

Satz 2.5 *Sei t die Anzahl der Zeichen, die in $\mathcal{A}$ zusammengefaßt werden. t ist dann die eindeutig bestimmbare Zahl zwischen 2 und r einschließlich, die den Bruch $\frac{n-t}{r-1}$ ganzzahlig macht.*

i	p(i)	f(i)	l(i)	
a	0.02	300	3	
b	0.05	301	3	
c	0.08	31	2	
d	0.1	32	2	$L_C = 1,47$
e	0.15	33	2	
f	0.18	0	1	
g	0.2	1	1	
h	0.22	2	1	

Tabelle 2.3: Verteilung und Codierung der Zeichen des Quellalphabets zum Beispiel 2.6

Beweis 2.5 $\mathcal{A}^1, \mathcal{A}^2, \ldots$ werden jeweils um $r-1$ Zeichen verringert, bis im n-ten Schritt nur noch ein Element in der Menge $\mathcal{A}^{n-1}$ übrig ist. Das bedeutet: Es existiert eine Zahl $j \in \mathbf{N}$ mit $|\mathcal{A}^1| - j(r-1) = 1$ bzw. $j = \frac{|\mathcal{A}^1|-1}{r-1}$. Weiterhin gilt $|\mathcal{A}^1| = n - t + 1$ und $2 \leq t \leq r$. Beides zusammen liefert die Aussage des Satzes. $\diamond$

Beispiel 2.6 Sei $\mathcal{A}$ ={a,b,c,d,e,f,g,h} und die Wahrscheinlichkeiten verteilt wie in der ersten Spalte der Tabelle 2.3, r sei 4, $\mathcal{B}$ ={0,1,2,3}. t kann dann die Werte 2,3,4 annehmen und $\frac{8-t}{3}$ soll eine natürliche Zahl sein. Folglich ist $t = 2$ und man erhält durch fortgesetztes Zusammenfassen die $\mathcal{A}^i$ wie in Tabelle 2.4.

Ordnet man die Schlußzeichen 0,1,2,3 wieder den zusammengefaßten Buchstaben von unten nach oben zu, erhält man die Codierung von Tabelle 2.3.

$\mathcal{A}$	$p(\mathcal{A})$	$\mathcal{A}^1$	$p(\mathcal{A}^1)$	$\mathcal{A}^2$	$p(\mathcal{A}^2)$	$\mathcal{A}^3$	$p(\mathcal{A}^3)$
h	0.22	h	0.22	{a,b,c,d,e}	0.4	{a,b,c,d,e,f,g,h}	1
g	0.2	g	0.2	h	0.22		
f	0.18	f	0.18	g	0.2		
e	0.15	e	0.15	f	0.18		
d	0.1	d	0.1				
c	0.08	c	0.08				
b	0.05	{a,b}	0.07				
a	0.02						

Tabelle 2.4: Mengen $\mathcal{A}^i$ zum Beispiel 2.6

Zusammenfassend kann folgender Satz formuliert werden:

Satz 2.7

- *Wird ein Einzelzeichen-Code gesucht, der zu jedem Zeitpunkt der Datenübertragung in der Lage ist, eine eindeutige Decodierung zu liefern, dann erstellt Huffmans Verfahren einen optimalen Code.*

- *Wird Huffmans Verfahren auf eine gegebene Buchstabenverteilung angewandt, dann liefert dies einen optimalen Präfix-Code, das heißt, es gibt keinen Präfix-Code mit kleinerer mittlerer Codewortlänge.*

2.4 Bäume mit minimalen Kosten

Ein Baum mit minimalen Kosten wird auch *optimal* genannt.

Sei C ein optimaler Präfix-Code zu den Alphabeten $\mathcal{A}, \mathcal{B}$; **B** sei der zugehörige Code-Baum. Die Codewörter befinden sich dann nach Satz 1.23 genau an den Blättern, die Kanten haben Gewichte aus $\{0, \ldots, r-1\}$. Auf diesem Baum wird eine Knotengewichtung eingeführt, indem den Blättern Gewichte zugeordnet werden mit

$$w(v) = p(a) \text{ genau dann, wenn } v \text{ das Wort } a \text{ repräsentiert.}$$

Dies liefert nach Bemerkung 1.21 eine vollständige Knotengewichtung. Da die Weglängen zu den Blättern in **B** mit den Längen der Codewörter übereinstimmen und die Gewichte entsprechend mit den Wahrscheinlichkeiten identifiziert wurden, erhält man folgende Aussage.

Satz 2.8 **B** *ist ein r-närer Baum mit n Blättern und minimalen Kosten unter allen r-nären Bäume mit n Blättern gleichen Gewichts.*

Sei andererseits **T** ein r-närer gewichteter Baum mit n Blättern und minimalen Kosten. Dann kann man mit Hilfe dieses Baumes einen Präfix-Code konstruieren, indem man die Kanten mit Zeichen aus $\{0, \ldots, r-1\}$ beschriftet, so daß Kanten mit gleichem Anfangsknoten unterschiedliche Bezeichnungen tragen.

Satz 2.9 *Der zu dem Baum* **T** *gehörige Präfix-Code hat ein Codealphabet mit r Zeichen und verschlüsselt ein Quellalphabet der Größe n. Dieser Code ist ein optimaler Präfix-Code über einem Codealphabet der Größe r bezüglich aller Quellalphabete mit gleicher Wahrscheinlichkeitsverteilung.*

Das Problem, einen Baum mit minimalen Kosten zu finden, steht also in direkter Beziehung zu dem Problem, einen optimalen Code zu finden. Mittels Satz 2.2 kann man die Eigenschaften eines solchen Baumes bestimmen.

Satz 2.10 *Sei* **B** *ein gewichteter r-närer Baum mit n Blättern und minimalen Kosten. Dann gelten folgende Aussagen:*

1. *Die Blätter können mit $v_1, \ldots, v_n$ so numeriert werden, daß $w(v_1) \geq w(v_2) \geq \ldots \geq w(v_n)$ und $l(v_1) \leq l(v_2) \leq \ldots \leq l(v_{n-1}) = l(v_n)$.*
2. *Alle inneren Knoten u mit $l(u) = l(v_n) - 1$ haben mindestens 2 und höchstens r Kinder.*
3. *Alle inneren Knoten u mit $l(u) < l(v_n) - 1$ haben genau r Kinder.*

Wegen der direkten Beziehung der Probleme ist man sofort in der Lage, einen Algorithmus anzugeben, der einen optimalen r-nären Baum mit n Blättern konstruiert.

2.5 Huffmans Algorithmus

Im allgemeinen wird man Huffmans Algorithmus auf Bäume anwenden, da dort die Manipulation der Mengen $|\mathcal{A}^i|$ durch den Algorithmus einfacher zu verfolgen ist. Wegen der Äquivalenz der beiden Problemstellungen „finde optimalen Code“, „finde optimalen Baum“ genügt es, wenn die Lösung eines Problems algorithmisch dargestellt wird.

Um den Algorithmus kurz aufschreiben zu können, werden einige Vereinbarungen getroffen, die auch für spätere Algorithmen gelten sollen. Sei **B** ein Baum mit Wurzel x, dann soll x der *Vertreter des Baumes* sein und der Baum wird $[x]$ genannt. w ist stets eine Gewichtsfunktion. Die gleiche Funktion wird benutzt, um einerseits einem Knoten ein Gewicht zuzuordnen und andererseits Kanten mit einem Gewicht zu belegen.

Die $[v_i]$ in der Initialisierung von $\mathcal{M}$ sind Bäume, die nur aus dem Knoten v_i bestehen.

Der Algorithmus ist in $\mathrm{O}(n \log_2 n)$ ausführbar, wenn die Menge $\mathcal{M}$ entsprechend verwaltet wird. Dazu ist ein Sortieren von $\mathcal{M}$ nach der Eingabe notwendig (mit Heap-Sort in $\mathrm{O}(n \log_2 n)$ möglich). Das Finden der t kleinsten Elemente ist dann in konstanter Zeit realisierbar, wenn das jeweils neue Element $[v]$ an die richtige Stelle in $\mathcal{M}$ eingefügt wird, dies kostet $\mathrm{O}(\log_2 n)$. Da die Schleife $\mathrm{O}(n)$ mal durchlaufen wird, kommt man zu oben genannter Abschätzung.

```
PROCEDURE Huffman(r:INTEGER, v_1, ..., v_n:gewichtete Knoten):Baum;
VAR t:INTEGER;
      v:Knoten;
      M:Menge von Bäumen;
BEGIN
M := {[v_1], ..., [v_n]};
Bestimme t so, daß 2 ≤ t ≤ r und (n-t)/(r-1) ganzzahlig;
WHILE |M| > 1 DO
    Finde [u_1], ..., [u_t] ∈ M mit den t kleinsten Wurzelgewichten;
    [v] := (v, [u_1], ..., [u_t]);
    w(v) := Σ_{i=1}^t w(u_i);
    M :={M \ ∪_{i=1}^t [u_i]} ∪ {[v]};
    t := r;
END;
RETURN M;
END.
```

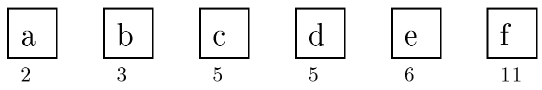
PROCEDURE Huffman(r:INTEGER, $v_1, \ldots, v_n$:gewichtete Knoten):Baum;
VAR t:INTEGER;
v:Knoten;
$\mathcal{M}$:Menge von Bäumen;
BEGIN
$\mathcal{M} := \{[v_1], \ldots, [v_n]\}$;
Bestimme t so, daß $2 \leq t \leq r$ und $\frac{n-t}{r-1}$ ganzzahlig;
WHILE $|\mathcal{M}| > 1$ DO
Finde $[u_1], \ldots, [u_t] \in \mathcal{M}$ mit den t kleinsten Wurzelgewichten;
$[v] := (v, [u_1], \ldots, [u_t])$;
$w(v) := \sum_{i=1}^{t} w(u_i)$;
$\mathcal{M} := \{\mathcal{M} \setminus \bigcup_{i=1}^{t} [u_i]\} \cup \{[v]\}$;
$t := r$;
END;
RETURN $\mathcal{M}$;
END.

Tabelle 2.5: Algorithmus von Huffman

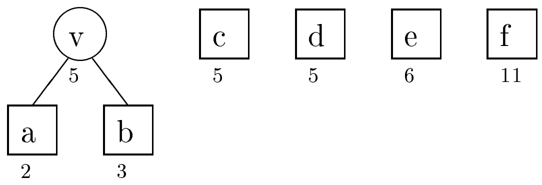
Sind die $[v_i]$ bereits vorsortiert, dann hat der Algorithmus bei geeigneter Verwaltung der Menge $\mathcal{M}$ eine $O(n)$-Komplexität: die $[v_i]$ werden in einem Stack geordnet abgelegt, das kleinste Element befindet sich oben. Neue Elemente, die $\mathcal{M}$ hinzugefügt werden sollen, werden in einer Warteschlange abgelegt. So kann man die zwei kleinsten Elemente in jedem Schleifen-Durchlauf in konstanter Zeit finden, da sie sich in der 4-elementigen Menge befinden, die aus den zwei obersten Stack-Elementen und den zwei vordersten Elementen der Warteschlange besteht. Da die Schleife $O(n)$ mal durchlaufen wird, liefert dies eine $O(n)$-Komplexität.

Beispiel 2.11 Führt man diesen Algorithmus mit den Knoten und der Statisik von Beispiel 2.3 aus, durchläuft die Menge $\mathcal{M}$ folgende Stadien:

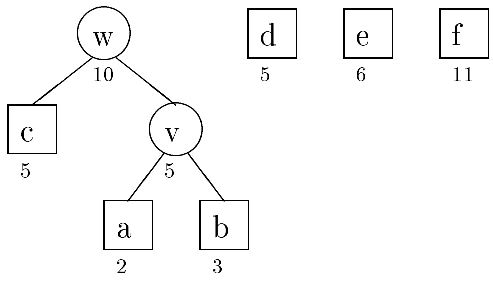
Eingabemenge



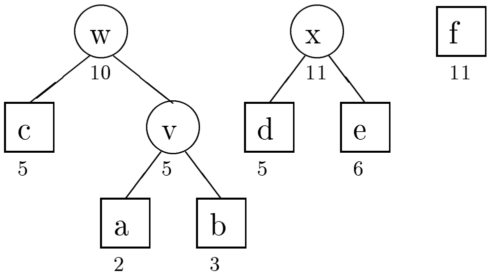
1.ter Durchlauf,
zusammenfassen
von a und b



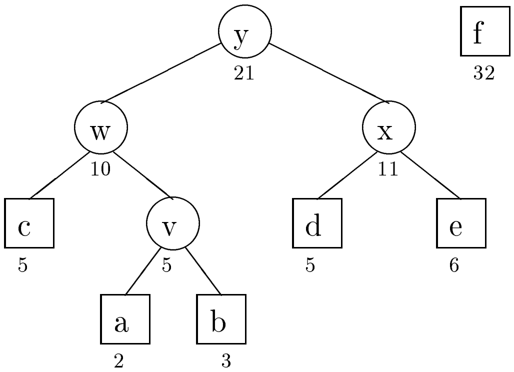
2.ter Durchlauf,
zusammenfassen
von c und v



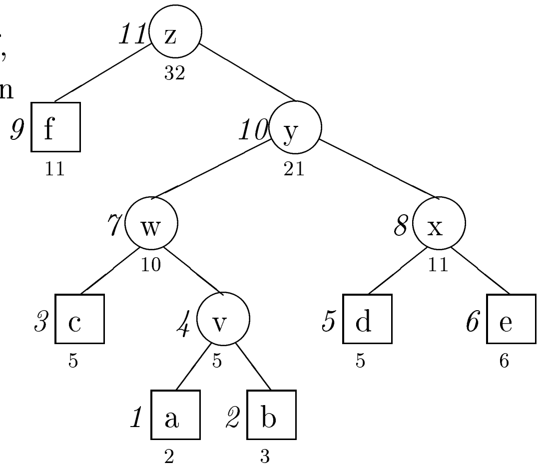
3.ter Durchlauf,
zusammenfassen
von d und e



4.ter Durchlauf,
zusammenfassen
von w und x



5.ter Durchlauf,
zusammenfassen
von f und y



2.6 Huffman-Bäume

Die Knoten eines Huffman-Baumes kann man in einfacher Art und Weise anordnen. Wie dies funktioniert, soll hier erklärt werden. Diese Erläuterung findet nur für $r = 2$ statt, wir beschränken uns auf binäre Codes und ihre Code-Bäume.

Definition 2.12 *Ein regulärer gewichteter binärer Code-Baum mit n Blättern erfüllt die* Geschwister-Bedingung, *wenn die m Knoten x_k des Baumes so aufgelistet werden können, daß*

- $w(x_1) \leq w(x_2) \leq \ldots \leq w(x_m)$,
- *Geschwisterknoten im Baum in der Liste aufeinander folgen.*

Die dieser Liste entsprechende Numerierung der Knoten eines Baumes wird Huffman-Numerierung *genannt, und ein Baum, der die Geschwister-Bedingung erfüllt, heißt* Huffman-Baum.

Bemerkung 2.13

1. Die Liste enthält $m = 2n - 1$ Elemente (siehe Bemerkung 1.20.3).
2. Das $2i - 1$.te Element und das $2i$.te Element $(i = 1, \ldots, n - 1)$ sind im Baum Geschwister.
3. Eltern haben stets eine höhere Nummer als ihre Kinder.

Satz 2.14 *Ein Baum kann genau dann mit dem Huffman-Algorithmus erzeugt werden, wenn er ein Huffman-Baum ist. Deshalb hat jeder Huffman-Baum minimale Kosten.*

Beweis 2.14

- Sei ein Baum gegeben, der aus dem Huffman-Algorithmus entstanden ist. Legt man während der Ausführung des Algorithmus eine Liste an, in der jeweils die beiden zu entfernenden Knoten so eingetragen werden, daß der Knoten mit kleinerem Gewicht zuerst eingetragen wird, dann erhält man eine Auflistung der Knoten, die mit der Geschwister-Bedingung verträglich ist.
- Sei ein Huffman-Baum gegeben. Dann existiert eine Liste, die die Geschwister-Bedingung erfüllt. Führe den Huffman-Algorithmus nun folgendermaßen aus:
  - nimm für die zusammenzufassenden Symbole stets die Symbole, die die kleinsten Nummern unter allen noch vorhandenen Symbolen besitzen

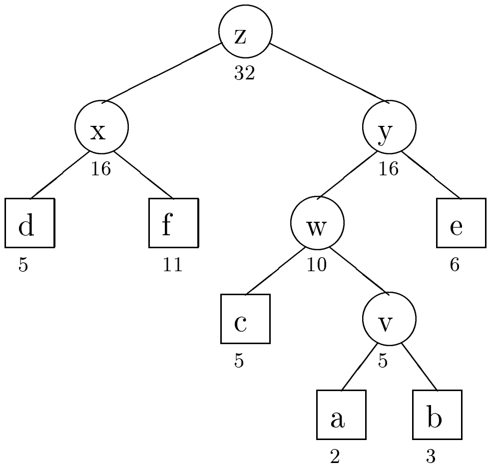
– beim Zusammenfassen sind die Symbole so anzuordnen, daß sie der gewünschten Baumform entsprechen.

Der Baum, der aus diesem Verfahren resultiert, ist identisch mit dem gegebenen Baum.

◇

Die Numerierung des Code-Baumes aus Beispiel 2.3 ist im 5.ten Durchlauf von Beispiel 2.11 angegeben. Diese Numerierung entspricht der Reihenfolge der vom Algorithmus ausgewählten Knoten.

Bemerkung 2.15 Es gibt optimale Bäume, die keine Huffman-Bäume sind. Das bedeutet, daß nicht jeder optimale Baum durch Huffmans Algorithmus erstellt werden kann. Ein Beispiel dafür ist der nebenstehende Baum, der ein optimaler Baum ist zu den gleichen Gewichten wie in Beispiels 2.3.



2.7 Redundanz von optimalen Codes

Dieses Kapitel gibt obere Schranken für die minimale mittlere Codewortlänge an. Daraus lassen sich Aussagen über die absolute Redundanz von optimalen Codes ableiten. Es werden also weitere obere Schranken für die Redundanz vorgestellt. Für optimale Codes wurde in Satz 1.35 bereits eine Schranke mit dem Wert Eins gefunden.

Satz 2.16 *Sei Q_1 eine Quelle mit n Zeichen mit beliebiger Wahrscheinlichkeitsverteilung. Q_2 sei eine Quelle mit n Zeichen, die alle gleiche Wahrscheinlichkeit besitzen, also $p_i = \frac{1}{n}$ für $i = 1, \ldots, n$. Dann stehen die minimalen mittleren Codewortlängen in folgendem Verhältnis zueinander:*

$$L^r_{opt}(Q_1) \leq L^r_{opt}(Q_2).$$

Beweis 2.16 Bei Gleichverteilung der Wahrscheinlichkeiten haben $n - x$ Zeichen eine Codierung maximaler Wortlänge L, die restlichen x Zeichen haben eine Codierung der Länge $L-1$. Deshalb gilt für die minimale mittlere Codewortlänge der Quelle Q_2:

$$L^r_{\text{opt}}(Q_2) = \frac{1}{n}x(L-1) + \frac{1}{n}(n-x)L = L - \frac{1}{n}x. \tag{2.1}$$

Seien die Wahrscheinlichkeiten der Quelle Q_1 fallend geordnet, also $p_1 \geq p_2 \geq \ldots \geq p_n$. Dann gilt

$$\sum_{i=1}^{k} p_i \geq \frac{k}{n} \text{ für alle } k = 1, \ldots, n. \tag{2.2}$$

(Denn gilt dies nicht, gibt es ein kleinstes k mit $\sum_{i=1}^{k} p_i < \frac{k}{n}$. Daraus folgt

$$\frac{k}{n} > \sum_{i=1}^{k} p_i = \sum_{i=1}^{k-1} p_i + p_k \geq \frac{k-1}{n} + p_k$$

und somit $p_k < \frac{1}{n}$ und $p_j < \frac{1}{n}$ für alle $j > k$. Damit erhält man einen Widerspruch zur Gleichung $\sum_{i=1}^{n} p_i = 1$, da

$$\sum_{i=1}^{n} p_i = \sum_{i=1}^{k} p_i + \sum_{i=k+1}^{n} p_i < \frac{k}{n} + \frac{n-k}{n} = 1.)$$

Nun konstruiert man einen Code C zur Quelle Q_1, indem man den Symbolen mit den Wahrscheinlichkeiten $p_1, \ldots, p_x$ Codewörter der Länge $L - 1$, den Symbolen mit den Wahrscheinlichkeiten $p_{x+1}, \ldots, p_n$ Codewörter der Länge L zuweist, wobei das L der Quelle Q_2 verwendet wird. Die mittleren optimalen Codewortlängen genügen dann folgender Ungleichung:

$$\begin{aligned} L^r_{\text{opt}}(Q_1) &\leq L^r_C(Q_1) = \sum_{i=1}^{x} p_i(L-1) + \sum_{i=x+1}^{n} p_i L = L - \sum_{i=1}^{x} p_i \stackrel{(2.2)}{\leq} L - \frac{x}{n} \\ &\stackrel{(2.1)}{=} L^r_{\text{opt}}(Q_2). \end{aligned}$$

◇

Satz 2.17 *Sei Q eine Quelle mit n Zeichen. Dann gilt für die minimale mittlere Codewortlänge folgende Ungleichung:*

$$L^r_{opt}(Q) \leq \lceil \log_r n \rceil - \frac{1}{n} \lfloor \frac{r^{\lceil \log_r n \rceil} - n}{r-1} \rfloor.$$

Sind die Wahrscheinlichkeiten gleichverteilt, dann gilt Gleichheit.

Beweis 2.17 Die Werte x und L aus Formel (2.1) genügen folgenden Gleichungen:

$$L = \lceil \log_r n \rceil$$
$$x = \lfloor \frac{r^L - n}{r-1} \rfloor.$$

Setzt man diese Gleichungen in Formel (2.1) ein und berücksichtigt Satz 2.16, ergibt sich die Behauptung. ⋄

Satz 2.18 *Sei Q eine Quelle mit n Zeichen. Dann läßt sich die minimale mittlere Codewortlänge durch folgende obere Schranke abschätzen:*

$$L^r_{opt}(Q) \leq \log_r n + \frac{1}{r-1} - \log_r e + \log_r(\log_r e) + \log_r(r-1) + \frac{\delta}{n},$$

wobei δ die Zahl zwischen 0 und 1 ist, die angibt, um wieviel $\frac{r^{\lceil \log_r n \rceil} - n}{r-1}$ nach unten abgerundet werden muß, um ganzzahlig zu werden.
Im binären Fall vereinfacht sich die Schranke zu folgender:

$$L^2_{opt}(Q) \leq \log_2 n + 1 - \log_2 e + \log_2(\log_2 e) \simeq \log_2 n + 0.086$$

Beweis 2.18 Man schreibt $\lceil \log_r n \rceil = \log_r n + \theta$ und $\lfloor \frac{r^L - n}{r-1} \rfloor = \frac{r^L - n}{r-1} + \delta$; $\theta, \delta \in [0,1]$ und setzt dies in Satz 2.17 ein. Dann ergibt sich

$$\begin{aligned}
L^r_{\text{opt}}(Q) &= \log_r n + \theta - \frac{1}{n}\frac{r^L}{r-1} + \frac{1}{r-1} + \frac{\delta}{n} \\
&= \log_r n + \theta - \frac{r^\theta}{r-1} + \frac{1}{r-1} + \frac{\delta}{n} \\
&= \log_r n + \frac{1}{r-1}(\theta(r-1) - r^\theta + 1) + \frac{\delta}{n} \\
&=: \log_r n + \frac{1}{r-1}\psi(\theta) + \frac{\delta}{n}.
\end{aligned}$$

Die Funktion $\psi(\theta) = \theta(r-1) - r^\theta + 1$ ist im Intervall [0,1] nicht-negativ und nimmt ihr Maximum an für $r^\theta = \frac{r-1}{\ln r}$, also bei

$$\theta = \log_r(r-1) + \log_r(\log_r e).$$

Einsetzen liefert die Behauptung.
Im binären Fall ist δ stets Null. ⋄

Satz 2.19 *Für die absolute, binäre Redundanz einer gleichverteilten Quelle gilt:*

$$R^2_{abs} \leq 1 - \log_2 e + \log_2(\log_2 e) \simeq 0.086.$$

Beweis 2.19 Dies ergibt sich aus Satz 2.18 und $E^r(Q) = \log_r n$. $\diamond$

Satz 2.20 *Sei Q eine Quelle mit n Zeichen, $\mathcal{B}$ sei ein Codealphabet der Größe r. p_1 sei die größte auftretende Wahrscheinlichkeit. C sei ein Huffman-Code. Dann gelten folgende Abschätzungen für die absolute Redundanz:*

1. $R^r_{abs}(Q) \leq \frac{p_1 r}{\ln r} + \frac{1}{r-1} - \log_r e + \log_r(\log_r e) + \log_r(r-1)$.
2. $R^2_{abs}(Q) \leq p_1 + 1 - \log_2 e + \log_2(\log_2 e) \simeq p_1 + 0.086$.
3. *Falls $p_1 \geq 1/2$ ist, dann gilt folgende, bessere Abschätzung:*
$$R^2_{abs}(Q) \leq 2 + p_1 \log_2 p_1 + (1-p_1)\log_2(1-p_1) - p_1 \leq p_1.$$

Beweis 2.20 Es wird nur (2) und (3) bewiesen, also nur die Ungleichungen für den binären Fall. Im allgemeinen Fall (1) benutzt man einen ähnlichen Beweis wie in (2).

Da C ein Huffman-Code ist, besitzt der zugehörige Codebaum **B** eine Huffman-Numerierung. Die Knoten dieses Baumes werden mit einer *-Numerierung folgendermaßen versehen: Man legt eine Liste an, die mit der Geschwister-Bedingung verträglich ist. Die *-Numerierung ist eine Umkehrung der Huffman-Numerierung ohne die Wurzel. Die Wahrscheinlichkeiten der Knoten werden mit q_i bezeichnet, $i = 1, \ldots, 2n-2$. Dann gilt mit Satz 1.22
$$L^2_C(Q) = \sum_{i=1}^{2n-2} q_i. \tag{2.3}$$
Sei $\varphi(x) := -x\log_2(x) - (1-x)\log_2(1-x)$, dann kann man die Entropie folgendermaßen aufschreiben:
$$E^2(Q) = \sum_{i=1}^{n-1}(q_{2i-1} + q_{2i})\varphi(\frac{q_{2i}}{q_{2i-1} + q_{2i}}). \tag{2.4}$$
(Dies sieht man so:
$$\begin{aligned}
E^2(Q) &= -\sum_{u\in\beta(\mathbf{B})} p(u)\log_2 p(u) \\
&= -\sum_{i=1}^{2n-2} q_i \log_2 q_i + \sum_{u\in\gamma(\mathbf{B})} p(u)\log_2 p(u) \\
&= -\sum_{i=1}^{n-1}(q_{2i-1}\log_2 q_{2i-1} + q_{2i}\log_2 q_{2i}) + \sum_{i=1}^{n-1}(q_{2i-1} + q_{2i})\log_2(q_{2i-1} + q_{2i}) \\
&= \sum_{i=1}^{n-1}(q_{2i-1} + q_{2i})\varphi(\frac{q_{2i}}{q_{2i-1} + q_{2i}}).)
\end{aligned}$$

Aus Formel 2.3 und Formel 2.4 erhält man folgende Darstellung für die absolute Redundanz

$$R^2_{\text{abs}}(Q) = \sum_{i=1}^{n-1}(q_{2i-1} + q_{2i})(1 - \varphi(\frac{q_{2i}}{q_{2i-1} + q_{2i}})). \tag{2.5}$$

Seien j,l und M Zahlen mit folgenden Eigenschaften:

- **B** hat in der l-ten Ebene $2^l =: M$ Knoten, deren Wahrscheinlichkeiten werden mit $\bar{q}_1 \geq \bar{q}_2 \geq \ldots \geq \bar{q}_M$ bezeichnet. **B** ist also in der l.ten Ebene vollständig.
- mindestens einer dieser Knoten ist innerer Knoten
- j ist die kleinste Zahl, so daß sich der Knoten mit der Wahrscheinlichkeit q_{2j-1} in der Ebene $l + 1$ befindet.

Dann gilt

$$\sum_{i=1}^{M} \bar{q}_i = 1. \tag{2.6}$$

Splittet man Formel (2.5) in zwei Summen auf und bringt Formel (2.6) und Formel (2.4) zur Anwendung, erhält man folgende Darstellung für die Redundanz:

$$R^2_{\text{abs}} = l + \sum_{i=1}^{M} \bar{q}_i \log_2 \bar{q}_i + \sum_{i=j}^{n-1}(q_{2i-1} + q_{2i})(1 - \varphi(\frac{q_{2i}}{q_{2i-1} + q_{2i}})). \tag{2.7}$$

Ist $x \in [0, \frac{1}{2}]$, dann gilt $\varphi(x) \geq 2x$. (Da $\varphi'(x) < 0$ für $x \in [0, \frac{1}{2}]$ und $\varphi(\frac{1}{2}) = 1$.) Diese Abschätzung darf man in Formel (2.7) anwenden, da stets $\frac{q_{2i}}{q_{2i-1}+q_{2i}} \leq \frac{1}{2}$ gilt:

$$R^2_{\text{abs}} \leq l + \sum_{i=1}^{M} \bar{q}_i \log_2 \bar{q}_i + \sum_{i=j}^{n-1}(q_{2i-1} - q_{2i}).$$

Dies läßt sich aufgrund der Ordnung der q_i weiter abschätzen und somit

$$R^2_{\text{abs}} \leq l + \sum_{i=1}^{M} \bar{q}_i \log_2 \bar{q}_i + q_{2j-1}. \tag{2.8}$$

Sei l_1 die Länge eines kürzesten Codewortes, welches die Wahrscheinlichkeit p_1 hat.

- Beweis von (3):
  Ist $p_1 \geq \frac{1}{2}$, dann ist $l_1 = 1$. Setzt man in Formel (2.8) $l = 1$, dann erhält man $j = 2$ und $M = 2$ und somit

  $$R_{\text{abs}}^2 \leq 1 - \varphi(p_1) + q_3.$$

  Mit $q_3 \leq 1 - p_1$ erhält man die erste Ungleichung des Satzes.
  Die zweite Ungleichung gilt, da $2 - \varphi(p_1) - 2p_1 = 0$ für $p_1 = 0$ und $p_1 = 1$ und die Ableitung monoton fallend ist auf $[0, \frac{1}{2}]$.

- Beweis von (2):
  Wenn alle Codewörter die gleiche Länge besitzen, dann setze $l := l_1 - 1$, ansonsten wird $l := l_1$ gesetzt. Dann gilt stets $q_{2j-1} \leq p_1$ und aufgrund der Eigenschaften eines Huffman-Baumes gilt $\bar{q}_M \geq \bar{q}_1/2$ und $\sum_{i=1}^{M} \bar{q}_i = 1$.

  Sei Q die Menge aller Vektoren aus $\mathbf{R^M}$, die diese Bedingungen erfüllen, dann gilt

  $$R_{\text{abs}}^2 \leq l + \max_{(\bar{q}_1,\ldots,\bar{q}_M)\in Q} (\sum_{i=1}^{M} \bar{q}_i \log_2 \bar{q}_i) + p_1.$$

  Das Maximum kann aber nur an Randpunkten von Q angenommen werden, dieses sind Vektoren mit $\bar{q}_i = \bar{q}_1$ für $i \leq x$, $\bar{q}_i = \bar{q}_1/2$ für $i > x$, $x \in \mathbf{N}$. Daraus folgt wiederum $\bar{q}_1 = \frac{2}{M+x}$ und man bekommt

  $$R_{\text{abs}}^2 = l + \max_{x\in[1,M]} (\frac{2x}{M+x} + \log_2 \frac{1}{M+x}) + p_1.$$

  Das Maximum wird angenommen für $x = \frac{2M}{\log_2 e} - M$ und daraus folgt die Ungleichung.

$\diamond$

Kapitel 3

Optimale alphabetische Bäume und Codes

Man kann Wörter lexikographisch ordnen, wenn auf den ihnen zugrundeliegenden Buchstaben eine Ordnung definiert ist. Ist die Ordnung beispielsweise a,A,b,B,..., dann steht „als“ in der Ordnung vor „Affe“ (man schreibt dann „als“ < „Affe“). Zur (lexikographischen) Ordnung von Codewörtern ist also eine Ordnung auf dem Codealphabet notwendig. Diese Ordnung sei 0,1,2,....

Ein Code heißt *alphabetisch*, wenn die Codewörter und die zugehörigen Quellbuchstaben in gleicher Weise geordnet sind. f ist also eine alphabetische Codierung, wenn für alle Quellbuchstaben genau dann $a < b$ gilt, wenn $f(a) < f(b)$ ist.

Betrachtet man von links nach rechts die Blätter eines Codebaumes zu einem alphabetischen Code, dann spiegelt diese Reihenfolge die lexikographische Ordnung der Codewörter wider. Deshalb heißt ein Baum *alphabetisch*, wenn diese Reihenfolge der Blätter identisch ist mit einer vorgegebenen Ordnung – nämlich der gegebenen Ordnung der Quellbuchstaben. Diese Ordnung wird *Konstruktions-Sequenz* genannt, da Algorithmen angegeben werden, die zu einer Ordnung einen alphabetischen Baum konstruieren.

Die wichtigste Anwendung von alphabetischen Bäumen sind Suchbäume, die unter anderem zum Lokalisieren vorhandener Daten benutzt werden.

In diesem Kapitel werden Algorithmen vorgestellt, die optimale binäre alphabetische Bäume/Codes konstruieren. Diese sind nicht immer optimal im Sinne von Kapitel 1 und 2.

Gilbert und Moore [8] stellen einen Algorithmus mit einer Komplexität von $O(n^3)$ vor, der optimale alphabetische Bäume/Codes auf Teilmengen konstruiert und diese mittels dynamischer Programmierung erweitert, bis die gesamte Menge erfaßt wird.

Knuth [15] zeigt, daß dieser Algorithmus sich verbessern läßt auf eine

Komplexität von $O(n^2)$.

Hu und Tucker [11] lösen das Problem ebenfalls in $O(n^2)$, indem sie in einer Klasse von Bäumen (den T.C.L. Bäumen), die die alphabetischen Bäume enthält, einen optimalen Baum suchen. Dieser wird dann in einen alphabetischen Baum umgewandelt – siehe dazu ebenfalls Hu und Tan [10]. Dieser Algorithmus hat bei Benutzung einer geeigneten Datenstruktur eine verbesserte Laufzeit von $O(n \log_2 n)$.

Garsia und Wachs [7] verändern den Algorithmus von Hu und Tucker geringfügig, so daß sich der Korrektheitsbeweis vereinfacht und die Laufzeit von $O(n \log_2 n)$ erhalten bleibt.

3.1 Allgemeine Sätze

Eine geordnete Liste $(l_1, \ldots, l_n) \in \mathbf{N}^n$ heißt *realisierbare Blattlängenliste*, wenn es einen regulären Baum gibt, dessen von links nach rechts geordnete Blätter diese Entfernungen von der Wurzel aufweisen.

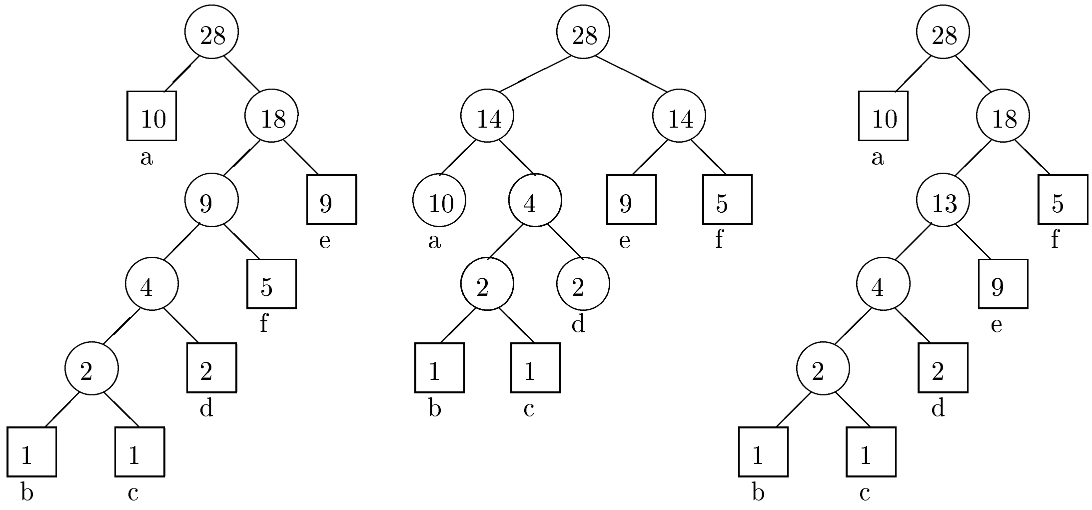
Gesucht wird ein Entscheidungskriterium zur Realisierbarkeit einer Blattlängenliste. Die Kraftsche Ungleichung 1.24 genügt nicht, da man mit ihr lediglich entscheiden kann, ob ein ungeordneter Baum mit diesen Blattlängen existiert. Nur wenn die Blattlängen nach Größe geordnet sind, läßt sich mit der Kraftschen Ungleichung entscheiden, ob ein zugehöriger alphabetischer Baum existiert.

Satz 3.1 *Sei $F = (l_1, \ldots, l_n)$ eine Liste von Blattlängen und $l := \max_{1 \leq i \leq n} l_i$. Sei $\bar{F}$ die reduzierte Liste, die entsteht, wenn man jedes Paar „l, l" in F durch den Wert „$l-1$" ersetzt. Die Liste ist genau dann realisierbar, wenn eine der folgenden Bedingungen erfüllt ist*

- *$F = (1, 1)$.*
- *Der Wert l tritt in der Liste nur in Folgen gerader Länge auf und $\bar{F}$ ist eine realisierbare Liste.*

Beweis 3.1

- Sei $F \neq (1,1)$ eine realisierbare Liste und B_F der zugehörige reguläre Baum. Da jeder Knoten auf der untersten Ebene l ein Geschwisterkind besitzt, tritt der Wert l in der Liste paarweise auf. Sei B der Baum, der durch Entfernen aller Blätter auf Ebene l entsteht. Dann werden ihre Väter zu neuen Blättern und es gilt $B = B_{\bar{F}}$. Also ist auch $\bar{F}$ realisierbar.



Huffman-Baum mit Kosten 61

optimaler alphabetischer Baum mit Kosten 62

Baum, den man aus der naiven Konstruktion erhält mit Kosten 65

Abbildung 3.1: Bäume zu den Gewichten 10,1,1,2,9,5

- $F = (1,1)$ ist realisierbar.
  Sei F eine Liste von Zahlen und $F_1, \dots, F_k$ die Folge von reduzierten Listen mit $F_k = (1,1)$. Aus dieser Folge konstruiert man (beginnend bei F_k) folgendermaßen einen Baum: Ist in F_i der Wert q aus einem Paar $q+1, q+1$ aus der Liste F_{i-1} entstanden, dann besitzt der zum Wert q gehörige Knoten zwei Kinder. Auf diese Weise erhält man einen Baum zur Liste F und diese ist somit realisierbar.

$\diamond$

3.2 Ein erster Versuch

Jeden alphabetischen Baum zu einer gegebenen Konstruktions-Sequenz kann man folgendermaßen in $n-1$ Schritten konstruieren: Ersetze je zwei in der Konstruktionssequenz benachbarte Knoten durch einen neuen Knoten, der die beiden ersetzten Knoten als Kinder besitzt (dabei ist der links liegende Knoten das linke Kind des neuen Knoten). Ist nur noch ein Knoten vorhanden, dann ist dieser die Wurzel eines alphabetischen Baumes.

Huffmans Algorithmus liefert die Grundlage für folgende Konstruktion optimaler alphabetischer Bäume: Es werden die zwei Knoten zu einem neuen

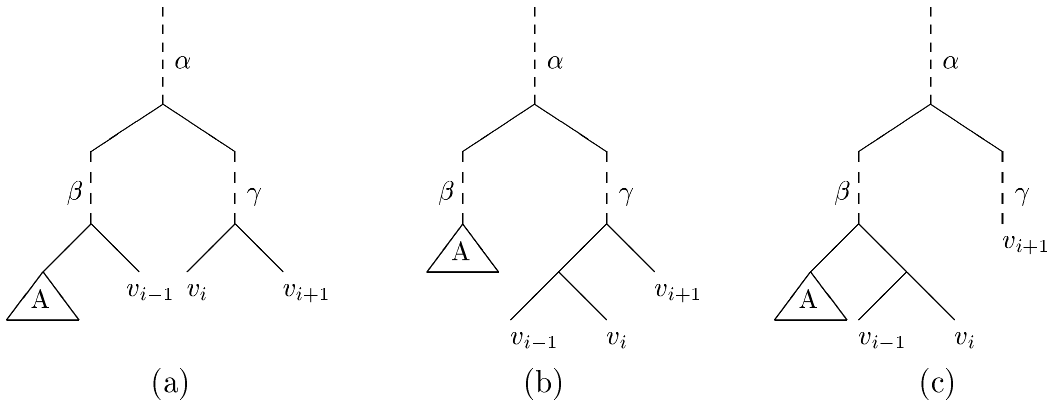


Abbildung 3.2: auszutauschende Strukturen zum Beweis 3.2

Knoten kombiniert, die minimales gemeinsames Gewicht unter allen benachbarten Knoten besitzen. Unglücklicherweise liefert diese Vorgehensweise zwar einen alphabetischen Baum, dieser ist jedoch nicht unbedingt optimal (siehe Abbildung 3.1).

Eine andere Möglichkeit ist die Reduktion der Ausgangsmenge unter Ausnutzung der folgenden Sätze nach Gilbert und Moore [8]. Falls diese Methode einen Baum liefert, dann ist dieser auch optimal alphabetisch. Leider erhält man durch diese Reduktionen oftmals nur einen Wald, aus dem man alle möglichen alphabetischen Bäume konstruieren muß. Unter diesen Bäumen muß dann jener mit kleinsten Kosten ausgewählt werden, dieser ist der optimale alphabetische Baum.

Satz 3.2 *Sei v_i ein Blatt mit kleinster Wahrscheinlichkeit w_i und T ein optimaler alphabetischer Baum auf den Blättern $v_1, \ldots, v_n$.*

1. *v_i hat entweder v_{i-1} oder v_{i+1} als Geschwisterkind.*
2. *Ist $w_{i+1} > w_i + w_{i-1}$, dann sind v_i und v_{i-1} Geschwister.*
3. *Ist $w_{i-1} > w_i + w_{i+1}$, dann sind v_i und v_{i+1} Geschwister.*

Beweis 3.2

1. Folgt aus Satz 2.2 und der Ordnung der Blätter in einem alphabetischen Baum.
2. Angenommen, v_i und v_{i-1} sind nicht Geschwister, dann sind wegen (1) v_i und v_{i+1} Geschwister. Dann enthält T die Baumstruktur (a) aus Abbildung 3.2.

- Ist $|\gamma| < |\beta|$, dann ist $|\beta| \geq 1$ und somit enthält A mindestens das Blatt v_{i-2}. Betrachte den Baum T_1, der aus T entsteht, indem Struktur (a) durch (b) ersetzt wird. Dann gilt

  $$K_T - K_{T_1} = \sum_{x \in A} w_x + (|\beta| - |\gamma| - 1)w_{i-1} - w_i \geq w_{i-2} - w_i \overset{w_i \text{ min.}}{>} 0$$

  und somit $K_{T_1} < K_T$, was einen Widerspruch zur Optimalität von T darstellt.
- Ist $|\gamma| \geq |\beta|$, dann betrachte den Baum T_2, der aus T entsteht, indem (a) durch (c) ersetzt wird. Dann gilt

  $$K_T - K_{T_2} = -w_{i-1} + (|\gamma| - |\beta| - 1)w_i + w_{i+1} \geq -w_{i-1} - w_i + w_{i+1} \overset{Vor.}{>} 0$$

  und somit $K_{T_2} < K_T$, was einen Widerspruch zur Optimalität von T darstellt.

3. Analog zu (2).

◇

Satz 3.3 *Seien v_i, v_j Blätter mit $w_i, w_j > \sum_{k=i+1}^{j-1} w_k$ und T ein optimaler alphabetischer Baum auf den Blättern $v_1, \ldots, v_n$.*
Dann enthält T einen Teilbaum mit genau den Blättern $v_{i+1}, \ldots, v_{j-1}$.

Beweis 3.3 Sei c so gewählt, daß $v_i, \ldots, v_c$ und $v_{c+1}, \ldots, v_j$ Teilmengen verschiedener Teilbäume sind (siehe Abbildung 3.3.(a)).
Gezeigt wird zunächst: Es gibt Teilbäume, deren Blätter genau $v_{i+1}, \ldots, v_c$ beziehungsweise $v_{c+1}, \ldots, v_{j-1}$ sind.
ρ sei so gewählt, daß $\pi 0 \rho$ der größte gemeinsame Präfix von v_i und v_c ist (siehe Abbildung 3.3.(b)). Zeigt man $v_{i+1} \in A_2$, dann ist die Behauptung bewiesen. Angenommen $v_{i+1} \in A_1$. Dann sei σ so gewählt, daß $\pi 0 \rho 0 \sigma$ der größte gemeinsame Präfix von v_i und v_{i+1} ist (siehe Abbildung 3.3.(c)). Ersetzt man in T die Baumstruktur (c) durch (d), so erhält man einen Baum T_1 mit

$$K_T - K_{T_1} = \sum_{x \in A_{11}} w_x - \sum_{x \in A_2} w_x \geq w_i - \sum_{k=i+1}^{c} w_k \overset{Vor.}{>} 0.$$

Somit gilt $K_T > K_{T_1}$ und dies ist ein Widerspruch zur Optimalität von T. Also ist $v_{i+1} \in A_2$ und somit enthält A_2 genau die Blätter $v_{i+1}, \ldots, v_c$. Analog existiert ein Teilbaum B_1 mit genau den Blättern $v_{c+1}, \ldots, v_{j-1}$.
Gezeigt wird nun, daß die Blätter $v_{i+1}, \ldots, v_{j-1}$ in genau einem Teilbaum liegen. Angenommen sie tun dies nicht, dann enthält T die Struktur (a) aus Abbildung 3.4.

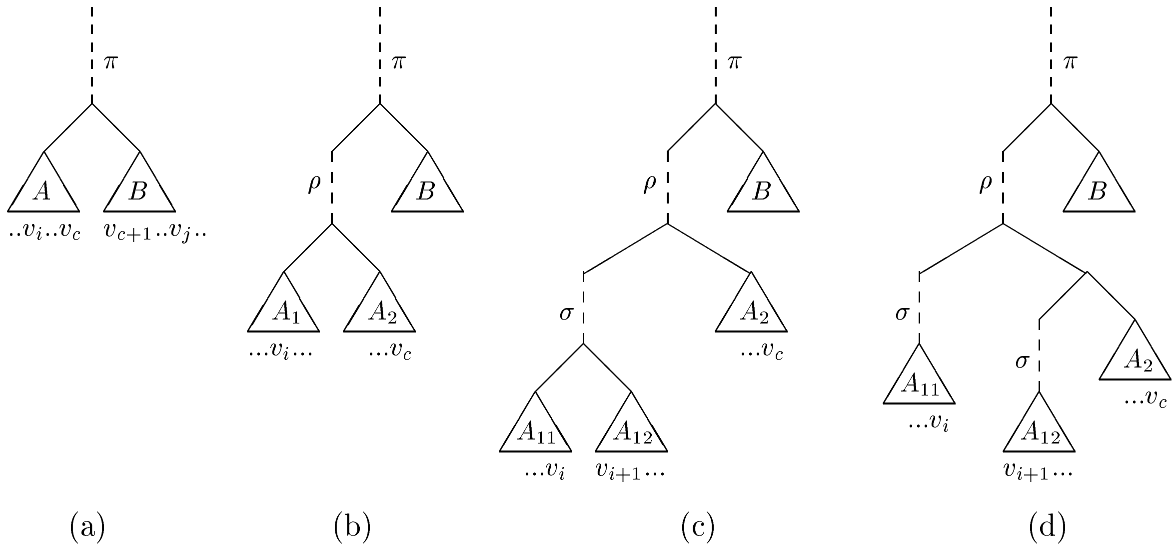


Abbildung 3.3: auszutauschende Strukturen zum Beweis 3.3

- Ist $|\rho| \leq |\tau|$, dann ersetze in T Struktur (a) durch (b) von Abbildung 3.4. Man erhält einen Baum T_2 mit

  $$K_T - K_{T_2} = -\sum_{x \in A_2} w_x + (|\tau| - |\rho| - 1) \sum_{x \in B_1} w_x + \sum_{x \in B_2} w_x \geq -\sum_{k=i+1}^{j-1} w_k + w_j \overset{Vor.}{>} 0.$$

  Somit gilt $K_T > K_{T_2}$, dieses ist ein Widerspruch zur Optimalität von T.

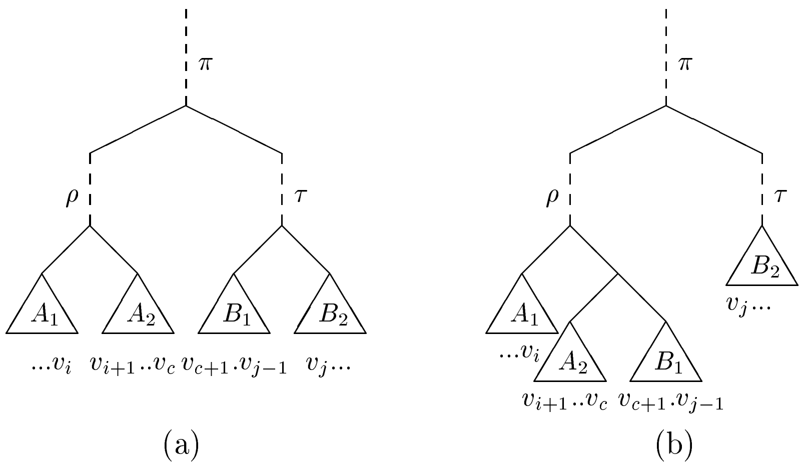


Abbildung 3.4: auszutauschende Strukturen zum Beweis 3.3

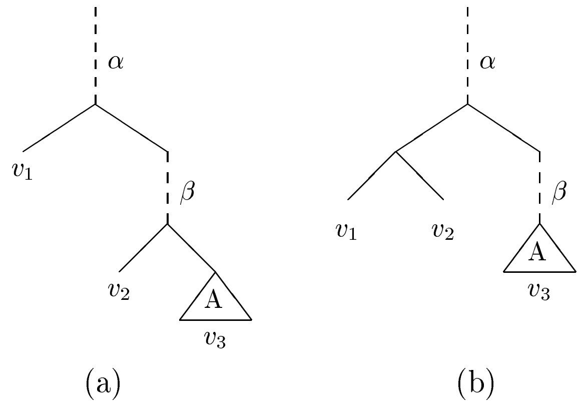


Abbildung 3.5: auszutauschende Strukturen zum Beweis 3.4

- Für $|\rho| \geq |\tau|$ läuft der Beweis analog.$\diamond$

Satz 3.4 *Sei T ein optimaler alphabetischer Baum auf den Blättern $v_1, \ldots, v_n$.*

1. *Ist $w_1 < w_3$, dann existiert ein Teilbaum, der genau die Blätter v_1 und v_2 enthält.*

2. *Ist $w_n > w_{n-2}$, dann existiert ein Teilbaum, der genau die Blätter v_n und v_{n-1} enthält.*

Beweis 3.4

1. Ist $w_1 < w_3$ und gibt es keinen Teilbaum mit den Blättern v_1 und v_2, dann enthält der Baum die Struktur (a) aus Abbildung 3.5. Ersetzt man diese durch die Struktur (b), erhält man einem Baum $\bar{T}$, für den gilt
$$K_T - K_{\bar{T}} = -w_1 + \beta w_2 + \sum_{x \in A} w_x \geq -w_1 + w_3 > 0$$
und somit $K_{\bar{T}} < K_T$. Das ist ein Widerspruch zur Optimalität von T.

2. Analog zu (1).

$\diamond$

3.3 Das Verfahren von Gilbert und Moore

Gilbert und Moore lösen das Problem mittels dynamischer Programmierung, indem sie zuerst alphabetische Bäume für kleinere Blattmengen suchen und diese solange vergrößern, bis die gesamte Blattmenge erfaßt ist. Die Grundlage dafür bildet folgender Satz.

Satz 3.5 *Jeder Teilbaum eines optimalen alphabetischen Baumes T ist ein optimaler alphabetischer Baum.*

Beweis 3.5 Gibt es einen Teilbaum, der nicht optimal alphabetisch ist, dann kann man ihn durch einen optimalen alphabetischen Baum auf der gleichen Blattmenge ersetzen. So konstruiert man einen alphabetischen Baum mit kleineren Kosten und dies ist ein Widerspruch zur Optimalität von T. $\diamond$

Sei $[i, j]$ die Aufgabe, einen optimalen alphabetischen Baum auf den Knoten $v_i, \ldots, v_j$ zu finden. $K[i, j]$ seien die Kosten dieses Baumes. Die Lösung des ursprünglichen Problems ist gefunden, wenn $[1, n]$ gelöst ist. $[i, j]$ führt man folgendermaßen auf ein „einfacheres“ Problem zurück: Man spaltet $\{v_i, \ldots, v_j\}$ auf in $\{v_i, \ldots, v_N\}$ und $\{v_{N+1}, \ldots, v_j\}$. Auf diesen Mengen müssen optimale alphabetische Bäume konstruiert werden. Die Kosten des Baumes auf $\{v_i, \ldots, v_j\}$ setzen sich also zusammen aus $K[i, N]$ und $K[N + 1, j]$ und der Summe $\sum_{p=i}^{j} w_p$, da alle Blätter im gesuchten Baum eine Ebene tiefer liegen als in den Teilbäumen. Da der Baum auf $\{v_i, \ldots, v_j\}$ optimal sein soll, ist das N zu suchen, welches die Summe der Kosten der Teilbäume minimiert.

Die Rekursionsformel drückt dies folgendermaßen aus.

Satz 3.6 *Sei $i < j$ und $i, j = 1, \ldots, n$.*

- $K[i, i] = 0$
- $K[i, j] = \sum_{p=i}^{j} w_p + \min_{i \leq N \leq j-1}(K[i, N] + K[N + 1, j,])$

Der Wert N, der den Ausdruck minimiert, muß gespeichert werden, um aus den berechneten Werten einen Baum konstruieren zu können. Dieser Wert wird in der Variablen $N[i, j]$ abgelegt mit $N[i, i] := i$. Nach Berechnung aller $K[i, j]$ konstruiert man den Baum über die gespeicherten $N[i, j]$ folgendermaßen:
Der Baum zur Lösung der Problems $[i, j]$ besteht aus einer nicht näher spezifizierten Wurzel, einem linken Teilbaum, der die Lösung des Problems $[i, N[i, j]]$ ist, und einem rechten Teilbaum, der die Lösung des Problems $[N[i, j] + 1, j]$ darstellt.

Bemerkung 3.7 Diese Lösungsmethode ähnelt der Methode von Huffman. Beim Huffman-Algorithmus ist allerdings nicht die Berechnung aller Teilbäume notwendig, sondern es werden nur die Teilbäume berechnet, die tatsächlich benötigt werden.

```
PROCEDURE Gilbert-Moore(v_1, ..., v_n:gewichtete Knoten);
VAR i, p, N:INTEGER;
BEGIN
FOR i := 1 TO n DO
    K[i, i] := 0;
    N[i, i] := i;
END;
FOR p := 1 TO n − 1 DO
    FOR i := 1 TO n − p DO
        Finde N ∈ {i, ..., i + p − 1},
          so daß K[i, N] + K[N + 1, i + p] minimal ist;
        K[i, i + p] = Σ_{j=i}^{i+p} w_j + K[i, N] + K[N + 1, i + p];
        N[i, i + p] = N;
    END;
END;
END.
```

Tabelle 3.1: Algorithmus zur Methode von Gilbert und Moore

3.3.1 Der Algorithmus

Der Algorithmus berechnet die $K[i, j]$ nach folgendem Schema:

- Die $K[i, i]$ werden berechnet für $i = 1, \ldots, n$
- $K[i, i + 1]$ wird berechnet für $i = 1, \ldots, n - 1$
- $K[i, i + 2]$ wird berechnet für $i = 1, \ldots, n - 2$
- $\ldots$
- $K[i, i + n - 2]$ wird berechnet für $i = 1, 2$
- $K[1, n]$ wird berechnet

Der Algorithmus füllt das linke obere Dreieck einer Tabelle mit n Spalten $p = 0, \ldots, n - 1$ und n Zeilen, in denen die Paare $(K[i, i + p], N[i, i + p])$ mit $i = 1, \ldots, n$ abgelegt werden.

Der Algorithmus hat die Komplexität $O(n^3)$: Das Suchen eines geeigneten N zur Minimierung von $K[i, N] + K[N + 1, i + p]$ beschränkt sich auf $p - 1$ viele Werte. Der Abschnitt des Algorithmus, in dem die Tabelle gefüllt wird

p=	0	1	2	3	4	5
$K[1, 1+p]$	0	11	14	20	42	62
$N[1, 1+p]$	1	1	1	1	1	4
$K[2, 2+p]$	0	2	6	19	37	
$N[2, 2+p]$	2	2	3	4	5	
$K[3, 3+p]$	0	3	15	32		
$N[3, 3+p]$	3	3	4	5		
$K[4, 4+p]$	0	11	27			
$N[4, 4+p]$	4	4	5			
$K[5, 5+p]$	0	14				
$N[5, 5+p]$	5	5				
$K[6, 6+p]$	0					
$N[6, 6+p]$	6					

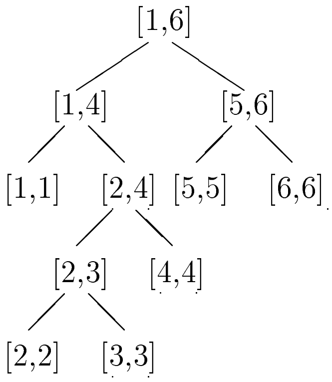


Tabelle 3.2: Tabelle und Baum zum Beispiel 3.8

für alle $p \neq 0$, umfaßt also

$$\sum_{p=1}^{n-1} \sum_{i=1}^{n-p} (p-1) = O(n^3)$$

viele Operationen.

Beispiel 3.8 Seien die Blätter a, b, c, d, e, f gegeben mit den Wahrscheinlichkeiten 10,1,1,2,9,5 wie in Abbildung 3.1. Die Anwendung des Algorithmus liefert Tabelle 3.2. Die Konstruktion des Baumes aus der Tabelle ist daneben angegeben.

3.3.2 Verminderung der Komplexität

Die Komplexität des Algorithmus läßt sich verbessert bestimmen zu $O(n^2)$. Dies ist möglich, da man die Anzahl der zu betrachtenden N zur Minimierung von $K[i, N] + K[N+1, i+p]$ einschränken kann auf $1 + N[i+1, p+i] - N[i, p+i-1]$ viele. Dies wird im folgenden bewiesen.

Satz 3.9 *Sei $i_1 \leq i_2 \leq j_1 \leq j_2$. Dann gilt*

$$K[i_1, j_2] + K[i_2, j_1] \geq K[i_1, j_1] + K[i_2, j_2].$$

Beweis 3.9 Der Beweis erfolgt mit vollständiger Induktion über die Differenz $j_2 - i_1$.

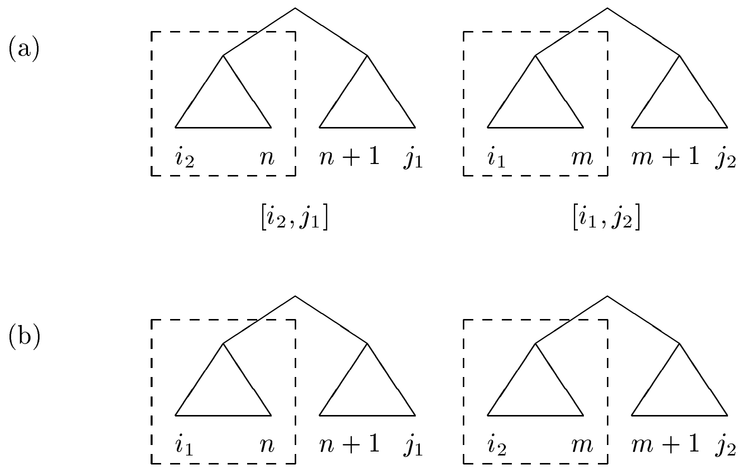


Abbildung 3.6: Bäume zum Beweis 3.9.1

Ist die Differenz Null, dann gilt $i_1 = i_2 = j_1 = j_2$, und die obige Ungleichung ist erfüllt.
Sei die Ungleichung für alle Differenzen kleiner als $j_2 - i_1$ schon bewiesen. Folgende Abkürzungen werden eingeführt: $n := N[i_2, j_1]$ und $m := N[i_1, j_2]$. Es müssen zwei Fälle unterschieden werden.

1. Es ist $i_2 < j_1$.

   - Ist $n \leq m$, dann gilt wegen $i_1 \leq i_2 \leq n \leq m$, $m - i_1 < j_2 - i_1$ die Induktionsannahme

   $$K[i_1, m] + K[i_2, n] \geq K[i_1, n] + K[i_2, m] \tag{3.1}$$

     $[i_1, j_2]$ und $[i_2, j_1]$ lassen sich wie in Abbildung 3.6.(a) darstellen. Anwendung der Induktionsannahme (3.1) auf die umrandeten Teilbäume liefert eine Verringerung der Gesamtkosten und die alphabetischen Bäume über $i_1, \ldots, j_1$ und $i_2, \ldots, j_2$ aus Abbildung 3.6.(b). Diese haben gleiche oder höhere Kosten als die optimalen Bäume $[i_1, j_1]$ und $[i_2, j_2]$.

   - Für $n > m$ geht man symmetrisch vor.

2. Es ist $i_2 = j_1$. Dann ist $n = i_2 = j_1$.

   - Ist $n \leq m$, dann gilt wegen $i_1 \leq n \leq m$, $m - i_1 < j_2 - i_1$ die Induktionsannahme

   $$K[i_1, m] + K[n, n] \geq K[i_1, n] + K[n, m] \tag{3.2}$$

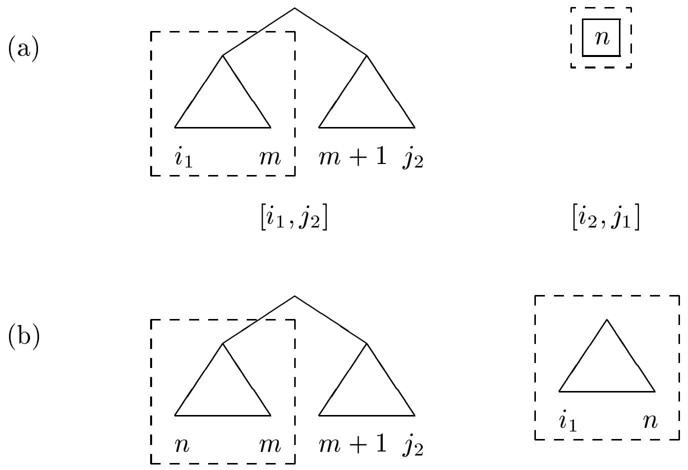


Abbildung 3.7: Bäume zum Beweis 3.9.2

$[i_1, j_2]$ und $[i_2, j_1]$ lassen sich wie in Abbildung 3.7.(a) darstellen. Anwendung der Induktionsannahme (3.2) auf die umrandeten Teilbäume liefert eine Verringerung der Gesamtkosten und die alphabetischen Bäume über $i_1, \ldots, j_1$ und $i_2, \ldots, j_2$ aus Abbildung 3.6.(b). Diese haben gleiche oder höhere Kosten als die optimalen Bäume $[i_1, j_1]$ und $[i_2, j_2]$.

- Für $n > m$ geht man symmetrisch vor.

$\diamond$

Mithilfe des letzten Satzes kann man folgende Aussage beweisen

Satz 3.10 *Ist $i_1 \leq i_2 \leq j_1 \leq j_2$, $i_1 < j_1$ und $i_2 < j_2$. Dann gilt*

$$N[i_1, j_1] \leq N[i_2, j_2].$$

Beweis 3.10 Sei $n := N[i_1, j_1]$ und $m := N[i_2, j_2]$. Angenommen, es ist $n > m$.
Aus Satz 3.9 folgt wegen $i_1 \leq i_2 \leq m < n$ und $m + 1 < n + 1 \leq j_1 \leq j_2$

$$\begin{aligned} K[i_1, n] + K[i_2, m] &\geq K[i_1, m] + K[i_2, n] \\ K[n+1, j_1] + K[m+1, j_2] &\geq K[n+1, j_2] + K[m+1, j_1] \end{aligned}$$

$[i_1, j_1]$ und $[i_2, j_2]$ lassen sich wie in Abbildung 3.8.(a) darstellen. Anwendung obiger Gleichungen auf alle Teilbäume liefert die alphabetischen Bäume über

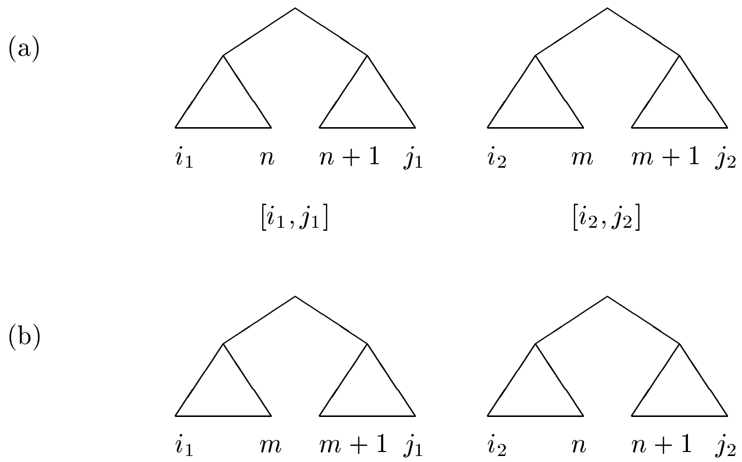


Abbildung 3.8: Bäume zum Beweis 3.10

$i_1, \ldots, j_1$ und $i_2, \ldots, j_2$ aus Abbildung 3.8.(b). Diese haben aufgrund der Abschätzung gleiche oder geringere Kosten wie die optimalen Bäume $[i_1, j_1]$ und $[i_2, j_2]$. Dies ist nicht möglich. $\diamond$

Für jedes $N[i,j]$ mit $j - i > 1$ gilt also sowohl $N[i, j-1] \leq N[i,j]$ wie auch $N[i,j] \leq N[i+1,j]$. Beim Suchen eines geeigneten N zur Minimierung von $K[i,N] + K[N+1, i+p]$ mit $p \geq 2$ brauchen also nur $1 + N[i+1, i+p] - N[i, i+p-1]$ viele Werte betrachtet werden, nämlich die Werte zwischen $N[i, i+p-1]$ und $N[i+1, i+p]$. Der Abschnitt des Algorithmus, in dem die Tabelle gefüllt wird für alle $p \geq 2$, umfaßt also

$$\begin{aligned}\sum_{p=2}^{n-1} \sum_{i=1}^{n-p} (1 + N[i+1, i+p] - N[i, i+p-1]) &= \\ \sum_{p=2}^{n-1} ((n-p) + N[n-p+1, n] - N[1,p]) &\leq \\ \sum_{p=2}^{n-1} (2n - p - 2) &= O(n^2)\end{aligned}$$

viele Operationen.

Um den Algorithmus mit dieser Komplexität realisieren zu können, muß man die Summen $\sum_{j=1}^{p} w_j$ geschickt berechnen, indem man sich auf vorher berechnete Summen bezieht.

3.4 Das Verfahren von Hu und Tucker

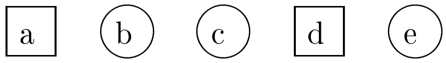
Hu und Tucker konstruieren einen Baum, der die Kosten eines optimalen alphabetischen Baumes besitzt, aber nicht alphabetisch ist. Dieser Baum wird in einen alphabetischen Baum gleicher Kosten transformiert.

3.4.1 T.C.Bäume und T.C.L.Bäume

Bisher wurden die Elemente einer Konstruktions-Sequenz nur als Knoten betrachtet. Man kann sie jedoch in gewohnter Weise mit Blättern und inneren Knoten identifizieren, je nachdem, ob sie bereits Kinder haben und so erst im Laufe der Veränderung der Sequenz entstanden sind, oder ob sie noch kinderlos sind und der ursprünglichen Sequenz entstammen.

Definition 3.11 *Die Knoten u und v einer Konstruktions-Sequenz heißen* tentative-connecting *oder* T.C.*, wenn sich in der Konstruktions-Sequenz keine Blätter zwischen ihnen befinden. Man schreibt dann $u \sim v$.*
Eine Huffman-Menge *ist eine maximal große Menge aufeinanderfolgender Knoten einer Konstruktions-Sequenz, so daß jedes Paar von Knoten in dieser Menge T.C. ist.*

Beispiel 3.12 In der unten stehenden Abbildung gilt $a \sim b$,$a \sim c$,$a \sim d$,$b \sim c$,$b \sim d$,$c \sim d$,$d \sim e$. Die beiden Huffman-Mengen sind $\{a, b, c, d\}$ und $\{d, e\}$.



Bemerkung 3.13

1. Eine Huffman-Menge enthält maximal 2 Blätter.

2. Sind in einer Konstruktions-Sequenz m Huffman-Mengen mit jeweils k_i Knoten vorhanden, dann gilt

$$\sum_{i=1}^{m} k_i \leq 2n - 2.$$

Einen *T.C.Baum* zu einer gegebenen Konstruktions-Sequenz erhält man, wenn man solange Knoten, die T.C. sind, durch einen neuen Knoten ersetzt, der die beiden Knoten als Kinder besitzt, bis nur noch ein Knoten übrig ist. Der neue Knoten nimmt dabei stets die Position seines linken Kindes ein. Jeder alphabetische Baum ist ein T.C.Baum, da Geschwister stets T.C. sind.

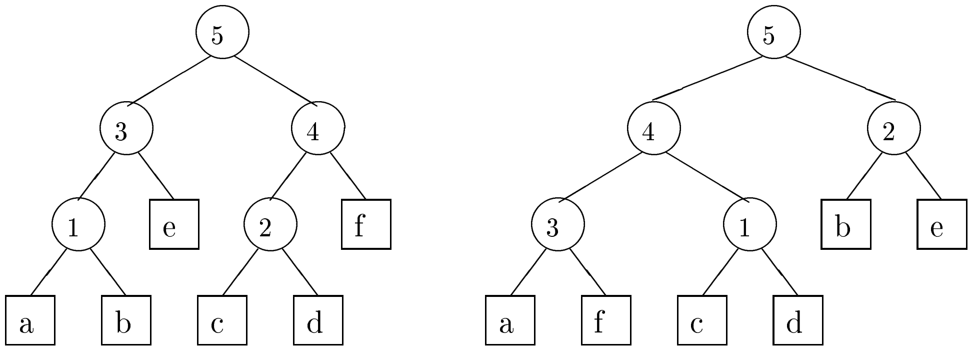


Abbildung 3.9: T.C.L.Baum und T.C.Baum zum Beispiel 3.14

Zu einem gegebenen T.C.Baum existiert eine *T.C. level-by-level Konstruktion*, wenn man (beginnend bei der untersten Ebene) erst dann Knoten einer höheren Ebene verbinden muß, wenn alle Knoten der aktuellen Ebene aus der Konstruktions-Sequenz entfernt sind. Ein solcher Baum heißt dann *T.C. level-by-level Baum* oder *T.C.L.Baum*. Jeder alphabetische Baum ist ein T.C.L.Baum, da stets eine triviale level-by-level Konstruktion möglich ist.

Beispiel 3.14 Sei die aus Blättern bestehende Konstruktions-Sequenz a, b, c, d, e, f gegeben. Dann sind beide Bäume in Abbildung 3.9 T.C.Bäume. Die Reihenfolge, in der die inneren Knoten erzeugt werden, wird durch ihre Numerierung angegeben.

Nur der linke Baum ist ein T.C.L.Baum, da im rechten Baum die Blätter b und e zu einem Zeitpunkt verknüpft werden müssen, zu dem a und f noch nicht verknüpft sind, obwohl sie sich eine Ebene tiefer befinden.

Nicht jeder T.C.L.Baum ist alphabetisch. Aber zu jedem dieser Bäume existiert ein *normalisierter Baum* T_N, der alphabetisch ist und die gleichen Kosten besitzt. Diesen konstruiert man folgendermaßen: Ebenenweise (beginnend in der letzten Ebene) werden die $2k$ Kinder in der i-ten Ebene (die als Väter k innere Knoten der Ebene $i-1$ besitzen) umsortiert. Dabei wird die gewünschte Ordnung des alphabetischen Baumes als Grundlage für die Umordnung benutzt. Das bedeutet, daß ein Knoten u genau dann links von v einsortiert wird, falls er auch im gesuchten alphabetischen Baum links von v auftauchen soll. Das dies möglich ist, wird in Beweis 3.15 gezeigt. Da die Blätter beim Umsortieren nicht ihre Entfernung von der Wurzel verändern, ändern sich auch die Kosten des Baumes nicht.

Satz 3.15 *Jeder nicht-alphabetische T.C.L.Baum besitzt einen zugehörigen alphabetischen Baum T_N gleicher Kosten.*

Beweis 3.15 T_N ist ein T.C.Baum, da die Form des ursprünglichen Baumes erhalten bleibt. Denn existieren zwischen Knoten einer Ebene Blätter, dann befinden diese sich auf einer höheren Ebene. Desweiteren ist die Anzahl der zwischen diesen Knoten liegenden Knoten gerade, da T_N die gleiche Struktur aufweist wie der T.C.L.Baum.

Gezeigt wird mit Induktion, daß nach dem Umsortieren einer festen Anzahl von Ebenen die Knoten der nächsten Ebene Wurzeln alphabetischer Bäume sind. Desweiteren sind deren Blätter aufeinanderfolgende Werte der ursprünglichen Sequenz.
Zu Beginn befindet man sich in der tiefsten Ebene, eine Umsortierung hat noch nicht stattgefunden. Alle dort befindlichen Knoten sind Blätter und somit triviale alphabetische Teilbäume.
Seien alle Ebenen ab $i+1$ umsortiert und somit laut Induktionsannahme die Knoten der i -ten Ebene Wurzeln von alphabetischen Bäumen aufeinanderfolgender Werte. Zu zeigen ist, daß nach Umsortieren der i-ten Ebene die Knoten der $i-1$-ten Ebene ebenfalls Wurzeln von alphabetischen Bäumen aufeinanderfolgender Werte sind. Aufgrund der Vorschrift des Umsortierens ist offensichtlich, daß sie Wurzeln von alphabetischen Bäumen sind. Bleibt zu zeigen, daß die Werte konsekutiv aufeinanderfolgen:
Gäbe es nach dem Umsortieren einen Knoten in Ebene $i-1$ mit den Blättern $v_s, \ldots, v_{k-1}, v_{k+1}, \ldots, v_j$, dann müßte v_k sich in Ebene $i-1$ oder höher befinden. Desweiteren gäbe es in Ebene $i+1$ Knoten, die Wurzeln von alphabetischen Teilbäumen mit Blättern $v_i, \ldots, v_{k-1}$ bzw.$v_{k+1}, \ldots, v_j$ wären. Dann befindet sich das Blatt v_k in der Konstruktions-Sequenz zwischen diesen beiden Teilbäumen und diese könnten nicht Kinder des selben Knotens auf Ebene i sein, da T_N ein T.C.Baum ist. $\diamond$

Da jeder alphabetischer Baum ein T.C.L.Baum ist, genügt es, in der Klasse der T.C.L.Bäume einen optimalen Baum zu suchen, um einen Baum mit optimalen alphabetischen Kosten zu finden. Zu diesem wird der normalisierte Baum konstruiert, der dann der gesuchte optimale alphabetische Baum ist.

3.4.2 Konstruktion eines optimalen T.C.L.Baumes

Den gesuchten Baum mit optimalen alphabetischen Kosten erhält man folgendermaßen: Es werden die zwei T.C. Knoten zu einem neuen Knoten kombiniert, die minimales gemeinsames Gewicht unter allen T.C. Knoten besitzen. Falls mehrere solcher Paare existieren, wählt man das Paar, dessen linker Knoten sich am weitesten links befindet. Ist trotzdem kein eindeutiges Paar festgelegt, wählt man das, dessen rechter Knoten sich am weitesten links befindet.

Es bleibt zu zeigen, daß der konstruierte Baum ein T.C.L.Baum ist und unter allen T.C.L.Bäumen minimales Gewicht hat. Diese Beweise sind so komplex, daß sie hier nicht besprochen werden können, und es wird an dieser Stelle auf Hu und Tucker [11], Hu [9] und Larmore, Karpinski und Rytter [20] verwiesen.

Die Komplexität des T.C.Algorithmus beträgt $O(n^2)$: Sei eine Konstruktions-Sequenz gegeben. Das Suchen des minimalen Paares in einer Huffman-Menge mit k_i Knoten kostet $O(k_i)$. Da man alle m Huffman-Mengen jeder Sequenz durchlaufen muß, bekommt man die Komplexität $O(\sum_{i=1}^{m} k_i) = O(n)$ (siehe Bemerkung 3.13.2) zum Auffinden des absoluten Minimums. Da zur Konstruktion eines Baumes aus n Blättern das Durchlaufen von $n-1$ Konstruktions-Sequenzen nötig ist, bekommt man eine Gesamtkomplexität von $(n-1)O(n) = O(n^2)$.

3.4.3 Schnellere Konstruktion eines optimalen T.C.L.Baumes

Benutzt man eine geeignete Datenstruktur zum Speichern der Huffman-Mengen, erreicht man eine Komplexität von $O(n \log_2 n)$. Diese Datenstruktur soll folgende Bedingungen erfüllen:

- das Finden des Minimums ist in $O(1)$ realisierbar,
- das Löschen des Minimums ist in $O(\log_2 n)$ realisierbar,
- das Einfügen eines neuen Elementes ist in $O(\log_2 n)$ realisierbar,
- das Vereinigen von Mengen ist in $O(\log_2 n)$ realisierbar.

Eine geeignete Datenstruktur wäre somit die Speicherung der Huffman-Mengen in Binominalhalden $hm_1, \ldots, hm_{n-1}$. Das Gewicht des minimalen Paares jeder Huffman-Menge wird ebenso in einer Binominalhalde min abgelegt.

Betrachtet man die Huffman-Mengen, dann beobachtet man: Änderungen treten nur an der Huffman-Menge mit dem minimalen Paar auf sowie an deren benachbarten Huffman-Mengen. Sei M die Huffman-Menge mit dem kleinsten Paar, L und R die linke und rechte benachbarte Menge. Dann gilt insbesondere:

- Besteht das kleinste Paar aus zwei inneren Knoten, verändert sich nur die Menge M.
- Besteht das kleinste Paar aus einem inneren Knoten und einem Blatt, dann verschmelzen und ändern sich die Huffman-Mengen L und M bzw. R und M, je nach dem, ob sich das Blatt links oder rechts vom inneren Knoten befindet.

```
PROCEDURE T.C.L.Baum(v_1, ..., v_n:gewichtete Knoten):Baum;
VAR hm_1, ..., hm_{n-1}:Liste von Bäumen;
      min:ARRAY [1, ..., n-1] OF INTEGER;
      i, I:INTEGER;
      p:Knoten;
BEGIN
FOR i := 1 TO n-1 DO
   hm_i := ([v_i], [v_{i+1}]);
   min[i] := w(v_i) + w(v_{i+1});
END;
WHILE |⋃_{i=1}^{n-1} hm_i| > 1 DO
    I := {i|min[i] ist minimal};
    finde [u], [v] ∈ hm_I mit kleinsten Gewichten,
       bei Mehrfachbelegung wähle jeweils die linkeste Wurzel;
    [p] := (p, [u], [v]);
    w(p) := w(u) + w(v);
    IF u ist Blatt THEN
       hm_I := (hm_{I-1}, hm_I);
       hm_{I-1} := ∅; min[I-1] := ∞;
    END;
    IF v ist Blatt THEN
       hm_I := (hm_I, hm_{I+1});
       hm_{I+1} := ∅; min[I+1] := ∞;
    END;
    hm_I := TAUSCHE^{[p]}_{[u],[v]}(hm_I);
    finde [u], [v] ∈ hm_I mit kleinsten Gewichten,
       bei Mehrfachbelegung wähle jeweils die linkeste Wurzel;
    min[I] := w(u) + w(v);
END;
RETURN ⋃_{i=1}^{n-1} hm_i;
END.
```

$\mathrm{TAUSCHE}^{y_3}_{y_1,y_2}(x_1, \ldots, x_k)$ ersetzt in dem Eingabe-Vektor die erste Komponente x_i mit $x_i = y_1$ durch $x_i := y_3$ und löscht danach alle Komponenten x_i mit $x_i = y_1$ oder $x_i = y_2$.

u ist genau dann ein Blatt, wenn u einer der Eingabeknoten v_i ist.

Tabelle 3.3: T.C.L.Algorithmus

PROCEDURE Hu-Tucker($v_1, \ldots, v_n$:gewichtete Knoten):Baum;
BEGIN
T:=T.C.L.Baum($v_1, \ldots, v_n$);
Konstruiere Blattlängenliste $F_T = (f_1, \ldots, f_n)$;
Konstruiere Baum T_N mit mit $l(v_i) := f_i$;
RETURN T_N;
END.

Tabelle 3.4: Algorithmus zur Methode von Hu und Tucker

- Besteht das kleinste Paar aus zwei Blättern, verschmelzen und ändern sich alle drei Mengen M, L und R.

Deshalb muß nur das minimale Paar **einer** Huffman-Menge pro Verschmelzung neu bestimmt werden.

Der Algorithmus läßt sich dann folgendermaßen programmieren:

- Initialisieren der hm_i und von min (in $O(n)$).
- Solange mehr als ein Knoten in der Konstruktions-Sequenz vorhanden ist
  - finde das Minimum in min (in $O(1)$).
  - verschmelze und ändere die geeigneten Mengen L, M, R (in $O(\log_2 n)$).
  - lösche die zugehörigen Werte in min (in $O(\log_2 n)$).
  - finde das Paar mit neuem kleinstem Gewicht in der neu entstandenden Huffman-Menge (in $O(\log_2 n)$).
  - füge das Gewicht dieses Paares in min ein (in $O(\log_2 n)$).

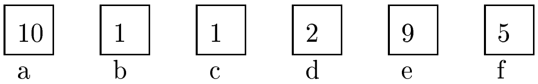
Da nach $n-1$ Durchläufen nur noch ein Knoten in der Konstruktions-Sequenz vorliegt, hat der Algorithmus eine Gesamtkomplexität von $O(n \log_2 n)$.

3.4.4 Der Algorithmus

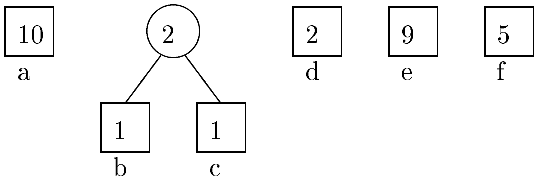
Da die Konstruktion des T.C.L.Baumes eine Komplexität von $O(n \log_2 n)$ besitzt und die Konstruktion des normalisierten Baumes in $O(n)$ möglich ist, besitzt der gesamte Algorithmus die Komplexität $O(n \log_2 n)$.

3.4.5 Beispiel

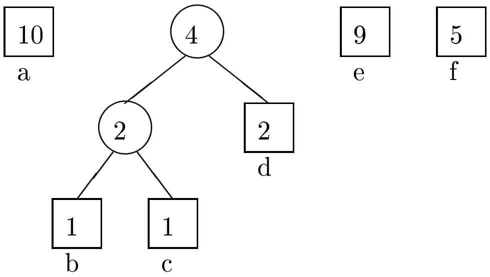
Seien die Werte wie bisher 10,1,1,2,9,5. Dann kann man den Ablauf des Algorithmus folgendermaßen veranschaulichen – der Einfachheit halber werden in den hm_i die Gewichte der Wurzeln aufgelistet und nicht die zugehörigen Bäume:



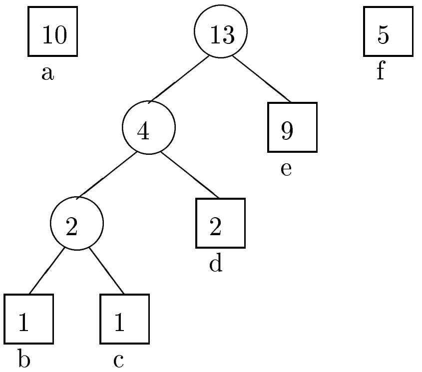
$hm_1 := (10, 1), hm_2 := (1, 1), hm_3 := (1, 2),$
$hm_4 := (2, 9), hm_5 := (9, 5)$
$min = (11, 2, 3, 11, 14)$



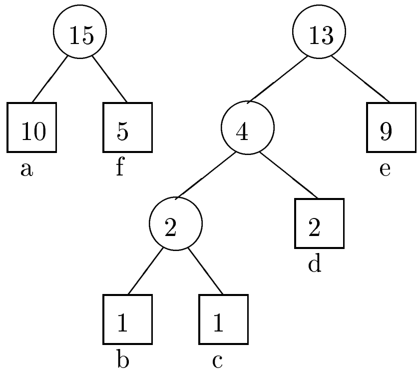
$hm_2 := (10, 2, 2), hm_4 := (2, 9), hm_5 := (9, 5),$
$min = (4, 11, 14)$



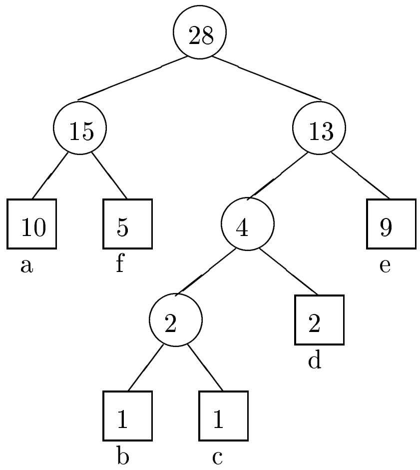
$hm_2 := (10, 4, 9), hm_5 := (9, 5), min = (13, 14)$



$hm_2 := (10, 13, 5), min = (15)$



$hm_2 := (15, 13), min = (28)$



$hm_2 := (28)$

3.5 Das Verfahren von Garsia und Wachs

Garsia und Wachs gehen ähnlich vor wie Hu und Tucker: Sie konstruieren einen nicht-alphabetischen Baum, der sich in einen optimalen alphabetischen Baum transformieren läßt. Die Transformation geht dabei genauso vonstatten wie bei Hu und Tucker, hat also ebenfalls eine Komplexität von $O(n)$. Die Konstruktion des nicht-alphabetischen Baumes ist ähnlich einfach und hat die Komplexität $O(n \log_2 n)$. Der Beweis, daß der konstruierte Baum optimale alphabetische Kosten besitzt und transformierbar ist, ist hier einfacher nachzuvollziehen.

3.5.1 Der Algorithmus

Auch in diesem Algorithmus wird die gegebene Konstruktions-Sequenz sukzessive geändert durch das Zusammenfassen geeigneter Knoten. Somit benötigt

auch dieser Algorithmus $n-1$ Durchläufe. Für das Zusammenfassen benutzt man folgende Regeln:
Sei $v_1, \ldots, v_n$ eine Konstruktions-Sequenz.

- Finde zwei benachbarte Knoten v_{i-1}, v_i mit kleinstem gemeinsamen Gewicht. Ist dieses Paar nicht eindeutig bestimmt, wähle das am weitesten rechts stehende Paar, also das mit größtem Index i. Fasse diese Knoten zu einem neuen Knoten p zusammen mit Gewicht $w(p) := w_{i-1}+w_i$. (Es wird später gezeigt, daß auch andere Paare gewählt werden können – sogenannte rechts minimale. Es ist sogar notwendig, die Paare aus dieser größeren Menge zu wählen, um eine optimale Laufzeit zu erhalten.)

- Suche eine Knoten v_k rechts von diesem Paar mit gleichem oder größerem Gewicht. Existieren mehrere solche Knoten, wähle den, der sich am nächsten zum Paar befindet. Es ist also ein minimaler Wert $k > i$ zu bestimmen, so daß $w_k \geq w_{i-1} + w_i$.
  - Falls solch ein Knoten existiert, ist die neue Konstruktions-Sequenz $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, p, v_k, \ldots, v_n$.
  - Falls kein solcher Knoten existiert, ist die neue Konstruktions-Sequenz $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_n, p$.

3.5.2 Korrektheit des Algorithmus

Der Beweis orientiert sich weitestgehend an Garsia und Wachs [7]. Verwiesen wird auch auf Kingston [14] und Larmore, Karpinski und Rytter [20].

Definition 3.16 *$\mathcal{K}(v_1, \ldots, v_n)$ gebe die Kosten eines optimalen alphabetischen Baumes über der Konstruktions-Sequenz $v_1, \ldots, v_n$ an.*
Ein Baum heißt Umsortierung eines Baumes über einer Konstruktions-Sequenz*, falls er alphabetisch ist über dieser Sequenz und die Blätter in beiden Bäumen die gleichen Blattlängen aufweisen.*

Definition 3.17 *Ein Paar v_{i-1}, v_i ist* rechts minimal *in $v_1, \ldots, v_n$, wenn die beiden folgenden Bedingungen erfüllt sind:*

1. $w_{i-2} + w_{i-1} \geq w_{i-1} + w_i$ *, falls* $i > 2$
2. $w_{i-1} + w_i < w_{j-1} + w_j$ *für alle* $j > i$

Die Korrektheit des Algorithmus ist gezeigt, wenn folgender Satz bewiesen ist.

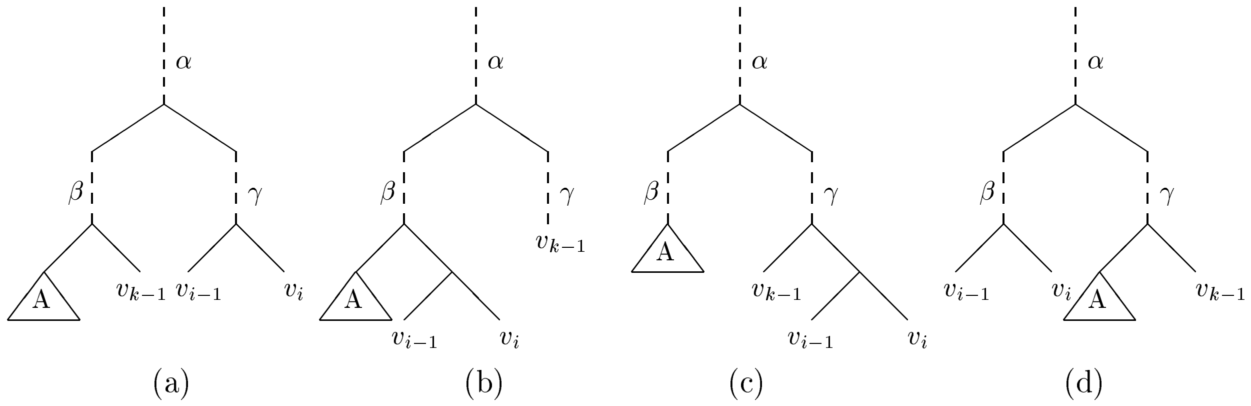


Abbildung 3.10: auszutauschende Strukturen zum Beweis 3.19

Satz 3.18 *Sei v_{i-1}, v_i ein rechts minimales Paar aus der Sequenz $v_1, \ldots, v_n$ und $k > i$ so gewählt, daß $w_j < w_{i-1} + w_i$ für alle $i \leq j < k$.*
Falls $w_k \geq w_{i-1} + w_i$ oder $k = n + 1$, dann ist

$$\mathcal{K}(v_1, \ldots, v_n) = \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n)$$

und zu jedem optimalen alphabetischen Baum über $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n$ existiert eine Umsortierung über $v_1, \ldots, v_n$.

Zum Beweis diese Satzes werden einige Hilfsaussagen benötigt

Satz 3.19 *Sei v_{i-1}, v_i ein rechts minimales Paar aus der Sequenz $v_1, \ldots, v_n$ und $k > i$ so gewählt, daß $w_j < w_{i-1} + w_i$ für alle $i \leq j < k$.*
Dann gilt

$$\mathcal{K}(v_1, \ldots, v_n) \leq \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n).$$

Gilt Gleichheit, dann existiert zu jedem optimalen alphabetischen Baum über $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n$ eine Umsortierung über $v_1, \ldots, v_n$.

Beweis 3.19 Der Beweis erfolgt mit Induktion über k.
Ist $k = i + 1$, dann gilt

$$\mathcal{K}(v_1, \ldots, v_n) \leq \mathcal{K}(v_1, \ldots, v_{i-2}, (v_{i-1}, v_i), v_{i+1}, \ldots, v_n).$$

Bei Gleichheit ist jeder optimale alphabetische Baum über $v_1, \ldots, v_{i-2}, (v_{i-1}, v_i), v_{i+1}, \ldots, v_n$ auch optimal alphabetisch über $v_1, \ldots, v_n$, da die Reihenfolgen der Blätter identisch sind.
Sei der Satz für $k - 1$ schon bewiesen und T ein optimaler alphabetischer Baum über der Sequenz $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n$. Dann enthält T die Baumstruktur (a) aus Abbildung 3.10.

1. Ist $|\beta| < |\gamma|$, dann betrachte den Baum T_1, der entsteht, wenn Struktur (a) durch Struktur (b) ersetzt wird. Dann gilt

$$K_T - K_{T_1} = (|\gamma| - |\beta| - 1)(w_{i-1} + w_i - w_{k-1}) \overset{Vorraus.}{\geq} 0.$$

Da T_1 ein alphabetischer Baum über $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-2}, (v_{i-1}, v_i)$, $v_{k-1}, \ldots, v_n$ ist und auf $\mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-2}, (v_{i-1}, v_i), v_{k-1}, \ldots, v_n)$ die Induktionsvorrausetzung anwendbar ist, gilt

$$\begin{aligned} & \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n) = K_T \geq K_{T_1} \\ \geq \; & \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-2}, (v_{i-1}, v_i), v_{k-1}, \ldots, v_n) \geq \mathcal{K}(v_1, \ldots, v_n). \end{aligned}$$

Gilt Gleichheit in der Abschätzung, dann muß $K_T = K_{T_1}$ sein und somit ist $|\gamma| = |\beta| + 1$. Dann ist T_1 eine Umsortierung von T. Da T_1 wegen der Induktionsannahme eine Umsortierung über $v_1, \ldots, v_n$ besitzt, besitzt somit auch T eine Umsortierung über $v_1, \ldots, v_n$.

2. Ist $|\beta| \geq |\gamma|$, dann betrachte den Baum T_2, der entsteht, wenn Struktur (a) durch Struktur (c) ersetzt wird. T_2 ist alphabetisch über der gleichen Konstruktions-Sequenz wie T. Da T optimal ist, gilt $K_T \leq K_{T_2}$ und somit

$$K_T - K_{T_2} = \sum_{x \in A} w_x + (|\beta| - |\gamma|) w_{k-1} - (w_{i-1} + w_i) \leq 0.$$

Daraus und wegen der Rechts-Minimalität von v_{i-1}, v_i folgt

(a) für $k > i + 2$

$$w_{k-2} + w_{k-1} \overset{r.m.}{>} w_{i-1} + w_i \geq \sum_{x \in A} w_x + (|\beta| - |\gamma|) w_{k-1} \geq w_{k-2} + (|\beta| - |\gamma|) w_{k-1}$$

(b) für $k = i + 2$

$$w_{i-2} + w_{i+1} \overset{r.m.}{>} w_{i-1} + w_i \geq \sum_{x \in A} w_x + (|\beta| - |\gamma|) w_{k-1} \geq w_{i-2} + (|\beta| - |\gamma|) w_{k-1}$$

(a) bzw. (b) ist nur richtig, falls $|\beta| = |\gamma|$ ist.

- Für $k > i + 2$ folgt aus (a), daß A nur den Knoten v_{k-2} enthält. Ersetzt man Struktur (a) durch (d) mit $A = v_{k-2}$, dann erhält man einen alphabetischen Baum über $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-3}$, $(v_{i-1}, v_i), v_{k-2}, \ldots, v_n$ und man kann analog zu (1) die Induktionsvorrausetzungen anwenden.

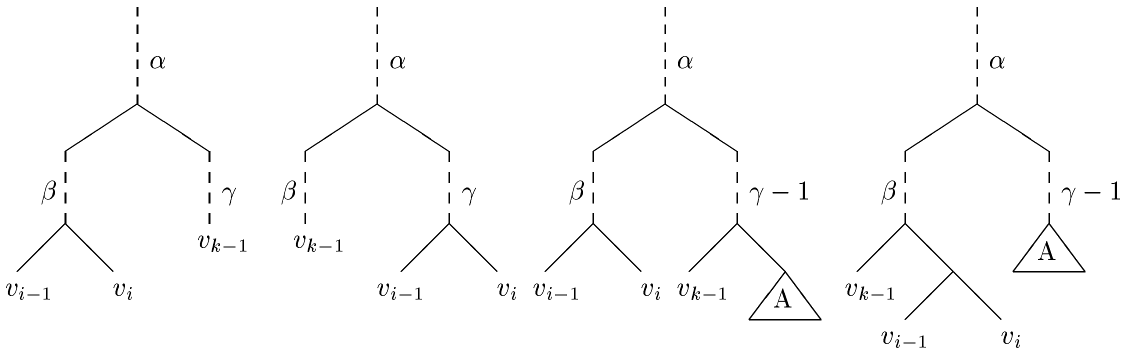


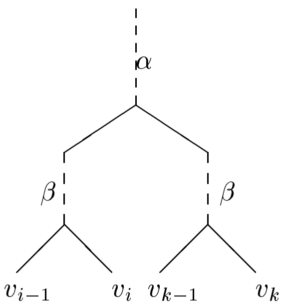
Abbildung 3.11: auszutauschende Strukturen zum Beweis 3.20

- Für $k = i + 2$ liefert bereits das Sortieren der Knoten v_{i-1}, v_i, v_{i+1} in Struktur (a) eine Umsortierung über $v_1, \ldots, v_n$.

$\diamond$

Satz 3.20 *Sei v_{i-1}, v_i ein rechts minimales Paar aus der Sequenz $v_1, \ldots, v_n$ und es existiere ein Wert $k > i + 1$ mit $w_{k-1} < w_{i-1} + w_i$.*
Ist

$$\begin{array}{rl} & \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-2}, (v_{i-1}, v_i), v_{k-1}, \ldots, v_n) \\ < & \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n), \end{array}$$



dann ist $k \leq n$, $w_k < w_{i-1} + w_i$ und jeder optimale alphabetische Baum über $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-2}, (v_{i-1}, v_i), v_{k-1}, \ldots, v_n$ enthält die nebenstehende Struktur.

Beweis 3.20 Sei T ein optimaler alphabetischer Baum über der Sequenz $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-2}, (v_{i-1}, v_i), v_{k-1}, \ldots, v_n$. Dann enthält T die Baumstruktur (a) aus Abbildung 3.11. Sei T_1 der Baum, der entsteht, wenn Struktur (a) durch Struktur (b) ersetzt wird. Dann ist T_1 ein alphabetischer Baum über $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n$ und es gilt

$$\begin{array}{rl} & K_T = \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-2}, (v_{i-1}, v_i), v_{k-1}, \ldots, v_n) \\ < & \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n \leq K_{T_1}. \end{array}$$

Daraus ergibt sich

$$0 > K_T - K_{T_1} = (|\beta| - |\gamma|)(w_{i-1} + w_i - w_{k-1}).$$

Da $w_{i-1}+w_i-w_{k-1} > 0$ ist, muß $|\beta|-|\gamma| < 0$ sein und somit ist $0 \leq |\beta| < |\gamma|$ und $|\gamma| \geq 1$. Deshalb besitzt v_{k-1} ein rechtes Geschwisterkind und es ist $k-1 < n$.
T besitzt also die Struktur (c) aus Abbildung 3.11. Ersetzt man diese Struktur durch Struktur (d), dann erhält man einen Baum T_2 mit $K_T < K_{T_2}$ und

$$K_T - K_{T_2} = \sum_{x \in A} w_x + (|\gamma| - |\beta| - 1)w_{k-1} - w_{i-1} - w_i < 0.$$

Zusammen mir der Rechts-Minimalität von v_{i-1}, v_i folgt

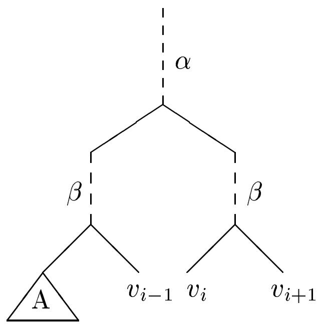
$$w_k+(|\gamma|-|\beta|-1)w_{k-1} \leq \sum_{x \in A} w_x+(|\gamma|-|\beta|-1)w_{k-1} < w_{i-1}+w_i \overset{r.m.}{<} w_{k-1}+w_k. \tag{3.3}$$

$$w_k+(|\gamma|-|\beta|-1)w_{k-1} \leq \sum_{x \in A} w_x+(|\gamma|-|\beta|-1)w_{k-1} < w_{i-1}+w_i \overset{r.m.}{<} w_k+w_{k+1}. \tag{3.4}$$

Aus Gleichung (3.3) folgt $|\gamma| = |\beta| + 1$. Aus Gleichung (3.4) folgt dann, daß A nur den Knoten v_k enthält.
Desweiteren folgt $w_k < w_{i-1} + w_i$ und der Baum enthält die gewünschte Struktur. $\diamond$

Satz 3.21 *Sei $v_1, \ldots, v_n$ eine Konstruktions-Sequenz und es existiere ein i mit*

1. $w_{i-2} + w_{i-1} \geq w_{i-1} + w_i$,
2. $w_{i-1} + w_i \leq w_i + w_{i+1}$ *für* $i + 1 \leq n$,
3. $w_{i-1} + w_i \leq w_{i+1} + w_{i+2}$ *für* $i + 2 \leq n$.



Ist $\mathcal{K}(v_1, \ldots, v_n) < \mathcal{K}(v_1, \ldots, v_{i-2}, (v_{i-1}, v_i), v_{i+1}, \ldots, v_n)$, dann ist $i+1 \leq n$, $w_{i+1} < w_{i-1} + w_i$ und jeder optimale alphabetische Baum über $v_1, \ldots, v_n$ enthält die nebenstehende Struktur.

Beweis 3.21 Sei T ein optimaler alphabetischer Baum über der Sequenz $v_1, \ldots, v_n$. Dann enthält T die Baumstruktur (a) aus Abbildung 3.12.

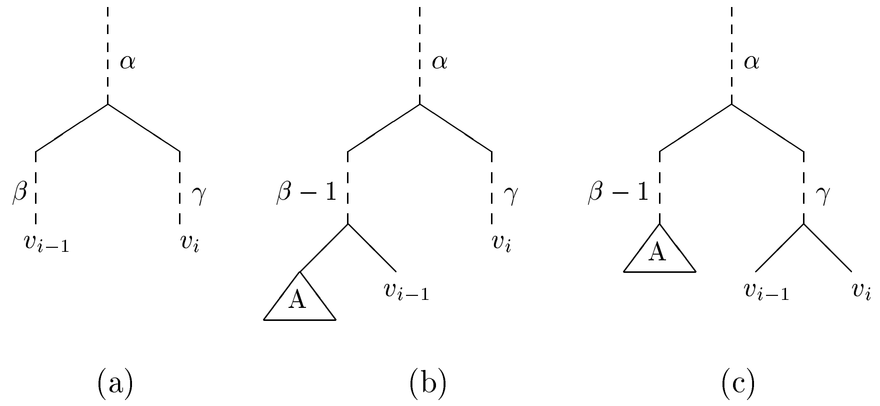


Abbildung 3.12: auszutauschende Strukturen zum Beweis 3.21

1. Ist $|\gamma| = |\beta| = 0$, dann ist T auch ein optimaler alphabetischer Baum über $(v_1, \ldots, v_{i-2}, (v_{i-1}, v_i), v_{i+1}, \ldots, v_n)$. Dann gilt

$$\mathcal{K}(v_1, \ldots, v_n) = \mathcal{K}(v_1, \ldots, v_{i-2}, (v_{i-1}, v_i), v_{i+1}, \ldots, v_n),$$

dies ist widersprüchlich zur Annahme.

2. Ist $|\beta| \geq 1$, dann enthält T die Baumstruktur (b) aus Abbildung 3.12. Ersetzt man dies durch Struktur (c), erhält man einen Baum $\bar{T}$ mit

$$K_T - K_{\bar{T}} = \sum_{x \in A} w_x + (|\beta| - |\gamma| - 1)w_{i-1} - w_i.$$

Da $\bar{T}$ alphabetisch über $v_1, \ldots, v_{i-2}, (v_{i-1}, v_i), v_{i+1}, \ldots, v_n$ ist, gilt $K_T < K_{\bar{T}}$ und somit

$$w_{i-2} + (|\beta| - |\gamma| - 1)w_{i-1} \leq \sum_{x \in A} w_x + (|\beta| - |\gamma| - 1)w_{i-1} < w_i \overset{3.21.1}{\leq} w_{i-2}.$$

Daraus folgt $|\gamma| \geq |\beta| \geq 1$, also $|\gamma| \geq 1$.

3. Ist $|\gamma| \geq 1$, dann hat v_i ein Geschwisterkind, das die Wurzel eines Baumes B ist und ähnlich wie bei (2) unter Benutzung von 3.21.2 folgt $|\beta| \geq |\gamma| \geq 1$, $|\beta| \geq 1$ und

$$\sum_{x \in B} w_x + (|\gamma| - |\beta| - 1)w_i < w_{i-1}.$$

Aus (2) und (3) folgt $|\beta| = |\gamma| \geq 1$ und $\sum_{x \in A} w_x, \sum_{x \in B} w_x < w_{i-1} + w_i$. Wegen 3.21.3 enthält B nur den Knoten v_{i+1} und T enthält die gewünschte Struktur. $\diamond$

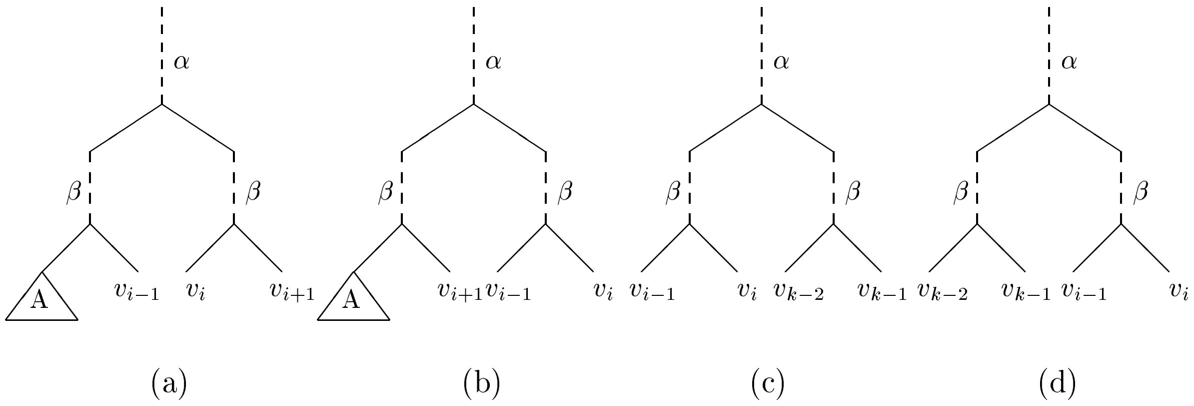


Abbildung 3.13: auszutauschende Strukturen zum Beweis 3.22

Satz 3.22 *Sei v_{i-1}, v_i ein rechts minimales Paar aus der Sequenz $v_1, \ldots, v_n$ und $k > i+1$ so gewählt, daß $w_j < w_{i-1} + w_i$ für alle $i \leq j < k$.*
Dann ist

$$\begin{aligned}\mathcal{K}(v_1, \ldots, v_n) = \min\{ \quad & \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-2}, (v_{i-1}, v_i), v_{k-1}, \ldots, v_n), \\ & \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n) \quad \}.\end{aligned}$$

Beweis 3.22 Der Beweis erfolgt mit Induktion über k.
Ist $k = i+2$, T ein optimaler alphabetischer Baum über $v_1, \ldots, v_n$ und $\mathcal{K}(v_1, \ldots, v_n) < \mathcal{K}(v_1, \ldots, v_{i-2}, (v_{i-1}, v_i), v_{i+1}, \ldots, v_n)$. Dann enthält T aufgrund von Satz 3.21 die Struktur (a) aus Abbildung 3.13. Ist T_1 der Baum, der durch Austausch von Struktur (a) durch Struktur (b) entsteht, dann ist T_1 eine Umsortierung von T über der Sequenz $v_1, \ldots, v_{i-2}, v_{i+1}, (v_{i-1}, v_i), v_{i+2}, \ldots, v_n$.
Dann gilt

$$\mathcal{K}(v_1, \ldots, v_n) = K_T = K_{T_1} \geq \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, (v_{i-1}, v_i), v_{i+2}, \ldots, v_n).$$

Mit Satz 3.19 folgt

$$\mathcal{K}(v_1, \ldots, v_n) = \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, (v_{i-1}, v_i), v_{i+2}, \ldots, v_n).$$

Sei der Satz für $k-1$ schon bewiesen, T ein optimaler alphabetischer Baum über der Sequenz $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-2}, (v_{i-1}, v_i), v_{k-1}, \ldots, v_n$ und $\mathcal{K}(v_1, \ldots, v_n) < \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-2}, (v_{i-1}, v_i), v_{k-1}, \ldots, v_n)$. Dann gilt aufgrund der Induktionsannahme

$$\begin{aligned} & \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-3}, (v_{i-1}, v_i), v_{k-2}, \ldots, v_n) = \mathcal{K}(v_1, \ldots, v_n) \\ < \quad & \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-2}, (v_{i-1}, v_i), v_{k-1}, \ldots, v_n).\end{aligned}$$

Dann enthält T aufgrund von Satz 3.20 die Baumstruktur (c) aus Abbildung 3.13. Ist T_2 der Baum, der durch Austausch von Struktur (c) durch Struktur (d) entsteht, dann ist T_2 eine Umsortierung von T über der Sequenz $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n$. Dann gilt

$$\mathcal{K}(v_1, \ldots, v_n) = K_T = K_{T_2} \geq \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n).$$

Mit Satz 3.19 folgt dann

$$\mathcal{K}(v_1, \ldots, v_n) = \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n).$$

$\diamond$

Mithilfe dieser Sätze ist nun die Korrektheit des Algorithmus beweisbar:

Beweis 3.18:

- Sei $k = i + 1$ gewählt.
  Ist $k = i + 1 > n$ oder $w_{i+1} \geq w_{i-1} + w_i$, dann gilt wegen Satz 3.21
  $$\mathcal{K}(v_1, \ldots, v_n) \geq \mathcal{K}(v_1, \ldots, v_{i-2}, (v_{i-1}, v_i), v_{i+1}, \ldots, v_n).$$
- Sei $k > i + 1$ gewählt.
  Ist $k > n$ oder $w_k \geq w_{i-1} + w_i$, dann gilt wegen Satz 3.20
  $$\begin{aligned} & \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-2}, (v_{i-1}, v_i), v_{k-1}, \ldots, v_n) \\ \geq \; & \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n). \end{aligned}$$

Aus Satz 3.22 folgt dann

$$\mathcal{K}(v_1, \ldots, v_n) = \mathcal{K}(v_1, \ldots, v_{i-2}, (v_{i-1}, v_i), v_{i+1}, \ldots, v_n).$$

bzw.

$$\mathcal{K}(v_1, \ldots, v_n) = \mathcal{K}(v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n).$$

Satz 3.19 liefert zu jedem optimalen alphabetischen Baum über $v_1, \ldots, v_{i-2}, (v_{i-1}, v_i), v_{i+1}, \ldots, v_n$ bzw. über $v_1, \ldots, v_{i-2}, v_{i+1}, \ldots, v_{k-1}, (v_{i-1}, v_i), v_k, \ldots, v_n$ eine Umsortierung über $v_1, \ldots, v_n$. $\diamond$

```
PROCEDURE Garsia_Wachs(v_1, ..., v_n:gewichtete Knoten):Baum;
VAR T, T_N:Baum;
      KS:Liste von Bäumen;
      i:INTEGER;
BEGIN
KS := ([v_1], ..., [v_n]);
FOR i := n TO 3 BY -1 DO
   IF v_{i-2} ≥ v_i THEN KS :=Zusammenfassen(i, KS);END;
END;
T := KS;
Konstruiere Blattlängenliste F_T = (f_1, ..., f_n);
Konstruiere Baum T_N mit mit l(v_i) := f_i;
END.


PROCEDURE Zusammenfassen(j:INTEGER,([u_1], ..., [u_t]):Liste von Bäumen):
                  Liste von Bäumen;
VAR KS̄:Liste von Bäumen;
      k:INTEGER;
      p:Knoten;
BEGIN
[p] := (p, [u_{j-1}], [u_j]);
w(p) := w(u_{j-1}) + w(u_j);
k := min_{s∈[j+1,...,t]} s mit w(u_s) ≥ w(p);
IF k ist nicht bestimmt THEN k := t + 1;END;
KS̄ := ([u_1], ..., [u_{j-2}], [u_{j+1}], ..., [u_{k-1}], [p], [u_k], ..., [u_t]);
IF w(p) ≥ w(u_{k+1}) THEN KS̄ :=Zusammenfassen(k + 1, KS̄);END;
IF w(u_{j-2}) ≥ w(u_{j+2}) THEN KS̄ :=Zusammenfassen(j + 1, KS̄);END;
RETURN KS̄;
END Zusammenfassen.
```

Tabelle 3.5: Algorithmus zur Methode von Garsia und Wachs

3.5.3 Komplexitätsanalyse einer geeigneten Programmierung

Eine geeignete Programmierung sucht in der Konstruktions-Sequenz das am weitesten rechts stehende rechts minimale Paar v_{i-1}, v_i. Dazu wird die Sequenz von rechts nach links durchlaufen. Da dieses Paar das rechteste ist, gilt mit Bedingung 3.17.1 $w_j < w_{j+2}$ für alle $j \geq i-1$. Die Elemente $v_{i-1}, v_{i+1}, v_{i+3}, \ldots$ und $v_i, v_{i+2}, v_{i+4}, \ldots$ sind also nach aufsteigenden Gewichten sortiert. Deshalb kann das Suchen des Wertes w_k mit $w_k \geq w_{i-1} + w_i$ und das Einfügen des neuen Knotens in jedem Durchlauf mittels binärer Suche in $O(\log_2 n)$ durchgeführt werden.

Wurde ein rechts minimales Paar gefunden, zusammengefaßt und an anderer Stelle eingefügt, kann das nächste rechts minimale Paare $v_{\bar{i}-1}, v_{\bar{i}}$ nur an folgenden Stellen vorkommen

- links des gelöschten Paares, also $\bar{i} < i$,
- an der Stelle v_k, v_{k+1}, also $\bar{i} = k+1$,
- an der Stelle v_{i+1}, v_{i+2}, also $\bar{i} = i+2$.

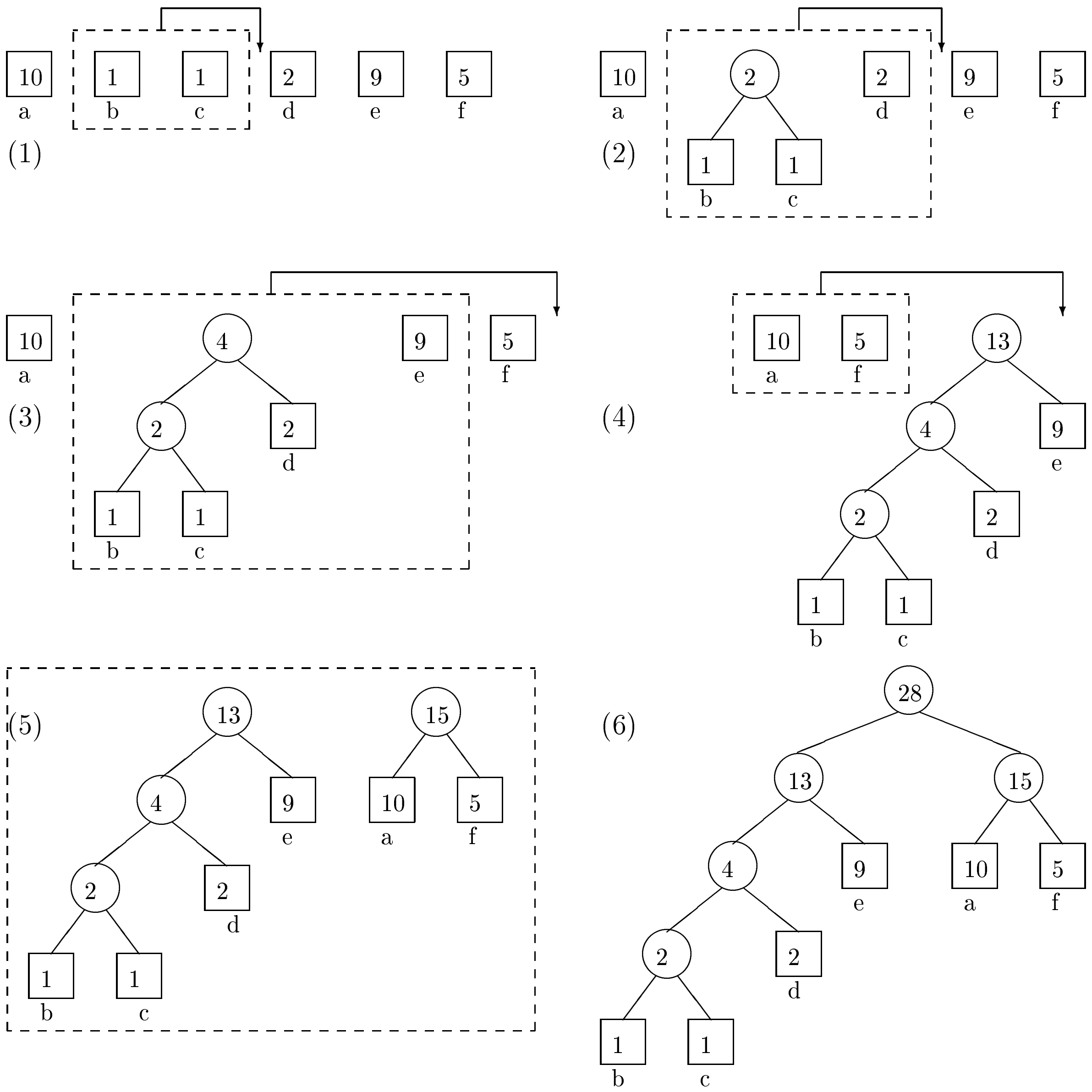
Rechts des aktuellen rechts minimalen Paares existieren also nur zwei Stellen, an denen sich das neue rechts minimale Paar befinden kann. Diese Möglichkeiten werden im Algorithmus rekursiv geprüft und behandelt.

Da in der rückwärts zu bearbeitenden Liste hinter dem Element v_i also nur lokale Änderungen auftreten, die in konstanter Zeit bearbeitet werden können, erhält man im wesentlichen einen einzigen Durchlauf durch die Konstruktions-Sequenz und $n-1$ Operationen, die ein w_k mit $w_k \geq w_{i-1} + w_i$ in $O(\log_2 n)$ bestimmen. Die Gesamtkomplexität berechnet sich deshalb zu

$$O(n) + (n-1)O(\log_2 n) = O(n \log_2 n).$$

3.5.4 Beispiel

Wie in den vorangehenden Abschnitten seien die Werte 10,1,1,2,9,5. Die Manipulationen des Algorithmus werden im folgenden dargestellt.


(1)
(2)
(3)
(4)
(5)
(6)

Kapitel 4

Optimale Bäume und Codes mit eingeschränkter Länge

In diesem Kapitel werden optimale Bäume gesucht, deren Blätter nicht mehr als L Vorgänger besitzen – für jedes Blatt v soll also gelten $l(v) \leq L$. Die gesuchten Bäume besitzen dann maximal L Ebenen. Bäume, die diese Anforderung erfüllen, heißen *L-beschränkt*.

Werden diese Bäume als Codebäume aufgefaßt, erhält man Codes, deren Codewörter maximal L Zeichen enthalten. Diese Beschränkung der Codewortlänge ist einer der wichtigsten Gründe für die Beschäftigung mit L-beschränkten Bäumen, da man beim Ablegen einer Verschlüsselungstabelle auf Einschränkungen hinsichtlich der Spaltenbreite stoßen kann.

Hu und Tan [10] lösen das Problem mit dynamischer Programmierung, die Laufzeit ihres Algorithmus ist exponentiell in L. Dieser Algorithmus ist in der Lage, einen optimalen L-beschränkten alphabetischen Baum zu konstruieren, da er die Reihenfolge der Blätter nicht manipuliert. Larmore [17] zeigt, daß der Algorithmus die Komplexität $O(Ln^2)$ besitzt, falls die Blätter nach Gewichten sortiert eingegeben werden.

Garey [6] stellt einen Algorithmus mit dynamischer Programmierung vor, der das Problem in $O(Ln^3)$ löst und ebenfalls einen optimalen L-beschränkten alphabetischen Baum liefern kann. Für nach Gewicht sortierte Blätter löst er das Problem in $O(Ln^2)$. Dieser Algorithmus wurde verbessert, so daß er auch optimale L-beschränkte alphabetische Bäume mit einer Komplexität von $O(Ln^2)$ konstruiert (Wessner [25]).

Larmore [17] kombiniert die Algorithmen von Hu, Tan und Garey und erhält einen Algorithmus der Komplexität $O(n^{3/2} L \log_2^{1/2} n)$.

Larmore und Hirschberg [18] geben einen coin-collector-Algorithmus an, der das Problem in $O(Ln)$ löst, falls die Knoten bereits nach Gewicht sortiert vorliegen.

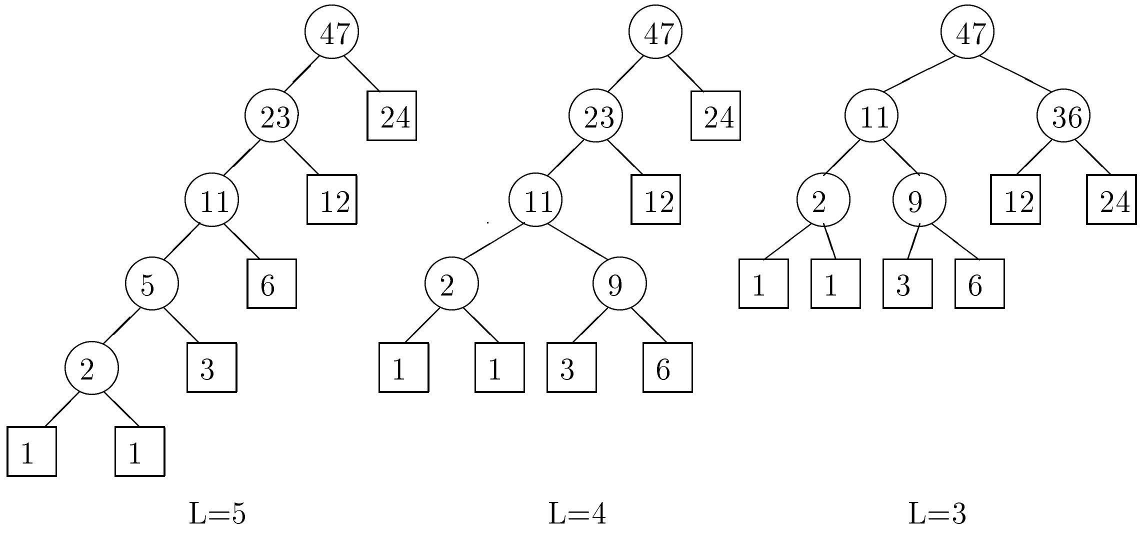


Abbildung 4.1: optimale L-beschränkte Bäume zu den Gewichten 1,1,3,6,12,24

4.1 Allgemeine Sätze

Man kann die Höhe eines Baumes bei vorgegebener Blattanzahl nicht beliebig einschränken. Da ein vollständiger r-närer Baum mit L Ebenen genau 2^L Blätter enthält, kann man den folgenden Satz formulieren:

Satz 4.1 *Ein L-beschränkter r-närer Baum mit n Blättern existiert genau dann, wenn* $L \geq \log_r n$.

4.2 Das Verfahren von Hu und Tan

Das Problem, einen optimalen binären Baum zu n gewichteten Blättern zu finden, läßt sich folgendermaßen formulieren:

$$\text{Finde } l_1, \ldots, l_n \ (l_i \geq 1), \text{ so daß} \sum_{i=1}^{n} w_i l_i \text{ minimal ist und} \sum_{i=1}^{n} 2^{-l_i} = 1.$$

$l_1, \ldots, l_n$ sind dabei die Pfadlängen der Blätter. $\sum_{i=1}^{n} 2^{-l_i} = 1$ ergibt sich aus der Kraftschen Ungleichung und der Tatsache, daß binäre optimale Bäume stets regulär sind (dies folgt aus Satz 2.10).

Da man die Pfadlängen nach oben durch L beschränken möchte, ist in diesem Kapitel folgendes Problem zu lösen:

$$\text{Finde } l_1, \ldots, l_n \ (L \geq l_i \geq 1) \text{, so daß} \sum_{i=1}^{n} w_i l_i \text{ minimal ist und} \sum_{i=1}^{n} 2^{-l_i} = 1. \tag{4.1}$$

Um dies zu lösen, betrachtet man für $k = 1, \ldots, n$ und $I = 1, \ldots, 2^L$ das Problem

$$\text{Finde } l_1, \ldots, l_k \ (L \geq l_i \geq 1) \text{, so daß} \sum_{i=1}^{k} w_i l_i \text{ minimal ist und} \sum_{i=1}^{k} 2^{-l_i} = I2^{-L}. \tag{4.2}$$

Setzt man die gefundenen $l_1, \ldots, l_k$ in $\sum_{i=1}^{k} w_i l_i$ ein, so nennt man den erhaltenen Wert $\psi_k(I2^{-L})$. Falls das Problem für ein Paar (k, I) nicht lösbar ist, setzt man $\psi_k(I2^{-L}) = \infty$.

Formuliert und löst man Problem (4.2) für $k = n$ und $I = 2^L$, dann ist auch Problemstellung (4.1) gelöst.

Der Grund für die Betrachtung der komplexeren Aufgabe (4.2) liegt in der rekursiven Berechenbarkeit der Werte $\psi_k(I2^{-L})$.

Satz 4.2

- *$\psi_1(I2^{-L}) = w_1(L - \log_2 I)$ genau dann, wenn $\log_2 I$ ganzzahlig ist.*
- *Sind die Werte für einen festen Index $k \geq 1$ und $I = 1, \ldots, 2^L$ bekannt, dann gilt für $I \neq 1$*

$$\psi_{k+1}(I2^{-L}) = \min_{L - \lceil \log_2 I \rceil + 1 \leq t \leq L} (\psi_k(I2^{-L} - 2^{-t}) + w_{k+1} t).$$

Beweis 4.2

- $2^{-l_1} = I2^{-L}$ ist nur für ganzzahlige $\log_2 I$ lösbar und man erhält $l_1 = L - \log_2 I$.
-

$$\begin{aligned}
\psi_{k+1}(I2^{-L}) &= \min_{\substack{1 \leq l_1, \ldots, l_{k+1} \leq L \\ \sum_{i=1}^{k+1} 2^{-l_i} = I2^{-L}}} \sum_{i=1}^{k+1} w_i l_i \\
&= \min_{1 \leq t \leq L} \Big(\min_{\substack{1 \leq l_1, \ldots, l_k \leq L \\ \sum_{i=1}^{k} 2^{-l_i} = I2^{-L} - 2^{-t}}} \sum_{i=1}^{k} w_i l_i + w_{k+1} t \Big) \\
&= \min_{1 \leq t \leq L} (\psi_k(I2^{-L} - 2^{-t}) + w_{k+1} t)
\end{aligned}$$

Es genügt für t nur Werte größer $L - \lceil \log_2 I \rceil$ zu betrachten, da $I2^{-L} - 2^{-t} > 0$ gelten muß.

$\diamond$

4.2.1 Der Algorithmus

Der Algorithmus berechnet die $\psi_k(I2^{-L})$ nach folgendem Schema:

- alle $\psi_1(I2^{-L})$ für $I = 1, \ldots, 2^L$ werden berechnet
- alle $\psi_2(I2^{-L})$ für $I = 1, \ldots, 2^L$ werden berechnet
- $\ldots$
- alle $\psi_n(I2^{-L})$ für $I = 1, \ldots, 2^L$ werden berechnet

Wenn man sich die $\psi_k(I2^{-L})$ tabellarisch angeordnet vorstellt, in den Spalten I, in den Zeilen k, dann berechnet der Algorithmus die Werte von links nach rechts und von oben nach unten. Die gesuchten Kosten stehen im Eintrag $\psi_n(1)$ in Zeile n und Spalte 2^L.

Da man mit dieser Tabelle nur die Kosten des Baumes erhält, nicht aber seine Form (also die gesuchten l_i), ist es nötig, noch einen weiteren Wert in der Tabelle zu speichern. Zu speichern ist das t, für das $\psi_{k-1}(I2^{-L} - 2^{-t}) + w_k t$ minimiert wurde. Dieses erhält den Namen $l_k(I)$ und gibt die Länge des Pfades des k.ten Gewichts an. Die Gestalt des Baumes erhält man daraus rekursiv: Das n.te Blatt hat die Länge $l_n(2^L)$. Ist die Länge $l_k = l_k(I)$ des k.ten Gewichts bekannt, dann hat das $k-1$.te Gewicht die Länge $l_{k-1}(I - 2^{L-l_k})$.

Der Algorithmus füllt also eine Tabelle mit 2^L Spalten, numeriert mit $1, \ldots, 2^L$, und n Doppelzeilen, in denen die Paare $(\psi_k(I2^{-L}), l_k(I))$ für $k = 1, \ldots, n$ abgelegt werden und hat die Komplexität $O(nL2^L)$. Denn das Suchen eines geeigneten t zur Minimierung von $\psi_{k-1}(I2^{-L} - 2^{-t}) + w_k t$ beschränkt sich auf $\lceil \log_2 I \rceil$ viele Werte. Der Abschnitt des Algorithmus, in dem die Tabelle gefüllt wird für alle $k \neq 1$, umfaßt also

$$(n-1) \sum_{I=2}^{2^L} \lceil \log_2 I \rceil = O(nL2^L)$$

viele Operationen.

```
PROCEDURE HuTan(v_1, ..., v_n:gewichtete Knoten,L:INTEGER);
VAR I, k, t:INTEGER;
BEGIN
FOR I = 1 TO 2^L DO
   IF log_2 I ist ganzzahlig
      THEN l_1(I) := L − log_2 I;
           ψ_1(I2^{−L}) := w_1 l_1(I);
      ELSE ψ_1(I2^{−L}) := ∞;
   END;
END;
FOR k = 2 TO n DO
   FOR I = 2 TO 2^L DO
      Finde t ∈ {L − ⌈log_2 I⌉ + 1, ..., L},
        so daß ψ_{k−1}(I2^{−L} − 2^{−t}) + w_k t minimal ist;
      ψ_k(I2^{−L}) := ψ_{k−1}(I2^{−L} − 2^{−t}) + w_k t;
      l_k(I) := t;
   END;
END;
END.
```

Tabelle 4.1: Algorithmus zur Methode von Hu und Tan

k, I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ψ_1	96	72		48				**24**								0
l_1	4	3		2				**1**								0
ψ_2		144	120	108	96	84		72	72	60		**48**				36
l_2		4	4	3	4	3		2	4	3		**2**				1
ψ_3			168	144	132	120	108	102	96	90	84	78	**72**	66		60
l_3			4	4	4	4	4	3	4	3	4	3	**4**	3		2
ψ_4				180	156	144	132	120	114	108	102	96	90	**84**	78	75
l_4				4	4	4	4	4	4	4	4	4	4	**4**	4	3
ψ_5					184	160	148	136	124	118	112	106	100	94	**88**	82
l_5					4	4	4	4	4	4	4	4	4	4	**4**	4
ψ_6						188	164	152	140	128	122	116	110	104	98	**92**
l_6						4	4	4	4	4	4	4	4	4	4	**4**

Tabelle 4.2: Werte für den 4-beschränkten Baum zum Beispiel 4.2.2

4.2.2 Beispiel

Die gegebenen Gewichte seien 24,12,6,3,1,1. Dann hat jeder Huffman-Baum zu diesen Gewichten fünf Ebenen, ist also 5-beschränkt. Die Kosten eines optimalen Baumes betragen 88.

Wird ein 4-beschränkter optimaler Baum gesucht, liefert der Algorithmus die Werte von Tabelle 4.2. Die hervorgehobenen Werte sind dabei jene, welche man beim Zurückverfolgen der Pfadlängen betrachten muß. Man erhält also $l_1 = 1, l_2 = 2, l_3 = 4, l_4 = 4, l_5 = 4, l_6 = 4$ und die Kosten des Baumes betragen damit 92.

Wird ein 3-beschränkter optimaler Baum gesucht, erhält man Tabelle 4.3 und die Pfadlängen $l_1 = 2, l_2 = 2, l_3 = 3, l_4 = 3, l_5 = 3, l_6 = 3$ mit Kosten 105.

Die zugehörigen Bäume sind in Abbildung 4.1 spiegelverkehrt dargestellt.

4.2.3 Einschränkung der zu berechnenden I

Es werden Aussagen vorgestellt, die die Komplexität nicht senken, aber die Berechnung vieler Einträge vereinfachen.

Die Menge der nicht berechenbaren Einträge läßt sich mittels des nächsten Satzes charakterisieren.

Satz 4.3 *Sei $\varphi(I)$ die Anzahl von Einsen in der binären Darstellung von I. Es existieren genau dann Werte $l_1, \ldots, l_k$ mit $\sum_{i=1}^{k} 2^{-l_i} = I 2^{-L}$, wenn $I \geq k \geq \varphi(I)$.*

k, I	1	2	3	4	5	6	7	8
ψ_1	72	**48**		24				0
l_1	3	**2**		1				0
ψ_2		108	84	**72**	60	48		36
l_2		3	3	**2**	3	2		1
ψ_3			126	102	**90**	78	66	60
l_3			3	3	**3**	3	3	2
ψ_4				135	111	**99**	87	75
l_4				3	3	**3**	3	3
ψ_5					138	114	**102**	90
l_5					3	3	**3**	3
ψ_6						141	117	**105**
l_6						3	3	**3**

Tabelle 4.3: Werte für den 3-beschränkten Baum zum Beispiel 4.2.2

Beweis 4.3

- Sei $\sum_{i=1}^{k} 2^{-l_i} = I2^{-L}$. Dann ist $\sum_{i=1}^{k} 2^{L-l_i} = I$ und jeder Summand erzeugt höchstens eine Eins in der binären Darstellung von I. Deshalb folgt $k \geq \varphi(I)$.
  $I \geq k$ folgt aus $I2^{-L} = \sum_{i=1}^{k} 2^{-l_i} \geq \sum_{i=1}^{k} 2^{-L} = k2^{-L}$.

- Sei $I \geq k \geq \varphi(I)$. Der Beweis erfolgt durch Angabe einer Konstruktion, die einen Baum erstellt mit $\sum_{i=1}^{k} 2^{-l_i} = I2^{-L}$:
  Man zählt in der binären Darstellung von I die Nullen und Einsen folgendermaßen von rechts nach links durch: Eine Ziffer steht an Stelle j, wenn sie für I den Wert 2^j liefert. Bestimmt man $\varphi(I) - 1$ viele l_i so, daß $L - l_i$ der Wert der Stelle der i-ten Eins ist, dann bleiben $k - \varphi(I) + 1$ viele l_i ohne Wertzuweisung. Also faßt man $k - \varphi(I) + 1$ viele Knoten zu einem Baum zusammen, der seine Wurzel auf Ebene q haben soll. Dabei soll q der Wert sein, den man erhält, wenn man von L die Stelle der $\varphi(I)$-ten Eins subtrahiert. Man hat dann einen Baum mit k Knoten konstruiert, der $\sum_{i=1}^{k} 2^{-l_i} = I2^{-L}$ erfüllt.

$\diamond$

Satz 4.4 *Es existiere zu gegebenen (k, I) eine Lösung des Problems (4.2). Sind die Blätter nach fallenden Gewichten geordnet, dann gelten folgende Aussagen.*

1. *Ist I ungerade, dann gilt $l_k(I) = L$.*

2. *Ist* $I = 2^p q$, *q ungerade, dann gilt* $l_k(I) \geq L - p$.

3. *Ist* $\min_{I=1,\ldots,2^L} l_k(I) = t$, *dann ist* $l_{k+1}(I) \geq t$ *für alle* $I = 1, \ldots, 2^L$. *Insbesondere gilt: Ist* $l_k(I) = L$ *für alle* $I = 1, \ldots, 2^L$, *dann gilt* $l_{k+1}(I) = \ldots = l_n(I) = L$.

Beweis 4.4 Aufgrund von Satz 2.10 existieren Werte $l_1, \ldots, l_n$ mit $l_1 \leq \ldots \leq l_n$.

1. Ist $l_k < L$, dann gilt auch $l_i < L$ für alle $i = 1, \ldots, k$. Dann ist $2^{L-l_i} > 1$ für $i = 1, \ldots, k$ und somit $I = \sum_{i=1}^{k} 2^{L-l_i}$ stets gerade.

2. Ist $l_k < L - p$, dann gilt auch $l_i < L - p$ für alle $i = 1, \ldots, k$. Dann ist $2^{L-l_i-p} > 1$ für $i = 1, \ldots, k$ und somit $q = \sum_{i=1}^{k} 2^{L-l_i-p}$ stets gerade.

3. Folgt aus $l_k \leq l_{k+1}$.

$\diamond$

4.2.4 Beschränkung der Komplexität und das äquivalente Graphenproblem

Larmore zeigt, daß die Anzahl der zu betrachtenden t bei der Minimierung von $\psi_{k-1}(I2^{-L} - 2^{-t}) + w_k t$ eingeschränkt werden kann, wenn die Blätter nach Gewichten sortiert eingegeben werden. Die Komplexität reduziert sich dann auf $O(Ln^2)$. Dazu führt er die Lösung der von Hu und Tan formulierten Problemstellung auf eine „kürzeste Pfad"-Suche in azyklischen, gerichteten Graphen zurück.

Abstraktion der Tabelle zum Hu-Tan-Graphen

Sei eine $k \times I$-Tabelle gegeben, in deren I-ter Spalte und k-ter Zeile der Wert $\psi_k(I2^{-L})$ eingetragen ist. Diese Position wird (k, I) genannt. Die Tabelle hat $n2^L$ Einträge. Einen Hu-Tan-Graphen erhält man folgendermaßen:

- Ein zusätzlicher Punkt außerhalb der Tabelle sei gegeben. Man verbinde diesen externen Punkt mit allen Positionen $(1, 2^p)$. Diese Verbindungslinien werden $\varepsilon_1[I, L - p]$ genannt und ihnen wird das Gewicht $w_1(L-p)$ zugewiesen. Die Kanten sind vom externen Punkt weggerichtet.

- Für jedes t zwischen $L - \lceil \log_2 I \rceil + 1$ und L verbindet man die Positionen $(k-1, I-2^{L-t})$ und (k, I). Die Verbindungslinie heißt $\varepsilon_k[I, t]$, zeigt auf (k, I) und trägt das Gewicht $w_k t$.

Man stellt fest, daß genau jene ψ_{k-1} mit $\psi_k(I2^L)$ verbunden sind, die beim rekursiven Bestimmen von $\psi_k(I2^{-L})$ betrachtet werden müssen.

Hu und Tans Algorithmus liefert für dieses Graphenkonstrukt einen minimal gewichteten Pfad vom externen Punkt zum Eintrag $\psi_n(1)$. Allgemein läßt sich feststellen, daß Hu und Tans Methode der klassische dynamische Programmierungs-Algorithmus zum Finden minimal gewichteter Wege in azyklischen gerichteten Graphen ist.

Die Komplexität des Algorithmus entspricht der Anzahl der zu betrachtenden Kanten im Hu-Tan-Graphen, da diese angeben, wieviele Berechnungen $\psi_{k-1}(I2^{-L}-2^{-t})+w_k t$ ausgeführt werden müssen. Die Anzahl der Kanten ist exponentiell in L. Jedoch wird durch den folgenden Satz die Anzahl der notwendig zu betrachtenden Kanten eingeschränkt.

Satz 4.5 *Seien die Gewichte monoton fallend geordnet. Es genügt, wenn man sich bei der Suche eines minimalen Weges im Hu-Tan-Graphen nur auf Kanten $\varepsilon_k[I,t]$ mit $2^t - I2^{t-L} \in \{0,\dots,n-k\}$ beschränkt.*

Beweis 4.5 Es wird gezeigt, daß ein Lösungsweg nur Kanten mit der geforderten Bedingung enthält. Ein Lösungsweg besteht aus Kanten $\varepsilon_k[I,l_k]$, die von $\psi_{k-1}(I2^{-L}-2^{-l_k})$ nach $\psi_k(I2^{-L})$ führen. Da die Gewichte monoton fallen, kann man annehmen, daß die zugehörigen Pfadlängen monoton steigen und somit $l_i \le l_k \le l_j$ für $i=1,\dots,k-1$ und $j=k+1,\dots,n$ ist. Dann gilt einerseits

$$2^t - I2^{t-L} = 2^{l_k} - I2^{l_k-L} = 2^{l_k} - \sum_{i=1}^{k} 2^{l_k-l_i} \text{ ist ganzzahlig}$$

und andererseits

$$2^{l_k} - I2^{l_k-L} = 2^{l_k}(1-\sum_{i=1}^{k} 2^{-l_i}) = 2^{l_k}\sum_{i=k+1}^{n} 2^{-l_i} = \sum_{i=k+1}^{n} 2^{l_k-l_i} \le n-k.$$

◇

Beispiel 4.6 Links in Abbildung 4.2 ist der Hu-Tan-Graph zum Beispiel 4.2.2 angegeben für $L=3$. Die Bezugstabelle ist also Tabelle 4.3. Rechts ist der Hu-Tan-Graph abgebildet, in dem nur die notwendig zu betrachtenden Kanten eingetragen sind. Der Lösungsweg ist in beiden Graphen **fett** markiert.

Satz 4.7 *Die Anzahl der zu betrachtenden Kanten ist $O(Ln^2)$.*

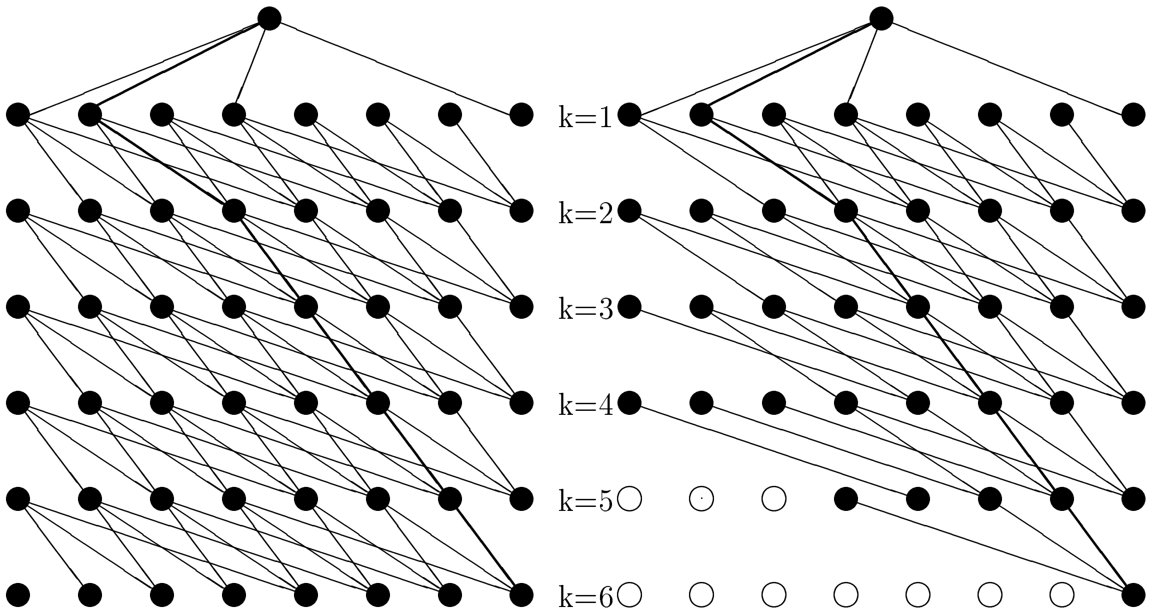


Abbildung 4.2: Hu-Tan-Graph zum Beispiel 4.2.2

Beweis 4.7 Seien k, t fest gewählt. Aufgrund von Satz 4.5 gibt es genau $n - k + 1$ mögliche I. Dies liefert

$$\sum_{t=1}^{L} \sum_{k=2}^{n} (n - k + 1) = LO(n^2) = O(Ln^2)$$

Kanten. ⋄

4.3 Das Verfahren von Garey

Garey löst das Problem rekursiv, indem er es auf ein Problem für die Teilbäume des gesuchten Baumes zurückführt. Dies geschieht analog zur Methode von Gilbert und Moore zur Berechnung optimaler alphabetischer Bäume (siehe Kapitel 3.3).

Satz 4.8 *Jeder Teilbaum eines optimalen L-beschränkten Baumes, der seine Wurzel auf Ebene k hat, ist ein optimaler $(L - k)$-beschränkter Baum.*

Beweis 4.8 Ist der Teilbaum nicht optimal, dann kann man ihn durch einen optimalen $(L - k)$-beschränkten Baum ersetzen. Damit findet man einen Baum mit kleineren Kosten, dies ist ein Widerspruch zur Optimalität. ⋄

Sei $[i, j, k]$ die Aufgabe, einen optimalen k-beschränkten Baum auf den Knoten $v_i, \ldots, v_j$ zu finden. $K[i, j, k]$ seien die Kosten dieses Baumes. Die Lösung des ursprünglichen Problems ist gefunden, wenn man $[1, n, L]$ gelöst hat. Auch bei dieser Problemstellung existiert, wie bei Hu und Tan, für bestimmte Werte keine Lösung. In diesem Fall setzt man $K[i, j, k] = \infty$.

Satz 4.9 *Es existiert genau dann eine Lösung von $[i, j, k]$, wenn $k \geq \log_2(j - i + 1)$ ist.*

Beweis 4.9 Folgt aus Satz 4.1. $\diamond$

$[i, j, k]$ reduziert man folgendermaßen auf ein „einfacheres“ Problem: Man spaltet $\{v_i, \ldots, v_j\}$ auf in $\{v_i, \ldots, v_N\}$ und $\{v_{N+1}, \ldots, v_j\}$. Aufgrund von Satz 4.8 müssen auf diesen Mengen optimale Bäume konstruiert werden. Die Kosten des Baumes auf $\{v_i, \ldots, v_j\}$ setzen sich also zusammen aus $K[i, N, k-1]$ und $K[N+1, j, k-1]$ und den Gewichten $w_i, \ldots, w_j$, da diese sich im gesuchten Baum eine Ebene tiefer befinden, als in den beiden Teilbäumen. Da der Baum auf $\{v_i, \ldots, v_j\}$ optimal sein soll, ist das N zu suchen, das die Summe der Kosten der Teilbäume minimiert.

Da die Reihenfolge der Blätter vom Algorithmus nicht geändert wird, müssen die Blätter nach Gewichten sortiert eingegeben werden. Wird dies nicht getan, erhält man einen optimalen L-beschränkten alphabetischen Baum, der nicht zwangsläufig optimal L-beschränkt ist. Die Sortierung der Gewichte ist notwendig, da ein optimaler alphabetischer Baum auch im allgemeinen Sinne optimal ist, wenn die Gewichte monoton geordnet sind.

Satz 4.10 *Sei $i < j$ und $i, j = 1, \ldots, n$.*

- $K[i, i, k] = 0$
- *Falls eine Lösung von $[i, j, k]$ existiert, gilt*

$$K[i, j, k] = \sum_{p=i}^{j} w_p + \min_{i \leq N \leq j-1} (K[i, N, k-1] + K[N+1, j, k-1])$$

Der Wert N, der den Ausdruck minimiert, muß gespeichert werden, um aus den berechneten Werten einen Baum konstruieren zu können. Dieser Wert wird in der Variablen $N[i, j, k]$ abgelegt mit $N[i, i, k] := i$. Nach Berechnung aller $K[i, j, k]$ (die man auch hier in Tabellenform ablegen kann) konstruiert man den Baum über die gespeicherten $N[i, j, k]$ folgendermaßen: Der Baum zur Lösung der Problems $[i, j, k]$ besteht aus einer nicht näher spezifizierten Wurzel, einem linken Teilbaum, der die Lösung des Problems $[i, N[i, j, k], k-1]$ ist, und einem rechten Teilbaum, der die Lösung des Problems $[N[i, j, k] + 1, j, k-1]$ darstellt.

```
PROCEDURE Garey(v_1, ..., v_n:gewichtete Knoten,L_max:INTEGER);
VAR i, p, k, N:INTEGER;
BEGIN
FOR (i := 1 TO n) AND (k := 0 TO L_max) DO
    K[i, i, k] := 0;
    N[i, i, k] := i;
END;
FOR k := 1 TO L_max DO
    FOR p := 1 TO n - 1 DO
        FOR i := 1 TO n - p DO
            Finde N ∈ {i, ..., i + p - 1},
              so daß K[i, N, k - 1] + K[N + 1, i + p, k - 1] minimal ist;
            K[i, i + p, k] = Σ_{j=i}^{i+p} w_j + K[i, N, k - 1] + K[N + 1, i + p, k - 1];
            N[i, i + p, k] = N;
        END;
    END;
END;
END.
```

Tabelle 4.4: Algorithmus zur Methode von Garey

4.3.1 Der Algorithmus

Der Algorithmus berechnet die $K[i, j, k]$ nach folgendem Schema:

- Die $K[i, i, k]$ werden berechnet für $k = 0, \ldots, L$ und $i = 1, \ldots, n$
- Für jedes feste k $(k = 1, \ldots, L)$ wird berechnet:
  - $K[i, i + 1, k]$ für $i = 1, \ldots, n - 1$
  - $K[i, i + 2, k]$ für $i = 1, \ldots, n - 2$
  - ...
  - $K[i, i + n - 2, k]$ für $i = 1, 2$
  - $K[1, n, k]$

Der Algorithmus füllt eine Tabelle mit $L + 1$ Spalten, numeriert mit $0, \ldots, L$, und $\frac{n^2+n}{2}$ Doppelzeilen, in denen die Paare $(K[i, j, k], N[i, j, k])$ mit $j - i \geq 0$ abgelegt werden.

Beim Erstellen der Werte der Tabelle werden alle $K[1, n, L]$ für $L \leq L_{\max}$ berechnet. Man erhält also die Kosten und die Form aller optimalen

L-beschränkten Bäume für $L \leq L_{\max}$ mit einer einzigen Ausführung des Algorithmus.

Der Algorithmus hat die Komplexität $O(Ln^3)$: Das Suchen eines geeigneten N zur Minimierung von $K[i,N,k-1]+K[N+1,i+p,k-1]$ beschränkt sich auf $p-1$ viele Werte. Der Abschnitt des Algorithmus, in dem die Tabelle für alle $p \geq 1$ gefüllt wird, umfaßt also

$$L \sum_{p=1}^{n-1} \sum_{i=1}^{n-p} (p-1) = O(Ln^3)$$

viele Operationen.

4.3.2 Beispiel

Mit den Gewichten 24,12,6,3,1,1 und $L_{\max} = 5$ erhält man Tabelle 4.5. Die relevanten Einträge stehen in $K[1,6,5], K[1,6,4]$ und $K[1,6,3]$, die Konstruktion der Bäume aus den Werten ist neben der Tabelle 4.5 dargestellt.

4.3.3 Verminderung der Komplexität

Die Komplexität des Algorithmus läßt sich verbessert bestimmen zu $O(Ln^2)$. Dies ist möglich, da man die Anzahl der zu betrachtenden N zur Minimierung von $K[i,N,k-1]+K[N+1,i+p,k-1]$ wie bei Gilbert und Moore in Kapitel 3.3 auf $1+N[i+1,p+i,k]-N[i,p+i-1,k]$ viele einschränken kann. Die Beweise sind analog zu den Beweisen 3.9 und 3.10 in Kapitel 3.

Satz 4.11 *Sei $i_1 \leq i_2 \leq j_1 \leq j_2$. Dann gilt*

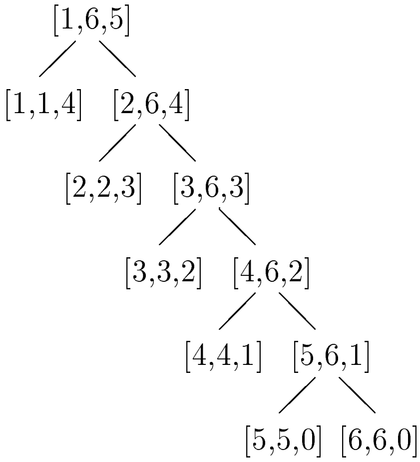
$$K[i_1,j_2,k]+K[i_2,j_1,k] \geq K[i_1,j_1,k]+K[i_2,j_2,k].$$

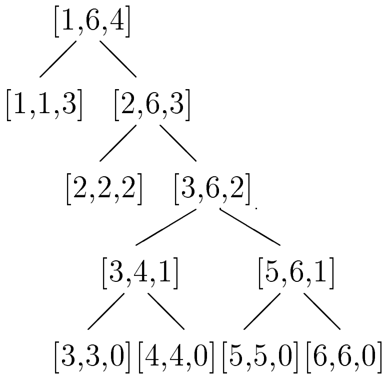
Satz 4.12 *Ist $i_1 \leq i_2 \leq j_1 \leq j_2$, $i_1 < j_1$ und $i_2 < j_2$. Dann gilt*

$$N[i_1,j_1,k] \leq N[i_2,j_2,k].$$

Für jedes $N[i,j,k]$ mit $j-i>1$ gilt also sowohl $N[i,j-1,k] \leq N[i,j,k]$ wie auch $N[i,j,k] \leq N[i+1,j,k]$. Beim Suchen eines geeigneten N zur Minimierung von $K[i,N,k-1]+K[N+1,i+p,k-1]$ mit $p \geq 2$ brauchen also lediglich $1+N[i+1,i+p,k]-N[i,i+p-1,k]$ viele Werte betrachtet werden, nämlich die Werte zwischen $N[i,i+p-1,k]$ und $N[i+1,i+p,k]$. Der

[i,j],k	0	1	2	3	4	5
K[1,1]	0	0	0	0	0	0
N[1,1]	1	1	1	1	1	1
K[1,2]		36	36	36	36	36
N[1,2]		1	1	1	1	1
K[1,3]			60	60	60	60
N[1,3]			1	1	1	1
K[1,4]			90	75	75	75
N[1,4]			2	1	1	1
K[1,5]				90	82	82
N[1,5]				1	1	1
K[1,6]				105	92	88
N[1,6]				2	1	1
K[2,2]	0	0	0	0	0	0
N[2,2]	2	2	2	2	2	2
K[2,3]		18	18	18	18	18
N[2,3]		2	2	2	2	2
K[2,4]			30	30	30	30
N[2,4]			2	2	2	2
K[2,5]			44	36	36	36
N[2,5]			3	2	2	2
K[2,6]				45	41	41
N[2,6]				2	2	2
K[3,3]	0	0	0	0	0	0
N[3,3]	3	3	3	3	3	3
K[3,4]		9	9	9	9	9
N[3,4]		3	3	3	3	3
K[3,5]			14	14	14	14
N[3,5]			3	3	3	3
K[3,6]			22	18	18	18
N[3,6]			4	3	3	3
K[4,4]	0	0	0	0	0	0
N[4,4]	4	4	4	4	4	4
K[4,5]		4	4	4	4	4
N[4,5]		4	4	4	4	4
K[4,6]			7	7	7	7
N[4,6]			4	4	4	4
K[5,5]	0	0	0	0	0	0
N[5,5]	5	5	5	5	5	5
K[5,6]		2	2	2	2	2
N[5,6]		5	5	5	5	5
K[6,6]	0	0	0	0	0	0
N[6,6]	6	6	6	6	6	6





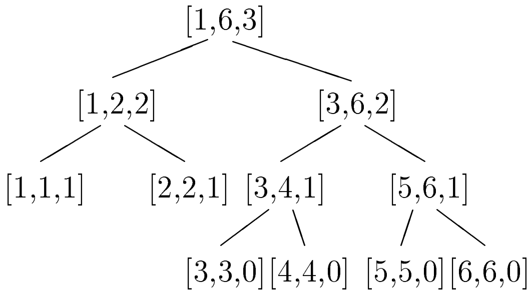


Tabelle 4.5: Werte für $L_{\max} = 5$ zum Beispiel 4.3.2

Abschnitt des Algorithmus, in dem die Tabelle gefüllt wird für alle $p \geq 2$, umfaßt folglich

$$\begin{aligned} L\sum_{p=2}^{n-1}\sum_{i=1}^{n-p}(1+N[i+1,i+p,k]-N[i,i+p-1,k]) &= \\ L\sum_{p=2}^{n-1}((n-p)+N[n-p+1,n,k]-N[1,p,k]) &\leq \\ L\sum_{p=2}^{n-1}(2n-p-2) &= O(Ln^2) \end{aligned}$$

viele Operationen.

Um den Algorithmus mit dieser Komplexität realisieren zu können, muß man die Summen $\sum_{j=1}^{p} w_j$ geschickt berechnen, indem man sich auf vorher berechnete Summen bezieht.

4.4 Das Verfahren von Larmore

Larmore kombiniert die Algorithmen von Hu und Tan und Garey unter Ausnutzung der Gleichung

$$\min_{\substack{1\leq l_s,\ldots,l_k\leq L \\ \sum_{i=s}^{k} 2^{-l_i}=2^{-t}}} \sum_{i=s}^{k} w_i l_i = K[s,k,L-t]+\sum_{i=s}^{k} w_i t \tag{4.3}$$

Zusätzlich nutzt er eine Erweiterung des Hu-Tan Graphen, genannt G^+.

Definition 4.13 *Der Graph G^+ erfüllt folgende Bedingungen:*

- *Die Eckenmenge von G^+ entspricht der Eckenmenge von G, enthält also den externen Punkt und die Positionen (k,I) für $k=1,\ldots,n$ und $I=1,\ldots,2^L$.*
- *Der externe Punkt wird mit allen Positionen $(k,2^p)$ verbunden. Diese Verbindungslinien heißen $\zeta_{0,k}[2^p,L-p]$ und führen vom externen Punkt weg. Die Gewichte sind $K[1,k,p]+\sum_{i=1}^{k} w_i(L-p)$.*
- *Die Positionen $(s,I-2^{L-t})$ und (k,I) werden verbunden, falls $t \in \{L-\lceil\log_2 I\rceil+1,\ldots,L\}$ und $s<k$ ist. Die Verbindungslinien heißen $\zeta_{s,k}[I,t]$ und zeigen auf die Position (k,I). Sie tragen das Gewicht $K[s+1,k,L-t]+\sum_{i=s+1}^{k} w_i t$.*

G ist ein Untergraph von G^+, da $\varepsilon_k[I,t] = \zeta_{k-1,k}[I,t]$ mit $w(\varepsilon_k[I,t]) = w_k t = K[k,k,L-t] + w_k t = w(\zeta_{k-1,k}[I,t])$ ist. Diese Beziehung macht man sich beim Beweis des folgenden Satzes zunutze:

Satz 4.14 *Der minimal-gewichtete Pfad zwischen den Positionen $(s, I - 2^{L-t})$ und (k, I) hat in G und G^+ denselben Wert.*

Beweis 4.14 Sei α ein minimal-gewichteter Graph in G von $(s, I-2^{L-t})$ nach (k,I). Dann hat dieser Pfad das Gewicht

$$
\begin{aligned}
w(\alpha) &= \psi_k(I2^{-L}) - \psi_s(I2^{-L} - 2^{-t}) \\
&\overset{(4.3)}{=} \min_{\substack{1 \le l_1,\ldots,l_k \le L \\ \sum_{i=1}^k 2^{-l_i} = I2^{-L}}} \sum_{i=1}^k w_i l_i - \min_{\substack{1 \le l_1,\ldots,l_s \le L \\ \sum_{i=1}^s 2^{-l_i} = I2^{-L} - 2^{-t}}} \sum_{i=1}^s w_i l_i \\
&= \min_{\substack{1 \le l_{s+1},\ldots,l_k \le L \\ \sum_{i=s+1}^k 2^{-l_i} = 2^{-t}}} \sum_{i=s+1}^k w_i l_i
\end{aligned}
$$

Da G Untergraph von G^+ ist, genügt es zu zeigen, daß für jeden Pfad α^+ in G^+ von $(s, I-2^{L-t})$ nach (k,I) gilt: $w(\alpha^+) \ge w(\alpha)$.
α^+ bestehe aus den Kanten $\zeta_{s_0,s_1}[I - \sum_{i=2}^p 2^{L-t_i}, t_1]$, $\zeta_{s_1,s_2}[I - \sum_{i=3}^p 2^{L-t_i}, t_2]$, $\ldots$, $\zeta_{s_{p-1},s_p}[I, t_p]$ mit $s_0 = s$ und $s_p = k$.
Dann gilt

$$
w(\zeta_{s_j,s_{j+1}}[I - \sum_{i=j+2}^p 2^{L-t_i}, t_{j+1}]) \overset{(4.3)}{=} \min_{\substack{1 \le l_{s_j+1},\ldots,l_{s_{j+1}} \le L \\ \sum_{i=s_j+1}^{s_{j+1}} 2^{-l_i} = 2^{-t_{j+1}}}} \sum_{i=s_j+1}^{s_{j+1}} w_i l_i \tag{4.4}
$$

und wegen $I - \sum_{j=1}^p 2^{L-t_j} = I - 2^{L-t}$ gilt weiterhin

$$
\sum_{j=1}^p 2^{-t_j} = 2^{-t} \tag{4.5}
$$

und es folgt

$$
w(\alpha^+) = \sum_{j=0}^{p-1} w(\zeta_{s_j,s_{j+1}}[I - \sum_{i=j+2}^p 2^{L-t_i}, t_{j+1}])
$$

$$\stackrel{(4.4)}{=} \sum_{j=0}^{p-1} \Big(\min_{\substack{1 \le l_{s_j+1}, \ldots, l_{s_{j+1}} \le L \\ \sum_{i=s_j+1}^{s_{j+1}} 2^{-l_i} = 2^{-t_{j+1}}}} \sum_{i=s_j+1}^{s_{j+1}} w_i l_i \Big)$$

$$\stackrel{(4.5)}{\ge} \min_{\substack{1 \le l_{s+1}, \ldots, l_k \le L \\ \sum_{i=s+1}^{k} 2^{-l_i} = 2^{-t}}} \sum_{i=s+1}^{k} w_i l_i = w(\alpha)$$

$\diamond$

Mithilfe des Graphen G^+ und einem vorerst beliebigen Wert g definiert man nun einen Untergraphen $\bar{G}$ von G^+. Für diesen soll gelten:

- er besitzt $O(Ln^2)$ viele Kanten
- ist $\bar{\psi}_k(I2^{-L})$ das minimale Gewicht eines Pfades in $\bar{G}$ vom externen Punkt zur Position (k, I), dann soll $\psi_n(1) = \bar{\psi}_n(1)$ sein.

Definition 4.15 *$\bar{G}$ wird folgendermaßen bestimmt:*

- *Der externe Punkt liegt in $\bar{G}$.*
- *Eine Position (k, I) aus G^+ liegt genau dann in $\bar{G}$, wenn ein t existiert, so daß $I2^{-L} = 1 - a2^{-t}$ für ein $a \in \{0, \ldots, \lfloor \frac{2n}{g} \rfloor\}$.*
- *Eine Kante $\zeta_{s,k}[I, t]$ aus G^+ liegt genau dann in $\bar{G}$, wenn $I2^{-L} = 1 - a2^{-t}$ für ein $a \in \{0, \ldots, \lfloor \frac{2n}{g} \rfloor\}$ und $0 \le k - s \le g$.*

Satz 4.16 *$\bar{G}$ hat $O(Ln^2)$ viele Kanten und es gibt $O(\frac{Ln}{g})$ viele Paare (I, t).*

Beweis 4.16 Da in $\bar{G}$ nur Paare (I, t) betrachtet werden, für die $I2^{-L} = 1 - a2^{-t}$ gilt und a nur $\frac{2n}{g} + 1$ viele Werte annehmen kann, erhält man für ein fest gewähltes t $O(\frac{n}{g})$ mögliche I. Da $t \le L$, gibt es $O(\frac{Ln}{g})$ mögliche Paare (I, t). Jedes dieser Paare erzeugt g Kanten, man erhält also $O(Ln^2)$ viele Kanten. $\diamond$

Satz 4.17 $\bar{\psi}_k(I2^{-L}) \ge \psi_k(I2^{-L})$.

Beweis 4.17 Folgt sofort aus Satz 4.14, da $\bar{G}$ Untergraph von G^+ ist. $\diamond$

Larmores Algorithmus sucht einen minimal gewichteten Pfad im Graphen $\bar{G}$. Dieses Verfahren löst das ursprüngliche Problem bei fallender Ordnung der Gewichte – siehe Satz 4.18. Die Gestalt des Baumes erhält man auch hier über den minimal gewichteten Pfad, da dieser Informationen zur Gestalt des Baumes enthält.

Satz 4.18 *Wenn die Gewicht $w_1, \ldots, w_n$ monoton fallend geordnet sind, dann gilt $\bar{\psi}_n(1) = \psi_n(1)$*

Beweis 4.18 Aufgrung von Satz 2.10 existieren Werte $l_1 \leq \ldots \leq l_n$, so daß $\sum_{i=1}^n w_i l_i$ minimal ist.
Sei $q_k := \sum_{i=1}^k 2^{L-l_i}$. Dann ist (k, q_k) eine Position in G, da q_k ganzzahlig ist und es gilt $\psi_k(q_k 2^{-L}) = \sum_{i=1}^k w_i l_i$.
Sei m die größte Zahl, so daß

1. ein t existiert mit $q_m 2^{-L} = 1 - a2^{-t}$ für ein ganzzahliges $a \in \{0, \ldots, \lfloor \frac{2n}{g} \rfloor\}$
2. $\bar{\psi}_m(q_m 2^{-L}) = \psi_m(q_m 2^{-L})$

Zeigt man $m = n$, ist der Satz bewiesen.
Angenommen, es ist $m < n$. Sei α ein Pfad in $\bar{G}$ vom externen Punkt nach (m, q_m) mit Gewicht $\psi_m(q_m 2^{-L})$. Man kann ohne Einschränkung annehmen, daß $a = 2^t(1 - q_m 2^{-L}) = \sum_{i=m+1}^n 2^{t-l_i}$ ungerade ist. Dann ist aber $t \leq l_{m+1}$.
Es werden zwei Fälle unterschieden:

- $t = l_{m+1}$
  Dann gilt

$$\begin{aligned} q_{m+1} 2^{-L} &= \sum_{i=1}^{m+1} 2^{-l_i} = \sum_{i=1}^{m} 2^{-l_i} + 2^{-l_{m+1}} = q_m 2^{-L} + 2^{-t} \\ &= 1 - a2^{-t} + 2^{-t} = 1 - (a-1)2^{-t}. \end{aligned}$$

  Somit erfüllt q_{m+1} ebenfalls Bedingung (1) und $\zeta_{m,m+1}[q_m, t]$ liegt in $\bar{G}$. Betrachtet man den Pfad $\bar{\alpha}$, der entsteht, wenn man an α die Kante $\zeta_{m,m+1}[q_{m+1}, t]$ anhängt, dann erhält man

$$\begin{aligned} \bar{\psi}_{m+1}(q_{m+1} 2^{-L}) &\leq w(\bar{\alpha}) = w(\alpha) + w(\zeta_{m,m+1}[q_{m+1}, t]) \\ &= \psi_m(q_m 2^{-L}) + w_{m+1} t = \sum_{i=1}^{m} w_i l_i + w_{m+1} l_{m+1} \\ &= \psi_{m+1}(q_{m+1} 2^{-L}). \end{aligned}$$

  Aufgrund von Satz 4.17 und dieser Abschätzung erfüllt auch q_{m+1} Bedingung (2). Dies ist ein Widerspruch zur Maximalität von m.

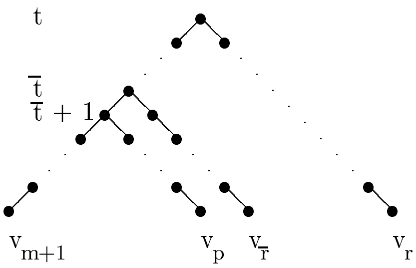


Abbildung 4.3: Lage der Variablen des Beweises 4.18

- $t < l_{m+1}$
  Dann existiert ein r, so daß $\sum_{i=m+1}^{r} 2^{t-l_i} = 1$ ist. (Dies ist nämlich der Fall, wenn der Teilbaum mit den Blättern $v_{m+1}, \ldots, v_r$ seine Wurzel auf Ebene t hat. Das ist möglich aufgrund der Bedingung $t < l_{m+1}$.)

  – Ist $r - m \leq g$, dann gilt

  $$q_m = q_m + 2^L\left(\sum_{i=m+1}^{r} 2^{-l_i} - 2^{-t}\right) = \sum_{i=1}^{m} 2^{L-l_i} + \sum_{i=m+1}^{r} 2^{L-l_i} - 2^{L-t} = q_r - 2^{L-t}.$$

  Also ist $(m, q_m) = (m, q_r - 2^{L-t})$. Ferner liegt $\zeta_{m,r}[q_r, t]$ in $\bar{G}$ und q_r erfüllt Bedingung (1), da

  $$q_r 2^{-L} = q_m 2^{-L} + 2^{-t} = 1 - (a-1)2^{-t}$$

  und

  $$r - m \leq g.$$

  Betrachtet man den Pfad $\bar{\alpha}$, der entsteht, indem man an α die Kante $\zeta_{m,r}[q_r, t]$ anhängt, dann erhält man

  $$\begin{aligned} \bar{\psi}_r(q_r 2^{-L}) &\leq w(\bar{\alpha}) = w(\alpha) + w(\zeta_{m,r}[q_r, t]) \\ &= \psi_m(q_m 2^{-L}) + K[m+1, r, L-t] + \sum_{i=m+1}^{r} w_i t = \psi_r(q_r 2^{-L}) \end{aligned}$$

  Damit erfüllt auch q_r Bedingung (2), und dies ist ein Widerspruch zur Maximalität von m.

  – Ist $r - m > g$, dann existiert ein $\bar{t}$ zu einem kleinsten $\bar{r}$ mit $\sum_{i=m+1}^{\bar{r}} 2^{\bar{t}-l_i} = 1$ und $\bar{r} - m > g$. (Man wählt den kleinsten Teilbaum mit den Blättern $v_{m+1}, \ldots, v_{\bar{r}}$ und der Wurzel auf Ebene $\bar{t}$.) p sei so gewählt, daß $\sum_{i=m+1}^{p} 2^{(\bar{t}+1)-l_i} = 1$ (v_p ist das am weitesten rechts liegende Kind des Teilbaumes, der v_{m+1} als am weitesten links liegendes Kind besitzt und die Wurzel auf Ebene $\bar{t}+1$ hat.) Dann gilt $p-m \leq g$ und

$$\begin{aligned} q_m &= q_m + 2^L(\sum_{i=m+1}^{p} 2^{-l_i} - 2^{-(\bar{t}+1)}) = \sum_{i=1}^{m} 2^{L-l_i} + \sum_{i=m+1}^{p} 2^{L-l_i} - 2^{L-(\bar{t}+1)} \\ &= q_p - 2^{L-(\bar{t}+1)}. \end{aligned}$$

Also ist $(m, q_m) = (m, q_p - 2^{L-(\bar{t}+1)})$. Es gilt ferner

$$\begin{aligned} q_p 2^{-L} &= q_m 2^{-L} + 2^{-(\bar{t}+1)} = 1 - a2^{-t} + 2^{-(\bar{t}+1)} \\ &= 1 - 2^{\bar{t}+1-t} a 2^{-(\bar{t}+1)} + 2^{-(\bar{t}+1)} = 1 - (2^{\bar{t}+1-t}a - 1)2^{-(\bar{t}+1)} \end{aligned}$$

und somit liegt $\zeta_{m,p}[q_p, \bar{t}+1]$ in $\bar{G}$ und q_p erfüllt Bedingung (1), wenn

$$2^{\bar{t}+1-t}a - 1 \leq \frac{2n}{g}$$

ist. Dies gilt aber, da

$$\begin{aligned} a &= 2^t(1 - q_m 2^{-L}) = 2^t(1 - \sum_{i=1}^{m} 2^{-l_i}) = 2^t \sum_{i=m+1}^{n} 2^{-l_i} \\ &= 2^t(\sum_{i=m+1}^{\bar{r}} 2^{-l_i} + \sum_{i=\bar{r}+1}^{2\bar{r}-m} 2^{-l_i} + \sum_{i=2\bar{r}-m+1}^{3\bar{r}-m} 2^{-l_i} + \ldots + \sum_{i=\frac{n-m}{\bar{r}-m}\bar{r}-m+1}^{n} 2^{-l_i}) \\ &\leq \frac{n-m}{\bar{r}-m} \sum_{i=m+1}^{\bar{r}} 2^{t-l_i} \leq \frac{n-m}{\bar{r}-m} 2^{t-\bar{t}} \sum_{i=m+1}^{\bar{r}} 2^{\bar{t}-l_i} = \frac{n-m}{\bar{r}-m} 2^{t-\bar{t}} \\ &\leq \frac{n}{g} 2^{t-\bar{t}} \end{aligned}$$

und somit

$$2^{\bar{t}+1-t}a - 1 \leq 2^{\bar{t}+1-t}\frac{n}{g}2^{t-\bar{t}} - 1 \leq \frac{2n}{g}.$$

Betrachtet man den Pfad $\bar{\alpha}$, der entsteht, indem man an α die Kante $\zeta_{m,p}[q_p, t]$ anhängt, dann erhält man

$$\begin{aligned} \bar{\psi}_p(q_p 2^{-L}) &\leq w(\bar{\alpha}) = w(\alpha) + w(\zeta_{m,p}[q_p, t]) \\ &= \psi_m(q_m 2^{-L}) + K[m+1, p, L-t] + \sum_{i=m+1}^{p} w_i t \\ &= \psi_p(q_p 2^{-L}) \end{aligned}$$

Damit erfüllt auch q_r Bedingung (2). Dies ist ein Widerspruch zur Maximalität von m.

$\diamond$

4.4.1 Der Algorithmus

Sei (k, I_h), $h = 1, \ldots, q$ und $k = 1, \ldots, n$, die Menge aller möglichen Positionen in $\bar{G}$. $T(I_h)$ sei die Menge der t, für die eine Kante $\zeta_{s,k}[I_h, t]$ in $\bar{G}$ liegt. Also $T(I_h) = \{t|$ es existiert ein $a \in \{0, \ldots, \lfloor \frac{2n}{g} \rfloor\}$ mit $I_h 2^{-L} = 1 - a2^{-t}\}$.

Zur Berechnung der $\bar{\psi}_k(I_h 2^{-L})$ und zur Konstruktion des Baumes aus dem minimal gewichteten Pfad ist die Berechnung und Speicherung verschiedener Minimierungswerte nötig.

- Für jede Position (I_h, t) wird die Ebene des Graphen gesucht, von der aus man mit einer Kante den Weg minimal zu dieser Position fortsetzen kann. Dies stellt sich formal so dar:
  Ist $t \in T(I_h)$ und k fest gewählt, dann gibt es ein $s \in \{\max\{0, k-g, k-2^{L-t}\}, \ldots, k-1\}$, so daß $\bar{\psi}_s(I_h 2^{-L} - 2^{-t}) + K[s+1, k, L-t] + \sum_{i=s+1}^{k} w_i t$ minimal ist. Dieser Wert wird $\sigma_k(I_h, t)$ genannt. Falls mehrere Lösungen existieren, wählt man die kleinste.

- Ausgehend von der Ebene $\sigma_k(I_h, t)$ wird eine Position bestimmt, so daß der fortgesetzte Weg minimal ist:
  Um $\bar{\psi}_k(I_h 2^{-L})$ zu bestimmen, muß unter allen Werten $t \in T(I_h)$ der bestimmt werden, der $\bar{\psi}_{\sigma_k(I_h,t)}(I_h 2^{-L} - 2^{-t}) + K[\sigma_k(I_h, t) + 1, k, L-t] + \sum_{i=\sigma_k(I_h,t)+1}^{k} w_i t$ minimiert. Dieser Wert wird $\delta_k^t(I_h)$ genannt. Desweiteren setzt man $\delta_k^s(I_h) := \sigma_k(I_h, \delta_k^t(I_h))$.

Der Algorithmus benötigt eine fallende Liste der Gewichte, da sich die Reihenfolge der Blätter wie bei Garey nicht ändert. Der Wert der Variablen g ist vorerst beliebig, es zeigt sich jedoch in der Komplexitätsanalyse, daß es sinnvoll ist $g = O(n)$ zu setzen. Der Algorithmus berechnet dann:

1. alle Werte $K[i, j, t]$, die zur Bestimmung der Kantengewichte nötig sind

2. für alle (k, I_h):

   (a) $\sigma_k(I_h, t)$ für alle $t \in T(I_h)$

   (b) $\delta_k^t(I_h)$ und $\delta_k^s(I_h)$

   (c) $\bar{\psi}_k(I_h 2^{-L})$.

```
PROCEDURE Larmore(v_1, ..., v_n:gewichtete Knoten,L:INTEGER,g:REAL);
VAR I, k, t:INTEGER;
BEGIN
Berechne K[i, j, k] für alle k ≤ L, 1 ≤ i ≤ j ≤ n mit j − i < g;
\bar{ψ}_0(0) := 0;
FOR k = 1, ..., n DO
    \bar{ψ}_k(0) := ∞;
END;
FOR I = I_1, ..., I_q DO
    \bar{ψ}_0(I) := ∞;
END;
FOR I = I_1, ..., I_q DO
    FOR t ∈ T(I) DO
        FOR k = 1, ..., n DO
            Finde s ∈ {max{0, k − g, k − 2^{L−t}}, ..., k − 1},
              so daß \bar{ψ}_s(I2^{−L} − 2^{−t}) + K[s + 1, k, L − t] + \sum_{i=s+1}^{k} w_i t minimal ist;
            σ_k(I, t) := s;
        END;
    END;
    Finde t ∈ T(I), so daß
      \bar{ψ}_{σ_k(I,t)}(I2^{−L} − 2^{−t}) + K[σ_k(I, t) + 1, k, L − t] + \sum_{i=σ_k(I,t)+1}^{k} w_i t minimal ist;
    δ_k^t(I) := t;
    δ_k^s(I) := σ_k(I, t);
    \bar{ψ}_k(I2^{−L}) := \bar{ψ}_{δ_k^s(I))}(I2^{−L} − 2^{−δ_k^t(I)}) + K[δ_k^s(I)) + 1, k, L − δ_k^t(I)] + \sum_{i=δ_k^s(I))+1}^{k} w_i t;
END;
END.
```

Tabelle 4.6: Algorithmus zur Methode von Larmore

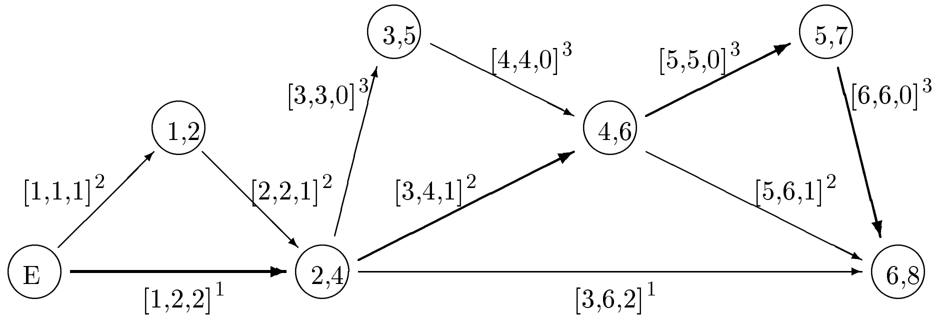


Abbildung 4.4: minimal-gewichtete Pfade zur Tabelle 4.7

Den minimal-gewichteten Pfad und die Form des Baumes erhält man folgendermaßen aus den berechneten Werten: Der Endpunkt des Pfades ist die Position $(n, 2^L)$. Gehört die Position (k, I) zum minimal-gewichteten Pfad, dann gehört auch die Kante $\zeta_{\delta_k^s,k}[I, \delta_k^t]$ und somit auch die Position $(\delta_k^s, I - 2^{L-\delta_k^t})$ dazu. Über das Kantengewicht erkennt man, daß $[\delta_k^s(I_h) + 1, k, L - \delta_k^t(I_h)]$ ein Teilbaum des gesuchten optimalen Baumes ist. Dessen Wurzel muß sich auf Ebene $\delta_k^t(I_h)$ befinden.

Die Komplexität des Algorithmus ist $O(Ln^2)$, da die Teilprobleme folgende Komplexitäten haben:

1. $O(Lng)$, wenn diese Aufgabe mit dem Algorithmus von Garey gelöst wird.

2. Die Berechnung eines $\sigma_k(I_h, t)$ für fest gewählte $k, I_h, t \in T(I_h)$ benötigt $O(g)$ Zeit. Da dies für alle Paare $I_h, t \in T(I_h)$ und k berechnet werden muß, ergibt sich für die Komplexität zur Berechnung aller $\sigma_k(I_h, t)$
$$n|(I_h, t_j)|O(g) \stackrel{4.16}{=} O(n\frac{Ln}{g}g) = O(Ln^2).$$
Zur Bestimmung von $\delta_k^t(I_h)$ müssen lediglich alle $\sigma_k(I_h, t)$ untersucht werden. Das sind $n|(I_h, t_j)| \stackrel{4.16}{=} O(Ln^2/g)$ viele. Der Algorithmus hat also in diesem Abschnitt eine Komplexität von $O(Ln^2 + \frac{Ln^2}{g})$.

Der gesamte Algorithmus hat damit eine Komplexität von $O(Lng + Ln^2 + \frac{Ln^2}{g})$. Minimierung von g in diesem Ausdruck liefert eine Komplexität von $O(Ln^2)$, wenn $g = O(n)$ ist.

4.4.2 Beispiel

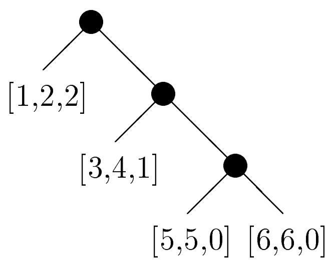
Es werden wieder die Werte von Beispiel 4.2.2 benutzt mit $g = 3$ und $0 < a < 4$. Damit sind die Positionen (k, I) in $\bar{G}$, für die $I = 2, 4, 5, 6, 7, 8$ und

I,k	2	4	5	6	7	8
$\bar{\psi}_1(I2^{-L})$	48	24				0
(δ_1^s, δ_1^t)	(0,2)	(0,1)				(0,0)
$\bar{\psi}_2(I2^{-L})$	108	**72**		48		36
(δ_2^s, δ_2^t)	(0,2)	**(0,1)**,(1,2)		(1,2)		(0,0),(1,1)
$\bar{\psi}_3(I2^{-L})$		102	90	78	66	60
(δ_3^s, δ_3^t)		(0,1),(1,2)	(2,3)	(1,2)	(2,3)	(0,0),(1,1),(2,2)
$\bar{\psi}_4(I2^{-L})$		135	111	**99**	87	75
(δ_4^s, δ_4^t)		(0,1),(2,2)	(3,3)	**(2,2)**,(3,3)	(3,3)	(0,0),(1,1),(2,2),(3,3)
$\bar{\psi}_5(I2^{-L})$			138	114	**102**	90
(δ_5^s, δ_5^t)			(4,3)	(3,2),(4,3)	**(4,3)**	(1,1),(3,2),(4,3)
$\bar{\psi}_6(I2^{-L})$				141	117	**105**
(δ_6^s, δ_6^t)				(4,2),(4,3)	(5,3)	(2,1),(4,2),**(5,3)**

Tabelle 4.7: Werte für den 3-beschränkten Baum zum Beispiel 4.4.2

$k = 1, 2, 3, 4, 5, 6$ ist. Man erhält außerdem $T(2) = \{2\}; T(4) = \{1, 2\}; T(5) = \{3\}; T(6) = \{2, 3\}; T(7) = \{3\}; T(8) = \{0, 1, 2, 3\}$. Der Algorithmus liefert die Werte von Tabelle 4.7.

Hier existieren mehrere minimal-gewichtete Pfade. Die Kanten, die man auswählen kann, und die verschiedenen Pfade sind in Abbildung 4.4 dargestellt. An den Kanten wurden die Bäume notiert, die man „verkleben" muß, um den gesuchten optimalen Baum zu erhalten. Der Index gibt dabei an, auf welcher Ebene sich die Wurzel befinden muß.



Der konstruierte Baum zu den **fett** markierten Einträgen in Tabelle 4.7 und zum **fett** markierten Pfad in Abbildung 4.4 ist rechts zu sehen.

4.4.3 Verbesserung der Komplexität

Auch bei diesem Algorithmus kann man die Anzahl der zu untersuchenden Werte bei einer Minimierung einschränken. Es wird gezeigt, daß bei der Bestimmung von $\sigma_k(I_h, t)$ nur $O(n \log_2 n)$ viele Werte betrachtet werden müssen.

Satz 4.19 *Sei I, t gegeben mit $I2^{-L} = 1 - a2^{-t}$ und $r < s$. Dann gilt*

$$\sigma_r(I, t) \leq \sigma_s(I, t)$$

Beweis 4.19 Sei $R := \sigma_r(I,t)$ und $S := \sigma_s(I,t)$. Da R und S die zugehörigen Ausdrücke minimieren, gilt

$$\begin{aligned} & \bar{\psi}_R(I2^{-L} - 2^{-t}) + K[R+1, r, L-t] + \sum_{i=R+1}^{r} w_i t \\ < \; & \bar{\psi}_S(I2^{-L} - 2^{-t}) + K[S+1, r, L-t] + \sum_{i=S+1}^{r} w_i t \end{aligned}$$

und

$$\begin{aligned} & \bar{\psi}_S(I2^{-L} - 2^{-t}) + K[S+1, s, L-t] + \sum_{i=S+1}^{s} w_i t \\ \leq \; & \bar{\psi}_R(I2^{-L} - 2^{-t}) + K[R+1, s, L-t] + \sum_{i=R+1}^{s} w_i t. \end{aligned}$$

Daraus folgt

$$K[R+1, r, t] + K[S+1, s, t] < K[S+1, r, t] + K[R+1, s, t].$$

Ist $R > S$, dann gilt wegen Satz 3.9 auch

$$K[S+1, r, t] + K[R+1, s, t] \leq K[S+1, s, t] + K[R+1, r, t].$$

Dies beides steht im Widerspruch und somit ist $R \leq S$. $\diamond$

Zur Verminderung der Komplexität müssen die Werte von k nach folgendem Schema anwachsen:

FOR $p := \lfloor \log_2 n \rfloor$ TO 0 DO
  FOR $u := 1$ TO $\lfloor n2^{-p-1} + 1/2 \rfloor$ DO
    $k := (2u-1)2^p$;
    $\vdots$
  END;
END;

Da $(u-1)2^{p+1} < (2u-1)2^p < u2^{p+1}$, müssen beim Berechnen von $\sigma_{(2u-1)2^p}(I_h, t)$ nur Werte zwischen $\sigma_{(u-1)2^{p+1}}(I_h, t)$ und $\sigma_{u2^{p+1}}(I_h, t)$ zur Minimierung herangezogen werden. Damit erhält man für die Berechnung der

$\sigma_k(I_h, t)$ bei festem I_h, t eine Komplexität von

$$\begin{aligned}\sum_{p=0}^{\lfloor \log_2 n \rfloor} \sum_{u=1}^{\lfloor n2^{-p-1}+1/2 \rfloor} \sigma_{u2^{p+1}}(I_h, t) - \sigma_{(u-1)2^{p+1}}(I_h, t) + 1 &= \\ \sum_{p=0}^{\lfloor \log_2 n \rfloor} \sigma_{n+2^p}(I_h, t) - \sigma_0(I_h, t) + \lfloor n2^{-p-1} + 1/2 \rfloor &= \\ \sum_{p=0}^{\lfloor \log_2 n \rfloor} n + \lfloor n2^{-p-1} + 1/2 \rfloor &= O(n \log_2 n)\end{aligned}$$

Für die Berechnung aller $\sigma_k(I_h, t)$ erhält man damit eine Komplexität von

$$|(I_h, t_j)| O(n \log_2 n) \overset{4.16}{=} O(\frac{Ln}{g} n \log_2 n) = O(\frac{Ln^2 \log_2 n}{g}).$$

Daraus ergibt sich für den kompletten Algorithmus eine Komplexität von $O(Lng + \frac{Ln^2 \log_2 n}{g} + \frac{Ln^2}{g}) = O(Lng + \frac{Ln^2 \log_2 n}{g})$. Minimierung diese Ausdrucks über g liefert eine Komplexität von $O(Ln^{\frac{3}{2}} \log_2^{\frac{1}{2}} n)$ für $g = O(n^{\frac{1}{2}} \log_2^{\frac{1}{2}} n)$

4.5 Das Verfahren von Larmore und Hirschberg

Larmore und Hirschberg führen den Begriff der Knotenmenge eines Baumes ein. Auf diesen Mengen suchen sie mit dem Münzsammler(coin collector)-Algorithmus eine geeignete Lösung.

Definition 4.20 *Seien n Werte $w_1, \ldots, w_n$ gegeben.*
Ein Knoten *ist ein geordnetes Paar (i, l) mit* Index $i = 1, \ldots, n$ *und* Ebene $l = 1, \ldots, L$. *Jedem Knoten werden zwei Werte zugewiesen, die* Breite $b(i, l) = 2^{-l}$ *und das* Gewicht $g(i, l) = w_i$.
Eine Knotenmenge N *ist die Vereinigung verschiedener Knoten. Die* Breite einer Knotenmenge *ist* $b(N) = \sum_{x \in N} b(x)$, *das* Gewicht einer Knotenmenge *ist* $g(N) = \sum_{x \in N} g(x)$.
Die Knotenmenge eines Baumes $\mathbf{B}$ *mit n Blättern der Tiefen $l_1, \ldots, l_n$ und Gewichten $w_1, \ldots, w_n$ ist eine Knotenmenge $N_{\mathbf{B}}$ mit allen Paaren (i, l), $(i = 1, \ldots, n,\ l = 1, \ldots, l_i)$.*

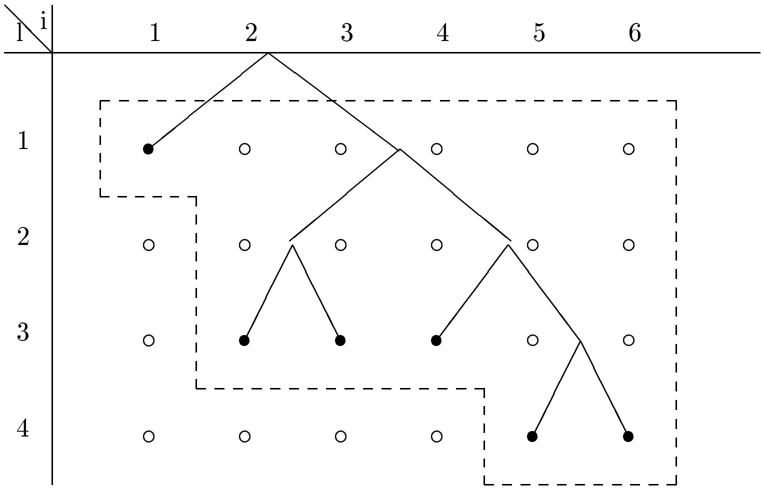


Abbildung 4.5: Knotenmenge eines Baumes

Bemerkung 4.21

1. Der graphische Zusammenhang wird in Abbildung 4.5 verdeutlicht. Die Knotenmenge $N_{\mathbf{B}}$ zum angegebenen Baum besteht aus den Knoten (1,1), (2,1),(2,2),(2,3), (3,1),(3,2),(3,3), (4,1),(4,2),(4,3), (5,1),...,(5,4), (6,1),...,(6,4).

2. Ist die Knotenmenge eines Baumes bekannt, dann sind die Blattlängen bestimmt durch:
$$l_i = \max_{(i,j)\in N_{\mathbf{B}}} j$$

Die Breite der Knotenmenge eines Baumes läßt sich folgendermaßen berechnen.

Satz 4.22 *Seien $l_1, \ldots, l_n$ gegeben und N eine Knotenmenge mit allen Paaren (i, l), $(i = 1, \ldots, n,\ l = 1, \ldots, l_i)$. Dann gilt*

$$b(N) = n - \sum_{i=1}^{n} 2^{-l_i}.$$

Beweis 4.22

$$b(N) = \sum_{i=1}^{n} \sum_{j=1}^{l_i} b(i, j) = \sum_{i=1}^{n} \sum_{j=1}^{l_i} 2^{-j} = \sum_{i=1}^{n} (1 - 2^{-l_i}) = n - \sum_{i=1}^{n} 2^{-l_i}.$$

◇

Satz 4.23 *Sei* $\mathbf{B}$ *ein regulärer Baum mit* n *gewichteten Blättern. Dann gilt:*

- $g(N_{\mathbf{B}}) = \sum_{i=1}^{n} w_i l_i$
- $b(N_{\mathbf{B}}) = n - 1.$

Beweis 4.23

- $$g(N_{\mathbf{B}}) = \sum_{i=1}^{n}\sum_{j=1}^{l_i} g(i,j) = \sum_{i=1}^{n}\sum_{j=1}^{l_i} w_i = \sum_{i=1}^{n} l_i w_i.$$
- Folgt aus Satz 4.22 und der Kraftschen Ungleichung 1.24.

$\diamond$

Die Breite einer Knotenmenge bestimmt die Existenz von Teilmengen.

Satz 4.24 *Sei* N *eine Knotenmenge mit* $b(N) = I2^{-l} + r$ *und* $0 < r < 2^{-l}$. *Dann existiert eine Knotenmenge* $M \subseteq N$ *mit* $b(M) = r$.

Beweis 4.24 Beweis mit Induktion über die Größe der Knotenmenge N.
Ist $|N| = 1$, dann ist $b(N) = r$ und man setzt $M = N$.
Sei der Satz für alle Knotenmengen $\bar{N}$ mit $|\bar{N}| < |N|$ bereits bewiesen und u sei ein Knoten kleinster Breite 2^{-k}.
Dann gilt für alle anderen Knoten v: $b(v) = i2^{-k}$ mit $i \in \mathbf{N}$. Somit gilt $b(N) = \bar{i}2^{-k}$ mit $\bar{i} \in \mathbf{N}$ und daraus folgt $k > l$ (sonst wäre $b(N) = \bar{j}2^{-l}$ mit $\bar{j} \in \mathbf{N}$) und $r = j2^{-k} \geq 2^{-k}$.

- Ist $r = 2^{-k}$, dann setzt man $M = \{u\}$.
- Ist $r > 2^{-k}$, dann betrachte man die Menge $\bar{N} = N \backslash \{u\}$ mit $b(\bar{N}) = I2^{-l} + r - 2^{-k}$. Zu dieser existiert aufgrund der Induktionsvoraussetzung eine Menge $\bar{M}$ mit $b(\bar{M}) = r - 2^{-k}$ und man setzt $M = \bar{M} \cup \{u\}$.

$\diamond$

Definition 4.25 *Eine Knotenmenge* N *heißt* monoton, *wenn die beiden folgenden Bedingungen erfüllt sind:*

1. *Für alle* $i \neq n$ *gilt: ist* (i,l) *in* N *enthalten, dann auch* $(i+1,l)$.
2. *Für alle* $l \neq 1$ *gilt: ist* (i,l) *in* N *enthalten, dann auch* $(i,l-1)$.

Eine Klasse monotoner Knotenmengen wird im nächsten Satz charakterisiert.

Satz 4.26 *Sei $w_1 > \ldots > w_n$, $n > x \in \mathbf{N}$ und N eine Knotenmenge minimalen Gewichts mit Breite x. Dann ist N monoton.*

Beweis 4.26

- Ist das erste Monotonie-Kriterium nicht erfüllt, dann gibt es ein Paar $(i,l) \in N$ mit $(i+1,l) \notin N$. Sei $\bar{N} = N \backslash \{(i,l)\} \cup \{(i+1,l)\}$, dann gilt $b(\bar{N}) = x$ und $g(\bar{N}) < g(N)$. Dies ist ein Widerspruch zur Forderung, daß N minimales Gewicht hat.

- Ist das zweite Monotonie-Kriterium nicht erfüllt, dann gibt es ein Paar $(i,l) \in N$ mit $(i,l-1) \notin N$. Sei $\bar{N} = N \backslash \{(i,l)\} \cup \{(i,l-1)\}$, dann ist $g(\bar{N}) = g(N)$ und $b(\bar{N}) = x + 2^{-l}$. Aufgrund von Satz 4.24 existiert eine Knotenmenge M mit $b(M) = 2^{-l}$. Die Knotenmenge $\bar{N} \backslash M$ hat dann die Breite x und ein kleineres Gewicht als N. Dies ist widersprüchlich zur Forderung nach minimalem Gewicht für N.

$\diamond$

Der folgende Satz erläutert den Zusammenhang zwischen Knotenmengen und optimalen Bäumen. Er ist der Schlüssel zur algorithmischen Lösung des Problems.

Satz 4.27 *Sei $w_1 > \ldots > w_n$ und N eine Knotenmenge minimalen Gewichts mit $b(N) = n - 1$. Dann ist N Knotenmenge eines optimalen Baumes mit eingeschränkter Pfadlänge L.*

Beweis 4.27 Aufgrund von Satz 4.26 ist N monoton. Sei $l_i = \max_{(i,j) \in N} j$, dann gilt aufgrund der Monotonie von N

$$l_i \leq l_{i+1}.$$

Aufgrund von Satz 4.22 und $b(N) = n - 1$ gilt

$$\sum_{i=1}^{n} 2^{-l_i} = 1.$$

Dann geben die l_i wegen der Kraftschen Ungleichung 1.24 die Pfadlängen eines Baumes zu den Gewichten w_i an. Also ist N Knotenmenge eines Baumes mit $l_i \leq L$.
Wäre dieser Baum nicht optimal, dann gäbe es einen Baum $\bar{\mathbf{B}}$, mit $b(N_{\bar{\mathbf{B}}}) = n - 1$ und $g(N_{\bar{\mathbf{B}}}) < g(N)$. Dies ist ein Widerspruch zur Forderung, daß $g(N)$ minimal ist. $\diamond$

4.5.1 Das Münzsammlerproblem

Aufgrund von Satz 4.27 ist zum Finden eines optimalen L-beschränkten Baumes folgendes Problem zu lösen: Finde in der Menge aller Knoten eine Teilmenge mit minimalem Gewicht und Breite $n-1$.

Dieses Problem läßt sich auf das *Münzsammlerproblem* zurückführen: Gegeben ist eine m-elementige Menge I (die Menge der Münzen/Knoten). Jedem dieser Elemente sind zwei Werte zugewiesen: das Gewicht (der Münzwert aus $\mathbf{R}$) und die Breite (der aufgedruckte Zahlenwert; dieser soll eine 2-er Potenz sein, da es Münzen des Landes Binar sind). Es soll eine Auswahl $S(I,X)$ der Elemente getroffen werden, so daß sich die Zahlenwerte zu einem geforderten, vom Münzsammler zu bezahlenden Wert X addieren, der Münzwert (den nur der Münzsammler kennt) aber minimal ist.

Zur Lösung dieses Problems ist die binäre Darstellung des zu zahlenden Betrags X notwendig, das heißt $X = \sum_{i=j}^{p} t_i 2^i$ mit $t_i \in \{0,1\}$.

Das Problem läßt sich rekursiv betrachten:

Satz 4.28 *Sei r die kleinste Breite eines Elementes in I, $\mathcal{R}$ die Menge der minimal gewichteten Elemente aus I mit Breite r und $X = 2^j + \sum_{i>j}^{p} t_i 2^i$ oder $X = 0$.*

- *Ist $X = 0$, dann ist $S(I,X) = \emptyset$.*
  Ist $X > 0$ und $I = \emptyset$, dann existiert keine Lösung.
- *Sei $X > 0$ und $I \neq \emptyset$:*
  - *Ist $r > j$, dann existiert keine Lösung.*
  - *Ist $r = j$, dann ist $S(I,X) = S(I\backslash\{a\}, X - r) \cup \{a\}$ für ein beliebiges $a \in \mathcal{R}$.*
  - *Ist $r < j$ und $\mathcal{R} = \{a\}$, dann ist $S(I,X) = S(I\backslash\{a\}, X)$.*
  - *Falls $r < j$ ist und $\{a,b\} \subseteq \mathcal{R}$, dann wird ein Element $x := (a,b)$ mit $b(x) = 2r$ und $g(x) = g(a) + g(b)$ konstruiert.*
    Ist $x \in S(I\backslash\{a,b\} \cup \{x\}, X)$, dann ist $S(I,X) = S(I\backslash\{a,b\} \cup \{x\}, X)\backslash\{x\} \cup \{a,b\}$.
    Ansonsten ist $S(I,X) = S(I\backslash\{a,b\} \cup \{x\}, X)$.

Beweis 4.28 Die Rekursionsverankerung ist trivial.

- Ist $r > j$, dann gibt es keine Möglichkeit, den Betrag 2^j für X beizusteuern, da r die minimale Breite ist.
- Ist $r = j$, dann muß ein Element mit minimalem Gewicht und Breite r in der Lösungsmenge liegen.

- Ist $r < j$ und $\mathcal{R} = \{a\}$, dann kann das Element a nicht in der Lösungsmenge liegen.
- Ist $r < j$ und $\{a, b\} \subseteq \mathcal{R}$, dann liegen entweder beide Elemente in der Lösung oder keines.

$\diamond$

4.5.2 Der Algorithmus und seine Komplexität

Der Münzsammleralgorithmus

Der Algorithmus initialisiert zu Beginn Listen L_d, in denen alle Elemente mit Breite 2^d nach Gewicht sortiert abgelegt werden. Für $r = j$ und $r > j$ führt der Algorithmus die Anweisung der Rekursion direkt aus. Der Fall $r < j$ wird indirekt am Ende der WHILE-Schleife ausgeführt: je zwei Elemente aus der aktuellen Liste L_d werden verschmolzen und sortiert in die nächste aktuelle Liste L_{d+1} eingefügt.

Der Münzsammler-Algorithmus hat lineare Laufzeit in der Mächtigkeit der Menge I. Diese Linearität zeigt man folgendermaßen: Das Verschmelzen zweier Listen (Merge) ist linear in der Summe der Längen beider Listen. Package ist linear in der Länge der Ergebnisliste. Weist man jedem Paar (i, l) aus I ein Guthaben von 3 Einheiten zu, dann kann man die Komplexität folgendermaßen bestimmen: Jedem Element der zu verschmelzenden Listen oder der gepackten Liste wird eine Einheit des Guthabens abgezogen, was die Komplexität dieser Operationen widerspiegelt. Dann gilt folgende Invariante.

Satz 4.29 (Invariante)

1. *Jedes Element einer Liste L_d hat ein Guthaben von mindestens 2 Einheiten.*
2. *Jedes Element einer Liste P_d hat ein Guthaben von mindestens 3 Einheiten.*

Beweis 4.29

- Ist L_d eine Liste, die im Algorithmus noch nicht verändert wurde, dann hat jedes Element ein Guthaben von 3 Einheiten, da so initialisiert wurde.
  Entsteht L_d durch Verschmelzen zweier Listen, dann handelt es sich stets um eine unveränderte Liste L_d mit 3 Einheiten Guthaben pro Element und eine Liste P_d, die aufgrund von (2) nur Elemente mit einem

```
Münzsammler(I:Münzmenge,X:INTEGER):Münzmenge;
VAR d, j, r:INTEGER;
      L_d, P_d:Listen;
      S(I, X):Münzmenge;
BEGIN
S(I, X) := ∅;
Bestimme j in der Darstellung X = 2^j + Σ_{i>j}^p t_i 2^i;
IFin I existiert ein Element der Breite 2^d
  THEN L_d := (x_1, ..., x_k) mit x_i ∈ I, b(x_i) = 2^d, g(x_1) ≤ ... ≤ g(x_k);
END;
WHILE X > 0 DO
    IF I = ∅ THEN RETURN ∅;
    ELSE
       d := min_{L_d ≠ ∅} d;
       r := 2^d;
       IF r > j THEN RETURN ∅;
       ELSIF r = j THEN
          L_d := L_d \ {x_1};
          S(I, X) := S(I, X) ∪ {x_1};
          X := X − r;
          Bestimme j in der Darstellung X = 2^j + Σ_{i>j}^p t_i 2^i;
       END;
    END;
    P_{d+1} :=Package(L_d);
    L_d := ∅;
    L_{d+1} :=Merge(L_{d+1}, P_{d+1});
END;
RETURN S(I, X);
END.
```

$(y_1, \ldots, y_{\lfloor \frac{k}{2} \rfloor})$:=Package$(x_1, \ldots, x_k)$ mit $g(y_i) := g(x_{2i-1}) + g(x_{2i})$ und $b(y_i) := b(x_{2i-1}) + b(x_{2i})$.
Merge ist das bekannte Verschmelzen und Sortieren von Listen nach Gewicht.

Tabelle 4.8: Münzsammler-Algorithmus

Guthaben von mindestens 3 Einheiten besitzt. Nach dem Verschmelzen und Bezahlen hat jedes Element der Liste deshalb mindestens ein Guthaben von $3 - 1 = 2$ Einheiten.

- Jedes Element einer Liste L_d hat aufgrund von (1) ein Mindestguthaben von 2 Einheiten. Dann hat jedes Element in P_d mehr als $2 + 2 - 1 = 3$ Einheiten Guthaben.

$\diamond$

Ist der Algorithmus beendet, dann besitzt jedes Element ein positives Guthaben. Das bedeutet, daß jedes Element der Liste linear oft betrachtet wurde. Deshalb besitzt der Algorithmus die Komplexität $O(|I|)$.

Die Anwendung des Algorithmus für höhenbeschränkte Bäume

Um einen optimalen L-beschränkten Baum zu n nach Gewicht sortierten Blättern mithilfe des Münzsammler-Algorithmus zu finden, ruft man ihn mit $X = n - 1$ und $I = \{(i,l)|i = 1,\ldots,n, l = 1,\ldots,L\}$ auf. Der Algorithmus liefert dann die Knotenmenge des gesuchten Baumes. Der Algorithmus besitzt die Komplexität $O(nL)$, falls die Blätter sortiert vorliegen und zum Bestimmen des Baumes aus der Knotenmenge ist eine lineare Laufzeit in nL nötig.

4.5.3 Beispiel

Seien die Werte des Beispiels wie bisher üblich gewählt mit $L = 4$ und den Wahrscheinlichkeiten 24,12,6,3,1,1. Dann liefert der Algorithmus folgende Variablenbelegungen:

- Initialisierung :
  $X = 2^0 + 2^2$,
  $S = \emptyset$,
  $j = 0$,
  $L_{-1} = \{(6,1),(5,1),(4,1),(3,1),(2,1),(1,1)\}$,
  $L_{-2} = \{(6,2),(5,2),(4,2),(3,2),(2,2),(1,2)\}$,
  $L_{-3} = \{(6,3),(5,3),(4,3),(3,3),(2,3),(1,3)\}$,
  $L_{-4} = \{(6,4),(5,4),(4,4),(3,4),(2,4),(1,4)\}$.

- 1.Durchlauf:
  $r = 2^{-4}$,

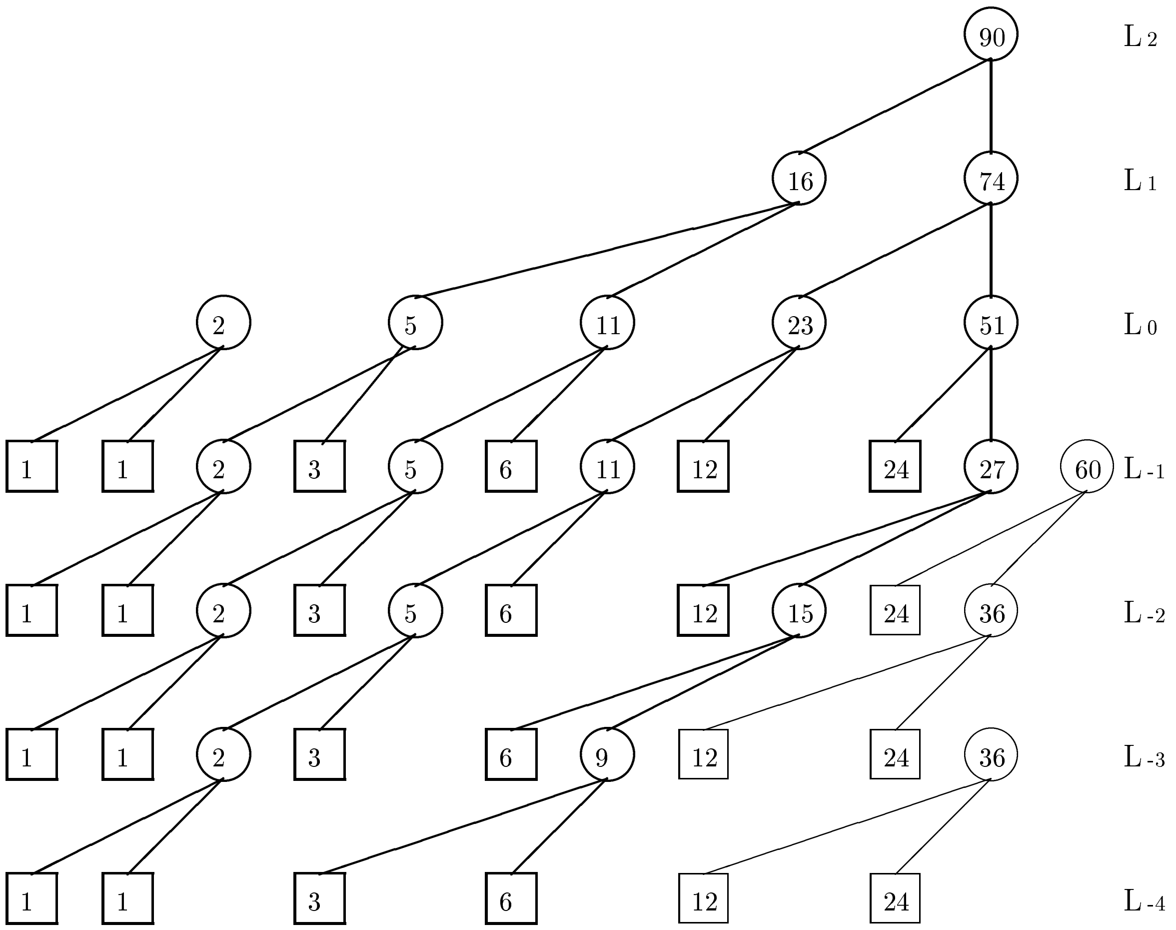


Abbildung 4.6: graphische Veranschaulichung der Listenmanipulation zum Beispiel 4.5.3

$$\begin{aligned}
L_{-1} &= \{(6,1),(5,1),(4,1),(3,1),(2,1),(1,1)\},\\
L_{-2} &= \{(6,2),(5,2),(4,2),(3,2),(2,2),(1,2)\},\\
L_{-3} &= \{(6,3),(5,3),[(6,4),(5,4)],(4,3),(3,3),[(4,4),(3,4)],\\
&\quad (2,3),(1,3),[(2,4),(1,4)]\}.
\end{aligned}$$

- 2.Durchlauf:
  $$\begin{aligned}
  r &= 2^{-3},\\
  L_{-1} &= \{(6,1),(5,1),(4,1),(3,1),(2,1),(1,1)\},\\
  L_{-2} &= \{(6,2),(5,2),[(6,3),(5,3)],(4,2),[(6,4),(5,4),(4,3)],(3,2),\\
  &\quad (2,2),[(3,3),(4,4),(3,4)],(1,2),[(2,3),(1,3)]\}.
  \end{aligned}$$

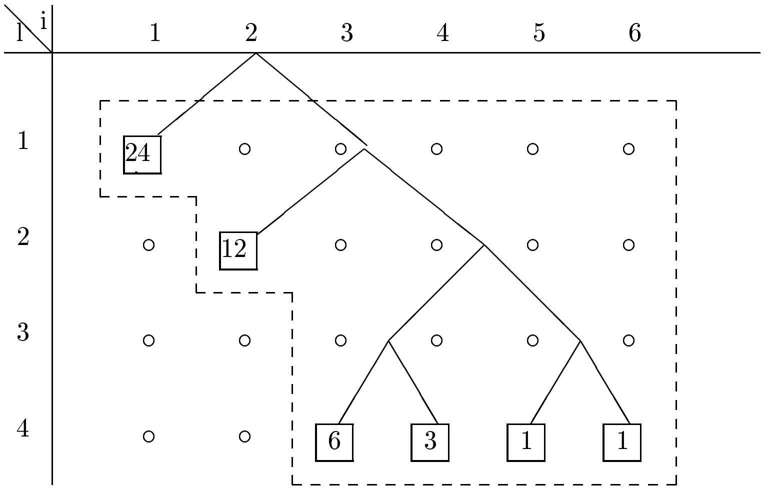


Abbildung 4.7: Baum zum Beispiel 4.5.3 und seine berechnete Knotenmenge

- 3.Durchlauf:
  $r \quad = 2^{-2}$,
  $L_{-1} = \{(6,1),(5,1),[(6,2),(5,2)],(4,1),[(6,3),(5,3),(4,2)],(3,1),$
  $[(6,4),(5,4),(4,3),(3,2)],(2,1),(1,1),[(2,2),(3,3),(4,4),(3,4)],$
  $[(1,2),(2,3),(1,3)]\}$.

- 4.Durchlauf:
  $r \quad = 2^{-1}$,
  $L_0 = \{[(6,1),(5,1)],[(6,2),(5,2),(4,1)],[(6,3),(5,3),(4,2),(3,1)],$
  $[(6,4),(5,4),(4,3),(3,2),(2,1)],[(1,1),(2,2),(3,3),(4,4),(3,4)]\}$.

- 5.Durchlauf:
  $r \quad = 2^0$,
  $S \quad = \{(6,1),(5,1)\}$,
  $X \quad = 2^2$,
  $j \quad = 2$,
  $L_1 = \{[(6,2),(5,2),(4,1),(6,3),(5,3),(4,2),(3,1)],$
  $[(6,4),(5,4),(4,3),(3,2),(2,1),(1,1),(2,2),(3,3),(4,4),(3,4)]\}$.

- 6.Durchlauf:
  $r \quad = 2^1$,
  $L_2 = \{[(6,2),(5,2),(4,1),(6,3),(5,3),(4,2),(3,1),$
  $(6,4),(5,4),(4,3),(3,2),(2,1),(1,1),(2,2),(3,3),(4,4),(3,4)]\}$.

- 7.Durchlauf:
  $r \quad = 2^2,$
  $S \quad = \{(1,1),(2,1),(2,2),(3,1),(3,2),(3,3),(3,4),(4,1),(4,2),$
  $\qquad (4,3),(4,4),(5,1),(5,2),(5,3),(5,4),(6,1),(6,2),(6,3),(6,4)\},$
  $X \quad = 0,$

Die Manipulation der Listen L_d läßt sich graphisch veranschaulichen. Dabei werden die initialisierten Listen übereinander aufgetragen und die zusammengefaßten Elemente sortiert in die Listen eingefügt. Die ausgewählten Elemente werden dabei **fett** markiert.

Die l_i bestimmen sich nach Bemerkung 4.21.2 zu $l_1 = 1$, $l_2 = 2$, $l_3 = 4$, $l_4 = 4$, $l_5 = 4$, $l_6 = 4$. Der berechnete Baum und seine Knotenmenge sind in Abbildung 4.5.3 dargestellt.

Kapitel 5

Optimale alphabetische Bäume und Codes mit eingeschränkter Länge

Dieses Kapitel führt die Problemstellungen der beiden vorhergehenden Kapitel zusammen. Es werden optimale Bäume zu einer vorgegebenen Blattordnung gesucht, bei denen kein Blatt mehr als L Vorfahren besitzt.

Teilweise wurde dieses Problem in Kapitel 4 angesprochen, denn die Algorithmen, bei denen keine Sortierung der Blätter nach Gewicht nötig ist, liefern einen alphabetischen Baum.

Zur Erinnerung: Hu und Tan [10] erstellten in exponentieller Zeit einen höhen-beschränkten alphabetischen Baum. Garey [6] konstruierte einen höhenbeschränkten alphabetischen Baum in $O(Ln^3)$, Wessner [25] zeigte, daß dies auch mit der Komplexität $O(Ln^2)$ möglich ist.

In diesem Kapitel wird der Algorithmus von Larmore und Przytycka [19] vorgestellt. Dieser verwendet Kenntnisse aus dem Algorithmus von Larmore und Hirschberg [18] und dem Algorithmus von Hu und Tucker [11] und hat eine Komplexität von $O(Ln \log_2 n)$.

5.1 Beispiel

Seien die Gewichte 24,1,1,3,12,6 gegeben. Dann zeigt Abbildung 5.1 folgende Bäume zu diesen Gewichten (siehe auch Abbildung 5.1).

1. optimaler Baum, Kosten 88
2. optimaler 4-beschränkter Baum, Kosten 92
3. optimaler 3-beschränkter Baum, Kosten 105

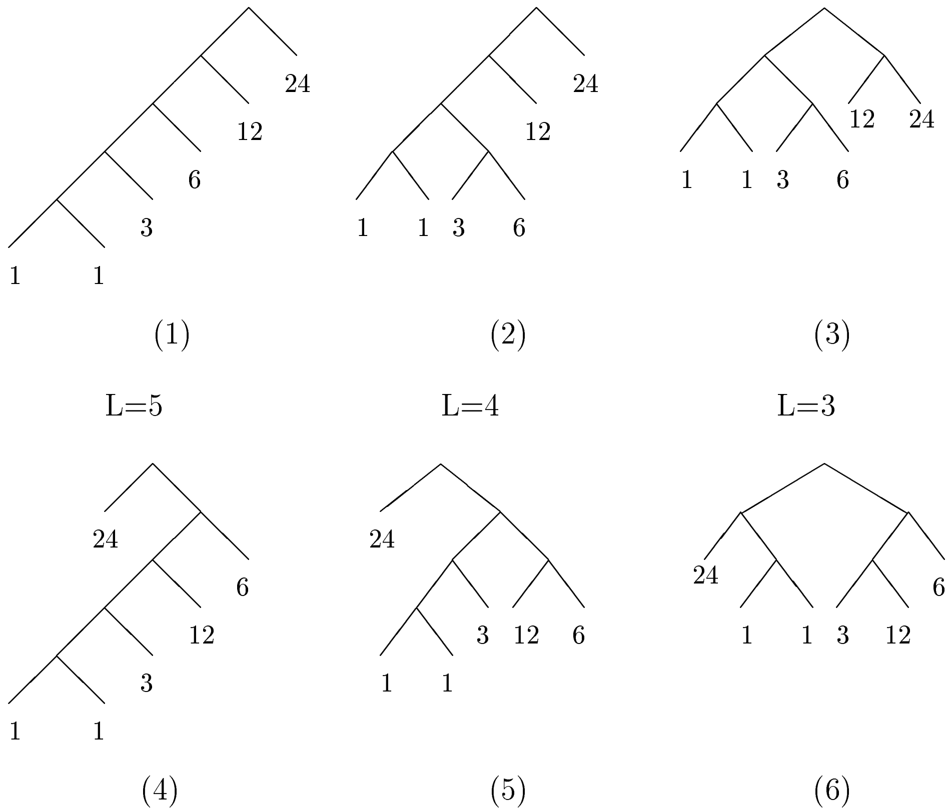


Abbildung 5.1: optimale und optimale alphabetische Bäume

4. optimaler alphabetischer Baum, Kosten 94
5. optimaler alphabetischer 4-beschränkter Baum, Kosten 95
6. optimaler alphabetischer 3-beschränkter Baum, Kosten 111

5.2 Das Verfahren von Larmore und Przytycka

Diese Methode ist die alphabetische Variante des Algorithmus von Larmore und Hirschberg [18] aus Kapitel 4.5. Die Beschreibung des Algorithmus ist ähnlich einfach, der Korrektheitsbeweis jedoch sehr lang, so daß auf Larmore und Przytycka [19] verwiesen wird.

Im Folgenden wird die Definition und Notation von Knoten aus Kapitel 4.5 benutzt. Desweiteren wird der Begriff des Paketes eingeführt. Pakete wurden schon in Kapitel 4.5 benutzt, jedoch nicht so bezeichnet. Sie wurden dort durch die Package-Operation erzeugt.

Definition 5.1 *Ein Paket ist ein Verband, der aus mindestens zwei Knoten besteht.* Breite/Gewicht eines Packetes *berechnen sich aus der Summe der Breiten/Gewichte seiner Knoten. Der Index eines Paketes ist der kleinste Knoten-Index im Verband.*

Wie bei Larmore und Hirschberg wird in der Liste der Elemente kleinster Breite das minimal gewichtete Paar zu einem Paket zusammengefaßt und in die Liste nächstgrößerer Breite eingefügt. Um alphabetische Bäume zu konstruieren, müssen die Elemente in den Listen sortiert vorliegen und es dürfen nur Paare mit besonderen Eigenschaften zusammengefaßt werden. Diese Eigenschaft ist identisch mit der T.C.Bedingung von Hu und Tucker aus Kapitel 3.4.1.

Definition 5.2 *Ist F_d eine Liste, bestehend aus Knoten und Paketen mit Breite 2^d, dann ist diese Liste so nach aufsteigendem Index geordnet, daß sich Knoten vor den Paketen mit gleichem Index befinden.*
Zwei Elemente aus dieser Liste sind tentative-connecting *oder* T.C., *wenn sich kein Knoten zwischen ihnen befindet. Eine Huffman-Menge ist eine maximal große Menge von Elementen einer Liste, so daß je zwei Paare dieser Menge T.C. sind.*

Beispiel 5.3 Eine Liste F_{-1} könnte beispielsweise so aussehen:

$$(1,1),(2,1),[(2,2),(3,2)],(3,1),[[(3,3),(4,3)],(4,2)],(4,1).$$

Die entsprechenden Huffman-Mengen sind

$$\{(1,1),(2,1)\},\{(2,1),[(2,2),(3,2)],(3,1)\},\{(3,1),[[(3,3),(4,3)],(4,2)],(4,1)\}.$$

5.2.1 Der Algorithmus und seine Komplexität

Beginnend bei der Liste mit kleinstem Gewicht werden minimal gewichtete T.C.Knoten zusammengefaßt und in die Liste nächst größerer Breite eingefügt. Die Liste F_0 liefert dann die Knotenmenge des zu berechnenden Baumes wie in Kapitel 4.5. Aus dieser ist der gesuchte Baum in linearer Zeit bezüglich der Größe der Knotenmenge bestimmbar.

Der Algorithmus hat die Komplexität $O(nL\log_2 n)$, falls man die Huffman-Mengen der Listen F_d wie im T.C.Algorithmus 3.3 in Kapitel 3.4.3 in Binominalhalden organisiert: Die Listen F_d enthalten stets weniger als $2n-1$

```
PROCEDURE Larmore_Przytycka(v_1, ..., v_n:gewichtete Knoten,L:INTEGER):
      Knotenmenge;
VAR d:INTEGER;
       x:Paket;
       F_d:Liste;
BEGIN
FOR d := 1 TO L DO
    F_{-d} := ((1,d), ..., (n,d)) mit g(i,d) = w_i;
END;
F_0 := ∅;
FOR d := -L TO -1 DO
    WHILE |F_d| ≥ 2 DO
         Finde T.C.Knoten u und v mit minimalem Gewicht in F_d;
         x := [u,v];
         F_d := F_d\{u,v};
         Einfügen(F_{d+1}, x);
    END;
END;
RETURN F_0;
END.
```

Einfügen(F_{d+1}, x) fügt das Paket x in die Liste F_{d+1} nach Index sortiert ein.

Tabelle 5.1: Algorithmus zur Methode von Larmore und Przytycka

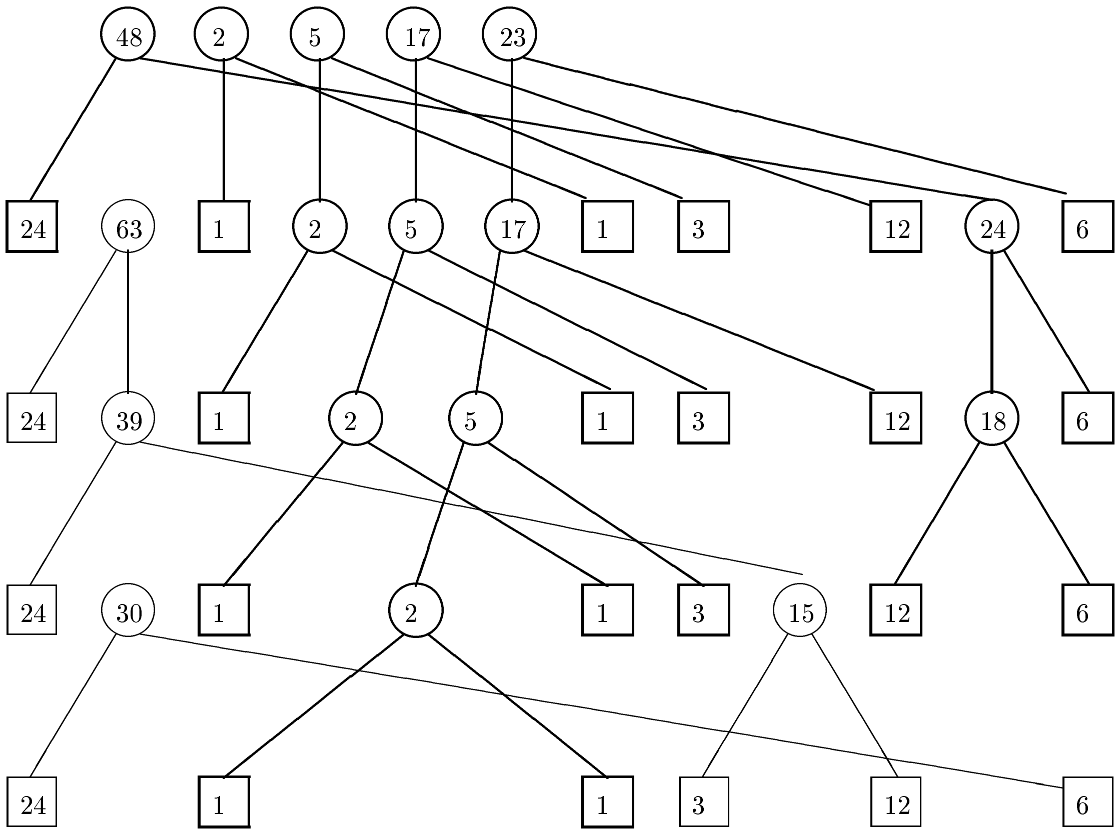


Abbildung 5.2: graphische Veranschaulichung der Listenmanipulation

Pakete und Knoten, da für die Anzahl der Elememte jeder Liste gilt

$$|F_d| \leq \sum_{i=0}^{d+L} \frac{n}{2^i} = 2n - \frac{n}{2^{d+L}}$$

und $n \leq 2^L < 2^{L+d}$ ist. Das Finden und Löschen eines T.C.Paares in einer beliebigen Liste hat damit die Komplexität $O(\log_2 n)$. Da maximal $L(2n-1)$ Elemente bearbeitet werden müssen, erhält man die oben genannte Komplexität.

5.2.2 Beispiel

Seien die Werte des Beispiels wie bisher üblich gewählt mit $L = 4$ und den Wahrscheinlichkeiten 24,1,1,3,12,6. Auch hier kann man die Manipulationen des Algorithmus graphisch darstellen (siehe Abbildung 5.2). Desweiteren wird angegeben, in welcher Reihenfolge der Algorithmus die Paare in jeder Liste zusammenfaßt.

- In der 4.ten Ebene
  - $(2,4),(3,4)$
  - $(4,4),(5,4)$
  - $(1,4),(6,4)$
- In der 3.ten Ebene
  - $(2,3),(3,3)$
  - $[(2,4),(3,4)],(4,3)$
  - $(5,3),(6,3)$
  - $(1,3),[(4,4),(5,4)]$
- In der 2.ten Ebene
  - $(2,2),(3,2)$
  - $[(2,3),(3,3)],(4,2)$
  - $[[(2,4),(3,4)],(4,3)],(5,2)$
  - $[(5,3),(6,3)],(6,2)$
  - $(1,2),[(1,3),[(4,4),(5,4)]]$
- der 1.ten Ebene
  - $(2,1),(3,1)$
  - $[(2,2),(3,2)],(4,1)$
  - $[[(2,3),(3,3)],(4,2)],(5,1)$
  - $[[[(2,4),(3,4)],(4,3)],(5,2)],(6,1)$
  - $(1,1),[[(5,3),(6,3)],(6,2)]$

Die Knotenmenge des Baumes ist damit

$$\begin{aligned} F_0 &= \{(1,1),(2,1),(2,2),(2,3),(2,4),(3,1),(3,2),(3,3),(3,4), \\ &\quad (4,1),(4,2),(4,3),(5,1),(5,2),(5,3),(6,1),(6,2),(6,2)\}. \end{aligned}$$

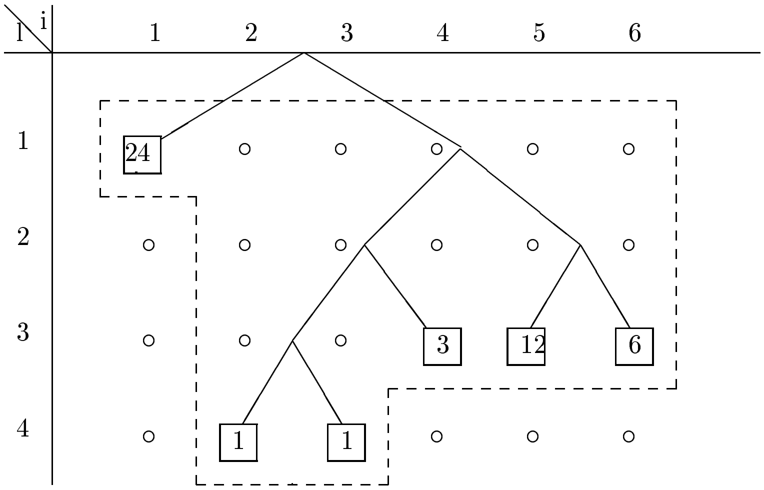


Abbildung 5.3: berechneter Baum mit seiner Knotenmenge

Die Blattlängen berechnen sich daraus zu

$$l_1 = 1, l_2 = 4, l_3 = 4, l_4 = 3, l_5 = 3, l_6 = 3.$$

Der zugehörige Baum mit der berechneten Knotenmenge ist in Abbildung 5.3 dargestellt.

Kapitel 6

Dynamische Huffman-Verfahren

In den vorangegangenen Kapiteln wurden statische Huffman-Codes betrachtet, bei denen der Quelle Kenntnisse über die zu sendenden Daten vorliegen, die bei der Verschlüsselung verwendet werden. Dies nennt man *off-line Codierung*. Dem gegenüber steht die *on-line Codierung*, bei der ein kontinuierlicher Datenstrom von der Quelle zum Empfänger fließt und beiden nur Teile der Daten bekannt sind. Bei dieser dynamischen Methode wird die Nachricht mit Hilfe des bekannten Nachrichtenteils und seiner zugrundeliegenden Statistik verschlüsselt.

Damit jeder Buchstabe möglichst optimal verschlüsselt wird, ist eine regelmäßige Anpassung der Codierung an die sich ändernde Statistik notwendig. Das geschieht durch die Konstruktion aktueller Code-Bäume. Diese werden nicht neu berechnet nach dem Verfahren von Huffman, sondern die vorhandenen Bäume werden an die veränderte Statistik angepaßt. Die dazu nötigen Operationen werden in diesem Kapitel vorgestellt. Sie bilden die Grundlage für dynamische Verfahren wie die one-pass Methode oder das Alterungsverfahren.

Der grundlegende Gedanke zur Baumtransformation kam von Faller [4] und Gallager [5]. Knuth [16] verändert ihren Algorithmus, so daß Nullgewichte betrachtet werden können. Der Algorithmus hat lineare Laufzeit in der Höhe des anzupassenden Blattes. Nach Vitter [24] ist die Länge der durch diese Methode entstandenden Codierung etwa doppelt so hoch wie durch die statische Methode plus einem Extrabit pro gesendetem Buchstaben.

Diese Analyse führt Vitter [24] zu einem neuen Verfahren, das nur eine ein Bit pro gesendetem Buchstaben längere Codierungen erzeugt als das statische Verfahren. Dieses Verfahren ist worst-case optimal bezüglich aller dynamischen Huffman-Verfahren.

Cormack und Horspool [3] verfeinern die Idee der Baumanpassung an eine veränderte Statistik. Ist es in allen anderen Verfahren nur möglich, ein Gewicht um eine Einheit zu verändern, so können jetzt Gewichte um beliebige Beträge erhöht oder vermindert werden. Dieser Algorithmus findet Anwendung bei der Implementierung von Alterungs- und Fenster-Methoden, die weiter entfernten Buchstaben eine geringere Priorität zuweisen als dichter liegenden Buchstaben.

6.1 Das klassische one-pass Huffman-Verfahren

Der Vorteil der on-line Codierung ist eine Verminderung der Anzahl von Datendurchläufen, da eine vorherige statistische Analyse der Nachricht durch den Sender entfällt: Sind beim statischen Huffman-Verfahren noch zwei Datendurchläufe notwendig (Bestimmen der Buchstabenfrequenzen mit Konstruktion des Huffman-Baumes und Versenden der Verschlüsselungs-Tabelle, Verschlüsselung und Senden der Datei), so ist mit der dynamischen Methode nur noch ein Datendurchlauf notwendig, da die Buchstabenhäufigkeiten nicht im Voraus bestimmt werden. Daher auch die Bezeichnung one-pass Methode für die dynamischen Verfahren und two-pass Methode für das statische Verfahren.

Zu Beginn wird eine statistische Gleichverteilung der Buchstaben angenommen. Dabei handelt es sich meistens um die Nullverteilung. Alle Buchstaben haben das gemeinsame Gewicht Null. Sind t Codewörter gesendet und entschlüsselt worden, dann wird der $t+1$.te Buchstabe anhand der Verteilung der ersten t Buchstaben codiert. Dies ruft eine Änderung der Buchstabenverteilung hervor und diese wird beim Senden des nächsten Buchstaben berücksichtigt. Deshalb sind Algorithmen nötig, die den Huffman-Baum zu der neuen Buchstabenverteilung bestimmen. Wie bereits erwähnt, ist es nicht notwendig, den Baum neu aufzubauen, sondern es werden Algorithmen vorgestellt, die eine Transformation des vorhandenen Baumes in den gesuchten Baum ermöglichen. Falls dem Sender und dem Empfänger die gleichen Adaptions-Algorithmen bekannt sind, ist keine Übermittlung der Baumstruktur mehr notwendig, da beiden während der gesamten Übertragungsdauer identische Codebäume vorliegen.

Beispiel 6.1 Sei das Quellalphabet $\mathcal{A} = \{a, b, c, d\}$ und bereits die Nachricht *abcd* gesendet worden. Dann ist der aktuelle Codebaum der linke Baum in Abbildung 6.1. Dieses Schema zeigt den jeweils aktuellen Codebaum während des anschließenden Sendens der Nachricht *bbb* und die jeweilige Verschlüsselung des Buchstabens.

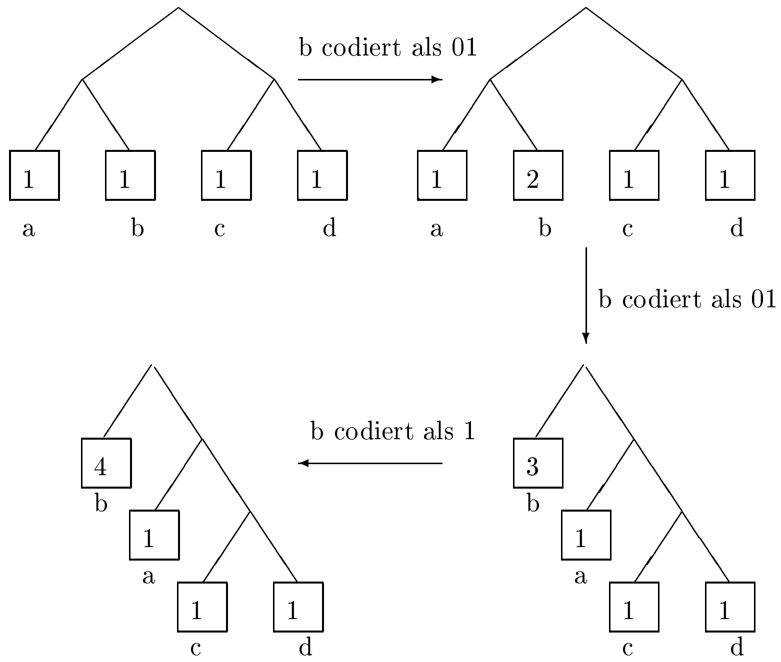


Abbildung 6.1: Codebäume zum Beispiel 6.1

6.2 Alterungs-Methode und Fenster-Methode

Diese Methoden sind Verfeinerungen der klassischen Methode, bei der alle Buchstaben von gleicher Wichtigkeit sind, unabhängig davon, *wann* sie gesendet wurden. Diese feineren Methoden gewichten aktuelle Buchstaben stärker. Inhomogene Texte werden damit besser behandelt, da sich die Codierung an neue Textarten anpaßt.

Der Grundgedanke der Alterungs-Methode ist eine Verminderung der Buchstabengewichte nach jeweils N gesendeten Buchstaben. Dadurch gewinnen neu gesendete Buchstaben relativ an Gewicht. Die Verringerung kann durch Multiplikation der Gewichte mit einer Zahl $r < 1$ durchgeführt werden.

Ein feineres Verfahren ist das exponentielle Altern, bei dem sich das Gewicht des t.ten gesendeten Buchstaben statt um Eins um den Wert s^{t-1} ($s > 1$) erhöht. Erreicht s^{t-1} die Berechenbarkeitsgrenzen, dann werden die Gewichte aller Buchstaben durch diesen Wert geteilt. Wie vorher erhöhen sich dann die Gewichte der gesendeten Buchstaben beginnend bei Eins um Potenzen von s.

Die Fenster-Methode wiederum legt ein „Sichtfenster“ über die letzten N gesendeten Buchstaben, das bei jeder Neuübermittlung um Eins nach rechts verschoben wird. Es wird also bei jedem neu eintreffenden Buchstaben das Gewicht des $N+1$-letzten gesendeten Buchstaben um Eins verringert und das Gewicht des neuen Buchstaben um Eins erhöht.

Beispiel 6.2 Die bereits gesendete Nachricht sei *abcd* und $N = 4$. Der nächste gesendete Buchstabe ist b. Bei den verschiedenen Methoden liegen vorher und nachher folgende Verteilungen vor:

Alterung durch Mutiplikation: Die Gewichtsverteilung ist vor dem Senden von b $w(a) = w(b) = w(c) = w(d) = 1$ und nach dem Senden $w(a) = w(c) = w(d) = r, w(b) = 1 + r$.

Exponentielles Altern: Die Gewichtsverteilung ist vor dem Senden von b $w(a) = 1, w(b) = s, w(c) = s^2, w(d) = s^3$ und nach dem Senden $w(a) = 1, w(b) = s + s^4, w(c) = s^2, w(d) = s^3$.

Fenster-Methode: Die Gewichtsverteilung ist vor dem Senden von b $w(a) = w(b) = w(c) = w(d) = 1$ und nach dem Senden $w(a) = 0, w(b) = 2, w(c) = w(d) = 1$.

6.3 Baumtransformation nach Faller, Gallager und Knuth

Das Verfahren der one-pass-Methode wurde bereits beschrieben. Von Interesse ist der Algorithmus zur Anpassung des Code-Baumes an die veränderte Statistik.

Definition 6.3 *T_t benennt den optimalen Codebaum zur Häufigkeitsverteilung der letzten t gesendeten Buchstaben.*

Im Folgenden wird ein Verfahren angegeben, das den Codebaum T_t in den Codebaum T_{t+1} umwandelt. Das den $t + 1$.ten Buchstaben repräsentierende Blatt (und damit auch alle seine Vorgänger) hat in T_{t+1} ein um Eins höheres Gewicht als in T_t. Aber erhöht man im Baum T_t lediglich die Gewichte der entsprechenden Knoten, dann liefert dies nicht unbedingt den optimalen Baum T_{t+1}.

Beispiel 6.4 Ist T_t wie in Abbildung 6.2 rechts und wird der Buchstabe b gesendet, dann liefert die naive Gewichtserhöhung den in der Abbildung links angegebenen Baum. Dieser ist nicht optimal.

Es ist also eine komplexere Vorgehensweise notwendig. Diese transformiert den Huffman-Baum T_t in einen anderen Huffman-Baum T_t^*, in dem die entsprechenden Gewichte problemlos erhöht werden können. Diese liefert den gesuchten Baum T_{t+1}.

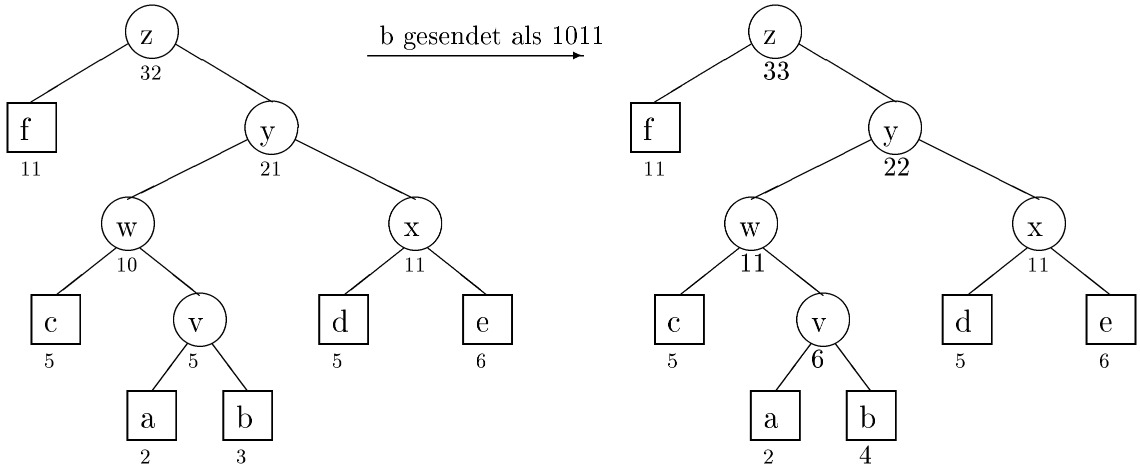


Abbildung 6.2: naive Gewichtserhöhung zum Beispiel 6.4

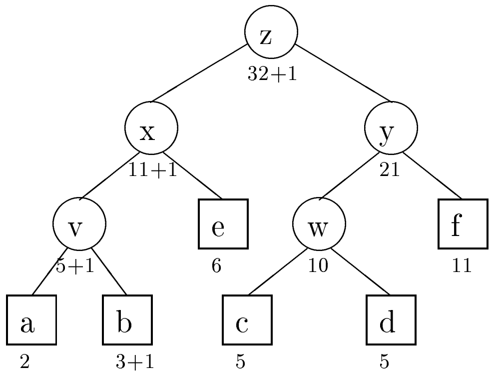
Transformation von T_t in T_t^*

Sei $v_1, \ldots, v_{2t-1}$ eine Knotenauflistung im Sinne der Huffman-Numerierung nach Definition 2.12. $v_{i_1}, \ldots, v_{i_l}$ seien die Vorfahren des zu erhöhenden Knotens v_{i_0}. Weil die Geschwister-Bedingung sowohl vor der Erhöhung wie auch nach der Erhöhung erfüllt sein soll, muß gelten

$$w(v_{i_j}) < w(v_{i_j+1}), j = 0, \ldots, l.$$

Ist dies in T_t nicht der Fall, dann werden die Knoten v_{i_j} (beginnend bei v_{i_0}) samt ihrer zugehörigen Teilbäume mit dem in der Liste letzten Knoten gleichen Gewichts getauscht. Danach ist die oben genannte Bedingung erfüllt und das Resultat ist der Baum T_t^*.

Beispiel 6.5 Die richtige Vorgehensweise zu Beispiel 6.4 liefert den rechts gezeigten Baum, die genauen Operationen sind im Beispiel 6.6 angegeben.



Falls der $t+1$.te gesendete Buchstabe nicht im Code-Baum berücksichtigt wurde, ist eine Kennzeichnung dieses Falles durch den Sender notwendig. Zu diesem

Zweck existiert im Huffman-Baum ein Blatt mit Gewicht 0, das alle unbekannten Buchstaben repräsentiert. Wird der Code dieses Null-Blattes übertragen, erfolgt anschließend die Spezifikation des neuen Buchstaben mittels eines festgelegten statischen Codes (zum Beispiel der ASCII-Tabelle). Nach Erhalt des neuen Buchstabens wird dieser in dem Huffman-Baum eingefügt, indem das Null-Blatt zum Vater zweier Blätter wird, des neuen Null-Blattes und des Blattes, das den neuen Buchstaben repräsentiert. Dieses neue Blatt erhält das Gewicht Eins, so daß die Gewichte aller Vorgänger des Nullblattes erhöht werden müssen. Um dies richtig zu behandeln, geht man wie bereits beschrieben vor.

6.3.1 Der Algorithmus

Ist x der $t+1$.te Buchstabe der Nachricht, dann wird der Algorithmus aufgerufen mit Baum_Transformieren(T_t, x). Er liefert den Baum T_{t+1}.

Liegt das Blatt des Buchstabens x in der Ebene l_i, dann hat der Algorithmus die Komplexität $O(l_i)$, da die WHILE-Schleife so oft durchlaufen wird und alle Operationen innerhalb dieser in konstanter Zeit ausführbar sind.

Beispiel 6.6 Liegt die gleiche Situation wie in Beispiel 6.4 vor, dann nimmt der Algorithmus die in Abbildung 6.3 beschriebenen Änderungen am Codebaum vor.

6.3.2 Analysierender Vergleich der one-pass und two-pass Methode

Im Folgenden werden die Kosten der statischen Methode mit den Kosten dieser dynamischen Methode verglichen. Dabei werden nur die Kosten für das Durchlaufen der Codebäume betrachtet. Nicht berücksichtigt werden

- bei der dynamischen Methode die Kosten der Spezifizierung eines unbekannten Buchstabens (beträgt $\lceil \log_2 n \rceil$ pro Buchstaben).
- bei der statischen Methode die Kosten zur Beschreibung der Baumform und zur Benennung der Blätter für den Empfänger ($O(n)$ mehr als die vernachlässigten Kosten der dynamischen Methode).

Bei langen Nachrichten sind diese Kosten im Vergleich zur Gesamtmenge der gesendeten Bits gering.

Definition 6.7 *Es seien bereits t Buchstaben gesendet worden.*
K_t gibt die Kommunikations-Kosten bei Benutzung eines statischen Huffman-Codes zu diesen t Buchstaben an, $s_t := K_t - K_{t-1}$.

```
PROCEDURE Baum_Transformieren(T:Baum,p:Knoten):Baum;
VAR u:Buchstabe aus 𝒜;
BEGIN
IF p ist Nullknoten THEN
    u :=Spezifikation des neuen Buchstaben durch statischen Code (ASCII);
    ersetze Nullknoten durch (Null, Null, u);
    w(u) := 0;
    p := u;
END;
IF p ist Geschwisterkind des Nullknotens THEN
    tausche p mit dem Blatt gleichen Gewichts und größter Huffman-Nummer;
    INC w(p);
    p:=Vater(p);
END;
WHILE p ist nicht die Wurzel DO
    tausche [p] mit dem Knoten (samt Nachfahren) gleichen Gewichts und
        größter Huffman-Nummer;
    INC w(p);
    p:=Vater(p);
END;
INC w(p);
RETURN [p];
END.
```

Tabelle 6.1: Algorithmus zur Methode von Faller, Gallager und Knuth

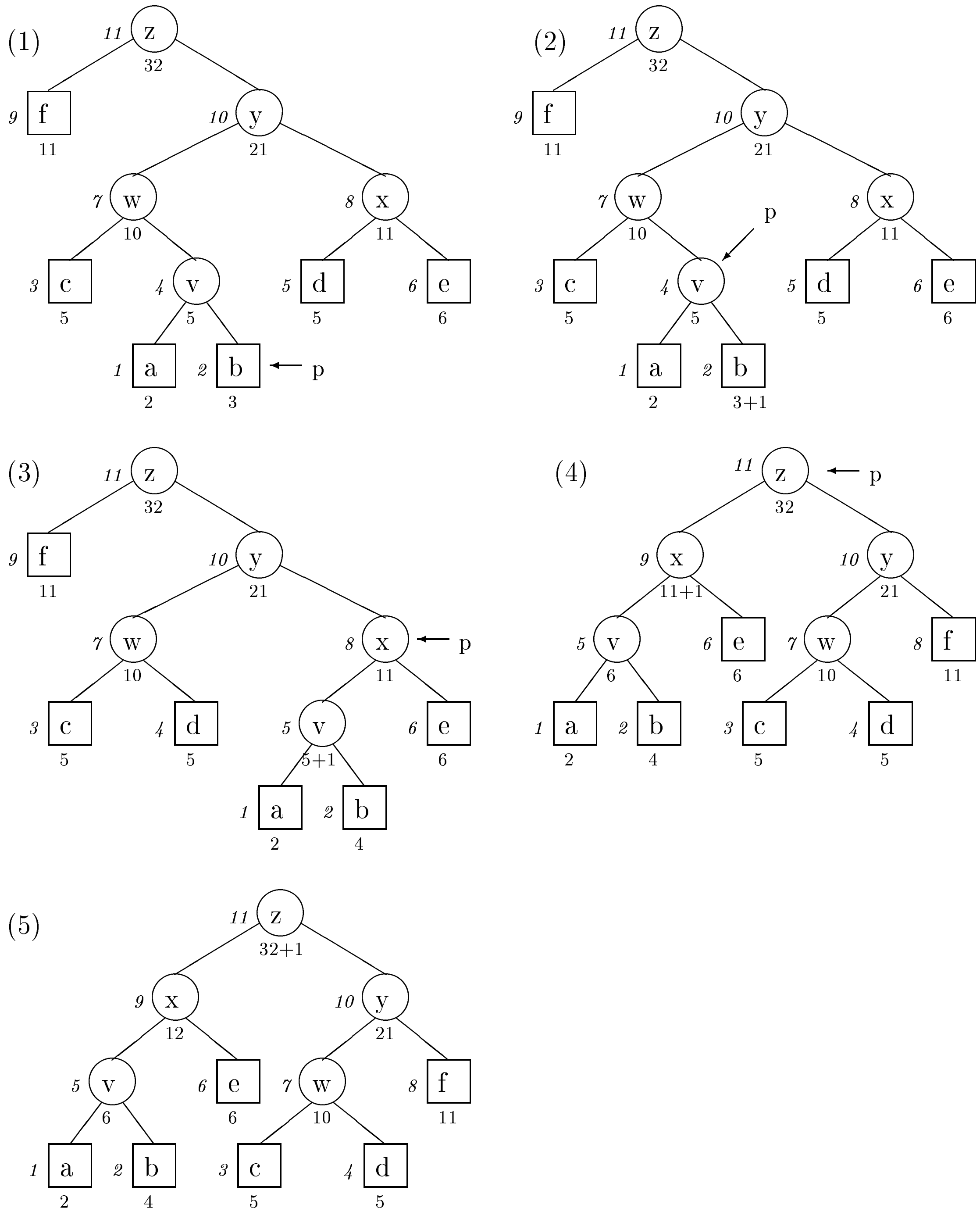


Abbildung 6.3: Manipulation des Codebaumes durch den Algorithmus

d_t gibt die Kommunikations-Kosten für das Senden des t-Buchstabens mit der dynamischen Methode an.
D_t gibt die gesamten Kommunikations-Kosten für alle t Buchstaben bei Benutzung eines dynamischen Verfahrens an.

Bemerkung 6.8

1. Die Kommunikationskosten der statischen Methode entsprechen der Länge aller zu durchlaufenden Pfade im Baum T_t, also $K_t = K_{T_t} := \sum_{v \in \beta(T_t)} w(v) l(v)$.

2. Das Versenden des t.ten Buchstabens verursacht Kosten, die der Länge des Pfades zu dem entsprechenden Blatt im Baum T_{t-1} entsprechen, also $d_t := l_t(T_{t-1})$, wobei $l_t(T_{t-1})$ angibt, welche Höhe das Blatt des t.ten gesendeten Buchstabes im Baum T_{t-1} hat.

3. D_t läßt sich mittels $D_t := D_{t-1} + d_t, D_0 := 0$ rekursiv berechnen.

Der folgende Satz zeigt, daß die Kosten für diese dynamische Codierung etwa doppelt so hoch sind wie für die statische Codierung plus einem Extrabit pro gesendetem Buchstaben. Dieses ist das vereinfachte Theorem von Vitter [24]. Die vollständige Abschätzung ist dort nachzulesen.

Satz 6.9 *D_t wird durch $\approx 2K_t + t$ nach oben beschränkt.*

Für den Beweis dieses Satzes werden zahlreiche Hilfsaussagen benötigt.

Jedem Knoten kann in Abhängigkeit seiner Bewegung während der Algorithmusausführung eine Attributfolge zugewiesen werden. Diese gibt an, welche Folge von Höhenänderungen der Knoten während der Ausführung erfuhr.

Definition 6.10 *Wandert ein Knoten während der Ausführung des Algorithmus eine Ebene nach oben, bekommt er das Attribut $\uparrow$.*
Wandert er zwei Ebenen nach oben, bekommt er das Attribut $\uparrow\uparrow$.
Wandert er eine Ebene nach unten, bekommt er das Attribut $\downarrow$.
Verändert er seinen Platz in einer Ebene, bekommt er das Attribut $\rightarrow$.
Der Nettowert der Höhenänderung des Blattes des t.ten Buchstabens durch die Ausführung des Algorithmus wird mit h_t bezeichnet.

Beispiel 6.11 Im Beispiel 6.6 erhält das zu b gehörige Blatt die Attributfolge $\rightarrow\uparrow$ und der Nettowert der Höhenänderung beträgt Eins.

Der folgende Satz zeigt, daß andere Attribute nicht notwendig sind.

Satz 6.12 *Sei T ein optimaler Baum mit Nullknoten.*

1. *Die Pfadlängendifferenz zweier Knoten gleichen Gewichts, von denen keiner Geschwisterkind des Nullknotens ist, ist maximal Eins.*

2. *Die Pfadlängendifferenz zweier Knoten gleichen Gewichts, von denen einer Geschwisterkind des Nullknotens ist, ist maximal Zwei.*

Beweis 6.12

1. Sind u, v Knoten mit gleichem Gewicht und $l(u) - l(v) > 1$, dann befindet sich der Vater von u mindestens eine Ebene unter dem Knoten v. Da der Vater von u größeres Gewicht als v hat, senkt der Austausch dieser Knoten das Gesamtgewicht des Baumes. Dies ist ein Widerspruch zur Optimalität.

2. Sind u, v Knoten gleichen Gewichts mit $l(u) - l(v) > 2$ und ist u Geschwisterkind des Nullknotens, dann hat der Vater von u das gleiche Gewicht wie v mit $l(Vater(u)) - l(v) > 1$. Dies ist aufgrund von (1) nicht möglich.

$\diamond$

Bemerkung 6.13

1. $h_t = |\uparrow| + 2|\uparrow\uparrow| - |\downarrow|$, wobei $|Attribut|$ die Häufigkeit dieses Attributs in der zugehörigen Attributfolge beschreibt.

2. Knoten, die nicht Geschwister des Nullknotens sind, bekommen niemals das Attribut $\uparrow\uparrow$ zugewiesen.

3. Da man beim Algorithmus höchstens einmal auf ein Geschwisterkind des Nullknotens trifft, kann es nur ein $\uparrow\uparrow$ pro Algorithmusaufruf geben.

Satz 6.14 *In der Attributfolge eines Blattes befindet sich zwischen je zwei Attributen $\downarrow$ stets ein $\uparrow$.*

Beweis 6.14 Angenommen, die Knoten a_1, b_1 und a_2, b_2 werden im Algorithmus durch ein $\downarrow$ vertauscht und dazwischen findet kein $\uparrow$ statt.
Sei $k = l(a_2) - l(b_1)$ und a_1^k der k.te Vorfahre von a_1. Dann gilt $l(a_1^k) = l(a_2) - 1$. Gezeigt wird zunächst

$$w(a_1^k) = w(a_2).$$

- Ist $w(a_1^k) < w(a_2)$, dann verringert der Austausch dieser Knoten das Gesamtgewicht des Baumes. Dies ist ein Widerspruch zur Optimalität des Baumes.

- Ist $w(a_1^k) > w(a_2)$, dann tauscht ein Vorfahre von a_1 den Platz mit dem aktuellen Knoten, der sich eine Ebene tiefer befindet. Dadurch befindet sich das Attribut $\uparrow$ zwischen den Attributen $\downarrow$:
  Sei $p_0 = a_1, p_2, \ldots, p_k = a_2$ eine Auflistung der betrachteten Knoten p. Ist $w(a_1^k) > w(a_2)$, dann gibt es einen Index j, so daß erstmalig $w(a_1^j) > w(p_{j-1})$ ist. Da $w(a_1^{j-1}) = w(p_{j-2})$, muß a_1^{j-1} die höhere Huffman-Nummer besitzen. Dies hat aber den oben beschriebenen Austausch von Knoten zur Folge.

Aus $w(a_1^k) = w(a_2)$ folgt $w(a_1^k) = w(b_2)$. Da die Pfadlängendifferenz dieser Knoten größer als Eins ist, ist dies ein Widerspruch zu Satz 6.12.1. $\diamond$

Satz 6.15 *Für den Nettowert der Höhenänderung jedes Knotens gilt* $h_t \leq \frac{d_t+1}{2}$.

Beweis 6.15

- Nach jedem Attribut $\rightarrow$ wandert der aktuell betrachtete Knoten p des Algorithmus eine Ebene nach oben. Ebenso wandert p nach jedem Attribut $\uparrow$ zwei Ebenen nach oben. Da für die Kinder der Wurzel kein $\uparrow$ möglich ist und jedes Attribut $\downarrow$ von einem $\uparrow$ gefolgt wird (auf Grund von Satz 6.14), gilt für die Knoten, die nicht der Nullknoten oder Geschwister dessen sind,
$$h_t \leq \lfloor \frac{d_t}{2} \rfloor.$$

- Da für die Geschwister des Nullknotens auf Grund von Bemerkung 6.13.3 maximal ein $\uparrow\uparrow$ auftritt, gilt für diese
$$h_t \leq \lceil \frac{d_t}{2} \rceil.$$

- Da der Nullknoten während des Algorithmusaufrufes mit keinem Knoten den Platz tauscht, gilt für ihn
$$h_t \leq \lceil \frac{d_t}{2} \rceil - 1.$$

Alle diese Werte sind durch $\frac{d_t+1}{2}$ nach oben beschränkt. $\diamond$

Satz 6.16 *Für Bäume, in denen kein Null-Blatt vorhanden ist, gilt* $h_t = d_t - s_t$.

Beweis 6.16 Es gilt
$$h_t = l_t(T_{t-1}) - l_t(T^*_{t-1})$$
und aus Bemerkung 6.8.2 folgt
$$d_t = l_t(T_{t-1}).$$
s_t läßt sich berechnen zu
$$s_t = K_t - K_{t-1} = \sum_{v \in \beta(T_t)} w(v)l(v) - \sum_{v \in \beta(T^*_{t-1})} w(v)l(v).$$
Da sich T_t und T^*_{t-1} nur durch die Erhöhung von $l_t(T^*_{t-1})$ vielen Gewichten unterscheiden, gilt $s_t = l_t(T^*_{t-1})$. Damit ist die Behauptung bewiesen. $\diamond$

Beweis 6.9:
Die Differenz der Kommunikationkosten läßt sich für Bäume ohne Null-Blatt folgendermaßen abschätzen:
$$D_t - K_t = \sum_{i=0}^{t} d_i - s_i = \sum_{i=0}^{t} h_i \leq \sum_{i=0}^{t} \frac{d_i + 1}{2} = \frac{D_t + t}{2}.$$
Daraus folgt
$$D_t \leq 2K_t + t.$$
$\diamond$

6.4 Baumtransformation nach Vitter

Vitter stellt eine Baumtransformation vor, die ein worst-case optimales dynamisches Huffman-Verfahren liefert. Zu jedem dynamischen Huffman-Verfahren mit einer anderen Baumtransformation gibt es also eine Nachricht mit Kommunikationskosten, die mindestens den maximalen Kosten für dieses Verfahren entsprechen. Die Minimierung der Kommunikationskosten basiert auf folgenden Ideen:

- Die Anzahl der Attribute $\uparrow$ pro Knoten wird beschränkt – in diesem Fall durch 1.

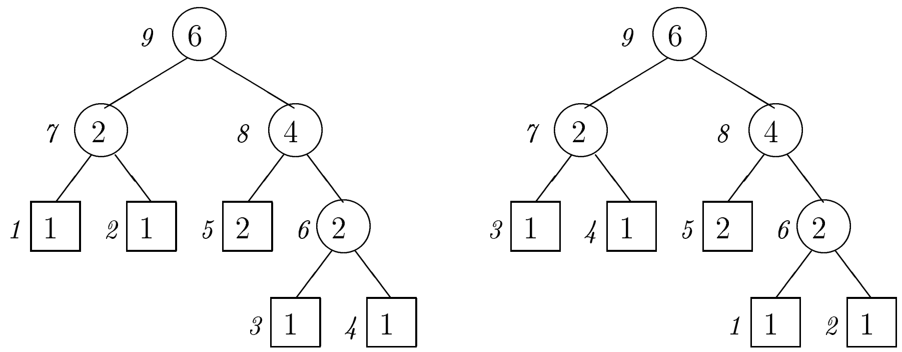


Abbildung 6.4: Huffman-Numerierung und implizite Numerierung

- Die Code-Bäume sollen nicht nur $\sum_{v \in \beta(T)} w(v) l(v)$ minimieren, sondern auch $\sum_{v \in \beta(T)} l(v)$ und $\max_{v \in \beta(T)} l(v)$. Dadurch wird die Codierungslänge des nächsten Buchstabens klein gehalten.

Diese Anforderungen werden durch eine neue Art der Knoten-Numerierung anstelle der Huffman-Numerierung erfüllt. Diese Numerierung wird *implizit* genannt.

Definition 6.17 *Die Numerierung der Knoten eines Huffman-Baumes heißt implizit, wenn sie folgende Eigenschaften erfüllt:*

1. *Je größer die Pfadlänge ist, umso kleiner ist die Nummer des Knotens.*
2. *Knoten einer Ebene haben von links nach rechts ansteigende Nummern.*

Bemerkung 6.18 Jede implizite Numerierung ist auch eine Huffman-Numerierung. Die Umkehrung gilt nicht, siehe Abbildung 6.4.

Wendet man den Algorithmus des Kapitels 6.1 mit dieser Numerierung statt der Huffman-Numerierung an, dann macht man folgende Beobachtungen hinsichtlich der Attribute der Knoten:

Satz 6.19

1. *Kein Knoten hat das Attribut $\downarrow$.*
2. *Besitzt ein innerer Knoten das Attribut $\uparrow$, dann hat er seinen Platz mit einem Blatt getauscht.*

Beweis 6.19

1. Da Knoten ihre Plätze mit Knoten höherer Nummer tauschen und die implizite Numerierung wegen Definition 6.17.1 ebenenweise ansteigt, ist kein $\downarrow$ möglich.

2. Sind u, v innere Knoten gleichen Gewichts mit $l(u) - l(v) = 1$, dann haben deren Kinder genau das halbe Gewicht ihrer Eltern (sonst würde ein Austausch der Kinder die Kosten des Baumes senken). Tauscht u mit v den Platz, dann wurde vor der Betrachtung von u das rechte Kind von u untersucht. Da dieses eine kleinere Nummer als die Kinder von v besitzt, muß zuvor ein Austausch dieser Knoten stattgefunden haben. In diesem Fall wird der Knoten u nicht mehr betrachtet, was der Annahme widerspricht.

$\diamond$

Die Einschränkung des Attributes $\uparrow$ auf Eins pro Knoten kann durch die Aufrechterhaltung folgender Invariante erreicht werden:

Satz 6.20 (Invariante) *Blätter haben eine kleinere implizite Nummer als innere Knoten gleichen Gewichts.*

Bemerkung 6.21 Jeder Huffman-Baum, der diese Invariante erfüllt, minimiert $\sum_{v \in \beta(T)} l(v)$ und $\max_{v \in \beta(T)} l(v)$, siehe Schwartz [22].

Satz 6.22 *Ist die Invariante erfüllt, dann gilt:*

1. *Kein Knoten hat das Attribut* $\uparrow\uparrow$.

2. *Nur Blätter besitzen das Attribut* $\uparrow$.

Beweis 6.22

1. Wegen Bemerkung 6.13.2 ist das Attribut $\uparrow\uparrow$ nur für Geschwister des Nullknotens möglich.
   Ist u Geschwisterkind des Nullknotens und v ein Knoten gleichen Gewichts mit $l(u) - l(v) = 2$, dann ist v ein innerer Knoten. Dies gilt, da der Vater von u ebenfalls gleiches Gewicht hat, ein innerer Knoten ist und die Invariante erfüllt sein soll.
   Die Kinder von v haben eine implizite Nummer zwischen der Nummer des Nullknotens und der Nummer des Knotens v. Da diese Numerierung auch eine Huffman-Numerierung ist und das Gewicht der Kinder kleiner ist als das Gewicht der Knoten u und v , ist dies ein Widerspruch zur ansteigenden Numerierung mit ansteigendem Gewicht.

2. Besitzt ein innerer Knoten das Attribut $\uparrow$, dann hat er auf Grund von Satz 6.19.2 seinen Platz mit einem Blatt gleichen Gewichts getauscht. Dies ist nicht möglich, da Blätter eine kleine Nummer haben, wenn die Invariante erfüllt ist.

$\diamond$

Satz 6.23 *Ist die Invariante erfüllt, dann gilt für den Nettowert der Höhenänderung jedes inneren Knotens $h_t = 0$ und für den Nettowert der Höhenänderung jedes Blattes $h_t = 1$.*

Beweis 6.23 Auf Grund von Bemerkung 6.13.1 ist $h_t = |\uparrow| + 2|\uparrow\uparrow| - |\downarrow|$. Wegen Satz 6.19.1 und Satz 6.22.1 gibt es keine Attribute $\downarrow$ und $\uparrow\uparrow$. Auf Grund von Satz 6.22.2 besitzen nur Blätter das Attribut $\uparrow$. $\diamond$

Satz 6.24 *D_t wird durch $\approx K_t + t$ nach oben beschränkt.*

Beweis 6.24 Die Differenz der Kommunikationkosten läßt sich folgendermaßen abschätzen:

$$D_t - K_t = \sum_{i=0}^{t} d_i - s_i = \sum_{i=0}^{t} h_i \leq t.$$

Daraus folgt

$$D_t \leq K_t + t.$$

$\diamond$

6.4.1 Der Algorithmus

Zur Aufrechterhaltung der Invariante 6.20 ist eine Veränderung des Algorithmus 6.1 zur getrennten Betrachtung von Blättern und inneren Knoten notwendig.

Definition 6.25 Blöcke *sind Mengen, die nur innere Knoten oder nur Blätter gleichen Gewichts enthalten.*
Der Block mit inneren Knoten (Blättern) des Gewichts x wird mit $I(x)$ ($B(x)$) bezeichnet.
Der Vertreter des Blocks ist der Knoten größter impliziter Nummer.
Auf den Blöcken existiert eine Ordnung, die der Invariante genügt: $B(x)$ steht vor $I(x)$ und Blöcke mit Knoten kleineren Gewichts befinden sich in der Ordnung weiter vorn.
Ist u ein Knoten, dann bezeichnet $N(u)$ den Block, der auf den u enthaltenden Block folgt.

PROCEDURE Baum_Transformieren(T:Baum,p:Knoten):Baum;
VAR u:Buchstabe aus $\mathcal{A}$;
BEGIN
IF p ist Nullknoten THEN
    u:=Spezifikation des neuen Buchstaben durch statischen Code (ASCII);
    ersetze Nullknoten durch $(Null^*, Null, u)$;
    $w(u) := 1$;
    $p := Null^*$;
ELSIF p ist Geschwisterkind des Nullknotens THEN
    INC $w(p)$;
    $p := Vater(p)$;
END;
WHILE p ist nicht die Wurzel DO
    tausche $[p]$ mit seinem Blockvertreter samt Teilbaum aus;
    Verschieben_Inkrementieren(p);
END;
RETURN T;
END.

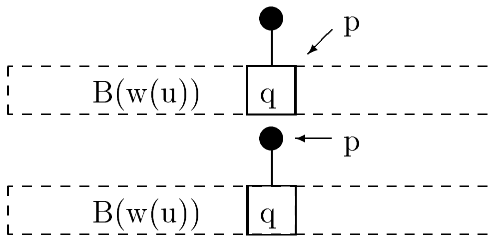
PROCEDURE Verschieben_Inkrementieren(VAR q:Knoten);
BEGIN
IF (q ist Blatt AND $N(q) = I(w(q))$)
OR (q ist innerer Knoten AND $N(q) = B(w(q)+1)$) THEN
    verschiebe den Knoten [q] hinter das Ende des Blockes $N(q)$;
END;
INC $w(q)$;
IF q ist Blatt THEN $q := neuer\_Vater(q)$;
ELSIF q ist innerer Knoten THEN $q := alter\_Vater(q)$; END;
END.

Tabelle 6.2: Algorithmus von Vitter

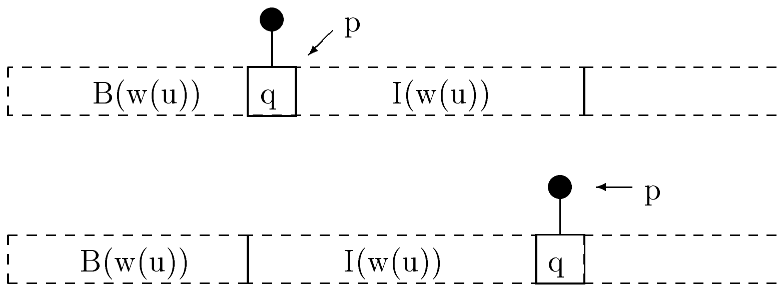
Der Algorithmus schiebt den zu erhöhenden Knoten mit seinem Teilbaum an das Ende seines Blocks, dabei rücken die dazwischen liegenden Knoten dieses Blocks mit ihren Teilbäumen einen Platz nach links. Der zu erhöhende Knoten ist dann Blockvertreter.

Falls notwendig, wird dieser Knoten nun weiter hinten in der Ordnung eingefügt, um die Invariante auch nach der Erhöhung aufrecht zu halten. Auch dabei rücken alle dazwischen liegenden Knoten mit ihren Teilbäumen einen Platz nach links. Dies ist nur für Knoten möglich, die weder der Nullknoten noch dessen Geschwisterkind sind. Diese beiden Fälle werden im Algorithmus auf den allgemeinen Fall zurückgeführt.

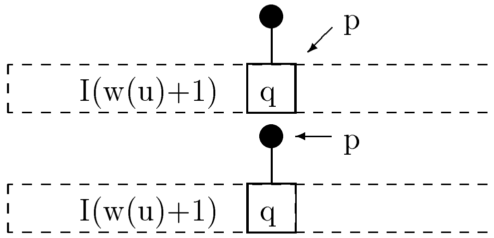
- Ist p ein Blatt und existiert kein Block mit inneren Knoten des Gewichts $w(p)$, dann kann das Gewicht von p ohne ein Verschieben des Knotens erhöht werden. Der danach zu erhöhende Knoten ist der Vater von p.



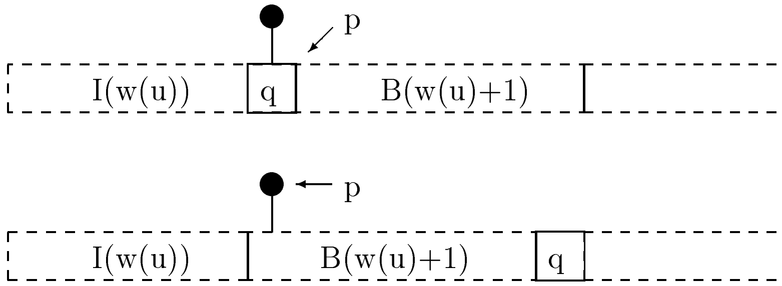
- Ist p ein Blatt und existiert ein Block mit inneren Knoten des Gewichts $w(p)$, dann wird p hinter dem Ende dieses Blockes eingefügt. Die inneren Knoten des Gewichts $w(p)$ mit ihren Teilbäumen rücken nach links auf. Der neue zu erhöhende Knoten ist der so erhaltene neue Vater von p.



- Ist p ein innerer Knoten und existiert kein Block mit Blättern des Gewichts $w(p)$, dann kann das Gewicht von p ohne ein Verschieben des Knotens erhöht werden. Der danach zu erhöhende Knoten ist der Vater von p.



- Ist p ein innerer Knoten und existiert ein Block mit Blättern des Gewichts $w(p)+1$, dann wird p hinter dem Ende dieses Blockes eingefügt. Die Blätter des Gewichts $w(p)+1$ rücken nach links auf. Der neue zu erhöhende Knoten ist der ursprüngliche Vater von p.



6.4.2 Die worst-case Optimalität

Gesucht wird zu jedem dynamischen Huffman-Verfahren mit einer beliebigen Art der Baumtransformation eine Nachricht mit Kommunikationskosten der Mindest-Größe $K_t + t$. Diese Nachricht erhält man mittels folgender Konstruktion.

Sei ab einem bestimmten Zeitpunkt jeder Buchstabe gleichhäufig in der gesendeteten Nachricht vorgekommen und der zu diesem Zeitpunkt vorliegende Baum vollständig mit Ausnahme eines Knotens in der letzten Ebene,

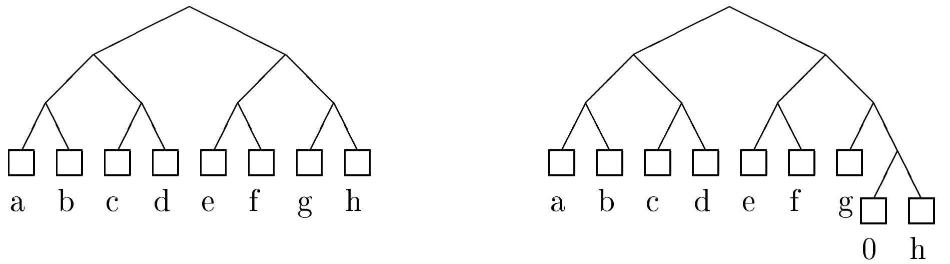


Abbildung 6.5: dynamischer und statischer Baum zur Nachricht aus 6.4.2

der der Vater des Nullknotens und dessen Geschwisterkindes ist (siehe Abbildung 6.5 links). Ist der nächste gesendete Buchstabe stets der durch das Geschwisterkind des Null-Blattes repräsentierte, dann hat jeder Buchstabe eine ein Bit längere Verschlüsselung als durch die statische Methode, da der Pfad in diesem Baum durch das Vorhandensein des Null-Knotens um Eins länger ist (siehe Abbildung 6.5 rechts).

Das Versenden dieser Nachricht erzeugt also unabhängig von der Art der Baumtransformation Kommunikationskosten der Mindest-Höhe $K_t + t$. Dies entspricht den maximalen Kosten, die durch das Benutzen der Baumtransformation nach Vitter hervorgerufen werden.

6.5 Das Verfahren von Cormack und Horspool

Das Alterungsverfahren verlangt die Möglichkeit der Gewichts-Erhöhung und -Verminderung um beliebige Beträge. Die bereits beschriebenen Verfahren erhöhen ein Gewicht nur um den Betrag Eins. Das folgende Verfahren berücksichtigt beliebige Änderungen.

Sei $v_1, \ldots, v_{2t-1}$ eine Knotenauflistung im Sinne der Huffman-Numerierung nach Definition 2.12. Zur Erhöhung des Gewichtes eines Knotens v_i um einen Betrag x unter Aufrechterhaltung der Geschwister-Bedingung muß folgendes erfüllt sein:

$$\begin{aligned} w(v_i) + x &\leq w(v_{i+1}) \quad \text{falls } v_{i+1} \text{ nicht Vater von } v_i \text{ ist} \\ w(v_i) + x &\leq w(v_{i+2}) \quad \text{falls } v_{i+1} \text{ Vater von } v_i \text{ ist} \end{aligned}$$

Diese Fallunterscheidung ist notwendig, damit die Knoten v_i und v_{i+1} im Verlauf des Algorithmus ihre Plätze tauschen können – dies ist nur definiert, falls kein Knoten Vorfahr des anderen ist: Sind v_i und v_{i+1} Sohn und Vater, dann ist v_{i+2} das Geschwisterkind des Knotens v_{i+1}. v_{i+2} ist somit nicht Großvater von v_i.

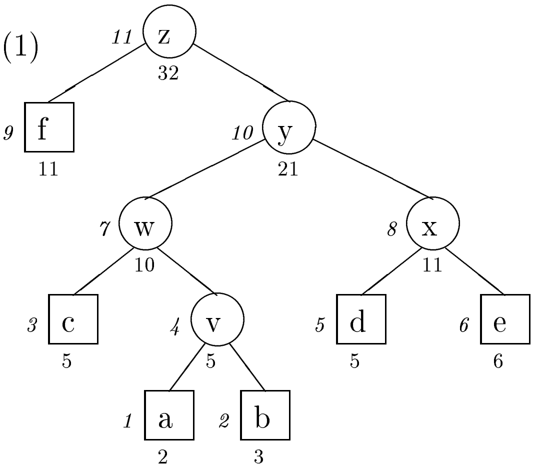
Ist obige Bedingung nicht erfüllt, dann wird das Gewicht dieses Knotens zunächst nur um den Betrag $diff = w(v_{i+1}) - w(v_i)$ bzw. $diff = w(v_{i+2}) - w(v_i)$ erhöht – dies ist anschließend auch für alle seine Vorfahren durchzuführen. Danach besitzen die Knoten v_i und v_{i+1} bzw. v_i und v_{i+2} das gleiche Gewicht und können samt ihrer Teilbäume ausgetauscht werden. Das Gewicht des Knotens v_i und seiner neuen Vorfahren ist anschließend um den noch fehlenden Betrag $x - diff$ zu erhöhen.

```
PROCEDURE Baum_Transformieren(T:Baum,p:Knoten,x:REAL):Baum;
VAR q:Knoten;
BEGIN
IF x ≠ 0 THEN
   q := Nachfolger(p);
   IF Vater(p) ist die Wurzel THEN
      w(Wurzel) := w(Wurzel) + x;
      w(p) := w(p) + x;
      IF w(p) > w(q) THEN tausche [p] und [q] aus; END;
   ELSE
      IF q = Vater(p) THEN q := Nachfolger(q); END;
      diff := w(q) - w(p);
      IF diff ≥ x THEN
         T:=Baum_Transformieren(T, Vater(p), x);
         w(p) := w(p) + x;
      ELSIF diff < x THEN
         T:=Baum_Transformieren(T, Vater(p), diff);
         w(p) := w(q);
         tausche [p] und [q] aus;
         T:=Baum_Transformieren(T, p, x - diff);
      END;
   END;
END;
RETURN T;
END.
```

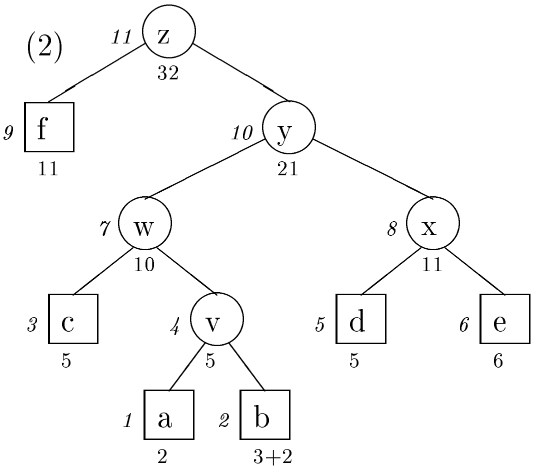
Nachfolger(p) bestimmt den in der Huffman-Numerierung auf p folgenden Knoten.

Tabelle 6.3: Algorithmus zur Methode von Cormack und Horspool

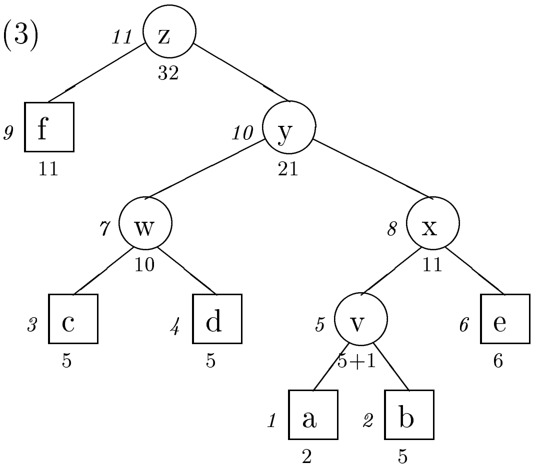
(1)


11 z
32
9 f
11
10 y
21
7 w
10
8 x
11
3 c
5
4 v
5
5 d
5
6 e
6
1 a
2
2 b
3

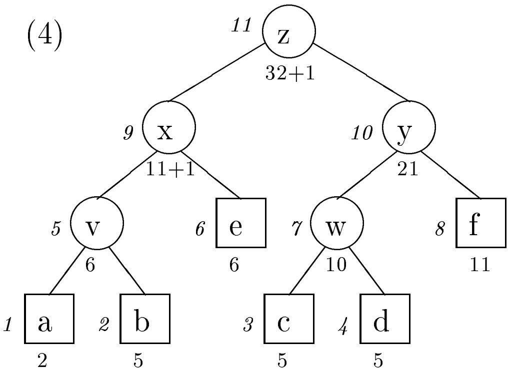
(2)


11 z
32
9 f
11
10 y
21
7 w
10
8 x
11
3 c
5
4 v
5
5 d
5
6 e
6
1 a
2
2 b
3+2

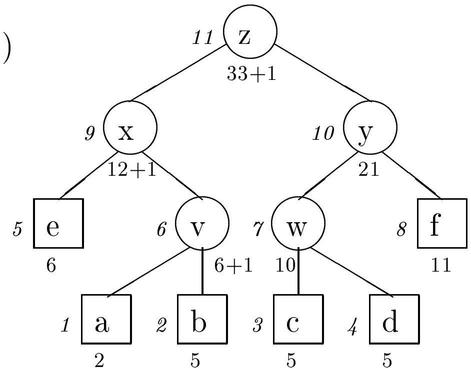
(3)


11 z
32
9 f
11
10 y
21
7 w
10
8 x
11
3 c
5
4 d
5
5 v
5+1
6 e
6
1 a
2
2 b
5

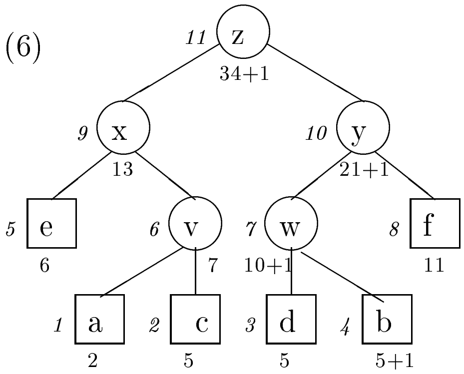
(4)


11 z
32+1
9 x
11+1
10 y
21
5 v
6
6 e
6
7 w
10
8 f
11
1 a
2
2 b
5
3 c
5
4 d
5

(5)


11 z
33+1
9 x
12+1
10 y
21
5 e
6
6 v
6+1
7 w
10
8 f
11
1 a
2
2 b
5
3 c
5
4 d
5

(6)


11 z
34+1
9 x
13
10 y
21+1
5 e
6
6 v
7
7 w
10+1
8 f
11
1 a
2
2 c
5
3 d
5
4 b
5+1

(7)

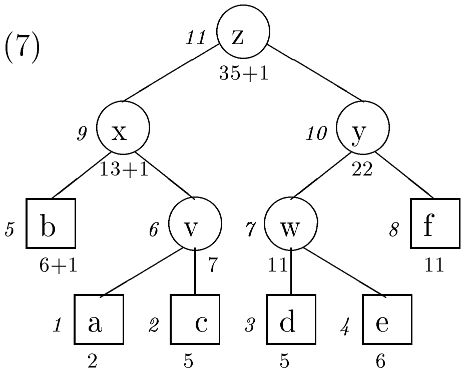


Abbildung 6.6: Manipulation des Codebaumes durch den Algorithmus

6.5.1 Beispiel

Ist T wie in Abbildung 6.6.(1) und soll das Gewicht des Blattes b um 4 erhöht werden, dann geschieht folgendes:

- Das Gewicht des Blattes b soll um 4 erhöht werden, es wird zunächst nur um 2 erhöht – siehe Abbildung 6.6.(2).

  1. Das Gewicht des Blattes v soll um 2 erhöht werden, dazu werden die Knoten v und d ausgetauscht.
  2. Das Gewicht des Blattes v soll um 2 erhöht werden, es wird zunächst nur um 1 erhöht – siehe Abbildung 6.6.(3).
     (a) Das Gewicht des Blattes x soll um 1 erhöht werden, dazu werden die Knoten x und f ausgetauscht.
     (b) Die Gewichte des Blattes x und seiner Vorfahren werden um 1 erhöht – siehe Abbildung 6.6.(4).
  3. Die Gewichte des Blattes v und seiner Vorfahren werden nach Austausch der Knoten v und e um den Restbetrag von 1 erhöht – siehe Abbildung 6.6.(5).

- Das Gewicht des Blattes b soll nach Austausch der Knoten b und c um den Restbetrag von 2 erhöht werden, dazu werden die Knoten b und d ausgetauscht.

- Das Gewicht des Blattes b soll immer noch um den Restbetrag von 2 erhöht werden, es wird zunächst mit den Gewichten seinen Vorfahren um den Betrag 1 erhöht – siehe Abbildung 6.6.(6).

- Die Gewichte des Blattes b und seiner Vorfahren werden nach Austausch der Knoten b und e um den Restbetrag von 1 erhöht – siehe Abbildung 6.6.(7).

Literaturverzeichnis

[1] Martin Aigner. *Combinatorial Search*. Wiley-Teubner Series in Computer Science. B.G.Teubner Stuttgart, John Wiley and Sons, 1988.

[2] A.Itai. Optimal Alphabetic Trees. *SIAM Journal On Computing*, 5:9–18, 1976.

[3] Gordon V. Cormack and R. Nigel Horspool. Algorithms for Adaptive Huffman Codes. *Information Processing Letters*, 18(3):159–165, March 1984.

[4] N. Faller. An Adaptive System for Data Compression. In *Record Of The 7th Asilomar Conference On Circuits, Systems, And Computers*, pages 593–597, 1973.

[5] Robert G. Gallager. Variations on a Theme by Huffman. *IEEE Transactions On Information Theory*, 24(6):668–674, November 1978.

[6] M.R. Garey. Optimal Binary Search Trees with Restricted Maximal Depth. *SIAM Journal On Computing*, 3(2):101–110, June 1974.

[7] Adriano M. Garsia and Michelle L. Wachs. A New Algorithm for Minimum Cost Binary Trees. *SIAM Journal On Computing*, 6(4):622–642, December 1977.

[8] E.N. Gilbert and E.F. Moore. Variable-Length Binary Encodings. *The Bell System Technical Journal*, 38(2):933–967, July 1959.

[9] T.C. Hu. A New Proof of the T-C Algorithm. *SIAM Journal On Applied Mathematics*, 25(1):83–94, July 1973.

[10] T.C. Hu and K.C. Tan. Path Length of Binary Search Trees. *SIAM Journal On Applied Mathematics*, 22(2):225–234, March 1972.

[11] T.C. Hu and A.C. Tucker. Optimal Computer Search Trees and Variable-Length Alphabetical Codes. *SIAM Journal On Applied Mathematics*, 21(4):514–532, December 1971.

[12] David A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings Of The I.R.E.*, 40:1098–1101, September 1952.

[13] R.M. Karp. Minimum Redundancy Coding for the Discrete Noiseless Channel. *Inst. Radio Engineers Transactions On Information Theory*, 7:27–38, 1961.

[14] Jeffrey H. Kingston. A New Proof of the Garsia-Wachs Algorithm. *Journal Of Algorithms*, 9:129–136, 1988.

[15] Donald E. Knuth. Optimum Binary Search Trees. *Acta Informatica*, 1:14–25, 1971.

[16] Donald E. Knuth. Dynamic Huffman Coding. *Journal Of Algorithms*, 6:163–180, 1985.

[17] Lawrence L. Larmore. Height Restricted Optimal Binary Trees. *SIAM Journal On Computing*, 16(6):1115–1123, December 1987.

[18] Lawrence L. Larmore and Daniel S. Hirschberg. A Fast Algorithm for Optimal Length-Limited Huffman-Codes. *Journal Of The Association For Computing Machinery*, 37(3):464–473, July 1990.

[19] Lawrence L. Larmore and Teresa M. Przytycka. A Fast Algorithm for Optimum Height-Limited Alpabetic Binary Trees. *SIAM Journal On Computing*, 23(6):1283–1312, December 1994.

[20] Lawrence L. Larmore Marek Karpinski and Wojciech Rytter. Correctness of Constructing Optimal Alphabetic Trees Revisited. Research Report No.85134-CS, Institut für Informatik der Universität Bonn, May 1995.

[21] R.-H. Schulz. *Codierungstheorie – Eine Einführung*. Vieweg Verlag, Friedr.Vieweg und Sohn Verlagsgesellschaft mbH,Braunschweig/Wiesbaden, 1991.

[22] E.S. Schwartz. An Optimum Encoding with Minimum Longest Code and Total Number of Digits. *Information Control*, 7(1):37–44, March 1964.

[23] James A. Storer. *Data Compression: Methods And Theory.* Computer Science Press, 1803 Reasearch Boulevard, Rockville, Maryland 20850, 6 edition, 1988.

[24] Jeffrey Scott Vitter. Design and Analysis of Dynamic Huffman Codes. *Journal Of The Association For Computing Machinery*, 34(3):825–845, October 1987.

[25] Russel L. Wessner. Optimal Alphabetic Search Trees with Restricted Maximal Heigth. *Information Processing Letters*, 4(4):90–94, January 1976.