4 Formatierung

Etwas vereinfacht sieht das Grundproblem der Formatierung wie folgt aus: Gegeben ist eine Folge von Absätzen (Paragraphen). Jeder Absatz besteht aus einer Folge von Wörtern und jedes Wort ist eine Folge von Buchstaben. Forme diese Eingabe in eine Folge von Seiten um, von denen jede wiederum eine Folge von Zeilen ist. Jede Zeile ist eine Folge von Buchstaben und Zwischenräumen. Daher ergeben sich die folgenden Teilprobleme:

- der Wortaufbau, d. h. das Zusammensetzen von Wörtern aus Buchstaben;
- der Zeilenumbruch, d. h. die Zerlegung eines Abschnitts in eine Folge von Zeilen;
- der Seitenumbruch, d. h. die Zerlegung einer Folge von umbrochenen Zeilen in eine Folge von Seiten;
- die Worttrennung, d. h. die manchmal erforderliche Trennung von Wörtern an Zeilenenden.

In Abschnitt 4.1 wird beschrieben, wie eine Texteingabe erstellt wird. Danach wird im Abschnitt 4.2 der Wortaufbau besprochen. Der Zeilenumbruch wird im Abschnitt 4.3 behandelt und der Seitenumbruch im Abschnitt 4.4. Zum Schluß betrachten wir die Worttrennung (Abschnitt 4.5). Die einzelnen Aspekte des Formatierungsproblems werden ausführlich in [BK89] behandelt.

In jedem der Abschnitte beschreiben wir zunächst, wie sich das besprochene Phänomen für die Autoren darstellt und wie sie es bei Bedarf beeinflussen können; erst allgemein und dann in *FrameMaker* und LaTeX. In jedem, außer dem ersten Abschnitt folgt dann eine Beschreibung der Verfahren, mit denen die besprochenen Funktionen ausgeführt werden, wobei wir uns im wesentlichen auf die TeX-Lösungen beschränken.

In der Realität ist die Formatierung eine schwierigere Aufgabe, da ein Dokument außer reinem Text im allgemeinen noch Tabellen, Formeln, Abbildungen, Fußnoten usw. enthält. Methoden zum Beschreiben und Setzen von Tabellen und Formeln werden in den beiden folgenden Kapiteln 5 und 6 vorgestellt. Abbildungen sind oft unabhängig vom Text verschiebbar; sie unterliegen dann nur der Bedingung, daß sie sich nicht überholen dürfen und daß sie in der Nähe der Stelle zu finden sein sollen, wo auf sie im Text verwiesen wird. Dadurch wird die Aufgabe des Seitenumbruchs deutlich komplizierter.

4.1 Texteingabe

Bei der Eingabe von reinem Text in ein Dokumentsystem gibt es nur wenig, was ein Autor wissen muß. Da wir uns den Text als in Wörter und Absätze gegliedert vorstellen, muß es eine Möglichkeit geben, diese Gliederung anzuzeigen. Ferner muß es, gerade bei deutschsprachigen Texten, möglich sein, noch gewisse Sonderzeichen wie Umlaute oder „ß“ zu erzeugen, auch wenn man eine amerikanische Tastatur benutzt.

Texteingabe mit *FrameMaker*

Im *FrameMaker* kann Text im Prinzip einfach fortlaufend eingegeben werden. Zeilenwechsel und Trennungen werden dabei nicht explizit von den Autoren ausgelöst, sondern automatisch vom System vorgenommen. Wegen der Interaktivität des *FrameMaker* wird dies sofort sichtbar. Während des Schreibens entstehen oder verschwinden Trennungen, die Wörter rücken zusammen oder auseinander und Zeilenwechsel entstehen.

Die Einteilung in Wörter erfolgt durch ein Leerzeichen. Werden mehrere Leerzeichen in Folge eingegeben, so ändert sich das Aussehen des Textes nach der Eingabe des ersten Leerzeichens nicht mehr. Das Ende von Paragraphen wird mit Hilfe der „Return“-Taste angezeigt.

Problematisch ist das Eingeben von Sonderzeichen wie z. B. der deutschen Umlaute. Dafür sind – wie meist im *FrameMaker* – zwei Methoden vorgesehen: Auswahl mit der Maus oder Eingeben von besonderen Tastenkombinationen.

Man kann auf dem Bildschirm ein Fenster anzeigen lassen, das alle möglichen Sonderzeichen auflistet. Aus dieser Auflistung kann mit der Maus das gewünschte Zeichen kopiert werden.

Die andere Möglichkeit ist, die Zeichen mittels besonderer Tastenkombinationen ohne Benutzung der Maus einzugeben. Die Umlaute können relativ systematisch mit *Control*-`r %a` für „ä“ bis *Control*-`r %U` für „Ü“ eingegeben werden. Das Zeichen „ß“ wird durch *Control*-`q '` erzeugt.

Texteingabe in LaTeX

In LaTeX wird Text einfach in die Dokumentbeschreibung aufgenommen. Die Autoren können den Text in der Beschreibung fast willkürlich durch Zeilenwechsel, Einrückungen und Zwischenräume gliedern. Diese Gliederung wird bis auf einige im folgenden zu beschreibende Ausnahmen vom Formatierer ignoriert, der seine eigenen Zeilenwechsel, Einrückungen und Zwischenräume erzeugt.

Die Einteilung des Textes in Wörter wird normalerweise durch Leerzeichen angezeigt. Dabei ist die Anzahl der Leerzeichen zwischen zwei Wörtern beliebig und

hat auch keinen Einfluß auf die Größe des Wortzwischenraums nach der Formatierung; es muß nur mindestens einer vorhanden sein. Ein einfacher Zeilenwechsel zählt genauso wie ein Leerzeichen.

Die Einteilung in Absätze wird normalerweise durch Leerzeilen angezeigt. Wie bei der Worttrennung ist die Anzahl der Leerzeilen beliebig; es muß nur mindestens eine vorhanden sein, d. h. mindestens zwei Zeilenwechsel direkt hintereinander. Alternativ kann der Beginn eines neuen Absatzes auch durch das Kommando `\par` angezeigt werden, das an beliebiger Position, auch inmitten einer Zeile der Dokumentbeschreibung, vorkommen kann.

Bei der englischen Version von LaTeX werden die Umlaute durch `\"a`, `\"A` usw. erzeugt und das Zeichen „ß“ durch das Kommando `\ss`, dessen Handhabung etwas kompliziert ist. In der deutschen Version ist die Eingabe einfacher: die Umlaute werden durch `"a` usw. erzeugt und „ß“ durch `"s`. Das Wort „Größe“ z. B. kann als `Gr"o"se` eingegeben werden.

4.2 Wortaufbau

4.2.1 Der Wortaufbau aus Autorensicht

Der erste Schritt zur Formatierung eines Textes ist der Aufbau von Wörtern aus Buchstaben. Eine Folge von solchen Wörtern wird dann vom Zeilenumbruchalgorithmus in eine Folge von Zeilen eingeteilt. Dabei bleiben die Wörter im allgemeinen unverändert, außer daß sie womöglich am Zeilenende getrennt werden. Verfahren zur Zeilenausrichtung, die nicht nur die Wortzwischenräume verändern, sondern auch die Abstände der Buchstaben im Innern eines Wortes oder gar die Breite der einzelnen Buchstaben, sind im Computersatz unüblich.

Traditionell wurden bei der Zusammenstellung von Zeichen zu einem Wort manche Gruppen von Zeichen zu einem neuen Zeichen verschmolzen, einer sogenannten *Ligatur*. Typische Ligaturen sind ’fi’ statt ’fi’, ’fl’ statt ’fl’, ’ff’ statt ’ff’, ’ffi’ statt ’ffi’ und ’ffl’ statt ’ffl’. Für diese Beispiele haben wir die Schriftfamilie Times benutzt, denn in der Familie Palatino, in der dieses Buch gesetzt ist, werden Ligaturen nur durch Engerstellen angedeutet: ’fi’ statt ’fi’.

Ligaturen wurden eingeführt, weil die beim Drucken benutzten Metallblöcke leicht brachen, wenn sie, wie bei den Buchstaben ’f’, ’i’ und ’l’, zu schmal waren. Für jede Ligatur wurde ein einzelner, breiter Block benutzt statt mehrerer schmaler. Obwohl dieser Grund heutzutage nicht mehr existiert, werden in manchen Texten immer noch Ligaturen benutzt. Sie sind aber nicht mehr so verbreitet wie früher.

Für deutschsprachige Texte gilt die Konvention, daß in zusammengesetzten Wörtern Ligaturen nicht über die Wortgrenzen der Teilworte hinwegreichen sollten. Statt „Auflage“ mit Ligatur sollte also „Auflage“ ohne Ligatur geschrieben werden.

In LaTeX werden geeignete Buchstabengruppen immer zu Ligaturen zusammengezogen. In der deutschen Version von LaTeX gibt es das Kommando `"|`, das die Ligaturbildung verhindert. Statt einfach `Auflage` sollte also `Auf"|lage` eingegeben werden.

4.2.2 TeX-Kästen: eine Einführung

Wir werden die Realisierung der einzelnen Teilaufgaben der Formatierung anhand des TeX-Systems erklären. Beim Formatieren wandelt das TeX-System das Dokument in eine Darstellung um, in der jede Seite aus einem Kasten besteht, der hierarchisch aus ineinandergeschachtelten Teilkästen aufgebaut ist. Die atomaren Teilkästen, die keine weiteren Teilkästen mehr enthalten, sind entweder mit einem einzelnen Zeichen gefüllt oder leer. Die leeren Kästen dienen als Abstandshalter.

Die Darstellung des Dokuments als eine Ansammlung ineinandergeschachtelter Kästen existiert nur intern in TeX. Sie wird schließlich von TeX in die Seitenbeschreibungssprache *dvi* übersetzt, die als Ausgabesprache von TeX dient (siehe Kapitel 9, insbesondere Abschnitt 9.3 und 9.4).

In TeX ist jeder *Kasten* (englisch *box*) rechteckig mit Kanten, die zu den Kanten der Seiten parallel sind. Wir werden diese Kästen in Abschnitt 6.4.3 genau beschreiben. Im Moment interessiert uns nur, daß jeder Kasten eine Breite w besitzt. Die Breite bezeichnet den Abstand des linken von dem rechten Rand des Kastens. Sie kann auch negativ sein. Das ist z. B. bei Kästen der Fall, die aus Kommandos zum Zurücksetzen hervorgehen.

Beim Wortaufbau und später bei der Zusammenstellung von Zeilen aus Wörtern werden Kästen horizontal nebeneinander gestellt und dann zu einem neuen umfassenden Kasten zusammengepackt, der die ursprünglichen Kästen als Teilkästen enthält. Dabei ist die Breite des neuen Kastens gleich der Summe der Breiten der Teilkästen, d. h. die Teilkästen werden ohne Zwischenraum nebeneinandergesetzt.

4.2.3 Wortaufbau in TeX

Beim Aufbau eines Wortkastens aus den einzelnen Buchstaben des Wortes werden zunächst geeignete Buchstabengruppen zu Ligaturen zusammengezogen. Diese Ligaturen zählen fortan als Einzelzeichen. Dann wird jedes Zeichen in einen Elementarkasten eingeschlossen, dessen Ausmaße TeX einer Tabelle entnimmt. Diese Elementarkästen werden durch horizontale Kombination zu einem Wortkasten zusammengestellt. Dabei gibt es eine Besonderheit: Zwischen manche Paare von Zeichen, die ihren Elementarkasten nicht voll ausfüllen, wird ein negativer Zwischenraum (ein Kasten mit Breite $w < 0$) eingefügt, um die Zeichen näher aneinanderzurücken. Dieses Verfahren heißt *Unterschnitt*. Vergleiche dazu das mit Unterschnitt gesetzte Ve mit Ve, wo der Unterschnitt künstlich verhindert worden ist.

4.3 Zeilenumbruch

In diesem Abschnitt wird das Wort „Zeilenumbruch“ in mehreren Bedeutungen benutzt. Einmal steht es für das Zeilenumbruch*problem*; dann für das Zeilenumbruch*verfahren*, außerdem für das davon erzeugte Ergebnis („der Zeilenumbruch“) und manchmal noch für die Stelle, an der eine neue Zeile beginnt. Die jeweilige Bedeutung geht aber aus dem Kontext hervor.

4.3.1 Zeilenumbruch aus Autorensicht

Der Vorgang des Zeilenumbruchs besteht logisch gesehen aus zwei Schritten, die nacheinander oder verschränkt ablaufen:

1. Die vorgegebene Folge von Wörtern muß so in Zeilen eingeteilt werden, daß die entstehenden Zeilen in etwa gleich lang sind und den zur Verfügung stehenden Raum optimal ausnutzen.
2. Die einzelnen Zeilen müssen im Seitenraum positioniert und dazu eventuell nachbehandelt werden.

Die Stellen des Zeilenumbruchs

Zu Punkt 1 ist zu sagen, daß der Zeilenumbruch bevorzugt zwischen Wörtern erfolgt. Es gibt natürlich auch die Möglichkeit, ein langes Wort zu trennen (siehe Abschnitt 4.5). Weiterhin können die Autoren in vielen Systemen Wortzwischenräume, an denen ein Zeilenumbruch unerwünscht ist, als untrennbar bezeichnen. Ein solcher Zwischenraum ist z. B. bei „Bild 3“ der Abstand zwischen dem Wort „Bild“ und der Zahl „3“. Für den Fall, daß der vom System gewählte Zeilenumbruch den Autoren nicht gefällt, bieten viele Systeme noch weitere Möglichkeiten, Zeilenumbrüche an gewissen Stellen mehr oder weniger zwingend vorzuschlagen oder zu verbieten.

Beeinflussung des Zeilenumbruchs im *FrameMaker*

Im *FrameMaker* wird ein gewöhnlicher Wortzwischenraum durch Druck auf die Leertaste eingegeben, während ein Zwischenraum, an dem der Zeilenwechsel verboten ist, durch gleichzeitiges Drücken der Control-Taste und der Leertaste erzeugt wird. Wenn es gewünscht wird, wird so ein spezieller Zwischenraum auf dem Bildschirm (aber nicht im Ausdruck) durch ein ’␣’ bezeichnet.

Die Tastenkombination Meta-Return (gleichzeitiger Druck auf die „Meta-Taste“, die je nach Tastatur verschieden beschriftet ist, und die Return-Taste) erzwingt einen Zeilenumbruch an der Einfügestelle.

Beeinflussung des Zeilenumbruchs in LaTeX

Wie bereits erwähnt, werden in LaTeX gewöhnliche Wortzwischenräume durch Leerzeichen dargestellt. Nichttrennbare Zwischenräume werden in der Dokumentbeschreibung als ~ geschrieben, z. B. wird `Abbildung~1` zu „Abbildung 1“.

Eine andere Methode zur Verhinderung des Zeilenumbruchs ist die Klammerung eines Textteils durch das Kommando `\mbox{...}`. Der in der Klammer stehende Text wird weder an Wortzwischenräumen noch innerhalb von Wörtern umbrochen.

Der Befehl `\linebreak[`*Zahl*`]` fordert einen Zeilenumbruch an einer bestimmten Stelle. Für die Zahl gibt es nur die Möglichkeiten 0, 1, 2, 3 und 4. Je höher die Zahl, desto dringlicher wird die Empfehlung zum Zeilenumbruch. Ein Wert von 4 erzwingt den Umbruch an der Stelle, wo das Kommando vorkommt. Die Form `\linebreak` ohne Zahl ist gleichbedeutend mit `\linebreak[4]`.

Die `\linebreak`-Befehle ändern nichts an der Zeilenausrichtung bzw. dem Randausgleich. Sie machen also nur Sinn, wenn sie in der Nähe der Stelle liegen, die LaTeX ohne Einflußnahme der Autoren als Umbruchstelle wählen würde. Bei unvorsichtigem Gebrauch von `\linebreak` entstehen überdehnte Zeilen mit riesigen Wortabständen.

Es gibt auch den Befehl `\nolinebreak[`*Zahl*`]`, der von einem Zeilenumbruch an einer bestimmten Stelle abrät. Die Zahl kann ebenfalls Werte zwischen 0 und 4 annehmen. Beim Wert 4 wird der Umbruch vollständig verhindert. Die Zahl 4 darf auch weggelassen werden.

Zeilenausrichtung

Zur Ausrichtung der Zeilen im Seitenraum gibt es im wesentlichen die vier Möglichkeiten, die im folgenden beschrieben werden, wobei der Text der Beschreibung jeweils so gesetzt ist, daß er die beschriebene Ausrichtung verdeutlicht.

- Linksbündig:

  Alle Zeilen beginnen am linken Rand, so daß die Zeilenanfänge genau übereinander liegen. Die Zeilen sind verschieden lang; die Zeilenenden liegen nicht übereinander. Klassischer Schreibmaschinentext sieht normalerweise so aus.

- Rechtsbündig:

  Diese Art der Zeilenanordnung ist spiegelbildlich zur linksbündigen Ausrichtung. Die Zeilen beginnen an verschiedenen horizontalen Positionen, aber alle Zeilenenden liegen genau übereinander.

- Zentriert:

  Alle Zeilen sind so angeordnet, daß sie genau in der Mitte des zur Verfügung stehenden Raums liegen, d. h. in jeder Zeile ist der leere Raum links genau so groß wie der leere Raum rechts.

- Ausrichtung mit beidseitigem Randausgleich:

  Wenn Zeilen mit Randausgleich gesetzt werden, dann liegen sowohl die Zeilenanfänge wie auch die Zeilenenden genau übereinander, außer daß die letzte Zeile eines Abschnitts normalerweise kürzer ist und die erste Zeile oft eingerückt ist. Das vorliegende Buch ist mit Randausgleich gesetzt.

  Im Gegensatz zu den drei anderen Ausrichtungen können die Zeilen bei Randausgleich nicht ihre „natürliche Länge“ behalten, sondern müssen etwas gedehnt oder gestaucht werden, um exakt gleich lang zu werden.

Im *FrameMaker* gehört die Ausrichtung zum Paragraphenformat. Sie kann daher durch Wahl eines geeigneten Formats festgelegt werden. Eine andere Möglichkeit ist, durch ein spezielles Menü die Ausrichtung eines schon existierenden Paragraphen zu ändern.

In LaTeX ist die Ausrichtung mit Zeilenausgleich der Normalfall. Für die drei anderen Arten gibt es eigene Umgebungen, nämlich `flushleft`, `flushright` und `center`. Linksbündiger Text wird also durch `\begin{flushleft}` und `\end{flushleft}` abgegrenzt.

4.3.2 Der Zeilenumbruchalgorithmus von TeX

In diesem Abschnitt werden wir den Zeilenumbruchalgorithmus von TeX in etwas vereinfachter und idealisierter Form vorstellen (siehe auch [KP81, PK82]). Da LaTeX mit Hilfe von Makros in TeX implementiert ist, wird dieser Algorithmus auch beim Formatieren in LaTeX benutzt. Trotzdem kann sich bei der Benutzung von LaTeX ein anderer Umbruch ergeben als bei der Benutzung von reinem TeX, da der Zeilenumbruch von verschiedenen Parametern (globalen Variablen) abhängt, die in LaTeX teilweise anders gesetzt sind als in reinem TeX.

Grob betrachtet, erzeugt der Algorithmus in einem ersten Schritt aus einer Folge von „Elementen“ eine Folge etwa gleichlanger Zeilen. Im Falle des beidseitigen Randausgleichs werden dann in einem zweiten Schritt die Zeilen etwas gedehnt oder gestaucht, um ihnen exakt die gleiche Länge zu geben.

Die „Elemente“ müssen natürlich alle für den Algorithmus notwendigen Informationen beinhalten: den Inhalt der Zeile, die möglichen Umbruchstellen sowie Dehn- und Schrumpfmöglichkeiten für den beidseitigen Randausgleich. Demnach gibt es drei Arten von Elementen: Inhaltskästen, Strafelemente und Kleber.

Inhaltskästen

Die Inhaltskästen sind die Träger des eigentlichen Inhalts des zu formatierenden Absatzes. Wenn der Absatz einfach nur aus Text besteht, dann entsprechen die Inhaltskästen den Stücken von Wörtern zwischen je zwei Trennstellen.

Für den Zeilenumbruchalgorithmus ist jeder Inhaltskasten ein atomares Objekt, das bereits fertig gesetzt ist und das er nicht mehr nachträglich verändern kann. Daher interessiert er sich gar nicht für den Inhalt eines solchen Kastens, sondern nur für seine *Breite*, die ihm angibt, wieviel Platz der Kasten in einer Zeile einnehmen wird. Diese Breite kann eine beliebige reelle Zahl sein.

Kleber

Außer Wortstücken werden die fertigen Zeilen auch Wortzwischenräume enthalten. Im Gegensatz zu den Inhaltskästen haben die Zwischenräume keine feste Breite, sondern können im Falle des beidseitigen Randausgleichs gedehnt oder gestaucht werden. Sie heißen in TEX (horizontaler) *Kleber* (englisch *glue*).

Jedes Stück Kleber hat drei Attribute, seine *natürliche Größe*, seine *Dehnbarkeit* und seine *Stauchbarkeit* Die natürliche Größe wird in Zeilen ohne Randausgleich angenommen. (Diese Objekte entsprechen also eher Spiralfedern als Leimtropfen. Eine Spiralfeder hat eine natürliche Länge, kann sich unter Zug dehnen oder unter Druck verkürzen.)

Beispiele:

- Ein Kleber mit der Beschreibung „3 pt plus 2 pt minus 1 pt“ hat eine natürliche Größe von 3 pt, kann um 2 pt wachsen und um 1 pt schrumpfen. Seine Mindestgröße ist also 2 pt und seine Maximalgröße 5 pt.
- Der normale Wortzwischenraum hat in TEX bei einer 10 pt-Schrift die Attribute 3,33333 pt plus 1,66666 pt minus 1,11111 pt.
- Bei einem Kleber nach einem Komma sind die Attribute anders: 3,33333 pt plus 2,08331 pt minus 0,88889 pt.
- In der englischen Version von TEX ist der Kleber nach einem Punkt besonders groß: 4,44444 pt plus 4,99997 pt minus 0,37036 pt.

Normalerweise werden die Kleberelemente nach den oben angedeuteten Regeln von TEX selbst erzeugt. Die angegebenen Zahlenwerte werden aus globalen Variablen ausgelesen, die auch umdefiniert werden können. Außerdem ist es möglich, auch explizit Kleber in eine Zeile einzufügen.

Strafelemente

Um Informationen über von den Autoren erzwungene, gewünschte, unerwünschte oder verbotene Umbruchstellen an den Algorithmus übergeben zu können, gibt

es *Strafelemente*, die im fertigen Layout unsichtbar sind, aber Informationen über die Qualität von Umbruchstellen tragen. Ebenso gibt es für jede mit einer Worttrennung verbundene Umbruchstelle ein Strafelement, um unnötige Trennungen zu vermeiden.

Das Hauptattribut eines Strafelements ist eine Zahl p (penalty), die die „Bestrafung“ für einen Zeilenumbruch an dieser Stelle angibt. Je höher p ist, desto schlimmer ist ein Umbruch. Ein Wert von $p = \infty$ verhindert also einen Umbruch völlig, während ein Wert von $p = -\infty$ einen Umbruch an dieser Stelle erzwingt. (In TEX wird ein Wert von 10000 oder mehr als ∞ angesehen, und entsprechend ein Wert ≤ -10000 als $-\infty$.)

Jedes Strafelement hat auch eine Breite. Sie gibt an, welcher Zusatzplatz benötigt wird, wenn an dieser Stelle umbrochen wird, z. B. durch einen Bindestrich. Zusätzlich besitzt jedes Strafelement ein Flag, das angibt, ob das Element von einer potentiellen Worttrennstelle herrührt. Es wird benutzt, um eine Reihung von mehreren aufeinanderfolgenden Zeilen mit Worttrennung am Ende möglichst zu vermeiden.

Eingabe in den Zeilenumbruchalgorithmus

Der Algorithmus erhält als Eingabe eine Folge von Elementen $x_1, x_2, \ldots, x_m$. Ein Element x_i kann sein:

- ein Inhaltskasten mit Breite w_i;
- Kleber mit natürlicher Breite w_i, Dehnbarkeit y_i und Stauchbarkeit z_i;
- ein Strafelement mit Strafe p_i, Breite w_i und Flag f_i.

Die Eingabe $x_1, x_2, \ldots, x_m$ entsteht auf die folgende Weise:

- Wenn die erste Zeile des Paragraphen eingerückt werden soll, dann ist das erste Element x_1 ein leerer Inhaltskasten, dessen Breite w_1 der Einrücktiefe entspricht. Es muß sich um einen Inhaltskasten handeln, nicht um Kleber fester Größe, weil Kleber am Zeilenanfang gelöscht wird, wie wir sehen werden.
- Jedes Wort ergibt eine Folge von Inhaltskästen und Strafelementen. Die Inhaltskästen enthalten die nicht mehr weiter trennbaren Bestandteile des Wortes. Ihre Breite ergibt sich aus ihrem Inhalt gemäß der Beschreibung des aktuellen Fonts, wobei Unterschneidungen berücksichtigt werden. Zwischen ihnen befinden sich Strafelemente, die den möglichen Trennstellen entsprechen. (Die Trennstellen werden vorher nach Verfahren ermittelt, die im Abschnitt 4.5 vorgestellt werden.)
- Zwischen je zwei Wörtern befindet sich ein Kleberelement. Seine Attribute ergeben sich aus der Größe des aktuellen Fonts und der Art seiner Umgebung. Wie oben angedeutet, werden „reine“ Wörter von Kleber anderer Art gefolgt als Kommas oder Punkte.

- Am Ende des Paragraphen werden drei besondere Elemente eingefügt. Das erste ist ein Strafelement x_{m-2} mit $p_{m-2} = \infty$. Das zweite ist ein Kleberelement x_{m-1} mit natürlicher Breite $w_{m-1} = 0$, Dehnbarkeit $y_{m-1} = \infty$ und Stauchbarkeit $z_{m-1} = 0$. Es erzeugt den leeren Raum am Ende der letzten Zeile des Paragraphen. Das Strafelement davor verhindert einen Zeilenumbruch direkt vor dem leeren Raum. Ganz zum Schluß folgt noch ein Strafelement x_m mit $p_m = -\infty$, das den Zeilenumbruch am Paragraphenende erzwingt. Die Einfügung dieser drei Schlußelemente erspart es dem Algorithmus, die letzte Zeile besonders zu behandeln.

Kerben

Die Folge der Elemente $x_1, x_2, \ldots, x_m$ kann nicht an beliebigen Stellen umbrochen werden. Eine *legale Bruchstelle* oder *Kerbe* befindet sich an folgenden Stellen:

- hinter einem Strafelement x_b mit $p_b < \infty$, oder
- zwischen einem Inhaltskasten x_b und einem Kleberelement x_{b+1}.

Die Zeilentrennung erfolgt also lokal so, daß Strafelemente an die Zeilenenden und Kleber an die Zeilenanfänge gelangen. Eine Kerbe wird durch den Index b des Elements charakterisiert, das direkt vor der Kerbe liegt. Dieses b ist also eine Zahl mit $1 \leq b \leq m$.

Selbstverständlich wird nicht an jeder Kerbe eine neue Zeile begonnen. Obligatorisch sind nur die Kerben b nach Strafelementen x_b mit $p_b = -\infty$, insbesondere also die Kerbe m nach dem Schlußelement x_m. Der Zeilenumbruchalgorithmus wählt eine Folge $b_1 < b_2 < \cdots < b_k$ von Kerben aus, die alle obligatorischen Kerben enthält; insbesondere gilt also $b_k = m$. Daher teilen die ausgewählten Kerben den Paragraphen in k Zeilen ein.

Bevor wir schildern, wie der Algorithmus die Zeileneinteilung findet, werden wir beschreiben, wie die erzeugten Zeilen weiter bearbeitet werden.

Weiterbearbeitung der erzeugten Zeilen

Eine frisch erzeugte Zeile liegt als eine Folge von Elementen vor. Da Wortzwischenräume, die genau an ein Zeilenende fallen, im Druckbild nicht sichtbar sein sollen, wird zunächst ein eventuell am Zeilenanfang liegendes Kleberelement gelöscht. (Am Zeilenende können nach Definition der Kerben keine Kleberelemente liegen.)

Die Strafelemente im Inneren der Zeile sind im Druck unsichtbar. Daher werden ihre Breiten auf 0 gesetzt. Nur ein eventuelles Strafelement am Ende der Zeile behält seine Breite, wenn es aufgrund der Zeilentrennung sichtbar wird, z. B. als Bindestrich.

Nun sind wir soweit, daß wir die natürliche Länge W der Zeile bestimmen können. Sie ergibt sich als die Summe der Breiten w_i aller Elemente in der Zeile.

Sei T die Breite des Textes auf einer Seite. Bei beidseitigem Randausgleich ist T die gewünschte Länge der Zeile, während sie bei den anderen drei Ausrichtungsarten die Maximallänge darstellt.

Nehmen wir zunächst an, daß die erzeugte Zeile zu kurz ist ($W < T$). Dann müssen die folgenden Maßnahmen ergriffen werden:

- Bei linksbündiger Ausrichtung wird die Zeile einfach an den linken Rand des Textraums gesetzt.
- Bei rechtsbündiger Ausrichtung wird die Zeile an einen Punkt gesetzt, der in einer Entfernung von $T - W$ rechts vom linken Textrand liegt.
- Bei Zentrierung liegt der Zeilenanfang $(T - W)/2$ rechts vom linken Textrand.
- Bei doppelseitiger Ausrichtung muß die Zeile auf die Länge T gestreckt werden.

Wenn die erzeugte Zeile zu lang ist ($W > T$), dann muß sie beim doppelseitigen Randausgleich auf die Länge T gestaucht werden. Bei den anderen Ausrichtungsarten kann sie ebenfalls gestaucht werden, oder man läßt sie über den Textrand überhängen.

Dehnen und Stauchen von Zeilen

Betrachten wir eine Zeile der natürlichen Länge W, die auf die vorgegebene Länge T gebracht werden soll. Dazu wird ein *Anpassungsverhältnis* r berechnet.

Falls $W = T$, dann ist natürlich $r = 0$, und die Kleberelemente behalten ihre natürliche Breite, d. h. ihre *berechnete Breite* w_i' ergibt sich als $w_i' = w_i$.

Falls $W < T$, dann wird die *totale Dehnbarkeit* Y der Zeile als Summe der Dehnbarkeiten y_i aller Kleberelemente der Zeile berechnet. Das Anpassungsverhältnis ist dann $r = (T - W)/Y > 0$. Jedes einzelne Kleberelement wird proportional zu seiner Dehnbarkeit gestreckt, d. h. es erhält die berechnete Breite $w_i' = w_i + r \cdot y_i$. Es wächst also um $r \cdot y_i$, d. h. der Gesamtzuwachs wird wie verlangt $r \cdot Y = T - W$.

Falls $W > T$, dann wird analog die *totale Schrumpfbarkeit* Z als Summe der Schrumpfbarkeiten z_i aller Kleberelemente der Zeile berechnet. Das Anpassungsverhältnis ist in diesem Falle $r = (T - W)/Z < 0$. Jedes einzelne Kleberelement wird proportional zu seiner Schrumpfbarkeit gestaucht, d. h. seine berechnete Breite wird $w_i' = w_i + r \cdot z_i$.

In der obigen Beschreibung haben wir einige unangenehme Sonderfälle vernachlässigt. Wenn $W < T$ ist und $Y = 0$, dann wird formal $r = \infty$. Die betrachtete Zeile ist dann zu kurz, aber überhaupt nicht dehnbar. Ähnlich problematisch ist der Fall $W > T$ und $Z = 0$, d. h. $r = -\infty$.

Während die beiden eben beschriebenen Situationen als Fehlerfälle anzusehen sind, tritt zumindest in der letzten Zeile eines Paragraphen regelmäßig der Fall unendlicher Dehnbarkeit $Y = \infty$ auf. Das Vorgehen ist in diesem Falle wie folgt: alle Kleberelemente mit endlicher Dehnbarkeit y_i behalten ihre natürliche Breite. Die zu erzielende Dehnung $T - W$ wird gleichmäßig auf alle Kleberelemente unendlicher Dehnbarkeit verteilt. Der unendlich dehnbare Kleber am Ende der letzten Zeile erhält also im Normalfall (d. h. wenn kein anderer unendlich dehnbarer Kleber in der Zeile ist) die Breite $T - W$, während alle anderen Wortzwischenräume in der letzten Zeile ihre natürliche Breite haben.

In Wahrheit ist das Verhalten von TEX viel komplizierter, da es eine dreistufige Hierarchie von unendlichen Dehnbarkeiten gibt, die auch noch mit Vorfaktoren versehen werden können. Diese komplizierte Struktur erlaubt es versierten TEX-Programmierern, auch exotische Zeilenausrichtungen zu definieren.

Die Bewertung potentieller Zeilen

Der Zeilenumbruchalgorithmus braucht eine Methode, um die Qualität potentieller Zeilen zu bewerten, damit er einen möglichst guten Umbruch des ganzen Paragraphen erzeugen kann.

Einen ersten Anhaltspunkt zur Bewertung bietet das Anpassungsverhältnis r. Im Falle $0 < r \leq 1$ werden wegen der Beziehung $w_i' = w_i + r \cdot y_i$ alle Kleberelemente der Zeile um $r \cdot y_i \leq y_i$ gedehnt, also um nicht mehr als ihre maximale Dehnbarkeit y_i. Ähnliches gilt für das Schrumpfen bei $-1 \leq r < 0$.

Wenn also $|r| \leq 1$, dann werden alle Kleberelemente der Zeile nicht mehr verändert, als ihr zugelassener Spielraum angibt. Bei $r > 1$ dagegen werden sie überdehnt; es ergibt sich eine „lose“ Zeile mit zu viel freiem Platz zwischen den Wörtern. Bei $r < -1$ entsteht eine „enge“ Zeile mit zu wenig Platz zwischen den Wörtern.

Die Zahl $|r|$ ist also ein erster Ansatz für ein Qualitätsmaß im umgekehrten Sinne: je höher diese Zahl, desto schlechter sieht die Zeile aus. Man kann sich $|r|$ also als die Anzahl von Strafpunkten vorstellen, die mit dieser Zeile verbunden sind.

In TEX wurde dieser Ansatz ausgebaut; es gibt eine *badness*-Funktion, die für jede Zeile die Anzahl der Strafpunkte festlegt. Der Umbruchalgorithmus muß die Einteilung in Zeilen so wählen, daß die Summe ihrer Strafpunkte minimal wird (unter der Nebenbedingung, daß alle obligatorischen Trennstellen beachtet werden).

Die Definition der Strafpunktfunktion ist kompliziert. Die Strafe für die j-te Zeile errechnet sich in den folgenden Schritten:

1. Zunächst wird der Anpassungsfaktor r_j der Zeile berechnet.
2. Aus r_j wird eine Zahl β_j wie folgt bestimmt:

$$\beta_j = \begin{cases} \infty & \text{falls } r_j = \infty \text{ oder } r_j < -1 \\ 100\,|r_j|^3 & \text{sonst} \end{cases}$$

Übermäßiges Schrumpfen ($r_j < -1$) wird also völlig verboten, während Überdehnung ($r_j > 1$) im Prinzip erlaubt ist, aber wegen des kubischen Wachstums der Funktion unter schwere Strafe gestellt wird.

3. Bisher wurden die Strafelemente noch nicht berücksichtigt. Wenn die Zeile mit einem Strafelement x_i endet, dann liefert dies einen Beitrag $\pi_j = p_i$ zur Strafe für die Zeile. Ansonsten ist $\pi_j = 0$. Die beiden Zahlen β_j und π_j werden wie folgt verknüpft:

$$\delta_j = \begin{cases} (l+\beta_j)^2 + \pi_j^2 & \text{falls } \pi_j \geq 0 \\ (l+\beta_j)^2 - \pi_j^2 & \text{falls } -\infty < \pi_j < 0 \\ (l+\beta_j)^2 & \text{falls } \pi_j = -\infty \end{cases}$$

   Dabei ist l der Wert einer globalen Variable namens `linepenalty`, deren Erhöhung TEX dazu bringt, Umbrüche mit weniger Zeilen zu bevorzugen.

   Der dritte Fall der Definition von δ_j erscheint etwas eigenartig, da er höhere Strafen liefert als der zweite Fall, obwohl π_j geringer ist. Zu bedenken ist aber, daß ein Strafelement mit $p = -\infty$ eine obligatorische Trennstelle ist, d. h. auf jeden Fall an dieser Stelle getrennt wird, egal wie hoch die Strafe ist. Die Bewertung der Zeile vor dieser obligatorischen Trennstelle ist trotzdem nicht sinnlos, da sie die Wahl des Anfangs dieser Zeile beeinflussen kann.

4. Zu der Zahl δ_j werden in gewissen Fällen zusätzliche Strafen addiert, deren Höhe von globalen Variablen geregelt wird, nämlich

`adjdemerits`,	wenn die Zeile sich von der vorhergehenden in der Dichte zu sehr unterscheidet,
`doublehyphendemerits`,	wenn sowohl die Zeile als auch die vorhergehende mit einer Trennung endet,
`finalhyphendemerits`	in der letzten Zeile des Paragraphen, wenn die vorletzte mit einer Trennung endet.

Durch Umdefinieren dieser und anderer noch nicht erwähnter globaler Variablen ist es möglich, die strategischen Ziele des Zeilenumbruchs zu beeinflussen.

Der eigentliche Zeilenumbruch

Im folgenden nehmen wir an, daß die zu formatierende Elementfolge nur noch ein Strafelement mit $p = -\infty$ enthält und zwar ganz am Ende. Dies kann erreicht werden, indem alle durch eingebettete „$-\infty$“-Strafelemente erzwungenen Zeilenumbrüche sofort durchgeführt werden. Die entstehenden Teile des Paragraphen können unabhängig voneinander bearbeitet werden.

Das Problem des Zeilenumbruchs lautet dann folgendermaßen: Gegeben sei eine Folge von Elementen, finde eine Einteilung in Zeilen, so daß die Summe der Strafpunkte der Einzelzeilen minimal ist. In Abschnitt 4.6 werden wir sehen, daß dieses

Problem sich auf das Problem zurückführen läßt, einen kürzesten Weg in einem kantenmarkierten Graphen zu finden. Hier werden wir jedoch das Problem direkt lösen.

Um einen optimalen Zeilenumbruch zu finden, müssen nicht alle Möglichkeiten durchprobiert werden, denn es gilt das folgende Prinzip: Wenn die ersten n Zeilen optimal gesetzt sind und die letzte Zeile davon weggelassen wird, dann entsteht ein optimaler Zeilenumbruch der ersten $n-1$ Zeilen. Im Umkehrschluß braucht man sich also nur alle optimalen Umbrüche der ersten $n-1$ Zeilen zu merken, um daraus einen optimalen Umbruch der ersten n Zeilen zu machen; die nichtoptimalen Umbrüche der ersten $n-1$ Zeilen brauchen nicht mehr weiter betrachtet zu werden.

Die Anzahl der Möglichkeiten, die untersucht werden müssen, wird dadurch reduziert, daß gewisse Zeileneinteilungen gleich als so schlecht erkannt werden, daß sie nicht mehr mit anderen verglichen werden müssen, sondern sofort ausscheiden.

Dazu gibt es eine durch eine Variable gegebene Toleranzgrenze. Wenn die Strafe für eine bestimmte Zeile die Toleranzgrenze überschreitet, dann ist sie nicht mehr akzeptabel. Eine Zeileneinteilung für ein Anfangstück des Paragraphen heißt *akzeptabel*, wenn jede einzelne Zeile akzeptabel ist. Eine Kerbe heißt *vernünftig*, wenn es eine akzeptable Zeileneinteilung gibt, die mit dieser Kerbe endet.

Die ersten paar Kerben in einem Paragraphen sind also normalerweise nicht vernünftig, weil die Zeile bis dahin viel zu kurz wäre. Wenn etwa die gewünschte Zeilenlänge erreicht ist, dann folgt im allgemeinen eine kleine Anzahl vernünftiger Kerben. Die nächsten Kerben sind es dann meist wieder nicht, da bis zu ihnen zu viel Material für eine und zu wenig für zwei Zeilen vorliegt.

Die Vernünftigkeit von Kerben kann auch inkrementell charakterisiert werden: eine Kerbe b ist vernünftig, wenn es eine vernünftige Kerbe a davor gibt, so daß die durch a und b begrenzte Zeile akzeptabel ist. Aufgrund dieser Charakterisierung kann der Umbruchalgorithmus relativ effizient alle vernünftigen Kerben ermitteln.

Der folgende Algorithmus findet alle vernünftigen Kerben b, merkt sich für jede die Strafpunkte $S(b)$ für einen optimalen Umbruch bis b und die Vorgängerkerbe $V(b)$, die in einem der optimalen Umbrüche bis b die letzte Zeile vor b beendet.

Algorithmus:
(0) Der Anfang des Paragraphen ist eine vernünftige Kerbe mit $S = 0$
    und undefiniertem V.
(1) Laufe vorwärts durch den Paragraphen.
(2) Für jede Kerbe b:
(3) Laufe von b aus rückwärts.
(4) Finde die Menge A aller vernünftigen Kerben a,
(5) so daß die Zeile von a bis b akzeptabel ist mit Strafpunkten s_{ab}.
(6) Wenn A nicht leer ist:
(7) Markiere b als vernünftig.

(8) Definiere die Strafpunkte von b als $S(b) = \min_{a \in A}(S(a) + s_{ab})$.
(9) Definiere $V(b)$ als eins der a aus A, die das Minimum liefern.

Am Ende gibt es zwei Fälle: Die Kerbe k am Ende des Paragraphen ist nicht vernünftig; dann gibt es keinen akzeptablen Umbruch. Oder sie ist es; dann können die Umbruchstellen für einen optimalen Umbruch gefunden werden, und zwar von hinten nach vorne als $V(k)$, $V(V(k))$ usw.

Beachten Sie, daß beim Rückwärtslaufen in Zeile (3) nicht bis zum Anfang zurückgelaufen werden muß, sondern nur so weit, bis das Zwischenstück zu lang geworden ist, um akzeptabel zu sein. Die Anzahl der Elemente, die bis dahin betrachtet werden, entspricht etwa der durchschnittlichen Anzahl von Silben (untrennbaren Wortteilen) pro Zeilenlänge. Also ist der Gesamtaufwand des Algorithmus ungefähr proportional zu dem Produkt | Silben / Zeile | · | Kerben im Paragraph |.

In Wirklichkeit arbeitet TEX zweistufig. Es gibt zwei Konstanten: `pretolerance` < `tolerance`. Zunächst sucht TEX nach einem optimalen Umbruch ohne Worttrennung (wobei also Kerben nur zwischen Wörtern liegen), in dem eine Zeile akzeptabel ist, wenn die Zahl ihrer Strafpunkte ≤ `pretolerance` ist. Wenn es keinen akzeptablen Umbruch dieser Art gibt, wird nach dem optimalen mit Worttrennung gesucht (wobei also Kerben auch an allen erlaubten Trennstellen liegen), in dem die Akzeptanzschranke gleich `tolerance` ist.

Bei englischen Texten sind Umbrüche oft ohne Trennung möglich, da englische Wörter relativ kurz sind. Oft genügt also der erste Durchlauf des Algorithmus, der effizienter ist als der zweite, weil er weniger Kerben hat, und bessere Ergebnisse liefert, weil er Trennungen völlig vermeidet.

4.4 Seitenumbruch

4.4.1 Seitenumbruch aus Autorensicht

Die Aufgabe des Seitenumbruchs ist, die vom Zeilenumbruch erzeugten Zeilen zusammen mit anders entstandenem Material (Formeln, Tabellen, Abbildungen und Fußnoten) möglichst geschickt in Seiten einzuteilen. Das Ziel ist dabei eine gleichmäßige Füllung der Seiten, d. h. ein ausgefüllter Satzspiegel und die gleichmäßige Verteilung von vertikalem Zwischenraum auf den verschiedenen Seiten.

Im Gegensatz zu der Situation beim Zeilenumbruch gibt es zwei Arten von Objekten, die beim Seitenumbruch bearbeitet werden müssen. Die Zeilen des normalen Textes sind in einer festen Reihenfolge angeordnet und machen normalerweise den Hauptteil des Seiteninhalts aus. Dazu kommen *Gleitobjekte* wie die Abbildungen in diesem Buch, die unabhängig vom Text an eine herausragende Position gelegt werden, oft oben auf einer Seite.

Seitenumbruch und Gleitobjekte im *FrameMaker*

Im *FrameMaker* gehört zu den Bestandteilen eines Paragraphenformats, das die graphische Erscheinungsform von Paragraphen regelt, auch eine Gruppe von Eigenschaften, die den Seitenumbruch beeinflussen.

- Es kann definiert werden, wo der Paragraph beginnen darf: überall, nur oben auf der Seite, nur oben in einer Spalte, nur auf einer linken Seite oder nur auf einer rechten Seite. Damit kann z. B. festgelegt werden, daß ein Kapitel nur oben auf einer neuen Seite beginnen darf.
- Man kann fordern, daß der Anfang des Paragraphen mit dem Ende seines Vorgängers und / oder das Ende des Paragraphen mit dem Anfang seines Nachfolgers auf derselben Seite stehen muß. Damit wird verhindert, daß der Paragraph ganz oben, ganz unten oder ganz alleine auf einer Seite steht. So ist es z. B. (auf zwei verschiedene Arten) möglich zu erzwingen, daß eine Kapitelüberschrift und der Anfang des ersten Abschnitts des Kapitels auf derselben Seite stehen.
- Die Mindestanzahl von Zeilen eines Paragraphenteils unten und oben auf einer Seite kann festgelegt werden. Damit können unter anderem die sogenannten Schusterjungen und Hurenkinder vermieden werden. Ein *Schusterjunge*, auch *Waise* (englisch *orphan*) genannt, ist die erste Zeile eines Paragraphen, die alleine unten auf einer Seite erscheint. Ein *Hurenkind*, auch *Witwe* (englisch *widow*) genannt, ist die letzte Zeile eines Paragraphen, die einsam oben auf einer neuen Seite steht.

Die oben angegebenen Merkmale lassen sich auch unabhängig vom Paragraphenformat für einen einzelnen Paragraphen definieren. Damit lassen sich unerwünschte Seitenumbrüche verhindern.

Gleitobjekte wie z. B. Abbildungen lassen sich im *FrameMaker* auf mehrere Arten behandeln. Ein Gleitobjekt kann völlig unabhängig vom Text an einer bestimmten Stelle fixiert sein. Änderungen des Textes beeinflussen seine Position dann nicht. Das andere Extrem besteht darin, das Gleitobjekt in den Text einzubetten. Es wandert dann bei jeder Texteinfügung oder -löschung entsprechend mit (und ist genaugenommen gar kein Gleitobjekt mehr).

Als Mittelweg kann ein Gleitobjekt mit dem Text durch einen *Anker* lose verknüpft sein. Der Anker ist in den Text eingebettet und wird bei Textveränderungen entsprechend verschoben. Das Gleitobjekt befindet sich an einer hervorgehobenen Position in der Nähe des Ankers, z. B. am Oberrand der Seite, auf der der Anker liegt. Dadurch wechselt es die Position nur, wenn der Anker durch eine Textänderung auf eine andere Seite verschoben wird.

Seitenumbruch und Abbildungen in LaTeX

In LaTeX gibt es die Befehle `\pagebreak[`*Zahl*`]` und `\nopagebreak[`*Zahl*`]`, die zu `\linebreak` und `\nolinebreak` analog sind. Außerdem gibt es eine Umgebung

`\begin{samepage}` ... `\end{samepage}`, die Seitenwechsel in ihrem Inneren ganz unterdrückt. Alle diese Befehle sind jedoch nicht ganz zwingend, d. h. es gibt Fälle, in denen der Formatierer nicht von seinem selbstgewählten Seitenumbruch abzubringen ist.

Gleitobjekte werden durch die `figure`-Umgebung definiert. Die Definition eines Gleitobjekts befindet sich in der Dokumentbeschreibung an einer gewissen Stelle im Text. Diese Stelle dient als Anker für das Objekt. Bei der Formatierung wird es nämlich normalerweise am Anfang einer Seite plaziert, und zwar frühestens auf der Seite, wo sein Anker liegt. Durch einen optionalen Parameter können auch abweichende Positionierungswünsche angegeben werden: in den Text eingebettet, unten auf einer Seite oder auf einer eigenen Seite, die keinen Text enthält.

Die Ausgabe der Gleitobjekte erfolgt genau in der Reihenfolge ihrer Definitionen. Es gibt also kein „Überholen“, etwa eines großen Gleitobjekts, das nicht mehr auf die aktuelle Seite paßt, durch ein kleines, welches noch paßt. Dadurch kann sich ein Stau von Gleitobjekten entwickeln, die auf ihre Ausgabe warten. Mit dem Befehl `\clearpage` kann ihre Ausgabe erzwungen werden, bevor weiterer Text ausgegeben wird.

4.4.2 Seitenumbruch in TEX

Im folgenden geben wir einen groben Überblick über den Seitenumbruchalgorithmus von TEX. Ähnlich wie beim Zeilenumbruch gibt es auch hier eine Liste von Elementen, die in Unterlisten eingeteilt werden muß. Elemente sind analog zum Zeilenumbruch Inhaltskästen, (vertikaler) Kleber und Strafelemente. Dazu kommt noch eine neue Art von Elementen, die den Gleitobjekten entsprechen.

Im allgemeinen gibt es bei dem vertikalen Kleber nicht viel Dehn- bzw. Stauchbarkeit. Variabilität des vertikalen Klebers gibt es um abgesetzte Formeln, Abbildungen und Tabellen herum, außerdem etwas zwischen zwei Paragraphen. Die Abstände zwischen den Zeilen im Innern eines Paragraphen sind im allgemeinen fest. Daher gibt es nicht viele verschiedene Möglichkeiten, einen guten Seitenumbruch zu erzielen.

Wie in Abschnitt 4.3.2 beschrieben, wird der Zeilenumbruch paragraphenweise berechnet. Erst wenn ein Paragraph ganz gelesen ist, wird die global beste Zeileneinteilung für ihn bestimmt. Um Speicherplatz zu sparen, wird dagegen die Seiteneinteilung lokal durchgeführt, d. h. ohne eine Vorausschau auf ein ganzes Kapitel oder gar das ganze Dokument. Immer, wenn eine Seite fertiggestellt ist, wird sie in die Sprache *dvi* (siehe Abschnitt 9.3) übersetzt, ausgegeben und danach „vergessen“.

Wie beim Zeilenumbruch kann der Seitenumbruch nur an gewissen Stellen erfolgen, die wir auch Kerben nennen wollen. Die Qualität einer Kerbe als Umbruchstelle wird wieder mit Hilfe einer badness-Funktion bestimmt. Die Strafpunkte für

einen Seitenumbruch an einer bestimmten Kerbe berechnet TEX nach der folgenden Formel:

$$c = \begin{cases} p & \text{falls } b < \infty \text{ und } p \leq -10000 \text{ und } q < 10000, \\ b + p + q & \text{falls } b < 10000 \text{ und } -10000 < p < 10000 \text{ und } q < 10000, \\ 100000 & \text{falls } b = 10000 \text{ und } -10000 < p < 10000 \text{ und } q < 10000, \\ \infty & \text{falls } (b = \infty \text{ oder } q \geq 10000) \text{ und } p < 10000. \end{cases}$$

Hierbei ist

b die Strafe für die Seite als Gesamtheit, die von der Dehnung oder Stauchung der Seite abhängt;

p die spezielle Strafe für diese Umbruchstelle und

q die Strafe, die von der Plazierung von Gleitobjekten herrührt.

Die Strafe *p* setzt sich unter anderem aus den folgenden Parametern zusammen:

`interlinepenalty`	zwischen je zwei Zeilen eines Paragraphen,
`clubpenalty`	Zusatzstrafe nach der ersten Zeile eines Paragraphen zur Verhinderung von Waisen,
`widowpenalty`	Zusatzstrafe vor der letzten Zeile eines Paragraphen zur Verhinderung von Witwen,
`brokenpenalty`	bei einer Worttrennung.

Dazu kommen noch negative oder positive Beiträge, die von expliziten Kommandos zum Fördern oder Verhindern eines Seitenwechsels an dieser Stelle herrühren.

Für jede Kerbe berechnet TEX die Strafe *c*. Ist die Strafe geringer als die aller bisher für die aktuelle Seite betrachteten Kerben, so merkt sich TEX diese Kerbe als die bisher günstigste. Wird die Strafe ∞ oder ist $p \leq -10000$ (erzwungener Seitenwechsel), so beendet TEX die aktuelle Seite an der bisher günstigsten Kerbe. Die Seite wird an eine Ausgaberoutine geschickt, die Kopf- und Fußzeilen sowie die Seitennummer hinzufügt und die Seite ausgibt. Der Inhalt der Seite wird dann aus dem Hauptspeicher gelöscht.

4.5 Worttrennung

Um einen zufriedenstellenden Zeilenumbruch zu ermöglichen, müssen in vielen Fällen Wörter am Zeilenende getrennt werden. In diesem Abschnitt werden Verfahren zur automatischen Ermittlung von Trennstellen vorgestellt.

4.5.1 Das Problem der Worttrennung

Sowohl in der deutschen als auch in der englischen Sprache gibt es zwei Grundregeln zur Worttrennung:

1. Zusammengesetzte Wörter werden nach ihren Bestandteilen getrennt.
2. Wörter werden nach Sprechsilben getrennt.

Unglücklicherweise sind beide Regeln zu vage formuliert, um automatisiert zu werden. Zudem widersprechen sie sich in gewissen Fällen.

Im Englischen ist insbesondere die Einteilung in Sprechsilben von der Betonung abhängig, die wiederum bei längeren Wörtern von der Endung abhängt. Dadurch kann es vorkommen, daß zwei nahezu gleichgeschriebene Worte völlig verschieden getrennt werden. In den folgenden Beispielen ist die betonte Silbe durch einen Akzent gekennzeichnet.

dém-on-strate	de-món-stra-tive	dem-on-strá-tion
dép-i-late	de-píl-a-to-ry	dep-i-lá-tion
réc-ord (Subst.)	re-córd (Verb)	

Das letzte Beispiel zeigt, daß manchmal sogar völlig gleichgeschriebene Wörter verschieden getrennt werden, weil sie verschieden ausgesprochen werden.

Obwohl im Deutschen die Sprechsilben nicht von der Betonung abhängen, gibt es ähnliche Fälle:

Bein-hal-tung	be-in-hal-ten
Er hat ein Ver-eins-amt.	Ver-ein-samt ist er trotz-dem.

Das Trennproblem ist im Deutschen verschärft wegen der Möglichkeit, Wörter nahezu beliebig zusammensetzen zu können. Die entstehenden langen Wörter müssen viel häufiger getrennt werden als etwa die im allgemeinen recht kurzen Wörter der englischen Sprache. Zusammengesetzte Wörter sollten eher zwischen vollständigen Teilwörtern getrennt werden als im Innern von Teilwörtern. Daher sollten die legalen Trennstellen in deutschen Wörtern verschiedene Priorität haben, z. B. im Wort „Ausnahmewörterbuch“ etwa so:

$$\text{Aus}-_3\text{nah}-_5\text{me}-_1\text{wör}-_4\text{ter}-_2\text{buch},$$

was heißen soll, daß die Trennung „Ausnahme-wörterbuch“ am günstigsten ist und „Ausnah-mewörterbuch“ am schlechtesten.

Ein weiteres Problem stellen Trennungen dar, die zwar grammatisch korrekt, aber psychologisch ungeschickt sind, z. B. Anal-phabet und Urin-stinkt. Die entsprechenden Trennungen der Grundworte Al-phabet und In-stinkt sind dagegen harmlos.

Ein speziell im Deutschen auftretendes Problem ist, daß sich die Schreibung von Wörtern durch das Trennen ändern kann:

Locke Lok-ke Schiffahrt Schiff-fahrt Auffahrt Auf-fahrt

Die beiden letzten Beispielwörter zeigen, daß sich diese Veränderungen nur schwer automatisieren lassen.

Wahrscheinlich werden die oben beschriebenen Veränderungen durch Trennung in der bevorstehenden Rechtschreibreform abgeschafft. Es ist geplant, „Locke“ als „Lo-cke“ zu trennen, und es soll auch ohne Trennung „Schifffahrt“ mit drei ’f’ geschrieben werden.

4.5.2 Trennung in Dokumentsystemen

Allgemeines

Die meisten Dokumentsysteme ermitteln mögliche Trennstellen automatisch mit Hilfe von Algorithmen, die wir später vorstellen werden. Da die exakten Trennregeln sprachabhängig sind, müssen die Autoren zunächst angeben, in welcher Sprache ihr Werk verfaßt ist.

Die automatischen Trennverfahren arbeiten in der Regel nicht fehlerfrei. Ein möglicher Fehler ist, korrekte Trennstellen nicht als solche zu erkennen. Das fällt meist gar nicht auf, außer wenn dadurch ein guter Zeilenumbruch verhindert wird. Der umgekehrte Fehler besteht darin, eine Stelle, an der nicht getrennt werden darf, irrtümlich als mögliche Trennstelle anzusehen. Dieser Fehler ist schwerwiegender, da er zu falschen Trennungen führen kann, wenn die für möglich gehaltene Trennstelle tatsächlich zur Trennung benutzt wird.

Um falsche Trennungen korrigieren zu können, bieten viele Dokumentsysteme zwei Verfahren an. Das erste ist eine globale Korrektur: Die Autoren können eine Liste von Wörtern mit expliziten Trennstellen anlegen. Diese Liste sollte jedoch auf Wörter beschränkt werden, die tatsächlich falsch getrennt werden und im Text häufig vorkommen. Die andere Möglichkeit ist eine lokale Korrektur. Wenn beim Probelesen eine falsche Trennung gefunden wurde, kann sie durch spezielle Kommandos richtiggestellt werden. Dies wirkt nur an dieser einen Stelle; andere Vorkommen des falsch getrennten Wortes werden nicht beeinflußt.

Trennung im *FrameMaker*

Im *FrameMaker* können verschiedene Sprachen gewählt werden. Eine davon ist „None“; dann erfolgt gar keine automatische Trennung.

Zur Überprüfung und globalen Korrektur von Trennungen gibt es ein spezielles Dialogfenster. Dort kann ein Wort eingegeben werden. Auf den Befehl „Show Hyphenation“ hin werden alle vom *FrameMaker* bestimmten Trennstellen im Wort

korrekte Trennstellen	LaTeX englisch	LaTeX deutsch
dém-on-strate	demon-strate	de-mon-stra-te
de-món-stra-tive	demon-stra-tive	de-mon-stra-ti-ve
dem-on-strá-tion	demon-stra-tion	de-mon-stra-ti-on
dép-i-late	de-pilate	de-pi-la-te
de-píl-a-to-ry	de-pila-tory	de-pi-la-to-ry
dep-i-lá-tion	de-pila-tion	de-pi-la-ti-on
réc-ord	record	re-cord
re-córd	record	re-cord

Tabelle 4.1: Beispiele für die Trennung englischer Wörter

angezeigt. Fehlerhafte Trennungen können in dem Fenster korrigiert werden. Der Befehl „Learn“ fügt das Wort mit den korrigierten Trennstellen in ein Wörterbuch ein.

Lokal kann die Trennung durch bestimmte Tastenkombinationen beeinflußt werden. Durch gleichzeitiges Drücken der Tasten „Shift“, „Meta“ und „–“ am Anfang eines Wortes werden alle Trennstellen im Innern des Wortes deaktiviert. Um eine mögliche Trennstelle im Innern eines Wortes anzuzeigen, kann man den Einfügepunkt dorthin setzen und dann gleichzeitig auf „Control“ und „–“ drücken.

Trennung in LaTeX

Standardmäßig wird von LaTeX englischer Text angenommen. Es gibt auch eine deutsche Version von LaTeX bzw. ein Kommando, das von Englisch nach Deutsch umschaltet.

Für englische Texte arbeitet der von LaTeX benutzte Trennalgorithmus so, daß er eher korrekte Trennstellen übersieht als falsche findet. In Tabelle 4.1 werden einige Beispiele angegeben. Es wird auch gezeigt, wie LaTeX trennt, wenn irrtümlich die deutsche statt der englischen Version benutzt wird.

In Problemfällen können Trennstellen im laufenden Text durch das Kommando `\-` explizit angegeben werden. Es kann auch ein (kleines) Ausnahmewörterbuch angegeben werden, das die Trennstellen von häufiger vorkommenden Problemwörtern explizit angibt.

Bei deutschen Texten ist die Qualität der Trennung deutlich schlechter. Es werden viele mögliche Trennstellen nicht gefunden und umgekehrt noch merklich oft falsche Trennstellen angegeben. Es werden keine Trennstellen verschiedener Priorität unterschieden; das von LaTeX benutzte Trennverfahren bewirkt im Gegenteil, daß Trennstellen in Teilwörtern eher gefunden werden als die zwischen

korrekte Trennstellen	LaTeX deutsch	LaTeX englisch
ver-ein-samt	ver-ein-samt	vere-in-samt
Ver-eins-amt	Ver-ein-samt	Vere-in-samt
Bein-hal-tung	Bein-hal-tung	Bein-hal-tung
be-in-hal-ten	bein-hal-ten	bein-hal-ten
ob-jekt-ori-en-tiert	ob-jek-t-ori-en-tiert	ob-jek-to-ri-en-tiert
Sei-ten-um-bruch	Sei-te-num-bruch	Seit-enum-bruch
Aus-nah-me-wör-ter-buch	Aus-nah-mewörter-buch	Aus-nah-mewörterbuch

Tabelle 4.2: Beispiele für die Trennung deutscher Wörter

Teilwörtern; bei letzteren gibt es auch die meisten Fehler, wie die Beispiele in Tabelle 4.2 zeigen.

Das Problem der Veränderung der Schreibweise ist (auf Kosten der Autoren) gelöst; die Wörter „Locke“, „Schiffahrt“ und „Auffahrt“ werden korrekt behandelt, wenn sie als `Lo"cke`, `Schi"ffahrt` und `Auf"|fahrt` eingegeben werden; allerdings werden die meisten Autoren das wohl nicht tun.

4.5.3 Trennalgorithmen

Die Trennregeln der einzelnen Sprachen erweisen sich als zu ungenau, um sie zu automatisieren. Außerdem wären solche regelbasierten Trennalgorithmen extrem sprachabhängig.

Eine Alternative besteht darin, ein Wörterbuch mit vorgetrennten Wörtern für die betreffende Sprache im Rechner abzuspeichern und bei jedem zu trennenden Wort die Trennstellen aus diesem Wörterbuch abzulesen. Wörter wie „record“ und „Vereinsamt“ sollten in diesem Wörterbuch nur die Trennstellen enthalten, die allen ihren Lesarten gemeinsam sind, also „record“ ohne Trennstelle und „Ver-einsamt“.

Ein wörterbuchbasiertes Trennverfahren hat den Vorteil, daß nur korrekte Trennstellen gefunden werden, vorausgesetzt, das Wörterbuch ist korrekt. Einige Trennstellen werden nicht gefunden, nämlich bei Wörtern wie „record“ und bei Wörtern, die gar nicht im Wörterbuch vorkommen.

Ein Problem dieses Ansatzes ist der Speicherplatzaufwand. Die englische Sprache z. B. hat einen großen Wortschatz. Der Vorrat an deutschen Grundwörtern ist zwar kleiner, aber aufgrund der Möglichkeit zur Wortzusammensetzung und der Vielzahl von Flexionsformen vergrößert sich die Anzahl möglicher Wörter ins Unermeßliche. Beachten Sie, wie die Trennung durch die verschiedenen Endungen beeinflußt wird: er lobt, er lob-te, wir lo-ben.

Bei einer großen Gruppe von Trennverfahren werden die möglichen Trennstellen anhand von typischen Buchstabenkombinationen gesucht. Diese Verfahren arbeiten zweistufig. Wenn das Textsystem, das so ein Verfahren benutzt, an eine neue Sprache angepaßt wird, dann wird in einem Vorverarbeitungsschritt ein mit Trennstellen und Worthäufigkeitsinformationen versehenes Wörterbuch der Sprache analysiert. Dabei werden Informationen darüber gesammelt, in welchen Kontexten Trennstellen wahrscheinlich sind. Diese Information nimmt kompakt dargestellt viel weniger Platz ein als das ursprüngliche Wörterbuch. Bei der eigentlichen Textverarbeitung wird nur diese Information benutzt, um Trennstellen zu ermitteln. Dabei werden, wie an den Beispielen im vorigen Abschnitt zu sehen war, sowohl mögliche Trennstellen übersehen als auch falsche Trennstellen gefunden.

Beim *Times-Magazine-Verfahren* besteht die Trenninformation aus drei Tabellen, die über Buchstabenpaare (x,y) indiziert sind (wobei Wortanfang und Wortende als spezielle Buchstaben behandelt werden). Die drei Tabellen enthalten die Wahrscheinlichkeiten $p_n(x,y)$, daß *nach* einem Teilwort *xy* eine Trennstelle ist, $p_z(x,y)$, daß *zwischen x* und *y* eine Trennstelle liegt, und $p_v(x,y)$, daß *vor* einem Teilwort *xy* getrennt werden darf. Für eine konkrete Trennstelle wie z. B. „Trenn-stelle“ ist die Wahrscheinlichkeit, daß sie legal ist, gleich dem Produkt der drei Wahrscheinlichkeiten $p_n(n,n) \cdot p_z(n,s) \cdot p_v(s,t)$. Wenn diese Gesamtwahrscheinlichkeit oberhalb eines bestimmten Schwellenwertes liegt, dann wird diese Trennstelle als legal akzeptiert.

Beim Verfahren nach *Liang* [Lia83], das auch in TEX benutzt wird, werden anstelle der Buchstabenpaare beliebig lange Teilworte, sogenannte *Muster*, betrachtet. Diese Teilworte sind mit ganzen Zahlen zwischen 0 und 9 versehen, die zwischen den Buchstaben, am Anfang und am Ende stehen. Ein Teilwort aus *n* Buchstaben hat also $n+1$ Zahlen. Ungerade Zahlen bedeuten, daß an dieser Stelle vermutlich getrennt werden darf. Gerade Zahlen bedeuten, daß vermutlich *nicht* getrennt werden darf. In beiden Fällen steigt die Sicherheit dieser Information mit der Größe der Zahl.

Die aus dem Wörterbuch extrahierte Information liegt als eine Liste von einigen Tausend Mustern vor (für englisch 4447 und für deutsch 5719). Zu einem konkreten Wort im Text, das auf Trennstellen untersucht werden soll, werden zunächst alle Muster gesucht, die zu Teilwörtern des Wortes passen. Dann wird zu jeder Stelle im Wort das Maximum aller Zahlen gebildet, die von allen Mustern, die diese Stelle überdecken, geliefert werden. Ist dieses Maximum ungerade, dann ist die betrachtete Stelle eine vermutlich legale Trennstelle. Sie wird von TEX allerdings nur dann zu einer möglichen Trennstelle gemacht, wenn davor mindestens zwei und dahinter mindestens drei Buchstaben in der englischen Version bzw. zwei Buchstaben in der deutschen Version liegen.

In den folgenden Beispielen wird die Zahl 0 durch eine Leerstelle ersetzt. Wortanfang und Wortende werden durch einen Punkt bezeichnet. Sie dürfen auch in Mustern vorkommen. Als englisches Beispiel betrachten wir in Abbildung 4.1 die beiden Wörter „de-píl-a-to-ry“ und „dep-i-lá-tion“, die von TEX (teilweise falsch)

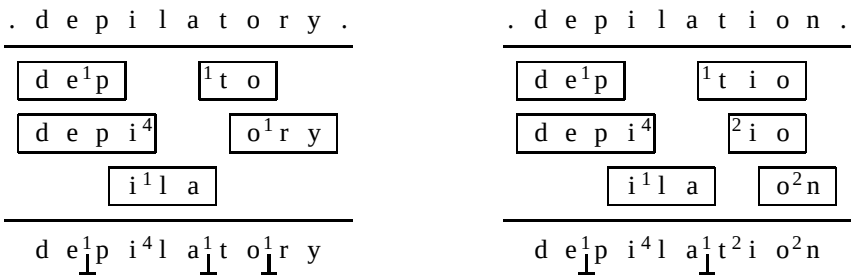

Abbildung 4.1: Englische Trennbeispiele mit Liang-Trennmustern

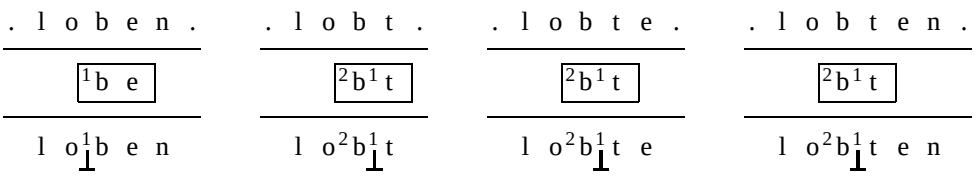

Abbildung 4.2: Deutsche Trennbeispiele mit Liang-Trennmustern I

wie „de-pila-tory“ und „de-pila-tion“ getrennt werden. Die gefundenen Trennstellen sind jeweils mit ⊥ gekennzeichnet. Die dritte Trennstelle in „de-pila-to-ry“ wird nicht berücksichtigt, weil dahinter nur noch zwei Buchstaben kommen.

Betrachten wir nun einige deutsche Wörter, natürlich unter Benutzung der deutschen Trennmuster (in Version 3.1a). Die Trennmuster bestehen nur aus Kleinbuchstaben, „treffen“ aber auch großgeschriebene Wortanfänge.

In Abbildung 4.2 wird die Trennung einiger von „Lob“ abgeleiteter Wörter gezeigt. Auf „Lob“ selbst treffen keine Trennmuster zu. Nach der in der deutschen Version gültigen Regel, daß hinter der letzten Trennstelle noch mindestens zwei Buchstaben folgen sollen, wird die Trennstelle in „lobt“ gestrichen. Es ergeben sich die Trennungen „Lob“, „lo-ben“, „lobt“, „lob-te“ und „lob-ten“.

Als nächstes betrachten wir die Wörter „beinhalten“ und „Seitenumbruch“ in Abbildung 4.3. Bei „beinhalten“ wird die erste Trennstelle übersehen und bei „Seitenumbruch“ ergibt sich eine völlig falsche Trennung an der Wortzusammensetzungsfuge. Auf das Wort „Beinhaltung“ treffen die Muster ^{1}b e, n^{1}h und ^{2}l^{1}t zu. Damit ergibt sich die korrekte Trennung „Bein-hal-tung“.

In Abbildung 4.4 werden die Trennmuster für das Wort „objektorientiert“ gezeigt. Die letzte Trennstelle fällt weg, weil nur ein Buchstabe folgt. Somit ergibt sich die teilweise falsche Trennung „ob-jek-t-ori-en-tiert“. Ähnlich wie bei „lob-ten“ gegenüber „lobt“ wird die letzte Trennstelle aktiv in der Flexionsform „ob-jek-t-ori-en-tier-ten“.

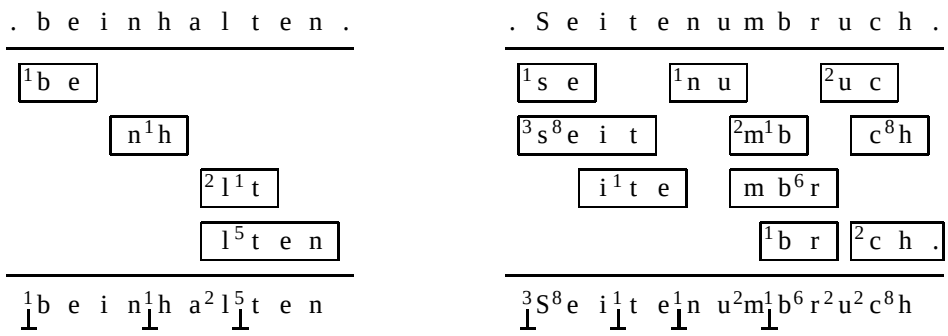

Abbildung 4.3: Deutsche Trennbeispiele mit Liang-Trennmustern II

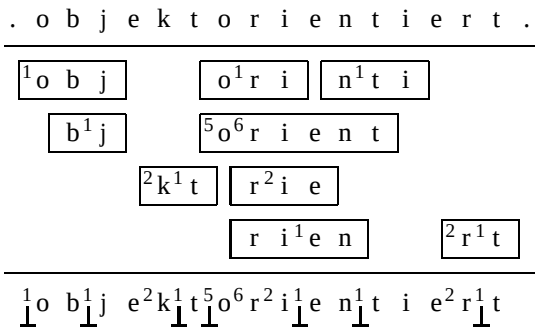

Abbildung 4.4: Deutsche Trennbeispiele mit Liang-Trennmustern III

Bei den deutschen Trennmustern scheint es zwei Arten zu geben. Relativ kurze Muster mit kleinen Zahlen wie b^1j, $^2k^1t$, $^1n\,u$ und $^2r^1t$ spiegeln wohl die deutschen Regeln zur Silbeneinteilung wider. Auf ihr Konto gehen die falschen Trennstellen „Seite-numbruch“ und „objek-torientiert“, die bei nichtzusammengesetzten Wörtern eine korrekte Silbeneinteilung darstellen würden.

Andererseits gibt es auch ziemlich lange Muster mit hohen Zahlen wie $^3s^8e\,i\,t$ und $^5o^6r\,i\,e\,n\,t$, die Ausnahmefälle regulieren. Das Muster $^5o^6r\,i\,e\,n\,t$ dient wohl dazu, ein mögliches Teilwort „orient“ vor der silbenmäßigen Trennung „o-rient“ zu schützen und dafür die Trennstelle vor das Teilwort zu legen, also wahrscheinlich an die Fuge zwischen zwei Teilwörtern.

Die obige Diskussion suggeriert, daß die Trennmuster von Hand unter Berücksichtigung irgendwelcher Regeln und Ausnahmen aufgestellt würden. In Wirklichkeit werden sie aber mit Hilfe eines hier nicht beschriebenen Verfahrens automatisch aus einem Wörterbuch mit Trennstellen hergeleitet.

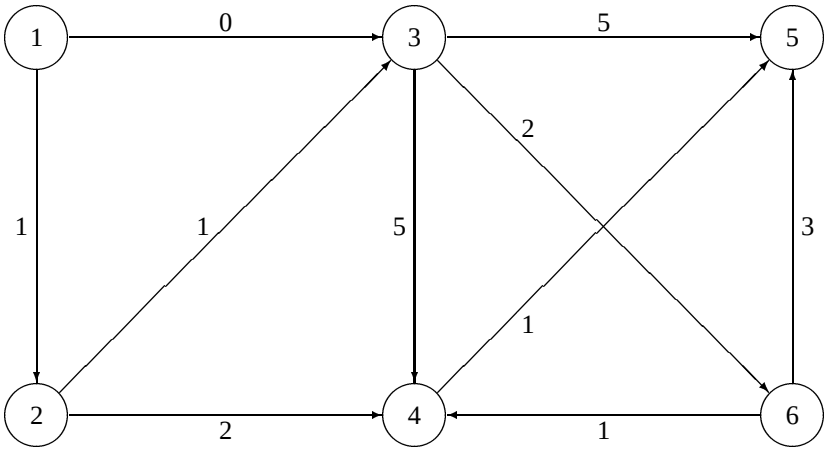

Abbildung 4.5: Ein Graph mit 6 Knoten und 10 Kanten

4.6 Graphenalgorithmen

Der in Abschnitt 4.3.2 vorgestellte Algorithmus zum Zeilenumbruch in TEX kann als Graphalgorithmus aufgefaßt werden, wie wir im folgenden zeigen werden. Dazu muß zunächst einmal definiert werden, was ein Graph ist.

4.6.1 Graphen

Ein *Graph* ist eine in der Informatik häufig benutzte Datenstruktur, mit der sich Objekte und ihre Beziehungen zueinander darstellen lassen. Mathematisch gesehen besteht ein Graph aus einer Menge V von Knoten (englisch *vertices*) und einer Menge E von Kanten (englisch *edges*). Bildlich werden die Knoten eines Graphen meist als Punkte, Kreise oder Kästen dargestellt und die Kanten als Verbindungen zwischen je zwei Knoten (siehe Abbildung 4.5).

In einem *gerichteten Graphen* ist jede Kante ein Paar (v,w) von zwei Knoten. So eine Kante kann bildlich als ein Pfeil von v nach w dargestellt werden. Da die Paare (v,w) und (w,v) verschieden sind, kommt es auf die Richtung der Kante bzw. des Pfeiles an. Der Knoten v heißt *Anfangsknoten* der Kante (v,w) und der Knoten w *Endknoten*. Der Graph aus Abbildung 4.5 ist gerichtet.

In einem *ungerichteten Graphen* ist jede Kante eine Menge $\{v,w\}$ von zwei Knoten. Da $\{v,w\} = \{w,v\}$ ist, kommt es auf die Reihenfolge der Knoten nicht an. Eine solche Kante kann bildlich als eine Linie dargestellt werden, die v und w miteinander verbindet. Im folgenden werden wir allerdings nur gerichtete Graphen betrachten.

Ein *Weg* in einem gerichteten Graphen ist eine Folge von Kanten (v_0,v_1), (v_1,v_2), $\ldots$, (v_{n-1},v_n), bei der der Endknoten jeder Kante (außer der letzten) mit dem Anfangsknoten der nächsten Kante übereinstimmt. Der Kürze halber kann ein Weg

als die Folge der Knoten $(v_0, v_1, \ldots, v_n)$ beschrieben werden, durch die er führt. Ein Weg wie oben angegeben führt von v_0 nach v_n. Der Vollständigkeit halber nimmt man an, daß es zu jedem Knoten v einen Weg von v nach v gibt, der durch eine leere Folge von Kanten gegeben ist. Er kann als (v) geschrieben werden. In dem Graph aus Beispiel 4.5 gibt es unter anderem die Wege $(1,2,3,4,5)$ und $(3,6,4,5)$.

4.6.2 Längeninformation und kürzeste Wege

In einem *kantenmarkierten Graphen* wird jeder Kante (v,w) eine reelle Zahl $l(v,w) \geq 0$ zugeordnet, die als die Länge der Verbindung von v nach w gedeutet wird. Wir werden später sehen, daß diese „Länge“ außer einer konkreten geometrischen Bedeutung auch eine übertragene Bedeutung haben kann. Der Beispielgraph aus Abbildung 4.5 ist kantenmarkiert; die Kantenmarkierungen sind durch die neben die Kanten geschriebenen Zahlen gegeben.

Der Längenbegriff kann von den Kanten auf die Wege eines Graphen erweitert werden. Dabei ist die Länge eines Weges die Summe der Längen der Kanten, aus denen der Weg besteht:

$$l(v_0, v_1, \ldots, v_n) \;=\; l(v_0, v_1) + \cdots + l(v_{n-1}, v_n)$$

Im Spezialfall des leeren Weges ist $l(v) = 0$.

Im Zusammenhang mit der Weglänge kann man nun verschiedene Fragen stellen, z. B. für zwei feste Knoten v und w, was ist der kürzeste Weg von v nach w, oder für einen festen *Startknoten* b, was sind die kürzesten Wege von b zu allen anderen Knoten w? Wir werden hier die zweite Frage betrachten, die ja die erste Frage gleich mitbeantwortet.

In dem Graph aus Abbildung 4.5 gibt es mehrere Wege von Knoten 1 nach Knoten 5, unter anderem $(1,3,5)$ mit Länge 5, $(1,2,3,4,5)$ mit Länge 8, $(1,2,4,5)$ mit Länge 4 und $(1,3,6,4,5)$ ebenfalls mit Länge 4. Die beiden letztgenannten Wege sind die kürzesten Wege von 1 nach 5. Von Knoten 4 nach Knoten 6 gibt es dagegen überhaupt keinen Weg.

Ein Algorithmus zum Finden kürzester Wege

Der folgende Algorithmus zum Finden kürzester Wege in Graphen stammt von E. W. Dijkstra. Er beruht auf der Erkenntnis, daß Teilwege von kürzesten Wegen immer auch kürzeste Wege sind. Wenn nämlich ein kürzester Weg von b nach d durch einen Knoten c geht, dann sind seine Teilwege von b nach c und von c nach d jeweils kürzeste Wege von b nach c bzw. von c nach d. Wenn das nicht der Fall wäre, könnte man offensichtlich durch Ersetzen des nichtkürzesten Teilwegs durch einen kürzeren einen insgesamt kürzeren Weg von b nach d finden. Das wäre aber ein Widerspruch.

Der Algorithmus benützt diese Erkenntnis, indem er schrittweise kürzeste Wege verlängert, genauer gesagt, bekannte kürzeste Wege zu Knoten v zu kürzesten Wegen zu Nachbarn w von v erweitert. Dabei merkt er sich in $L(w)$ die Länge dieses kürzesten Weges und in $N(w)$ den Vorgänger auf diesem Weg, in unserem Falle v.

Zu jedem Zeitpunkt gibt es eine Menge $F \subseteq V$ von Knoten v, für die die Berechnung des kürzesten Weges vom Startknoten b nach v schon beendet ist. Eine Kante, die in F beginnt und außerhalb endet, heißt *Schnittkante*. Der Algorithmus verwaltet die Menge C aller Schnittkanten.

Da es Knoten geben kann, die von b aus nicht errreichbar sind, initialisieren wir $L(v)$ für alle Knoten außer b mit ∞. (In der Praxis genügt es, eine sehr große Zahl zu benutzen, die größer ist als die Summe aller Kantenlängen.) $L(b)$ setzen wir auf 0, weil der leere Weg (b) von b nach b die Länge 0 hat. Die Menge F besteht aus allen Knoten v mit $L(v) < \infty$. Sie braucht daher nicht explizit verwaltet zu werden.

Der Algorithmus sieht folgendermaßen aus:

Setze $L(b)$ auf 0;
Setze $L(v)$ auf ∞ für alle $v \neq b$;
Initialisiere C als die Menge aller Kanten (b,v) mit $v \neq b$;
Solange C nicht leer ist,
$(*)$ sei $e = (v,w)$ eine der Kanten in C, für die $L(v) + l(e)$ minimal ist;
Setze $L(w)$ auf $L(v) + l(e)$ und $N(w)$ auf v;
Lösche alle Kanten mit Endknoten w aus C;
Füge zu C alle Kanten (w,u) mit $L(u) = \infty$ hinzu.

Auf den ersten Blick könnte man vermuten, daß Zeile $(*)$ lauten müßte: „Sei $e = (v,w)$ diejenige Kante aus C mit dem kleinsten Wert $L(v) + l(e)$“. Dieser kleinste Wert könnte aber von mehreren Kanten erzielt werden. Von diesen darf dann beliebig eine Kante ausgewählt werden.

Wenn der Algorithmus terminiert, dann gibt L zu jedem Knoten v die Länge $L(v)$ des kürzesten Weges von b nach v an. Den Weg selbst kann man wie folgt ermitteln: Sei $v_0 = v$. Solange $v_i \neq b$, berechne $v_{i+1} = N(v_i)$. Die so erhaltenen Knoten geben den kürzesten Weg in *umgekehrter Reihenfolge* an, d. h. der kürzeste Weg von b nach v ist $(b = v_n, \ldots, v_1, v_0 = v)$.

Dieses Verfahren erinnert an den von uns vorgestellten Zeilenumbruchalgorithmus. Wir werden bald sehen, daß diese Ähnlichkeit nicht zufällig ist.

Wieso ist der vorgestellte Algorithmus korrekt, d. h. wieso ist es sicher, daß in der Zeile nach $(*)$ der kürzeste Weg von b nach w tatsächlich über v führt und e als letzte Kante hat? Betrachten Sie einen beliebigen Weg W von b nach w. Da b in F liegt und w nicht, muß W eine Kante $e' = (v', w')$ aus C beinhalten. Wegen der Wahl von e ist $L(v) + l(e) \leq L(v') + l(e')$. Da W noch den Teilweg von w' nach w enthält, gilt $L(v') + l(e') \leq l(W)$. Damit ist W mindestens genauso lang wie der Weg von b nach w mit e als letzter Kante. Beachten Sie, daß dieses Argument davon abhängt, daß

negative Längen verboten sind. Andernfalls könnte sich der Teilweg von w' nach w durch eine negative Länge $l(W)$ so weit verkürzen, daß W kürzer als der Weg über e würde.

Ein Beispiellauf des Algorithmus

Betrachten wir den Ablauf des Algorithmus an unserem Beispielgraphen aus Abbildung 4.5 mit Startknoten $b = 1$. Nach der Initialisierung ergeben sich die folgenden Werte für L und N:

Knoten	1	2	3	4	5	6
L	**0**	∞	∞	∞	∞	∞
N	–	–	–	–	–	–

Dabei bedeutet ein Strich bei N, daß der Wert undefiniert (nicht initialisiert) ist. Die Menge C besteht aus den Kanten $(1,2)$ mit dem Wert $L(v) + l(e) = 0 + 1 = 1$ und $(1,3)$ mit $0 + 0 = 0$. Daher wird beim ersten Erreichen der Zeile $(*)$ die Kante $(1,3)$ ausgewählt. Danach ergeben sich als neue Werte von L und N die folgenden:

Knoten	1	2	3	4	5	6
L	0	∞	**0**	∞	∞	∞
N	–	–	**1**	–	–	–

Nach der Neuberechnung enthält C die Kanten $(1,2)$ mit dem Wert $0+1=1$, $(3,4)$ und $(3,5)$ mit $0+5=5$ sowie $(3,6)$ mit $0+2=2$. Daher wird beim nächsten Erreichen der Zeile $(*)$ Kante $(1,2)$ ausgewählt und es ergibt sich:

Knoten	1	2	3	4	5	6
L	0	**1**	0	∞	∞	∞
N	–	**1**	1	–	–	–

Die Menge C enthält immer noch die Kanten $(3,4)$ und $(3,5)$ mit Wert 5 sowie $(3,6)$ mit 2. Dazu kommt noch $(2,4)$ mit $1+2=3$. Somit wird in Zeile $(*)$ Kante $(3,6)$ genommen. Die neuen Werte von L und N sind:

Knoten	1	2	3	4	5	6
L	0	1	0	∞	∞	**2**
N	–	1	1	–	–	**3**

Schnittkanten sind jetzt $(3,4)$ und $(3,5)$ mit Wert 5, $(2,4)$ mit $1+2=3$, $(6,4)$ mit $2+1=3$ und $(6,5)$ mit $2+3=5$. Diese Situation ist in Abbildung 4.6 dargestellt. Die Knoten mit endlichem L-Wert sind fett umrandet und der L-Wert ist jeweils

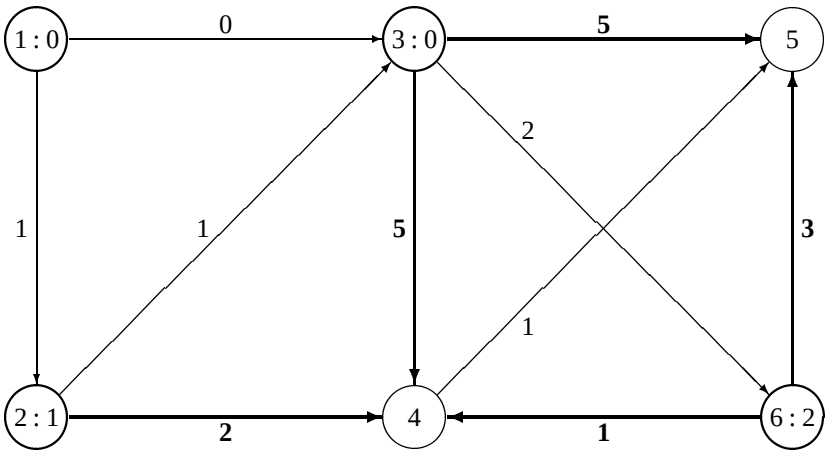

Abbildung 4.6: Beispielsituation während der Bestimmung kürzester Wege

nach dem Doppelpunkt angegeben. Die fünf Schnittkanten und ihre Längen sind ebenfalls fettgedruckt.

Wenn in dieser Situation wieder Zeile $(*)$ ausgeführt wird, besteht eine echte Wahlmöglichkeit: Sowohl Kante $(2,4)$ als auch $(6,4)$ hat einen Wert von $L(v)+l(e)=3$. Falls $(6,4)$ ausgewählt wird, ergibt sich

Knoten	1	2	3	4	5	6
L	0	1	0	**3**	∞	2
N	–	1	1	**6**	–	3

Die Menge C hat jetzt die Elemente $(3,5)$ mit $0+5=5$, $(6,5)$ mit $2+3=5$ und $(4,5)$ mit $3+1=4$. Daher wird Kante $(4,5)$ ausgewählt und man erhält

Knoten	1	2	3	4	5	6
L	0	1	0	3	**4**	2
N	–	1	1	6	**4**	3

Da C jetzt leer ist, ist das das Endergebnis. Es läßt sich ablesen, daß der kürzeste Weg von 1 nach 5 eine Länge von 4 Einheiten hat. Der Weg selbst ergibt sich aus $N(5)=4$, $N(4)=6$, $N(6)=3$, $N(3)=1$ als $(1,3,6,4,5)$. Hätte man in der Situation von Bild 4.6 Kante $(2,4)$ ausgewählt, dann hätte sich nichts geändert, außer daß $N(4)=2$ wäre. Dann hätte man den anderen kürzesten Weg $(1,2,4,5)$ gefunden.

4.6.3 Anwendungen des Problems kürzester Wege

Eine Menge interessanter Probleme kann so formuliert werden, daß die Lösung sich als kürzester Weg in einem geeigneten kantenmarkierten Graphen ergibt.

Das naheliegendste Beispiel ist das Problem, den geographisch kürzesten Weg zwischen zwei Orten zu finden. Dazu benutzen wir einen Graphen, den wir aus einer Straßenkarte gewinnen. Seine Knoten sind geographische Punkte. Dazu zählen Städte und Dörfer, aber auch außerhalb von solchen liegende Straßenknoten. Eine Kante zwischen zwei Knoten stellt die Straße dar, die die entsprechenden Punkte verbindet. Markiert ist die Kante mit der Länge dieser Straße. Der oben vorgestellte Algorithmus zum Finden kürzester Wege in Graphen löst unser Problem.

Natürlich wäre nicht zu erwarten, daß wir auf diese Art und Weise auch Wege mit kürzesten Fahrtzeiten berechnet bekämen; denn leider wird der Algorithmus Autobahnen, Bundesstraßen und Landstraßen gleich behandeln und dadurch wahrscheinlich Wege finden, die sehr viele Landstraßenstücke enthalten. Wir möchten vermutlich Autobahnstrecken vorziehen, auch wenn dadurch die Gesamtstrecke länger wird. Eine Lösung besteht darin, die Kanten in dem Straßengraphen nicht mit der geographischen Länge der betreffenden Straße, sondern mit der zum Durchfahren der Straße benötigten Zeit zu markieren. Auf dem so modifizierten Graphen würde der Algorithmus, der „kürzeste“ Wege, d. h. Wege mit der kleinsten Kantensumme, findet, jetzt Wege finden, die die schnellste Verbindung zwischen zwei Orten liefern.

Im täglichen Leben läßt sich die zum Durchfahren einer Straße benötigte Zeit schlecht abschätzen, da sie von zu vielen Faktoren, z. B. Staus abhängt. Dieses Beispiel soll nur zeigen, daß man sogar im Fall der geographischen Wegesuche andere Kantenmarkierungen als Entfernungen wählen kann. Allgemein interpretiert man deshalb die Kantenmarkierungen als *Kosten*. Dahinter können sich dann Entfernungen, Zeit, Energie oder auch Geld verbergen.

Welche Probleme lassen sich überhaupt auf das Problem kürzester Wege zurückführen? Ein dazu geeignetes Problem muß natürlich eine bestimmte Struktur aufweisen. Es muß verschiedene prinzipiell mögliche Lösungen haben, denen jeweils Kosten zugeordnet sind. Gesucht wird eine Lösung mit minimalen Kosten. Jeder Weg zur Lösung des Problems muß sich in einer Folge von Einzelentscheidungen ergeben. Bei jeder einzelnen Entscheidung gibt es eventuell mehrere Alternativen. Jede Entscheidung führt von einer Teillösung zu einer weiteren Teillösung und trägt unabhängig von anderen getroffenen oder zu treffenden Entscheidungen gewisse Kosten zu den Kosten der Gesamtlösung bei. Meist berechnen sich die Kosten der Gesamtlösung als Summe der Kosten der Einzelentscheidungen. Das Geflecht der möglichen Schritte und ihrer Kosten muß in einen gerichteten kantenmarkierten Graphen übersetzt werden. Die Knoten in dem zu konstruierenden Graphen entsprechen Teillösungen. Kanten entsprechen den Einzelentscheidungen. Sie sind mit positiven Zahlen markiert, nämlich den Kosten, die diese Entscheidungen zu den Gesamtkosten beitragen. Der Algorithmus der kürzesten Wege liefert dann eine Gesamtlösung mit minimalen Gesamtkosten.

4.6.4 Zeilenumbruch als Problem kürzester Wege

Das in Abschnitt 4.3.2 beschriebene Problem des Zeilenumbruchs kann man auf das Problem der kürzesten Wege zurückführen. Dazu betrachten wir alle Kerben im gegebenen Abschnitt als mögliche Umbruchstellen. Die Auszeichnung einer Kerbe als Umbruchstelle ist eine der oben angesprochenen Einzelentscheidungen. Eine prinzipiell mögliche Lösung des Zeilenumbruchproblems für einen ganzen Abschnitt ist eine Folge von Umbruchstellen, so daß alle entstehenden Zeilen akzeptabel sind.

Es gibt also eine Kante zwischen zwei Knoten *a* und *b*, wenn die Kerbe k_a zu *a* vor der Kerbe k_b zu *b* steht, und wenn das Strafmaß (badness) der Zeile zwischen k_a und k_b die vorgegebene Toleranzschwelle nicht überschreitet. Die Kante ist mit diesem Strafmaß markiert. Der Startknoten in diesem Graphen entspricht der Kerbe am Anfang des Abschnitts. Es gibt einen Endknoten, der der Kerbe am Ende des Abschnitts entspricht. Der kürzeste Weg vom Start- zum Endknoten in diesem Graphen entspricht dann dem Zeilenumbruch mit der geringsten Gesamtstrafe.

Der Zeilenumbruchalgorithmus ist in gewisser Hinsicht einfacher als der allgemeine Algorithmus zum Finden kürzester Wege, da nichts vorkommt, was den Mengen F und C entsprechen würde. Das liegt daran, daß wir beim Zeilenumbruch über eine Zusatzinformation verfügen, die bei allgemeinen Graphen nicht gegeben ist: die Kerben in einem Absatz (sprich die Knoten im Graphen) sind durch ihr Vorkommen im Text linear geordnet. Die Kerben können also der Reihe nach betrachtet werden. Die Menge F des allgemeinen Falls entspricht der Menge der Kerben, die vor der gerade betrachteten liegen, und die Menge C der Menge der potentiellen Zeilen, die vor der gerade betrachteten Kerbe beginnen und bei ihr oder nach ihr enden. Außerdem wird nicht eine minimale Kante in ganz C gesucht, sondern eine minimale Kante in der Teilmenge von C, die in der gerade betrachteten Kerbe endet. Das ist eine zulässige Modifikation, die die Suche erleichtert und garantiert, daß die Menge F auch weiterhin einem Anfangsstück der Kerben entspricht.