Prof. Dr. Matthias Hemmje

Kurs 01875

Multimediainformationssysteme I

LESEPROBE

Fakultät für
Mathematik und Informatik

FernUniversität in Hagen

Inhalt

Kurseinheit 1
Zeichenkodierungen und Audioformate

Dieser Text beinhaltet die erste Kurseinheit des Kurses 1875 „Multimediainformationssysteme I“. Der Kurs beschäftigt sich mit Methoden der Repräsentation, Speicherung, Verwaltung und Verarbeitung großer Mengen von Multimedia-Dokumenten, die nicht nur aus Textdokumenten bestehen, sondern auch aus Graphiken, Fotos sowie Video- und Tonsequenzen. Solche Kollektionen spielen in multimedialen Informationssystemen eine zentrale Rolle.

Die Anwendbarkeit dieser Systeme hängt sehr stark davon ab, inwieweit der Zugriff auf diese Daten sowie deren effiziente Erschließung und Indexierung unterstützt wird. Ein Thema der Vorlesung sind daher neue Ansätze aus dem Bereich des Information Retrieval, die einen inhaltsorientierten, strukturorientierten oder Kontext/Meta-Daten-orientierten Zugriff auf Multimedia-Dokumente ermöglichen.

Ein weiteres Themengebiet sind neue Konzeptionen für Multimedia-Informationen-Retrieval-Programmier- und Benutzungsschnittstellen. Diese haben insbesondere im Bereich der Multimedia-Informationssysteme eine hohe Relevanz, da konventionelle Anfragesprachen und die darauf basierenden Interfaces auf die Erfordernisse und Möglichkeiten des bislang vorherrschenden textorientierten Information Retrieval zugeschnitten sind.

Ziel der Vorlesung ist es, einen einführenden Überblick über aktuelle Technologien sowie zum gegenwärtigen Stand der Forschung und Entwicklung in den relevanten Themengebieten Multimediatechnologien sowie zu Hypermedia- und Multimedia-Information-Retrieval-Methoden und den dazu korrespondierenden Technologien zu geben.

Die vorliegende erste Kurseinheit stellt dabei zunächst Methoden und Technologien zur adäquaten Repräsentation, Speicherung und Verarbeitung von Text sowie von Audiosignalen vor. Sie überlappt somit hinsichtlich der Textkodierung mit der ersten Kurseinheit des Kurses 1873 „Daten- und Dokumentmanagement im Internet“, verzichtet aber bewusst auf das weitere Wissen aus diesem Kurs zum Thema Dokumentstrukturen und Dokumentverwaltung sowie auf das Wissen des Kurses 1874 „Informations- und Wissensmanagement im Internet“, wenngleich deren Inhalte sehr wohl ebenfalls für den vorliegenden Kurs 1875 „Multimediainformationssysteme I“ als Grundlagenwissen relevant sind und vorausgesetzt werden. Weiterhin gibt der Kurs in seinen Kurseinheiten 1 und 2 über das Themenfeld Multimediatechnologien nur einen ersten einführenden und sehr selektiven Überblick. Dieser kann vertieft werden durch die Inhalte des Kurses 1876 „Multimediainformationssysteme II“ sowie durch die Inhalte des Kurses 21792 „Multimediatechnologien II.

Der vorliegende Kurs baut darüber hinaus in seinen Kurstexten auf Inhalten einer Seminarveranstaltung auf, die an der FernUniversität in Hagen im SS 2007 durchgeführt wurde. Dank für einen wertvollen Seminarbeitrag zum Thema Audioformate, der in der vorliegenden Kurseinheit quasi als Rohtext Verwendung gefunden hat, gilt somit an dieser Stelle zunächst der Fernstudentin Birgit Ianniello sowie den wissenschaftlichen Mitarbeitern Holger Brocks und Gunter Sterr in der Unterstützung der Vorbereitung bzw. redaktionellen Aufarbeitung des Kursmaterials.

1 Zeichenkodierungen

1.1 Übersicht

Im weiteren Verlauf befasst sich die Kurseinheit nun zunächst mit den grundlegenden Bestandteilen eines Textdokumentes, nämlich seinem Inhalt, und damit, wie dieser Inhalt im Computer repräsentiert – man sagt auch „kodiert“ – werden kann. Es wird dabei davon ausgegangen, dass der Inhalt aus „glattem“ Text besteht, also aus einer Folge von elementaren Texteinheiten wie Buchstaben, Ziffern, Interpunktionszeichen und anderen Zeichen. Diese Zeichen müssen letztendlich in Bits und Bytes kodiert werden. Eine Zeichenkodierung gibt dazu eine Vorschrift an.

Die bekannteste Zeichenkodierung ist US-ASCII, standardisiert als ANSI X3.4 (1968) und als die Variante ISO 646-US von ISO 646 (1972). Der Zeichensatz von US-ASCII (Abbildung 1.1) reicht lediglich, um lateinische und US-amerikanische Texte sowie Texte in einigen wenigen weiteren Sprachen zu kodieren. Für das Deutsche, das Dänische und vierzehn weitere Sprachen sieht ISO 646 deshalb nationale Varianten vor, die die US-ASCII-Zeichen „#“ (Hash), „$“ (Dollar), „@“ (At), „[“ (eckige Klammer auf), „\“ (Backslash), „]“ (eckige Klammer zu), „^“ (Caret), „`“ (Grave), „{“ (geschweifte Klammer auf), „|“ (Strich), „}“ (geschweifte Klammer zu) und „~“ (Tilde) durch nationale Zeichen ersetzen. Die deutsche Variante ISO 646-DE (Abbildung 1.2) nimmt beispielsweise die in Tabelle 1.1 angegebenen Ersetzungen vor.

ISO 646-US	ISO 646-DE
[	Ä
\	Ö
]	Ü
{	ä
|	ö
}	ü
~	ß
@	§

Tabelle 1.1: Die Ersetzungen der deutschen Variante von ISO 646

Aus diesem Grund erscheint ein Absender „Technische Universität München, Arcisstraße 21“ gelegentlich auch heute noch in amerikanischen E-Mail-Programmen in der Form „Technische Universit{t M}nchen, Arcisstra˜e 21“; d. h. die ursprünglich mit ISO 646-DE angelegte Kodierung der Adresse wurde mithilfe von ISO 646-US ausgegeben.

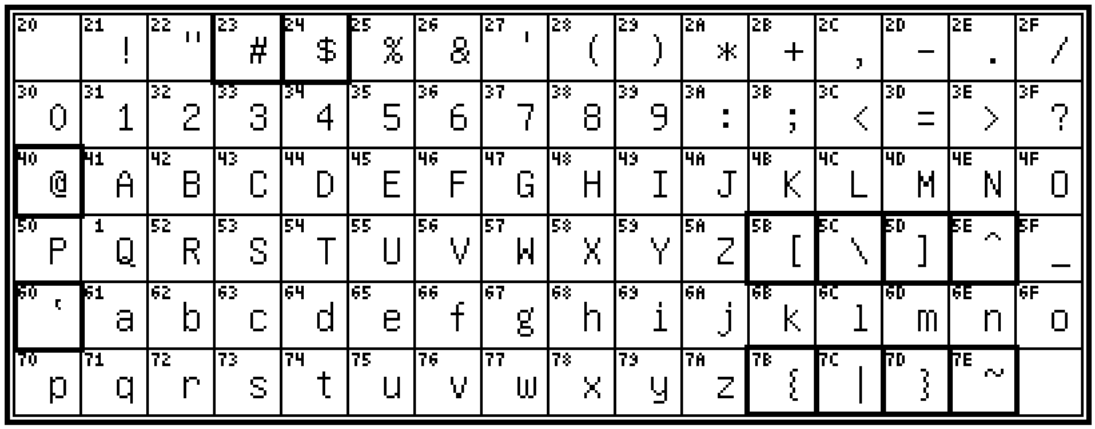


Abbildung 1.1: Der Zeichensatz ISO 646-US (US-ASCII)

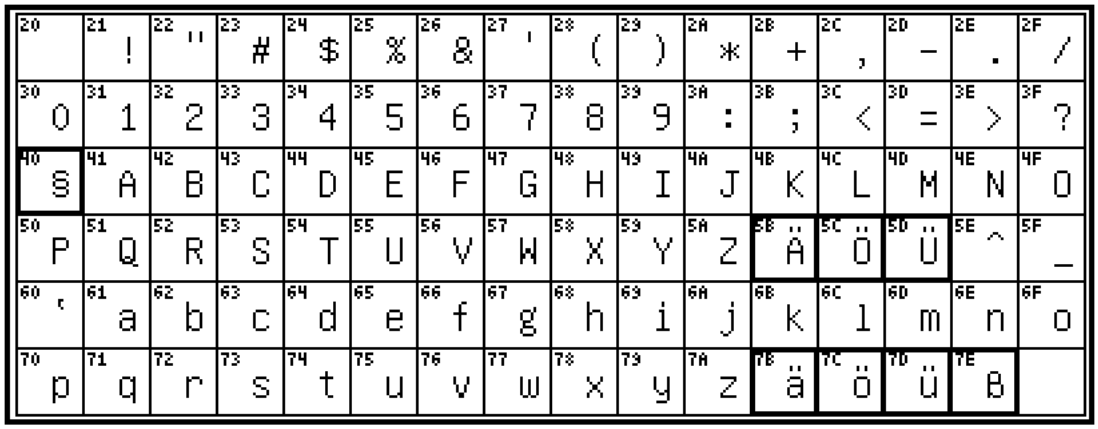


Abbildung 1.2: Die deutsche Variante ISO 646-DE von ISO 646

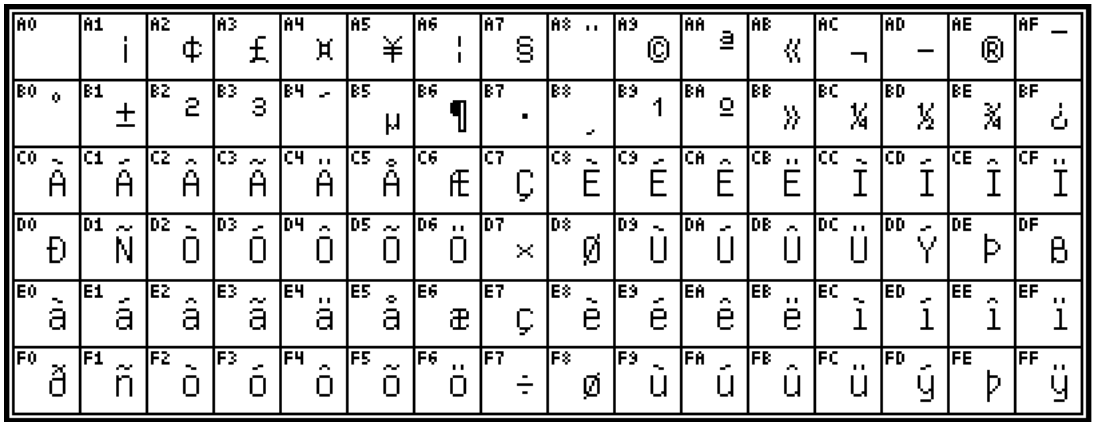


Abbildung 1.3: Der Zeichensatz ISO 8859-1 (ISO Latin-1)

Mitte der achtziger Jahre erweiterte die European Computer Manufacturer's Association (ECMA) den Zeichensatz US-ASCII zu einer Familie von Zeichensätzen, mit denen die alphabetischen Schriftsysteme kodiert werden können. Die inzwischen von der ISO unter dem Namen ISO 8859 kodierte Zeichensatzfamilie besteht aus den Zeichensätzen ISO 8859-1 bis ISO 8859-15. Jeder Zeichensatz ISO 8859-X umfasst die 128 US-ASCII-Zeichen und ergänzt sie um weitere 128 Zeichen. Für die meisten westeuropäischen Sprachen, z. B. das Deutsche, Dänische oder Französische, ist ISO 8859-1, auch ISO Latin-1 genannt (Abbildung 1.3) relevant. ISO 8859-7 deckt das griechische Alphabet ab und ISO-8859-15 ersetzt im Wesentlichen das internationale Währungssymbol mit dem Eurozeichen. Die *Code Page* 1252 (Abbildung 1.4) von Microsoft Windows ist in weiten Teilen identisch mit ISO 8859-1, ersetzt aber einige Kontrollzeichen von ISO 8859-1 durch druckbare Zeichen, u. a. das Eurozeichen.
Anfang der neunziger Jahre kamen Unicode und ISO/IEC 10646 heraus. Das Ziel dieser beiden in einem Sinne, den wir noch genauer kennen lernen werden, äquivalenten Zeichenkodierungen ist Universalität, also Texte aus sämtlichen Sprachen der Welt eindeutig zu kodieren. Die aktuellen Versionen Unicode 3.0 und ISO/IEC 10646-1:2000 kodieren 49.194 Zeichen, die alle modernen und viele klassische Sprachen abdecken. Unicode und ISO/IEC 10646 werden laufend um weitere historisch bedeutsame Zeichen und Sprachen ergänzt.

1.2 Was gehört zu einer Zeichenkodierung?

Die Kodierungen ISO 8859 und Unicode sind von zentraler Bedeutung für die Internetformate HTML und XML. Ohne internationale Standards für die Kodierung universeller Zeichensätze wäre der grenzüberschreitende Dokumentenaustausch im Internet und im Web zum Scheitern verurteilt.

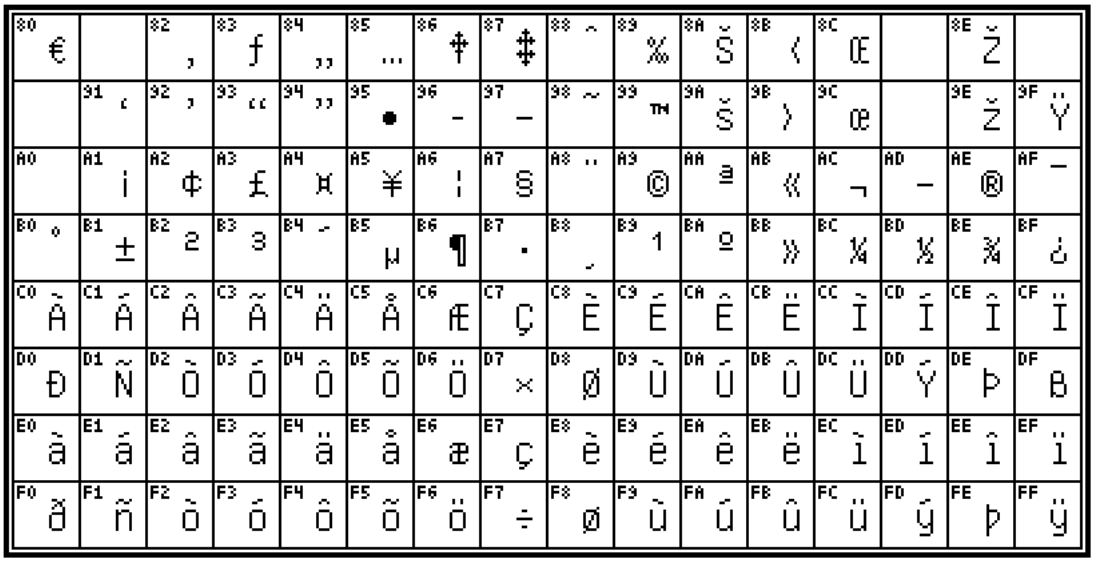


Abbildung 1.4: Der Windows-Zeichensatz 1252

Im Folgenden werden diese Kodierungen deshalb in dieser Kurseinheit genauer besprochen, und zwar im Rahmen eines abstrakten Kodierungsmodells für Zeichen [1,2,3].
Das Kodierungsmodell stellt einen konzeptuellen Rahmen dar, innerhalb dessen man die Kodierung von potentiell einigen Milliarden Zeichen in Bitmuster strukturieren und so besser verstehen kann.
In den folgenden Abschnitten werden die einzelnen Bestandteile des Kodierungsmodells besprochen und auf konkrete Zeichenkodierungen angewendet, wobei Unicode die zentrale Rolle spielt. Die einzelnen Bestandteile sind:

1. ein abstrakter Zeichensatz

2. eine Kodetabelle

3. ein Kodierungsformat

4. ein Kodierungsschema

5. eine Übertragungssyntax, die über so genannten „glatten Text“ hinaus geht

Die einzelnen Bestandteile des Kodierungsmodells entsprechen zunehmend konkreteren Repräsentationen von Zeichen, von einem abstrakten Begriff hin zu Bitmustern; zwischen einer abstrakten Repräsentationsebene und der nächstliegenden konkreteren Repräsentationsebene besteht eine Abbildung. Um über mehrere Ebenen hinweg von einer Zeichenposition in einer Kodetabelle direkt zu seiner Kodierung in einem Kodierungsschema zu gelangen, können die verschiedenen Abbildungen hintereinander ausgeführt werden und man erhält damit eine Zeichenkarte (engl. *Character Map*).

1.3 Abstrakter Zeichensatz

Die oberste, abstrakteste Ebene des Kodierungsmodells ist der abstrakte Zeichensatz, der aus einem Satz d. h. einer Menge abstrakter Zeichen besteht. Ein abstraktes Zeichen ist eine Informationseinheit, die zur Repräsentation, Organisation oder Kontrolle von Text dient, also beispielsweise Buchstaben, Ziffern, Interpunktionszeichen, Akzente, graphische Symbole, ideographische Zeichen, Leerzeichen, Tabulatoren, Zeilenweiterschaltung und -vorschub (engl. *Line Feed* und *Carriage Return)*, Kontrollkodes für Auswahl-Markierungen (engl. *Start of Selected Area* und *End of Selected Area* etc.). Der Zeichensatz von US-ASCII beispielsweise besteht aus 33 Kontrollzeichen und 95 druckbaren Zeichen.
Ein abstrakter Zeichensatz beinhaltet jedoch noch keine Ordnung oder Nummerierung der im Zeichensatz enthaltenen Zeichen; dieser Aspekt kommt in der nächsten Stufe des Kodierungsmodells in Form einer Kodetabelle hinzu.

Abstrakte Zeichensätze wurden in der Regel festgelegt, um alle Texte einer bestimmten Sprache oder Sprachfamilie angemessen zu kodieren. Der Anspruch, alle lebenden und alle historisch bedeutsamen Sprachen mit einem einzigen Zeichensatz kodieren zu können, führte zu den (identischen) Zeichensätzen von Unicode und ISO/IEC 10646, dem *Universal Character Set (UCS)*.
Zur visuellen Präsentation von Text dienen abstrakte graphische Einheiten, die Glyphen genannt werden. Glyphen entsprechen auf der Präsentationsebene den Zeichen auf der Kodierungsebene. Die Liste

A, A, **A**, *A*, A, ***A***

enthält graphische Ausprägungen des Glyphen für den Buchstaben „A“, so genannte Glyphenbilder. Nach Design, Größe, Gewicht und anderen Charakteristika zusammengehörige Glyphenbilder bilden einen Font.
Die Beziehung zwischen abstrakten Zeichen und Glyphen ist komplex. Tabelle 1.2 gibt eine Gegenüberstellung. Im einfachsten Fall präsentiert sich ein einzelnes Zeichen als genau ein Glyph. Einzelne Glyphen können jedoch auch ganze Folgen von abstrakten Zeichen darstellen, wie etwa bei den Ligaturen „fi“, „fl“, „ff“ oder „ffl“. Ein einziges abstraktes Zeichen mit Akzent, wie das Zeichen „ö“, kann durch zwei Glyphen dargestellt sein, nämlich den Glyphen für das Basiszeichen „o“ und den darüber positionierten Glyphen für den Akzent. Im Tamilischen gibt es Buchstaben, die durch ein Paar von Glyphen repräsentiert werden; dieses Glyphenpaar umrahmt dann in der formatierten Sicht einen weiteren Glyphen, der den Nachbarbuchstaben im Text darstellt. Im Arabischen schließlich gibt es *kontextuelle Formen*; d. h. ein Zeichen wird je nach dem Kontext seiner Nachbarzeichen durch Glyphen ganz verschiedener graphischer Form dargestellt. Ähnliche Phänomene sind aus der mittelalterlichen Satztechnik bekannt: Gutenberg beispielsweise verwendete für ein- und denselben Buchstaben verschieden breite Glyphen oder abkürzende Symbole für Wörter oder Silben, um einen gleichmäßigen rechten Rand zu erzeugen; hier gibt also die Formatierung den Kontext für die Wahl eines von mehreren Glyphen. In mathematischen Ausdrücken stellt die Semantik einen Kontext für die Auswahl von Glyphen dar. Beispielsweise kann ein langer oder kurzer Pfeil gewählt werden, je nachdem ob es sich um die Typfestlegung einer Funktion (f: A −! B) oder um eine Produktion in einer Grammatik (S ! [S]) handelt.

Abstrakte Zeichen	**Glyphen**
A	*A*
f+i	Fi
f+f+l	Ffl
ö	o+¨
→	oder →

Tabelle 1.2: Das Verhältnis von abstrakten Zeichen und Glyphen

In vielen Fällen sind die elementaren Texteinheiten, die als abstrakte Zeichen Eingang in einen Zeichensatz finden sollen, nicht offensichtlich. Sowohl aus den verschiedenen Sprachen, deren Texte kodierbar sein sollen, als auch aus den verschiedenen Verarbeitungsfunktionen, die auf die Texte Anwendung finden sollen, können Anforderungen und Konflikte erwachsen.
Im Schwedischen beispielsweise sind „ä“ und „ö“ Buchstaben für sich, die hinter „z“ im Alphabet angeordnet sind. Im Französischen dagegen gibt die Diaräse über einem Vokal lediglich einen Hinweis, dass der Vokal separat zu sprechen ist, etwa in „duët“. Anstelle eines einzigen abstrakten Zeichens „e mit Diaräse“ wären also die beiden abstrakten Zeichen „e“ und „¨“ mit zwei getrennten, aber kombinierbaren Glyphen natürlicher.
Für die Darstellung eines deutschen Textes in gedruckter Form ist die Unterscheidung von Groß- und Kleinbuchstaben im Zeichensatz äußerst bequem; für Sortierfunktionen oder Vergleiche dagegen stellt sie eine (wenn auch kleine) Hürde dar. Im Spanischen gilt „ll“ für das alphabetische Sortieren als ein einziger Buchstabe, für die Formatierung wird „ll“ als Folge von zwei Buchstaben behandelt.
Die Festlegung eines abstrakten Zeichensatzes erfordert also eine sorgfältige Abwägung von verschiedenen Alternativen, wobei Sprachen, Schriften und Bearbeitungsfunktionen zu berücksichtigen sind. Das zugegebenermaßen sehr allgemein formulierte Ziel bei der Entwicklung des Unicode-Zeichensatzes und der gesamten Unicode-Kodierung war es, die Implementierung nützlicher Verarbeitungsprozesse für Textdaten zu ermöglichen.

Unicode 3.0 formuliert zehn Designprinzipien für den Unicode-Standard. Zwei davon lassen sich bereits mit den Begriffen auf der Ebene des Zeichensatzes formulieren:

1. Unicode kodiert Zeichen, nicht Glyphen.

2. Unicode kodiert glatten Text, keine Formatinformation.

Im Folgenden beschäftigt sich diese Kurseinheit fast ausschließlich mit abstrakten Zeichen als den elementaren zu kodierenden Einheiten eines Schriftsystems. Glyphen spielen bei der Formatierung von Texten eine Rolle und liegen außerhalb unseres Themenspektrums.

1.4 Kodetabelle

Auf der nächsten Stufe des Kodierungsmodells steht die Kodetabelle, die den abstrakten Zeichen im Zeichensatz eine Kodeposition zuweist. Eine Kodeposition ist immer eine natürliche Zahl größer oder gleich null.
Der Koderaum einer Zeichenkodierung ist immer ein Abschnitt der nichtnegativen natürlichen Zahlen, der die Null und alle Kodepositionen enthält. Der Koderaum kann jedoch auch Zahlen enthalten, die keine zulässigen Kodepositionen sind.
Kodepositionen werden üblicherweise in Hexadezimalnotation angegeben, wodurch auch eine natürliche Beziehung zu Bitmustern hergestellt wird. Diese Beziehung kommt auf der nächsten Stufe des Kodierungsmodells, dem Kodierungsformat, zum Tragen. Im Folgenden werden kurz die wesentlichen Fakten der Binärkodierung und der Hexadezimalnotation wiederholt.
Ein Bit ist eine der beiden Ziffern 0 oder 1. Ein Byte, manchmal auch Oktett genannt, ist eine Sequenz von 8 Bits. Ein Wyde ist eine Sequenz von zwei Bytes oder sechzehn Bits.
Die sechzehn möglichen Sequenzen von vier Bits werden üblicherweise mit den Ziffern 0,...,9 und den Buchstaben A,...,F - auch Hexadezimal-Ziffern genannt - bezeichnet. Im Folgenden werden die Bits **0** und **1** fett gedruckt, um sie von den Hexadezimal-Ziffern 0 und 1, also 4-Bit-Folgen, zu unterscheiden.
Die Hexadezimal-Ziffern 0,...,9,A,...,F haben die natürlich-zahligen Werte von null bis fünfzehn (im Dezimalsystem). Damit kann dann auch jede Hexadezimal-Zahl, also jede Sequenz von Hexadezimal-Ziffern $h_n...h_0$ als natürliche Zahl im Hexadezimalsystem aufgefasst werden, nämlich als $h_0 + h_1 16 + h_2 16^2 + ... + h_n 16^n$, wobei die Zeichenfolge „16“ für die natürliche Zahl sechzehn in Dezimalschreibweise steht. Tabelle 1.3 drückt diese Beziehung zwischen numerischen Werten und Bitmustern aus.
Im Folgenden wird darüber hinaus bei einer Hexadezimal-Zahl nicht zwischen dem numerischen Wert und dem Bitmuster differenziert, das sie repräsentiert. Es sollte jeweils aus dem Kontext klar werden, welche Interpretation gemeint ist.
Bytes oder Zahlen von 0 bis 255 (dezimal) können somit durch Sequenzen von zwei Hexadezimal-Ziffern dargestellt werden und Wydes oder Zahlen von 0 bis 65.535 (dezimal) durch Sequenzen von vier Hexadezimal-Ziffern.
Der fortlaufende Text enthält sowohl Dezimaldarstellungen als auch Hexadezimaldarstellungen von Zahlen parallel, ohne syntaktisch zwischen ihnen zu unterscheiden. Um Missverständnisse zu vermeiden, erscheint an einigen Stellen das Wort „dezimal“ in Klammern hinter einer Dezimaldarstellung; an anderen Stellen erscheint der Wert der Zahl in Worten (z. B. „sechzehn“).

Selbsttestaufgabe 1.1 Erstellen Sie eine Additions- und eine Multiplikationstabelle für Hex-Ziffern.

Wie Tabelle 1.4 auflistet, umfasst der Koderaum für US-ASCII die 128 (dezimal) Positionen von 0 bis 7F, der für ISO 8859-X die 256 (dezimal) Positionen von 0 bis FF, der für Unicode die 65.536 (dezimal) Positionen von 0 bis FFFF plus die 1.048.576 (dezimal) Positionen von 10000 bis 10FFFF und der für ISO/IEC 10646 die 2.147.483.648 (dezimal) Positionen von 0 bis 7FFFFFFF.
Die Koderäume können weiter strukturiert sein:

Hex-Ziffer	Bitmuster	Wert in Dezimalschreibweise
0	0000	0
1	0001	1

2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Tabelle 1.3: Die sechzehn Hex-Ziffern und ihre Repräsentationen als Bitmuster

Kodierung	Koderaum	Zahl der Kodepositionen (dezimal)
US-ASCII	0-7F	128
ISO 8859-X	0-FF	256
Unicode	0-10FFFF	65.536+1.048.576
ISO / IEC 10646	0-7FFFFFFF	2.147.483.648

Tabelle 1.4: Koderäume für verschiedene Kodierungen

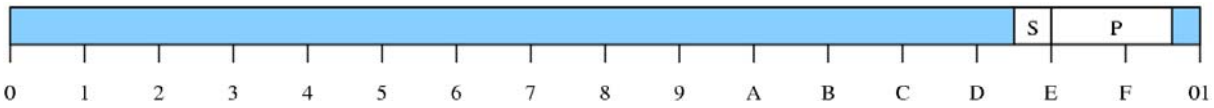


Abbildung 1.5: Der Unicode-Coderaum mit Surrogatpositionen (S) und privatem Bereich (P)

Die Positionen von 0 bis 1F und die Position 7F von US-ASCII sind beispielsweise Kontrollzwecken vorbehalten, ebenso die Positionen 80 bis 9F von ISO 8859-X. Beide Koderäume können durch ein Byte repräsentiert werden.
Der Unicode-Koderaum (siehe Abbildung 1.5) enthält Lücken an den 2.048 (dezimal) *Surrogatpositionen* von D800 bis DFFF und an den Positionen FFFE sowie FFFF. Diese Positionen sind für spezielle Funktionen vorgesehen, die unter den Aspekten Kodierungsformat und Kodierungsschema noch genauer besprochen werden; ihnen werden auch künftig keine Unicode-Zeichen zugeordnet werden.
Die 6.400 Unicode-Kodepositionen von E000 bis F8FF, knapp 10 % des Koderaums von 0 bis FFFF, sind für private Zwecke reserviert. Die Positionen sind etwa für Logos, für Icons oder für in speziellen Notationen (Musik, Tanz) benötigte Zeichen außerhalb des Unicode-Zeichensatzes freigegeben. Die Kodepositionen in diesem Bereich werden auch in späteren Unicode-Ergänzungen für private Verwendung freigehalten.

Selbsttestaufgabe 1.2 Berechnen Sie den prozentualen Anteil der Kodepositionen von E000 bis F8FF am Koderaum von 0 bis FFFF in Hexadezimalnotation.

Von den übrigen 57.086 (dezimal) Unicode-Positionen im Bereich von 0 bis FFFF sind 65 (dezimal) mit Kontrollzeichen belegt, ist 49.194 (dezimal) Positionen ein druckbares Zeichen zugeordnet und sind 7.827 (dezimal) Positionen noch frei für Erweiterungen von Unicode. Im nächsten Abschnitt bei der Diskussion von Kodierungsformaten wird erläutert, wie der Unicode-Koderaum im Bereich von 0 bis FFFF durch zwei Bytes und im Bereich von 10000 bis 10FFFF durch vier Bytes repräsentiert wird. Derzeit, d. h. bis zur Version 3.0, enthält die Unicode-Kodetabelle nur Werte im Bereich von 0 bis FFFF.

Der Koderaum von ISO/IEC 10646 ist konzeptuell unterteilt in 128 Gruppen von 256 Ebenen, wobei jede Ebene 256 Spalten mit je 256 Seiten, also insgesamt 65.536 Zeichen, enthält. Derzeit weist ISO/IEC 10646-1:2000 nur den Kodepositionen von 0 bis FFFF, die in Ebene 0 von Gruppe 0 liegen, Werte zu. Diese Ebene heißt auch Basic Multilingual Plane (BMP). Die Kodetabellen von Unicode Version 3.0 und ISO/IEC 10646-1:2000 sind Position für Position identisch. Das Unicode Konsortium und die ISO haben sich verpflichtet, diese Übereinstimmung auch bei späteren Ergänzungen der Kodetabelle aufrechtzuerhalten; sie haben eine Organisationsform wechselseitiger Liaisons geschaffen, die dies garantieren kann.
Unicode beinhaltet für jedes in seiner Kodetabelle kodierte Zeichen die folgenden Daten:

- die Kodeposition des Zeichens
- ein typisches Glyphenbild für das Zeichen
- einen Namen
- semantische Information

Die semantische Information ist eine Spezialität von Unicode. ISO/IEC 10646 stellt diese Information nicht zur Verfügung.
Für die Kodeposition 00AF steht beispielsweise in Unicode die folgende Information zur Verfügung: Das Zeichen hat (1) den Unicode-Namen „Macron“, (2) alternative Namen „Overline“ und „APL Overline“ und ist (3) graphisch ein hochgestellter waagerechter Strich. Das Macron bewirkt (4) einen horizontalen Vorschub (*Spacing Character*) und steht (5) in Beziehung zu weiteren Unicode-Zeichen, u. a. dem Zeichen 0304 „Combining Macron“, das genauso aussieht wie das Macron selbst, aber beim Formatieren keinen Vorschub bewirkt (Nonspacing Character), sondern sich über das vorangehende Basiszeichen positioniert. Das Macron kann (6) zerlegt werden in das Leerzeichen 0020, gefolgt von dem Combining Macron 0304. Das Macron 00AF ist (7) im Unterschied zum Combining Macron kein Kombinationszeichen, sondern ein Basiszeichen.
Das Macron wurde, wie viele andere Unicode-Zeichen, aus Kompatibilitätsgründen in die Kodetabelle aufgenommen. Hier wirkt sich das Designziel der Konvertierbarkeit von existierenden Standards nach Unicode und zurück aus. Ein Zeichen, das nur aus Kompatibilitätsgründen in der Kodetabelle steht, lässt sich gemäß einer Unicode-Vorgabe immer zerlegen in eine Sequenz aus *Primärzeichen*.

Für Basiszeichen, die von Kombinationszeichen (Akzente und andere Modifikatoren) gefolgt sind, definiert Unicode einen Äquivalenzbegriff. Grob gesagt können Kombinationszeichen, die sich graphisch an unterschiedlichen Stellen am Basiszeichen positionieren, miteinander ausgetauscht werden. Beispielsweise können die beiden Kombinationszeichen 0307 („Combining Dot Above“) und 0323 („Combining Dot Below“) ihre Plätze tauschen, sodass die beiden Kodierungen 006F 0307 0323 und 0067 0323 0307 für ein „o“ mit einem Punkt über sich und einem Punkt unter sich miteinander äquivalent sind.
Auf dem Äquivalenzbegriff für Zeichenfolgen und der Zerlegung von Kompatibilitätszeichen in Folgen von Primärzeichen und umgekehrt der Komposition von Kompatibilitätszeichen aus Folgen von Primärzeichen beruhen auch zwei Normalisierungen von Unicode Zeichenketten, nämlich die Precomposed Form und die Decomposed Form.
Webtexte sollten stets Zeichen benutzen, die schon soweit wie möglich zusammengesetzt sind; verbleibende Kombinationszeichen sollten in normierter Reihenfolge angegeben werden. Webanwendungen sollten so liberal wie möglich sein bei Texten, die sie einlesen, und so konservativ wie möglich bei Texten, die sie ausgeben, um Kompatibilitätsprobleme zu vermeiden.
Von den zehn Designprinzipien, die Unicode anführt, lassen sich fünf auf der Ebene der Kodetabelle erklären:

Kodierung/Kodierungsformat	**Koderaum/Kodeeinheit/Länge**
US-ASCII / kanonisch	0-7F / 1Byte / fest (1)
ISO 8859-X / kanonisch	0-FF / 1 Byte / fest (1)
ISO / IEC 10646, Ebene 0 / UCS-2	0-FFFF / 1 Wyde / fest (1)
ISO / IEC 10646 / UCS-4	0-7FFFFFFF / 2 Wyde / fest (1)
Unicode 3.0 / UTF8	0-10FFFF / 1 Byte / variabel (1-4)
Unicode 3.0 / UTF16	0-10FFFF / 1 Wyte / variabel (1-2)

Tabelle 1.5: Kodierungsformate für verschiedene Kodierungen

1. Unifikation: Unicode repräsentiert Zeichen, die in verschiedenen Alphabeten vorkommen, aber vom Aussehen her ähnlich sind, nur einmal.

2. Konvertierbarkeit zwischen etablierten Standards: Eine eineindeutige Abbildung aus einem etablierten Standard nach Unicode soll ermöglicht werden. Zeichenpositionen aus einem einzelnen international verbreiteten Zeichensatz, die nach dem Designprinzip der Unifikation eigentlich miteinander identifiziert werden müssten, erhalten trotzdem getrennte Unicode-Kodepositionen.

3. Semantik: Unicode definiert semantische Eigenschaften für Zeichen.

4. Dynamische Komposition: Jedes Basiszeichen kann mit beliebig vielen Kombinationszeichen gleichzeitig gepaart werden.

5. Charakterisierung äquivalenter Kodierungen: Für die verschiedenen Kodierungen eines Basiszeichens mit Kombinationszeichen oder eines primären und eines aus Kompatibilitätsgründen in den Zeichensatz aufgenommenen Zeichens kann eine normalisierte Kodierung gefunden werden.

1.5 Kodierungsformat

Ein Kodierungsformat für eine Kodetabelle legt Bitrepräsentationen für die Kodeposition fest. Dazu bedient es sich einer Kodeeinheit, die in der Regel aus acht oder sechzehn Bits besteht. Ein Kodierungsformat bildet dann die Positionen eines Koderaums in Sequenzen von Kodeeinheiten und somit in Bitmuster ab. Wird jede Kodeposition einer Kodetabelle auf die gleiche Anzahl von Kodeeinheiten abgebildet, sprechen wir von einem Kodierungsformat fester Länge; andernfalls hat das Kodierungsformat variable Länge.
Ein kanonisches Kodierungsformat fester Länge ergibt sich aus der Dualzahldarstellung von Kodepositionen. Die Anzahl der jeweils benötigten Kodeeinheiten ergibt sich aus der Größe des Koderaums und der Zahl der Bits in einer Kodeeinheit. Tabelle 1.5 gibt eine Übersicht über die verschiedenen Kodierungsformate.
Für US-ASCII bzw. seine ISO 646-Varianten sowie für die ISO 8859-X-Kodetabellen ist das kanonische Kodierungsformat das einzig gebräuchliche. Für die höchstens 256 (dezimal) vielen Positionen genügen eine Kodeeinheit von acht Bits sowie eine Kodeeinheit pro Zeichenposition.
Die beiden kanonischen Kodierungsformate für die Koderäume von 0 bis FFFF bzw. von 0 bis 7FFFFFFF heißen UCS2 und UCS4. Beide Kodierungsformate repräsentieren eine Kodeposition mit genau einer Kodeeinheit; im Falle von UCS2 ist die Kodeeinheit 16 Bits lang und im Falle von UCS4 ist sie 32 Bits lang.
Prinzipiell kann eine Kodierung mehrere Kodierungsformate definieren und das kanonische Kodierungsformat muss nicht darunter sein. Unicode 3.0 beispielsweise definiert nur zwei Kodierungsformate, nämlich UTF8 und UTF16, wobei UTF für Unicode Transfer Format steht. UTF8 hat eine Kodeeinheit von 8 Bits Länge und repräsentiert Kodepositionen mit einer bis vier Kodeeinheiten. UTF16 hat eine Kodeeinheit von 16 Bits Länge und repräsentiert Kodepositionen mit einer bis zwei Kodeeinheiten.
Die vorliegende Kurseinheit wird sich nun zunächst mit UTF16 beschäftigen. Ursprünglich war Unicode für einen Koderaum von 0 bis FFFF ausgelegt, sodass sechzehn Bits genügt hätten, um Kodepositionen darzustellen. Tatsächlich hätte dieser Koderaum ja auch mindestens bis Version 3.0 ausgereicht. Als jedoch klar wurde, dass zumindest für alle historisch interessanten Sprachen und Alphabete der Koderaum nicht ausreichen würde, und weil man mit ISO/IEC 10646 und seinem größeren Koderaum kompatibel bleiben wollte, wurde das Konzept der Surrogat-Kodepositionen eingeführt und der Koderaum von Unicode bis zur Position 10FFFF erweitert. Das Kodierungsformat UTF16 benutzt jeweils ein Paar von Surrogatpositionen, um Positionen jenseits von FFFF darzustellen.

UTF16 stellt wie die kanonische Kodierung jede gültige Kodeposition im Bereich von 0 bis FFFF durch ein einziges Wyde dar. Die ungültigen Kodepositionen im hohen Surrogatbereich von D800 bis DBFF und in dem niedrigen Surrogatbereich von DC00 bis DFFF entsprechen den hohen und niedrigen Surrogat-Wydes der Form **110110xxxxxxxxxx** und **110111xxxxxxxxxx**, in denen jeweils zehn Bitpositionen für 2^{10} verschiedene Werte frei wählbar sind. Die Gegenrechnungen DBFF+1-D800=DC00-D800=400=4x16^2 (dezimal)=2^{10} (dezimal) und DFFF+1-DC00=E000-DC00=400=2^{10} (dezimal) bestätigen, dass in jedem der beiden Surrogatbereiche 2^{10}, also 1024 (dezimal) ungültige Kodepositionen liegen. Ein Paar von Surrogatpositionen, von denen die erste im hohen und die zweite im niedrigen Bereich liegen, ist somit eindeutig charakterisiert durch zwanzig Bits, die wiederum die 2^{20} Werte von 10000 bis 10FFFF repräsentieren können:
10FFFF+1-10000=110000-10000=100000=16^5 (dezimal)=2^{20} (dezimal).
Es ist deshalb stimmig, wenn UTF16 eine Kodeposition P im Bereich von 10000 bis 10FFFF durch die kanonische Bitdarstellung der beiden hohen und niedrigen Surrogatpositionen H und L repräsentiert, wobei sich H und L wie folgt berechnen:

$$H = (P - 10000)\ DIV 400 + D800,\ L = (P - 10000) MOD 400 + DC00.$$

$$P = (H\ D800) 400 + (L\ DC00) + 10000.$$

Einheiten	Darstellungsform	freie Bits	Maximum
1	0xxxxxxx	7	7F
2	110xxxxx 10xxxxxx	11	7FF
3	1110xxxx $(10xxxxxx)^2$	16	FFFF
4	11110xxx $(10xxxxxx)^3$	21	1FFFFF
5	111110xx $(10xxxxxx)^4$	26	3FFFFFF
6	1111110x $(10xxxxxx)^5$	31	7FFFFFFF

Tabelle 1.6: Grunddaten zu Kodierungseinheiten

Umgekehrt bestimmen eine hohe und eine niedrige Surrogatposition H und L eine Unicode-Position P im Bereich von 10000 bis 10FFFF wie folgt:

$$P = (H - D800) \cdot 400 + (L - DC00) + 10000.$$

Selbsttestaufgabe 1.3 Wie wird die Unicode-Position F0000 unter UTF16 dargestellt?

Selbsttestaufgabe 1.4 Welche Unicode-Position wird unter UTF16 durch die beiden Surrogat-Wydes DAFF und DEFF dargestellt?

Das zweite Unicode-Kodierungsformat, nämlich UTF8, hat die besondere Eigenschaft, die ersten 128 Kodepositionen kanonisch mit einem Byte darzustellen. UTF8 ist damit US-ASCII-transparent.
Die Kodierungseinheit von UTF8 ist 8 Bits lang. UTF8 stellt jede Position im Bereich von 0 bis FFFF mit ein bis drei Kodierungseinheiten, im Bereich von 0 bis 10FFFF mit ein bis vier Kodierungseinheiten und im Bereich von 0 bis 7FFFFFFF mit ein bis sechs Kodierungseinheiten dar.
Die UTF8-Darstellung einer Position mit einer einzigen Kodierungseinheit hat die Form **0xxxxxxx** mit einer führenden **0** und beliebigen Bits an den mit **x** markierten Stellen. Die UTF8-Darstellung einer Position mit n Kodierungseinheiten hat die Form $\mathbf{1^n 0 x^{7-n}}$, gefolgt von n − 1 Kodierungseinheiten der Form **10xxxxxx**. Bei drei Kodierungseinheiten sind also genau sechzehn Bits frei wählbar, sodass mit drei Kodierungseinheiten unter UTF8 Positionen bis FFFF darstellbar sind.
Die mit n Kodierungseinheiten darstellbaren Positionsbereiche lassen sich aus der Tabelle 1.6 entnehmen.
Für die Darstellung einer Kodeposition unter UTF8 muss immer die kleinstmöglichste Zahl von Kodierungseinheiten gewählt werden. Eine Kodeposition im Bereich von 800 bis FFFF muss also mit drei Kodierungseinheiten und darf nicht − obwohl das technisch möglich wäre − mit vier Kodierungseinheiten dargestellt werden.

Die Umkehrfunktion von UTF8, die aus einer Folge von Kodeeinheiten eine Kodeposition berechnet, darf jedoch auch zu lange Darstellungen korrekt umrechnen. Die zwei Bytes C1BF dürfen also unter UTF8 in die Position 7F zurückgerechnet werden, obwohl die UTF8-Darstellung der Position 7F das Byte 7F ist.
Das Unicode-Designprinzip der Effizienz lässt sich auf Ebene von Kodierungsformaten erläutern. Für jedes der beiden Unicode-Kodierungsformate UTF8 und UTF16 lässt sich in einem Strom von Positionsdarstellungen von jeder Kodierungseinheit aus der Anfang der zugehörigen Positionsdarstellung mit beschränktem Backup finden. Im Falle von UTF16 muss höchstens um eine Kodierungseinheit zurückgegangen werden, im Falle von UTF8 höchstens um drei Positionen.

Selbsttestaufgabe 1.5 Kodieren Sie die ASCII-Zeichenfolge „<?xml“ ohne die Anführungsstriche unter UTF8 und UTF16.

Selbsttestaufgabe 1.6 Zeigen Sie, dass unter UTF8 die Kodierungen der Positionen 192 bis 255 alle dasselbe führende Byte haben, nämlich C3.

Selbsttestaufgabe 1.7 Ein Textsystem stellt das Zeichen „ü“ als Kombination zweier Glyphen Ã und ¼ dar. Überlegen Sie, in welchem Kodierungsformat der Textstrom vorliegen könnte und welche Annahmen das Textsystem über das Kodierungsformat macht.

1.6 Kodierungsschema

Damit Daten über Netzwerke zuverlässig ausgetauscht werden können, müssen sie in eine Folge von Bytes serialisiert werden. Mithilfe eines Kodierungsformats ist es bis jetzt gelungen, einen Text als Folge von Kodierungseinheiten darzustellen. Ein Kodierungsschema hat nun die Aufgabe, zu einem Kodierungsformat zusätzlich festzulegen, wie die Kodierungseinheiten in Bytefolgen zu serialisieren sind.
Ist die Kodierungseinheit eines Kodierungsformats selbst schon acht Bits lang, so ist nichts mehr festzulegen und das Kodierungsschema ist mit dem Kodierungsformat identisch. Man kann also UTF8 sowohl als Kodierungsformat als auch als Kodierungsschema ansehen.
Ist die Kodierungseinheit ein Wyde, wie bei UTF16, so gibt es zwei Möglichkeiten der Serialisierung: *big endian* (das höherwertige Byte kommt zuerst) und *little endian*. Dementsprechend gibt es zu UTF16 zwei Kodierungsschemata, genannt UTF16-BE und UTF16-LE. Analoges gilt für UCS2 und UCS4.
Mit dem *Byte Order Mark* (BOM) an Position FEFF kann ein UTF16-repräsentierter Datenstrom signalisieren, welche der beiden Serialisierungen, *big endian* oder *little endian*, vorliegt. Da FFFE keine zulässige Kodeposition in Unicode ist, lässt sich, wenn als die ersten zwei Bytes FF und FE gelesen werden, schließen, dass das Kodierungsschema UTF16-LE vorliegt. Ein BOM am Anfang eines UTF16-kodierten Datenstroms ist nicht Bestandteil der Daten und wird − abgesehen von seiner Signalwirkung für die Serialisierung − ignoriert. Tritt die Kodeposition FEFF dagegen an anderer Stelle im Datenstrom auf, so wird sie als das entsprechende Unicode-Zeichen *Zero Width No-Break Space* interpretiert, das dieser Position zugewiesen ist.
Generell darf ein Unicode-Zeichenstrom am Anfang mit dem zusätzlichen Zeichen BOM versehen werden, um Anwendungen, die über keine Metainformation über die Natur des Datenstroms verfügen, zu signalisieren, dass es sich um Unicode-Daten handelt und welches Kodierungsschema vorliegt. Das erste BOM eines Zeichenstroms bildet dann keinen Teil der eigentlichen Daten.

Selbsttestaufgabe 1.8 Was ist die Kodierung des BOM FEFF unter dem Kodierungsschema UTF8? Überprüfen Sie, ob das Vorausschicken eines BOM ausreicht, um zwischen den bisher besprochenen Kodierungsschemata zu differenzieren.

1.7 Jenseits von glattem Text – eine Übertragungssyntax

Die vorliegende Kurseinheit hat sich in den bisherigen Abschnitten zu Zeichenkodierungen darauf konzentriert, wie man glatten Text als Folge von Unicode-Zeichen kodieren kann. Damit kann man zunächst den reinen Inhalt von strukturierten Dokumenten handhaben. Da auch eingebettetes Markup,

wie z. B. bei den Tags der Form <XXX> und </XXX>, eine Folge von Zeichen ist, scheint es auf den ersten Blick, als wäre das Problem, strukturierte Dokumente zu kodieren, bereits vollständig gelöst.
Es ergibt sich jedoch eine zusätzliche Komplikation, da Markuptext von Inhaltstext syntaktisch unterscheidbar sein muss. Zur Abgrenzung von Markup und Inhalt werden bestimmte Funktionszeichen eingesetzt, die dann außerhalb ihrer syntaktischen Rolle weder im Markup noch im Inhalt im Klartext vorkommen dürfen. In XML ist „<“ ein solches Funktionszeichen, das den Anfang eines Tags charakterisiert. Das Zeichen „<“ darf im Inhaltstext eines XML-Dokuments nicht vorkommen.
Die klassische Methode, Funktionszeichen in glattem Text unterzubringen, ist die Verwendung von *Escape*-Zeichen, die selbst nicht wörtlich als Bestandteil des Textes verstanden werden, sondern die Präsenz eines Zeichens signalisieren, das nicht direkt im Text vorkommen darf. Im Falle von XML bezeichnet die Formel &#<<Hexziffern>>; oder als &<<Dezimalziffern>>; im Inhaltstext das Unicode- Zeichen an Position <<Hexziffern>> bzw. an Position <<Dezimalziffern>>. Wir können also ein „<“ im Inhaltstext z. B. als < notieren und das zusätzliche Funktionszeichen „&“ durch &.
Im Allgemeinen wird glatter Text nicht direkt als Folge von Zeichen auftreten. Vielmehr wird die Zeichenfolge, die einen Text ausmacht, konstruiert aus einer weiteren Folge von *Eingabezeichen*, von denen ganze Teilsequenzen einzelne Zeichen des Textes repräsentieren. In der Praxis hat man es also mit einer zweistufigen Kodierung von Text zu tun.
Die hier dargestellte Technik, Zeichen entweder durch sich selbst oder durch spezielle Zeichenfolgen wie &#<<Hexziffern>>; zu kodieren, löst insgesamt drei bekannte Probleme:
Das erste Problem, wie sich in glattem Text Funktionszeichen an Stellen unterbringen lassen, an denen sie nicht im Klartext vorkommen dürfen, wurde bereits angesprochen.
Dieselbe Technik ermöglicht es zweitens auch, Zeichen in einen Text einzufügen, die das verwendete Eingabewerkzeug nicht unterstützt. So kann man also beispielsweise in ein XML-Dokument mit einer amerikanischen Tastatur den dort nicht unterstützten Umlaut „ä“ als ä eingeben.
Drittens ist es üblich, Paare aus den beiden ASCII-Steuerzeichen LF (engl. *linefeed*, *newline*) und CR (engl. *carriage return*) sowie jedes der beiden Zeichen für sich allein als ein- und dasselbe Zeilenendezeichen zu interpretieren. Auf diese Weise werden die beiden gängigen verschiedenen Konventionen, im Text ein Zeilenende zu signalisieren, vereinheitlicht. Werkzeuge, die die Texte verarbeiten, verfügen damit über eine plattformunabhängige Methode, Zeilen zu nummerieren.
Bisher wurde die Technik, Zeichen durch Zeichenfolgen spezieller Syntax zu kodieren, am Beispiel von XML illustriert. Die Technik ist aber Standard in allen Bereichen, in denen Texte erstellt werden, speziell auch bei der Erstellung von Programmtexten.
Java-Programme beispielsweise dürfen beliebige Unicode-Zeichen der Basic Multilingual Plane, also im Kodebereich von 0 bis FFFF, enthalten, z. B. in Namen, in Literalen für Zeichenketten oder in Kommentaren. Grundsätzlich kann im Programmtext jedes Zeichen durch sich selbst oder durch eine Sequenz \u<<xxxx>> mit einer ungeraden Anzahl von Zeichen „\“, einer positiven Anzahl von Zeichen „u“ und einer Sequenz <<xxxx>> von genau vier Hexziffern dargestellt werden. Die einzige Ausnahme betrifft das Zeichen „\“, das nur dann für sich selbst stehen darf, wenn es nicht in einer Sequenz der Form \u<<xxxx>> vorkommt und somit missinterpretiert werden kann.

Das klassische Java-Programm HelloWorld

```
public class HelloWorld {
public static void main(String[] args) {
System.out.println("Hello World");
}
}
```

dürfte also auch in folgender Form erscheinen:

```
\u0070ublic class HelloWorld {
public static void main(String[] args) {
System.out.println("\uuuu0048ello World");
\\\u007D
}
```

Die Java-Notation macht es also möglich, einen beliebigen Unicode-Text mit Zeichen im Bereich zwischen 0 und FFFF als einen 7-Bit-ASCII-Text zu kodieren.

1.8 Unicode-Werkzeuge

Bei der Diskussion von Werkzeugen zur Bearbeitung von Unicode-Texten unterscheidet man zwischen der Eingabe und der Darstellung von Unicode-Zeichen. Wird ein genuiner Unicode-Editor gewünscht, so ist die Auswahl derzeit noch nicht allzu groß. Eine der wenigen Optionen ist ein Forschungsprototyp von der Duke University namens UniEdit, der eine Vielzahl von Kodierungsschemata und Eingabeformaten unterstützt (http://www.humancomp.org/uniintro.htm).
Am anderen Ende des Spektrums, was Unicode-Unterstützung angeht, kann jeder beliebige Editor für 7-Bit-ASCII verwendet und die damit erstellten Texte als Unicode-Texte in UTF8-Kodierung ausgegeben werden. Damit kann man direkt natürlich auch nur die 7-Bit-ASCII-Zeichen in eigenen Texten verwenden. Indirekt hat man über eine weitere Kodierungsstufe jedoch auch Zugriff zu dem vollen Unicode-Zeichenrepertoire. Im Falle der Java-Kodierung steht auch ein Werkzeug zur Verfügung, mit dem man die Java-Notation wieder dekodieren und in verschiedene Kodierungsformate übersetzen kann. Das Werkzeug heißt native2ascii und ist Bestandteil des Java Software Development Kit.

Selbsttestaufgabe 1.9 Installieren Sie, falls erforderlich, das Werkzeug native2ascii (http://java.sun.com) aus dem Java Software Development Kit. Erstellen Sie eine Textdatei mit Umlauten und anderen Zeichen außerhalb von US-ASCII und übersetzen Sie diese Datei mithilfe von native2ascii nach UTF8.

Selbsttestaufgabe 1.10 Erstellen Sie eine Textdatei, in der Sie in Java-Notation die Zeichen außerhalb des Koderaums von 0 bis FF referenzieren. Versuchen Sie, diese Datei mit native2ascii nach ISO Latin-1 bzw. Code Page 1252 zu übersetzen. Was passiert?

2 Audioformate

Menschen sind ständig von Schallwellen umgeben und nehmen diese auch mehr oder weniger bewusst wahr. Im Gegensatz zur visuellen Wahrnehmung ist der Mensch jedoch beim Hören nicht in der Lage, seine Ohren zu verschließen und so die Dauerbeschallung einfach auszublenden. Das menschliche Hörspektrum reicht von 20 Hz bis 20 000 Hz und Schallwellen in eben diesen Frequenzen werden Töne genannt. Jedoch gibt es auch Schallwellen, die nicht durch die Ohren wahrgenommen werden und dennoch den menschlichen Körper beeinflussen. Ein Beispiel hierzu ist der *Infraschall* mit sehr niedrigen Frequenzen etwa bei einer Größenordnung von 10 Hz. Werden Menschen diesen Frequenzen ausgesetzt, so können sie Unwohlsein und erhöhten Pulsschlag hervorrufen. Diesen Effekt macht sich beispielsweise auch die Unterhaltungsindustrie zu Nutze. Aus diesem Grund werden z. B. Filme mit Infraschallsoundtracks sehr intensiv empfunden [4].

Bei Schall handelt es sich also um Druckwellen der Umgebungsatmosphäre, die sich mehr oder weniger stark verbreiten. Zum Beispiel mithilfe von Schallplatten kann dieser Schall auf der Grundlage von *Analog*-Technologie dauerhaft konserviert werden und ist im Prinzip jederzeit wieder abspielbar. Jedoch ist hierbei die gleichmäßige Abspielgeschwindigkeit genauso wichtig wie die unterbrechungsfreie Wiedergabe – ansonsten werden die Klangabfolgen schnell als unangenehm empfunden. Wegen dieser Eigenschaften sind Tonträger zeitkontinuierliche bzw. zeitkritische Medien. Die Schallwellen sind zudem auch noch wertekontinuierlich, denn jede simple Sinuswelle besteht aus unendlich vielen Werten. Die Kunst der Digitalisierung besteht also darin, ein analog so nicht per Computer zu verarbeitendes Audiosignal in computergerechte Häppchen zu zerlegen und dies möglichst so zu tun, dass beim anschließenden Abspielen der neu entstandenen Audiodatei zumindest der Wiedererkennungseffekt beim menschlichen Hörer eintritt.

2.1 Digitalisierung

Wie weiter oben erläutert, sind die Klänge, die von einer akustischen Umgebung an das menschliche Ohr gelangen analog und können so zunächst nicht im Computer ohne weitere Vorbereitung verarbeitet werden, denn die Verarbeitungsverfahren in der moderneren EDV-basierten Informations- und Kommunikationstechnik beruhen ja gerade auf digitalen Repräsentationsmethoden für jegliche Art von Daten und Information. So müssen folglich auch die analogen Schallwellen für eine Verarbeitung in Multimedia- und Informationssystemen zunächst digitalisiert werden. Dieser Vorgang verläuft nach dem *Drei-Stufen-Modell*: Abtasten, Diskretisieren und Kodieren der Schallwelle und wird im folgenden Abschnitt näher erläutert.

2.2 Sampling, Quantisierung und Kodierung

Beim *Sampling* und der *Quantisierung* findet eine Abtastung des zeit- und wertekontinuierlichen Signals einer Schallwelle in gewissen Intervallen statt. Die *Samplingrate* wird als Frequenz der Abtastung in der Einheit *Hertz* (abgekürzt *Hz)* angegeben. Alle Werte, die ggf. nicht zu den durch die Samplingrate festgelegten Zeitpunkten erfasst worden sind, werden dementsprechend verworfen. Nach dem Sampling ist das Signal zwar in eine endliche Anzahl von Werten zerlegt, diese können aber grundsätzlich immer noch unendlich in ihrer Ausprägung hinsichtlich des Wertebereichs sein. Deshalb folgt nun der zweite Schritt: die *Quantisierung*.
Für den Computer nicht ausreichend präzise repräsentierbare und damit korrekt verarbeitbare Werte – wie etwa 1/3 oder Pi − werden dabei zunächst auf den nächsten diskret repräsentierbaren Wert gerundet. Welcher Wert der am nächsten liegende Wert ist, wird durch die *Quantisierungsintervalle* festgelegt. Was bei der Abtastung die Samplingrate ist, ist beim Diskretisieren die Bitrate. Durch die Quantisierung entstehen Ungenauigkeiten, deren Effekt auch als *Quantisierungsrauschen* bekannt ist. Je höher die gewählte Bitrate zur Quantisierung gewählt wird, desto geringer ist die Wahrscheinlichkeit, dass störendes Quantisierungsrauschen auftritt.

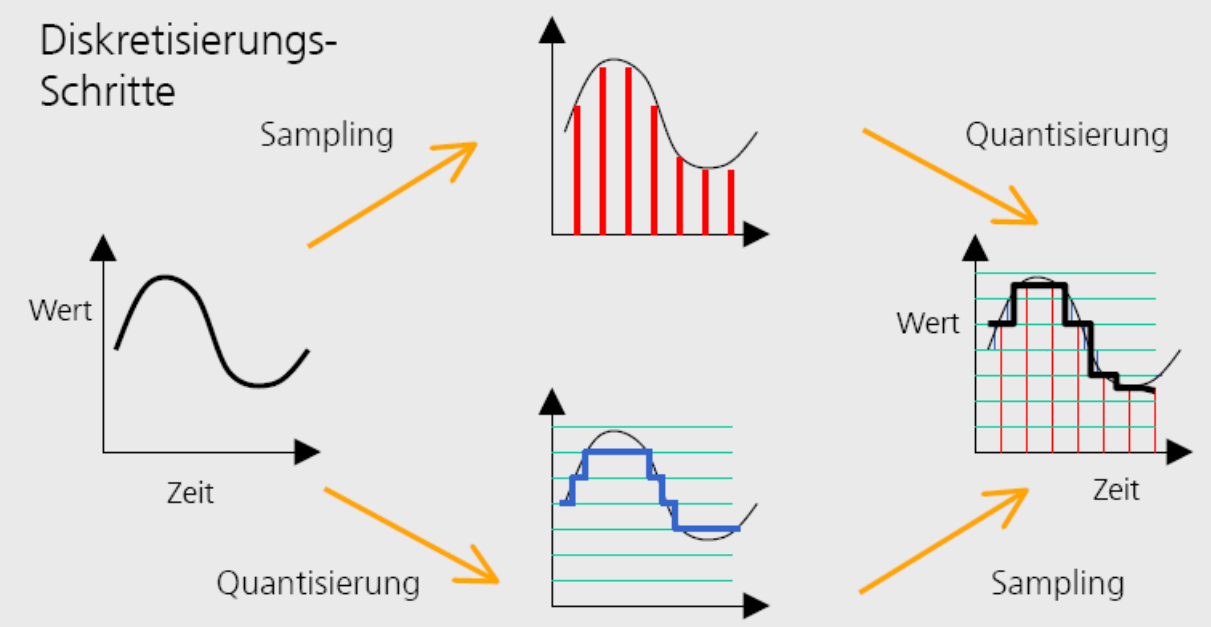


Abbildung 2.1: Sampling und Quantisierung, Quelle [5]

Der letzte Schritt des Drei-Stufen-Modells ist die *Kodierung*. Hierbei werden die verschiedenen Quantisierungsintervalle mit binären Codewörtern gekennzeichnet. Das so entstandene digitalisierte Signal wird mitsamt dem *Quantisierungsfehler* übertragen, der jedoch bei geschickter Wahl der Bitrate nicht im Bereich des Hörbaren liegt. Abbildung 2.1 zeigt den gesamten Vorgang noch einmal im Überblick. Das hier beschriebene Verfahren zur Analog-/Digitalwandlung von Signalen wird auch

Waveform-Encoding bzw. *PCM* (**P**ulse **C**ode **M**odulation) genannt. Es wird im weiteren Verlauf noch genauer erläutert.

Es gibt zunächst drei grundlegend unterschiedliche PCM-Verfahren. Welches der Verfahren zur Digitalisierung des Klanges ausgewählt werden sollte, hängt vom Anspruch an das Ergebnis ab. Sicher ist die *lineare PCM* bei hinreichend kleinen Quantisierungsintervallen sehr genau am Original, soll diese Datei jedoch anschließend zum Beispiel über das Internet übertragen werden, ist eine entsprechend hohe Bandbreite nötig. Die Quantisierungsintervalle sind also in diesem Fall alle gleich groß. Das so digitalisierte Signal hat somit eine gute Qualität – allerdings auf Kosten einer hohen Bitrate. Bei *dynamischer PCM* wird die Größe der Quantisierungsintervalle dynamisch angepasst, wobei zum Beispiel die Quantisierungsintervalle bei leisen Passagen kleiner gewählt werden und so das Quantisierungsrauschen geringer ausfällt. Die dynamische Anpassung kann mithilfe einer logarithmischen Funktion erreicht werden.

Die *differenzielle PCM* (abgekürzt DPCM) hingegen braucht eine ausreichend große Rechenleistung – dafür sind die Dateien kleiner. Die Abweichungen zwischen den einzelnen abgetasteten Werten sind oft nur gering. Bei diesem Verfahren wird daher lediglich die Differenz zwischen den aufeinanderfolgenden Sampling-Werten gespeichert. Optimale Ergebnisse werden hierbei per *Predictive Coding* erzielt (siehe unten). Das Prinzip der DPCM ist noch weiter entwickelt worden zur *adaptiven DPCM* (abgekürzt ADPCM).

Selbsttestaufgabe 2.1 Bei welchem PCM-Verfahren wird bei gleichem Ausgangsmaterial die geringste Dateigröße erreicht?

Für die Kodierung von Sprachdateien bieten sich anschließend auch die nicht-linearen Verfahren an, denn bei starken Signalen reichen größere Quantisierungsintervalle, wohingegen bei schwachen Signalen eine genauere Näherung gefragt ist. Es gilt also, je nach Einsatzgebiet, die Vorteile gegen die Nachteile abzuwägen. Wie später im weiteren Verlauf der Kurseinheit noch aus der Abbildung 3.2.1 hervorgeht, ist das Herausfinden der optimalen Abtastrate von essentieller Bedeutung. Deshalb folgt an dieser Stelle zunächst noch ein kleiner mathematischer Exkurs zum Abtasttheorem.

2.3 Exkurs: Abtasttheorem

Ist die Abtastrate zu gering, kann das Signal nur fehlerhaft abgebildet werden. Dem entgegen steht der Wunsch, möglichst wenige Werte zu erhalten, damit die anschließende Verarbeitung nicht zu umfangreich gerät. Von der optimalen Abtastrate handelt das berühmte *Abtasttheorem* nach *Nyquist*, *Kotelnikow*, *Raabe* und *Shannon* welches ein grundlegendes Theorem der Nachrichtentechnik, Signalverarbeitung und Informationstheorie ist. Zur Entstehungsgeschichte dieses Abtasttheorems lässt sich u. a. bei Wikipedia nachlesen:

> Claude Elwood Shannon formulierte das Theorem 1948 als Ausgangspunkt seiner Theorie der maximalen Kanalkapazität, d. h. der maximalen Bitrate in einem frequenzbeschränkten, rauschbelasteten Übertragungskanal. Dabei stützte er sich auf Überlegungen von Harry Nyquist (1928) zur Übertragung endlicher Zahlenfolgen mittels trigonometrischer Polynome und auf die Theorie der Kardinalfunktionen von Edmund Taylor Whittaker (1915) und seinem Sohn John Macnaughten Whittaker (1929) [2]. Unabhängig davon wurde das Abtasttheorem 1933 von Wladimir Alexandrowitsch Kotelnikow [3] in der sowjetischen Literatur eingeführt, was im Westen allerdings erst in den 1950er Jahren bekannt wurde. Ansätze zur Interpolation mittels Kardinalreihen oder ähnlicher Formeln lassen sich bis in die Mitte des 19. Jahrhunderts zurückverfolgen.

Das Abtasttheorem besagt, dass eine Signalfunktion, die nur Frequenzen in einem beschränkten Frequenzband enthält, wobei $f$max gleichzeitig die höchste auftretende Signalfrequenz ist, durch ihren diskreten Amplitudenwert im Zeitabstand $T_0 \leq \frac{1}{2 \cdot f_{\max}}$ vollständig bestimmt wird.

Aus diesem Theorem lässt sich folgern, dass ein Signal durch eine Abtastfrequenz, die doppelt so hoch ist wie die höchste im Signal vorkommende Frequenz, vollständig bestimmt werden kann. Die höchste Frequenz sei *f*max und man erhält somit $f_A \geq 2 \cdot f_{\max}$

Da der Mensch Frequenzen hören kann, die von etwa 20 Hz bis 22 kHz liegen, reicht dem Abtasttheorem zufolge bei *f*max := 22 kHz eine Samplingrate >= 44 kHz vollkommen aus um analoge Audiosignale optimal abzutasten.

2.4 Methoden der Komprimierung von Audiodaten

Nach der Digitalisierung der Audioinformationen liegen diese zwar nun als von einem Computer verarbeitbare und abspielbare Dateien vor, haben jedoch aus Sicht der notwendigen Kapazität zur Speicherung einen sehr hohen Bedarf. Eine Minute Hörerlebnis benötigt in dieser Form gespeichert immerhin schon rund 10 MB. Um also z. B. ein Musik-Album mit ungefähr einer Stunde Laufzeit abzuspeichern, sind etwa 600 MB nötig – soviel wie bequem auf eine CD passt. Wie vielfach behauptet wird, soll dies kein Zufall sein, sondern die Größe der CD wurde ursprünglich einmal so dimensioniert, dass genau Beethovens Neunte, dirigiert von Karajan, auf einer CD-ROM abgespeichert werden konnte. Zur Abspeicherung auf Festplatten, für die Übertragung übers Internet oder gar für die Nutzung in mobilen Abspielgeräten ist diese Datenmenge jedoch viel zu groß und so wurden folglich nach und nach verschiedene Ansätze zur Audiokomprimierung entwickelt und verbessert, die im weiteren Verlauf der Kurseinheit anhand einer Auswahl der gängigen Kompressionsmethoden und -verfahren überblicksartig vorgestellt werden.

2.4.1 Huffman-Kodierung

Die Huffman-Kodierung [6] gehört zum *Entropy Coding* Verfahren und beruht auf der Idee, dass der benötigte Speicherplatz für eine bestimmte Anzahl von Zeichen im Binärcode erheblich verringert werden kann, wenn die Zeichen, die häufig vorkommen nur wenig Stellen bei ihrer Darstellung und Speicherung als Binärzahl beanspruchen. Zu diesem Zweck wird also ein Binärbaum angelegt, bei dem die längsten Pfade zu den am seltensten vorkommenden Knoten führen. Bei jedem Knoten, den der Pfad berührt, wird der Binärzahl entweder eine 1 oder 0 hinzugefügt – abhängig von der Richtung, in die der Pfad anschließend weiterführt. Ein einfaches Beispiel soll dies nachfolgend verdeutlichen.
Der Satz „Barbara mag Rhabarber“ braucht bei 8-Bit ASCII Kodierung 168 Bits Speicherplatz. Um nach Huffman diesen Platz zu verringern, wird zunächst die Anzahl der Vorkommen für jedes Zeichen gezählt und als Gewichtung gemerkt:

B a r b m g R h e
1 6 4 3 2 1 1 1 1 1

Nun wird jedes Zeichen gemeinsam mit seiner Gewichtung als Knoten wie in Abbildung 2.2. dargestellt.

Abbildung 2.2: Knotengewichtung bei der Huffman-Codierung

Jetzt werden jeweils die Knoten mit der geringsten Gewichtung paarweise zusammengefasst und erhalten einen Elternknoten, der die Summe der Gewichtung der Kinder als eigene Gewichtung bekommt. Dies wird so lange wiederholt, bis ein Binär-Baum entstanden ist, wie er z. B. in Abbildung 2.3. dargestellt ist.

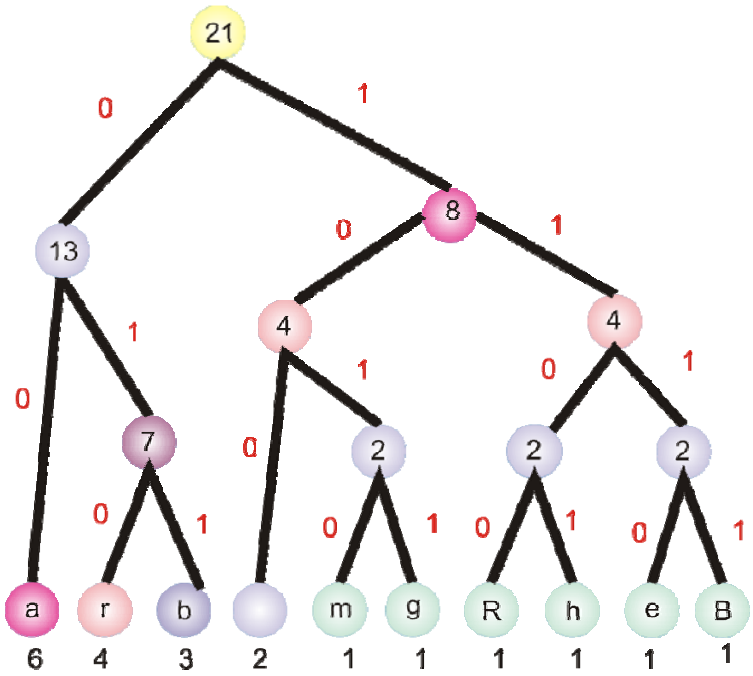


Abbildung 2.3: Der Huffman-Baum

Die neuen Binärcodes für die Zeichen werden nun gefunden, indem von der Wurzel – hier 21 – bis hinunter zum jeweiligen Buchstaben der Pfad verfolgt wird. Die jeweiligen Binärziffern werden entsprechend notiert. Da zum Beispiel nach „a“ zwei linke Kanten führen, ist die neue Kodierung für „a“ gleich „00“. Genauso wird zur Kodierung mit den restlichen Zeichen verfahren und es entsteht folgende Codetabelle:

a	r	b		m	g	R	h	e	B
00	010	011	100	1010	1011	1100	1101	1110	1111

Der so entstandene Code lautet:

„11110001001100010001001010001011100110011010001100010011111010“

und es werden nur noch 63 Bits benötigt um die Information zu speichern.

Die mit Huffman kodierten Zeichen haben also eine variable Bitlänge, die angepasst ist auf die Häufigkeit des Erscheinens des einzelnen Zeichens. Bei dem Satz „Barbara mag Rhabarber“ werden für die binäre Darstellung höchstens 4 Bit pro Zeichen verwendet, wodurch wertvoller Speicherplatz gespart wird, ohne dass der Inhalt der Botschaft verändert wird. In Bezug auf Audiokodierung wird mit dem Huffmanverfahren durchschnittlich eine Reduktion von 1:2 erreicht. Da die Kodierungsmethode nicht eindeutig ist, muss die Struktur des Baumes bei der so kodierten Datei mitgeliefert werden.

Selbsttestaufgabe 2.2 Geben Sie den Huffmann-Baum und die dadurch entstehende Codetabelle für die Zeichenfolge „Abrakadabra Simsalabim“ an.

2.4.2 Verdeckungsschwelle

Mit Verdeckungsschwelle bzw. *Maskierung* ist ein Verfahren gemeint, welches sich einen Selektionseffekt des menschlichen Ohres zu Nutze macht. Diesen Effekt kann jeder nachvollziehen, der schon einmal versucht hat sich an einer Autobahn zu unterhalten. Die Stimme des Gegenübers wird unhörbar, sobald ein LKW vorbeibraust. Die leisen Töne werden also durch gleichzeitiges

Auftreten von lauten Tönen überdeckt. Dieses Phänomen wird auch *simultane Verdeckung* genannt. Töne, die sowieso nicht wahrgenommen werden können, verbrauchen also nur Speicherplatz und können deshalb aus der Datei eliminiert werden.
Der Maskierungseffekt hält sogar noch über die eigentliche Abspielzeit des lauten Signals an. Ein extremes Beispiel hierzu: Jeder mit normalem Hörvermögen, der eine Viertelstunde neben einem laufenden Presslufthammer verbracht hat, wird auch eine ganze Zeit lang danach kaum noch etwas hören können. Flüstertöne nach dieser Art Störgeräusch kommen beim Zuhörer nicht an und können deshalb eingespart werden. Diese Verschiebung der Verdeckungsschwelle wird *zeitliche Maskierung* genannt.
Der deutsche Physiker Heinrich Barkhausen (1881 - 1956) hat den Maskierungseffekt verwendet, um den menschlichen Hörbereich in 24 kritische Bänder aufzuteilen (siehe Abbildung 2.4). Er hat dabei festgestellt, dass die Breite dieser Bänder nicht konstant ist, sondern sich mit der mittleren Bandfrequenz verändert. Für die Breite der Bänder wird daher in der Psychoakustik als Maßeinheit *Bark* verwendet.

Z Bark	f_u, f_o Hz	f_m Hz	Z Bark	Δfg Hz	10 lg Δfg* dB
0	0				
		50	0,5	100	
1	100				
		150	1,5	100	20
2	200				
		250	2,5	100	20
3	300				
		350	3,5	100	20
4	400				
		450	4,5	110	20
5	510				
		570	5,5	120	21
6	630				
		700	6,5	140	21
7	770				
		840	7,5	150	22
8	920				
		1,000	8,5	160	22
9	1,080				
		1,170	9,5	190	23
10	1,270				
		1,370	10,5	210	23
11	1,480				
		1,600	11,5	240	24
12	1,720				
		1,850	12,5	280	25
13	2,000				
		2,150	13,5	320	25
14	2,320				
		2,500	14,5	380	26
15	2,700				
		2,900	15,5	450	27
16	3,150				
		3,400	16,5	550	27
17	3,700				
		4,000	17,5	700	28
18	4,400				
		4,800	18,5	900	29
19	5,300				
		5,800	19,5	1,100	30
20	6,400				
		7,000	20,5	1,300	32
21	7,700				
		8,500	21,5	1,800	32
22	9,500				
		10,500	22,5	2,500	34
23	12,000				
		13,500	23,5	3,500	35
24	15,500				

* 10 lg Δfg = Zunahme des Pegels (in dB) von Ausschnitten aus weissem Rauschen bei Verbreiterung des Frequenzbandes von 1 Hz auf die Breite Δfg der Frequenzgruppe.
Z = Frequenzgruppe, f_u = untere Grenzfrequenz, f_o = obere Grenzfrequenz, f_m = Mittenfrequenz, Δfg = Bandbreite

Abbildung 2.4: Wertetabelle zur Einheit Bark

Für Frequenzen (*f*) unter 500 Hz ist ein Bark gleich *f*/100 und für die Frequenzen über 500 Hz wird ein Bark mit 9+4*log(*f*/1000) berechnet, wie bei [4] nachgelesen werden kann. Aufgrund der Signalstärke in einem Frequenzbereich können anhand der *Barkhausen-Bänder* die Dauer der Maskierung errechnet und Informationen zu den verdeckten Tönen können ohne Beeinträchtigung des Hörgenusses entfernt werden.

2.4.3 Predictive Coding

Predictive Coding ist eine Methode, bei der aus den bereits abgespielten Signalen das wahrscheinlich folgende Signal errechnet wird. Entspricht das folgende Signal nicht der Vorhersage, wird lediglich

die Differenz zum vorher angenommenen Signal gespeichert. Diese Differenzen brauchen weniger Speicherplatz als das ursprüngliche Signal und deshalb kann die Audiodatei so effektiv komprimiert werden.
Tilman Liebchen von der TU Berlin hat mit diesem Verfahren den bekannten *LPAC-Codec* (engl. ***L**ossless **P**redictive **A**udio **C**ompression*) entwickelt, der Audiodateien im *pac*-Format für Unixsysteme verlustfrei komprimieren kann. Dieser Codec hat mittlerweile seinen Nachfolger im MPEG-4 *LAC-Verfahren* (engl. ***L**ossless **A**udio **C**oding*) bzw. in Ogg-*FLAC-Verfahren (engl. **F**ree **L**ossless **A**udio **C**oding*) gefunden.

2.4.4 Transform Coding

Durch die *Transformationskodierung* werden Daten in einen besser zu komprimierenden, mathematischen Raum übertragen. Ein hierbei weit verbreitetes Verfahren ist die *Fourier-Transformation* von Zeit- nach Frequenzbereich. Die Schallwellen werden als Polynome aufgefasst und die Koeffizienten werden auf Häufigkeit ihres Vorkommens untersucht. Hierbei fallen kleinere Koeffizienten weg und so entsteht der übersichtliche Frequenzraum.

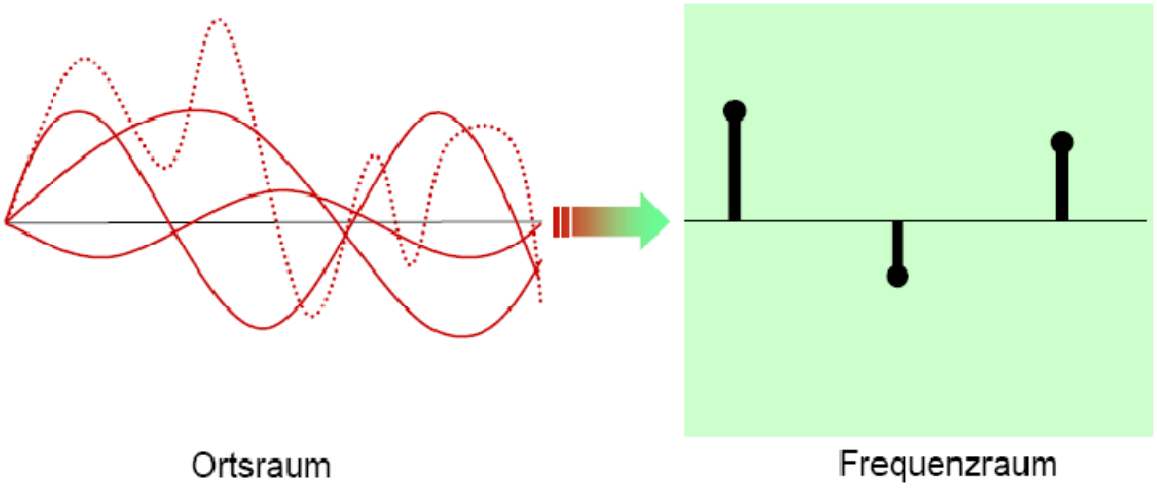


Abbildung 2.5: Fourier Transformation [7]

Die Formel für die DFT lautet für 2n:

$$f_j = \sum_{k=0}^{2n-1} x_k \, e^{-\frac{2\pi i}{2n} jk} \qquad j = 0, \ldots, n-1.$$

Die effektivsten Verfahren dieser Art sind die *diskrete Cosinustransformation* (DCT) und die schnelle Fourier-Transformation (engl. *Fast Fourier Transformation*, FFT), die deshalb als schnell bezeichnet werden kann, weil sie nach dem klassischen *Divide-and-Conquer* Prinzip arbeitet. Ausführliche Informationen zu den beiden Berechnungsverfahren können bei [8] nachgelesen werden.

2.4.5 Sub Band Coding

Beim *Sub Band Coding* wird das zu kodierende Audiosignal durch eine Filterbank vom Zeit- in den Frequenzbereich transformiert, wobei es in zum Beispiel bei MP3 in 32 Frequenzbänder mit je 625 Hz (Subbänder) gleicher Breite unterteilt wird.

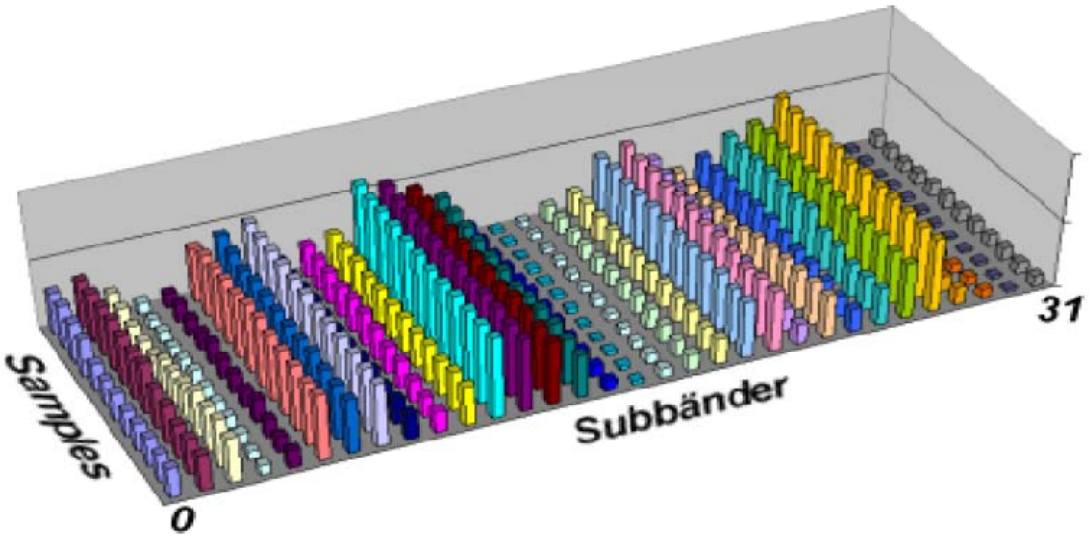


Abbildung 2.6: Subband Coding, Quelle [9]

Beim Sub Band Coding wird davon ausgegangen, dass für die korrekte Klangwiedergabe immer nur bestimmte Frequenzmaxima benötigt werden. Dieses Verfahren wird deswegen auch *selektive Frequenztransformation* genannt und eignet sich insbesondere gut zur Komprimierung von Sprachdateien [8]. Bei MP3 wird z. B. für 32 eingelesene Samples pro Subband ein Sample ausgegeben. Die Subbänder werden durch eine modifizierte diskrete Cosinus-Transformation (abgekürzt *MDCT*) jeweils nochmals in 18 Teilbereiche unterteilt. Dadurch ergibt sich eine höhere Spektralauflösung. Eine mögliche Überlappungsgefahr der Bänder wird dadurch vermindert und somit wird auch die Wahrscheinlichkeit des Auftretens von *Aliasing-Artefakten* vermindert.

2.5 Dateiformate und Codecs

Zur Abspeicherung von digitalen Audiosignalen auf entsprechenden Speichermedien gibt es mittlerweile sehr viele verschiedene Dateiformatierungsverfahren, wobei nachfolgend auf eine Auswahl der wichtigsten eingegangen wird. Der Schwerpunkt liegt hier auf dem bei MP3 verwendeten Verfahren, denn die Entwicklung dieses Verfahrens hat einen besonders nachhaltigen Einfluss auf die Welt der Endverbraucher-orientierten Multimediaanwendungen in Massenmärkten gehabt und findet daher bis heute auch die weiteste Verbreitung.

2.5.1 WAV

Das weit verbreitete *Wave Form Audio File* Format (Dateisuffix lautet in der Regel .wav) zur Abspeicherung von Audiodateien wurde gemeinsam von IBM und Microsoft entwickelt und heißt eigentlich RIFF – WAVE, denn es ist ein Bestandteil des RIFF – **R**esource **I**nterchange **F**ile **F**ormat - von Windows. Bei WAV handelt es sich um ein unkomprimiertes Dateiformat und die Daten werden häppchenweise in *Chunks* unterteilt und abgespeichert.

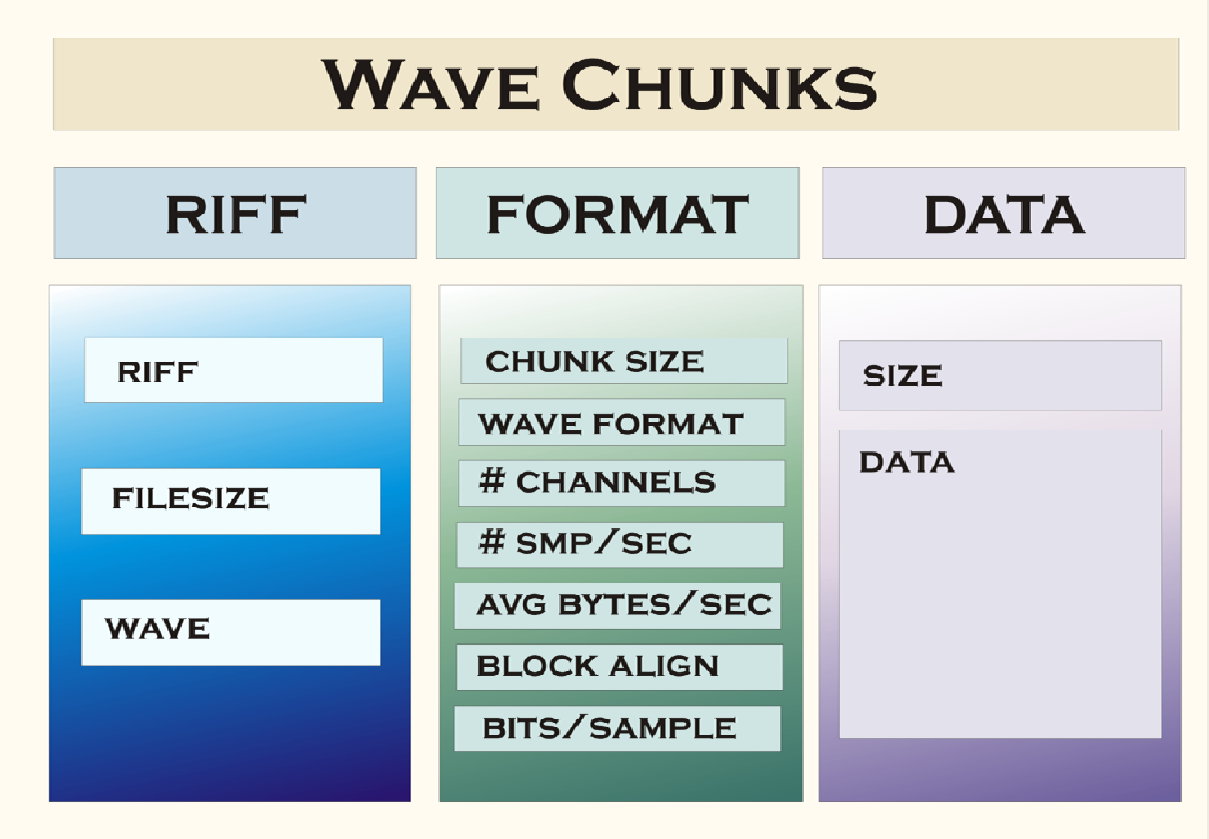


Abbildung 2.7: Dateikonzept von WAV

Im *Riff Chunk* werden die Audioattribute wie WAVE angegeben, wo hingegen im *Format Chunk* Aussagen über die genauen Parametereinstellungen zur Formatierung der vorliegenden wav-Datei gespeichert werden. Erst im *Datenchunk* befinden sich die eigentlichen Audiodaten. Diese drei Chunks stellen somit das notwendige strukturelle Grundgerüst zur Formatierung und Abspeicherung von wav-Dateien zur Verfügung. Es existieren jedoch noch weitere Chunk-Typen wie zum Beispiel die *Cue*- und *Playlist*-Chunks, die eingesetzt werden um die Abspielreihenfolge festzulegen.

2.5.2 MIDI

Das Akronym *MIDI* steht für den englischen Begriff ***M**usical **I**nstrument **D**igital **I**nterface*, welcher die ursprüngliche Idee, die hinter diesem ungewöhnlichen Audio-Datenformat steckt, sehr treffend umschreibt. Das MIDI-Protokoll wurde 1983 entwickelt [7], um die Kommunikation zwischen verschiedenen Synthesizern zu ermöglichen. Es werden Kontrollsignale zu der jeweiligen Hardware gesendet, die daraufhin den gewünschten Ton produziert. Das heißt, dass die Hardware die Musik erzeugt und lediglich die Steuersignale gespeichert werden müssen. Diese Technik sorgt für vergleichsweise kleine Dateien – allerdings ist die Qualität der Wiedergabe immer von der beim Empfänger installierten Hardware abhängig.
Soll diese Methode zum Abspeichern von Audiodateien genutzt werden, stellt sich ein weiterer Nachteil heraus: Das Abspeichern von Gesang ist nahezu unmöglich. Es kann bestenfalls die Melodie gespeichert werden. So kann beispielsweise der Song *Daniel* von Elton John im MIDI-Format in nur 20 KB abgespeichert werden [10], auf die Stimme von Elton John muss der Musikfreund beim Anhören dann jedoch verzichten. Allerdings wird dieser Nachteil durch nicht zu unterschätzende Vorteile wieder aufgewogen, von denen hier einer besonders hervorgehoben werden soll: Da die Töne als Steuersignale abgespeichert werden, können sie 1:1 als Noten wiedergegeben werden und der interessierte Musiker hat mit Programmen wie *Capella* [11] die Möglichkeit, sich die Notenblätter von Musikstücken, die im MIDI – Format gespeichert wurden, ausdrucken zu lassen (siehe Abbildung 2.8).

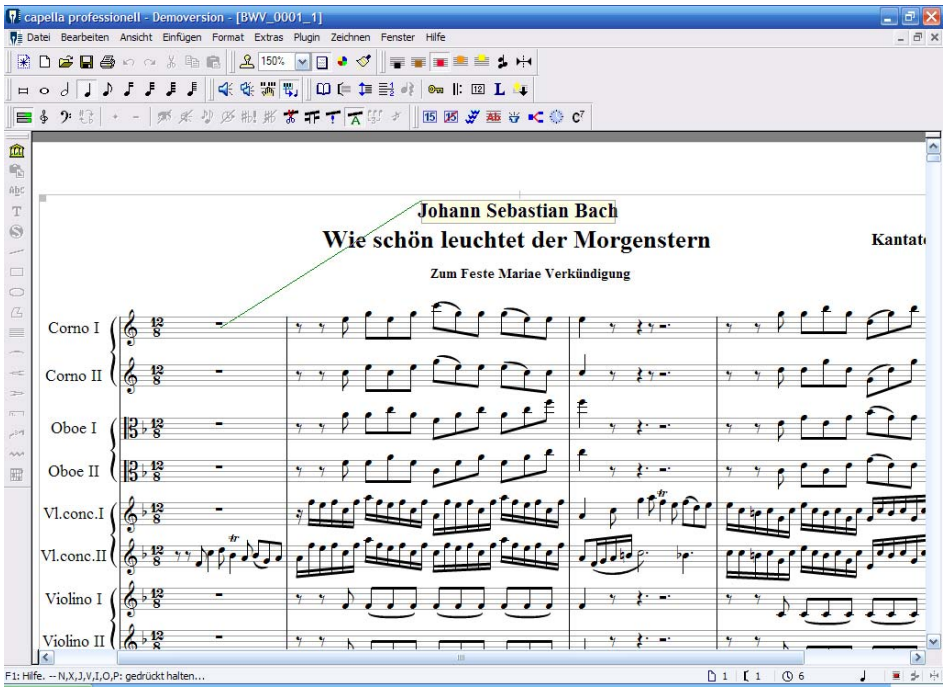


Abbildung 2.8: MIDI-Dateibearbeitung mit Capella

Wer weiterhin über ein MIDI-Keyboard oder ein anderes MIDI-fähiges Musikinstrument verfügt, kann dieses als Eingabeinstrument nutzen und sich die eingespielten Noten mit Programmen wie Capella direkt als Partitur anzeigen lassen. Zur Verbindung von MIDI-fähigen Geräten werden dabei zwei Kabel mit 5-poligen Steckern benötigt, die die beiden Geräte direkt MIDI-In beziehungsweise MIDI-Out miteinander verbinden. Alternativ dazu kann der Computer über die USB-Schnittstelle mit dem Keyboard verbunden werden. Im einschlägigen Fachhandel sind zum Beispiel USB-MIDI-Hubs und entsprechende Kabel erhältlich. Die ankommenden Steuersignale werden von der Hardware mittels *Wavetables* in möglichst realistische Klänge umgewandelt. Besitzer von Soundkarten ohne Wavetable können diese durch Installation von *Soundfonts* entsprechend ergänzen [12].
Eine Erweiterung des Standard-MIDI-Formates ist *General MIDI*. Hier sind die Kanäle festgelegt, die für die jeweiligen Instrumente stehen [13]. GM1 ist ein Standard, der 1991 von der Firma Roland entwickelt wurde und seither quasi als kleinster gemeinsamer Nenner im Sinne einer Standardisierung auch von unterschiedlichen anderen Firmen erkannt und genutzt worden ist. Allerdings haben unter diesen Herstellern sowohl *Roland* selbst als auch *Yamaha* noch zusätzliche Standards – GS und XS – entwickelt, was dazu führte, dass die unterschiedlichen Geräte nicht mehr einfach miteinander verkoppelt werden konnten. Seit 1998 haben sich Yamaha und Roland auf GM2 als gemeinsamen Standard geeinigt und so wird dieser Standard bis heute gemeinsam genutzt und weiterentwickelt.

Selbsttestaufgabe 2.3 Erläutern Sie kurz die Vor- und Nachteile von MIDI sowie dessen Funktionsprinzip.

2.5.3 MP3

Das MP3-Format wird zurzeit weltweit am häufigsten verwendet um Audiodateien verkleinert abzuspeichern. Eigentlich heißt es MPEG-1 Layer-3 Format, wobei das Akronym MPEG für ***M**otion **P**icture **E**xperts **G**roup* steht, denn dieses Format ist vollumfänglich eigentlich zur Komprimierung von Videodaten gedacht – es wird jedoch vermehrt auch der Audiocodec dieses Formates ausschließlich benutzt. Der Ansporn zur Entwicklung bestand darin, ein Audioformat zu entwickeln, welches die Versendung von hochwertigen, aber wertvolle Übertragungszeit sparenden Audiodateien über ISDN ermöglichte. Der technische Sprung sollte so eindrucksvoll sein, wie seinerzeit die Umstellung von schwarz/weiß auf Farbfernsehen. Als das MP3-Verfahren jedoch vor 15 Jahren entwickelt wurde, glaubte in Deutschland noch kaum jemand daran, dass es möglich sein könnte, einen Chip zu kreieren, der dermaßen komplizierte Vorgänge wie zum Beispiel das psychoakustische Modell beinhalten könnte. Auch wurde vermutet, dass die Kosten für eine etwaige Entwicklung immens hoch wären und so räumte die deutsche Wirtschaft dem Projekt keinerlei Erfolgschancen ein. Somit war Geburtsort des

ersten MP3-Players zwar Erlangen – denn hier entwickelten das Fraunhofer-IIS und der Halbleiterhersteller Intermetall 1994 das weltweit erste Modell –, jedoch der wirtschaftliche Siegeszug von MP3 begann zunächst außerhalb Deutschlands, wie auch aus dem Interview mit Prof. Gerhäuser und Prof. Brandenburg [14] hervorgeht.

Abbildung 2.9: Der erste MP3-Player, Quelle [15]

Der Prototyp des ersten MP3-Players wog etwa 2 Kilo und speicherte die Audiodaten mit einer Minute Spieldauer auf einem eingebauten EPROM ab. Dies war zwar noch nicht von großem praktischem Nutzen, stellte jedoch einen Durchbruch dar, denn nun war es endlich möglich, Musik ohne den Einsatz mechanischer Teile abzuspielen. Der von MP3 verwendete Koomprimierungsprozess wird durch das in Abbildung 2.10 dargestellte Kompressionsschema sehr gut verdeutlicht.

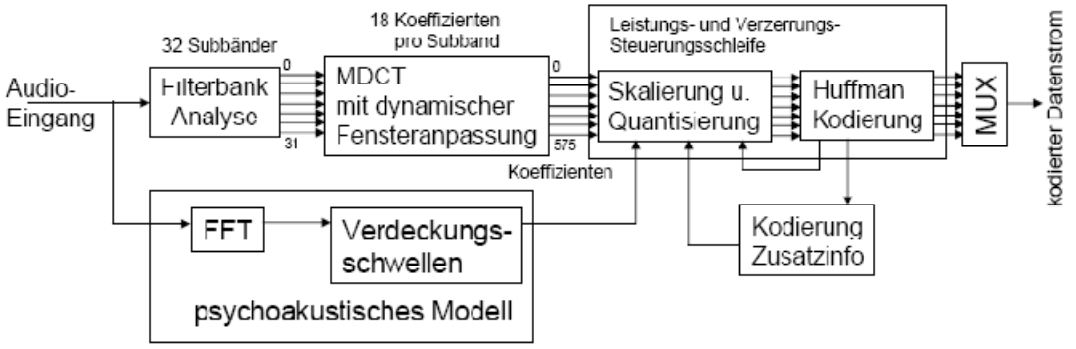


Abbildung 2.10: MP3-Komprimierungsschema, Quelle [7]

Wie aus dieser Abbildung ersichtlich wird, durchläuft der Audiodatenstrom einen relativ komplexen Bearbeitungsvorgang der Kodierung, der sich aber hinsichtlich der erzielbaren Kompressionsraten, also des erreichbaren Maßes an Verringerung des Speicherplatzbedarfes durchaus lohnt. Bei MP3 können durch die variablen Bitraten teilweise sogar enorme Kompressionsraten erreicht werden. So wird bei Telefonqualität mit 8 kBit/s eine Komprimierung von 96:1 erzielt. Für den Anspruch auf CD-Qualität wird die Audiodatei immerhin bei 128 kBit/s im Verhältnis 12:1 verkleinert. Weitere Details zum Aufbau des Kodierungsschemas eines *MP3-Encoders* lassen sich in [15] wie folgt nachlesen:

Filterbank
Das Eingangssignal wird in einer hybriden Polyphasen/MDCT-Filterbank in unterabgetastete Spektralkomponenten zerlegt. Die Filterbank und die entsprechende inverse Filterbank im Decoder

bilden zusammen ein Analyse-/Synthese-System.

Wahrnehmungsmodell
Mittels psychoakustischer Regeln wird eine Abschätzung der (zeit- und frequenzabhängigen) Maskierungsschwellen berechnet.

Quantisierung und Codierung
Die Spektralkomponenten werden quantisiert und codiert, wobei das dabei entstehende Quantisierungsrauschen, soweit möglich, unter der Maskierungsschwelle gehalten wird.

Erzeugung des Bitstroms
Ein Bitstrom-Formatierer setzt aus den quantisierten und codierten Spektralkoeffizienten und Seiteninformationen wie z. B. der Bit-Verteilung den MP3-Bitstrom zusammen.

Mono und Stereo
MP3 funktioniert sowohl mit Mono- als auch mit Stereo-Audiosignalen. Eine Technik namens „Joint Stereo Codierung" kann für die effiziente kombinierte Codierung des rechten und linken Kanals genutzt werden. Alternativ wird Mitten-/Seiten-Codierung oder Intensitäts-Stereocodierung eingesetzt. Die letztere Methode ist besonders bei niedrigen Bitraten nützlich, verändert aber eventuell das Klangbild.

Mehrkanal-Audio
Das herkömmliche MP3-Format ist in der Lage, Audiosignale mit einem oder zwei Kanälen zu kodieren. 2004 hat das Fraunhofer IIS eine rückwärtskompatible Erweiterung für Mehrkanalton im 5.1-Kanal-Format eingeführt, den sog. MP3 Surround.

Abtastraten
MP3 funktioniert mit verschiedenen Abtastraten. Bei MPEG-1 Layer III sind 32 kHz, 44,1 kHz und 48 kHz definiert. In MPEG-2 sind zusätzlich Abtastraten von 16 kHz, 22.05 kHz und 24 kHz zugelassen. „MPEG-2.5" ist der Name einer vom Fraunhofer IIS eingeführten Erweiterung für MP3, die schon bei sehr niedrigen Datenraten zufrieden stellend arbeitet und die zusätzlichen Abtastraten 8 kHz, 11,025 kHz und 12 kHz einführt.

Datenrate
Bei MP3 bleibt die Wahl der Datenrate – in bestimmten Grenzen – dem Programmierer oder dem Nutzer des MP3-Encoders überlassen. Der Standard definiert ein Set von Datenraten zwischen 8 kBit/s und 320 kBit/s. Außerdem muss der MP3-Decoder die Umschaltung von Datenraten zwischen einzelnen Datenblöcken unterstützen. Durch Verwendung der so genannten Bitsparkasse ist so die Codierung mit variablen und konstanten Datenraten bei jedem Wert innerhalb der Grenzen des Standards möglich.

Datenrate und Audioqualität
Im MP3-Encoder wird die verfügbare Datenrate so verteilt, dass das Quantisierungsrauschen möglichst nicht hörbar ist. Bei niedrigen Datenraten kann das Quantisierungsrauschen also hörbar werden. Die Audioqualität des kodierten Materials ist deshalb direkt von der Datenrate abhängig. Obwohl MP3 im Bereich von 8 kBit/s bis 320 kBit/s genutzt werden kann, ist zu empfehlen, Datenraten ab 80 kBit/s für Mono oder 160 kBit/s für Stereosignale zu verwenden. Für Anwendungen mit sehr niedrigen Datenraten sind die MPEG-4 Audiocodecs besser geeignet als MP3.

Natürlich muss die Kodierung auch wieder rückgängig gemacht werden können. Auch dieser Vorgang stellt einen nicht unerheblichen Rechenaufwand dar. Abbildung 2.11 illustriert daher nachfolgend das Schema zur Dekodierung von MP3-Formaten.

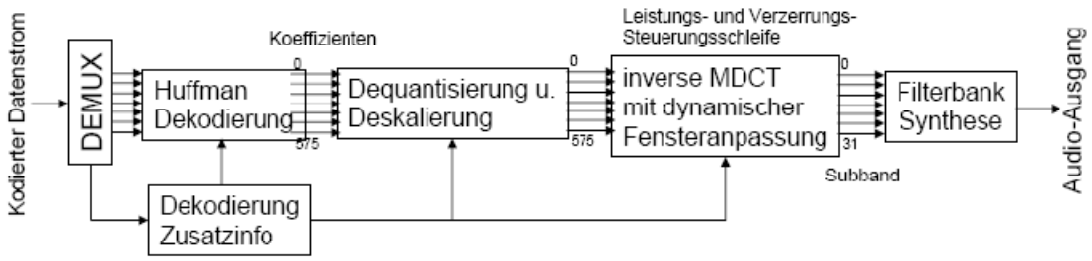


Abbildung 2.11: MP3-Dekodierungsschema, Quelle [7]

Zunächst durchläuft der kodierte Datenstrom den Huffman-Dekodierungsvorgang und wird dequantisiert. Nach der Umkehrung der MDCT erfolgt die Zusammenführung der unterabgetasteten Spektralkomponenten, die auch Subbänder genannt werden. Nach Abschluss der Filterbanksynthese wird der Audiodatenstrom an den Audio-Ausgang weitergegeben und die Audiodatei ist abspielbar. Als der verbesserte Nachfolger zum erfolgreichen MP3 gilt MPEG2 / 4 AAC [14].

Selbsttestaufgabe 2.4 Welchen der bisher genannten Codecs würden Sie für eine platzsparende Speicherung eines Liedes mit Gesang empfehlen? Begründen Sie Ihre Entscheidung.

2.5.4 Weitere Codecs im Überblick

Ogg

Der etwas merkwürdig anmutende Name entstammt einem Computerspiel namens *Netrek* und bedeutet soviel wie „Etwas furchtlos angehen, aber dabei die Zukunft nicht aus den Augen verlieren“. Bei Ogg-Vorbis und Ogg-Flac handelt es sich um lizenzfreie Audiocodecs, deren Entwicklung begann, als die FhG und Thomson 1998 begannen, Lizenzgebühren für MP3-Encoder zu verlangen. Das Format ist MP3 recht ähnlich und insbesondere bei niedrigen Bitraten sogar noch besser. Mittlerweile haben ogg-Dateien eine große Verbreitung gefunden und sind insbesondere wegen der Streaming Technologie interessant. Ferner gibt es von Ogg noch den Text-Codec Writ, den Sprachdaten-Codec Speex und den Video-Codec Theora [16].

RM

Das Realmedia-Format von der Firma Realnetworks hat sich insbesondere in der Audiostream-Technologie bewährt und eignet sich sowohl zur Verarbeitung von Live-Streams als auch für On Demand-Streams. Der Player zur Wiedergabe kann frei im Web unter *www.real.com* heruntergeladen werden und dementsprechend weit verbreitet ist das RM-Format. Es eignet sich auch dazu, zum Beispiel Radiosendungen live zu einem Audiostream zu enkodieren oder auch Vorlesungen inklusive Video als Stream über das Web zur Verfügung zu stellen. Für die Übertragung über das Internet wird die zu übertragende Datei in kleine Häppchen aufgeteilt, die on the fly komprimiert werden und jeweils einen eigenen Header erhalten. So kann der Anwender parallel zum Herunterladen der Datei diese schon konsumieren. Optimal ist, wenn die Bandbreite beim Empfänger höher ist als die beim Sender. Da RM ein Format ist, welches sich insbesondere für die schnelle Live-Übertragung im Internet eignet, ist die Qualität generell entsprechend niedrig gehalten. Zur Archivierung eignet es sich also weniger, wie eigentlich alle verlustbehafteten Formate. Je geringer die Bandbreite beim Empfänger, desto stärker wird die Audiodatei reduziert. Zur Komprimierung werden zunächst nur die Frequenzen entfernt, die für den Menschen unhörbar sind. Die Übertragungsqualität ist aber auch abhängig vom Inhalt der Datei. Soll es zum Beispiel um die Übertragung von Sprache gehen, können dementsprechend noch mehr irrelevante Frequenzen entfernt werden. Eine zu geringe Bandbreite sorgt jedoch für sehr schlecht zu verstehende Audiodateien und ist deshalb ein wichtiges Qualitätskriterium.

WMA/ASF

Bei dem **W**indows **M**edia **A**udio Codec handelt es sich um ein Containerformat, das von Microsoft entwickelt worden ist, eigens um Audiostreams im ***A**dvanced **S**treaming **F**ormat* über das Internet gut und schnell hörbar zu machen. Nach Angaben von Microsoft soll hier mit Bitraten, die kleiner oder gleich 64 kbps sind, CD-Qualität erreicht werden – was jedoch bei Hörproben derzeit subjektiv nicht so empfunden wird. Da sich wma-Dateien jedoch nicht so leicht kopieren lassen, meinte man zunächst, dass sie sich für die kommerzielle Verbreitung und Nutzung von Audioinhalten über das Web gut eigneten. Allerdings gibt es zum Beispiel eine relativ simple Methode über die Soundkarteneinstellungen auch diese Dateien zu kopieren und im WAVE-Format abzuspeichern. Da hierfür ein relativ hoher Zeitaufwand notwendig ist, haben findige Softwareentwickler aber auch schon für dieses Problem bereits 2004 eine Lösung entwickelt, z. B. in Form von Programmen wie *TuneBite*, welches u. a. WMA-Dateien in OGG umwandelt und abspeichern kann. Da also der eigentliche Algorithmus zum Kopierschutz relativ problemfrei ausgehebelt werden kann, ist die mitunter ziemlich schlechte Qualität dieser Audiodateien wohl das einzige Mittel, dass Audiophilisten von unerwünschten Downloads fernhält und somit eignet sich das Format zumindest dazu, Musikdateien zu veröffentlichen, die einen ersten Eindruck vom Künstler vermitteln und somit zum Kauf der Audio-CD animieren sollen.

Dolby

Die Firma Dolby hat mit den Audioformaten AC-1 bis AC-3 proprietäre Audiocodecs entwickelt, die nicht offen gelegt sind. AC-1 wurde seinerzeit im Hinblick auf HDTV entwicklt; AC-2 stellt eine verbesserte Variante im Vergleich zu AC-1 dar und AC-3 unterstützt das Mehrkanaltonverfahren für den *Surroundsound* mit Dolby 5.1. Die Ziffer 5 steht hierbei für die 3 Primärkanäle zusammen mit den beiden Surroundkanälen und die 1 für den niederfrequenten Basskanal. Dieses Format wird in den U.S.A für HDTV und DVD genutzt und ist deshalb nicht kompatibel zum europäischen HDTV, denn dies wird mit dem MP3-Verfahren kodiert. Weiterführendes über die Firma Dolby und die verschiedenen Formate kann im Internet unter *www.dolby.com* nachgelesen werden.

NeXT/Sun Audio File Format

Dieses Format wurde zeitgleich mit der Verbreitung den NeXT-Computern bekannt und ist eine Entwicklung von Sun Microsystems. Die übliche Dateiendung ist *.au*. Der Header einer AU-Datei ist in 4 Byte Blöcke unterteilt und im *big-endian* Format abgelegt. Daran schließt sich ein Textbereich an, der die eigentlichen Audioinformationen beinhaltet. Das au-Format unterstützt viele Audiformatierungsmethoden aber zumeist wird das so g. *µ-law logarithmic encoding* verwendet. Ein Überblick über die verschiedenen Erscheinungsweisen einer AU-Datei verschafft die folgende Tabelle:

32 bit word	Feld	Beschreibung/ Inhalt Hexadecimal Zahlen in C-Notation
0	magic number	Der Wert `0x2e736e64` (vier ASCII Zeichen „.snd“)
1	data offset	Offset der Daten in Bytes. Die kleinste gültige Zahl ist 24 (dezimal), da dies die Länge des Header ist (5 32-bit words) plus midestens noch 4 bytes für den *information chunk*.
2	data size	Dateigröße in Bytes. Falls unbekannt, sollte `0xffffffff` genutzt werden.

3	encoding	Datenkodierformat: • 1 = 8-bit G.711 • 2 = 8-bit linear PCM • 3 = 16-bit linear PCM • 4 = 24-bit linear PCM • 5 = 32-bit linear PCM • 6 = 32-bit IEEE Fließkommazahl • 7 = 64-bit Fließkommazahl • 8 = Fragmentierte Samples • 9 = DSP Programm • 10 = 8-bit Festkommazahl • 11 = 16-bit Festkommazahl • 12 = 24-bit Festkommazahl • 13 = 32-bit Festkommazahl • 18 = 16-bit linear mit Betonung • 19 = 16-bit linear komprimiert • 20 = 16-bit linear mit Betonung and komprimiert • 21 = „Music-kit“ DSP Kommandos • 23 = 4-bit ISDN μ-law komprimiert mit ITU-T G.721 ADPCM • 24 = ITU-T G.722 ADPCM • 25 = ITU-T G.723 3-bit ADPCM • 26 = ITU-T G.723 5-bit ADPCM • 27 = 8-bit G.711 A-law
4	sample rate	Die Anzahl der Samples/Sekunde (z.B.., 8000)
5	channels	Die Anzahl der ineinander verschränkten Kanäle (z. B.., 1 für mono, 2 für stereo, weitere Kanäle sind möglich – können aber vielleicht nicht vom Empfänger dekodiert werden)

Tabelle 2.1: Kodiertypen bei au-Dateien, Quelle Wikipedia (englische Version)

Aus der Tabelle wird ersichtlich, dass das AU-Format sowohl verlustfreie – wie PCM – als auch verlustbehaftete – wie ADPCM – Codecs unterstützt. Die Verkleinerung der Dateigröße anhand ADPCM beträgt in etwa 1:4. Das AU-Format ist auf Unix-Systemen sehr verbreitet.

2.6 Resümee

Audiokodierung wird auch in Zukunft ein nicht mehr wegzudenkender Bestandteil unserer bereits stark von einer Vielzahl multimedialer Anwendungen geprägten Alltagswelt sein. Sowohl im Internet aus auch in der Unterhaltungselektronik werden akzeptable Ergebnisse erst durch geschickt programmierte Komprimierungsalgorithmen erreicht. Doch sind diese mittlerweile nicht nur ein wichtiger Beitrag zur Muße und Unterhaltung im 21. Jahrhundert, sondern werden auch bereits bei der Spracherkennung und Prothetik eingesetzt. Durch Audiocodecs komprimierte Dateien sind heutzutage nicht nur klanglich einwandfrei, sondern eben auch klein genug um eine genügend schnelle Übertragungs- und Verarbeitungszeit zu gewährleisten. Dies ist die Grundlage für den Einsatz in medizinischen Bereichen und somit kann die Entwicklung von passenden Audiocodecs auch eine große Hilfe und Unterstützung für zum Beispiel Patienten mit Gehörschäden sein. Bei kleinen Endgeräten und geringer Bandbreite sind die verlustbehafteten Codecs zurzeit die erste Wahl. Wenn es jedoch darum geht Audiodateien persistent zu archivieren, dann sollten eher verlustfreie Formate benutzt werden.

Lösungen der Selbsttestaufgaben

Selbsttestaufgabe 1.1 Tabelle 1.8 gibt die Additionstabelle, Tabelle 1.9 gibt die Multiplikationstabelle für Hex-Ziffern an.

+	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Tabelle 2.8: Die Additionstabelle für Hex-Ziffern

x	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	6A	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Tabelle 2.9: Die Multiplikationstabelle für Hex-Ziffern

Selbsttestaufgabe 1.2 Die Dezimalzahl 100 ist 64 in Hexadezimalnotation. Der prozentuale Anteil der Kodepositionen von E000 bis F8FF am Koderaum von 0 bis FFFF ist also in Hexadezimalnotation (F8FF − E000 + 1) · 64/(FFFF + 1), also 1900 · 64/10000, also 9C4/100. Knapp 10 % der Unicode-Positionen im Bereich von 0 bis FFFF sind also für den privaten Gebrauch reserviert.

Selbsttestaufgabe 1.3 Die Unicode-Position F0000 wird unter UTF16 durch die beiden Surrogat-Wydes (F0000 − 10000) DIV 400 + D800 = DB80 und (F0000 − 10000) DIV 400 + DC00 = DC00 dargestellt.

Selbsttestaufgabe 1.4 Die Surrogat-Wydes DAFF und DEFF stellen die Unicode-Position (DAFF − D800) · 400 + DEFF − DC00 + 10000 = CFEFF dar.

Selbsttestaufgabe 1.5 Alle Zeichen der Zeichenfolge liegen im ASCII-Bereich von Unicode. Die Kodierung unter UTF8 und ASCII ist deshalb identisch; die Kodierung unter UTF16 ist um führende Nullen angereichert. Die Kodierung von „<?xml“ unter UTF8 ist 3C3F786D6C, die unter UTF16 ist 003C003F0078006D006C.

Selbsttestaufgabe 1.6 Die Positionen von 192 bis 255 werden unter UTF8 mit zwei Bytes kodiert, nach dem Schema **110xxxxx 10xxxxxx**. Da zur Kodierung der genannten Positionen nur acht Bits benötigt werden, die im Schema **110xxxxx 10xxxxxx** von rechts aufgefüllt werden, hat das führende Byte des Kodifikats die Form **110000xx**, wobei **xx** die führenden beiden Bits der Hexzahlen C bis F sind, also **11000011**, was C3 entspricht.

Selbsttestaufgabe 1.7 Das Zeichen „ü“ hat die Position FC (Dualzahldarstellung **11111100**). Die UTF8-Kodierung ist die Bitfolge **1100001110111100**, bestehend aus den Unicode- oder ISO Latin-1-

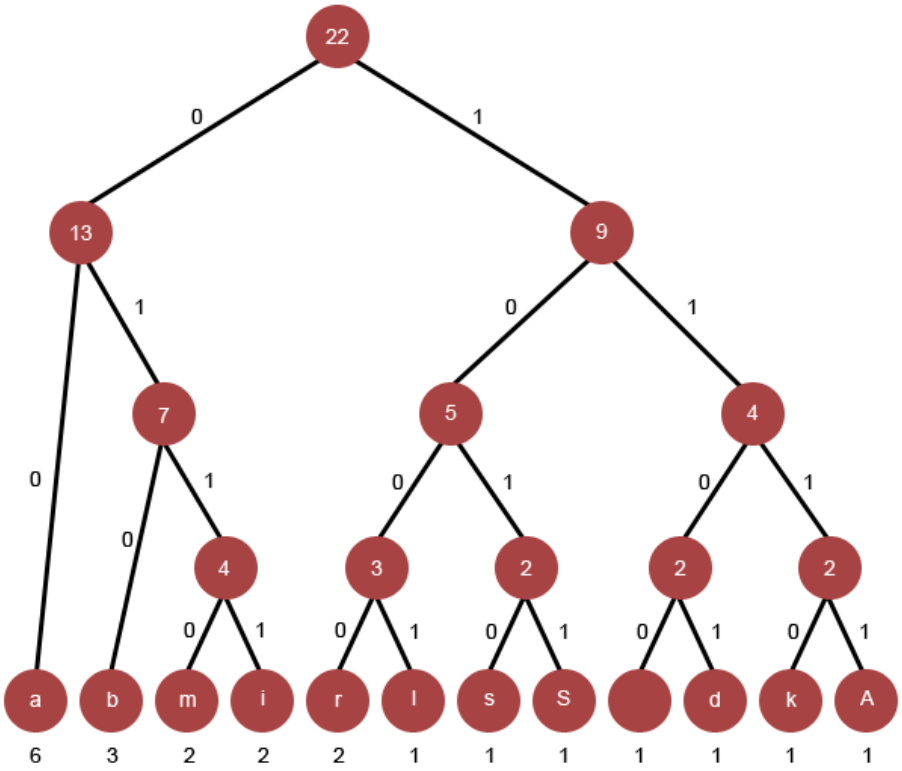
Positionen C3 für „Ã“ und BC für „¼“. Es ist also denkbar, dass das Textsystem einen UTF8-kodierten Zeichenstrom als ISO-Latin-1-kodiert interpretiert.

Selbsttestaufgabe 1.8 Unter UTF8 wird FEFF als Folge von drei Bytes dargestellt, nämlich EF, BB und BF.
Sowohl unter UTF16-BE als auch unter UCS2-BE wird BOM als dieselbe Folge von zwei Bytes dargestellt, nämlich FE und FF. Das BOM ist also kein hinreichendes Signal, um zwischen den bisher definierten Kodierungsschemata zu differenzieren.

Selbsttestaufgabe 1.9 native2ascii < test.ntv | native2ascii -reverse - encoding UTF8 > test.utf.

Selbsttestaufgabe 2.1 Die geringste Dateigröße wird durch die differenzielle PCM erreicht. Insbesondere bei der „Adaptiven Differenziellen PCM“ (ADPCM) wird bei nahezu gleicher Sprachqualität nur etwa die halbe Bitrate benötigt.

Selbsttestaufgabe 2.2



a	b	m	i	r	l	s	S		d	k	A
00	010	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Selbsttestaufgabe 2.3 Beim MIDI-Standard werden nur Steuerungsbefehle an einen Klangerzeuger (z. B. Soundkarte oder digitales Musikinstrument) übermittelt. Der Klang wird somit nicht direkt in der MIDI-Datei gespeichert. Der größte Vorteil von MIDI ist demnach die sehr geringe Dateigröße. Weiterhin ist es möglich, sich die Notenblätter von Musikstücken ausgeben zu lassen, die im MIDI-Format gespeichert sind. Neben der Unfähigkeit von MIDI Gesang zu speichern, ist ein weiterer Nachteil, dass viele Musikinstrumente (z. B. solche ohne Tastatur) nur unter großem Aufwand mittels MIDI ausgedrückt werden können.

Selbsttestaufgabe 2.4 MP3 ist das am besten geeignete Format. Mittels WAV lassen sich Audiodateien in sehr hoher Qualität abspeichern – allerdings auf Kosten eines großen Speicherbedarfs. MIDI erlaubt vergleichsweise kleine Dateien, lässt aber keine komplett originalgetreue Speicherung eines Musikstückes zu. MP3 arbeitet zwar nicht verlustfrei, beeinträchtigt die Audioqualität bei optimaler Kompression jedoch nur gering und verringert den Speicherbedarf gegenüber WAV um ein Vielfaches.

Literatur

[1] M. J. Dürst, F. Yergeau, R. Ishida, M. Wolf, A. Freytag, and T. Texin. Character model for the World Wide Web 1.0. http://www.w3.org/TR/charmod, February 2002. W3C Working Draft.

[2] J. Korpela. A tutorial on character code issues, January 1999. Looking for a new home on the Internet.

[3] K. Whistler and M. Davis. Character encoding model. Unicode Technical Report 17-3.1, Unicode, Inc., August 2000. http://www.unicode.org/unicode/reports/tr17/tr17-3.1/.

[4] P. A. Henning, Multimedia, Carl Hanser-Verlag, 2003

[5] J. Watkinson, Television Fundamentals, Elsevier, 1996

[6] D. Huffman, Minimum Redundancy Codes, IRE, Vol. 40 Sept. 1952

[7] H. Sack, C. Meinel, WWW xpert.press , Springer-Verlag, 2004

[8] R. Steinmetz, Multimedia-Technologie, Springer-Verlag, 2000

[9] M. Klein-Dasdamirov, Proseminar Audiokompression, TU-München, 2005

[10] Tech JD, http://www.ekn.net/midi/Elton-John/index.html, 2003

[11] E. Werner, http://www.whc.de/capella.cfm, 2007

[12] Personalcopy, http://www.personalcopy.com/sfarkfonts1.htm, 2003

[13] MMA, http://www.midi.org/, 2007

[14] W. Back, http://www.media01-live.de/CC-Zwei-35.mp3, WDR, 2007

[15] R. Ulrich, http://www.iis.fraunhofer.de/, 2007

[16] Xiph Open Source, http://www.vorbis.com/, 2005