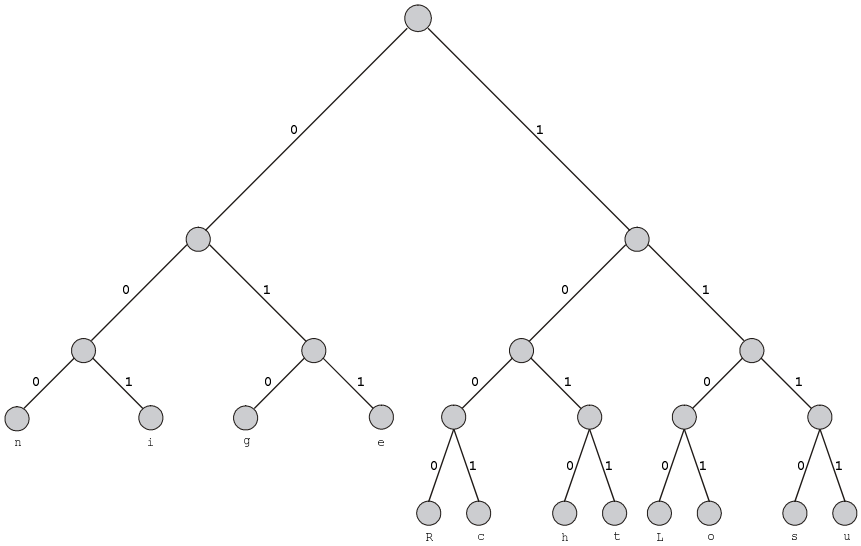
KOMPRIMIERUNGSVERFAHREN

Eine Unterrichtsplanung



Wolfgang Eiden

Kaiserslautern, Juni 1999

Inhaltsverzeichnis

1. Einleitung

Bedingt durch das Wachstum von Informationsnachfrage und -angebot werden effizientere Wege zur Repräsentation von Informationen aller Art benötigt. Dies kann sowohl durch Assimilation und Optimierung der gewählten Datenstruktur und ihrer Repräsentation als auch (additiv) durch Komprimierung derselbigen erreicht werden. Diese Ausarbeitung soll in pragmatischer Art und Weise in das Themengebiet der Komprimierung einführen. Vorgestellt werden insgesamt 3 Stellvertreter aus unterschiedlichen Bereichen :

- Komprimierung von Texten mittels Huffman-Code
- Komprimierung von Bitlisten mittels Laufkomprimierung (RLE-Komprimierung)
- Komprimierung von - auf dem RGB-Farbmodell basierenden - Grafiken mittels eines eigenen Verfahrens

Während die ersten beiden Verfahren Vertreter verlustfreier Komprimierung sind, ist das Dritte ein Vertreter der verlustbehafteten Komprimierung. Die vorgestellten Verfahren werden zur Arrondierung an konkreten Beispielen eingeübt und schließlich sogar in der Programmiersprache Pascal implementiert. Die konkrete Realisation in einer gegebenen Programmiersprache birgt kanonischerweise die Gefahr, den Blick für das Wesentliche zu verlieren. Deshalb wurde bei der Erstellung dieser Ausarbeitung (im speziellen der Programmieraufgaben) akribisch auf Abstrahierung unnötiger Details geachtet. Diese Unterrichtseinheit wurde ebenfalls so konzipiert, dass die Voraussetzungen zum erfolgreichen Behandeln dieses Themengebiets möglichst minimal sind, da dies - meiner Meinung nach - die Prädisponierung der Lerngruppe erleichtert.

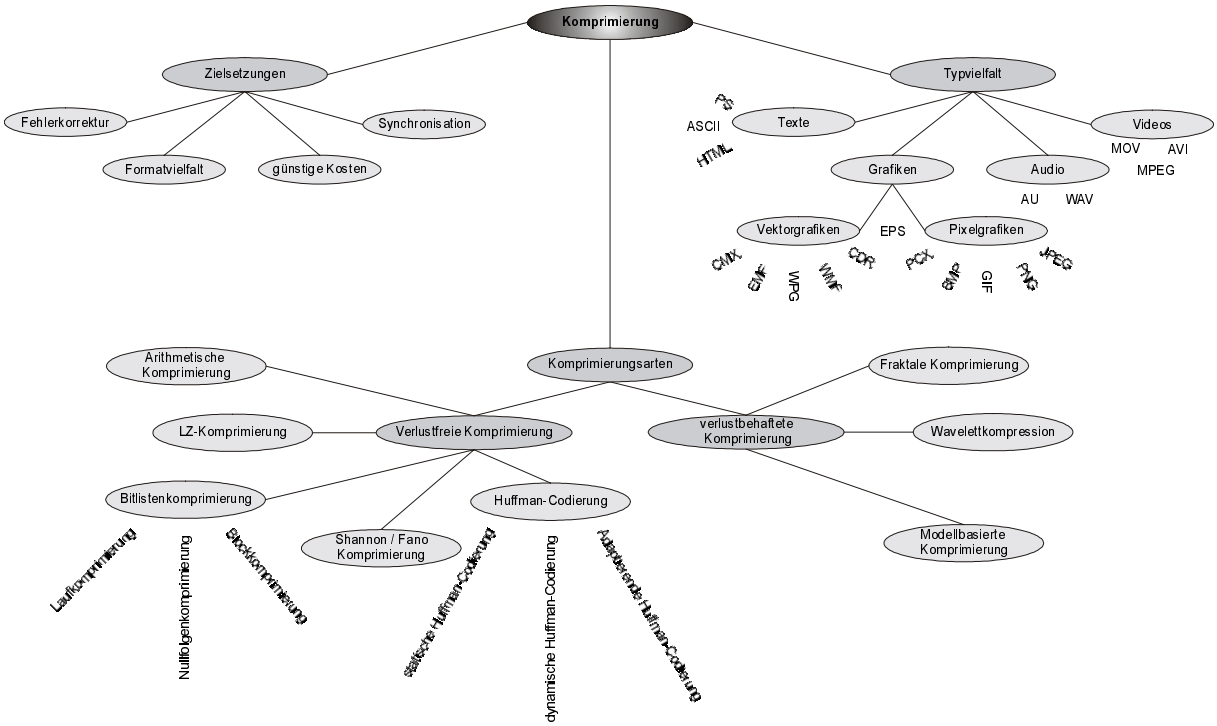
2. Fachliche Klärung des Unterrichtsgegenstandes

Das Ziel der Komprimierung ist es, eine möglichst effiziente Repräsentation gegegebener Informationen zu finden. Primär findet eine Unterscheidung zwischen verlustbehafteter und verlustfreier Komprimierung statt :

- Bei *verlustfreier Komprimierung* ist es möglich, die ursprünglichen Informationen aus den komprimierten Daten vollständig wiederherzustellen (Datenkompression).

- Bei *verlustbehafteter Komprimierung* erhält man bei der Rekonstruktion Informationen, die den ursprünglichen Informationen zwar ähnlich sind, aber nicht exakt mit diesen übereinstimmen (Datenreduktion).

Es stellt sich somit die Frage, welche Komprimierungsart bei einer - durch einen speziellen Datentyp, respektive einem Datenformat, gegebenen - Information angewendet werden soll. Beispielsweise sollten digital abgelegte Texte (insbesondere Metadaten) verlustfrei komprimiert werden, während normalerweise bei Bild- und Toninformationen eine Datenreduktion vertretbar scheint, da im allgemeinen bereits bei der Digitalisierung nicht alle Informationen erfaßt werden. Neben dieser Typvielfalt spielt ebenfalls die Realisation gewisser Zielsetzungen, wie beispielsweise unterstützte Formatvielfalt und günstige Kosten, eine bedeutende Rolle. Folgende Mindmap soll die Komplexität des Themengebietes verdeutlichen :



2.1. Verlustfreie Komprimierung (Datenkompression)

Vertreter dieser Komprimierungsart sind beispielsweise :

- **Huffman – Komprimierung :**
  Basierend auf der Idee des Morsealphabets wird hier die Wahrscheinlichkeit des Auftretens einzelner Buchstaben berücksichtigt: Häufig verwendeten Zeichen werden kürzere Codes zugeordnet als seltener verwendeten. Hierbei gibt es zur Festlegung der Codierung zunächst folgende Möglichkeiten :

  - *statische Huffman-Codierung* :
    Die Zeichenhäufigkeiten - und somit die Codierung - werden festgelegten Tabellen entnommen (z.B. Tabelle für deutschsprachige Texte) .

  - *dynamische Huffman-Codierung :*
    Durch vollständiges Lesen der konkreten Information werden die Zeichenhäufigkeiten ermittelt und im Anschluß der Huffman-Baum gemäß Bottom-Up-Methode aufgebaut.

  - *adaptierende Huffman-Codierung :*
    Die Zeichenhäufigkeiten werden analog zur dynamischen Huffman-Codierung ermittelt, wobei allerdings mit festen Vorgaben begonnen wird. Die adaptierende Huffman-Codierung vereinigt also statische und dynamische Huffman-Codierung und erzielt somit den Vorteil, die konkrete Information nur einmal vollständig lesen zu müssen.

  Die Rekonstruktion einer - nach diesem Verfahren - komprimierten Information erfordert allerdings die Verfügbarkeit des verwendeten Huffman-Baumes.

- **Shannon/Fano-Komprimierung :**
  Die Shannon/Fano-Komprimierung ist der dynamischen Huffman-Komprimierung sehr ähnlich, allerdings wird der verwendete Binärbaum nach der Top-Bottom-Methode generiert.

- **Bitlistenkomprimierung :**
  Ausgehend von der Binärdarstellung der jeweilen Informationen, wird die Reduzierung von Redundanzen erstrebt. Die bekanntesten Komprimierungstechniken hierbei sind :

  - *Laufkomprimierung (Run-Length-Komprimierung) :*
    Die unkomprimierte Bitliste wird in aneinanderhängende, alternierende Null- und Einsfolgen aufgeteilt, um anschließend die Länge eines jeden Laufes (Bitfolge gleichartig gesetzer Bits) in einer Codierfolge darstellen zu können.

- *Nullfolgenkomprimierung :*
  Die Bitliste wird durch aufeinanderfolgende Nullfolgen (Folge von 0-Bits zwischen zwei 1-Bits) dargestellt, wobei jeweils ein 1-Bit implizit ausgedrückt wird. Man unterscheidet primär zwischen Nullfolgenkomprimierung mit fester und variabler Länge der Codiereinheiten.

- *Blockkomprimierung :*
  Durch Reservierung von Kennbits können verschiedene Modi der Codierfolge fester Länge gekennzeichnet werden. So können beispielsweise mit zwei Kennbits Bitmuster übernommen und Null- respektive Einsfolgen codiert werden.

- **Arithmetische Komprimierung :**
  Ebenso wie beim Huffman-Code berücksichtigt diese Komprimierungsart die Wahrscheinlichkeit des Auftretens der einzelnen Buchstaben: Den einzelnen Buchstaben werden disjunkte Teile des Intervalls [0..1) zugewiesen, wobei die Länge des Teilintervalls der Auftrittswahrscheinlichkeit entspricht. Durch Zuordnung der binären Fließkommazahl, welche die untere Grenze des entsprechenden Teilintervalls angibt, werden die Buchstaben codiert. Mit Hilfe der Teilintervalle wird eine einzige Zahlenfolge bestimmt, die dann letztendlich den komprimierten Code repräsentiert.

- **LZ-Komprimierung :**
  Es handelt sich hierbei um Substitutionsalgorithmen, die aus den unkomprimierten Informationen Stringtabellen generieren und diese dann referenzieren. Dazu wird der unkomprimierte Text in Zeichenketten zerlegt und deren Existenz jeweils in der Stringtabelle geprüft, um ggf. bei Nichtexistenz die neue Zeichenkette in die Stringtabelle einzutragen. Anschließend wird dieser Eintrag dann referenziert.

2.2. Verlustbehaftete Komprimierung (Datenreduktion)

Durch diese Art der Komprimierung können Informationen in einem viel größeren Umfang komprimiert werden, als dies bei einer Komprimierung ohne Verlust möglich ist. Dies erscheint im allgemeinen allerdings nur dann sinnvoll, falls entweder ausschließlich eine Reduzierung (bzw. Beseitung) von Redundanzen oder eine Abstrahierung von vollständig irrelevanten Informationen stattfindet. Die Entscheidung, welche Teilinformationen in dem konkreten Zusammenhang redundant sind, ist sicherlich nicht trivial und hängt in hohem Maße von der Art der Information ab. So können beispielsweise bei Pixelgrafiken - zu einem gewissen Grad - ähnliche Farben zusammengefasst werden, ohne das der menschliche Betrachter davon Notiz nimmt. Um diesen Effekt zu erreichen, basieren viele Verfahren der Datenreduktion auf Erkenntnissen diverser anderer Wissenschaftsgebiete wie beispielsweise

Mathematik, Physik, Biologie und Musik. Aus diesem Grunde - und nicht zuletzt wegen des hohen Komplexitätsgrades - führt eine detailierte Beschreibung der folgenden Verfahren an dieser Stelle zu weit (hierzu sei die entsprechende Fachliteratur empfohlen) :

- **Fraktale Kompression :**
  Dieses Verfahren versucht Selbstähnlichkeiten (Natureffekt) zu finden, um somit eine Komprimierung zu erreichen.

- **Wavelett-Kompression :**
  Durch Nutzung verschiedener Wellenformen (beispielsweise Sinus- und Cosinuswellen) können die umkomprimierten Bild- oder Toninformationen frequenzgemäß zerlegt werden. Es zeigt sich, dass die wesentlichen Bildinformationen im niederfrequenten Bereich angesiedelt sind.

- **Modellbasierte Kompression :**
  Die modellbasierte Komprimierung kombiniert Ansätze aus Bildverarbeitung und -verstehen (im speziellen: OCR / Texterkennung)

- **Kompression durch Blockbildung und Filterung :**
  Innerhalb dieser Unterrichtsreihe wird ein verlustbehaftetes Verfahren zur Komprimierung von - auf dem RGB-Farbmodell basierenden - Grafiken vorgestellt:

  - *Blockbildung :*
    Zunächst wird eine Einteilung des Bildes in Blöcke gleicher Größe vorgenommen. Das komprimierte Bild enthält dann nur die gemittelten Rot-, Grün- und Blauanteile der einzelnen Blöcke.

  - *Filterung :*
    Alle darstellbaren Farben eines Monitors können eindeutig über ihren Anteil an Rot, Grün und Blau definiert werden. Diese drei Farbkomponenten bilden die Basis des RGB-Farbmodells und können jeweils einen Wert von 0 bis 255 annehmen. Je höher dabei der RGB-Wert ist, desto heller ist dabei die dargestellte Farbe. Daher ist das RGB-Modell ein additives Farbmodell, da die Farben durch Hinzufügen von Licht entstehen. Durch Filterung eines Farbanteils (also durch dessen Nullsetzung) kann somit sein Einfluss aus dem Bild genommen werden.

3. Didaktische Analyse

3.1. Begründung des Unterrichtsvorhabens

Gerade in einer von Globalisierungstendenzen geprägten Zeit wie heute - in der sowohl das Angebot als auch die Nachfrage an Informationen exponentiell wächst - gewinnt die Komprimierung von Daten jeglicher Art an zunehmender Bedeutung. Die Minimierung von Übertragungszeit, Speicherplatzbedarf und der damit verbundenen Kosten sind folglich zu lösende Herausforderungen. Bei der Betrachtung von aktuellen Forschungsgebieten wird sowohl die enorme Wichtigkeit von effizienter und kostengünstiger Datenübertragung als auch die fachliche und gesellschaftliche Relevanz deutlich:

- Zur Zeit entwickelt die *NASA* einen Service-Roboter, der (gekoppelt an einen Satelliten) Wartungs- und Reparaturarbeiten an anderen Satelliten durchführen soll. Sowohl die Statusinformationen als auch die aufgenommenen Videobilder werden von diesem Roboter an die Bodenstation gesendet. Als Format für die Videobilder wird aufgrund des hervorragenden Komprimierungsverhältnisses MPEG verwendet (Quelle: „Computer-Club“, Westdeutscher Rundfunk).

- Eine amerikanische Firma entwickelt zur Zeit ein Verfahren um digitale Videofilme via Satellit an Kinos auszustrahlen, welche die Filme nur für die Dauer der Aufführung lokal zwischenspeichern.

- Seit ein paar Jahren ist man in Deutschland mit der Entwicklung eines Medizin-Netzes beschäftigt. Neben relativ einfachen Zielsetzungen, wie der Übermittlung von Röntgenbildern, sind ebenfalls Ferndiagnose und -operationsunterstützung im Bereich des Erstrebten.

- Aufgrund der hohen Applizierbarkeit (beispielsweise bei Datenübertragung, Datenspeicherung und Datenrepräsentation) ist die Komprimierung ein zentrales Forschungsgebiet der Informatik.

Ebenso können exemplarisch fundamentale Ideen der Informatik (vgl. Schwill, 1993) aus den Bereichen der Algorithmisierung und strukturierten Zerlegung vorgeführt werden :

- Verifikations-Evaluation (z.B. Terminierung, Vollständigkeit und Konsistenz).

- Komplexitäts-Evaluation (z.B. Reduktion).

- Modularisierung (z.B. Black-Box-Prinzip).

- Programmierkonzepte (z.B. Iteration).

- Hierarchisierung (z.B. Darstellungsformen: Baum, Einrückung).

Desweiteren genügt die Themenwahl folgenden Punkten:

- Die Problemstellung ist hinreichend interessant und gibt somit den Reiz zur Entwicklung eigenständiger Lösungen.

- Das Problem ist modifizierbar und erweiterbar, bietet also noch genügend Spielraum für weitere Betrachtungen.

- Die Transfermöglichkeit der Lösung auf andere Probleme und die Bereitstellung einer Lösungsstrategie ist gegeben.

- Die Realitätsnähe des Problems ermöglicht eine Diskussion gesellschaftlicher Aspkete und fördert somit auch die Interaktion und Kooperation zwischen den Schülerinnen und den Schülern.

3.2. Verankerung des Themas im Lehrplan und fachspezifisch allgemeine Lernziele

Zunächst liegt im Lehrplanentwurf Informatik [L93] keine explizite Aussage über die Behandlung dieses Themengebietes vor. Da es sich jedoch bei Komprimierungsverfahren - wie bereits in der Begründung des Unterrichtsvorhabens angeführt - um ein aktuelles Thema handelt, ist eine Einbettung in den Komplex „Aktuelle Entwicklungen und Gesellschaft“ möglich. „Dieser Teil des Lehrplans bietet in besonderem Maße Freiraum zur Gestaltung eines Unterrichts, der aktuelle Themen der Informatik und ihre Wechselwirkungen mit Gesellschaft und Umwelt aufgreifen soll.“ [L93, S.32]. Auch die im Lehrplanentwurf geforderte Thematisierung der Beziehung zwischen Gesellschaft, Individuum und Informationstechnik [L93, S.16] kann hier erfolgen. Durch das Thema Komprimierungsverfahren werden desweiteren folgende fachspezifisch allgemeine Lernziele des Lehrplanentwurfs Informatik [L93] abgedeckt :

- Kenntnis von Prinzipien und Methoden der Fachwissenschaft Informatik [...].

- Fähigkeit zur Problemlösung durch Anwendung von Methoden, Werkzeugen und Standardverfahren der Informatik.

- Kenntnis wichtiger Bewertungskritierien [...].

- Fähigkeit zur [...] Beurteilung von Problemlösungen, Methoden, Werkzeugen und Standardverfahren der Informatik.

- Bewußtsein, dass eine Problemlösung in verschieden Sprachebenen (Umgangssprache, Programmiersprache, [...]) und Beschreibungsformen (Text, Graph, [...]) dargestellt werden kann.

- Einsicht, daß die umfassende Anwendung der Informations- und Kommunikationstechnik vielschichtige Auswirkungen auf unser Leben hat [...].

- Die Schülerinnen und Schüler sollen die Informatik als Wissenschaft begreifen, die sich in ihrem Kern mit allgemeinen Methoden zur Problemlösung, ihrer sprachlichen Präzisierung und ihrer technischen Realisierung beschäftigt.

Darüber hinaus steht dieses Themengebiet in enger Verwandschaft mit folgenden Lehrplaninhalten :

- Multimediale Repräsentation von Informationen.

- Gegenüberstellung von Kosten und Nutzen verschiedener Repräsentationsformen.

- Interpretationsproblemtatik, Informationsverluste, Informationsverfälschungen.

- Gewinnung eines Einblicks in die Entwicklung der elektronischen Datenverarbeitung.

- Darstellung der Chancen und Risiken der Informationstechniken.

Ferner wird auch dem - nach der Meinung von Peter Hubwieser und Manfred Broy - primären Ziel des Informatikunterrichts Rechnung getragen, welches den Schülerinnen und Schülern wichtige Fähigkeiten im Umgang mit Informationen vermitteln und das zentrale Paradigma der Informationsverarbeitung näherbringen soll (vgl. [HBR]): „Diffuse Informationen werden durch formale Repräsentation konkretisiert und dadurch maschinell verarbeitbar. Durch Informationsverarbeitung wird die Repräsentation umgeformt. Die dadurch entstandene neue Repräsentation muß schließlich wiederum interpretiert werden, um menschlich verwertbare Informationen zu liefern.“

3.3. Auswahl und Strukturierung der Inhalte und Ziele

Bedingt durch die essentielle Unterscheidung zwischen verlustfreier und verlustbehafteter Komprimierung scheint mir die Vorstellung von Verfahren beider Arten angemessen. Aus mehren Gründen bietet sich die *Huffman-Komprimierung* als Einstieg an:

- Im allgemeinen sind verlustfreie Komprimierungsverfahren intuitiv leichter erfaßbar, da keine Entscheidung über etwaige Redundanzen stattfinden muss.

- Die Komprimierung von (ASCII-) Texten - mit einem greifbaren Informationsgehalt ist sicherlich einfacher als die Komprimierung von Bitlisten oder gar von Grafikdateien unterschiedlichster Art.
- Das Grundprinzip der Substitution von häufig vorkommenden Grundmustern wird hier besonders deutlich.
- Zur Einführung werden keine besonderen Hilfsmittel benötigt.

Unter Berücksichtigung der Codierungsmöglichkeiten unterscheidet man hier – wie bereits erwähnt und genauer ausgeführt – zwischen statischer, dynamischer und adaptierender Huffman-Codierung. Aus folgenden Gründen ist hierbei die dynamische Huffman-Codierung als Einstiegsbeispiel prädestiniert :

- Da bei der adaptierenden Huffman-Codierung die Generierung des Huffman-Baumes durch dynamisch gewichtete Binärbäume realisiert wird, erscheint mir diese Form als Einstiegsbeispiel völlig ungeeignet. Darüber hinaus bin ich der Meinung, dass eine Behandlung dieser speziellen Form höchstens in einem Leistungskurs Informatik erfolgen könnte, aber auch dort auf keinen Fall als Einstiegsbeispiel herangezogen werden sollte.
- Der einzige Unterschied zwischen statischer und dynamischer Huffman-Codierung ist, dass bei der dynamischen Variante erst die Häufigkeiten des Auftretens der einzelnen Buchstaben ermittelt werden muss. Da die Grundidee einer optimalen Substitutionskomprimierung auf den tatsächlichen Häufigkeiten der einzelnen Muster basiert, scheint mir die dynamische Form als Einstieg angebracht. Nach erfolgreicher Einführung dieser Form ist die Behandlung der statischen Huffman-Codierung trivial.

Im Anschluß kann die *LZ-Komprimierung* im Sinne einer Erweiterung der Huffman-Codierung angesprochen werden. Unter Berücksichtigung des eng begrenzten Zeitrahmens halte ich eine kurze Erwähnung jedoch für völlig ausreichend, zumal die Idee der Mustererweiterung auf komplette Strings recht einsichtlich erscheint.

Als weiteres (und letztes Beispiel) für eine verlustfreie Komprimierung bieten sich - aufgrund der vom Huffman-Algorithmus gelieferten Binärdarstellung komprimierter Texte - Bitlisten-komprimierungsverfahren an. Darüberhinaus spielen Bitlisten in vielen Gebieten der Informatik eine zentrale Bedeutung: Beispielsweise lassen sie sich als Verweislisten und als Mittel zur Darstellung von dünn besetzen Matrizen und Bildinhalten wirkungsvoll einsetzen. (vgl. [HÄR]). Die bekanntesten Komprimierungstechniken hierbei sind Laufkomprimierung, Nullfolgenkomprimierung und Blockkomprimierung. Als Repräsentant dieser Komprimierungsart ist die *Laufkomprimierung* aufgrund ihres einfachen Aufbaus am besten geeignet.

Als Arrondierung sollte - wie bereits oben erwähnt - die Behandlung eines verlustbehafteten Komprimierungsverfahrens stattfinden. Die ersten drei vorgestellten Verfahren (fraktale Komprimierung, Wavelettkompression und modellbasierten Komprimierung) sind für die Behandlung innerhalb eines Schulunterrichts zu komplex. Dies wird allein dadurch deutlich,

dass diese Gegenstand intensivster Forschungsarbeit sind. Aus diesem Grunde habe ich mich für das Kompressionsverfahren „Kompression durch Blockbildung und Filterung“ entschieden, da hier keine größeren Schwierigkeiten beim Verständnis zu erwarten sind.

3.4. Fixierung der Lernziele

Folgende Lernziele sollten erreicht werden :

- **Groblernziele :**
  Die Schülerinnen und Schüler sollten

  - die Idee der Informationskomprimierung verstehen,
  - wissen wann eine Komprimierung sinnvoll ist,
  - den Unterschied zwischen verlustfreien und verlustbehafteten Komprimierungsverfahren kennen,
  - verstehen, dass für unterschiedliche Datentypen und -formate jeweils verschiedene Verfahren benötigt werden, um die Information optimal zu komprimieren.

- **Feinlernziele :**
  Die Schülerinnen und Schüler sollten

  - die Zielsetzung der Komprimierung (das Finden einer möglichst effizienten Repräsentation einer gegegeben Information) kennen,
  - wissen, dass sowohl die Datenübertragung als auch der Speicherplatz im Vergleich zur Rechenleistung relativ teuer sind und dass Speichermedien in ihrer Kapazität beschränkt sind,
  - zur Einsicht gelangen, dass komprimierte Daten schneller übertragen werden können,
  - die Anwendungsgebiete der vorgestellten Verfahren kennen,
  - wissen, bei welchen Daten nur verlustfreie Komprimierungsverfahren sinnvoll sind (z.B. bei Metadaten),
  - die dynamische und statische Huffman-Komprimierung verstehen und anwenden können,

- die Idee der Laufkomprimierung verstanden haben,

- wissen, dass durch ungeignete Wahl eines Komprimierungsverfahrens respektive seiner Parameter unerwünschte Nebeneffekte auftreten können (beispielsweise kann bei der Bitlistenkomprimierung durch eine ungeignete Wahl der Codierungslänge sogar eine Vergrößerung der Informationsrepräsentation erfolgen),

- die Problematik von verlustbehafteten Komprimierungsverfahren kennen (z.B. Gefahr des zu großen Informationsverlustes, Schwierigkeit in der Festlegung von Redundanz-Kriterien),

- das vorgestellte Komprimierungsverfahren für Bilder verstanden haben,

- wissen, dass nicht alle Informationsverluste vom menschlichen Betrachter wahrgenommen werden (z.B. durch unser beschränktes Wahrnehmungsvermögen) und dass somit gewisse Informationsverluste vertretbar sind.

3.5. Voraussetzungen und Vorkenntnisse

Aufgrund der Struktur des Lehrplans Informatik [LE93] kann - bei Behandlung dieses Themengebietes in der Klassenstufe 12/2 - davon ausgegangen werden, dass die Schülerinnen und Schüler Elemente der Programmierung kennengelernt haben und selbständig kleinere Programme entwickeln können. In der hiesigen Unterrichtsreihe wird vom Können der imperativen Programmiersprache Pascal ausgegangen, da sich an Schulen die imperativen „Programmiersprachen wie z.B. Pascal, als Ausbildungssprache zur Einübung der Grundelemente algorithmischer Problemlösungen bewährt“ haben (vgl. LE93, S15). Die Implementierung des vorgestellten Bildkomprimierungsverfahrens ist zwar als objektorientiertes Pascal-Programm für das Betriebssystem Windows realisiert, allerdings ist (begründet durch das Black-Box-Prinzip) keine Kenntnis im Bereich der objektorientierten Programmierung notwendig.
Ansonsten sollten die Schülerinnern und Schüler mit der Binärdarstellung von Informationen und den mathematischen Operationen der Ganzzahldivision und Modulo-Bildung vertraut sein. Falls dies nicht der Fall sein sollte, so kann dies gegebenfalls an dieser Stelle ohne größeren Aufwand nachgeholt werden.

3.6. Lernkontexte

Als Lernkontexte bietet sich sowohl die Komprimierung von Texten als auch von Grafiken an, da dies den folgenden Anforderung genügt :

- Das Thema ist hinreichend komplex, so dass für unterschiedliche Ausgangsformate die Lösung nicht unmittelbar auf der Hand liegt.

- Allein bedingt durch den Lehrplan der Klassenstufe 11 und 12 hat jede Schülerin und jeder Schüler praktische Erfahrung mit Computern sammeln können. Dadurch ist das Thema sowohl interessant als auch aus dem Erfahrungsbereich der Schülerinnen und Schüler.

- Das benötigte Vorwissen ist minimal.

Die speziell auf Tondatenkomprimierung ausgerichteten Verfahren erfordern ein hohes Maß an Detailwissen der verschiedensten Bereiche, so dass die Behandlung dieser Komprimierungsart in diesem Zusammenhang ausscheidet. Daher ist auch zwangsläufig die Komprimierung von Videodaten (nicht zuletzt aufgrund der Synchronisationsproblematik bei audiovisuellen Daten) zu komplex. Als Motivierung dieser Unterrichtseinheit könnte den Schülerinnen und den Schülern die Aufgabe gestellt werden, eine – auf der Festplatte gespeicherte – Postscript-Datei (in ASCII-Formatierung und einer Größe über 1.44 MB) auf einer 1.44 MB - Diskette zu speichern. Ohne Komprimierung dürfte dies wohl kaum möglich sein.

3.7. Lehrbuch

Für diese Unterrichtseinheit wird kein spezielles Lehrbuch benötigt, da alle Materialien zur Verfügung gestellt werden. Darüber hinaus erscheint mir die spezielle Neuanschaffung eines Lehrbuches - sowohl in Hinblick des eng begrenzten Zeitrahmens als auch aus altruistischen Aspekten - nicht sonderlich sinnvoll. Sollte allerdings ein dieses Themengebiet umfassendes Lehrbuch vorhanden sein, so stellt dieses sicherlich eine sinnvolle Ergänzung und Vertiefungsmöglichkeit dar.

4. Methodische Bemerkungen

4.1. Lern- und Unterrichtsmethodische Aspekte

Zur Einübung, Vertiefung und Arrondierung der einzelnen Verfahren befinden sich im Anhang Arbeitsblätter. Während die meisten Aufgaben zur Einübung bzw. Vertiefung und Vorbereitung des Stoffes dienen, beschäftigen sich andere mit der Implementierung des jeweiligen Verfahrens. Als Programmiersprache wurde hierbei Borland Pascal 7.0 für Windows verwendet. Durch kleinere Änderungen (die in Quelltexten angegeben sind) können die (ersten beiden) Programme allerdings auch unter Turbo Pascal 6.0 kompiliert werden. Um nicht den Blick für das Wesentliche zu verlieren, werden in ausgegliederten Programmteilen (realisiert durch Units) alle Routinen und Datenstrukturen zur Verfügung gestellt die der Übende benötigt. Ebenso läßt sich so der unnötige Verlust von wertvoller Zeit und die profuse Beschäftigung mit den Interna der (letztendlichen) Realisation vermeiden. Es sollte also nicht zuviel Gewicht auf die Implementierung gelegt werden. Vielmehr halte ich die gewählte Sozialform der Gruppenarbeit am Computer für sinnvoll, da durch den Gedankenaustausch zwischen den einzelnen Übungspartnern ein größerer Lernerfolg erzielt werden dürfte.

Wie bereits früher begründet, finde ich den Einstieg mit einem verlustfreien Komprimierungsverfahren am sinnvollsten. Ausgehend von der - vom Huffman-Algorithmus gelieferten - Binärdarstellung komprimierter Textpassagen bietet sich der Übergang zu Bitlistenkomprimierungsverfahren an, wobei hier die RLE-Komprimierung das wohl einfachste Verfahren darstellt. Die Routinen der Unit „Bitliste“ sind so allgemeingültig geschrieben, dass auch andere Verfahren zur Komprimierung von Bitlisten leicht realisiert werden könnten und diese Unterrichtseinheit somit sehr weit gestreckt werden könnte.

Da aber auch die verlustbehafteten Komprimierungsverfahren von wesentlicher Bedeutung sind, sollte zumindest ein Repräsentant im Anschluß behandelt werden. Es wird hierbei allerdings keine explizite Trennung zwischen synthetischem und analytischem Zugang vollzogen, da durch deren Mischung wesentlich größere Vorteile entstehen: Zunächst erarbeiten und lernen die Schülerinnen und Schüler ein Verfahren. Aufgrund der Implementierung ist es möglich, die Verfahren analytischen Tests zu unterziehen um somit deren Vor- und Nachteile bzw. deren Grenzen praktisch zu erleben und zu diskutieren. So kann beispielsweise bei der RLE-Komprimierung durch eine geeignete Wahl der Codiereinheitslänge den Schülerinnen und Schülern auch ein unerwünschter Nebeneffekt von Komprimierungsverfahren demonstriert werden : Je nach gewählten Parametern und Eingaben kann die „komprimierte“ Datei sogar mehr Speicherplatz beanspruchen als die zu komprimierende Datei (siehe Folie „Vergleich der Dateigrößen bei RLE-Komprimierung“).

Nähere Begründungen zur gewählten Strategie finden Sie bei den Erläuterungen zur Auswahl und Strukturierung der Inhalte und Ziele (Paragraph 3.3).

5. Unterrichtsplanung

5.1. Anlage des Unterrichts

Nach der Motivation für dieses Thema - bei der die Folien „*Das Verfahren der Komprimierung, Vorteile der Komprimierung*“ und „*Anwendungsbeispiele*“ unterstützen sollen - können verschiedene Lösungsansätze zur Komprimierung von „reinen“ ASCII-Texten diskutiert werden. Dabei dürfte relativ schnell die Idee nach einem Code entstehen, bei dem gilt:

> Häufig vorkommende Buchstaben haben einen kürzen Code als Buchstaben, die eher selten auftreten.

Dadurch können die Schülerinnen und Schüler für die dynamische Huffman-Codierung prädisponiert werden. Zunächst sollte die Huffman-Codierung an einem konkreten Beispiel demonstriert werden (Folien: „*Huffman-Algorithmus – Beispiel (...)*“) und anschließend die einzelnen Schritte des Verfahrens festgehalten werden (Folien: „*Huffman-Algorithmus (...)*“). Nun sollten die Schülerinnen und Schüler in Einzelarbeit sowohl die Huffman-Komprimierung, als auch die -Dekomprimierung an geeigneten Beispielen üben und Effizienzkriterien finden (*Arbeitsblatt 1*). Es ist wichtig, dass die Schüler die Eindeutigkeit der Rekonstruierbarkeit erkennen, da diese Form der Codierung sonst nicht sinnvoll erscheint (*Aufgabe 1.2.2*). Im Anschluß wird das Verfahren zur weiteren Vertiefung implementiert, wobei den Schülerinnen und Schülern Hilfestellungen und maßgeschneiderte Routinen zur Verfügung gestellt werden (*Arbeitsblatt 2 mit Lösungshinweisen*). Im Anschluß kann die Frage nach der Erweiterbarkeit der Huffman-Komprimierung (*Aufgabe 2.2)* erörtert werden. In diesem Zusammenhang kann das Prinzip der LZ-Komprimierung (Zulassung von kompletten Zeichenketten als Muster) kurz erläutert werden.

Durch Diskussion über zusätzliche Komprimierungsmöglichkeiten (*Aufgabe 2.3*) kann leicht zur Bitlistenkomprimierung - im speziellen der Laufkomprimierung – übergegangen werden. Nach der Vorstellung der Idee kann die Laufkomprimierung an einem konkreten Beispiel vorgeführt werden (Folie „*Laufkomprimierung (RLE-Komprimierung)*“) und danach verallgemeinert werden (Folie „*Laufkomprimierung (2)*“). Anschließend kann die Dekomprimierung erläutert werden. Die Bearbeitung der *Aufgabe 3.1* dient zur Sicherung und Vertiefung des bisher Gelernten. Da den Schülerinnen und Schüler im Alltag die Komprimierung von Bitlisten wohl kaum in ihrer reinen Form begegnen wird, ist die konkrete Implementierung der RLE-Komprimierung als fakultativ anzusehen. Andererseits bieten diese Verfahren allerdings eine hervoragende Möglichkeit der Wiederholung von Dualsystemen. In jedem Fall, sollte den Schülerinnen und Schülern die Wichtigkeit der geeigneten Wahl der Codierlänge bewußt werden, da hier unerwünschte Effekte von Komprimierungsalgorithmen (bei ungeeigneter Wahl der Parameter) in praktischer Weise vorgeführt werden können (vgl. Folie: „*Laufkomprimierung (3) – Beispiel*“).

Da die verlustbehafteten Komprimierungsverfahren (im Vergleich zu den verlustfreien) wesentlich höhere Kompressionsraten erreichen können, sollte im Anschluß unbedingt ein solches behandelt werden. Nach einer Motivation, bei der die Folie „*Komprimierung von Bildern – Beispiel*“, eine nützliche Hilfestellung bietet, können dann verschiedene Lösungsansätze zur Komprimierung von Grafiken diskutiert werden. Anschließend kann dann die Vorstellung des – auf Blockbildung und Filterung basierenden – Komprimierungsverfahrens stattfinden. Die Theorie hierzu ist recht einfach, so dass im unmittelbaren Anschluß das Verfahren direkt implementiert werden kann (*Aufgabe 4.1*). Aufgrund der zur Verfügung stehenden Routinen dürfte dies auch kein größeres Problem darstellen. Ist das Verfahren einmal realisiert, können die Schülerinnen und Schüler durch Wahl der Parameter „Blockgroesse“ und „Filterung“ für eine beliebige (Windows 3.0)-Bitmap die Auswirkungen eines verlustbehafteten Komprimierungsverfahrens visuell erfahren und so die Probleme und Grenzen von Komprimierungsverfahren intuitiv aufdecken. (*Aufgabe 4.2*). Die hier verwendeten Bilder dürften sich dafür eignen (vgl. Folien „*Bildkomprimierung durch Blockbildung (..)*“). Nach einer anschließenden Klassifikation der vorgestellten Verfahren (Folie „*Einordnung der Komprimierungsverfahren*“) können dann zur Arrondierung dieser Unterrichtseinheit allgemeine Vor- und Nachteile der Komprimierungsverfahren diskutiert werden.

5.2. Detailierte Unterrichtsplanung

Im folgenden wird eine tabellarische Übersicht des geplanten Unterrichtsverlaufs gegeben, in dem die einzelnen Phasen mit ihren jeweiligen Inhalten, Methoden, Sozialformen und den verwendeten Medien dargestellt werden. Hierbei werden folgende Abkürzungen verwendet:

- EA : Einzelarbeit
- GA : Gruppenarbeit
- LV : Lehrvortrag
- SV : Schülervortrag
- UG : Unterrichtsgespräch
- A : Arbeitsblatt
- C : Computer
- D : Diskussion
- F : Folien
- (f) : fakultativer Schritt / Phase

1. Schritt:

Phase	Inhalt	Methode / Sozialform	Medien
Motivation	Einführung in das Thema und Motivation	UG	F
Erarbeitung	Verschiedene Lösungsansätze zur Komprimierung von Texten	D, GA / EA	-
Sicherung	Besprechung und Abgrenzung der gefundenen Schülerlösungen	SV, D	-
Erarbeitung	Vorstellung der dynamischen Huffman-Komprimierung anhand eines konkreten Beispiels	LV	F
Sicherung, Vertiefung	Üben der Huffman-Codierung und finden von Effizienzkriterien (Aufgabe 1.1)	EA	A
Erarbeitung	Dekomprimierung einer Huffman-codierten Textpassage	LV / UG	F
Sicherung, Vertiefung	Üben der Huffman-Dekomprimierung und erkennen der eindeutigen Rekonstruierbarkeit (Aufgabe 1.2)	EA	A
Vorbereitung	Austeilung von Arbeitsblatt 2.	-	A

2. Schritt:

Phase	Inhalt	Methode / Sozialform	Medien
Programmierung	Implementierung des Huffman-Verfahrens (Aufgabe 2.1)	GA / EA	A, C
Vertiefung	Hausaufgabe: Abschließen der Programmier-aufgabe und Bearbeitung der Aufgabe 2.2 (Erweiterbarkeit des Huffman-Verfahrens) und Aufgabe 2.3 (Zusätzliche Komprimierungs-möglichkeiten)	GA / EA	A, C

3. Schritt:

Phase	Inhalt	Methode / Sozialform	Medien
Wiederholung	Besprechung des Hausaufgabe	UG, D	-
Überleitung	In Anlehnung an Aufgabe 2.3 (Hausaufgabe): Diskussion über zusätzliche Komprimierungs-möglichkeiten um eine glatte Überleitung zur Bitlistenkomprimierung zu erhalten.	UG, D	-
Erarbeitung	Vorstellen der Laufkomprimierung	LV	F
Sicherung, Vertiefung	Üben der Laufkomprimierung (Aufgabe 3.1): Komprimierung / Dekomprimierung		A
Vorbereitung	Hausaufgabe: Wiederholung der Lauf – komprimierung	EA	-

4. Schritt(f):

Phase	Inhalt	Methode / Sozialform	Medien
Programmierung(f)	Implementierung der RLE-Komprimierung (Aufgabe 3.2)	GA / EA	A, C

5. Schritt:

Phase	Inhalt	Methode / Sozialform	Medien
Überleitung, Motivation	Motivation und Übergang zur verlustbehafteten Komprimierung	LV / UG, D	F
Erarbeitung	Diskussion verschiedener Lösungsansätze zur Komprimierung von Grafiken	D	-
Sicherung	Besprechung und Abgrenzung der gefundenen Schülerlösungen	SV, D	-
Erarbeitung	Vorstellung des – auf Blockbildung und Filterung basierenden – Komprimierungsverfahrens	LV	F
Programmierung	Implementierung des vorgestellten Verfahrens (Aufgabe 4.1)	GA / EA	A, C

6. Schritt:

Phase	Inhalt	Methode / Sozialform	Medien
Programmierung	Abschließen der Programmieraufgabe (Aufgabe 4.1)	GA / EA	C
Sicherung, Vertiefung und Abrundung	Arbeiten mit der erstellten Software, Kennenlernen des Anwendungsgebiets, Grenzen des vorgestellten Verfahrens, und Diskussion über generelle Vor- und Nachteile der Komprimierung (Aufgabe 4.2)	UG, D	A, C

A. Quelltexte

A.1. Unit „Huffman“

Quelltext:	Huffman.pas
Programmiersprache:	Borland Pascal 7.0 für Windows
Bemerkung:	Durch Entfernen der unterstrichenen Wörter ist die Unit ebenfalls unter Turbo Pascal 6.0 verwendbar.

```
UNIT HUFFMAN;

INTERFACE

{ ------------------------------------------------------------------
  - Typdeklarationen
  ------------------------------------------------------------------ }

TYPE
  PListe                = ^TListe;
  PKnoten               = ^TKnoten;
  PHuffman              = ^THuffman;
  PCodierung            = ^TCodierung;

  TListe                = RECORD
                            NaechsterKnoten      : PListe;
                            Knoten               : PKnoten;
                          END;
  TKnoten               = RECORD
                            linkerSohn           : PKnoten;
                            rechterSohn          : PKnoten;
                            Muster               : String;
                            relativeHaeufigkeit  : Longint;
                          END;
  THuffman              = RECORD
                            OrginalText          : String;
                            TextKomprimiert      : String;
                            BaumWurzel           : PKnoten;
                            Binaercodetabelle    : PCodierung;
                          END;
  TCodierung            = RECORD
                            NaechsteCodierung    : PCodierung;
                            Muster               : String;
                            Code                 : String;
                          END;

{ ------------------------------------------------------------------
  - Liste
  ------------------------------------------------------------------ }

FUNCTION NeuerKnoten : PKnoten;
PROCEDURE NeueListe(VAR AList:PListe);
PROCEDURE SetzeKnotenInListe(AKnoten:PKnoten;VAR AList:PListe);
PROCEDURE ErweitereListe(VAR AList:PListe;AMuster:String;
  AAnzahl:Longint);
FUNCTION ElementeInDerListe(VAR AList:PListe):Longint;
FUNCTION NummerDesKleinstenElementes(AList:PListe):Word;
PROCEDURE VerschiebeInKnoten(VAR AList:PListe;APos:Word;VAR AKnoten:PKnoten);

{ ------------------------------------------------------------------
  - Hilfsroutinen
  ------------------------------------------------------------------ }

FUNCTION NaechstesMuster(VAR AText:String;VAR AMuster:String;
  VAR AAnzahl:Longint) : BOOLEAN;
PROCEDURE ErmittleBinaercodes(VAR AHuffman:THuffman);
FUNCTION ZugehoerigerBinaercode(VAR AHuffman:THuffman;
  Muster:String) : String;
PROCEDURE GebeSpeicherFrei(VAR AHuffman:THuffman);

{ ------------------------------------------------------------------
  - Ausgaben
  ------------------------------------------------------------------ }

PROCEDURE AusgabeListe(Titel:String;VAR AList:PListe);
PROCEDURE AusgabeHuffmanBaum(VAR AHuffman:THuffman);
PROCEDURE AusgabeHuffmanCodes(VAR AHuffman:THuffman);
```

```
{ ---------------------------------------------------------------
  - Init
  --------------------------------------------------------------- }

PROCEDURE InitHuffman(VAR Text:String);
PROCEDURE DoneHuffman;

IMPLEMENTATION

USES
  WinCrt,Strings;

VAR
  Dateiname             : String;

{ ---------------------------------------------------------------
  - Liste
  --------------------------------------------------------------- }

FUNCTION NeuesBlatt(AMuster:String;AAnzahl:Longint) : PKnoten;
VAR
  ABlatt                : PKnoten;
BEGIN
  New(ABlatt);
  WITH ABlatt^ DO BEGIN
    linkerSohn := NIL;
    rechterSohn := NIL;
    Muster := AMuster;
    relativeHaeufigkeit := AAnzahl;
  END; { WITH }
  NeuesBlatt := ABlatt;
END; { NeuesBlatt (P) }

FUNCTION NeuerKnoten : PKnoten;
VAR
  Knoten                : PKnoten;
BEGIN
  New(Knoten);
  WITH Knoten^ DO BEGIN
    linkerSohn := NIL;rechterSohn := NIL;
    Muster := '';relativeHaeufigkeit := 0;
  END; { WITH }
  NeuerKnoten := Knoten;
END; { NeuerKnoten (F) }

PROCEDURE NeueListe(VAR AList:PListe);
BEGIN
  AList := NIL;
END; { NeueListe (P) }

PROCEDURE SetzeKnotenInListe(AKnoten:PKnoten;VAR AList:PListe);
VAR
  LetzterEintrag        : PListe;
BEGIN

  { Einfuegeposition lokalisieren }
  IF (AList = NIL) THEN BEGIN

    { Die Liste war bislang leer }
    New(AList);
    LetzterEintrag := AList;

  END ELSE BEGIN

    { Ans Ende der Liste neuen Knoten eintragen}
    LetzterEintrag := AList;
    WHILE (LetzterEintrag^.NaechsterKnoten <> NIL) DO
      LetzterEintrag := LetzterEintrag^.NaechsterKnoten;
    New(LetzterEintrag^.NaechsterKnoten);
    LetzterEintrag := LetzterEintrag^.NaechsterKnoten;

  END; { ELSE }

  { Knoten eintragen }
  WITH LetzterEintrag^  DO BEGIN
    NaechsterKnoten := NIL;
    Knoten := AKnoten;
  END; { WITH }
END; { SetzeKnotenInListe (P) }


PROCEDURE ErweitereListe(VAR AList:PListe;AMuster:String;AAnzahl:Longint);
VAR
  Knoten                : PKnoten;
BEGIN
  Knoten:=NeuesBlatt(AMuster,AAnzahl);
  SetzeKnotenInListe(Knoten,AList);
END; { ErweitereListe (P) }
```

```
FUNCTION ElementeInDerListe(VAR AList:PListe):Longint;
VAR
  Anzahl               : Longint;
  LetzterEintrag       : PListe;
BEGIN
  Anzahl := 0;
  IF (AList <> NIL) THEN BEGIN
    LetzterEintrag := AList;
    WHILE (LetzterEintrag <> NIL) DO BEGIN
      Anzahl := Anzahl+1;
      LetzterEintrag := LetzterEintrag^.NaechsterKnoten;
    END; { WHILE }
  END; { IF }
  ElementeInDerListe := Anzahl;
END; { ElementeInDerListe (F) }

FUNCTION NummerDesKleinstenElementes(AList:PListe):Word;
VAR
  KleinstesElement     : Word;
  Nummer               : Word;
  LetzterEintrag       : PListe;
  Wert                 : Longint;
BEGIN

  { Initialisierung }
  KleinstesElement := 0; Nummer := 1;

  { Kleinstes Element suchen }
  LetzterEintrag := AList;

  WHILE (LetzterEintrag <> NIL) DO BEGIN

    { Vergleich durchfuehren, ggf. neue Position uebernehmen }
    IF (LetzterEintrag^.Knoten <> NIL) THEN BEGIN
      WITH LetzterEintrag^.Knoten^ DO BEGIN
        IF (KleinstesElement = 0) THEN BEGIN
          KleinstesElement := 1;
          Wert := relativeHaeufigkeit
        END ELSE IF (Wert > relativeHaeufigkeit) THEN BEGIN
          Wert := relativeHaeufigkeit;
          KleinstesElement := Nummer;
        END; { IF }
      END; { WITH }
    END; { IF }

    { Naechstes Element pruefen }
    LetzterEintrag := LetzterEintrag^.NaechsterKnoten;
    Nummer:=Nummer+1;

  END; { WHILE }

  { Ergebnis zurueckliefern }
  NummerDesKleinstenElementes := KleinstesElement;
END; { NummerDesKleinstenElementes (F) }

PROCEDURE VerschiebeInKnoten(VAR AList:PListe;APos:Word;VAR AKnoten:PKnoten);
VAR
  Nummer               : Word;
  Eintrag              : PListe;
BEGIN
  IF (APos<=ElementeInDerListe(AList)) THEN BEGIN
    IF (APos = 1) THEN BEGIN
      AKnoten := AList^.Knoten;
      AList:=AList^.NaechsterKnoten;
    END ELSE BEGIN
      Eintrag := AList;
      Nummer := 2;
      WHILE (Nummer < APos) DO BEGIN
        Eintrag := Eintrag^.NaechsterKnoten;
        Nummer := Nummer + 1;
      END; { WHILE }
      WITH Eintrag^ DO BEGIN
        AKnoten := NaechsterKnoten^.Knoten;
        NaechsterKnoten := NaechsterKnoten^.NaechsterKnoten;
      END; { WITH }
    END; { ELSE }
  END; { IF }
END; { VerschiebeInKnoten (P) }
```

```
{ ------------------------------------------------------------------
  - Hilfsroutinen
  ------------------------------------------------------------------ }

FUNCTION NaechstesMuster(VAR AText:String;VAR AMuster:String;
  VAR AAnzahl:Longint) : BOOLEAN;
BEGIN
  IF AText<>'' THEN BEGIN

    { Neues Zeichen ermitteln }
    AMuster := Copy(AText,1,1);

    { Hauefigkeit ermitteln und aus dem Text entfernen }
    AAnzahl:=0;
    WHILE Pos(AMuster,AText)<>0 DO BEGIN
      AAnzahl:=AAnzahl+1;
      Delete(AText,Pos(AMuster,AText),1);
    END; { WHILE }
    NaechstesMuster := TRUE;
  END ELSE
    NaechstesMuster := FALSE;
END; { NaechstesMuster (F) }

PROCEDURE BinaercodeEintragen(VAR AHuffman:THuffman;AMuster:String;
  ACode:String);
VAR
  LetzterEintrag        : PCodierung;
BEGIN
  WITH AHuffman DO BEGIN
    IF (Binaercodetabelle = NIL) THEN BEGIN

      { Die Liste war bislang leer }
      New(Binaercodetabelle);
      LetzterEintrag := Binaercodetabelle;

    END ELSE BEGIN

      { Ans Ende der Liste neuen Code eintragen}
      LetzterEintrag := Binaercodetabelle;
      WHILE (LetzterEintrag^.NaechsteCodierung <> NIL) DO
        LetzterEintrag := LetzterEintrag^.NaechsteCodierung;
      New(LetzterEintrag^.NaechsteCodierung);
      LetzterEintrag := LetzterEintrag^.NaechsteCodierung;

    END; { ELSE }

    { Code eintragen }
    WITH LetzterEintrag^ DO BEGIN
      NaechsteCodierung := NIL;
      Muster := AMuster;Code := ACode;
    END; { WITH }
  END; { WITH }
END; { BinaercodeEintragen (P) }

PROCEDURE NaechstenBinaercodeSuchen(VAR AHuffman:THuffman;
  VAR AKnoten:PKnoten;ACode:String);
VAR
  Blatt                 : Boolean;
BEGIN
  IF AKnoten <> NIL THEN WITH AKnoten^ DO BEGIN
    Blatt := (linkerSohn = NIL) AND (rechterSohn = NIL);
    IF Blatt THEN
      BinaercodeEintragen(AHuffman,Muster,ACode)
    ELSE BEGIN
      NaechstenBinaercodeSuchen(AHuffman,linkerSohn,ACode+'0');
      NaechstenBinaercodeSuchen(AHuffman,rechterSohn,ACode+'1');
    END; { ELSE }
  END; { WITH }
END; { NaechstenBinaercodeSuchen (P) }

PROCEDURE ErmittleBinaercodes(VAR AHuffman:THuffman);
BEGIN
  WITH AHuffman DO BEGIN
    Binaercodetabelle := NIL;
    NaechstenBinaercodeSuchen(AHuffman,BaumWurzel,'');
  END;
END; { ErmittleBinaercodes (P) }
```

```
FUNCTION ZugehoerigerBinaercode(VAR AHuffman:THuffman;
  Muster:String) : String;
VAR
  LetzterEintrag        : PCodierung;
  Ergebnis              : String;
BEGIN
  Ergebnis:='';
  IF (AHuffman.Binaercodetabelle <> NIL) THEN BEGIN
    LetzterEintrag := AHuffman.Binaercodetabelle;
    WHILE ((LetzterEintrag <> NIL) AND (Ergebnis='')) DO BEGIN
      IF (LetzterEintrag^.Muster = Muster) THEN
        Ergebnis := LetzterEintrag^.Code;
      LetzterEintrag := LetzterEintrag^.NaechsteCodierung;
    END; { WHILE }
  END; { IF }
  ZugehoerigerBinaercode := Ergebnis;
END; { ZugehoerigerBinaercode (F) }

PROCEDURE EntferneHuffmanBaum(VAR Wurzel:PKnoten);
BEGIN
  IF (Wurzel<>NIL) THEN WITH Wurzel^ DO BEGIN
    EntferneHuffmanBaum(linkerSohn);
    EntferneHuffmanBaum(rechterSohn);
    Dispose(Wurzel);
    Wurzel := NIL;
  END; { IF }
END; { EntferneHuffmanBaum (P) }

PROCEDURE EntferneHuffmanCodierung(VAR Wurzel:PCodierung);
BEGIN
  IF (Wurzel<>NIL) THEN WITH Wurzel^ DO BEGIN
    EntferneHuffmanCodierung(NaechsteCodierung);
    Dispose(Wurzel);
    Wurzel := NIL;
  END; { IF }
END; { EntferneHuffmanCodierung (P) }

PROCEDURE GebeSpeicherFrei(VAR AHuffman:THuffman);
BEGIN
  EntferneHuffmanBaum(AHuffman.BaumWurzel);
  EntferneHuffmanCodierung(AHuffman.Binaercodetabelle);
END; { GebeSpeicherFrei (P) }

{ ---------------------------------------------------------------
  - Ausgaben
  --------------------------------------------------------------- }

PROCEDURE AusgabeKnoten(VAR AKnoten:PKnoten;ATiefe:Word;
  AMark:Boolean;AVorspann,AHfg:String);
BEGIN
  IF AKnoten <> NIL THEN WITH AKnoten^ DO BEGIN
    IF ATiefe <> 0 THEN
      write(' ':ATiefe);
    IF AMark THEN
      write(AVorspann);
    write('(',relativeHaeufigkeit,AHfg,') ');
    IF (linkerSohn = NIL) AND (rechterSohn = NIL) THEN
      writeln(Muster)
    ELSE BEGIN
      writeln;
      AusgabeKnoten(linkerSohn,ATiefe+3,AMark,'0: ',AHfg);
      AusgabeKnoten(rechterSohn,ATiefe+3,AMark,'1: ',AHfg);
    END; { ELSE }
  END; { WITH }
END; { AusgabeKnoten (P) }

PROCEDURE AusgabeListe(Titel:String;VAR AList:PListe);
VAR
  LetzterEintrag        : PListe;
  Nummer                : Longint;
BEGIN
  writeln;
  writeln('*** ',Titel,' ***');
  writeln;
  LetzterEintrag := AList;
  Nummer:=1;
  WHILE (LetzterEintrag <> NIL) DO BEGIN
    IF (LetzterEintrag^.Knoten <> NIL) THEN BEGIN
      writeln('[',Nummer,']');
      AusgabeKnoten(LetzterEintrag^.Knoten,5,False,'','');
      Nummer := Nummer + 1;
    END; { IF }
    LetzterEintrag := LetzterEintrag^.NaechsterKnoten;
  END; { WHILE }
  writeln;
END; { AusgabeListe (P) }
```

```
PROCEDURE AusgabeHuffmanBaum(VAR AHuffman:THuffman);
VAR
  s                     : String;
  LetzterEintrag        : PListe;
  Nummer                : Longint;
BEGIN
  Str(Length(AHuffman.Orginaltext),s);
  writeln;
  AusgabeKnoten(AHuffman.BaumWurzel,0,True,'','/'+s);
END; { AusgabeHuffmanBaum (P) }

PROCEDURE AusgabeHuffmanCodes(VAR AHuffman:THuffman);
VAR
  LetzterEintrag        : PCodierung;
  Nummer                : Longint;
BEGIN
  IF (AHuffman.Binaercodetabelle <> NIL) THEN BEGIN
    LetzterEintrag := AHuffman.Binaercodetabelle;
    writeln;
    writeln('Muster       Binaercode');
    writeln('-----------------------');
    WHILE (LetzterEintrag <> NIL) DO BEGIN
      WITH LetzterEintrag^ DO BEGIN
        writeln(Muster,' ':13-Length(Muster),Code);
      END; { WITH }
      LetzterEintrag := LetzterEintrag^.NaechsteCodierung;
    END; { WHILE }
    writeln;
  END; { IF }
END; { AusgabeListe (P) }

{ ---------------------------------------------------------------
  - Verwaltung
  --------------------------------------------------------------- }

PROCEDURE InitHuffman(VAR Text:String);
BEGIN
  StrCopy(WindowTitle,'HUFFMAN-Komprimierung');
  ScreenSize.Y:=800;
  writeln('Zu komprimierender Text: ');
  readln(Text);
  IF Text<>'' THEN BEGIN
    writeln;
    writeln('Ausgabedatei : (keine Angabe führt zu Bildschirmausgabe)');
    readln(Dateiname);
    IF Dateiname<>'' THEN BEGIN
      Assign(Output,Dateiname);
      Rewrite(Output);
    END;
  END ELSE BEGIN
    DoneWinCrt;
    Halt;
  END; { IF }
END; { InitHuffman (P) }

PROCEDURE DoneHuffman;
BEGIN
  IF Dateiname<>'' THEN BEGIN
    Close(Output);
    DoneWinCrt;
  END;
END; { DoneHuffman (P) }

END.
```

A.2. Unit „BitListe“

Quelltext:	BitListe.pas
Programmiersprache:	Borland Pascal 7.0 für Windows
Bemerkung:	Durch Entfernen der unterstrichenen Wörter ist die Unit ebenfalls unter Turbo Pascal 6.0 verwendbar.

```
UNIT BitListe;

INTERFACE

VAR
  MaxWert               : Longint;
  StartBit              : Char;

FUNCTION DateienVorbereitet:Boolean;
PROCEDURE DateienAbschliessen;

PROCEDURE BestimmeMaxWert(CodeLaenge:Byte);
PROCEDURE EingabeCodeLaenge(VAR CodeLaenge:Byte);

FUNCTION LeseWert(Breite:Byte):Longint;
FUNCTION LeseBits(AEinsBit:Boolean):Longint;

PROCEDURE SchreibeWert(Breite:Byte;Wert:Longint);
PROCEDURE SchreibeBits(EinsBits:Boolean;Anzahl:Longint);

FUNCTION WeitereBitsVorhanden : BOOLEAN;

PROCEDURE Done;

IMPLEMENTATION

USES
  WinCrt,Strings;

VAR
  { Dateien }
  Quelle                : File of Byte;
  Ziel                  : File of Byte;
  QuelleGroesse         : Longint;

  { Pufferung }
  Lesespeicher          : String;
  Schreibspeicher       : String;

FUNCTION ByteNachBit(AByte:Byte):String;
VAR
  ByteStr               : String[8];
  B,Faktor              : Byte;
BEGIN
  Faktor := 128; ByteStr:='00000000';
  FOR B := 1 TO 8 DO BEGIN
    ByteStr[B]:=Chr(48+Byte((AByte DIV Faktor)=1));
    AByte := AByte MOD Faktor;
    Faktor := Faktor SHR 1;
  END; { FOR }
  ByteNachBit := ByteStr;
END; { ByteNachBit (P) }

PROCEDURE SchreibspeicherErweitern(AErweiterung:String);
VAR
  Bit8Str               : String[8];
  B,Faktor,Wert         : Byte;

  PROCEDURE Fuelle_Schreibspeicher;
  VAR
    FreieBits           : Byte;
  BEGIN
    FreieBits := 255-Length(Schreibspeicher);
    Schreibspeicher := Schreibspeicher +
      Copy(AErweiterung,1,FreieBits);
    Delete(AErweiterung,1,FreieBits);
  END; { Fuelle_Schreibspeicher (P) }
```

```
BEGIN
  Fuelle_Schreibspeicher;
  WHILE Length(Schreibspeicher)>=8 DO BEGIN
    Bit8Str := Copy(Schreibspeicher,1,8);
    Delete(Schreibspeicher,1,8);
    Fuelle_Schreibspeicher;
    Faktor := 128; Wert := 0;
    FOR B := 1 TO 8 DO BEGIN
      Inc(Wert,Faktor * (Ord(Bit8Str[B])-48));
      Faktor := Faktor SHR 1;
    END; { FOR }
    write(Ziel,Wert);
    gotoxy(1,wherey);
    write('Status: ',filepos(quelle),' Byte(s) von ',QuelleGroesse,' Byte(s) gelesen.');
    clreol;
  END; { WHILE }
END; { SchreibspeicherErweitern (P) }

FUNCTION WeitereBitsVorhanden : BOOLEAN;
BEGIN
  WeitereBitsVorhanden := (Length(Lesespeicher) <> 0) OR (NOT EOF(Quelle))
END; { WeitereBitsVorhanden (F) }

PROCEDURE Fuelle_Lesespeicher;
VAR
  B                    : Byte;
BEGIN
  WHILE (NOT EOF(Quelle)) AND ((Length(Lesespeicher)+8)<=255) DO
  BEGIN
    read(Quelle,B);
    Lesespeicher := Lesespeicher + ByteNachBit(B);
  END; { WHILE }
END; { Fuelle_Lesespeicher (P) }

FUNCTION LeseBits(AEinsBit:Boolean):Longint;
VAR
  Loop                 : Boolean;
  Ergebnis             : Longint;
  AnderesBit           : Char;
  P                    : Byte;
BEGIN
  AnderesBit := Chr(48 + Byte(AEinsBit = FALSE));
  Ergebnis := 0; Loop := True;
  WHILE Loop DO BEGIN
    Fuelle_Lesespeicher;
    Loop := False;
    P := Pos(AnderesBit,Lesespeicher);
    IF (P <> 0) THEN BEGIN

        { Anderes Bit ist noch im Speicher vorhanden }
        Ergebnis := Ergebnis + P - 1;
        IF P<>1 THEN
          Delete(Lesespeicher,1,P-1);

    END ELSE BEGIN

      { Anderes Bit ist nicht im Speicher }
      Ergebnis := Ergebnis + Length(Lesespeicher);
      Lesespeicher := '';
      IF (NOT EOF(Quelle)) THEN
        Loop := True;

    END; { IF }
  END; { WHILE }
  LeseBits := Ergebnis;
END; { LeseBits (F) }

PROCEDURE SchreibeWert(Breite:Byte;Wert:Longint);
VAR
  S                    : String;
BEGIN
  S := '';
  WHILE Wert<>0 DO BEGIN
    Insert(Chr(48 + Wert MOD 2),S,1);
    Wert := Wert DIV 2;
  END; { WHILE }
  WHILE (Length(S) < 32) DO
    Insert('0',S,1);
  S:=Copy(S,32-Breite+1,Breite);
  SchreibspeicherErweitern(S);
END; { SchreibeWert (P) }
```

```
FUNCTION LeseWert(Breite:Byte):Longint;
VAR
  Bits                  : String;
  Ergebnis              : Longint;
BEGIN
  Ergebnis := 0;
  IF Length(Lesespeicher) < Breite THEN
    Fuelle_Lesespeicher;
  IF Length(Lesespeicher) >= Breite THEN BEGIN
    Bits := Copy (Lesespeicher,1,Breite);
    Delete(Lesespeicher,1,Breite);
    WHILE Length(Bits) > 0 DO BEGIN
      Ergebnis := 2* Ergebnis + Byte(Bits[1]='1');
      Delete(Bits,1,1);
    END; {WHILE }
  END ELSE BEGIN
    Lesespeicher:='';
    Ergebnis := -1;
  END;
  LeseWert := Ergebnis;
END; { LeseWert (P) }

FUNCTION DateiVorhanden(ADatei:String): Boolean;
VAR
  F                     : FILE;
BEGIN
  {$I-} Assign(F, ADatei);Reset(F);Close(F); {$I+}
  DateiVorhanden := (IOResult = 0) AND(ADatei<> '');
END; { DateiVorhanden (F) }

FUNCTION DateienVorbereitet:Boolean;
VAR
  Ergebnis              : Boolean;
  Dateiname             : String;
BEGIN
  ClrScr;write('Quelldatei : ');
  readln(Dateiname);
  IF DateiVorhanden(Dateiname) THEN BEGIN
    Assign(Quelle,Dateiname);
    Reset(Quelle);
    QuelleGroesse := FileSize(Quelle);
    write('Zieldatei : ');
    readln(Dateiname);
    Writeln;
    Assign(Ziel,Dateiname);
    Rewrite(Ziel);
    Lesespeicher:='';
    Fuelle_Lesespeicher;
    StartBit := Lesespeicher[1];
    Schreibspeicher:='';
    Ergebnis := True;
  END ELSE BEGIN
    Writeln('Quelldatei ',Dateiname,' existiert nicht. <RETURN>');
    Readln;
    Ergebnis := False;
  END; { IF }
  DateienVorbereitet := Ergebnis;
END; { DateienVorbereitet (F) }

PROCEDURE DateienAbschliessen;
VAR
  Nullen                : String;
  GroesseZ              : Longint;
BEGIN

  { Buffer noch komplett schreiben, ggf. mit Nullen auffüllen }
  SchreibspeicherErweitern('0000000');
  Schreibspeicher:='';

  { Dateigroessen ausgeben und Dateien schließen }
  GroesseZ := FileSize(Ziel);
  {$I-} Close(Quelle);Close(Ziel); {$I+}
  writeln;writeln;
  writeln('Dateigroesse der Orginaldatei : ',QuelleGroesse,' Bytes .');
  writeln('Dateigroesse der Zieldatei : ',GroesseZ,' Bytes .');
  write('Dies entspricht einem Unterschied von ');
  write(Abs(QuelleGroesse-GroesseZ),' Bytes (ca. ');;
  writeln(Round(100*(QuelleGroesse-GroesseZ)/QuelleGroesse),'% Ersparnis) .');;
  writeln('<RETURN>');{readln;}
END; { DateienAbschliessen (P) }

PROCEDURE Done;
BEGIN
  DoneWinCrt;
END; { Done (P) }
```

```
PROCEDURE SchreibeBits(EinsBits:Boolean;Anzahl:Longint);
BEGIN
  WHILE Anzahl > 0 DO BEGIN
    SchreibeWert(1,Byte(Einsbits = TRUE));
    Dec(Anzahl);
  END;  { WHILE }
END; { SchreibeBits (P) }

PROCEDURE BestimmeMaxWert(CodeLaenge:Byte);
BEGIN
  { Berechnet 2^CodeLaenge -1
    Anmerkung:  Nur MaxWert := (1 SHL CodeLaenge)-1 funktioniert nicht,
    da SHL hier nur auf Integer operieren würde, Ab CodeLaengen >= 16 würden
    hier somit Fehler entstehen => Auf Longint-Basis zwingen}
  MaxWert := 1;
  MaxWert := MaxWert SHL CodeLaenge;
  Dec(MaxWert);
END; { BestimmeMaxWert (P) }

PROCEDURE EingabeCodeLaenge(VAR CodeLaenge:Byte);
BEGIN
  write('Codierungslänge k (1 <= k <= 31) : ');
  readln(CodeLaenge);writeln;
END; { EingabeCodeLaenge (P) }

BEGIN
  StrCopy(WindowTitle,'Bitlistenkomprimierung');
  InitWinCrt;
END.
```

A.3. Unit „GFenster“

Quelltext:	GFenster.pas
Programmiersprache:	Borland Pascal 7.0 für Windows
Bemerkung:	Dieser Quelltext ist leider nicht nach Turbo Pascal 6.0 übertragbar, da dort primär die GDI-Funktionen nicht verfügbar sind. Um eine Implementierung in Turbo Pascal 6.0 zu erreichen, ist es notwendig eine eigene Unit mit den benötigten Grafikroutinen zu entwickeln, da die mitgelieferten Grafiktreiber der Unit „Graph“ für den hiesigen Zweck völlig unreichend sind (auf Standardsystemen, die keine zur „IBM 8514“ kompatiblen Grafikkarte besitzen, sind maximal 16 Farben möglich)

```
UNIT GFenster;

{$R GFenster.res }

INTERFACE

USES
  WinTypes,OWindows;

CONST
  menue_GrafikLaden     = 101;
  menue_Einstellungen   = 102;
  menue_Beenden         = 103;

TYPE
  TEinstellung          = RECORD
    Blockgroesse            : Integer;
    RFilter                 : Boolean;
    GFilter                 : Boolean;
    BFilter                 : Boolean;
    StatusText              : String;
  END; { TEinstellung (R) }

  PGrafikfenster        = ^TGrafikfenster;
  TGrafikfenster        = OBJECT(TWindow)
    Cfg                     : TEinstellung;
    BitMapHandle            : HBitmap;
    Bildbreite              : Longint;
    Bildhoehe               : Longint;
    Quelle,Ziel             : TRect;
    PaintDCHandle           : HDC;
    Dateiname               : String;

    CONSTRUCTOR Init(AParent: PWindowsObject; Title: PChar);
    DESTRUCTOR Done;
      Virtual;
    PROCEDURE GetWindowClass(VAR WndClass: TWndClass);
      Virtual;
    FUNCTION OpenDIB(VAR TheFile: File): Boolean;
    PROCEDURE Paint(PaintDC: HDC;VAR PaintInfo: TPaintStruct);
      Virtual;
    PROCEDURE ErmittleFenstergroesse(VAR W,H:Integer);
    PROCEDURE ErmittlePixelFarbe(X,Y:Integer;VAR R,G,B:Byte);
    PROCEDURE SetzePixelFarbe(X,Y:Integer;R,G,B:Byte);
    PROCEDURE Komprimiere;
      Virtual;
    PROCEDURE BildLaden(VAR Message: TMessage);
      Virtual cm_first + menue_GrafikLaden;
    PROCEDURE Einstellungen(VAR Message: TMessage);
      Virtual cm_first + menue_Einstellungen;
    PROCEDURE Beenden(VAR Message: TMessage);
      Virtual cm_first + menue_Beenden;
  END; { TGrafikfenster (O) }

IMPLEMENTATION

USES
  WinProcs, WinDos, BWCC, Strings, ODialogs, OMemory, OStdDlgs;

CONST
  Abstand               = 3;
  Infozeile             = 16;
  Filter                ='Bitmap (*.bmp)'#0'*.bmp'#0#0;

PROCEDURE AHIncr; FAR; External 'KERNEL' index 114;
```

```
CONSTRUCTOR TGrafikfenster.Init;
VAR
  DC                    : HDC;
BEGIN
  TWindow.Init(AParent, Title);
  BitMapHandle := 0;Attr.W := 400;Attr.H := 100;
  Dateiname := '';Cfg.Blockgroesse := 2;
  Cfg.RFilter := False;Cfg.GFilter := False;Cfg.BFilter := False;
END; { TGrafikfenster.Init (P) }

DESTRUCTOR TGrafikfenster.Done;
BEGIN
  IF BitMapHandle <> 0 THEN DeleteObject(BitMapHandle);
  TWindow.Done;
END; { TGrafikfenster.Init (D) }

PROCEDURE TGrafikfenster.GetWindowClass;
BEGIN
  TWindow.GetWindowClass(WndClass);
  WndClass.HIcon := 0;
  WndClass.hIcon := LoadIcon(HInstance, 'ICON');
  WndClass.lpszMenuName := 'MENUE';
  WndClass.hbrBackground := color_Background;
END; { TGrafikfenster.GetWindowClass (P) }

FUNCTION TGrafikfenster.OpenDIB;
TYPE
  LongType              = RECORD CASE Word OF
                            0: (Ptr: Pointer);
                            1: (Long: Longint);
                            2: (Lo: Word;Hi: Word);
                          END;
VAR
  BitmapInfo            : PBitmapInfo;
  Start, ToAddr, Bits   : LongType;
  BitsPtr               : Pointer;
  longWidth,Count       : Longint;
  BitsHandle            : THandle;
  NewBitmapHandle       : THandle;
  NewPixelWidth         : Word;
  NewPixelHeight        : Word;
  bitCount              : Word;
  Size                  : Word;
  DCHandle              : HDC;
BEGIN
  OpenDIB := True;
  Seek(TheFile, 28);BlockRead(TheFile, bitCount, SizeOf(bitCount));
  IF (bitCount <= 8) THEN BEGIN
    Size := SizeOf(TBitmapInfoHeader) + ((1 shl bitCount) * SizeOf(TRGBQuad));
    BitmapInfo := MemAlloc(Size);
    Seek(TheFile, SizeOf(TBitmapFileHeader));
    BlockRead(TheFile, BitmapInfo^, Size);
    NewPixelWidth := BitmapInfo^.bmiHeader.biWidth;
    NewPixelHeight := BitmapInfo^.bmiHeader.biHeight;
    longWidth := (((NewPixelWidth * bitCount) + 31) div 32) * 4;
    BitmapInfo^.bmiHeader.biSizeImage := longWidth * NewPixelHeight;
    GlobalCompact(-1);
    BitsHandle := GlobalAlloc(gmem_Moveable or gmem_Zeroinit,BitmapInfo^.bmiHeader.biSizeImage);
    Start.Long := 0;
    Bits.Ptr := GlobalLock(BitsHandle);
    Count :=  BitmapInfo^.bmiHeader.biSizeImage - Start.Long;
    WHILE Count > 0 DO BEGIN
      ToAddr.Hi := Bits.Hi + (Start.Hi * Ofs(AHIncr));
      ToAddr.Lo := Start.Lo;
      IF Count > $4000 THEN Count := $4000;
      BlockRead(TheFile, ToAddr.Ptr^, Count);
      Start.Long := Start.Long + Count;
      Count := BitmapInfo^.bmiHeader.biSizeImage - Start.Long;
    END; { WHILE }
    GlobalUnlock(BitsHandle);
    DCHandle := CreateDC('Display', nil, nil, nil);
    BitsPtr := GlobalLock(BitsHandle);
    NewBitmapHandle := CreateDIBitmap(DCHandle, BitmapInfo^.bmiHeader, cbm_Init,BitsPtr,BitmapInfo^, 0);
    DeleteDC(DCHandle);
    GlobalUnlock(BitsHandle);
    GlobalFree(BitsHandle);
    FreeMem(BitmapInfo, size);
    IF NewBitmapHandle <> 0 THEN BEGIN
      IF BitmapHandle <> 0 THEN DeleteObject(BitmapHandle);
      BitmapHandle := NewBitmapHandle;
      BildBreite := NewPixelWidth;
      BildHoehe := NewPixelHeight;
    END ELSE
      OpenDIB := False;
  END ELSE
    OpenDIB := False;
END; { TGrafikfenster.OpenDIB (F) }
```

```
PROCEDURE TGrafikfenster.Paint;
VAR
  LF                    : TLogFont;
  FStr                  : String;
  Rahmen                : TRect;
  PinselAlt,Pinsel1     : HBrush;
  StiftAlt,Stift3       : HPen;
  SchriftAlt,Schrift    : HFont;
  MemDC                 : HDC;

  PROCEDURE ModifiziertesRechteck(VAR Res:TRect;R:TRect;DX,DY:Integer);
  BEGIN
    Res := R;
    Res.Left := Res.Left + DX;
    Res.Right := Res.Right - DX;
    Res.Top := Res.Top + DY;
    Res.Bottom := Res.Bottom + DY;
  END; { TGrafikfenster.Paint.ModifiziertesRechteck (P) }

  PROCEDURE ZeichneRahmen(R:TRect;Invert:Boolean);
  VAR
    Stift1,Stift2       : HPen;
  BEGIN
    IF Invert THEN BEGIN
      Stift1 := CreatePen(ps_Solid, 1, RGB(255,255,255));
      Stift2 := CreatePen(ps_Solid, 1, RGB(128,128,128));
    END ELSE BEGIN
      Stift1 := CreatePen(ps_Solid, 1, RGB(128,128,128));
      Stift2 := CreatePen(ps_Solid, 1, RGB(255,255,255));
    END; { IF }
    SelectObject(PaintDC,Stift1);MoveTo(PaintDC,R.Left,R.Bottom);
    LineTo(PaintDC,R.Left,R.Top);LineTo(PaintDC,R.Right,R.Top);
    SelectObject(PaintDC,Stift2);MoveTo(PaintDC,R.Left+1,R.Bottom);
    LineTo(PaintDC,R.Right,R.Bottom);LineTo(PaintDC,R.Right,R.Top);
    SelectObject(PaintDC,StiftAlt);
    DeleteObject(Stift1);DeleteObject(Stift2);
  END; { TGrafikfenster.Paint.ZeichneRahmen (P) }

BEGIN
  IF (BitMapHandle <> 0) AND (NOT IsIconic(HWindow)) THEN BEGIN

    { Initialisierung }
    Stift3 := CreatePen(ps_Solid,1,RGB(192,192,192));
    Pinsel1 := CreateSolidBrush(RGB(192,192,192));
    FillChar(LF, SizeOf(LF), 0);
    WITH LF DO BEGIN
      lfHeight := 12;
      lfWidth := 6;
      lfWeight := fw_Light;
      lfOutPrecision := out_Character_Precis;
      lfQuality := Proof_Quality;
      StrCopy(lfFaceName, 'Arial')
    END;
    Schrift := CreateFontIndirect(LF);
    StiftAlt := SelectObject(PaintDC,Stift3);
    PinselAlt := SelectObject(PaintDC,Pinsel1);
    SchriftAlt := SelectObject(PaintDC,Schrift);

    { Grafikpositionen berechnen }
    Quelle.Left := 2 + Abstand;
    Quelle.Top :=  2 + Abstand;
    Quelle.Right := Quelle.Left + BildBreite-1;
    Quelle.Bottom := Quelle.Top + BildHoehe + 2*(Abstand+1);
    Ziel := Quelle;
    Ziel.Left := 1 + Bildbreite + 3*(Abstand+2);
    Ziel.Right := Ziel.Left + Bildbreite - 1;

    { Rahmen zeichnen }
    ModifiziertesRechteck(Rahmen,Quelle,-(2+Abstand),-(2+Abstand));
    ZeichneRahmen(Rahmen,True);
    ModifiziertesRechteck(Rahmen,Rahmen,1,1);
    ZeichneRahmen(Rahmen,False);
    ModifiziertesRechteck(Rahmen,Ziel,-(2+Abstand),-(2+Abstand));
    ZeichneRahmen(Rahmen,True);
    ModifiziertesRechteck(Rahmen,Rahmen,1,1);
    ZeichneRahmen(Rahmen,False);

    { Informationszeile vorbereiten, kann in Komprimiere()
      evtl. modifiziert werden}
    Cfg.StatusText := 'Datei: ' + Dateiname + '  -  Einstellungen: ';
    Cfg.StatusText := Cfg.StatusText + 'Blockgroesse:';
    Str(Cfg.Blockgroesse,FStr);
    Cfg.StatusText := Cfg.StatusText + FStr + ' x ' + FStr;
```

```
    FStr := ', Filter:';
    IF Cfg.RFilter THEN FStr := FStr+'Rot,';
    IF Cfg.GFilter THEN FStr := FStr+'Grün,';
    IF Cfg.BFilter THEN FStr := FStr+'Blau,';
    Delete(FStr,Length(FStr),1);
    IF FStr[Length(FStr)] = 'r' THEN
      FStr := FStr + ':(keine)';
    Cfg.StatusText := Cfg.StatusText + FStr;

    { Orginalgrafik zeichnen }
    MemDC := CreateCompatibleDC(PaintDC);
    SelectObject(MemDC, BitMapHandle);
    BitBlt(PaintDC, Quelle.Left, Quelle.Top, BildBreite, BildHoehe,
      MemDC, 0, 0, SRCCopy);

    { Und Komprimierungsvisualisierung vollziehen }
    PaintDCHandle := PaintDC;
    Komprimiere;

    { Inormationszeile zeichnen }
    Rahmen.Left := Quelle.Left - (Abstand + 2);
    Rahmen.Top := Quelle.Bottom;
    Rahmen.Bottom := Rahmen.Top + Infozeile;
    Rahmen.Right := Ziel.Right + Abstand + 2;
    ZeichneRahmen(Rahmen,True);
    ModifiziertesRechteck(Rahmen,Rahmen,3,3);
    WITH Rahmen DO BEGIN
      SetTextColor(PaintDC,RGB(0,0,0));
      SetBKColor(PaintDC,RGB(192,192,192));
      SelectObject(PaintDC,Stift3);
      SelectObject(PaintDC,Pinsel1);
      Rectangle(PaintDC,Left,Top,Right,Bottom-4);

      TextOut(PaintDC,Rahmen.Left+2,Rahmen.Top,
        @Cfg.StatusText[1],Length(Cfg.StatusText));
      DeleteObject(Stift3);DeleteObject(Pinsel1);
    END; { WITH }
    ModifiziertesRechteck(Rahmen,Rahmen,-3,-3);
    ZeichneRahmen(Rahmen,True);
    ModifiziertesRechteck(Rahmen,Rahmen,1,1);
    ZeichneRahmen(Rahmen,False);
    SelectObject(PaintDC,StiftAlt);
    SelectObject(PaintDC,PinselAlt);
    SelectObject(PaintDC,SchriftAlt);
    DeleteObject(Stift3);
    DeleteObject(Pinsel1);
    DeleteObject(Schrift);
    DeleteDC(MemDC);
  END; { IF }
END; { TGrafikfenster.Paint (P) }

PROCEDURE TGrafikfenster.ErmittleFenstergroesse;
BEGIN
  W := 2*BildBreite + 2 * GetSystemMetrics(sm_CXFrame)+4*(Abstand+2);
  H := BildHoehe + 2 * GetSystemMetrics(sm_CYFrame)
    + GetSystemMetrics(sm_CYCaption) + GetSystemMetrics(sm_CYMenu)
    + 2*(Abstand+2) + Infozeile + 5;
END; { TGrafikfenster.ErmittleFenstergroesse (P) }

PROCEDURE TGrafikfenster.ErmittlePixelFarbe;
VAR
  Farbe                : TColorRef;
BEGIN
  Farbe := GetPixel(PaintDCHandle,Quelle.Left+X-1,Quelle.Top+Y-1);
  R := GetRValue(Farbe);G := GetGValue(Farbe);B := GetBValue(Farbe);
END; { TGrafikfenster.ErmittlePixelFarbe (P) }

PROCEDURE TGrafikfenster.SetzePixelFarbe;
VAR
  Farbe                : TColorRef;
BEGIN
  Farbe := SetPixel(PaintDCHandle,Ziel.Left+X-1,Ziel.Top+Y-1,RGB(R,G,B));
END; { TGrafikfenster.SetzePixelFarbe (P) }

PROCEDURE TGrafikfenster.Komprimiere;
BEGIN
END; {  TGrafikfenster.Komprimiere (P) }
```

```
PROCEDURE TGrafikfenster.BildLaden;
VAR
  tmpDateiname          : ARRAY[0..fsPathName] OF Char;
  TestWin30Bitmap       : Longint;
  W,H                   : Integer;
  Datei                 : File;
BEGIN
  IF Application^.ExecDialog(New(PFileDialog,Init(@Self,
    PChar(sd_FileOpen),StrCopy(tmpDateiname, '*.bmp')))) = id_Ok
  THEN BEGIN
    Assign(Datei, tmpDateiname);Reset(Datei, 1);Seek(Datei, 14);
    BlockRead(Datei, TestWin30Bitmap, SizeOf(TestWin30Bitmap));
    IF TestWin30Bitmap = 40 THEN IF OpenDIB(Datei) THEN BEGIN
      Dateiname := StrPas(tmpDateiname);
      ErmittleFenstergroesse(W,H);
      SetWindowPos(HWindow, 0, 0, 0, W,H, swp_NoMove);
    END; { IF }
    Close(Datei);
  END; { IF }
  InvalidateRect(HWindow, nil, False);
END; { TGrafikfenster.BildLaden (P) }

PROCEDURE TGrafikfenster.Einstellungen;
VAR
  Buffer                : RECORD
                            Blockgroesse : Array[0..4] OF Char;
                            R,G,B        : WordBool;
                          END;
  P                     : PWindowsObject;
  Dialog                : PDialog;
  I,Code,W,H            : Integer;
  S                     : String;
  R                     : TRect;
BEGIN

  { Initialisierung }
  Str(Cfg.Blockgroesse,S);StrPCopy(Buffer.Blockgroesse,S);
  Buffer.R := Cfg.RFilter;
  Buffer.G := Cfg.GFilter;
  Buffer.B := Cfg.BFilter;
  Dialog := New(PDialog, Init(@Self, 'Einstellungen'));
  Dialog^.TransferBuffer := @Buffer;
  P := New(PEdit, InitResource(Dialog, 1000, 5));
  P := New(PCheckbox, InitResource(Dialog, 1001));
  P := New(PCheckbox, InitResource(Dialog, 1002));
  P := New(PCheckbox, InitResource(Dialog, 1003));

  { Einstellungsdialog aufrufen }
  IF Application^.ExecDialog(Dialog) = id_OK THEN WITH Cfg DO BEGIN

    { Einstellungen übernehmen }
    RFilter := Buffer.R; GFilter := Buffer.G; BFilter := Buffer.B;
    Val(StrPas(Buffer.Blockgroesse),I,Code); IF Code<>0 THEN I:= 1;
    Blockgroesse := I;

    { Neuzeichnung des Fensters veranlassen }
    GetWindowRect(HWindow,R);
    MoveWindow(HWindow,R.Left,R.Top,0,0,True);
    MoveWindow(HWindow,R.Left,R.Top,R.Right-R.Left,
      R.Bottom-R.Top,True);
  END;
END; { TGrafikfenster.Einstellungen (P) }

PROCEDURE TGrafikfenster.Beenden;
BEGIN
  TGrafikfenster.CloseWindow;
END; { TGrafikfenster.Beenden (P) }

END.
```

A.4. Resourcen-Datei „GFenster.rc“

Quelltext:	GFenster.rc
Programmiersprache:	Borland Pascal 7.0 für Windows / Resource-Workshop 1.02
Bemerkung:	Enthält die Definition des Menüs, des Einstellungsdialogs und des Programmicons.

```
MENUE MENU
BEGIN
  POPUP "&Beispiel"
  BEGIN
    MENUITEM "&Grafik laden...", 101
    MENUITEM "&Einstellungen", 102
    MENUITEM SEPARATOR
    MENUITEM "&Beenden", 103
  END
END

EINSTELLUNGEN DIALOG 20, 20, 208, 140
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CLASS "BorDlg"
CAPTION "Einstellungen"
FONT 8, "Helv"
BEGIN
  CONTROL "", 1, "BorBtn", 1 | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 167, 6, 37, 24
  CONTROL "", 3, "BorShade", 3 | WS_CHILD | WS_VISIBLE, 161, 0, 1, 141
         CONTROL "Unterichtsreihe von Wolfgang Eiden", -1, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE
          | WS_GROUP, 9, 11, 137, 9
  CONTROL "", 999, "BorShade", 32769 | WS_CHILD | WS_VISIBLE, 4, 6, 152, 57
  CONTROL "Beispiel für eine Grafikkomprimierung", -1, "STATIC", SS_LEFT | WS_CHILD |
          WS_VISIBLE | WS_GROUP, 9, 49, 125, 9
  CONTROL "", 999, "BorShade", 32769 | WS_CHILD | WS_VISIBLE, 5, 71, 151, 64
  CONTROL "Thema: Komprimierungsverfahren", -1, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE |
          WS_GROUP, 9, 36, 141, 10
  CONTROL "Blockgröße:", -1, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 9, 76, 40,
          11
  CONTROL "", 1000, "EDIT", ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 54, 74,
          21, 12
        CONTROL "Rot-Filter", 1001, "BorCheck", 3 | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 9, 96, 45,
          10
        CONTROL "Grün-Filter", 1002, "BorCheck", 3 | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 9, 109, 45,
          10
        CONTROL "Blau-Filter", 1003, "BorCheck", 3 | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 9, 122, 45,
          10
END

ICON ICON
BEGIN
  '00 00 01 00 01 00 10 10 10 00 00 00 00 00 28 01'
  '00 00 16 00 00 00 28 00 00 00 10 00 00 00 20 00'
  '00 00 01 00 04 00 00 00 00 00 80 00 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 00 00 80 00 00 80 00 00 00 80 80 80 00 80'
  '00 00 80 00 80 00 80 80 00 00 80 80 80 00 C0 C0'
  'C0 00 00 00 FF 00 00 FF 00 00 00 FF FF 00 FF 00'
  '00 00 FF 00 FF 00 FF FF 00 00 FF FF FF 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 08 80 00 00 00 00 00 00 00 08 80 00 00 00'
  'FF F8 80 F0 08 80 00 0F FF 88 0F FF F0 00 00 FF'
  'F8 80 FF FF FF 00 08 FF 88 0F FF FF F0 00 00 88'
  '80 FF FF FF 00 00 00 00 00 00 FF F0 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF'
  '00 00 FF FF 00 00 FF FF 00 00 FF 9F 00 00 FF 07'
  '00 00 F0 01 00 00 E0 00 00 00 C0 01 00 00 80 01'
  '00 00 00 03 00 00 00 07 00 00 C4 0F 00 00 FF 1F'
  '00 00 FF FF 00 00 FF FF 00 00 FF FF 00 00'
END
```

Das Verfahren der Komprimierung

Abbildung 1: Datentransfer ohne Komprimierung

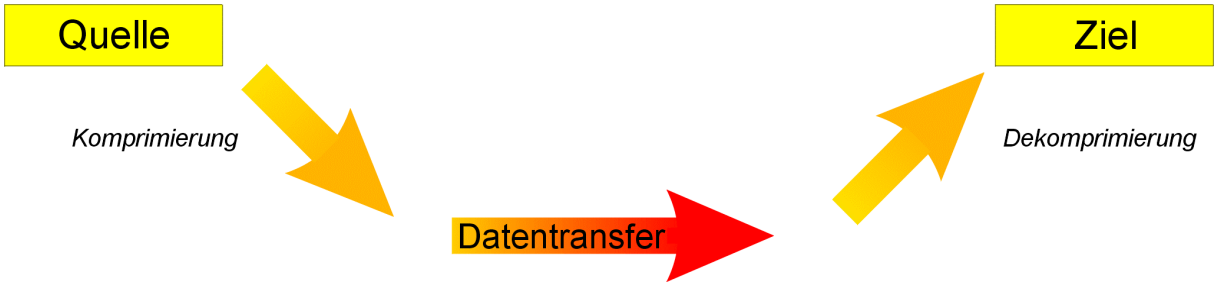


Abbildung 2: Datentransfer mit vorheriger Komprimierung und anschließender Dekomprimierung

Vorteile der Komprimierung

Vorteile der Komprimierung :

- Minimierung von Übertragungszeit
- Minimierung von Speicherplatzbedarf

Es gilt der **Grundsatz der EDV :**

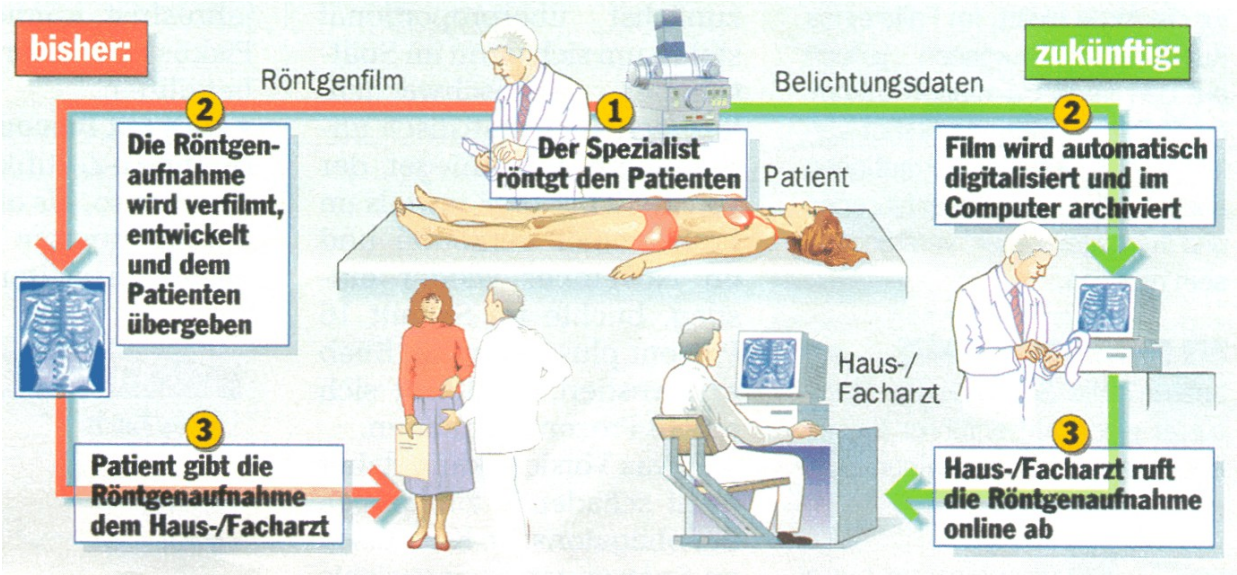
Datenübertagung und Speicherplatz sind im Vergleich zur Rechenleistung relativ teuer.

Anwendungsbeispiele

Typische Anwendungsbereiche der Datenkomprimierung :

- Datenfernübertragung
- Backup-Systeme
- Multimediale Anwendungen und Spiele
- Bildverarbeitung
- Videokonferenzen

Beispiel : Online-Abfrage von Patienteninformationen



Quelle: Stern 21 / 1999

Huffman-Algorithmus – Beispiel (1)

Idee:

Häufig verwendeten Zeichen werden kürzere Codes zugeordnet als seltener verwendeten.

Ausgangstext:

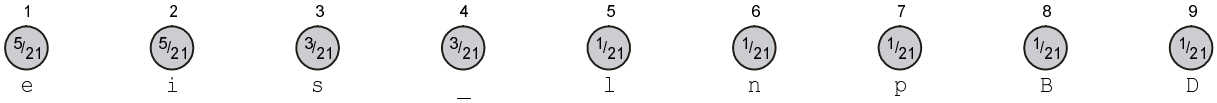
```
Dies_sei_ein_Beispiel
```

Zeichenhäufigkeiten:

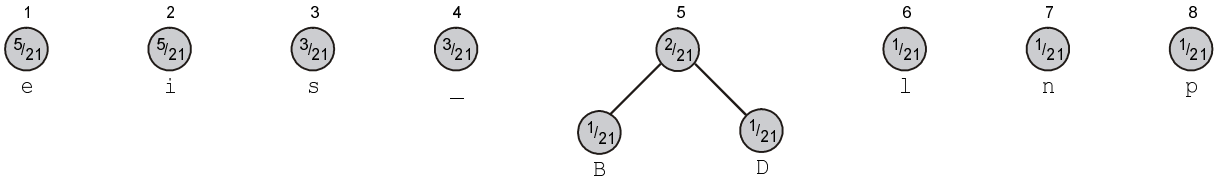
Zeichen	Häufigkeit	
	relativ	absolut
D	1	1/21
i	5	5/21
e	5	5/21
s	3	3/21
_	3	3/21
n	1	1/21
B	1	1/21
p	1	1/21
l	1	1/21

Huffman-Algorithmus – Beispiel (2)

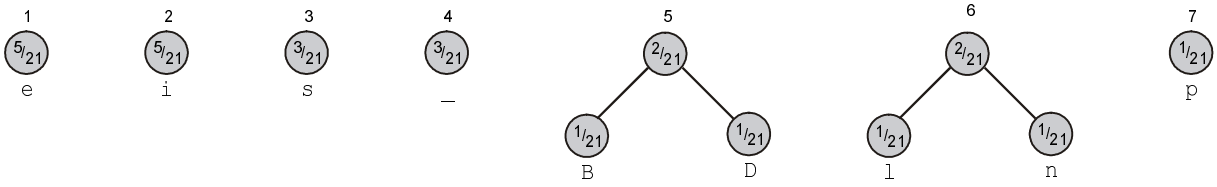
Finden der optimalen Codierung: Aufbau des Huffman-Baums



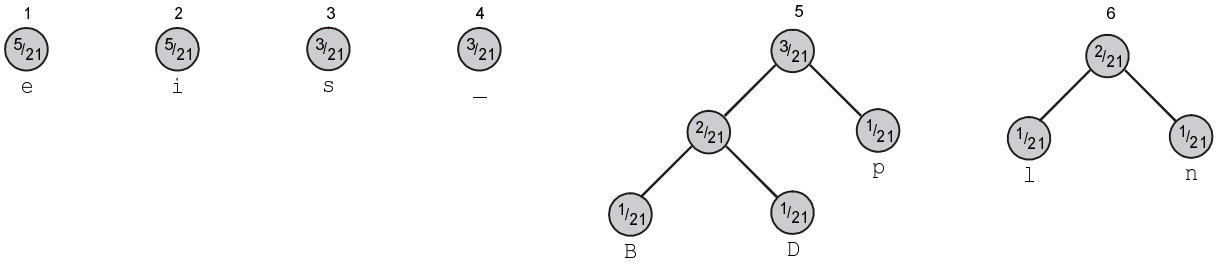
Initialisierung



Ergebnis nach dem 1. Schritt



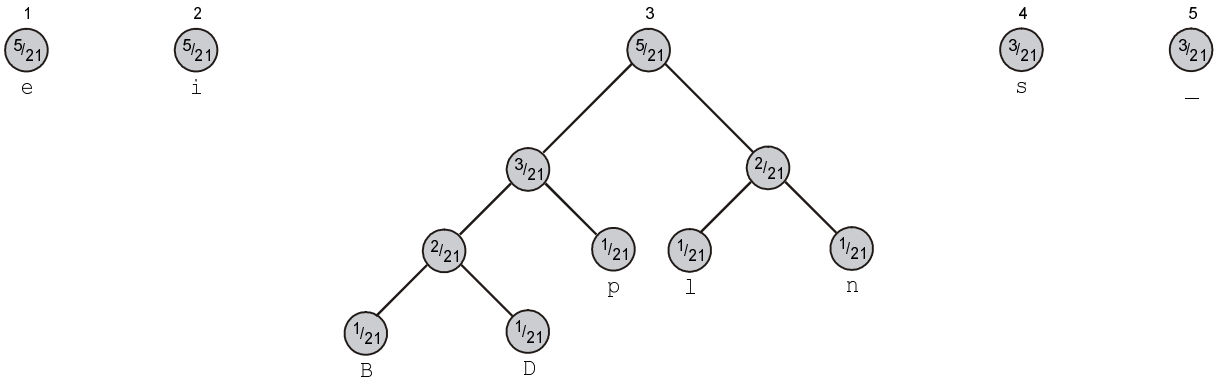
Ergebnis nach dem 2. Schritt



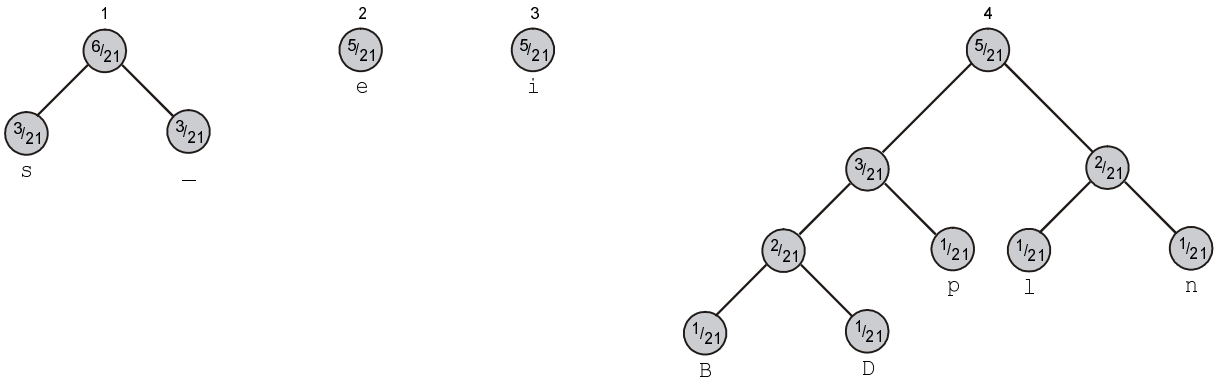
Ergebnis nach dem 3. Schritt

Huffman-Algorithmus – Beispiel (3)

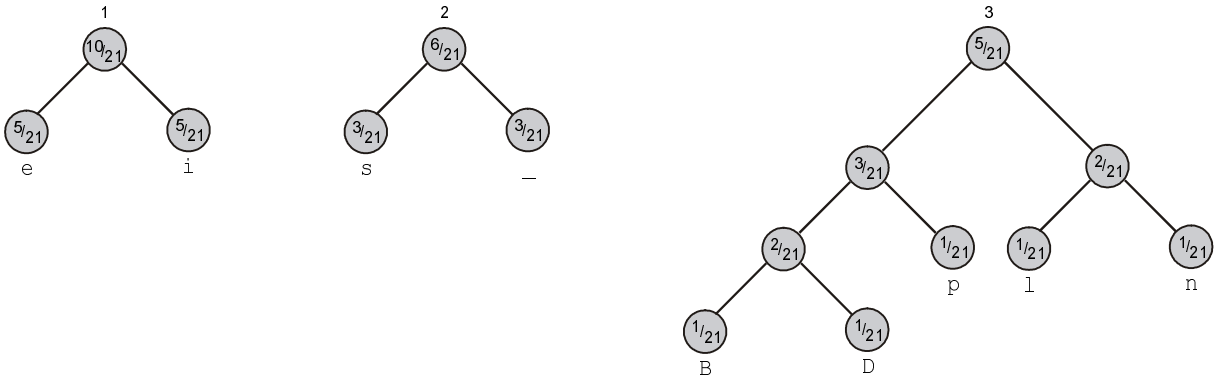
Aufbau des Huffman-Baums (Fortsetzung)



Ergebnis nach dem 4. Schritt



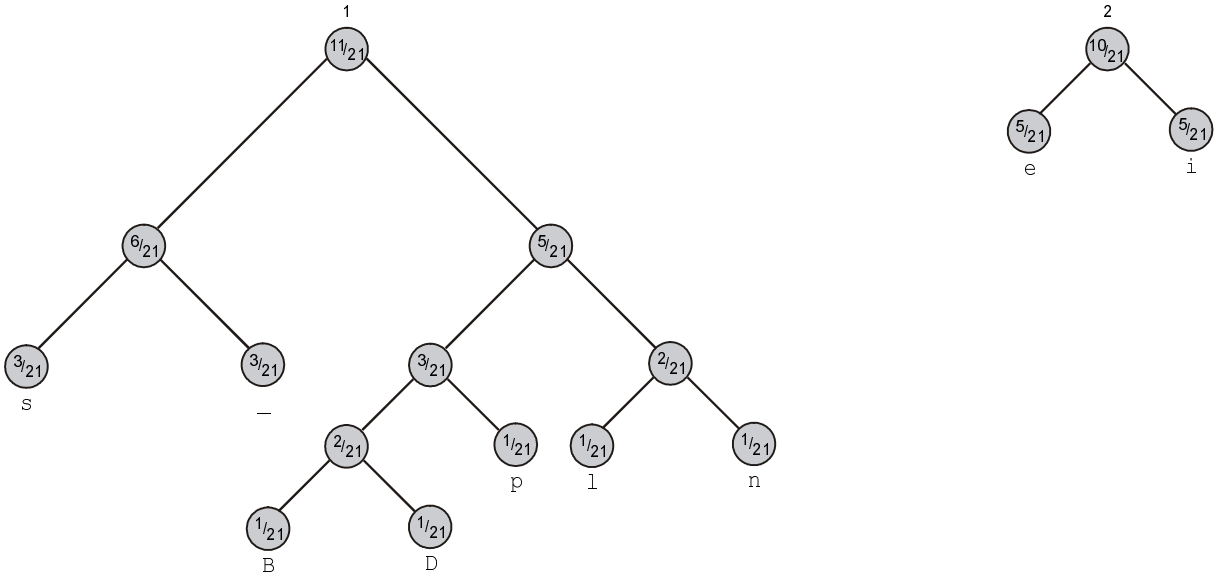
Ergebnis nach dem 5. Schritt



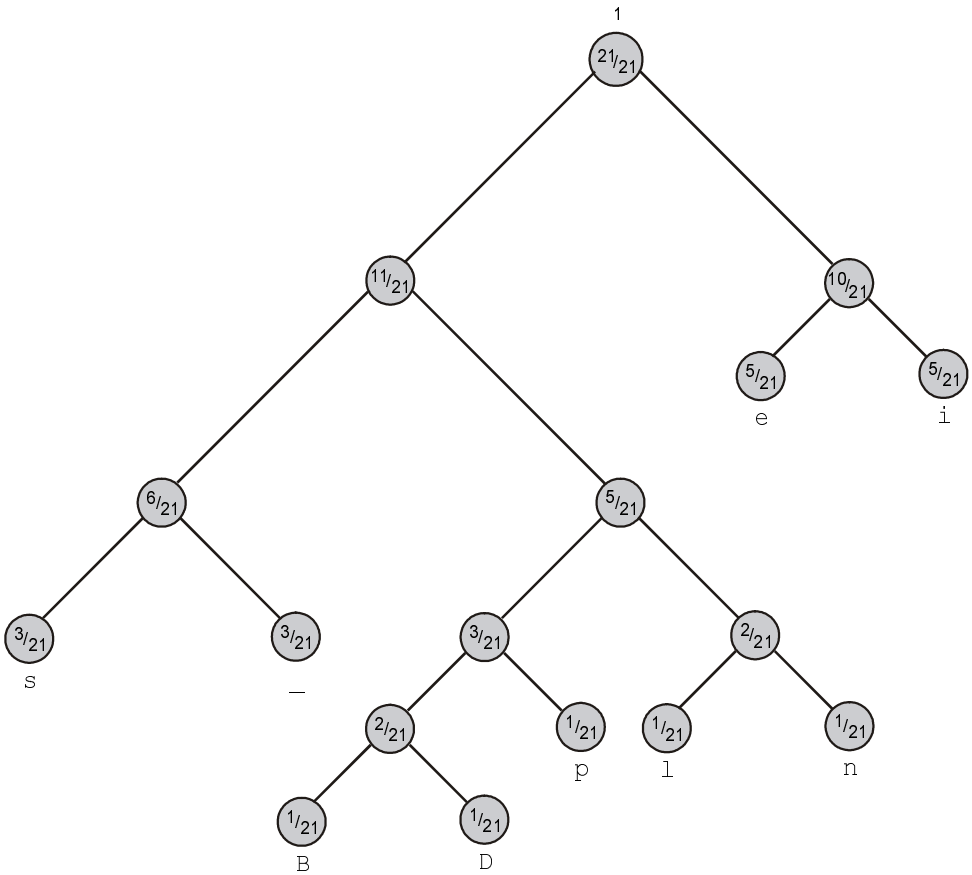
Ergebnis nach dem 6. Schritt

Huffman-Algorithmus – Beispiel (4)

Aufbau des Huffman-Baums (Fortsetzung)



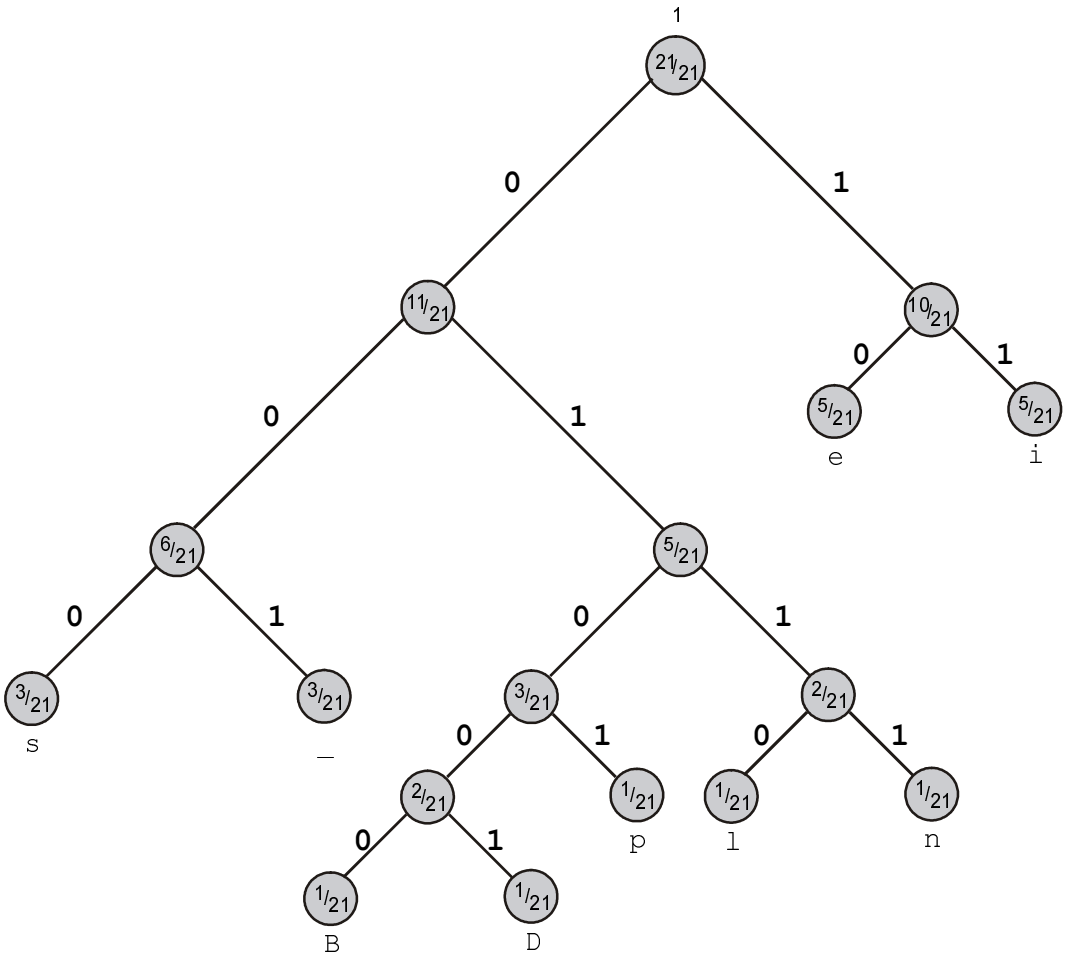
Ergebnis nach dem 7. Schritt



Ergebnis nach dem 8. Schritt

Huffman-Algorithmus – Beispiel (5)

Aufstellen der Binärcodetabelle



Zeichen	s	_	B	D	p	l	n	e	i
Binärcode	000	001	01000	01001	0101	0110	0111	10	11

Ergebnis:

01001	11	10	000	001	000	10	11	001	10	11	0111	001	01000	10	11	000	0101	11	10	0110
D	i	e	s	_	s	e	i	_	e	i	n	_	B	e	i	s	p	i	e	l

Huffman-Algorithmus (1)

1. Aufbau des Huffman-Baums

Gegeben:

Liste von Bäumen, die anfänglich aus n Knoten als Wurzel besteht

Algorithmus-Beschreibung:

Suche die beiden Bäume mit der geringsten Wahrscheinlichkeit und entferne diese aus der Liste. Die beiden gefundenen Bäume werden als linker und rechter Unterbaum mit Hilfe eines neuen Wurzelknotens zu einem neuen Baum zusammengefügt und in die Liste eingetragen.

Struktogramm:

<table>
<tr><td colspan="2">n = Anzahl der Listenelemente</td></tr>
<tr><td>Wiederhole (n-1)-mal</td><td>p1 = Listenelement mit der kleinsten Häufigkeit
Entferne p1 aus der Liste

p2 = Listenelement mit der kleinsten Häufigkeit
Entferne p2 aus der Liste

Erzeuge neuen Knoten p
Hänge p1 und p2 als Unterbäume an p an
Häufigkeit von p = Häufigkeit von p1 + Häufigkeit von p2

Füge p in Liste ein</td></tr>
</table>

Huffman-Algorithmus (2)

2. Aufstellen der Binärcodetabelle

- Beschriften Sie die Äste des Huffman-Baums: linke Äste werden mit 0, rechte mit 1 gekennzeichnet.

- Der Code eines Buchstabens ergibt sich unmittelbar durch seinen Pfad durch den Binärbaum.

3. Codierung

Weisen Sie jedem Muster seinen Code aus der Binärcodetabelle zu.

4. Decodierung

- Lesen Sie die Information bitweise ein und wandern Sie dabei entsprechend durch den binären Huffman-Baum.

- Wenn sie auf Blattebene angekommen sind ist dieses Zeichen decodiert.

- Führen Sie denselben Vorgang ab dem nächsten Bit wieder durch, bis keine Bits mehr vorhanden sind.

Laufkomprimierung (RLE-Komprimierung)

Idee:

Aufteilung der Bitliste in Null- und Einsfolgen. Danach Codierung der Länge einer jeden Folge (genannt: Lauf)

Codierung:

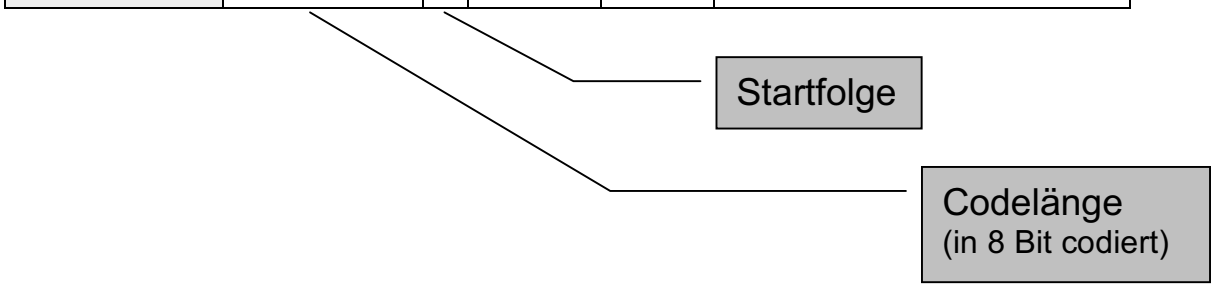
Codierung durch Codelängen fester Länge (k Bits)

Beispiel:

Lauflänge	Codierung	Anzahl der Nullwerte	Zahl
1	0001	0	1
2	0010	0	2
⋮	⋮	⋮	⋮
15	1111	0	$2^4 - 1$
16	0000 0001	1	1
17	0000 0010	1	2
⋮	⋮	⋮	⋮

Beispielscodierung (k=4)

Bitliste			11111	00	1111111111111111
Runlänge			5	2	17
RLE-Code	00000100	1	0101	0010	0000 0010



Laufkomprimierung (2)

Aufstellen der Codierungstafel:

Maximalwert = 2^k-1

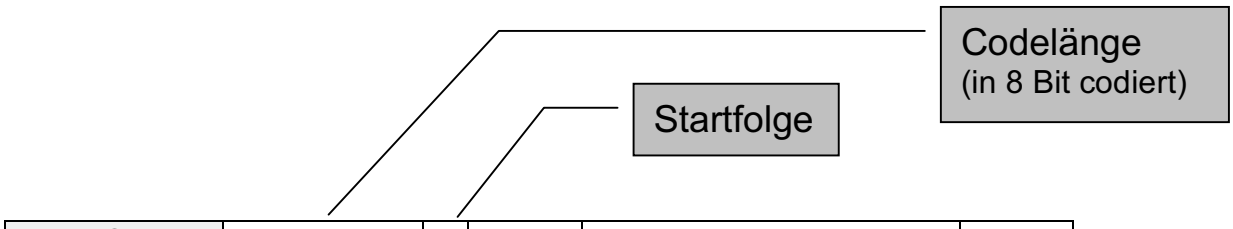
AnzahlNullwerte = (Lauflänge-1) DIV Maximalwert

Zahl = ((Lauflänge-1) MOD Maximalwert)+1

Decodierung:

Runlänge = AnzahlNullwerte * Maximalwert + Zahl

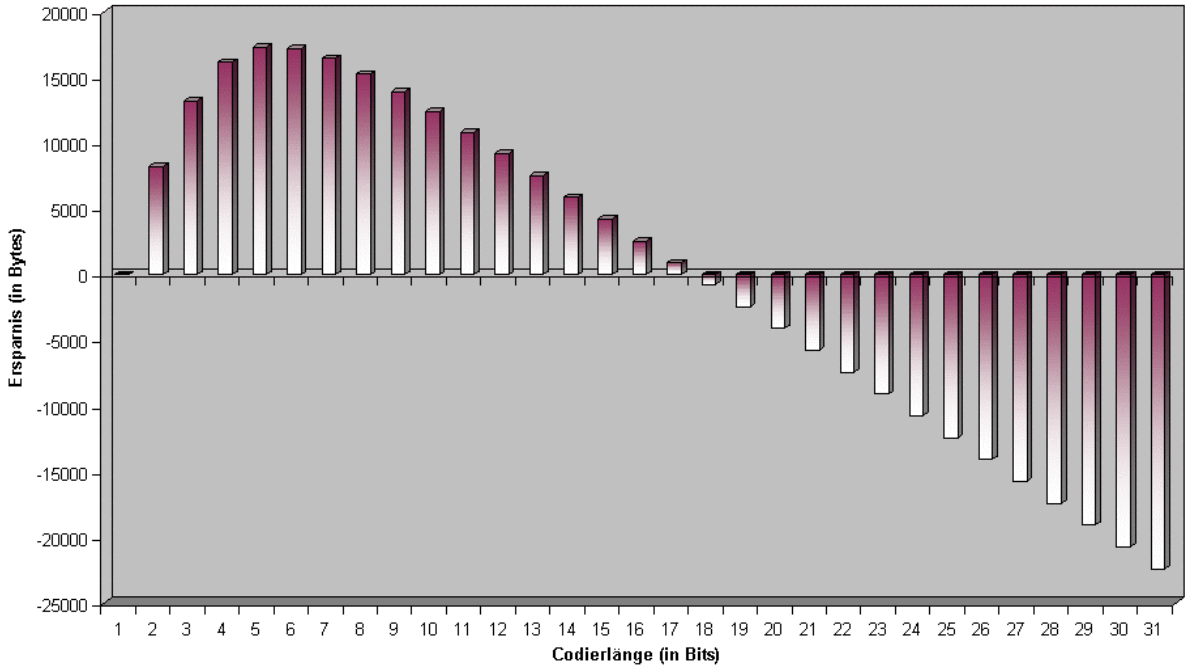
Beispiel :



RLE-Code	00000100	0	0010	0000 0001	0001
Runlänge			2	16	1
Bitliste			00	1111111111111111	0

Laufkomprimierung (3) - Beispiel

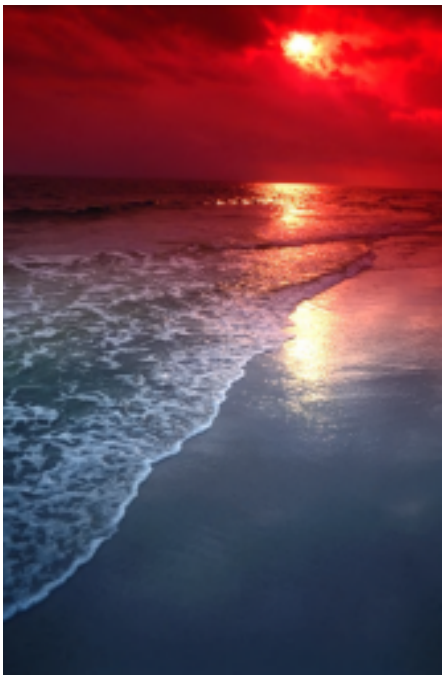
Ersparnis (je nach gewählter Codierlänge):



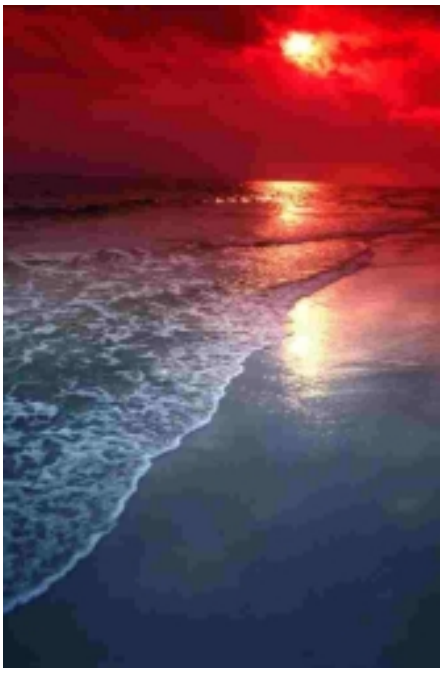
Beachten Sie :

Bei ungeeigneter Wahl der Codierlänge kann die „komprimierte“ Datei sogar an Umfang zunehmen.

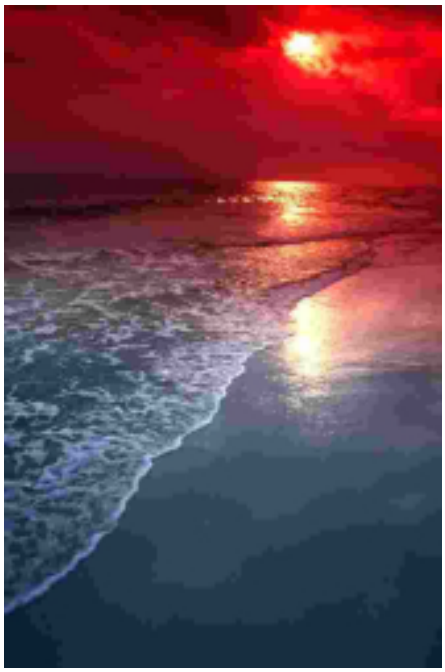
Komprimierung von Bildern - Beispiel



2016 KB (Bitmap)



340 KB (JPEG)



11 KB (JPEG)

Bildkomprimierung durch Blockbildung (1)

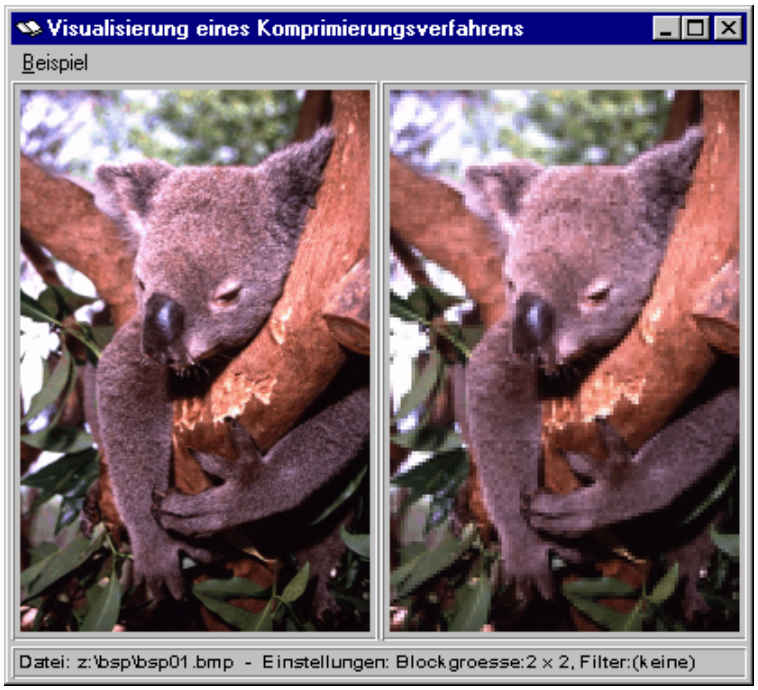


Bild 1.1: Komprimierung eines Farbfotos (ohne Filterfunktionen)

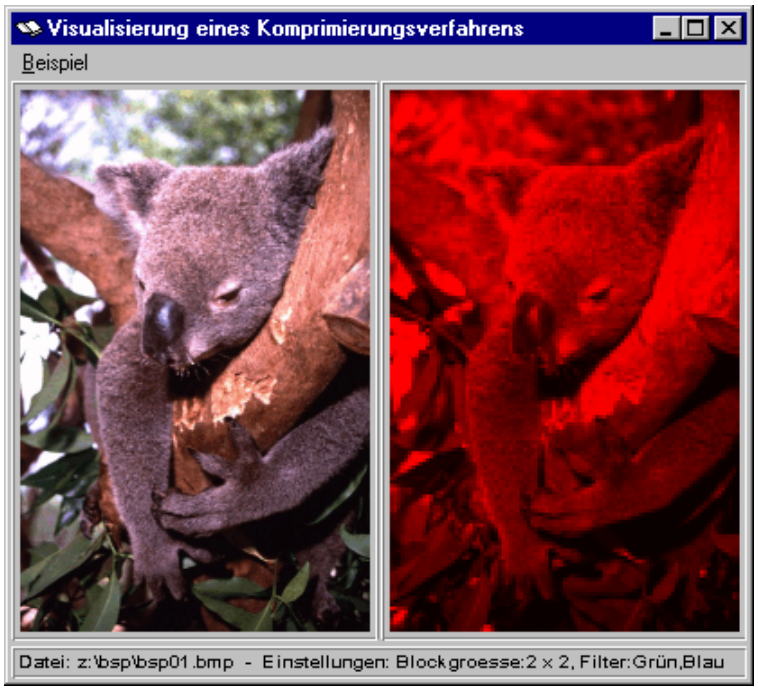
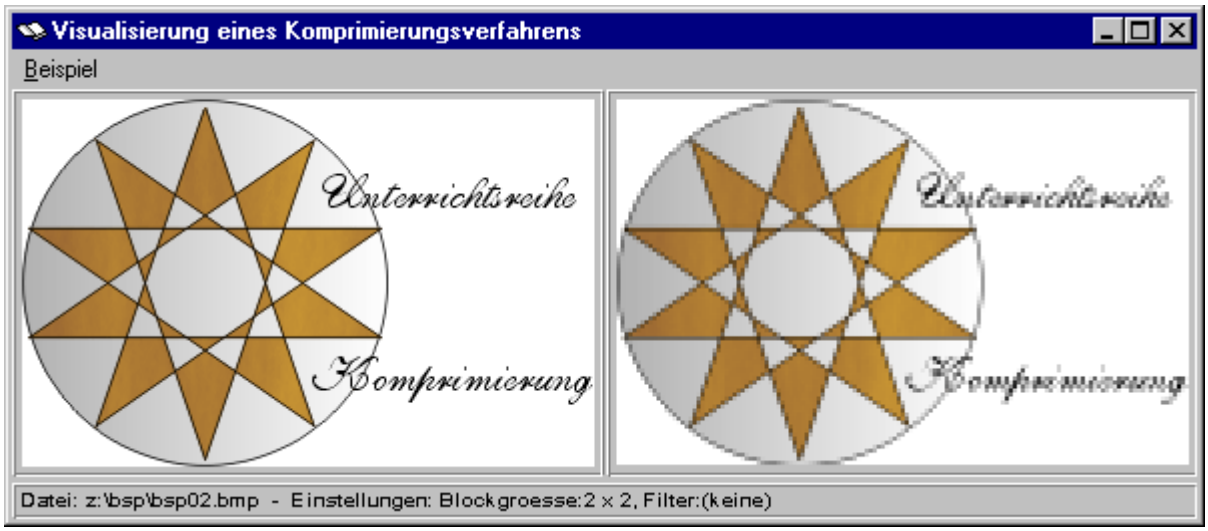
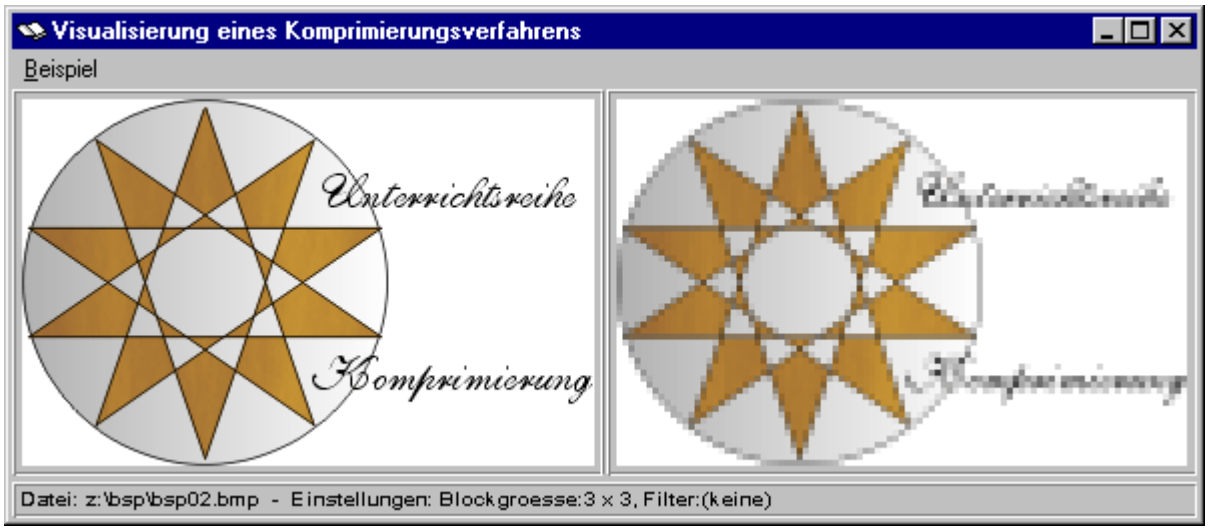
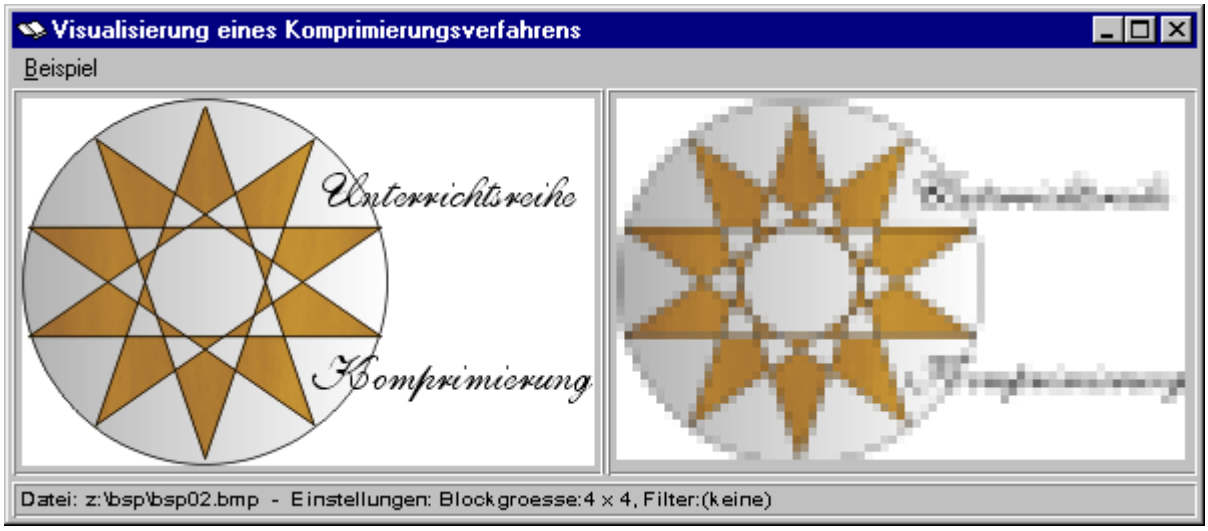


Bild 1.2: Komprimierung eines Farbfotos (mit Grün- und Blaufilterung)

Bildkomprimierung durch Blockbildung (2)







Einordnung der Komprimierungsverfahren

- **verlustfreie Komprimierung**

  - **Merkmal :**
    es ist möglich die ursprünglichen Informationen aus den komprimierten Daten vollständig wiederherzustellen

  - **Beispiele :**
    Huffman-Komprimierung, RLE-Komprimierung

- **verlustbehaftete Komprimierung**

  - **Merkmal :**
    Bei der Rekonstruktion erhält man Informationen, die den ursprünglichen Informationen zwar ähnlich sind, aber nicht exakt mit diesen übereinstimmen.

  - **Beispiele :**
    Blockbildungsverfahren, Filterverfahren

Arbeitsblatt 1

Aufgabe 1.1 : Komprimierung

1. Komprimieren Sie folgende Wörter mit dem Huffman-Algorithmus:

   ```
   Beispiel
   AAAAAAAAAABCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBBBBBC
   ```

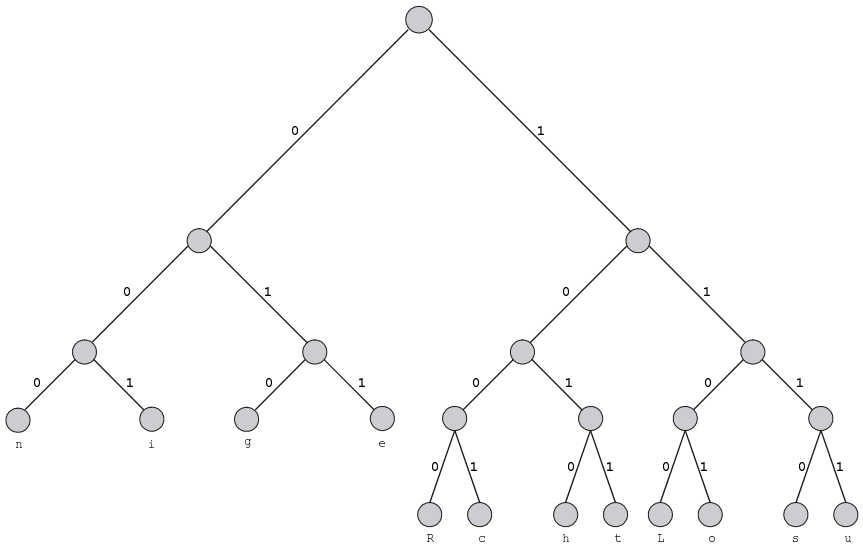
   Welche Komprimierungsrate wird erreicht ? Berücksichtigen Sie hierbei nicht den für die Speicherung des Huffman-Baumes benötigten Speicherplatz.
2. Welche Merkmale muß ein Text aufweisen, damit er besonders effizient bzw. ineffizient nach dem vorgestellten Huffman-Algorithmus komprimiert wird ?

Aufgabe 1.2 : Dekomprimierung

1. Gegeben sei folgende Binärdarstellung einer komprimierten Textpassage :

   ```
   1000001100110101011001010011110011010111101111000010
   ```

   Ermitteln Sie, falls möglich, mit Hilfe des folgenden Huffman-Baumes den Orginaltext :



2. Ist der Orginaltext anhand der gegebenen Informationen eindeutig rekonstruierbar ? Begründen Sie dies.

Musterlösung zu Arbeitsblatt 1

Lösung zu Aufgabe 1.1

1.1.1.1. Text: „Beispiel“

Initialisierung

```
[1]
     (1) B
[2]
     (2) e
[3]
     (2) i
[4]
     (1) s
[5]
     (1) p
[6]
     (1) l
```

1. Rekursionsschritt

```
[1]
     (2) e
[2]
     (2) i
[3]
     (1) p
[4]
     (1) l
[5]
     (2)
        (1) B
        (1) s
```

2. Rekursionsschritt

```
[1]
     (2) e
[2]
     (2) i
[3]
     (2)
        (1) B
        (1) s
[4]
     (2)
        (1) p
        (1) l
```

3. Rekursionsschritt

```
[1]
     (2)
        (1) B
        (1) s
[2]
     (2)
        (1) p
        (1) l
[3]
     (4)
        (2) e
        (2) i
```

4. Rekursionsschritt

```
[1]
     (4)
        (2) e
        (2) i
[2]
     (4)
        (2)
           (1) B
           (1) s
        (2)
           (1) p
           (1) l
```

5. Rekursionsschritt

```
[1]
     (8)
        (4)
           (2) e
           (2) i
        (4)
           (2)
              (1) B
              (1) s
           (2)
              (1) p
              (1) l
```

Huffman-Baum

```
(8/8)
   0: (4/8)
      0: (2/8) e
      1: (2/8) i
   1: (4/8)
      0: (2/8)
         0: (1/8) B
         1: (1/8) s
      1: (2/8)
         0: (1/8) p
         1: (1/8) l
```

Codetabelle

Muster	Binaercode
e	00
i	01
B	100
s	101
p	110
l	111

Die komprimierte Textpassage besitzt somit folgende Binärdarstellung:

```
10000011011100100111
```

Es ergibt sich ein Komprimierungsverhältnis von ca. 1:3.

1.1.1.2. Text: „AAAAAAAAAABCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBBBBBC“

Initialisierung

```
[1]
     (10) A
[2]
     (6) B
[3]
     (30) C
```

1. Rekursionsschritt

```
[1]
     (30) C
[2]
     (16)
        (6) B
        (10) A
```

2. Rekursionsschritt

```
[1]
     (46)
        (16)
           (6) B
           (10) A
        (30) C
```

Huffman-Baum

```
(46/46)
   0: (16/46)
      0: (6/46) B
      1: (10/46) A
   1: (30/46) C
```

Codetabelle

Muster	Binaercode
B	00
A	01
C	1

Die komprimierte Textpassage besitzt somit folgende Binärdarstellung:

```
01010101010101010100111111111111111111111111111000000000001
```

Es ergibt sich ein Komprimierungsverhältnis von ca. 1:6.

1.1.2. Effizienz / Ineffizenz

Besonders effizient ist die Komprimierung eines Textes, in dem ein großer Unterschied in der Anzahl des Auftretens einzelner Zeichen besteht. Ineffizient wird die Huffman-Komprimierung bei Texten, die eine große Anzahl unterschiedlicher Zeichen besitzen die wiederum alle in etwa der gleichen Häufigkeit auftreten.

Lösung zu Aufgabe 1.2

1.2.1. Der Orginaltext lautet: „RichtigeLoesung“

1.2.2. Da die Codes der – durch den Huffman-Baum implizit folgenden - Binärcodetabelle präfixfrei sind, ist die eindeutige Rekonstruierbarkeit gewährleistet.

Arbeitsblatt 2

Aufgabe 2.1 : Programmierung

1. Erstellen Sie eine Prozedur

   ```
   ErzeugeHuffmanBaum()
   ```

   die zu einem beliebigen Text einen zugehörigen Huffman-Baum erzeugt.

2. Erstellen Sie eine Prozedur

   ```
   KomprimiereText()
   ```

   die den Orginaltext gemäß Binärcodetabelle komprimiert.

Anbei erhalten Sie einen gesonderten Text mit Lösungshinweisen.

Aufgabe 2.2 : Erweiterbarkeit

Sehen Sie eine Möglichkeit den Huffman-Algorithmus derart zu modifizieren, dass bei längeren Texten (z.B. einem ganzen Romantext) noch bessere Komprimierungsraten erreicht werden ? Begründen Sie dies.

Aufgabe 2.3 : Zusätzliche Komprimierungsmöglichkeiten

Überlegen Sie sich anhand der Beispiele aus Aufgabe 2.1.1, ob bereits komprimierte Texte (gegebenenfalls mit einem anderen Verfahren) weiter komprimierbar sind.

Lösungshinweise zu Aufgabe 2.1.

Benutzen Sie zur Lösung folgendes Programmfragment:

```
PROGRAM HuffmanAlgorithmus;

USES
  Huffman;

PROCEDURE ErzeugeHuffmanBaum(VAR ADaten:THuffman);
VAR
  Text                  : String;
  AList                 : PListe;
BEGIN

  NeueListe(AList);
  Text := ADaten.Orginaltext;

  { ... }

  IF AList<>NIL THEN ADaten.BaumWurzel := AList^.Knoten;
  AList := NIL;

END; { ErzeugeHuffmanBaum (P) }

PROCEDURE KomprimiereText(VAR AHuffman:THuffman);
VAR
  Text                  : String;
BEGIN
  Text := AHuffman.OrginalText;
  AHuffman.TextKomprimiert:='...';

  { ... }

END; { KomprimiereText (P) }

VAR
  Beispiel              : THuffman;
  Dateiname             : String;
BEGIN

  { Eingaben }
  writeln('HUFFMAN-Komprimierung');
  writeln;
  writeln('Zu komprimierender Text: ');
  readln(Beispiel.OrginalText);
  writeln('Name der Ausgabedatei :');
  readln(Dateiname);
  Assign(Output,Dateiname);
  Rewrite(Output);

  ErzeugeHuffmanBaum(Beispiel);

  AusgabeHuffmanBaum(Beispiel);
  ErmittleBinaercodes(Beispiel);
  AusgabeHuffmanCodes(Beispiel);

  KomprimiereText(Beispiel);

  WITH Beispiel DO BEGIN
    writeln('Orginaltext (1 Zeichen = 1 Byte) ');
    writeln(Orginaltext);
    writeln;
    writeln('Text komprimiert (8 Ziffern = 1 Byte) :');
    writeln(TextKomprimiert);
    writeln;
    write('Es ergibt sich ein Komprimierungsverhaeltnis von 1 : ');
    writeln(Length(Orginaltext)/(Length(TextKomprimiert)/8):2:2);
  END; { WITH }

  { Aufraeumen... }
  Close(Output);
  GebeSpeicherFrei(Beispiel);

END.
```

Sie finden diese Datei unter „v_huf.pas“. Die verwendete Unit „Huffman“ stellt Ihnen die benötigten Datenstrukturen und Hilfsroutinen zur Verfügung.

Die Unit „Huffman“

Der zu komprimierende Text, der Huffman-Baum, die Binärcodetabelle und der komprimierte Text werden in der Datenstruktur THuffman abgelegt, die wie folgt deklariert ist:

```
THuffman              = RECORD
                          OrginalText          : String;
                          TextKomprimiert      : String;
                          BaumWurzel           : PKnoten;
                          Binaercodetabelle    : PCodierung;
                        END;
```

Den vorgegebenen Text können Sie mit der Funktion

```
FUNCTION NaechstesMuster(VAR AText:String;VAR AMuster:String;
  VAR AAnzahl:Longint) : BOOLEAN;
```

scannen, die die Anzahl des ersten Zeichens der in *AText* übergebenen Zeichenkette in derselbigen ermittelt, das Zeichen in *AMuster* und die Anzahl in *AAnzahl* speichert und anschließend alle Vorkommen dieses Zeichens aus der Zeichenkette entfernt. Ergreifen Sie also bitte geeignete Maßnahmen um keine relevanten Daten (z.B. den Orginaltext) zu verfälschen. Mit

```
PROCEDURE ErweitereListe(VAR AList:PListe;AMuster:String;
AAnzahl:Longint);
```

können Sie dann die verwendete Liste *AList* erweitern. Die Anzahl der Listenelemente ermitteln Sie mit

```
FUNCTION ElementeInDerListe(VAR AList:PListe):Longint;
```

Es wird nur die Baumwurzel in der Liste gespeichert. Daher besteht ein Listenelement entweder aus einem elementaren Knoten oder einem Baum. Einen neuen Knoten erzeugen Sie mit der Funktion

```
FUNCTION NeuerKnoten : PKnoten;
```

die einen Zeiger auf

```
TKnoten               = RECORD
                          linkerSohn           : PKnoten;
                          rechterSohn          : PKnoten;
                          Muster               : String;
                          relativeHaeufigkeit  : Longint;
                        END;
```

Die Unit „Huffman“ (Fortsetzung)

zurückliefert. Einen Knoten können Sie mit der Anweisung

```
PROCEDURE SetzeKnotenInListe(AKnoten:PKnoten;VAR AList:PListe);
```

in die Liste aufnehmen und das Listenelement an der Position *APos* mit der Prozedur

```
PROCEDURE VerschiebeInKnoten(VAR AList:PListe;APos:Word;VAR
AKnoten:PKnoten);
```

in einen Knoten verschieben. Die Positionsnummer des kleinsten Listenelementes ermitteln Sie mit

```
FUNCTION NummerDesKleinstenElementes(AList:PListe):Word;
```

Ist der Huffman-Baum vollständig generiert, so sollte sein Zeiger in der verwendeten Datenstruktur gespeichert werden. Durch Aufruf von `ErmittleBinaercodes` kann dann die Binärcodetabelle erzeugt werden die dann mit der Funktion

```
FUNCTION ZugehoerigerBinaercode(VAR AHuffman:THuffman;
Muster:String):String;
```

abgefragt werden kann.Verwenden Sie die Prozedur

```
PROCEDURE AusgabeListe(Titel:String;VAR AList:PListe);
```

als Ausgabe-Routine.

Musterlösung zu Arbeitsblatt 2

Lösung zu Aufgabe 2.1

```
PROCEDURE ErzeugeHuffmanBaum(VAR ADaten:THuffman);
VAR
  Text,Muster           : String;
  AList                 : PListe;
  S                     : String;
  Knoten                : PKnoten;
  Anzahl,n              : Longint;
  Elemente,p1,p2        : Longint;
BEGIN

{ ////////////////////////////////////////////////////////////////
  / 1. Schritt: Initialisierung
  //////////////////////////////////////////////////////////////// }

  NeueListe(AList);
  Text := ADaten.Orginaltext;
  WHILE NaechstesMuster(Text,Muster,Anzahl) DO
    ErweitereListe(AList,Muster,Anzahl);
  AusgabeListe('1. Schritt: Initialisierung',AList);

{ ////////////////////////////////////////////////////////////////
  / 2. - Elemente. Schritt: Rekursion
  //////////////////////////////////////////////////////////////// }

  Elemente := ElementeInDerListe(AList);
  FOR n := 2 TO Elemente DO BEGIN
    Knoten := NeuerKnoten;
    WITH Knoten^ DO BEGIN
      p1 := NummerDesKleinstenElementes(AList);
      VerschiebeInKnoten(AList,p1,linkerSohn);
      p2 := NummerDesKleinstenElementes(AList);
      VerschiebeInKnoten(AList,p2,rechterSohn);
      relativeHaeufigkeit := linkerSohn^.relativeHaeufigkeit +
                             rechterSohn^.relativeHaeufigkeit;
    END; { WITH }
    SetzeKnotenInListe(Knoten,AList);
    Str(n,s);AusgabeListe(s+'. Schritt',AList);
  END; { FOR }
  IF AList<>NIL THEN ADaten.BaumWurzel := AList^.Knoten;
  AList := NIL;
END; { ErzeugeHuffmanBaum (P) }


PROCEDURE KomprimiereText(VAR AHuffman:THuffman);
VAR
  Text,Muster,Code      : String;
BEGIN
  Text := AHuffman.OrginalText;
  AHuffman.TextKomprimiert:='';
  WHILE (Length(Text) > 0) DO BEGIN
    Muster := Copy(Text,1,1);
    Delete(Text,1,1);
    Code:= ZugehoerigerBinaercode(AHuffman,Muster);
    AHuffman.TextKomprimiert := AHuffman.TextKomprimiert + Code;
  END; { WHILE }
END; { KomprimiereText (P) }
```

Lösung zu Aufgabe 2.2

Eine Erweiterung würde darin bestehen nicht nur einzelne Zeichen sondern zusätzlich komplette Zeichenketten als Muster zuzulassen (vgl. LZ-Komprimierung). Häufig verwendete Wörter (wie z.B. Bindewörter) würden somit einer besseren Komprimierung unterzogen werden können. Der Huffman-Algorithmus ist somit lediglich bei der Initialisierung zu modifizieren. Allerdings bedeutet dies eine Steigerung des Aufwandes für die Erkennung der am häufigst vorkommenden Muster. Diese zusätzlichen Kosten fallen aber allerdings nur bei der Komprimierung an.

Lösung zu Aufgabe 2.3

Ausgehend von der gewonnenen Binärcodedarstellung (des durch den Huffman-Algorithmus komprimierten Textes) bieten sich Bitlistenkomprimierungsverfahren (z.B. Laufkomprimierung und Nullfolgenkomprimierung) in Abhängigkeit von der konkreten Codedarstellung an.

Arbeitsblatt 3

Aufgabe 3.1 : RLE-Komprimierung / -Dekomprimierung

1. RLE-komprimieren Sie folgende Bitlisten: (Wähle als Länge der Codiereinheit k den Wert 4)

   ```
   00001001111111111111111111111111111111
   ```

   ```
   10000000000000000000000000000000000100
   ```

2. Dekomprimieren Sie folgende (RLE-komprimierten) Bitlisten:

   ```
   000001000000000101111
   ```

   ```
   000001000000000101111
   ```

Aufgabe 3.2 : Programmierung

1. Erstellen Sie eine Prozedur

   ```
   RLE_Komprimierung()
   ```

   die (zu einer beliebige Datei und einer beliebig gewählten Codierlänge) eine entsprechende RLE-komprimierte Datei anlegt.

2. Erstellen Sie eine Prozedur

   ```
   RLE_Dekomprimierung()
   ```

   die eine RLE-komprimierte Datei dekomprimiert und das Ergebnis in einer neuen Datei speichert.

Anbei erhalten Sie einen gesonderten Text mit Lösungshinweisen.

Lösungshinweise zu Aufgabe 3.2

Benutzen Sie zur Lösung folgendes Programmfragment:

```
PROGRAM Bitlistenkomprimierung;

USES
  WinCrt,BitListe;

VAR
  Einsfolge             : Boolean;
  AnzahlBits            : Longint;
  AnzahlNullen          : Longint;
  CodeLaenge            : Byte;

PROCEDURE RLE_Komprimierung;
BEGIN
  writeln('RLE Komprimierung');writeln;
  IF DateienVorbereitet THEN BEGIN
    EingabeCodeLaenge(CodeLaenge);
    BestimmeMaxWert(CodeLaenge);

    { ... }

    DateienAbschliessen;
  END; { IF }
END; { RLE_Komprimierung (P) }

PROCEDURE RLE_Dekomprimierung;
VAR
  Wert                  : Longint;
BEGIN
  writeln('RLE Dekomprimierung');writeln;
  IF DateienVorbereitet THEN BEGIN

    { ... }

    DateienAbschliessen;
  END; { IF }
END; { RLE_Dekomprimierung (P) }

VAR
  Ende                  : Boolean;
BEGIN
  Ende:=False;
  REPEAT
    ClrScr;
    writeln('Auswahl : ');writeln;
    writeln('(1) RLE-Komprimierung');
    writeln('(2) RLE-Dekomprimierung');
    writeln('(0) Programmende');
    CASE readkey OF
      '1': RLE_Komprimierung;
      '2': RLE_Dekomprimierung;
      ELSE
         Ende := True;
    END; { CASE }
  UNTIL Ende;
  Done;
END.
```

Sie finden diese Datei unter „v_rle.pas“.

Die Unit „Bitliste“

Die verwendete Unit „BitListe“ stellt Ihnen die benötigten Hilfsroutinen zur Verfügung. Die Routinen operieren jeweils auf bereits geöffneten Dateien. Durch Verwendung des angegebenen Programmfragments geschieht die Verwaltung der Dateien automatisch, so dass Sie sich darum nicht kümmern müssen.

1. **Leseroutinen :**

   Die Funktion

   ```
   FUNCTION LeseBits(AEinsBit:Boolean):Longint;
   ```

   gibt Ihnen (ausgehend von der aktuellen Leseposition) die Länge des übergebenen Lauftyps an (1er-Folge: AEinsBit=True ; 0er-Folge: AEinsBit=False) und setzt die aktuelle Leseposition auf das nächste Bit nach diesem Lauf. Mit der Funktion

   ```
   FUNCTION LeseWert(Breite:Byte):Longint;
   ```

   wird die - im Parameter *Breite* (mit *1 ≤ Breite ≤ 31)* - übergebene Bitanzahl ausgelesen und der Dezimalwert der Dualzahl zurückgeliefert (Bitte beachten Sie: diese Funktion interpretiert die Dualzahl auf natürliche Art und Weise). Schließlich können Sie mit der Funktion

   ```
   FUNCTION WeitereBitsVorhanden:BOOLEAN;
   ```

   abfragen, ob noch zu lesende Bits vorhanden sind (Falls ja, liefert diese Funktion den Wert *True*).

2. **Schreibroutinen :**

   Mit der Prozedur

   ```
   PROCEDURE SchreibeBits(EinsBits:Boolean;Anzahl:Longint);
   ```

   schreiben Sie in die Ausgabedatei insgesamt *Anzahl*-mal das übergebenenes Bit und mit

   ```
   PROCEDURE SchreibeWert(Breite:Byte;Wert:Longint);
   ```

   den übergeben Wert als Dualzahl mit der angegebenen Länge (Beachten Sie: Anlog zur Funktion *LeseWert* findet auch hier nur die natürliche Interpretation statt).

Musterlösung zu Arbeitsblatt 3

Lösung zu Aufgabe 3.1

1. Komprimierung

Bitliste			0000	1	00	11111111111111111111111111111111
Runlänge			4	1	2	30
RLE-Code	00000100	0	0100	0001	0020	0000 1111

Bitliste			1	00000000000000000000000000000000	1	00
Runlänge			1	32	1	2
Ergebnis	00000100	1	0001	0000 0000 0010	0001	0020

2. Dekomprimierung

RLE-Code	00000100	0	0000 0010	1111
Runlänge			17	15
Bitliste			00000000000000000	111111111111111

RLE-Code	00000**01**0	0	00 00 00 10	11	11
Runlänge			10	3	3
Bitliste			0000000000	111	000

Lösung zu Aufgabe 3.2

```
PROCEDURE RLE_Komprimierung;
BEGIN
  writeln('RLE Komprimierung');writeln;
  IF DateienVorbereitet THEN BEGIN
    EingabeCodeLaenge(CodeLaenge);
    BestimmeMaxWert(CodeLaenge);

    { Codelaenge und Startfolge festhalten }
    SchreibeWert(8,CodeLaenge);
    IF StartBit = '1' THEN BEGIN
      SchreibeWert(1,1);Einsfolge := True;
    END ELSE BEGIN
      SchreibeWert(0,1);Einsfolge := False;
    END; { IF }

    { Alternierende Bitfolgen analysieren }
    WHILE WeitereBitsVorhanden DO BEGIN

       AnzahlBits := LeseBits(Einsfolge); { Anzahl ermitteln }
       IF AnzahlBits > MaxWert THEN BEGIN { Lauflänge größer als (2^k)-1 ! }
        AnzahlNullen := (AnzahlBits - 1) DIV MaxWert;
        SchreibeBits(False,AnzahlNullen * CodeLaenge);
        AnzahlBits := ((AnzahlBits-1) MOD MaxWert)+1;
      END; { IF }
      SchreibeWert(CodeLaenge,AnzahlBits);
      { Andersgeartete Folge betrachten }
      Einsfolge := NOT Einsfolge;

    END; { WHILE }
    DateienAbschliessen;
  END; { IF }
END; { RLE_Komprimierung (P) }
```

```
PROCEDURE RLE_Dekomprimierung;
VAR
  Wert                  : Longint;
BEGIN
  writeln('RLE Dekomprimierung');writeln;
  IF DateienVorbereitet THEN BEGIN

    { 1. Byte enthält die Codierungslänge, 1.Bit des 2. Bytes die Startfolge }
    CodeLaenge := LeseWert(8);
    BestimmeMaxWert(CodeLaenge);
    Wert:=LeseWert(1);
    IF Wert=1 THEN
      Einsfolge := True
    ELSE
      Einsfolge := False;
    WHILE (Wert <> -1) DO BEGIN
      AnzahlNullen := 0;
      Wert := LeseWert(CodeLaenge);
      WHILE (Wert = 0) DO BEGIN
        Inc(AnzahlNullen);
        Wert := LeseWert(CodeLaenge);
      END; { WHILE }
      IF (Wert <> -1) THEN BEGIN
        AnzahlBits := AnzahlNullen * MaxWert + Wert;
        SchreibeBits(Einsfolge,AnzahlBits);
      END; { IF }

      { Andersgeartete Folge betrachten }
      Einsfolge := NOT Einsfolge;
    END; { WHILE }
    DateienAbschliessen;
  END; { IF }
END; { RLE_Dekomprimierung (P) }
```

Arbeitsblatt 4

Aufgabe 4.1 : Programmierung

1. Implementieren Sie das vorgestellte Verfahren zur Bildkomprimierung für eine Blockgröße von 2 x 2.

2. Verallgemeinern Sie Ihr Programm durch Verwendung beliebiger Parameter: Die zu verwendeten Werte ergeben sich durch Auslesen der Variablen `Cfg.Blockgroesse` (Größe des zu verwendeten Blocks) und `Cfg.RFilter`, `Cfg.GFilter` und `Cfg.RFilter` (True, falls die entsprechenden Farbanteile ausgeblendet werden sollen). Diese Einstellungen können im fertigen Programm unter dem Menüpunkt `Beispiel|Einstellungen` bequem vorgenommen werden.

Anbei erhalten Sie einen gesonderten Text mit Lösungshinweisen.

Aufgabe 4.2 : Anwendung

Öffnen Sie verschiedene Bilder und diskutieren Sie untereinander die unterschiedlichen Ergebnisse. Für welche Bildarten ist dieses Verfahren Ihres Erachtens geeignet bzw. ungeeignet ? Begründen Sie dies. Diskutieren Sie auch über generelle Vor- und Nachteile einzelner Komprimierungsverfahren.

Lösungshinweise zu Aufgabe 4.1

Benutzen Sie zur Lösung folgendes Programmfragment:

```
PROGRAM v_Bitmap;

USES
  OWindows , GFenster;

TYPE
  PBeispiel             = ^TBeispiel;
  TBeispiel             = OBJECT(TGrafikFenster)
    PROCEDURE Komprimiere;
      Virtual;
  END; { TBeispiel (O) }

  TApplikation          = OBJECT(TApplication)
    PROCEDURE InitMainWindow;
      Virtual;
  END; { TApplikation (O) }

PROCEDURE TBeispiel.Komprimiere;
VAR
  R,G,B,ResR,ResG,ResB  : Byte;
  SummeR,SummeG,SummeB  : Longint;
  XC,YC,BX,BY,X,Y       : Integer;
  SchritteX,SchritteY   : Integer;
  Breite,Hoehe          : Integer;
BEGIN

  Cfg.Statustext := 'Beipielskomprimierung';

  { ... }

END; { TBeispiel.Komprimiere (P) }

PROCEDURE TApplikation.InitMainWindow;
BEGIN
  MainWindow := New(PBeispiel, Init(NIL,
    'Visualisierung eines Komprimierungsverfahrens'));
END; { TApplikation.InitMainWindow (P) }

VAR
  App                   : TApplikation;
BEGIN
  App.Init('Unterrichtsreihe Komprimierungsverfahren');
  App.Run;
  App.Done;
END.
```

Bemerkungen:

Mit der Prozedur

```
PROCEDURE ErmittlePixelFarbe(X,Y:Integer;VAR R,G,B:Byte);
```

können Sie die RGB-Farbanteile (der Orginalbilds) an der Bildposition (X;Y) lesen und entsprechend mit

```
PROCEDURE SetzePixelFarbe(X,Y:Integer;R,G,B:Byte);
```

die RGB-Farbanteile (des Zielbildes) setzen.

Musterlösung zu Arbeitsblatt 4

Lösung zu Aufgabe 4.1.1

```
PROCEDURE TBeispiel.Komprimiere;
VAR
  R,G,B,ResR,ResG,ResB  : Byte;
  SummeR,SummeG,SummeB  : Longint;
  XC,YC,BX,BY,X,Y       : Integer;
  SchritteX,SchritteY   : Integer;
  Breite,Hoehe          : Integer;
BEGIN

  Cfg.Statustext := 'Beipielskomprimierung';

  { Anzahl der Schritte bestimmen }
  SchritteX := BildBreite DIV 2;
  SchritteY := BildHoehe DIV 2;
  FOR XC := 0 TO SchritteX-1 DO BEGIN
    FOR YC:=0 TO SchritteY-1 DO BEGIN

      { Auslesepositon bestimmen }
      X := 2 * XC;
      Y := 2 * YC;

      { Farben auslesen und Summe der Farbwerte bestimmen }
      SummeR := 0; SummeG := 0; SummeB := 0;
      ResR := 0; ResG := 0; ResB := 0;
      FOR BX:=1 TO 2 DO BEGIN
        FOR BY:=1 TO 2 DO BEGIN
           ErmittlePixelFarbe(X+BX,Y+BY,ResR,ResG,ResB);
          SummeR := SummeR + ResR;
          SummeG := SummeG + ResG;
          SummeB := SummeB + ResB;
        END; { FOR }
      END; { FOR }

      { Mittelwert berechnen }
      ResR := Round(SummeR/4);
      ResG := Round(SummeG/4);
      ResB := Round(SummeB/4);

      { Farben setzen }
      FOR BX:=1 TO 2 DO FOR BY:=1 TO 2 DO
        SetzePixelFarbe(X+BX,Y+BY,ResR,ResG,ResB);

    END; { FOR }
  END; { FOR }

END; {  TBeispiel.Komprimiere (P) }
```

Lösung zu Aufgabe 4.1.2 :

Die notwendigen Änderungen / Erweiterungen gegenüber der Lösung von Aufgabe 4.1.2 sind deutlich kenntlich gemacht worden.

```
PROCEDURE TBeispiel.Komprimiere;
VAR
  R,G,B,ResR,ResG,ResB  : Byte;
  SummeR,SummeG,SummeB  : Longint;
  XC,YC,BX,BY,X,Y       : Integer;
  SchritteX,SchritteY   : Integer;
  Breite,Hoehe          : Integer;
BEGIN

  { Anzahl der Schritte bestimmen }
  SchritteX := BildBreite DIV Cfg.Blockgroesse;
  SchritteY := BildHoehe DIV Cfg.Blockgroesse;
```

```
  FOR XC := 0 TO SchritteX-1 DO BEGIN
    FOR YC:=0 TO SchritteY-1 DO BEGIN

      { Ausleseposition bestimmen }
      X := Cfg.Blockgroesse * XC;
      Y := Cfg.Blockgroesse * YC;

      { Farben auslesen und Summe der Farbwerte bestimmen }
      SummeR := 0; SummeG := 0; SummeB := 0;
      ResR := 0; ResG := 0; ResB := 0;
      FOR BX:=1 TO Cfg.Blockgroesse DO BEGIN
        FOR BY:=1 TO Cfg.Blockgroesse DO BEGIN
           ErmittlePixelFarbe(X+BX,Y+BY,ResR,ResG,ResB);
          SummeR := SummeR + ResR;
          SummeG := SummeG + ResG;
          SummeB := SummeB + ResB;
        END; { FOR }
      END; { FOR }

      { Mittelwert berechnen, Filter anwenden}
      ResR := Round(SummeR/(Cfg.Blockgroesse*Cfg.Blockgroesse));
      ResG := Round(SummeG/(Cfg.Blockgroesse*Cfg.Blockgroesse));
      ResB := Round(SummeB/(Cfg.Blockgroesse*Cfg.Blockgroesse));
      IF Cfg.RFilter THEN ResR := 0;
      IF Cfg.GFilter THEN ResG := 0;
      IF Cfg.BFilter THEN ResB := 0;

      { Farben setzen }
      FOR BX:=1 TO Cfg.Blockgroesse DO FOR BY:=1 TO Cfg.Blockgroesse DO
          SetzePixelFarbe(X+BX,Y+BY,ResR,ResG,ResB);

    END; { FOR }
  END; { FOR }

END; {  TBeispiel.Komprimiere (P) }
```

Lösung zu Aufgabe 4.2

Dieses Verfahren eignet sich nur für nicht speziell-strukturierte Bilder (wie beispielsweise Naturbilder), da dort im allgemeinen fließende Farbübergänge stattfinden.

Literatur

[HÄR] Theo Härder – *Ergänzende Unterlagen zur Vorlesung Datenstrukturen*
Universität Kaiserslautern, 1997

[HBR] Peter Hubwieser, Manfred Broy - *Grundlegende Konzepte von Informations- und Kommunikationssystemen für den Informatikunterricht.*

[L93] Ministerium für Bildung und Kultur - *Lehrplanentwurf Informatik*
Rheinland Pfalz, 1993

[SAY] Khalid Sayood – *Introduction to Data Compression*
University of Nebraska at Lincoln
Morgan Kaufmann Publishers, Inc., San Francisco, California

[VOR] Oliver Vornberger – *Vorlesung Multimedia*
Universität Osnabrück, 1997