

Programmierpraktikum

Sommersemester 2007 - 08.06.2007

Björn Wilmsmann, B.A.
Sprachwissenschaftliches Institut
Ruhr-Universität Bochum
wilmsmann@linguistics.rub.de

Heute

- Datenkompression
- Motivation
 - effiziente Übertragung und Speicherung
 - Speicherung linguistischer Daten
 - Suchindices
 - akustische Signale (z.B. Sprachsignale)

Datenkompression

- Aspekte
 - effizientes Komprimieren und Dekomprimieren
 - Kompressionsfaktor
 - Durchsuchbarkeit komprimierter Daten
 - Was ist ein elementares Zeichen? (Chars vs. Worte)

Fouriertransformation

- Transformation einer Funktion in ein Spektrum ihrer Frequenzkomponenten
- Wandlung von zeitlicher in spektrale Ansicht
- Fourierpolynom n -ten Grades ist einer Näherung / Abstraktion einer Funktion

LPC

- Linear Predictive Coding
- kodiert Daten (üblicherweise eine Kurve, die ein akustisches Signal beschreibt) mit Hilfe von Parametern
- nur diese Parameter werden übertragen
- bei Sprachsignalen: 2 Parameter (Polstellen) pro Formant

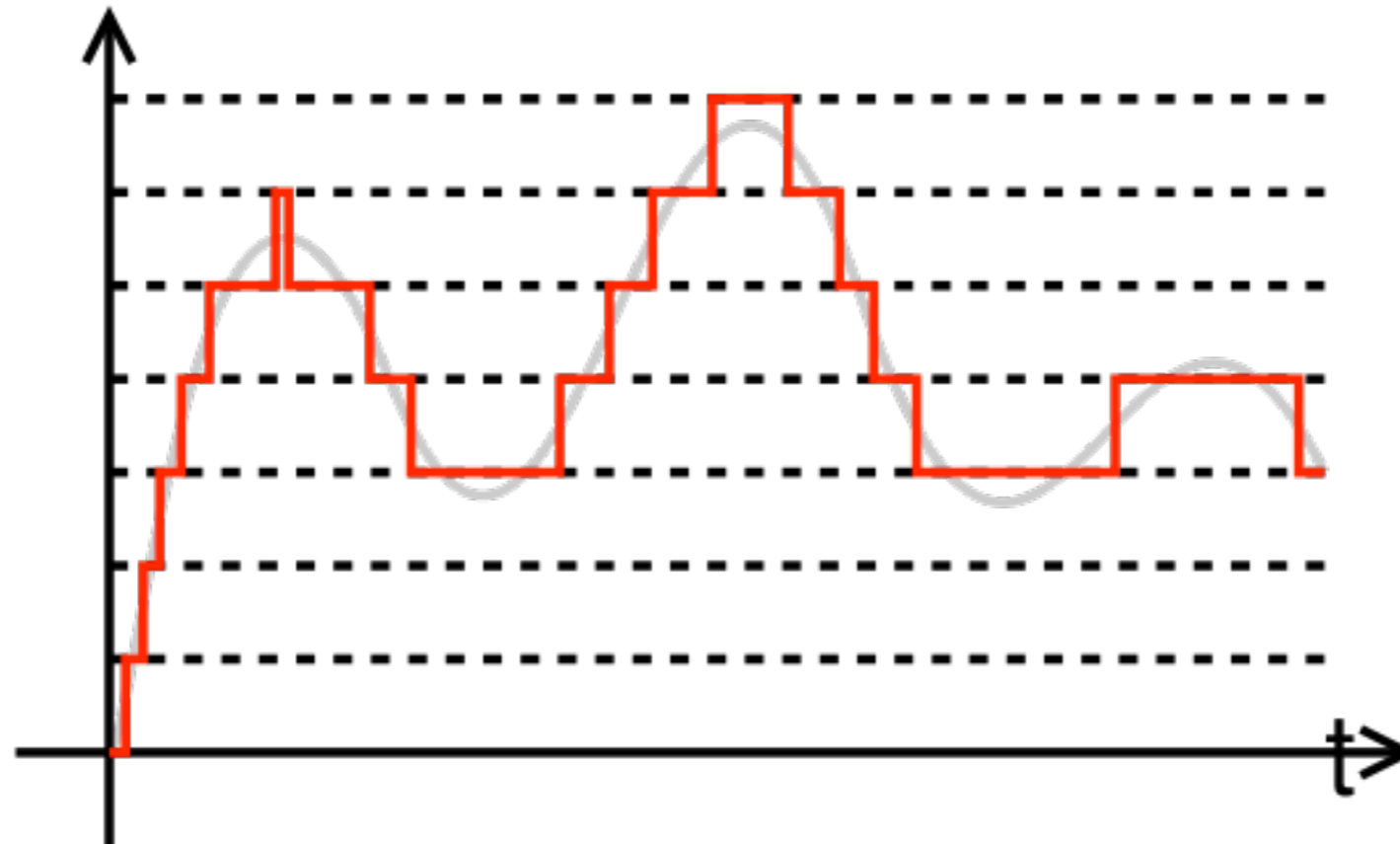
Quantisierung

- verlustbehaftete (lossy) Kompression
- Approximierung von kontinuierlichen Werten durch diskrete Werte
- Nutzung in der Signalverarbeitung zum Digitalisieren von Signalen

Skalare Quantisierung

- komprimiert anhand von skalaren Werten des Datensatzes
- einfachst mögliches Verfahren
- z.B. Quantisierung in der ISDN-Telefonie: A-law (Europa), μ -law (USA, Japan)

Skalare Quantisierung



Vektorquantisierung

- bedient sich eines Vektorraummodells zur Repräsentation von Daten
- fasst mehrere Werte im Datensatz zu einem Vektor zusammen
- nutzt das Konstrukt des Centroiden / Schwerpunktes
- LBG (Linde-Buzo-Gray) Algorithmus

Centroid / Schwerpunkt

- analog zum physikalischen Schwerpunkt
- bester Repräsentant einer Menge
- im Vektorraum: Centroid hat gleiche Entfernung zu allen Repräsentanten (= Vektoren) einer Menge

LBG-Algorithmus

- Initialzustand: Feste Anzahl zufällig oder per Heuristik gewählter Centroiden
- alle Vektoren werden ihrem nächsten Centroiden zugeordnet
- Centroiden werden entsprechend der zugeordneten Vektoren angepasst
- ...

Clustering

- starker Zusammenhang mit Vektorquantisierung
- hauptsächlich anderer Anwendungsbereich als Vektorquantisierung, die Algorithmen und Strukturen ähnlich sich stark
- k-means nahezu äquivalent zu LBG

Verlustfreie Datenkompression

- Methoden
 - statistische Kodierung
 - Kodierung über Wörterbücher (dictionaries)

Statistische Kodierung

- Schätzung der Wahrscheinlichkeiten für Symbole (Chars, Worte)
- Kodierung und Dekodierung anhand des daraus resultierenden statistischen Modells
- Claude Shannon: Ein Symbol der Wahrscheinlichkeit p sollte mit einem Code der Länge $\log_2(1/p)$ Bit kodiert werden.

Modellierung

- adaptiv
- statisch
- semi-statisch

Adaptive Modelle

- benötigen keine Vorinformationen
- single pass Kompression
- Nachteil: Dekompression muss am Anfang der Datei beginnen, ungeeignet für Information Retrieval, NLP, eher für generellen Einsatz

Statische Modelle

- Vorannahme: Durchschnittliche Verteilung der Eingabesymbole
- Nachteil: Over-Fitting, funktioniert nur gut bei ‚passenden‘ Daten

Semi-statische Modelle

- keine Vorannahme
- lernen im ersten Durchgang
- komprimieren im zweiten Durchgang
- Vorteil: Direkter Zugriff auf komprimierte Daten
- Nachteil: Zwei Durchgänge

Statistische Kodierung

- Huffman Coding (allgemein hauptsächlich zwischen 1950 bis 1970 eingesetzt)
- Arithmetische Kodierung (ab 1970)
- beide Kodierungen gibt es sowohl in statischer, als auch in semi-statischer, sowie adaptiver Ausprägung

Huffman-Coding

- Idee
 - weist jedem Type ein Bitmuster zu
 - je häufiger ein Type ist, desto weniger Bits werden für dessen Kodierung verwandt
 - höhere Frequenz bedeutet geringeren Beitrag des Types zur Entropie (Information) des Modells

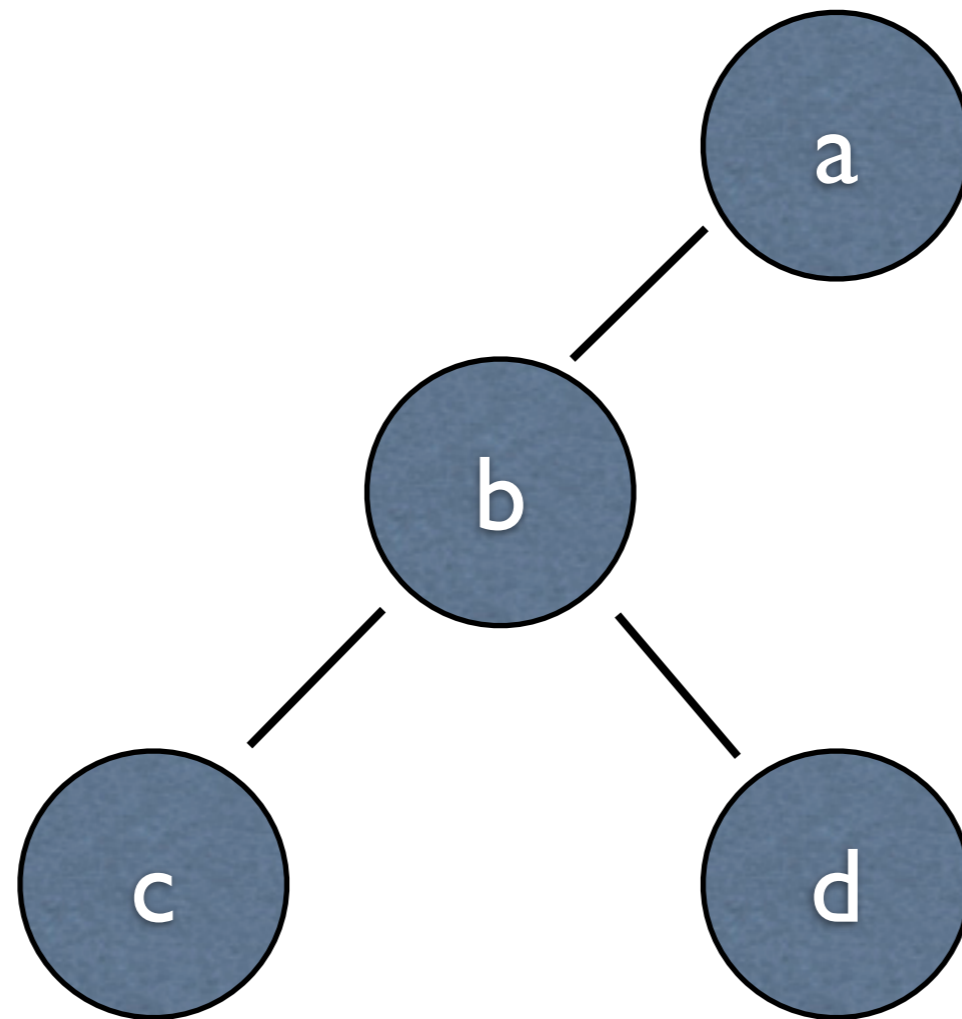
Huffman-Coding

- Eigenschaften
 - Suchen in komprimierten Daten möglich
 - wichtig für Information Retrieval
Anwendungen
 - dazu wird die Anfrage ebenfalls komprimiert und dann auf die komprimierten Token gematcht

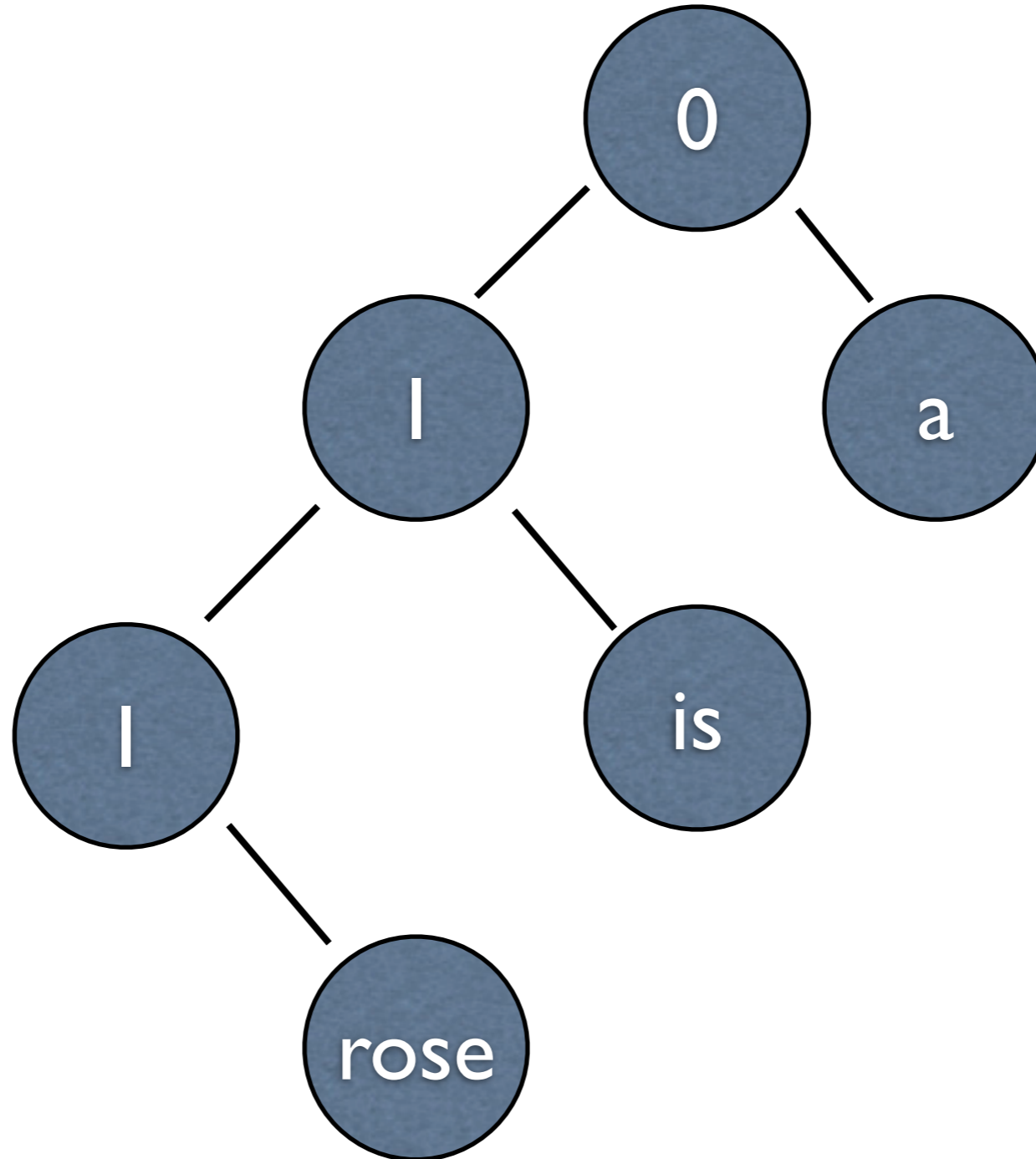
Huffman-Coding

- Eigenschaften
- Char-basiertes Verfahren erreicht Kompression von 2 bis 3 Bit pro Char
- Wort-basiertes Verfahren erreicht je nach Sprachmodell Kompression von 5 bis 6 Bit pro Char (Entropie des Modells!)

Trie allgemein



Bit-Trie



Huffman-Coding

- Ablauf Dekodierung
 - komprimierte Datei wird durchlaufen
 - gleichzeitig wird ein Trie mit Bitmustern durchlaufen
 - wenn Blatt erreicht wird, wird das Wort ausgegeben und Trie-Durchlauf neu gestartet

Huffman-Coding

- Ablauf Kodierung
 - Initial: Merge-Find Set mit ein-elementigen Bäumen, deren Wahrscheinlichkeiten sich auf 1 summieren
 - Knoten mit den niedrigsten Wahrscheinlichkeiten werden Kinder eines neuen Knoten

Huffman-Coding

- Ablauf Kodierung
 - dieser Vorgang wird iterativ für alle Knoten wiederholt, die noch keine Kinder sind
 - Ergebnis: Knoten mit der höchsten Wahrscheinlichkeit oben (benötigen daher weniger Code-Bits)

Huffman-Coding

- Anmerkungen
 - große Anzahl an verschiedenen Bäumen für eine Verteilung möglich
 - kanonischer Baum bevorzugt
 - kanonischer Huffman-Tree: Höhe des linken Teilbaums eines Knotens niemals geringer als die des rechten Teilbaums

Huffman-Coding

- Anmerkungen
 - existiert auch als Byte-Variante
 - Folge: Grad von bis zu 256 anstelle von 2 bei Bit-Variante

Algorithm::Huffman

- `(sudo) perl -MCPAN -e shell`
- `install Algorithm::Huffman`

Algorithm::Huffman

```
use Algorithm::Huffman;

my %char_counting = map {$_ => int rand(100)} ('a' .. 'z', 'A' .. 'Z');

my $huff = Algorithm::Huffman->new(\%char_counting);
my $encode_hash = $huff->encode_hash;
my $decode_hash = $huff->decode_hash;

my $encode_of_hello = $huff->encode_bitstring("Hello");

print "Look at the encoding bitstring of 'Hello': $encode_of_hello\n";
print "The decoding of $encode_of_hello is '", $huff->decode_bitstring($encode_of_hello), "'";
```

Arithmetische Kodierung

- Idee
- berechnet Codes für die einzelnen Symbole inkrementell
- Code für das betrachtete Symbol wird anhand der vorhergegangenen Berechnungen abgeleitet
- Kodierung reelle Zahlen > 0 und < 1

Arithmetische Kodierung

- Eigenschaften
- ermöglicht höhere Kompressionsraten als Huffman-Coding
- Suchen in komprimierten Daten nicht möglich, da Symbole in der Mitte nicht zugreifbar, da von Vorzustand abhängig
- langsamer als Huffman-Coding

Arithmetische Kodierung

- Eigenschaften
 - durchschnittliche Codelänge sehr nah der Entropie des jeweiligen Modells
 - keine Abspeicherung des Code-Baumes wie bei Huffman nötig
 - erreicht Kompression von 5 bis 6 Bit pro Char

Kodierung über Wörterbücher

- Idee
 - ersetzen wiederholtes Auftreten von Sequenzen durch Hinzufügen eines Positionszeiger in einem Lexikon
 - Positionszeiger benötigt normalerweise weniger Speicher als das kodierte Symbol
 - a: 1, 2; b: 3, 4, 5; c: 7, 8, 9

Kodierung über Wörterbücher

- Bekanntester Vertreter: LZW, Lempel-Ziv-Welch Algorithmus
- Erreichen Kompression von 3 bis 4 Bit pro Char

Fazit

- Huffman Coding ist nicht für alle Anwendungen der beste Kompressionsalgorithmus
- allerdings bieten wortbasierte Varianten konkurrenzfähige Kompressionsraten
- im Information Retrieval und NLP ist Huffman Coding unverzichtbar, da komprimierte Daten so durchsuchbar sind

Weitere Information

- Baeza-Yates, Ricardo / Ribeiro-Neto, Berthier (1999), Modern Information Retrieval: pp. 173-190

Aufgabe

- Erstellen Sie ein Modul, das
 - einen ASCII-String aus einer Datei (> ca. 10 KB) nimmt
 - diesen nach Leerzeichen tokenisiert
 - per Huffman-Coding (in der bitweisen Variante) komprimiert
 - das Ergebnis abspeichert

Aufgabe

- die Kompressionsrate berechnet
- zählt, wie häufig ein als Argument gegebener String im komprimierten Text vorkommt
- versuchen Sie die Funktionsweise des Verfahrens nochmals anhand von ‚Modern Information Retrieval‘ und dem Quelltextes des Moduls nachzuvollziehen

Aufgabe vom letzten Mal

- Erstellen Sie ein Modul, das
 - ein HTML-Dokument von einer URL holt
 - Plain Text extrahiert
 - Types und Token zählt und deren jeweilige Gesamtzahl ausgibt
 - Links (`<a href ...>...`) extrahiert und deren Titel und Ziel ausgibt

Aufgabe vom letzten Mal

- für jedes Dokument einen Term-Vektor mit den jeweiligen Typen als Schlüssel und den jeweiligen Häufigkeiten als Werten erstellt

Aufgabe

- Abgabe der Aufgabe in Teams per E-Mail
- Abgabe bis zum 21.06.2007, 0 Uhr

Referenzen

- [1] Baeza-Yates, Ricardo / Ribeiro-Neto, Berthier (1999): Modern Information Retrieval. Boston, MA: Addison-Wesley.
- [2] Güting, R. H. u. Dieker, S. (2004): Datenstrukturen und Algorithmen, Teubner Verlag, Wiesbaden.

Referenzen

- [3] http://en.wikipedia.org/wiki/Quantization_%28signal_processing%29
- [4] <http://search.cpan.org/src/BIGJ/Algorithm-Huffman-0.09/Huffman.pm>

**Vielen Dank für Ihre
Aufmerksamkeit!**