COM3110/4115/6115:

Text Processing

Text Compression

Rob Gaizauskas

Department of Computer Science

University of She eld

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page26) 1 / 26

Overview

Models

Static

Semi-static Adaptive

Coding

Hu man Coding Arithmetic Coding

Further topics:

Symbolwise Models Dictionary Methods Synchronisation

Performance Issues

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 2 / 26

Introduction

Have seen a dramatic increase of

low cost disk storage transmission bandwith processor speed

But also a massive increase in data volume:

text, sound and images

so, techniques to compress data remain signi cant

We shall concentrate on techniques for text compression

Text compression distinct from some other forms of data compression:

the text must be exactly reconstructable

not so critical for digitised analogue signals, such as image/sound text compressions requires so-called lossless coding

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 3 / 26

Introduction (ctd)

Distinguish lossless vs. lossy compression methods Lossless Compression:

class of algorithms allowing original data to be perfectly reconstructed from compressed data

Lossy Compression:

achieve data reduction by discarding (i.e. losing) information

suitable for certain media types, esp.: image / video / audio data

widely used in data streaming contexts

e.g. achieve data reduction of an image by computing a version with lower pixel density

Text data requires lossless compression

\text from which N% of info discarded" doesn't make sense expect decompression to return text identical to original in

form/content

(See Wikipedia pages for Lossy and Lossless Compression)

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 4 / 26

Introduction (ctd)

Text compression techniques may be classi ed in several ways.

symbolwise vs. dictionary methods static vs. adaptive methods

Dictionary methods work by replacing word/text fragments with an index to an entry in a dictionary

Symbolwise methods work by estimating the probabilities of symbols (characters/words) and coding one symbol at a time using shorter codewords for the more likely symbols

rely on a modeling step and a coding step

modeling : estimation of probabilities for the symbols in the text

the better the probability estimates, the higher the compression that can be achieved

coding : conversion of probabilities from model into a bitstream

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 5 / 26

Introduction (ctd)

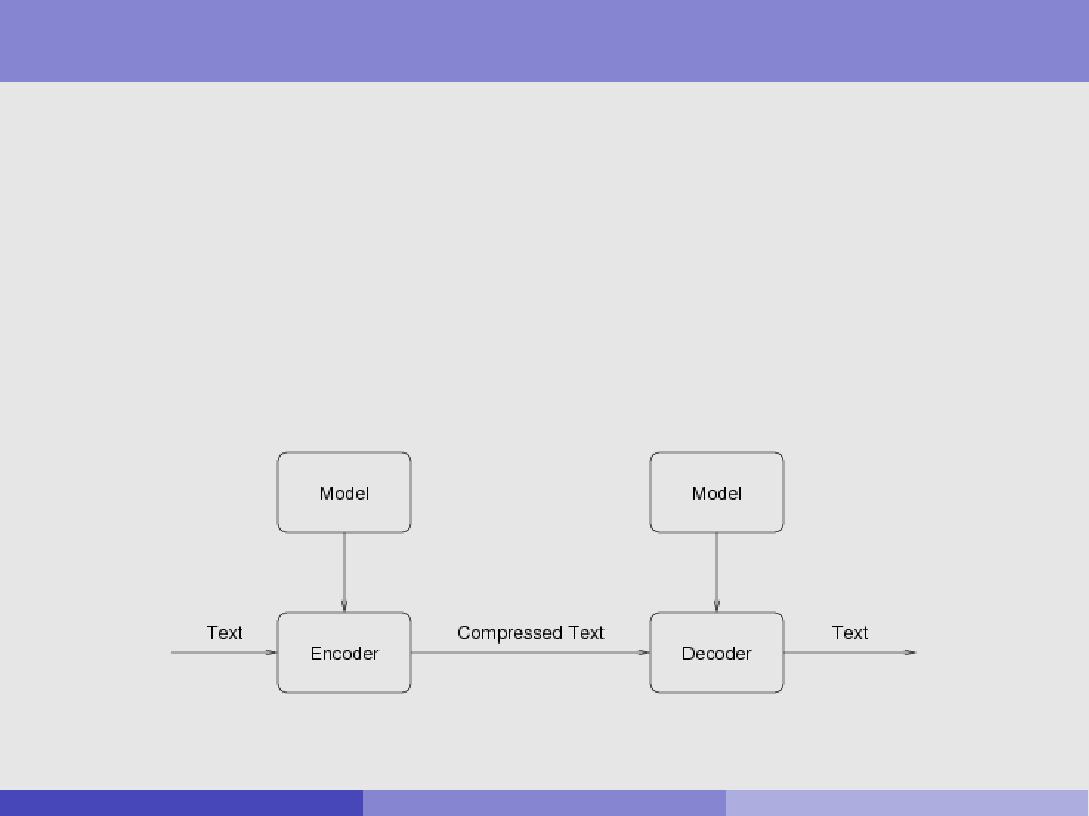
Static vs. adaptive methods:

Static: use a xed model or xed dictionary derived in advance of any text to be compressed

Semi-static: use current text to build a model or dictionary during one pass, then apply it in second pass

Adaptive: build model or dictionary adaptively during one pass

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 6 / 26



Coding and Decoding

Function of model is to predict symbols

model amounts to a probability distribution for all possible symbols, i.e. the \alphabet"

The encoder uses the model to encode (compress) the text The decoder must use the same model to decode it

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 7 / 26

Information Content

The number of bits in which a symbol s should be coded is its information content, denoted I (s)

Information content related to predicted probability P[s], as follows:

I (s) = log2P[s]

e.g. for a fair coin toss in which the outcome is \heads", the best an encoder can do it use log2( 12 ) = 1 bit.

The entropy H of the probability distribution | the average information per symbol over the whole alphabet | is given by:

X X

H = P[s] I (s) = P[s] log2P[s] bits/character

s s

akin to `average' of code lengths weighted by probability

The entropy H places a lower bound on compression

Shannon's Source Coding Theorem

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 8 / 26

Information Content (ctd)

Examples:

With a `fair' coin, P(head) = P(tail) = 0:5

hence: I (head) = I (tail) = log2(0:5) = 1

hence: H = 0:5 1 + 0:5 1 = 1

With a `biased' coin, such that P(head) = 0:99, P(tail) = 0:01

I (head) = log2(0:99) = 0:0145, I (tail) = log2(0:01) = 6:644

H = 0:99 0:0145 + 0:01 6:644 = 0:0808

For a `fair' 8-sided dice, P(s) = 18 = 0:125 for each side s

I (s) = log2( 18 ) = 3, and hence H = 3 also

For a `baised' 8-sided dice, with P(s0) = 0:9 and other P(si ) = 0:0143

I (s0) = log2(0:9) = 0:152, I (si ) = log2(0:0143) = 6:13

H = 0:7498

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 9 / 26

Models and Context

Probability of encountering a given symbol at a particular place in a text is in uenced by preceding symbols

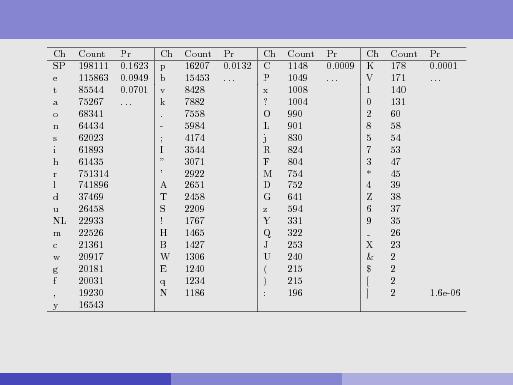
e.g. probability of u following q much higher than u occurring on average

Models that take immediately preceding symbols into account are called nite-context models

best text compression results consider contexts of 3-5 characters

Models that ignore preceding content are called zero order models

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 10 / 26



Static Zero Order Character-based Model for Moby Dick

81 characters in alphabet

total character occurrences = 1,220,150 entropy using this model: 4.4953 bits/char

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 11 / 26

Static vs. Adaptive Modelling

Probabilities may be estimated in various ways

Static modelling derives, and then uses, a single model for all texts

will perform poorly on texts di erent from those used in constructing the model, e.g. texts with tables of numbers

Semi-static modelling derives model for the le in a 1st pass

model derived will be better suited to the text than a static one but, is ine cient because:

* + must make two passes over text
  + must also transmit model

Adaptive modelling derives model during encoding

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 12 / 26

Adaptive Models

Adaptive models begin with a base probability distribution

re ne the model as more symbols are encountered, during encoding so, the text being encoded itself re-de nes the model

Decoder starts with same base probability distribution

it is decoding the same symbol sequence

so, it can re ne the model in the same way

Issues:

Care must be taken to ensure no character is ever predicted with zero probability simply because it has not been seen yet

Principal disadvantage: not suitable for random access to les

a text can only be decoded from the beginning poses di culties for retrieval applications

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 13 / 26

Adaptive Models (ctd)

EXAMPLE : encoding Moby Dick

might start assuming a uniform probability of 1=81 per char

implies same minimal possible code length for all symbols at log2(1=81) = 6:34 bits

after encoding (say) 100,000 chars, and observing char distributions, we have a much more accurate 0-order model

suppose now about to encode e of whale:

by now have observed that 10% of chars are e

thus, the e can be encoded inlog20:1 = 3:32 bits

decoder will re ne its model in parallel manner for use in decoding

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 14 / 26

Hu man Coding

Coding is the task of determining the output representation of a symbol, given the probability distribution supplied by the model

Coder should output:

short codes for high probability symbols long codes for low probability symbols

Speed of coder may also be signi cant

computing optimal codes can be slow

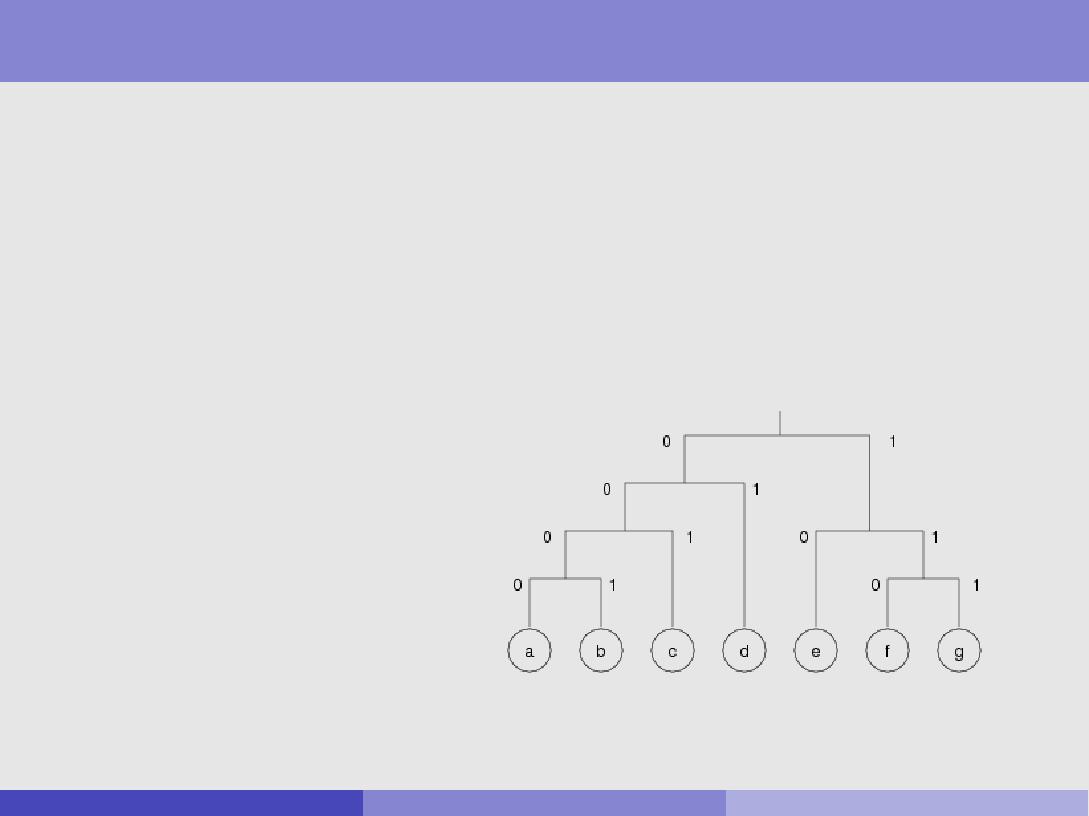
hence, there is a trade-o between speed of compression and compression rate

Hu man coding dates from 1952

was the dominant model till the 1970's

with re nements, it still has applications today, e.g. in text retrieval

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 15 / 26



Hu man Coding (ctd)

Technique uses a code tree for encoding and decoding

each branch is labelled with a 0 or 1

each leaf node is a symbol in the alphabet

code is pre x-free { no codeword is the pre x of another

Consider a seven symbol alphabet (example from Witten et al.)

|  |  |  |
| --- | --- | --- |
| Symbol | Prob | Codeword |
|  |  |  |
| a | 0.05 | 0000 |
| b | 0.05 | 0001 |
| c | 0.1 | 001 |
| d | 0.2 | 01 |
| e | 0.3 | 10 |
| f | 0.2 | 110 |
| g | 0.1 | 111 |

e.g. eefggfed is coded as 10101101111111101001

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 16 / 26

Hu man Coding (ctd)

Given a set of codewords produced by the algorithm, can compute the expected average code length as follows:

multiply each symbol code length by associated probability sum results across all symbols

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbol | Prob | Code | len | p len |
| a | 0.05 | 0000 | 4 | 0.2 |
| b | 0.05 | 0001 | 4 | 0.2 |
| c | 0.1 | 001 | 3 | 0.3 |
| d | 0.2 | 01 | 2 | 0.4 |
| e | 0.3 | 10 | 2 | 0.6 |
| f | 0.2 | 110 | 3 | 0.6 |
| g | 0.1 | 111 | 3 | 0.3 |
|  |  |  |  |  |
|  |  |  |  | 2.6 |

Compare this to minimal xed-length code length for same symbol set e.g. for above symbol set, could use a 3-bit xed length code

3-bit xed length code allows for 23 = 8 symbols

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 17 / 26



Hu man | the coding algorithm

The code tree is constructed bottom up from the probabilistic model according to the following algorithm:

* 1. Probabilities are associated with leaf nodes
  2. Identify the two nodes with smallest probabilities

join them under a parent node, whose probability is their sum

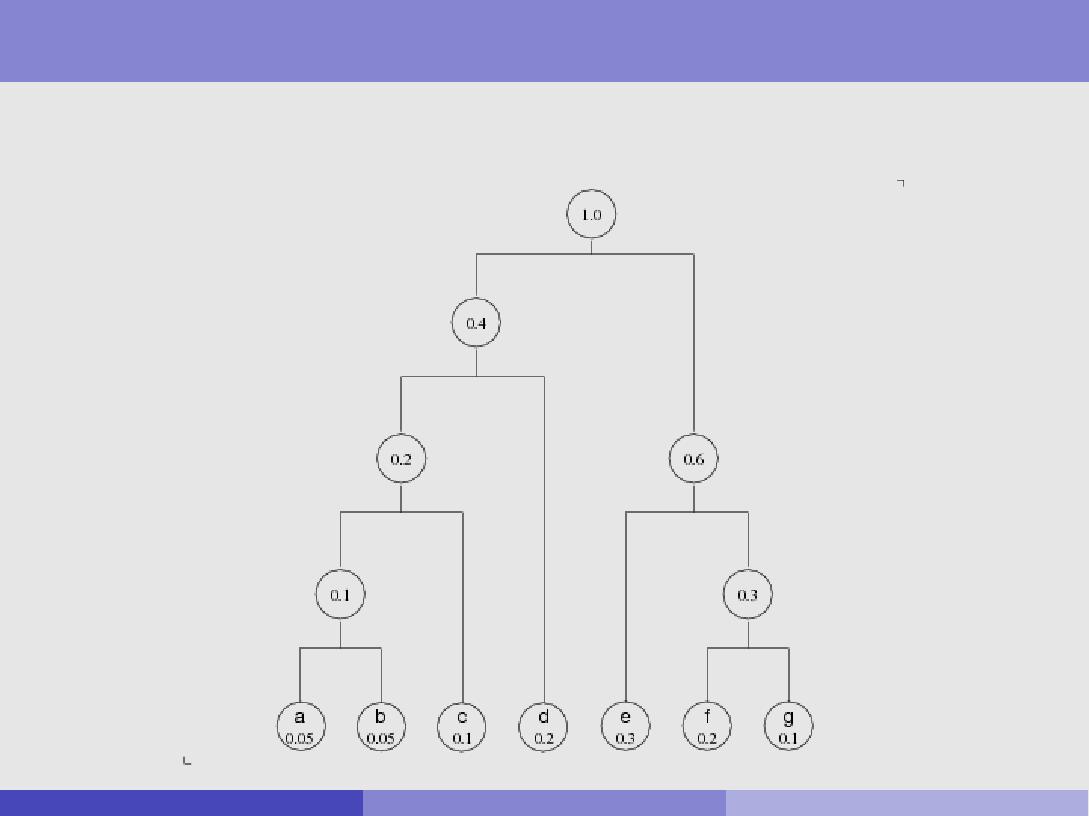
* 1. Repeat step (b) until only one node remains
  2. 0's and 1's are then assigned to each binary split

EXAMPLE :

start with leaf nodes:

join two nodes with smallest probabilities:

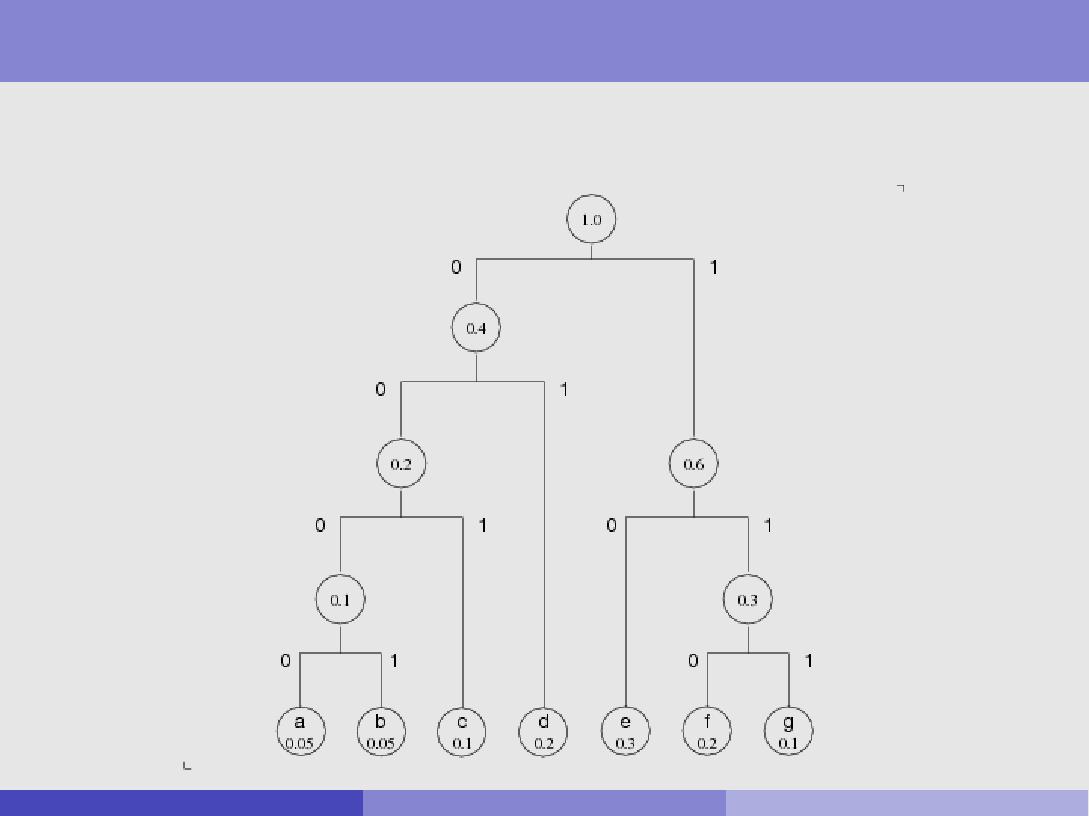
Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 18 / 26



Hu man | the coding algorithm (ctd)

repeat until only one node remains . . .

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 19 / 26



Hu man | the coding algorithm (ctd)

assign 0/1 to branches of each binary fork

doesn't matter which is which

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 20 / 26

Hu man Coding | further details

Hu man coding is fast for both encoding and decoding

provided probability distribution is static

Variants of Hu man coding developed for adaptive models

but these are complex

generally better to use arithmetic coding for adaptive models

Hu man e ective when used with a word-based model, rather than a character-based model

gives good compression fast

supports random access to compressed les

(given some additional requirements on how method used)

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 21 / 26

Canonical Hu man codes

Hu man gives e ective compression where there are many symbols, with a highly skewed distribution

arises, e.g. for word-based models

i.e. many symbols (words), some v.frequent, some rare

Tree-based storage of v.large models is costly & ine cient

tree nodes store pointers to child nodes | costly for memory

traversing tree involves much jumping between locations | ine cient

These issues addressed by use of canonical Hu man codes

special Hu man code, where codes generated in a standardised format speci cally, all codes for given code-length assigned values sequentially this feature allows for both

e cient storage and/or transmission of codebook more e cient decoding algorithm

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 22 / 26

Canonical Hu man codes (ctd)

To create a canonical code:

rst determine length of code for each symbol

can do this by applying standard Hu man coding algorithm

group symbols having same code-length into blocks, and order

e.g. could sort alphabetically, or in some other way

assign codes by `counting up' { addressing blocks in code-length order

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| To illustrate, consider our | Symbol | Prob | Code | len |
| 7-letter example again: | a | 0.05 | 0000 | 4 |
|  | b | 0.05 | 0001 | 4 |
|  | c | 0.1 | 001 | 3 |
|  | d | 0.2 | 01 | 2 |
|  | e | 0.3 | 10 | 2 |
|  | f | 0.2 | 110 | 3 |
|  | g | 0.1 | 111 | 3 |

ignore original codes, and group symbols by their code-length, i.e.:

* 1. d e [3] c f g [4] a b

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 23 / 26

Canonical Hu man codes (ctd)

Example { continued: creating a canonical code . . .

First, group symbols by their code lengths:

[2] d e [3] c f g [4] a b

Assign codes:

assign rst symbol a `zero' code of required length for successive symbols, simply count up 1

if code length goes up, then (i) count up & (ii) add 0s to get new len

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [2] d | e | [3] c | f | g | [4] a | b |
| 00 | 01 | 100 | 101 | 110 | 1110 | 1111 |

To store / transmit this code, is su cient to specify:

sequence of symbols: d e c f g a b

number of symbols at each code-length: (0, 2, 3, 2)

i.e. len 1: 0 items; len 2: 2 items; len 3: 3 items; len 4: 2 items

this is su cient info to reconstruct entire code

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 24 / 26

Canonical Hu man codes (ctd)

This approach also supports an e cient decoding algorithm

does not require a code tree

Method:

store blocks of symbols in sequential order

compute code for only the rst symbol in each block

by comparing pre xes of input to these rst-symbol codes (<; >), can determine which block next code belongs to, and hence its length

binary di erence between next code and rst-symbol code gives position of symbol in block sequence

Example:

Example on last slide has rst-symbol codes: 00, 100, 1110

Assume input: 11001111.......

Find: pre x 11 > 00; pre x 110 > 100; pre x 1100 < 1110

so next codeword is 110, of the length 3 block

Di erence of 100 and 110 shows symbol is 3rd item in block sequence

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 25 / 26

Reading

Main:

Baeza-Yates and Ribeiro-Neto, Ch 7.4-7.5

Other:

I. H. Witten, A. Mo at, T. C. Bell. Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd ed. Morgan Kaufmann. 1999.

Nam Phamdo. Theory of Data Compression. www.data-compression.com/theory.html

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 26 / 26